Landmark Retrieval & Recognition

Kenneth Emeka Odoh

12th July 2018 (Kaggle Data Science Meetup | SFU Ventures Lab)

Vancouver, BC

Plan

- Bio
- Background
- Winning Solutions
- My solutions
- Conclusions
- References

Bio

Education

- Masters in Computer Science (University of Regina) 2013 - 2016
- A number of MOOCs in statistics, algorithm, and machine learning

Work

- Research assistant in the field of Visual Analytics 2013 - 2016
  - (https://scholar.google.com/citations?user=3G2ncgwAAAAJ&hl=en)
- Software Engineer in Vancouver, Canada 2017 -
- Regular speaker at a number of Vancouver AI meetups, organizer of distributed systems meetup, and Haskell study group and organizer of Applied Cryptography study group 2017 -

Background

- Competitions
  - Landmark retrieval
  - Landmark recognition
- Data Exploration
- Evaluation metric
  - Landmark retrieval
  - Landmark recognition
- Primer on information retrieval
- Approaches
  - Supervised
  - Unsupervised

Competitions

- Landmark retrieval
  - { https://www.kaggle.com/c/landmark-retrieval-challenge/overview }
- Landmark recognition
  - { https://www.kaggle.com/c/landmark-recognition-challenge/overview }

Both competitions have the following prizes

- 1st place - $ 1,250
- 2nd place - $ 750
- 3rd place - $ 500

Data Set

Both competition share roughly similar data, but the landmark recognition data set has label, while the landmark retrieval data set has no labels. There is a **fuzzy** nature of the description of landmarks.

- 15k unique landmarks, 1M training images, 0.1M testing images.
- Images have various sizes and high resolution

Total: 329GB

Data Exploration

Figure 1: Image containing landmarks and their matches

Figure 2: Image containing landmarks and their mismatches (part 1)

Figure 3: Image containing landmarks and their mismatches (part 2)

Approaches

- Supervised
  - neural architecture
  - Traditional machine learning models

- Unsupervised
  - Locality sensitive hashing, DFT, Wavelet

GOOGLE LANDMARK RETRIEVAL

Evaluation Metric

Submissions are evaluated according to mean Average Precision @ 100 ($mAP@100$):

$$mAP@100 = \frac{1}{Q}\sum_{q=1}^{Q}\frac{1}{min(m_q, 100)}\sum_{k=1}^{min(n_q,100)}P_q(k)rel_q(k)$$

where:

- Q is the number of query images that depict landmarks from the index set
- m_q is the number of index images containing a landmark in common with the query image q (note that this is only for queries which depict landmarks from the index set, so $m_q \neq 0$)
- n_q is the number of predictions made by the system for query q
- $P_q(k)$ is the precision at rank k for the q-th query
- $rel_q(k)$ denotes the relevance of prediciton k for the q-th query: it's 1 if the k-th prediction is correct, and 0 otherwise

[Copied from competition site]

Precision is fraction of the result of a query that is relevant to the information needs

Recall is the fraction of relevant document returned by query in relation to the total collection of information.

Submission format

id,images
000088da12d664db,0370c4c856f096e8 766677ab964f4311
e3ae4dcee8133159...
etc.

Information Retrieval

Given a query image, **V** and a collection of images I_D.

return a set of images, I_V that are most similar to **V**.

V can contain zero or more landmarks.

$$I_V = \bigcup_{u \in I_D} \{ u \mid sim(u,v) \leq threshold \}$$

How is similarity defined?

sim (v_i, v_j): is the distance between images of v_i, v_j respectively as a proxy for the presence of a number of common landmarks between the images.

Winning Solutions

- Represent images as vectors.
- Build a efficient searching and querying of images for matches.
- Perform query expansion
    - Most people used diffusion based query expansion described in { https://arxiv.org/abs/1611.05113 }.

Key Glossary

- A descriptor encodes an image in a vector to allow for comparison between images.
- Global descriptor describes the entire image. It is very general.
- Local descriptor describe patches between the entire image. It is very specific. A group of local descriptors is required to described the entire image.

1st place

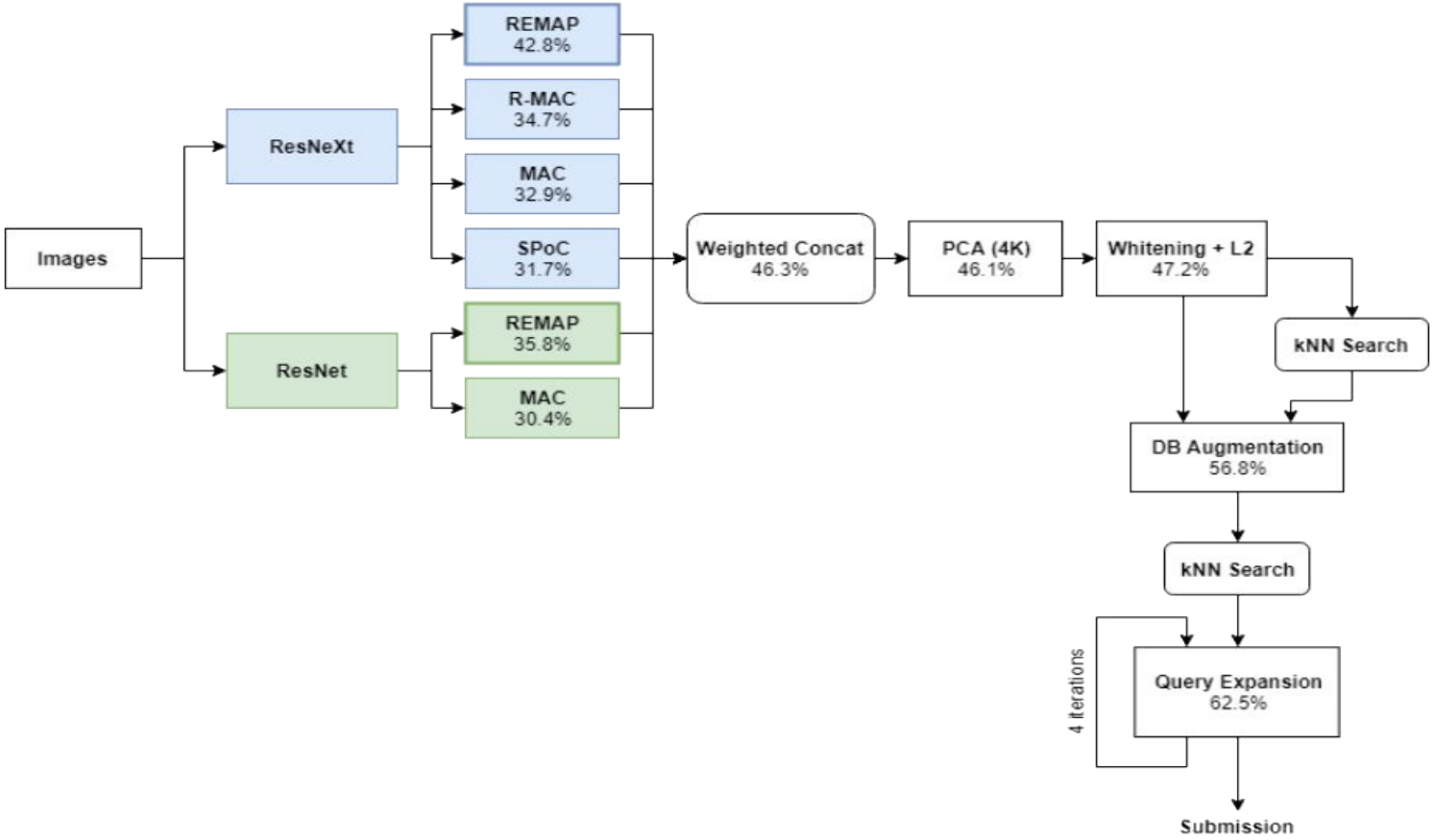


Figure 4: 1st place solution pipeline with corresponding LB scores

1. Create vector representation of images using global descriptors using pre-trained network (ResNet, ResNext) and a number of aggregation methods (REMAP, MAC, SPOC, RMAC).

The aggregation methods are:

- Region-Entropy based Multi-layer Abstraction Pooling (REMAP) [NOT YET PUBLISHED]
- Maximum Activations of Convolutions (MAC) { https://arxiv.org/abs/1511.05879 }
- Sum-pooling of Convolutions (SPoC) { https://arxiv.org/abs/1510.07493 }
- Regional Maximum Activations of Convolutions (RMAC) { https://arxiv.org/abs/1610.07940 }

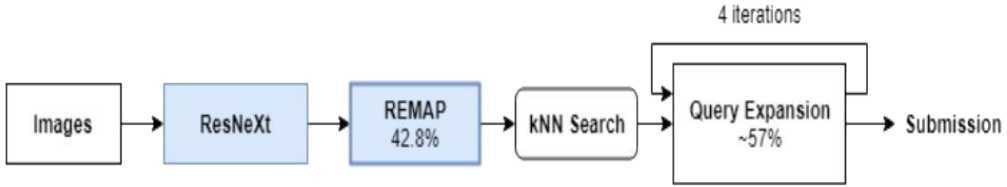
This weighted concatenation vector, XG
XG = [2× ResNeXt+REMAP; 1.5× ResNeXt+RMAC; 1.5× ResNeXt+MAC; 1.5× ResNeXt+SPoC; ResNet+MAC; ResNet+REMAP]

- L2 norm + whitening perform a XG.
- PCA perform a XG.

The authors claimed that PCA and whitening gave boost in **query expansion**.

2. Build efficient scheme for matching vectors to most likely query images.
The resulting image has a 4096-vectors. They perform k-nearest neighbour to identify 100 neighbours of each test image by using L2 distance.

3. Database-side augmentation
In the training dataset, Use a image vector and 10 nearest neighbours as a weighted combination to leverage information from neighbours.
weights = logspace(0, -1.5, 10)



4. Query expansion
Expanding search query to increase matches and improve performance.
A <-> B, and B <-> C, then A <-> C
This is a **classified model** that is recursive to capture long-distance connection between images, thereby identifying images with the same landmarks.

Things that didn't work

- Difficulty handling rotated images.
- Better to ensemble at beginning than at end

{ https://www.kaggle.com/c/landmark-retrieval-challenge/discussion/57855 }

6th place

generalized-mean pooling (GeM)

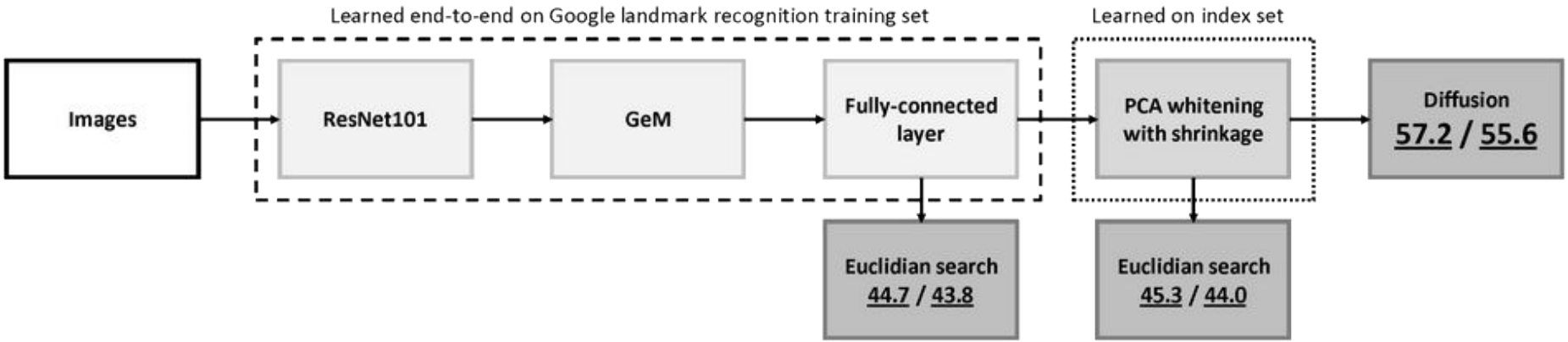


Figure 5: 6th place solution pipeline with corresponding LB scores

{ https://www.kaggle.com/c/landmark-retrieval-challenge/discussion/58482 }

11th place

Extract image features using RMAC { https://arxiv.org/pdf/1511.05879.pdf }

Each image is represented as a 2048-d feature, and the database feature is about 8GB. Compute cosine similarity for image matching.

The database side augmentation is performed by the knngraph {https://github.com/aaalgo/kgraph }, which is fast but may not be accurate in finding the neighbors between every database images.

We use a desktop computer with 32GB memory; uses about 40 mins to process about 10k images in a GTX 1060 GPU

My Solution

Unsupervised Method

- Create a hash of images in binary vectors of 64 bits.
- Obtain a measure of similarity between images using hamming distance.
  - Score of 0, is identical image. However, score < 5 can represent images with distortion.
- Restrict results by thresholding to filter unnecessary images
- Perform pairwise comparisons of images to a query image and sort the score in decreasing order.
- Choose 100 most relevant results.

Will the solution perform well in the face of rotated images and their upright versions?

Inspired by { https://tech.okcupid.com/evaluating-perceptual-image-hashes-okcupid/ }

```
from heapq import heappop, heappush

def hammingDistance ( aval ,bval ):
        return np.count_nonzero(aval != bval)

def getNearestList (id, cur_id, cur_avghash, avghash, c_avgheap, threshold):
        ncur_avghash, navghash = convertHexToBinary (cur_avghash),
convertHexToBinary (avghash)
        diffHash = hammingDistance ( ncur_avghash, navghash )
        if id != cur_id and diffHash < threshold:
            heappush(c_avgheap, (-1 * diffHash, cur_id)) # hamming distance,
id tuple
        del ncur_avghash
        del navghash
        del diffHash
        return c_avgheap

def solution (traindata, testdata ):
        avghashResultDF = pd.DataFrame(columns=["id", "images"])
        for i in range(len(testdata)):
            id = testdata.loc[i]['id']
            avghash = testdata.loc[i]['avghash']
            c_avgheap = []
            for ind in range(len(traindata)):
               cur_id = traindata.loc[ind]['id']
               #average
               cur_avghash = traindata.loc[ind]['avghash']

        c_avgheap = getNearestList (id, cur_id, cur_avghash, avghash,
c_avgheap, threshold=12)
        sortedlist_avg = [heappop( c_avgheap )[1] for v in
range(len(c_avgheap))] [:100]
        idlist_avg = ' '.join( sortedlist_avg )
        avghashResultDF.loc[i] = [id, idlist_avg]
        del idlist_avg
        avghashResultDF.to_csv('results/avghash.csv', encoding='utf-8',
index=False)
```

```
def getNearestAvgList (id, cur_id, chash, hashlist, c_avgheap, threshold,
ftype="geometric_mean"):
        '''
        ftype="geometric_mean", "harmonic_mean", "average_mean",
"squared_root_mean"
        '''
        currentDiffHashList = []
        diffHash = 0

        for localhash in hashlist:
            nchash, nlocalhash = convertHexToBinary (chash),
convertHexToBinary (localhash)
            cdiffHash = hammingDistance ( nchash, nlocalhash )
            if cdiffHash > 0:
                currentDiffHashList.append ( cdiffHash )

        if ftype == "geometric_mean":
            #geometric mean
            diffHash = geo_mean_overflow( currentDiffHashList )

        elif ftype == "harmonic_mean":
            #harmonic mean threshold 17.5
            diffHash = len(currentDiffHashList) /
np.sum(1.0/np.array(currentDiffHashList))

        elif ftype == "average_mean":
            #Avg mean threshold 18
            diffHash = sum(currentDiffHashList) / len(currentDiffHashList)
            #print "threshold {}".format(diffHash)

        elif ftype == "squared_root_mean":
            #average squared root threshold 4.3
            diffHash = np.average( np.power(currentDiffHashList, 0.5) )

        if id != cur_id and diffHash < threshold:
            heappush(c_avgheap, (-1 * diffHash, cur_id)) # hamming distance,
id tuple

        del currentDiffHashList
        del diffHash
        return c_avgheap
```

Image Hashes

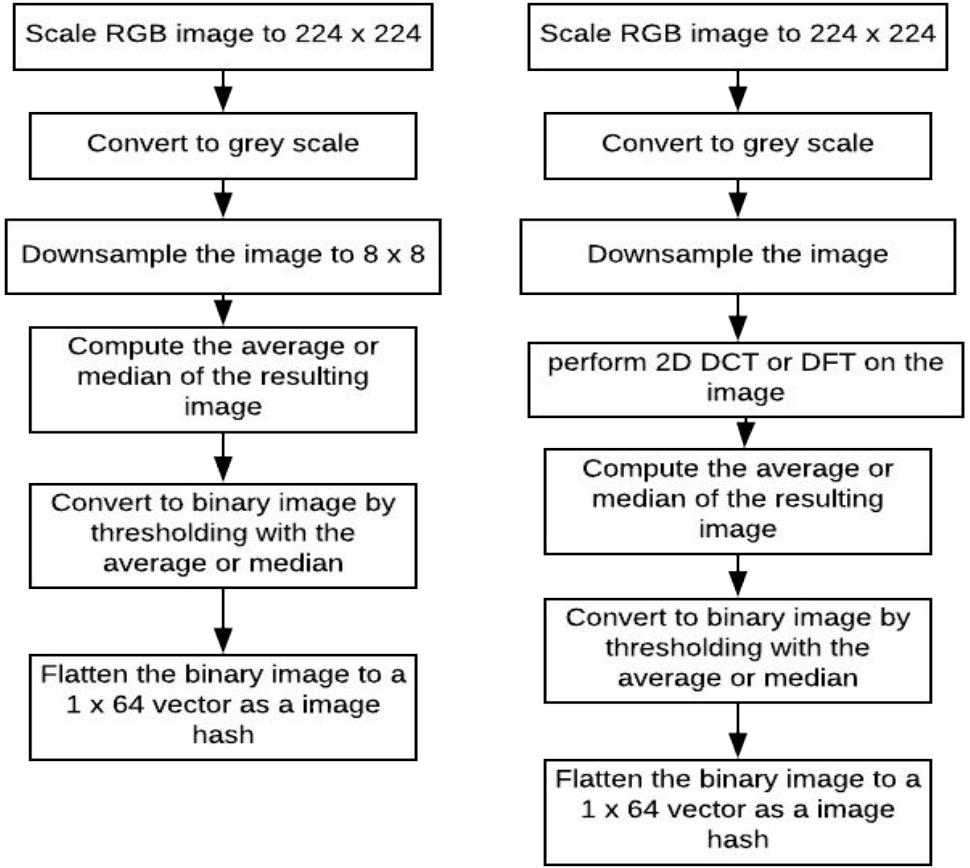
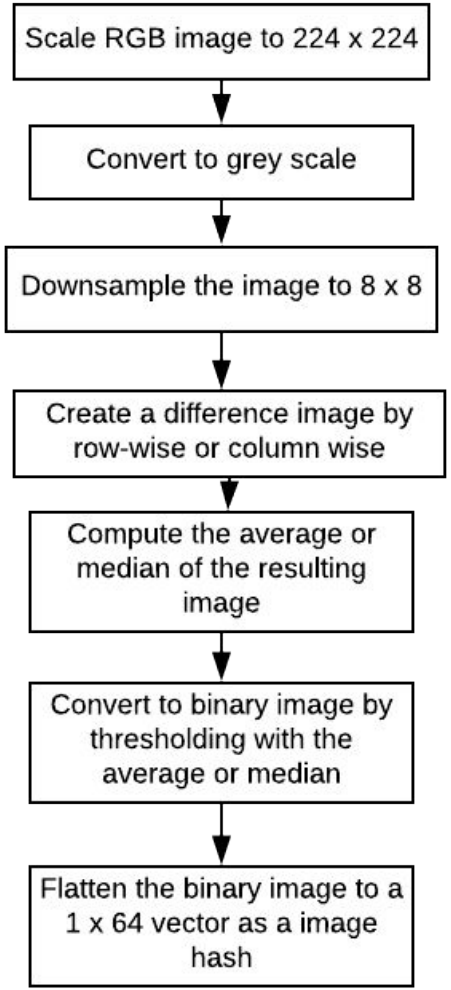


Figure 6: Average and DCT hashes of images



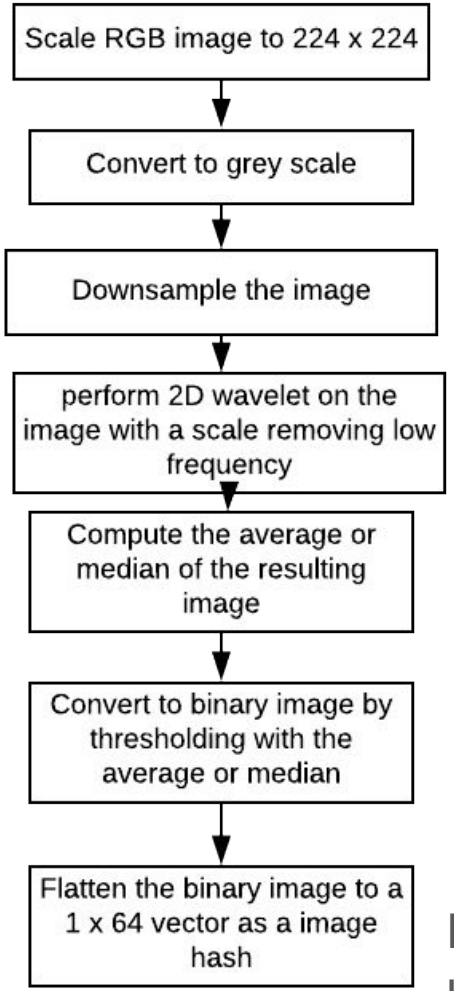


Figure 7: Difference and Wavelet hashes of images

Locality Sensitive Hashing

hash function is a one-way function.

Hashing is the processing of taking an object and returning a unique representation of the object. It is impossible to reconstruct the original object from its representation.

Unlike a number of cryptographic related hashes like MD5, SHA256 e.t.c whose aim is to generate representations that are a far as possible for objects with slight differences. e.g

MD5 (“ken”) →”f632fa6f8c3d5f551c5df867588381ab”
MD5 (“ken.”) →”c4d8165ebd5bfa74e29f02f387b50259”
Locality sensitive hashing is generating hashes that make similar objects as close as possible and different objects as far as possible. For more information, see { https://blog.insightdatascience.com/elastik-nearest-neighbors-4b1f6821bd62 }

Finding Similar Items

There are a few techniques

- shingling (convert to set of short phrase)

Use k-shingles. If k is small, the jaccard coefficient will be high even if the documents are not similar.

- minhashing

This make use of a hash function to create a signature of the image that helps avoid pairwise comparison of jaccard distance.

Optimizations

I could not **think straight** in the competition

- Try to optimize using **trie**.
- Create a form of **locality sensitive hashing** by using a linear function with positions of the bits and hashing to a bucket.
- Exploiting triangle inequality of hamming distance with an updatable key heap data structure.

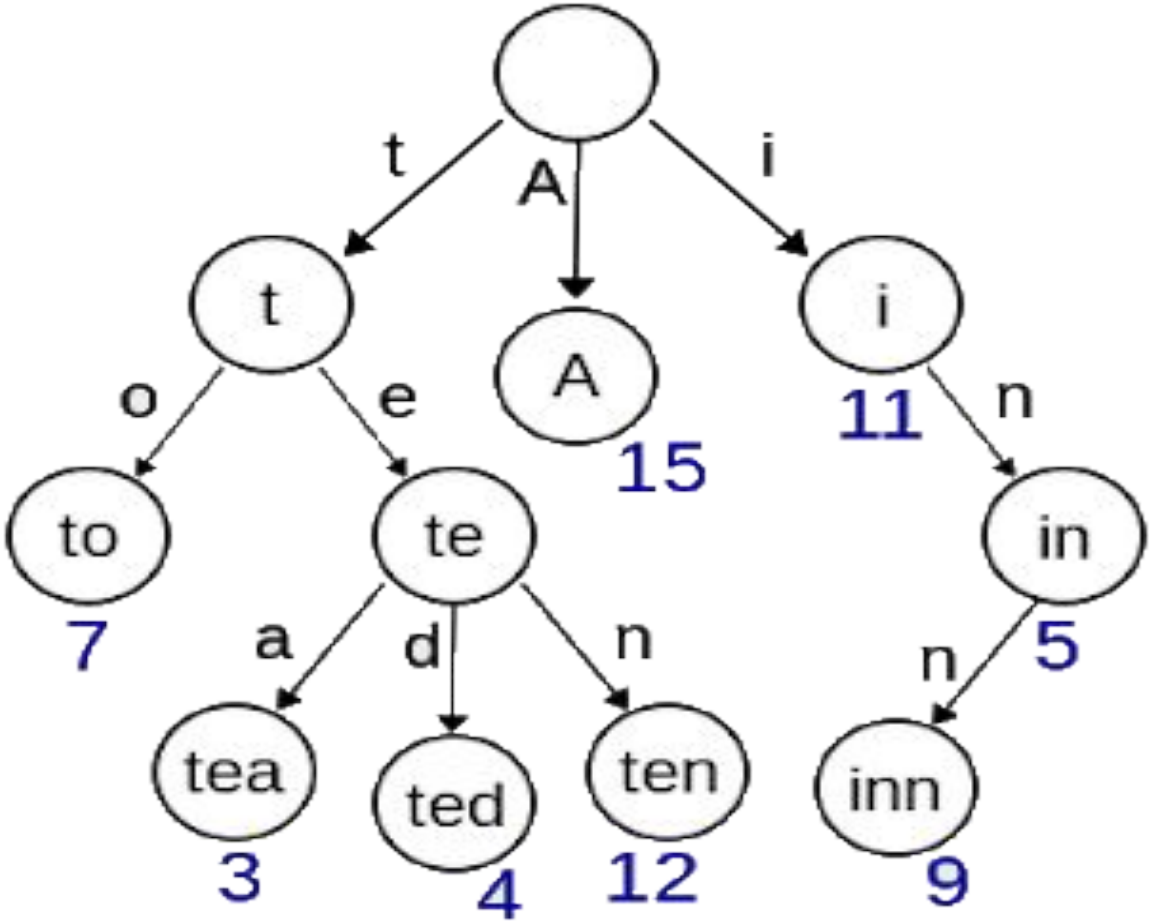


Figure 8: Trie Data structure

{ http://arxoclay.blogspot.com/2011/04/implementing-trie-with-python.html }

Discrete Fourier Transform (DFT)

DFT aims to show that any signal can be approximated by a weight combination of trigonometric functions. This applies to both periodic signal and aperiodic signals.

Consider the example of a signal

$$Y = f(X) = m + a \cdot \cos(2\pi X / L)$$

$$Y_0 = m + a \cdot \cos(0) = m + a \quad \ldots @ X = 0$$

$$Y_1 = m + a \cdot \cos(\pi) = m - a \quad \ldots @ X = L/2$$

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} 1 & \cos 0 \\ 1 & \cos \pi \end{bmatrix} \cdot \begin{bmatrix} m \\ a \end{bmatrix}$$

Inverse DFT

$$f_k = \frac{1}{N} \sum_{n=0}^{N-1} F_n \, e^{2\pi i k n / N}$$

$$\begin{bmatrix} m \\ a \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ \cos 0 & \cos \pi \end{bmatrix} \cdot \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix}$$

DFT

$$F_n \equiv \sum_{k=0}^{N-1} f_k \, e^{-2\pi i n k / N}$$

[Thibo, 2014]

Wavelet

- Fourier theory implies that a signal can be represented by sum of series of sinusoidal curve.
- Fourier basis is localized in frequency. Little changes in the frequency domain lead to changes everywhere in the time domain.
- While we can extract the frequency component of the signal, we lose track of their time component. Probably, an implication of **heisenberg uncertainty principle**.
- Wavelets allows for joint time-frequency representation with multiple resolutions.
- This makes use of variable size of sliding window doing convolutions on the signal.
- Wavelets are local in both time and frequency domain.

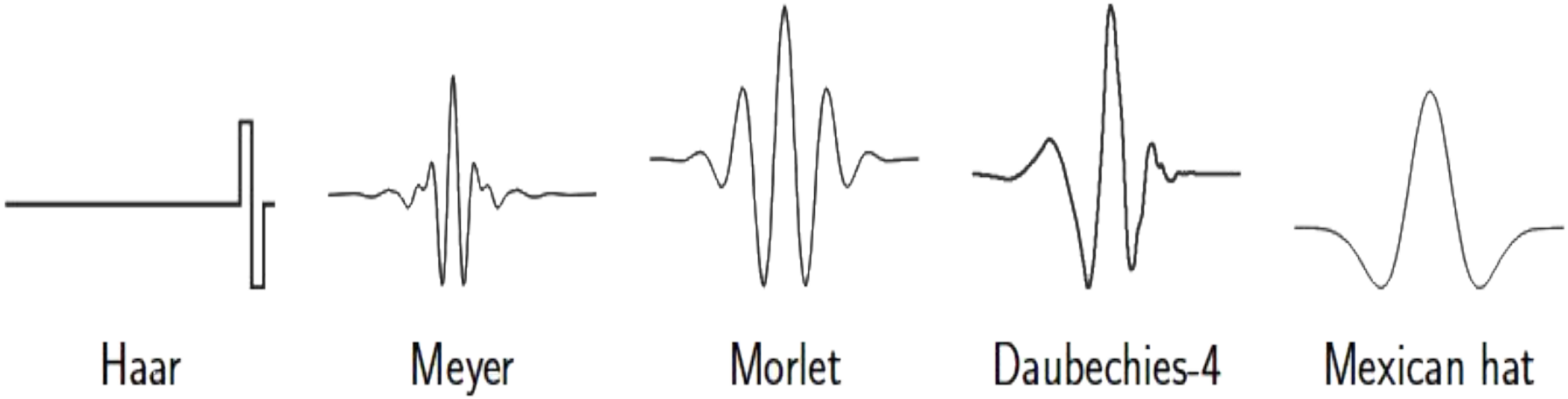


Figure 9: Types of Wavelet basis functions

{ http://people.ciirc.cvut.cz/~hlavac/TeachPresEn/11ImageProc/14WaveletsEn.pdf }

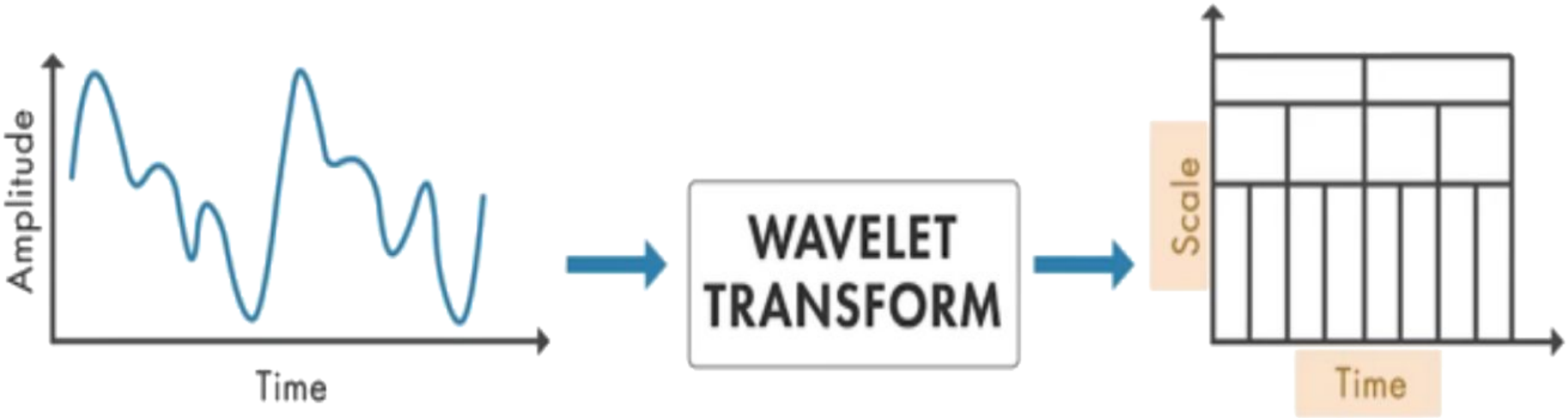


Figure 10: Wavelet Transforms in action

{ https://www.youtube.com/watch?v=F7Lg-nFYooU }

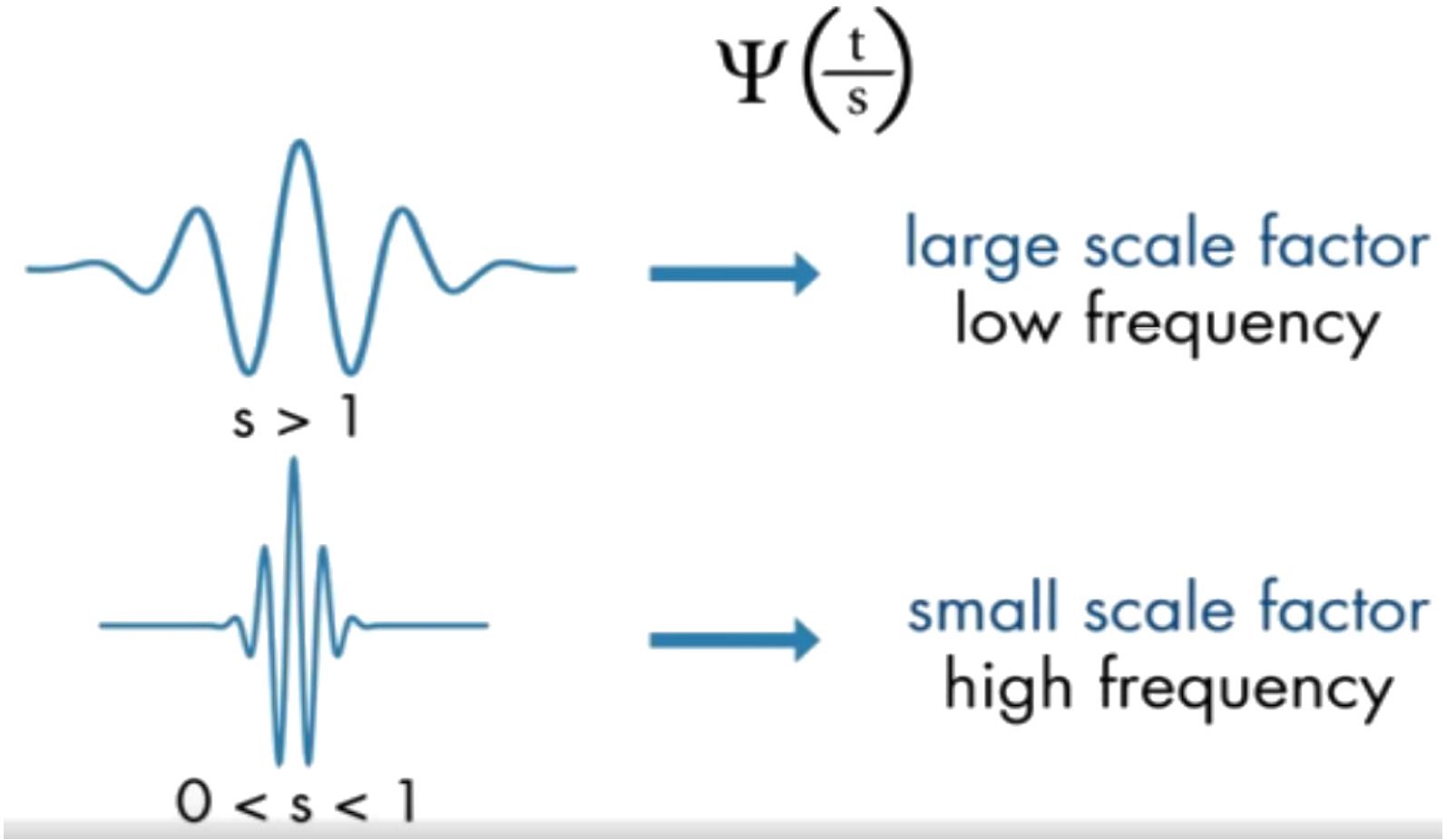


Figure 11: Effect of scale on Wavelet basis function

$$\Psi(t) = \frac{\sin(2\pi t) - \sin(\pi t)}{\pi t}.$$

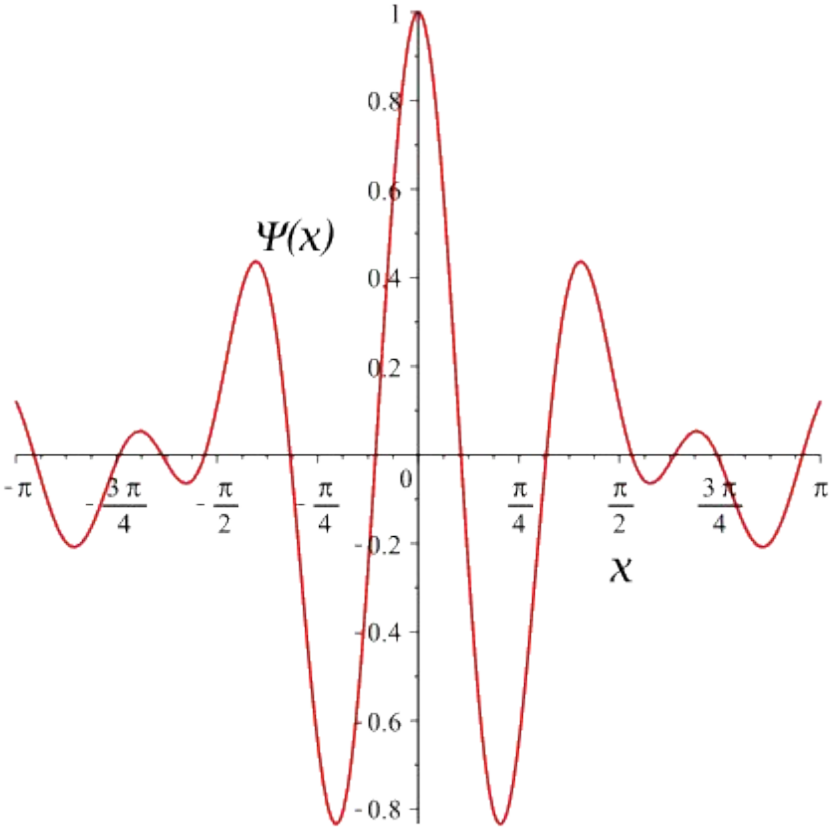


Figure 12: A mathematical representation of a wavelet basis

Continuous Wavelet Transformation

$$c(s,\tau) = \int_R f(t)\, \Psi^*_{s,\tau}(t)\, \mathrm{d}t$$

Forward wavelet transform
(wavelet coefficient)

$$\Psi_{s,\tau}(t) = \frac{1}{\sqrt{s}} \Psi \left(\frac{t-\tau}{s} \right)$$

Basis function

$$f(t) = \int_{R^+} \int_R c(s,\tau)\, \Psi_{s,\tau}(t)\, \mathrm{d}s\, \mathrm{d}\tau$$

Inverse wavelet transform

{ http://people.ciirc.cvut.cz/~hlavac/TeachPresEn/11ImageProc/14WaveletsEn.pdf }

Discrete Wavelet Transformation

$$c(j,k) = \sum_t f(t)\ \Psi^*_{j,k}(t)$$

Forward wavelet transform
(wavelet coefficient)

$$f(t) = \sum_k \sum_j c(j,k)\ \Psi_{j,k}(t)$$

Inverse wavelet transform

{ http://people.ciirc.cvut.cz/~hlavac/TeachPresEn/11ImageProc/14WaveletsEn.pdf }

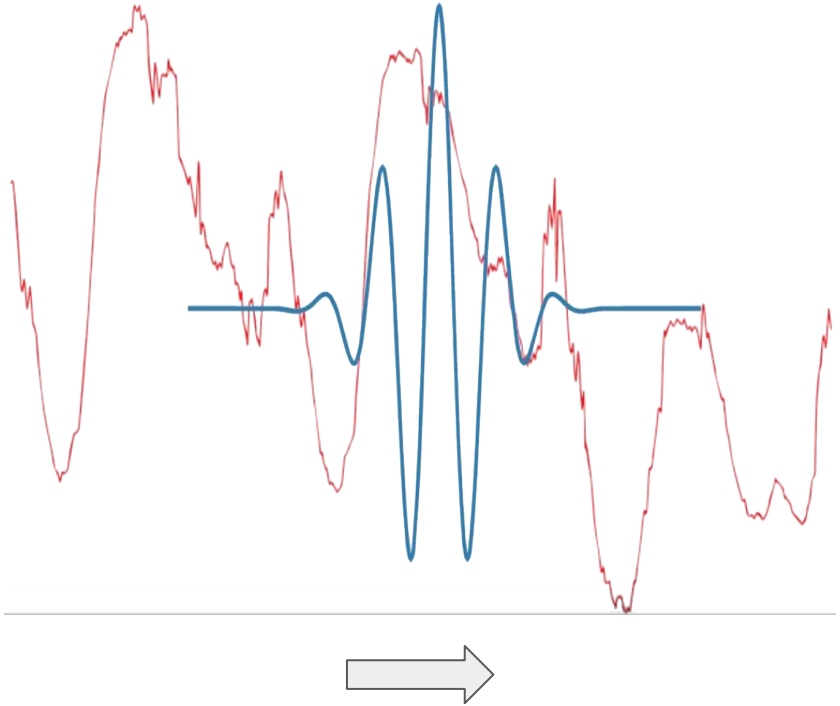


Figure 13: Shifting the basis function over the signal

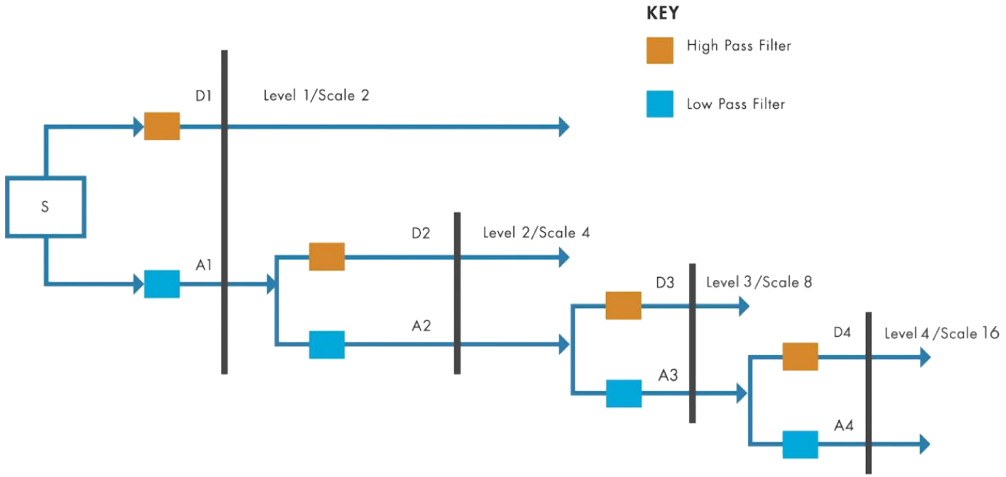


Figure 14: Fast wavelet transform

Transfer learning

This is an attempt to use the method applied for the **Landmark recognition competition** to solve the landmark retrieval competition.

Given that the data for **Landmark recognition competition** has label can be solved by supervised learning.

The data for the Landmark retrieval competition is without labels.

- Create a neural network model for Landmark recognition. Optimize the network for maximum performance. Save the model.
- Load the model and pass the unlabelled data of Landmark retrieval competition as input through the model to create one-hot encoding with dimension of size of data x number of unique labels.

- The lower embedding space of the one-hot encoded matrix can be used for subsequent **clustering**.
- Each column can be taken to be the list of images containing that label.
- Each query image result in a one-hot encoded vector with size as number of unique labels. Get the **max value** and **position** which identifies the **predicted label**.
- Perform a form of clustering that feels like flood filling while incrementing the threshold by creating a query interval on the max value and query all matching images using the position information. This optimization helps to reduce computational time with little or no loss of predicted performance.
- If necessary, perform clustering in batch if one-hot vector cannot be loaded in memory
- See the code for details.

```
from heapq import heappop, heappush
import numpy as np
from keras.models import load_model
import cv2
import os
import gc
"""
Xtrain: is the list of dictionary image
Xtest: is a list of image that is a query to get a
list of training

Both matrices are output of neural networks.

XtestIDs: list of Xtest ids
XtrainIDs: list of Xtrain ids
"""
def getSortedResult (y, sortedy, XtrainIDs, res):
     for sindx in sortedy:
     score = y[sindx]
     cur_id = XtrainIDs [sindx]
     heappush(res, (-1 * score, cur_id)) #
distance, id tuple
     return res
```

```
def matchingIndexOfSingleRecord (X, pind,
pval):
     """
     X, Xtrain is the prediction matrix, pind
is the max predict value column index in a rows
of X, pval is the max predict value column
element in a rows of X
     """
     #get a column of the matrix
     y = X[:, pind]  # array([ 0.91147624,
0.0760729 ,  0.10286262,  0.74501551,
0.16317767, 0.94554172])
     totalArrIndx = np.array([])
     #query intervals
     threshold = 0.01
     limit = 100
     while threshold < 1.0:
          queryIndices =
     np.where((y>=pval-threshold) &
     (y<=pval+threshold))[0]  #array([0, 3])
          totalAr = np.append(queryIndices)
          threshold = threshold + 0.01
          if len (totalArrIndx) > limit:
               break
     return y, totalArrIndx
```

```
def matchingElementOfSingleRecord (y, queryIndices,
XtrainIDs):
      res = []
      res = getSortedResult (y, queryIndices, XtrainIDs,
res)
      sortedlist = [heappop( res )[1] for v in range(len(
res ))]
      return sortedlist

def getDataTrain(model):
      x_train = []
      x_trainids = []
      ListOfFiles = os.listdir('./data/recog/train_dir')
      for filename in ListOfFiles:
            img =
      cv2.imread('data/recog/train_dir/{}'.format(filenam
      e))
            x_train.append(cv2.resize(img, (224, 224)))
            idval = filename.split()
            idstr = "".join( idval[:-1] )
            x_trainids.append( idstr )
      x_train = np.array(x_train, np.float16) / 255.
      Xtrain = model.predict(x_train, batch_size=128)
      return Xtrain, x_trainids
```

```
def matchingRecords (Xtest, Xtrain, XtrainIDs,
XtestIDs):
      pindlist = np.argmax(Xtest, axis=1).tolist()
      pvallist = np.amax(Xtest, axis=1).tolist()
      subm = pd.DataFrame(columns=["id", "images"])
      total = Xtest.shape[0]

      for ind, pind, pval in enumerate ( zip(
pindlist, pvallist ) ):
            y, queryIndices
      =matchingIndexOfSingleRecord (Xtrain, pind,
      pval)

            sortedres =
      matchingElementOfSingleRecord (y,
      queryIndices, XtrainIDs)

            subm.loc[ind] = [XtestIDs[ind],
      sortedres]

            print "test data #{} of {} --
      {}".format(ind+1, total, XtestIDs[ind])

      subm.to_csv('output/submissionretrieval.csv.gz
', compression = 'gzip', index=False)
      print('Done!')
```

```
def getDataTest(model):
      x_test = []
      x_testids = []
      ListOfFiles =
os.listdir('./data/recog/test_dir')
      for filename in ListOfFiles:
            img =
      cv2.imread('data/recog/test_dir/{}'.format(f
      ilename))
            x_test.append(cv2.resize(img, (224,
      224)))

            idval = filename.split()
            idstr = "".join( idval[:-1] )

            x_testids.append( idstr )
      x_test = np.array(x_test, np.float16) / 255.

      Xtest = model.predict(x_test,
batch_size=128)
      return Xtest, x_testids
```

```
if __name__ == "__main__":
      #point to load model from recognition
challenge model
      model = load_model('models/submit1.h5')

      Xtrain, XtrainIDs = getDataTrain(model)
      Xtest, XtestIDs = getDataTest(model)

      #prepare for submission
      matchingRecords (Xtest, Xtrain, XtrainIDs,
XtestIDs)
```

GOOGLE LANDMARK RECOGNITION

Evaluation Metric

Global Average precision (GAP) is also known as micro Average Precision

$$GAP = \frac{1}{M} \sum_{i=1}^{N} P(i) rel(i)$$

where:

- N is the total number of predictions returned by the system, across all queries
- M is the total number of queries with at least one landmark from the training set visible in it (note that some queries may not depict landmarks)
- $P(i)$ is the precision at rank i
- $rel(i)$ denotes the relevance of prediciton i: it's 1 if the i-th prediction is correct, and 0 otherwise

[Copied from competition site]

Submission format

id,landmarks
000088da12d664db,8815 0.03
0001623c6d808702,
0001bbb682d45002,5328 0.5
etc.

Winning Solutions

- Get global descriptor to get general features
- Use local descriptors to get local feature and improve the recognition.

3rd place

- Same as in 6th place (**Landmark retrieval competition**)
- Generalized mean pooling provides CNN-based global descriptors which provides vector and perform knn.
- Perform IDF-like norm on the vector per class.
- Exploit geometric information in crafting local descriptors using DELF features and ASMK with built-in inverted file structure for fast querying.
- More detailed description is found here {https://drive.google.com/file/d/1NFhfkqKjo_bXM-yuI3KbZt_iHRmiUyTG/view}

Delf { https://arxiv.org/abs/1612.06321 }

Geometric verification { http://www.robots.ox.ac.uk:5000/~vgg/publications/2007/Philbin07/philbin07.pdf }
ASMK {https://hal.inria.fr/file/index/docid/864684/filename/iccv13_tolias.pdf }

{ https://www.kaggle.com/c/landmark-recognition-challenge/discussion/60112 }

4th place

1. The author used a number of pretrained (ResNet34, ResNet50, ResNet101, ResNet152, ResNeXt101, InceptionResNetV2, DenseNet201).

- They performed data preprocessing cropping to 224 x 224, resizes, scales, shifts, rotates,and flips.
- initially 128 x 128 crops and then increased crop size up to 256 x 256.
- CNN training using fast.ai library.

2. Use data from landmark retrieval and train a ResNet50 to classify images as either being from the retrieval challenge or the recognition challenge.
A simple heuristic: if is_from_another_competition_probability > 0.95: confidence *= 0.1;

3. Train a ResNet50 to classify images as images with landmark or no landmark. The heuristics is same as in step 2.

4. A trick to improve result

- Convert images to vectors using pre-trained network
- For each image from test set they found K closest images from train set (K=5)
- For each class_id we computed: scores[class_id] = sum(cos(query_image, index) for index in K_closest_images)
- For each class_id we normalized its score: scores[class_id] /= min(K, number of samples in train dataset with class=class_id)
- label = argmax(scores), confidence = scores[label]

5. kNN with features from local crops

- Obtain 100 local crops from the image.
- Filter crops without landmark by identifying crops with landmarks using step 3.
- They used the technique at step 4 with K=20 for every local crop from query image (this is an image from test dataset).
- For every image from the test set they got a set of local crops (up to 100) and a list of pairs (label, confidence) for each crop.
- "For each class_id we computed its score as a sum of 5 - class rank for each crop. Class rank means the position of this class_id in the sort order of pairs (label, confidence)."
- "If at least one class rank was 5 or more, then we reset the entire sum. Thus, based on this technique we were able to choose approximately one thousand examples with high confidence in our prediction."

Merging:
They had used these heuristics:

- "Compute confidence score for each label using predictions from steps 1-4 as follow: score[label] = label_count / models_count + sum(label_confidence for each model) / models_count. Here label_count is a number of models where the prediction with max confidence is equal to the label".
- "We also used each prediction from step 5 with confidence = 1 + confidence_from_step_5 / 100"

{ https://www.kaggle.com/c/landmark-recognition-challenge/discussion/57896 }

10th place

1. The author created a number of neural architectures (ResNet, Wide ResNet, Inception-v3, DenseNet).

The training time augmentation include random resized crops, color jittering, horizontal flipping. CNNs were trained on 224x224 crops.

2. They trained their model using ArcFace loss { https://arxiv.org/abs/1801.07698 }.

3. inference step, Obtain a centroid of each class

- Get a sample of 100 image from each training class.
- Calculate the mean vector
- Normalize the vector

They did test time augmentation by averaging 10 crops giving a boost of 0.1 GAP.

- An image is passed as input
- This output a vector which is compared to all centroid by cosine distance.
- The class of the nearest centroid becomes the class of the image.
- If the distance value > 0.25 then the image has no landmarks.

4. Ensemble by majority voting

{ https://www.kaggle.com/c/landmark-recognition-challenge/discussion/58050 }

My Solution

- Due to huge data size, perform stratified sampling with 50 images per label as the training data.
- Best performing model on a littled sample was using pre-trained network with inception v3.
- Keep the first 249 layers frozen, and add a dense network with a few batch normalization subsequently matching the number of unique labels in the softmax layer.
- Unbalanced data set {https://www.kaggle.com/c/landmark-recognition-challenge/discussion/55569}
- Handling >15000 labels in a neural network?
  - Hierarchical softmax
  - Negative sampling
- Given the number of parameters and inception v3 requiring 224 x 224 images as input. We run out of **computational resources**.

Predicting Images With no Labels

The softmax layer at the tail of a neural architecture produces a one-hot representation which is a approximation of the probability distribution.

However, an image without a label is likely to belong to every label in the one-hot encoding vector. This is when the entropy is at the highest.

With careful thresholding, it is possible to identify instances with no labels. The best value of the threshold can be obtained by hyperparameter search.

This is a way of achieving **multi-label classification**.

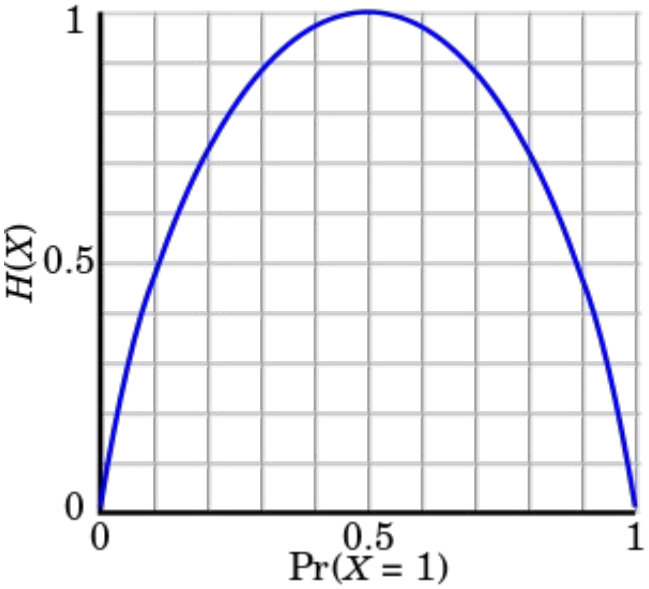


Figure 15: Entropy vs probability

[Image: Wikipedia]

Practical Considerations

- Parallelism / Concurrency (Python-specific)
  - By model e.g mergesort
  - By data
    - For process-bound compute use multiprocessing to parallelize your solution.
    - For IO-bound
      - Use asyncio for Python 3
      - Use gevent or greenlet event-based processing in Python 2

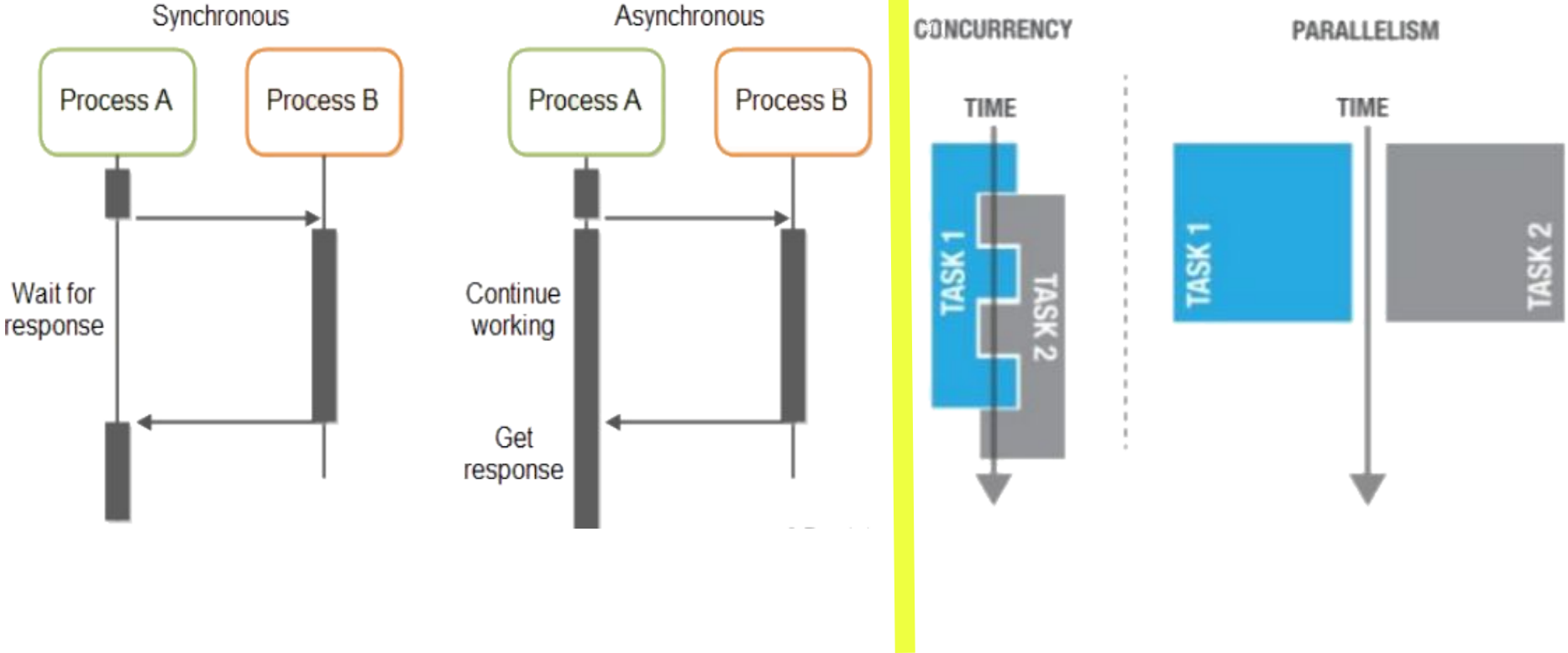


Figure 16: Parallelism vs Concurrency

{ https://drive.google.com/file/d/1vv-GoTGD1LYLq9QK9EW4NXGgspNp3ZSL/view }

- Batch processing
- Careful coding
  - Unallocate memory as soon as possible.
  - Avoid copying as much as possible. Do things in-place wherever necessary.
  - Vectorize wherever possible.
  - Disable GUI window. If necessary
- Use JIT (pypi) if you need to squeeze out more performance. At the moment of writing this tutorial, support for numpy is **not clear.**

```
import multiprocessing
from multiprocessing import Pool

def solution (traindata, testdata ):
     .
     .
     .
     return resultDF1, resultDF2


def convertToParallelSolution (ind, traindata, testdata, numOfProcesses):
     totalnumberOfSamples = len( testdata )
     numOfBins = round ( totalnumberOfSamples / numOfProcesses ) + 1
     start =  int (ind * numOfBins )
     end = int (start + numOfBins )
     result = ()
     if end >= totalnumberOfSamples:
         end = totalnumberOfSamples
     if end <= start:
         return result
     if end > start:
         result = solution (traindata, testdata[start : end].reset_index( ) )
     return result
```

```
def parallel_solution (traindata, testdata, numOfProcesses=64 ):
     pool = Pool(processes=numOfProcesses)              # start 64 worker processes
     # launching multiple evaluations asynchronously *may* use more processes
     multiple_results = [pool.apply_async(convertToParallelSolution, args=(ind, traindata,
testdata, numOfProcesses, )) for ind in range(numOfProcesses)]
     tresultDF1 = pd.DataFrame(columns=["id", "images"])
     tresultDF2 = pd.DataFrame(columns=["id", "images"])
     for res in multiple_results:
         rowObj = res.get()
         if rowObj:
                  resultDF1, resultDF2 = rowObj
                  tresultDF1 = pd.concat([tresultDF1, resultDF1])
                  tresultDF2 = pd.concat([tresultDF2, resultDF2])
     #write to file
     tresultDF1.to_csv('tresultDF1.csv', encoding='utf-8', index=False)
     tresultDF2.to_csv('tresultDF2.csv', encoding='utf-8', index=False)


if __name__ == "__main__":
     traindata = pd.read_csv('input/trainDp2.csv')
     testdata = pd.read_csv('input/testDp2.csv')
     print "Data is fully loaded"
     parallel_solution (traindata, testdata )
```

Conclusions

- Unsupervised learning can be difficult.
- Transfer learning is here to stay.
- CNN in its many variants are the state of the art in implicit or explicit object recognition.
- **Representation learning** is very important in learning.
- The use of **magic number** can be obtained by forms of leaderboard probing.
- Even with deep learning, you still need some **feature engineering.**

Thanks for listening

https://github.com/kenluck2001?tab=repositories

@kenluck2001

https://www.linkedin.com/in/kenluck2001/

kenneth.odoh@gmail.com

References

1. https://pymotw.com/2/threading/
2. https://stackoverflow.com/questions/3044580/multiprocessing-vs-threading-python
3. https://askubuntu.com/questions/668538/cores-vs-threads-how-many-threads-should-i-run-on-this-machine
4. http://people.ciirc.cvut.cz/~hlavac/TeachPresEn/11ImageProc/14WaveletsEn.pdf
5. https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf
6. ResNext code, https://github.com/facebookresearch/ResNeXt
7. https://drive.google.com/file/d/1NFhfkqKjo_bXM-yuI3KbZt_iHRmiUyTG/view
8. Thibos L.N, Fourier analysis for beginners, 2014
9. http://infolab.stanford.edu/~ullman/mmds/book.pdf