Bilkent University

Department of Computer Engineering

CS-425
Algorithms for Web-Scale Data

Final Report
Image Similarity Analysis with LSH

Group Number: Group 11

Group Members: Kerem Ayöz 21501569

Muhammed Safa Aşkın 21502860

Mert Epsileli 21502933

Contents

1. Introduction

Finding the similar pairs of an image is a simple problem when pairwise comparison applied. The time complexity of pairwise comparison is simply $O(n^2)$. In this project we've tried to find the similar pairs of a particular image with significantly shorter time complexity which is O(n). Because for datasets with more than 1.000 images this complexity difference causes extreme performance gain. In order to find similar pairs, We've used the Locality Sensitive Hashing with various feature extraction techniques to generate binarized arrays of images or to extract feature vectors and various distance metrics.

Feature Extraction Methods:

- Edge Detection
- Segmentation
- Sobel Filtering
- CNN

Distance Metrics:

- Jaccard Distance
- Cosine Distance (Random Hyperplanes)
- Euclidean Distance

Although lots of different algorithms have used, some of them has eliminated for some problems. The detailed descriptions of algorithms and results of our experiments are listed in detail below.

2. Problem Description

In this project, we've focused on decreasing the cost of finding similar pairs of an image. As described above current pairwise comparison algorithms takes $O(n^2)$ time complexity and this causes huge performance loss for datasets with lots of images. Furthermore, with pairwise comparison it is easy to find the exact copy of an image but it's hard to define similar pairs of a particular image. For example a dataset with 10.000 images with 300-300 resolution contains 90.000.000 pixels and it takes huge amount of pixel comparison for pairwise algorithm (90.000.000 x 90.000.000). Therefore, we've tried to approach the finding similar pairs problem from another perspective: **Locality Sensitive Hashing**

3. Feature Extraction Methods

Computers are understanding and processing images as three matrices filled with values from 0 to 256. Each element of these matrices determines the tones of RGB (red-green-blue) for particular pixel. First matrix for shades of red, second for green and third for blue. In this section we've tried to describe our ways to manipulate these matrices to extract features from images.

3.1. Image Processing

The image processing techniques that we've tried to get better and meaningful (binarized image arrays, N dimensional feature vectors) result before applying LSH are listed below with the importance and advancement order.

3.1.1. Edge Detection

The simplest feature extraction method that We've used in our project is Edge Detection. According to its formal definition "Edge detection is an image processing technique for finding the boundaries of objects within images." [1] We thought the output of Edge Detection algorithm will be pure enough to directly use it in our project. However, results were not satisfactory enough for clarifying the features of images.

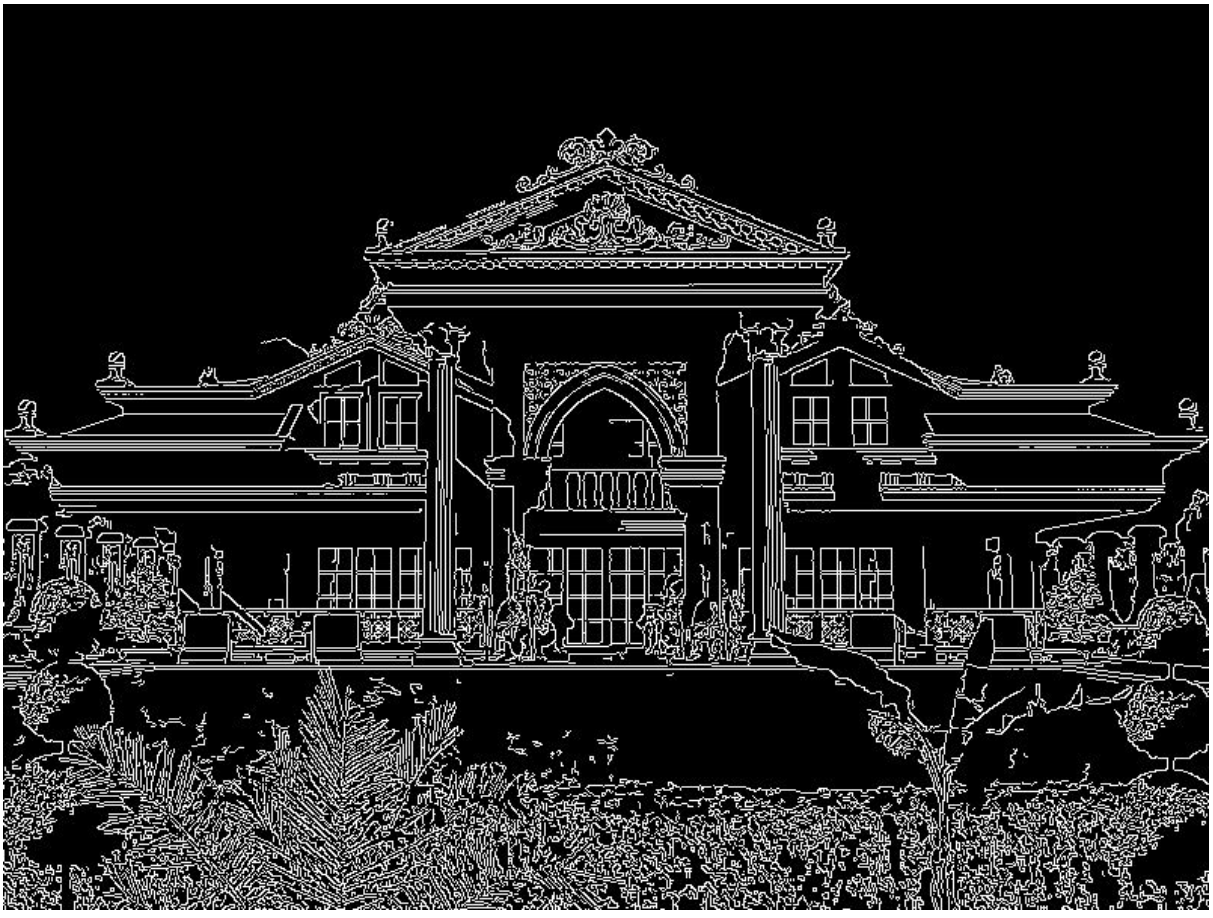


Fig.1. Result of Edge Detection Algorithm

We could easily binarize this outputs (Example: Figure 1) by representing white colors as 1 and black colors as 0 and apply LSH with Jaccard similarity directly. Although, the binarized structure of Edge Detection algorithm perfectly fits with our aim, we didn't use it. Because as it is seen from Figure 1 the output is noisy and includes lots of irrelevant

details. In short, this algorithm doesn't represent the relevant features of our images and did not used.

3.1.2. Segmentation

We've eliminated the Edge Detection algorithm because the outputs had lots of irrelevant details. In order to get rid of these details and obtain more pure images, we decided to apply simple segmentation to our datasets.

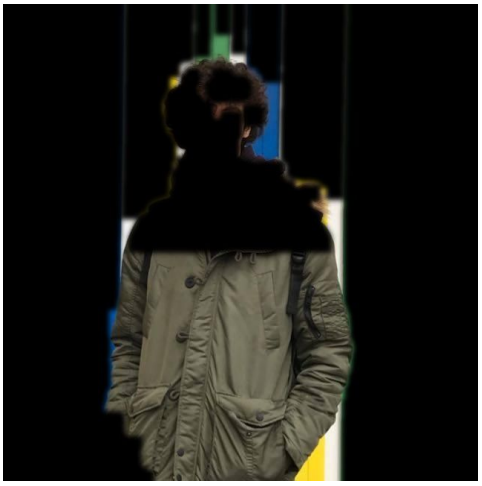


Fig.3. Segmented Image

Fig.4. Original Image

Since our segmentation algorithm is not machine learning based as its current versions, some results were not proper to use. The given examples (Fig.3 and Fig.4) above shows the result after segmentation applied to basic (not crowded) human image. Although the image was basic enough to easily segmentate human from the background, the result was unsatisfactory. Therefore, the Segmentation solution has also failed and didn't applied.

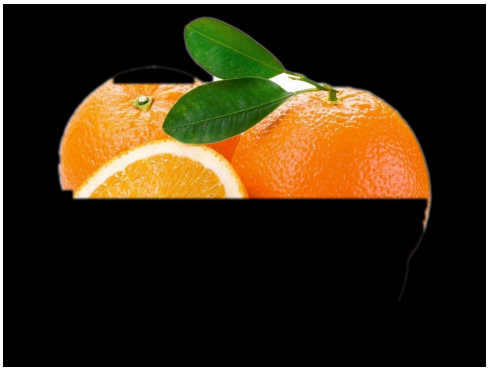


Fig.5. Segmented Image

Fig.6. Original Image

3.1.3. Sobel Feature Extraction

After the failure of Edge Detection algorithm we've started to implement our second algorithm called Sobel Filtering. These algorithms (Edge and Sobel) both detecting the boundaries of images and recoloring these boundaries as white and other parts black. However, there is a significant difference between two algorithms is Sobel Filtering algorithm recoloring the edges with respect to their importance.

Fig.7. Result of Sobel Filtering

As shown in Figure 7 the tone of edges are changing from white to gray with respect to their importance. Since LSH with Jaccard Similarity is working on binarized documents, these Sobel Filtering results were not applicable. In order to handle that problem we've tried to find a threshold value for tones. At the end, after working with various values, we've calculated 120 as threshold value. Pixels with values below 120 filled with 0 and above 120 filled with 1.

3.1.4. SIFT & KAZE

SIFT and Kaze are algorithms are find the key points in the images. While the number of key points in a image may change depends on image's size, type and resolution, the size of feature vectors of key points do not change. The size of a feature vector for a key point is 64 for Kaze and 128 for SIFT. Therefore, the SIFT algorithm gives more detailed feature vectors from Kaze algorithm.

Figure.8. Result of SIFT Algorithm Algorithm

Figure.9. Result of Kaze

Although these algorithms could find efficient feature vectors, we could not use them because the number of feature vectors was different for every image and we could not apply any hashing algorithm for them. For example, for Figure 9 Kaze algorithm finds 4807 key points. As shown above an image could be defined by 4807 feature vectors, so we eliminated these methods.

3.2. Convolutional Neural Network

Convolutional Neural Network (CNN) is a machine learning algorithm which combines image convolution with fully connected neural network. It is used for image classification and they are quite successful and popular. CNN's over perform simple neural networks because of the technique called "***Convolution***" which basically extracts the local features of an image. Convolution operation described in the following figures.

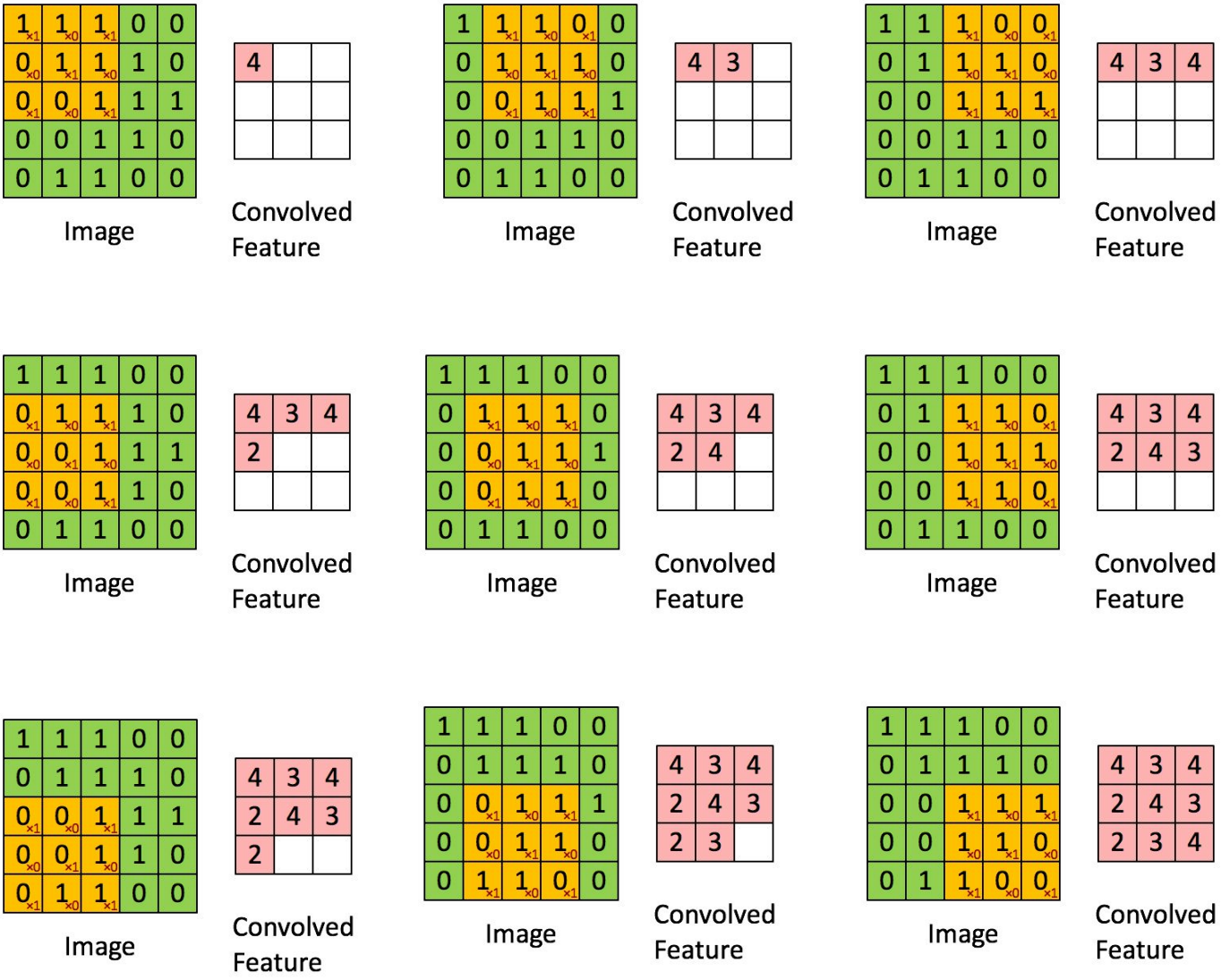


Fig.6. Example convolution operation

Image convolution has the following formula:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

Here f is input image given, g is 2 dimensional vector and it is called ***convolution kernel***. When we convolve our image with kernel, we get another 2 dimensional vector which describes extracted features of image f by applying the kernel g. In above example green matrix is input image f and yellow matrix is kernel g. The resulting image contains extracted features. Here is the example convolution operation that is applied to an building image:

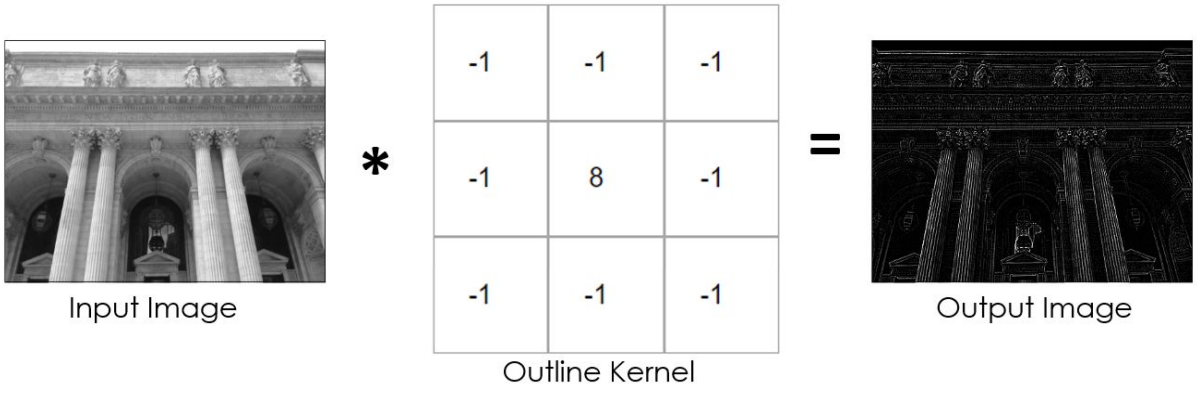


Fig.7. Result of Image Convolution

Here a kernel called outline kernel used to highlight large differences in pixel values. It is also called edge kernel since it extract edge features from image.

CNN's apply various kernels to given image at each convolution layer to learn hidden features of given image to classify it correctly. To extract these hidden features it applies convolution-activation-pooling operations one after another. After several times, it feeds these features into a fully connected neural network to classify these features.

There are thousands of CNN models available in the literature, however idea in all of them is same; extract hidden image features by applying convolution operation, then classify it with a neural network.

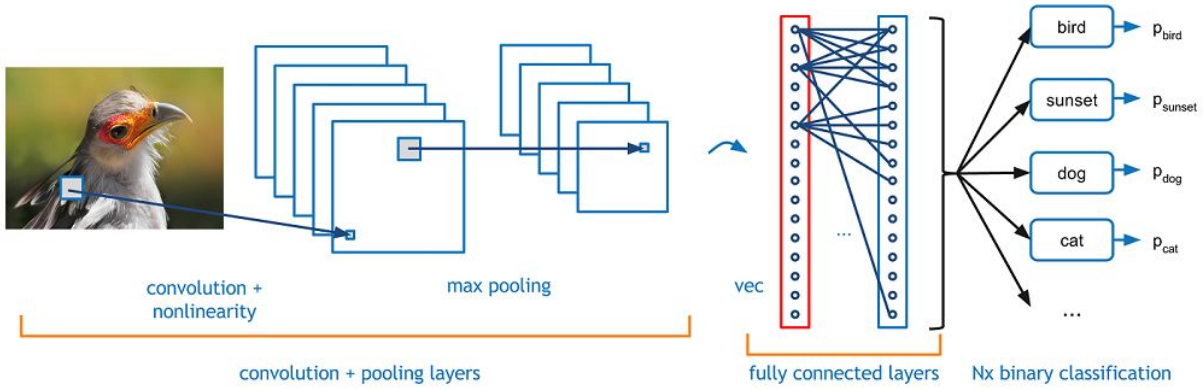


Fig.8. An example CNN architecture

We used CNN to extract hidden image features from the images in our dataset. We used ImageNet [x] dataset which is customly built and consists of 17 classes of images and each class contain at least 1000 images. In order to get meaningful image features, we used a pre-trained CNN model which is called VGG-16 [x]. Its %90.0 percent top-5 classification accuracy and model is simpler compared to others. It is trained on entire ImageNet dataset and we thought that it could extract meaningful hidden features from our images.

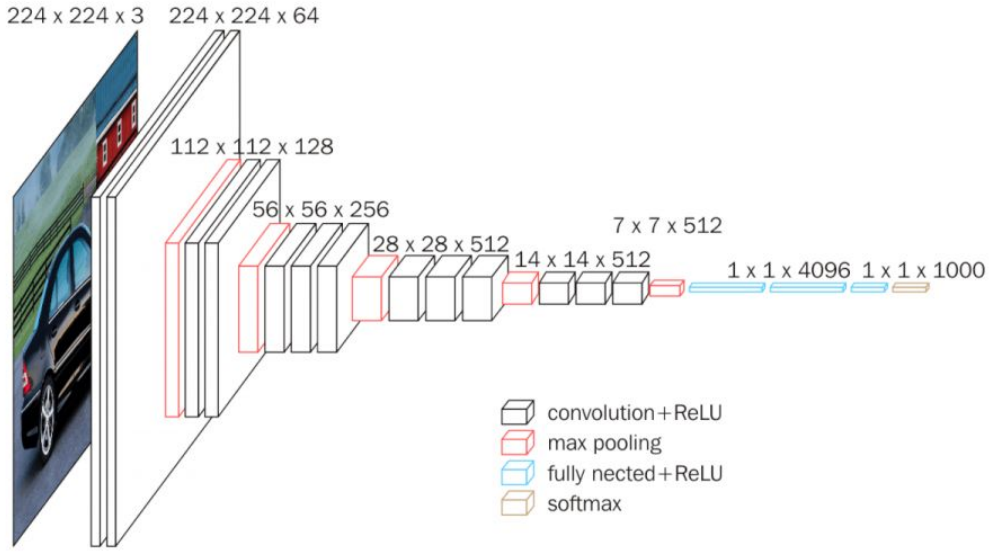


Fig.9. VGG-16 Architecture

We fed our images to the CNN and take the last convolution layer features for each image. We also changed the layers which we take their outputs as features to optimize the performance of our algorithms. We rescaled our images into (224,224) pixels to get equal number features for each image. For instance, if we take the last convolution layer as feature vector, we obtain a vector with (7, 7, 512) dimension. If we flatten that vector, we get 25.088 dimensional feature vector for each image. We could use these layer features vector directly in Euclidean LSH and Cosine LSH. We thought that we could use convolution layer features with Euclidean LSH and Jaccard LSH; fully-connected layer features with Cosine LSH.

In order to use convolution layer features with Jaccard LSH, we needed to binarize these feature vectors. We know that each of the dimension in a single feature vector represent a neuron. If a neuron value is zero it is called ***'not-fired'*** whereas if it has non-zero value it is called ***'fired'*** neuron. If a neuron is not fired, then its contribution to next layers are 0 since we take weighted sum of neurons in layer d to calculate neuron values in layer d+1.

By using this knowledge, we binarized the convolutional layer features by converting non-zero values to 1 and keeping zeros in their places. On the average %7 percent of the 25.088 features are non-zero in our dataset and we thought that it is a good candidate to be binary shingle of our images.

4. Locality Sensitive Hashing

Before using hashing, we defined every image as a point in N-Dimensional space. We used feature extraction methods listed in part 3 to create these points. Where N can be;

- Total number of pixels in single image after applying Sobel Filtering.
- Total number of elements in feature vector which is obtained from edge detection algorithm.
- Total number of elements in feature vector which is obtained from Convolutional Neural Network.
- Total number of elements in feature vector which is obtained from Kaze.
- Total number of elements in feature vector which is obtained from SIFT.

4.1. Jaccard Distance

Jaccard distance is 1 - Jaccard Similarity. Jaccard Similarity is calculated in following example;

$$A = [1,0,0,0,1,1,0]$$

$$B = [1,0,1,1,1,0,0]$$

$$|A \cap B| = [1,0,0,0,1,0,0] = 2 \text{ (There are two 1.)}$$

$$|A \cup B| = [1,0,1,1,1,1,0] = 5 \text{ (There are five 1.)}$$

$$\text{Jaccard Similarity} = 2 / 5$$

$$\text{Jaccard Distance} = 1 - 2 / 5 = 3 / 5$$

We used this method to find similarity between signature arrays of images. If jaccard distance is below a threshold value, we chose them as candidate pairs.

4.2. Euclidean Distance

One of the method which we used for hashing the points in N-dimensional space was Euclidean Distance. Firstly, we created a random line and found the projection of the image points on this line. We did these operations for all images. After that, every image was represented as a point on a line. We created a random point and computed the dot product between the random point and projection points to find unique hash values. We did this 100 times to be able to find most candidate pairs.

A) Create a Random Line and Find the Projection of the Image Points

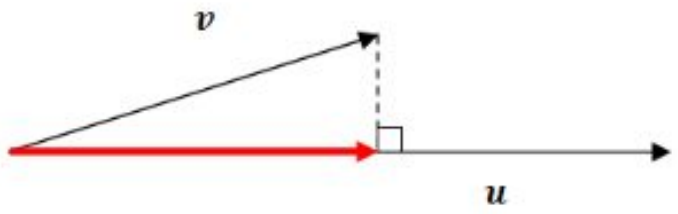


Fig.6. Calculation in 2 Dimension

Here v is the vector of an image point. u is the random line. We adapted it to N-Dimension. to make our algorithm more general. We find projection of image feature vector v on randomly generated unit vector u by applying scalar projection which uses dot product at its core.

$$\cos\theta = \frac{u \cdot v}{|u||v|} \qquad \cos\theta = \frac{\text{scalar projection}}{|v|} \qquad \frac{u \cdot v}{|u||v|} = \frac{\text{scalar projection}}{|v|} \qquad \frac{u \cdot v}{|u|} = \text{scalar projection} \qquad proj_u v = \left(\frac{u \cdot v}{|u|}\right)\frac{u}{|u|}$$

Fig.7. Vector projection formula derivations

B) Create a Random Vector to Create Unique Value for Every Point by Using Dot Product

Uniqueness of Points (Proof in 2 Dimension)

1. $ax + by = c$ $(A\ line)$
2. (m, n) $(A\ random\ vector)$
3. $(x_1, \frac{c-ax_1}{b}),\ (x_2, \frac{c-ax_2}{b})$ $(2\ different\ points\ on\ the\ line)$
4. $mx_1 + n * \frac{c-ax_1}{b} = mx_2 + n * \frac{c-ax_2}{b}$ $(Equality\ of\ dot\ product\ of\ two\ points)$
5. $\frac{m}{a} = \frac{n}{b}$ $(Condition\ for\ same\ value\ of\ different\ points)$

In N-dimensional (N >= 100), this probability is close to 0. Therefore, we can create unique values for different points by using this points.

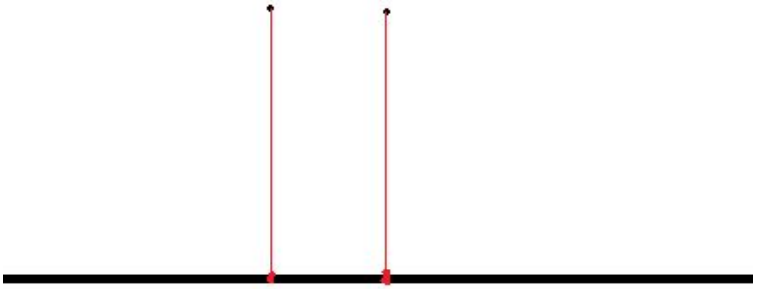
C) Find Hash Values by Using Unique Values and Find Candidate Pairs

Hash value = ((Unique Value) / z) (mod k)
(z is determined according to range of hash values.)
(k is determined according to size of hash table.)

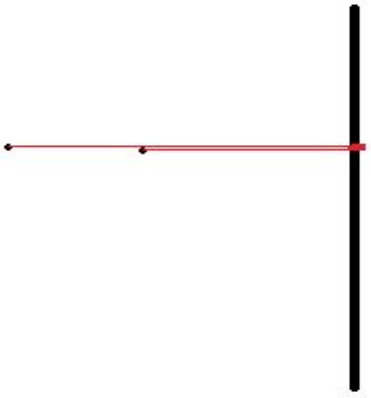
Group the values according to range of unique values and place them into the hash table. For example, if range of unique values is [0,10], we cannot say 1 and 2 are close to each other, so we cannot place them into same bucket. However, if range of unique values is [0,1000], we can place 1 and 2 into same bucket. After that, we choose the images in the same bucket as candidate pair.

D) Do All Steps 100 Times to Find All Candidate Pairs

We assume two images, which are similar, are defined by these two points.



In this case, their projection are not close to each other and we cannot choose them as a candidate pair. If the random line (black line) changes;



Their projection will become closer each other, so we can choose them as a candidate pair.

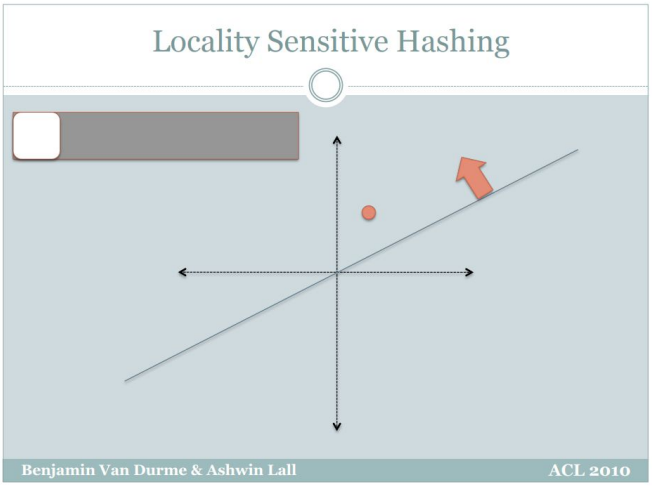
As a result, if we did steps 1-4 only one time, we may not find some candidate pairs. Therefore, we repeated these steps 100 times to find all candidate pairs.

4.3. Cosine Distance

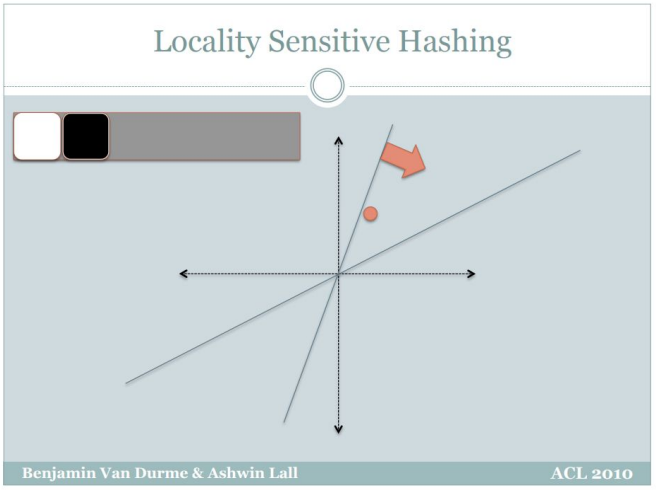
The main idea of the cosine distance method is dividing the N-Dimensional space regions by using planes. After that, finding the points in the same region and choosing them as a candidate pair.

Visualisation of Cosine Distance Method in 2 Dimension[3]
(1 for White Squares, 0 for Black Squares)

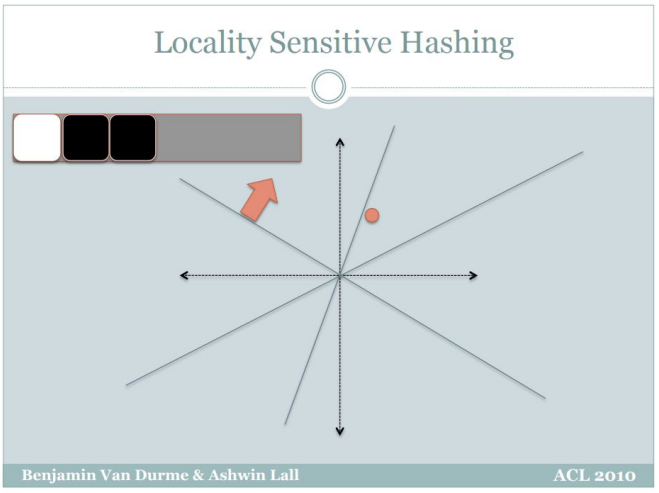
Firstly, draw a line and find whether the point is on the left side or right side of the line. If it is on the left side, add 1 to signature matrix. If it is on the right side, add 0 to signature matrix.

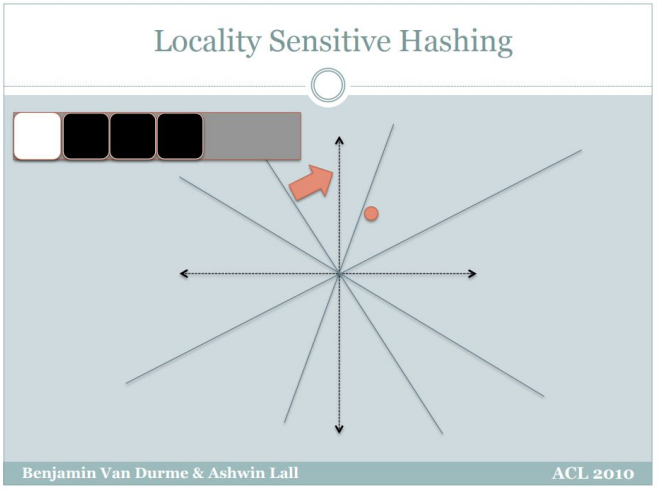


1. It is on the left side of the line. Signature matrix = (1)



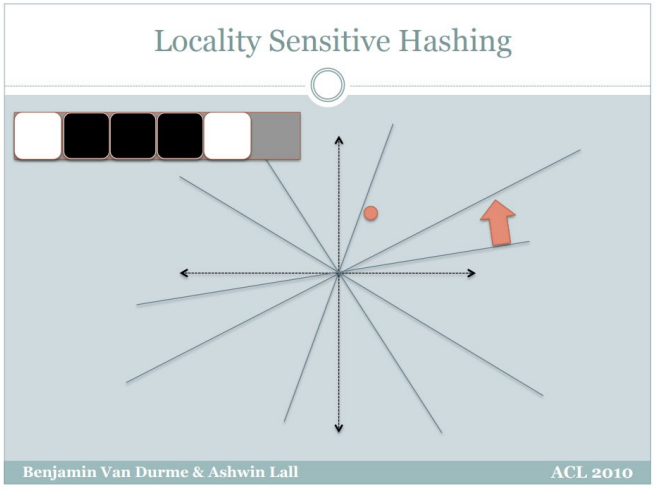
2. Draw a new line. It is on the right side of the new line.
Signature matrix = (1,0)

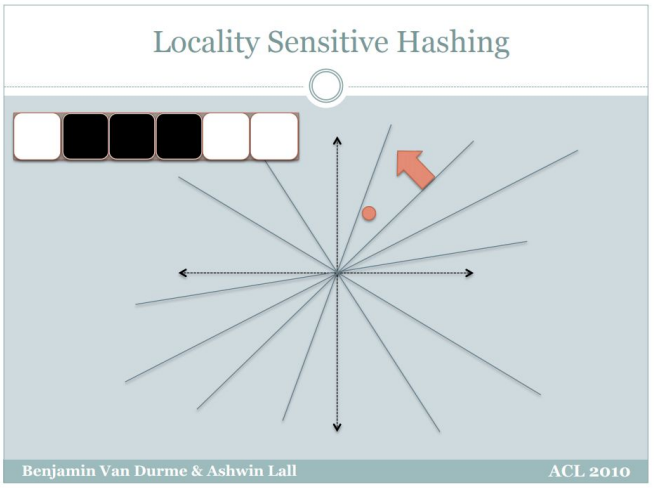




3. Draw a new line. It is on the right side of the new line.
Signature matrix = (1,0,0)

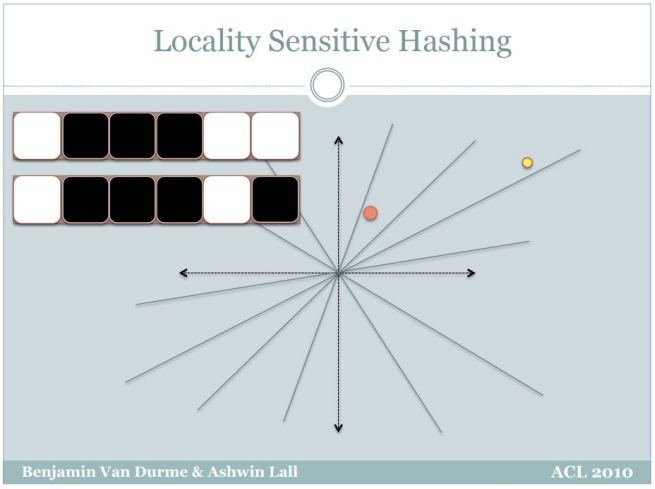
4. Draw a new line. It is on the right side of the new line.
Signature matrix = (1,0,0,0)

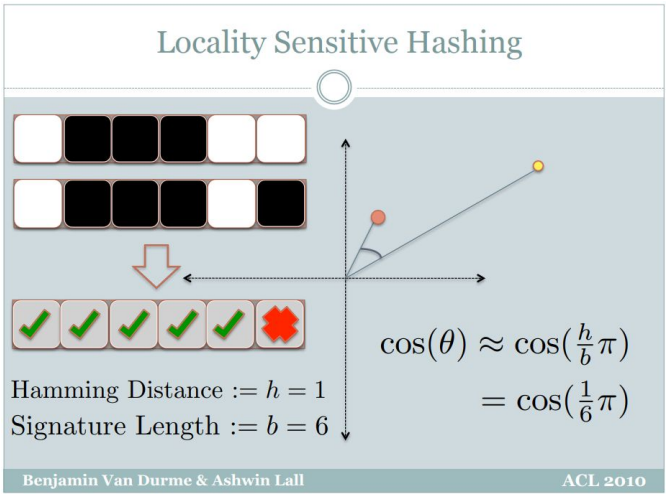




5. Draw a new line. It is on the left side of the new line.
Signature matrix = (1,0,0,0,1).

6. Draw a new line. It is on the left side of the new line.
Signature matrix = (1,0,0,0,1,1)





7. Do same things for another point.
Signature matrix for red point = (1,0,0,0,1,1).
Signature matrix for red point = (1,0,0,0,1,0).

8. Calculate the Cosine Similarity by Hamming Distance formula given above.

In the document, hamming distance between signature matrices is used to find cosine distance as in the above example. However, we did min hashing and locality sensitive hashing for signature matrices to find candidate pairs. We used CNN image features, but this time we extracted the features at the prediction layer. We thought that the cosine distance between the prediction layer of the CNN is smaller between same type of images. The prediction layer represents a probability distribution over all available 1000 classes. We used this feature to test the performance of cosine distance LSH.

5. Results

5.1 Jaccard LSH

RESULTS OF CNN FEATURE EXTRACTION ALGORITHM

Experiment 1:

In this experiment number of classes was fixed variable, number of bands and number of rows were free variable.

Ingredient Numbers			Candidate Pair Numbers				
Class	Band	Row	True Positive	False Positive	True Negative	False Negative	Precision
3	24	5	24870	19263	7424290	3733479	56%
3	20	6	4464	1649	5410170	2175619	72%
3	15	8	175	55	4443584	2048088	75%
3	12	10	20	2	3153714	1568166	91%

Experiment 2:

In this experiment number of rows and bands were fixed variable, number of classes was free variable.

Ingredient Numbers			Candidate Pair Numbers				
Class	Band	Row	True Positive	False Positive	True Negative	False Negative	Precision
2	24	5	9452	1554	2266311	2275336	86%
3	24	5	24870	1649	7424290	3733479	56%
4	24	5	17543	16384	13154785	4467941	51%
5	24	5	20489	20263	22689421	5752205	50%
6	24	5	23028	36344	31670556	6459502	39%
7	24	5	61634	183352	41128808	7024247	35%
8	24	5	88659	213076	54794972	7965358	29%

RESULTS OF SOBEL FEATURE EXTRACTION ALGORITHM

Experiment 1:

In this experiment number of classes was fixed variable, number of bands and number of rows were free variable.

Ingredient Numbers			Candidate Pair Numbers				
Class	Band	Row	True Positive	False Positive	True Negative	False Negative	Precision
3	24	5	571	824	6415562	3274046	41%
3	20	6	118	43	4416561	2231759	73%
3	15	8	6	1	4870120	2477734	86%
3	12	10	4	0	6027672	3022709	100%

Experiment 2:

In this experiment number of rows and bands were fixed variable, number of classes was free variable.

Ingredient Numbers			Candidate Pair Numbers				
Class	Band	Row	True Positive	False Positive	True Negative	False Negative	Precision
2	24	5	2582	1280	1673533	1685726	67%
3	24	5	4720	5750	5095693	2594687	45%
4	24	5	6374	13626	9499469	3204021	32%
5	24	5	10097	28974	20470507	5478867	50%
6	24	5	14604	63671	32249474	6802532	39%
7	24	5	25568	133718	43422845	7528974	18%
8	24	5	13208	78887	58305222	8691419	16%

While we were doing these experiments, we had 17 different classes with different number of images. We have chose random classes for every trial, so total image number and total candidate pair number changed for every trial. (Result: True Positive + False Positive + True Negative + False Negative is not equal for trials because number of images varies for different classes)

5.2 Euclidean LSH

We couldn't obtain meaningful results for Euclidean LSH.

5.3 Cosine LSH

Since we've obtained most optimum results with 3 classes for Jaccard LSH, we have only made Cosine LSH with 3 classes. The result of Cosine LSH experiment was surprisingly perfect. We've found more than 700.000 pairs for 3 classes and %77 of these pairs were true (precision).

Ingredient Numbers			Candidate Pair Numbers				
Class	Band	Row	True Positive	False Positive	True Negative	False Negative	Precision
3	12	10	747603	294540	7146177	2995226	77%

6. Discussion and Conclusion

Our aim in this project was finding similar images in dataset with O(N) time complexity. We needed to use LSH (locality sensitive hashing) to achieve our goal. We also needed to extract the features of the images. Therefore, we used different methods.

6.1 Edge Detection

We obtained a signature array for every image by using edge Detection Algorithm. However, this array contained insignificant details. As a result, we eliminated this algorithm because when we used these signature arrays, the program found many irrelevant candidate pair.

6.2 Segmentation

We obtained a signature array for every image by using Segmentation Algorithm. However, when we looked at the images which is obtained by using segmentation algorithm, we noticed that this algorithm could not recognize the images well. For example, it could not differentiate the image from the background. As a result, we eliminated this algorithm because we could not obtain efficient signature arrays.

6.3 Sobel Feature Extraction

We obtained efficient signature arrays for every image by using Sobel Algorithm. There was not any insignificant details. In addition, it could recognize the images very well. This algorithm was appropriate for our project. As a result, we tried these signature arrays with LSH methods and got results better than we expected. Therefore, Sobel is one of the algorithms which we used.

6.4 SIFT & Kaze

SIFT and Kaze were creating a signature array for every key point in an image. However, images were including different numbers of key points. As a result, we could not apply LSH for images because every image had different number of signature arrays. Therefore, we eliminated these algorithms.

6.5 CNN Feature Extraction

We found that CNN is quite powerful when it comes to extracting features of an image. We obtained highest accuracy among other feature extraction methods with CNN. We used different layers to obtain features of image. To apply Jaccard LSH, we used binarized convolutional layer features. To apply Cosine LSH, we used prediction layer output to test our algorithm's performance. In future work, we could use combined layer features to extract features. For instance we could use a technique used in style transfer algorithms [4] which extracts content and style features of image, simply edge and color features.

6.6 Jaccard LSH

We tested our Jaccard LSH algorithm with image features extracted with CNN and Sobel Filtering. Some of the results are described as following:

This pair is found to be candidate pair by our Jaccard LSH algorithm and it is an example of true positive pair. Here result is intuitive, both of the images belong to race cars and our algorithm found them similar.

This pair is found to be candidate pair by our Jaccard LSH algorithm but they are belonging to different classes of images. It is an example of a false positive sample. We thought that CNN focused on color features in these samples and generated similar feature vectors for both of the images, thus we found them candidate pair. Another possibility is that this error occured due to the nature of Min-Hashing algorithm.

This pair is found to be candidate pair by our Jaccard LSH algorithm but they are belonging to different classes of images. It is an example of a false positive sample. However, if we look at the images intuitively, we could say that these images are similar, but they belong to different classes in our dataset. First image belong to race car class while second image belong to fire truck class. Here, we could see that for some false positive examples, our algorithm actually did a great job to

identify similar images. Our performance metric just looks for image labels thus we could miss these kind of images without explicitly looking to the candidate pair images.

This pair is not found to be candidate pair by our Jaccard LSH algorithm even both of the images belong to the same class. It is an example of false negative sample. However, because of the nature of our feature extraction algorithms extracting similar feature vectors for these type of examples are quite difficult. They are both clock images but we cannot find them similar. Intuitively our algorithm did a great job since they are not similar images, they only share labels. Again our performance metric caused a problem, it misses the success of our algorithm.

6.7 Euclidean LSH

We implemented Euclidean LSH as described above. We tested our algorithm with toy examples however we could not be able to test it with the actual image feature vectors. We had two reasons for it. First one is we had performance issues such that when we feed large shingle matrix to Euclidean LSH algorithm we got poor running time performance. Second one is we did not have enough time to generate appropriate feature vectors to test with Euclidean LSH algorithm. Testing CNN's convolutional layer features with Euclidean distance is not meaningful

since CNN does not extract this features in that way. Thus we could not obtain meaningful results from Euclidean LSH algorithm.

6.8 Cosine LSH

We implemented Cosine LSH with Random Hyperplanes just described above. We extracted prediction layer features from CNN and feed it into the Cosine LSH algorithm. Then we obtained a binary matrix for each image with Random Hyperplane method. Then we applied Min-Hashing and Locality Sensitive Hashing to that matrix. We concluded that this method gives the best results among all other methods.

There are 2 main reasons why we get such good results;

- Prediction layer features are quite powerful since we use Image Net dataset and VGG-16 model to extract features. VGG-16 has %90 percent top-5 accuracy which means it correctly extracts features at prediction layer with %90 percent accuracy. Since we eliminated problems that we encountered with the features extracted with different layers (similar images-non similar feature vectors and non similar images-similar feature vectors).

- Cosine LSH method is quite powerful since we can define as many hyperplanes as we want. Theory proves that when the number of hyperplanes increase, probability of our algorithm find two cosine neighbor points similar increase also. We conclude that the best method to find similar images with Locality Sensitive Hashing is Cosine Distance Locality Sensitive Hashing, which applies random hyperplane method with min-hashing and locality sensitive hashing.

7. References

[1] "Figure 2f from: Irimia R, Gottschling M (2016) Taxonomic revision of Rochefortia Sw. (Ehretiaceae, Boraginales). Available: https://reference.wolfram.com/language/ref/CosineDistance.html [Accessed: May. 12, 2019].

[2] C. K.-12 Foundation, "12 Foundation," *CK*. [Online]. Available: https://www.ck12.org/book/CK-12-College-Precalculus/section/9.6/. [Accessed: May. 12, 2019].

[3]Van Durme, B. and Lall, A. [Online]. Cs.jhu.edu. Available: http://www.cs.jhu.edu/~vandurme/papers/VanDurmeLallACL10-slides.pdf [Accessed: May. 12, 2019].

[4] L. Gatys, A. Ecker, and M. Bethge, "A Neural Algorithm of Artistic Style," Journal of Vision, vol. 16, no. 12, 2016. [Accessed: May. 12, 2019].