Technion - Computer Science Department

Center for Intelligent Systems

-Project Book-

Large Databases for Object Recognition

Project by:

Cirill Aizenberg, Dima Altshuler

Supervisor:

Erez Berkovich

Table of contents

Introduction

In the recent years everyone knows that Internet contains infinite amount of free data that can be used for large number of applications. Along all the free data out there, we can find billions of images that can be widely used. But there are two major problems that preventing easy access to this reach incredible database, one is that we need a technique to recognize the visual content of the images and the second is the ability for fast search of the huge amount of images.
In our project we implemented one of the new and the promising algorithms that can be found in the computer science community, we will review all the steps of the algorithm, describe the logic behind each step and finally compare the results to other available simple approaches.

Image Recognition Process

Before we will describe the algorithms in depth, we will make fast overview over some operations over our image database. These operations are standard for almost all the modern image recognitions algorithms and they are necessary for our algorithm in order to make his execution a lot more efficient.

Features extraction

The first problem we need to solve before we can apply our algorithm is how we can analyze the very large amount of image data. If for example we have billion images of even small size of 32Kb we will soon understand that holding and analyzing all the data is almost impossible. This is why we will use a machine learning techniques to reduce the data size of each image and hold only the relevant data of each image. The approach that was chosen for this mission is to convert each Image to Gist descriptor [6] (a real valued vector that describes orientation energies at different scales and orientations within an image). With the help of the Gist descriptor we will hold only 512 bytes per image, and that will be enough for us to hold all the relevant data needed for next steps.

Dimension reduction

Even thought we can reduce a large image to a 512 dimension vector of real data, we can go one step further and reduce the small Gist descriptor to only 32 most important dimensions of the vector. The procedure we use in this step called Principal component analysis [4]. PCA is widely known statistic linear method that can convert a set of correlated variables into a set of values of linearly uncorrelated variables. With a use of PCA we not only achieve very small vector of data but we also reduce possible noise that can interrupt us in further step because of the variety of the dimensions.

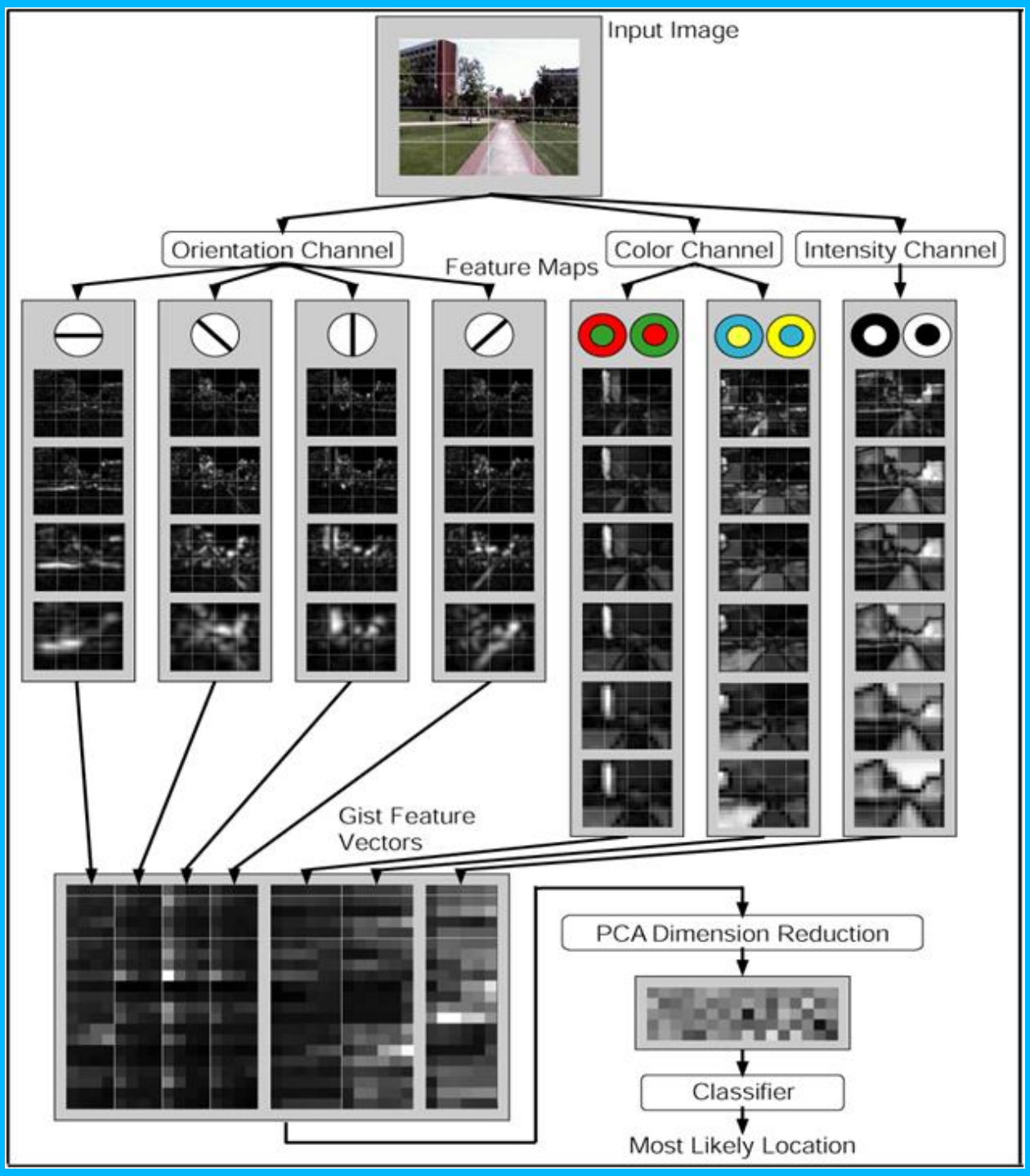

Figure 1: Summary of the preparation we done on image. First we separate each image to different channels and then we reduce dimensions and the result we send to the algorithm classifier.

Semi-supervised Learning Algorithm

As mentioned in the introduction, we have access to billion of images over the internet, some of the images are labeled (with human help) but a lot of them are unlabeled. The main idea behind the SSL algorithm [1] is that we can use not only the labeled data (supervised algorithms) but also the unlabeled data. We will see that if the algorithm will take in to the consideration the density of the whole data we can receive much better results.

Before we will describe the algorithms in depth, we will make fast overview over some operations over our image database. These operations are standard for almost all the modern image recognitions algorithms and they are necessary for our algorithm in order to make his execution a lot more efficient.

Classification using Graph Laplacian

Our purpose in this section is to define a way to find classification function f that will be binary classifier for all unlabeled data, in other words we want to know for each unlabeled point if that point classified as some specific type.

In order to find classification function we will use graph-based semi-supervised learning. We will define graph Laplacian L that will define a smoothness operator that takes into account the unlabeled data.

The combinatorial graph Laplacian is defined as:

$$L = D - W$$

$W_{ij} = \exp(-\| x_i - x_j \|^2 / 2\varepsilon^2)$ x – data point.

$D_{ii} = \sum_j W_{ij}$

Now we want to find f that minimizes:

$f^T Lf + (f-y)^T \Lambda (f-y)$
$f^T Lf$ – smoothness $(f-y)^T \Lambda (f-y)$ – agreement with labels

y – labels
Λ – if point is labeled $\Lambda_{ii} = \lambda$ otherwise $\Lambda_{ii} = 0$

The solution is:
$(L-\Lambda) f = \Lambda y$ $n \times n$ system (n is the number of points)

Alternative way finding Classification function

We do not need to work with the whole graph Laplacian instead we can find smooth vectors that will be linear combination of eigenvectors U with small eigenvalues. We can significantly reduce the dimension of f by requiring it to be of the form $f = U\alpha$ where U is a $n \times k$ matrix whose columns are the k eigenvectors with smallest eigenvalue.

If we choose some values for k like 100 the optimal α will be solution to $k \times k$ system $(\Sigma + U^T \Lambda U)\alpha = U^T \Lambda y$

Problem finding Classification Function

As we saw in previous section in order to calculate the Classification function we can choose one of two approaches.

- Direct resolving – we have to invert L matrix
- Use of eigenvectors – we have to find the eigenvectors by diagonalizining L

If we take for example 100 million images we will find out that it's impossible to invert or diagonalize 100 million x 100 million matrix. Because our algorithm developed for large scale databases, 100 million images is a reasonable number for database and this is why there is need to find another solution for finding Classification function.

The use of Eigenfunctions

The strength of the SSL is the ability to overcome the technical obstacle of calculating the Classification function. The main idea is to find approximation of the k smallest eigenvalues by using the eigenfunctions of the graph Laplacian.

First we will look at the data set as the number of points goes to $n \to \infty$
This way we will see the density of the data. We define an operator that measures smoothness of a continuous label function and then analyze the eigenfunctions of this operator.

A key assumption that we make in this method is that the input distribution is separable, so for our images we assume that the joint density is modeled as a product of 32 distributions.

For each marginal distribution we will compute the eigenfunctions. Given a large set of data points we will create a histogram h_i which will be an approximation to the true marginal.

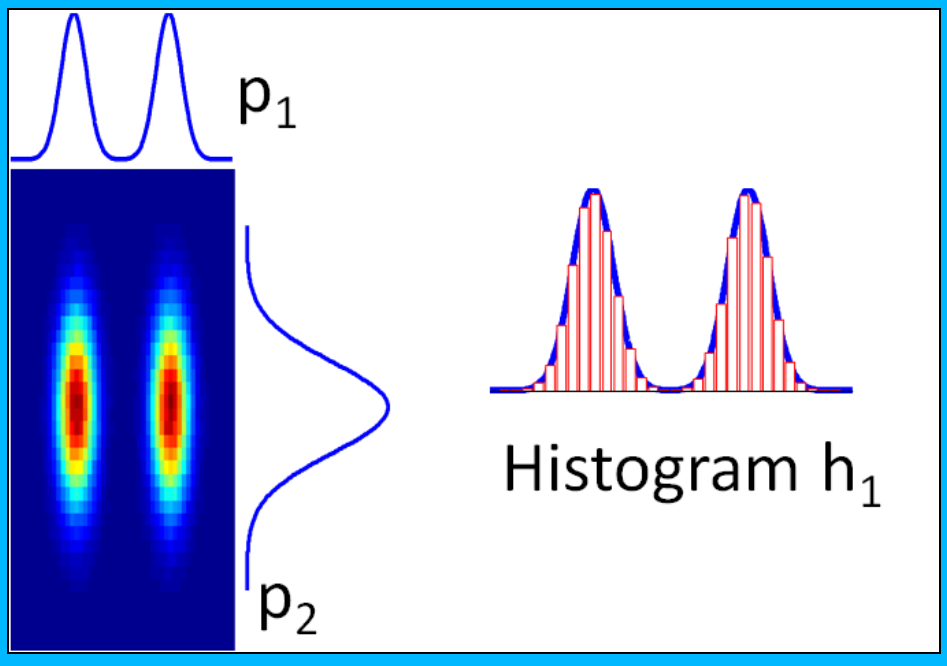

Figure 2: Example of density distribution of data and the histogram of one of the dimensions.

After we calculate all the histograms we can solve for the eigenfunctions of each 1D distribution. We solve for values of the eigenfunction and their associated eigenvalues at the locations of the bin centers. If we will chose bin number as 20 we will get small system 20×20 that is easily solved.

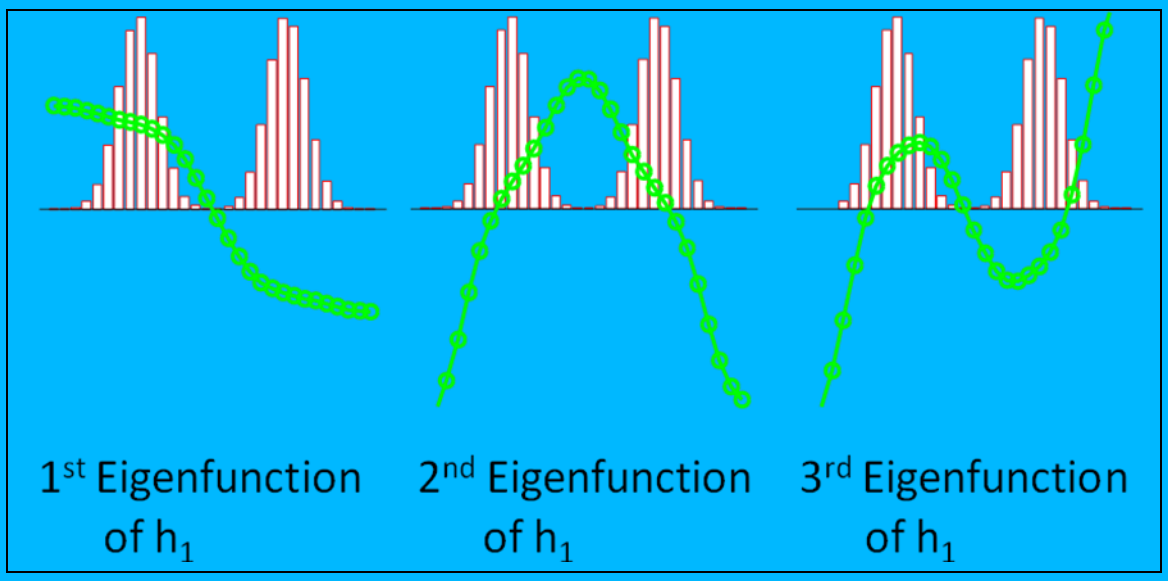

Figure 3: Example of three eigenfunctions.

After we have all the eigenfunction for each dimension we can easily find the approximation for the eigenvectors. For each data point we will do a 1D interpolation in the eigenfunction to find the eigenvector value. Even though this operation is very quick we will perform the interpolation only for the k smallest eigenvalues.

Algorithm summary

To summarize all the complicated steps of the algorithms we will see all the operations we done on the images database, and at the end display the complexity of the algorithm.

1. For each image from input data base create Gist descriptor

2. For each Gist descriptor vector rotate the data to maximize separabillity (using PCA)

3. For each of the input dimensions
   - Construct 1D histogram
   - Solve numerically for eigenfunction and values

4. Order eigenfunctions from all dimensions by increasing eigenvalue and take the first k.

5. Interpolate data into k eigenfunctions $\rightarrow$ yields approximate eigenvectors of Laplacian

6. Solve $k \times k$ system to give Classification function.

After we summarize all the steps of the algorithm we can see that his complexity is linear in number of images.

Alternative Algorithm - KNN

In order to test our results after the implementation we have chosen alternative algorithm that works in different way to classify images. K-nearest neighbor (KNN) [3] is very simple algorithm that can classify objects based on closest training examples in the tested dataset. In other words for each unlabeled data point in our space we will test the k nearest labeled neighbors to that point and we will determine the type of the data according to the most common type amongst the k neighbors.
It is easy to see that KNN is much simpler than SSL, and he can work perfectly in some specific datasets but as we will see in the test section, without considering the density of the values we can sometimes receive wrong results.

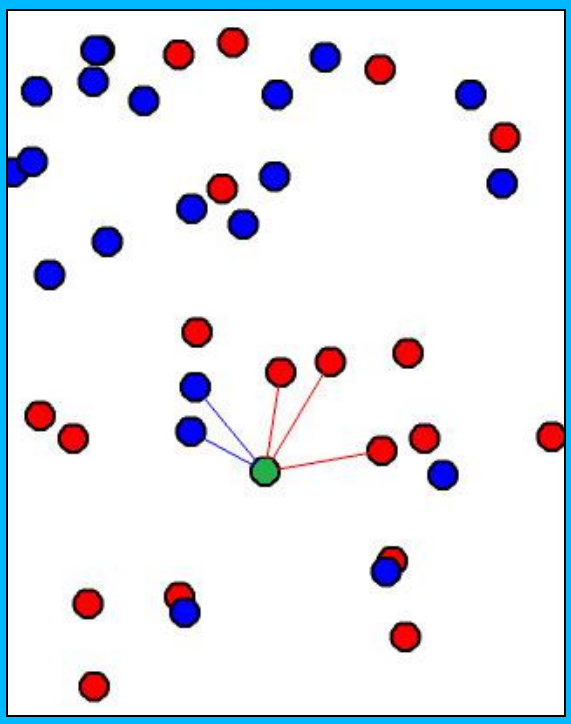

Figure 4: Example of classifying one unlabeled point (green) in space of data set of two types (red and blue) with k=5. In this example test point will be classified as 'red'.

Testing on Data and Results

After we introduced the main Algorithm (SSL) and the simple algorithm that will be used for comparison of the results (KNN), we will run the tests on two types of data. First we want to test our algorithm on simple data (toy data) that have nothing to do with images, and only after we will see that we can receive expected result we will continue testing the SSL algorithm on real images database.

Toy Data

As we mentioned earlier, after the first processing of each image (feature extraction and dimension reducing) we receive a vector of 32 dimensions. Although this vector significantly smaller than the original image we can't make testing with it, because we would like to see some graphical results. This is why the first tests will be done with imaginary data of only two dimensions.

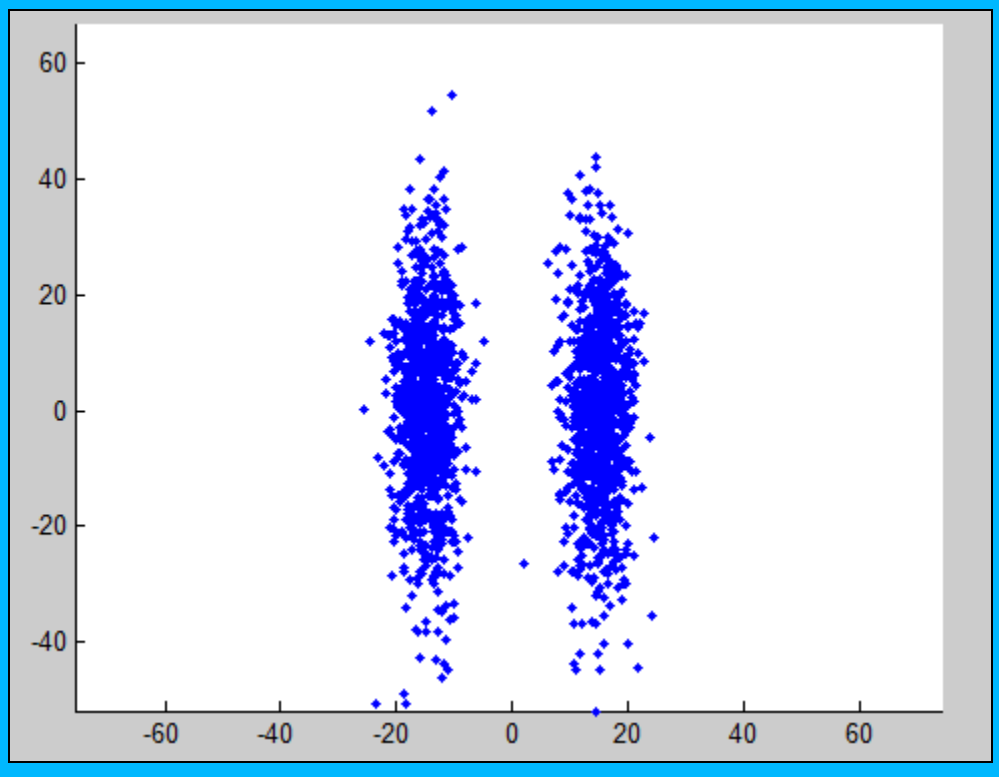

Figure 5: Data set of all the points of the toy data.

In figure 5 we can see the toy data, all the unlabeled data and the two labeled points. We can see that all the data combined of two sets of points with the same density and with only one labeled point in each set. As expected from SSL algorithm we want to see classification function that are smooth with respect to the density of the data, in other words we except the two sets to be classified as two different sets of points. And as expected from KNN algorithm we want to see classification function only with respect of the position of the data without any reference of the organization of the points in space.

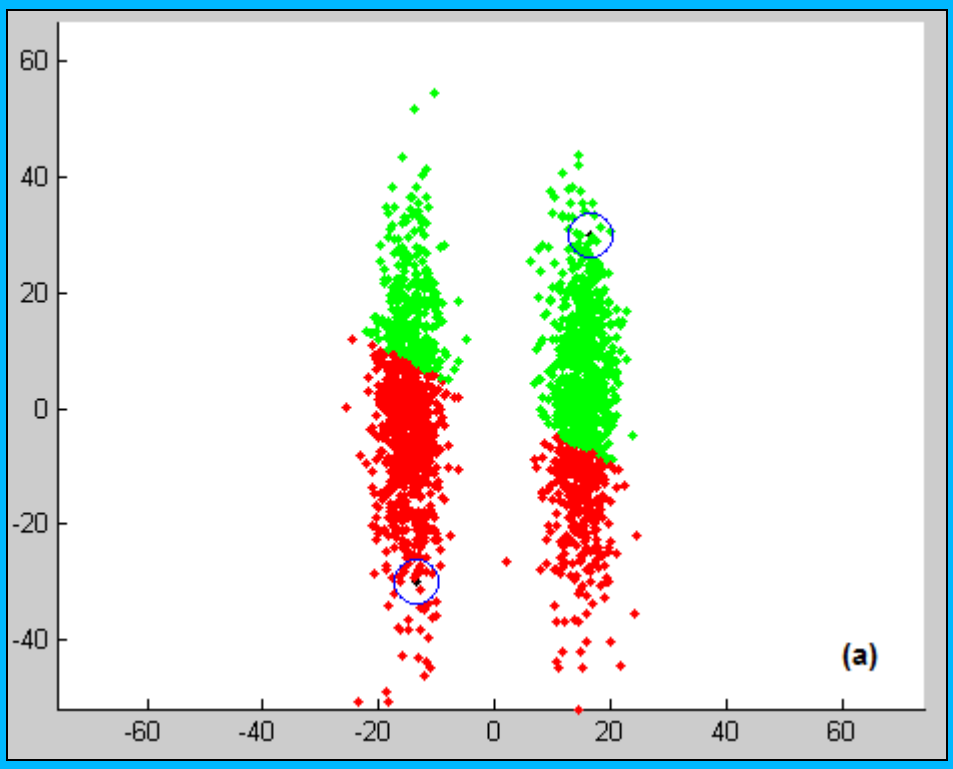

Figure 6a: Classification result of the KNN algorithm, as we can see all the points classifies according to the position of the two labeled points.

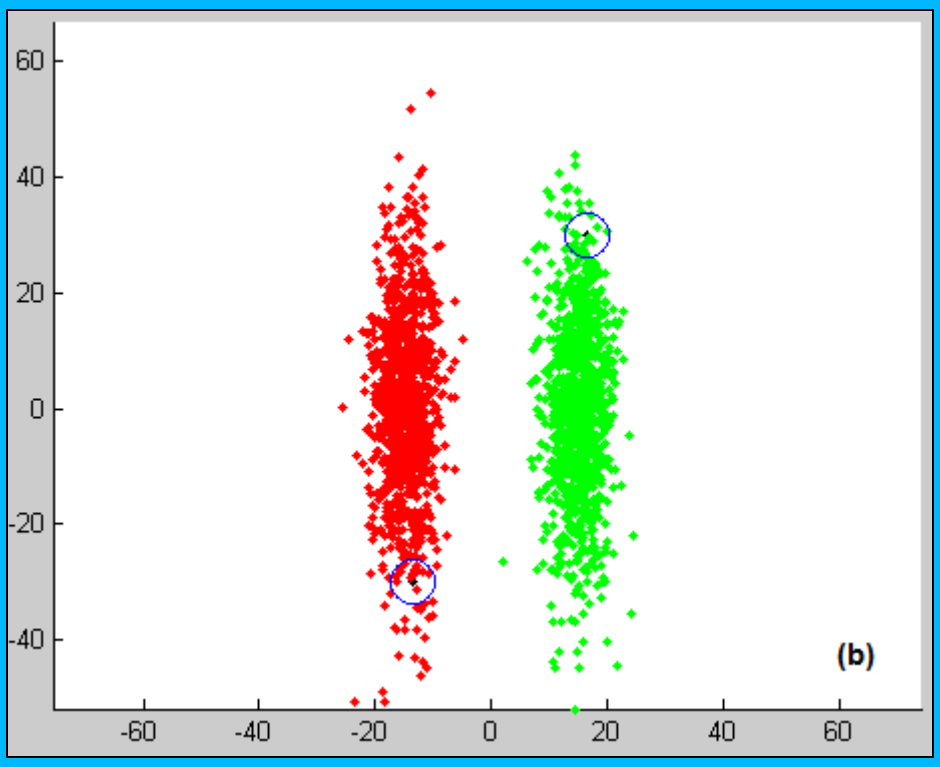

Figure 6b: Classification result of the SSL algorithm, as we can see all the points classifies according to density of the sets.

As we can see from figure 6, the results of both of the algorithms are as expected.
In figure 6a we can see that KNN classified points from both of the sets of the same type.
In figure 6b we can see that SSL classified all the point not by position but only by the density of the data.

Real Data

After we have received expected result on the toy data we can continue testing the algorithm on real images. For this propose we have chosen online image database called CIFAR-10 [5]. This database contains thousands of labeled images but all of them categorized only to ten categories. This database will allow us easy testing because we have limited set of categories and more importantly they all were labeled manually by human. Because the SSL algorithm returns binary classification function each execution we can find images from one of the categories.

Correctness of the Classifier

In our analysis each sample is either positive or negative. The binary classifier will receive some labeled data, which part of it marked as positive and part negative.
And unlabeled data which we want to classify. Our binary classifier will focus on a particular class of samples, and try to classify the unlabeled data. The unlabeled samples which would be detected as of that class will be marked as positive, and the other samples will be marked as negative.

To illustrate the correctness level of the classifier we will use the ROC graph.
The horizontal axis is the percentage of samples what were detected falsely as positive out of the whole unlabeled database (FPR).
The vertical axis is the percentage of unlabeled samples out of the whole positive samples that were detected as positive and they actually were positive (TPR).

When we receive the result of the Eigen-Functions algorithm which is the classification function, we analyze it using different values of thresholds. For each threshold value we calculate the false positive rate and the true positive rate.

The unlabeled test data which we classify contains 10,000 samples from 10 different classes.

Correctness of the Classifier – Single Class

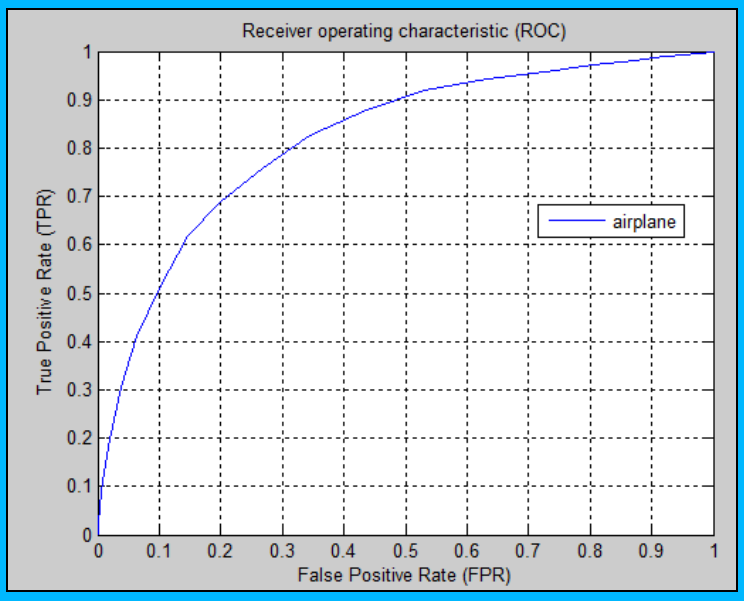

Figure 7a: classification results of class “airplane”. Labeled samples contained 50 positives and 100 negatives.

In order to increase correctness of results, we can use more positive and negative samples

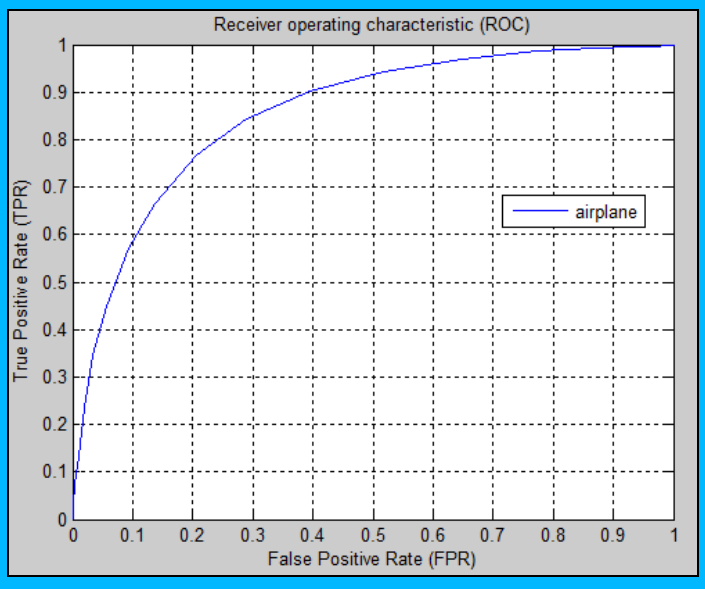

Figure 7b: classification results of class “airplane”. Labeled samples contained 700 positive and 2500 negative

On figure 7a we used smaller amount of positive and negative samples than on figure 7b, which resulted in weaker correctness.
On figure 7a for FPR=0.1 we got TPR=0.51 while in figure 7b we got 0.6

Correctness of the Classifier – All Classes Classification

The following diagram shows the results of classifying the whole test data by its label classes.
For each class we took 100 positive samples and 900 negatives.

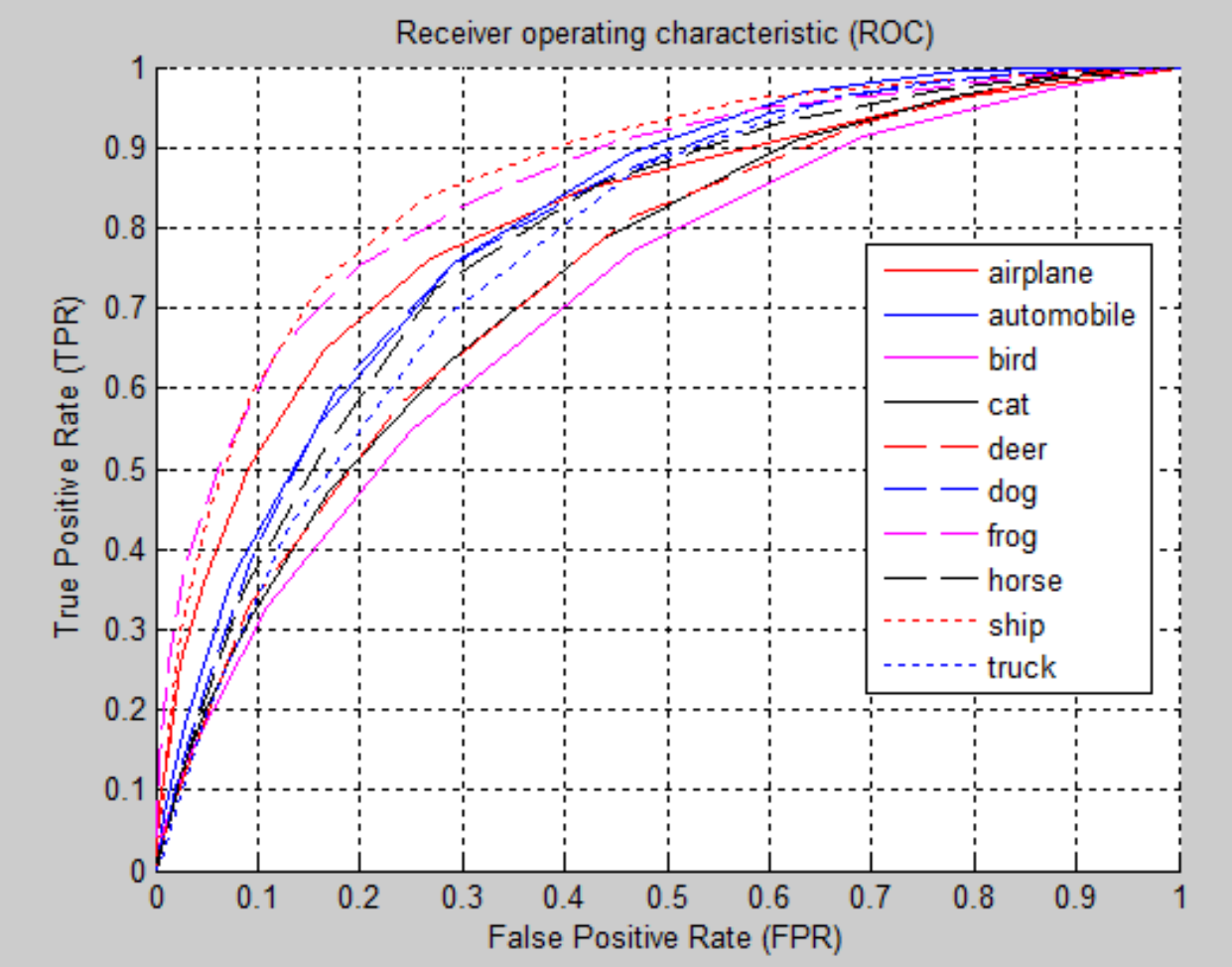

Figure 8: Summary of all the classification results for all the classes.

From the diagram above, it is clear that some of the classes have low correctness like the "cat" and the "bird" that have about 0.5 TPR for 0.2 FPR.
While other classes like "airplane" and "frog" has 0.75 TPR for 0.2 FPR.

Comparing SSL with KNN

We will compare the correctness of KNN and SSL through two diagrams. The first diagram will allow us to see the general tendency of correctness.
The second diagram will be a zoom in of the first one, and allow us to see in more details and concrete values and compare them in both algorithms.

For comparison we took 100 positive and 900 negative samples for both algorithms.
For KNN algorithm K=5 was used.

On both diagrams the colored lines represent the results for SSL algorithm.
The '+' mark represent the results for the KNN algorithm.

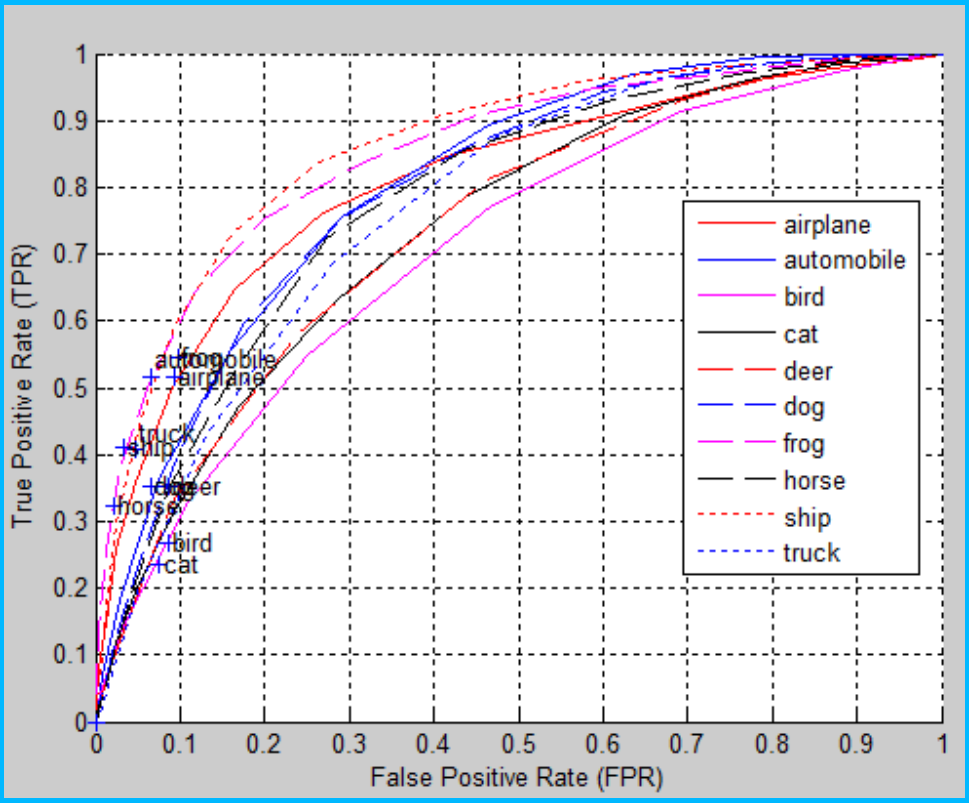

Figure 8a: general overview of KNN and SSL correctness.

From the above diagram we can lean that the general tendency of SSL's TPR is to grow rapidly as we allow the higher FPR value. While only few classes achieve 0.6 TPR for 0.1 FPR, almost all classes has more than 0.7 TPR for 0. 3 FPR.

Unlike the SSL, the KNN has static correctness. We cannot achieve greater TPR in exchange for allowing higher FPR. For extremely low positive rate we achieve TPR in the ranges 0.2 – 0.5.

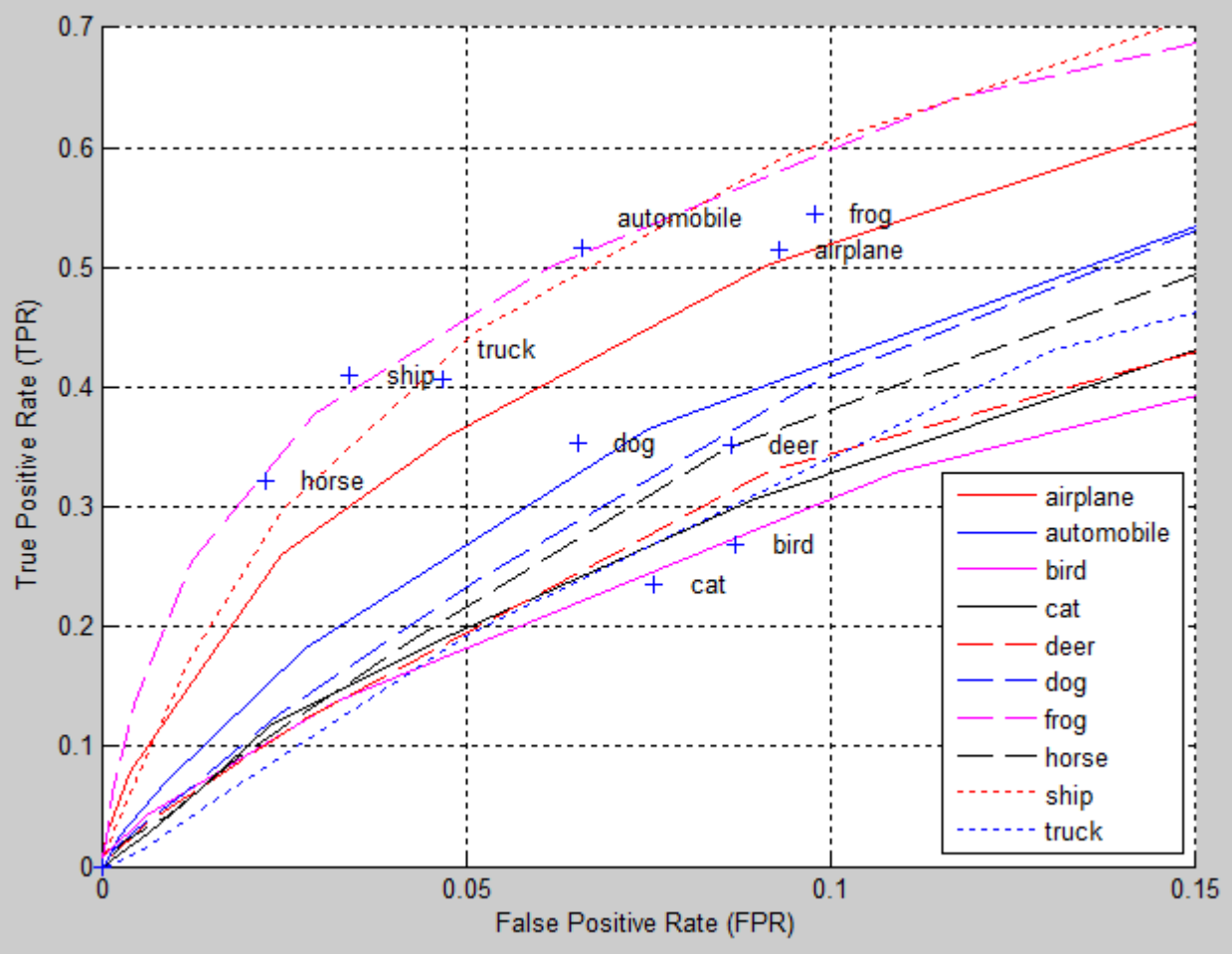

Figure 8b: zoom in of KNN and SSL correctness.

From the diagram above we learn that the KNN and the SSL has approximately the same TPRs for the same FPRs.
Some classes like “bird” and “airplane” has similar TPR for the same levels of FPR.
For “ship”, the KNN has higher TPR than the SSL.
For “frog” and “cat”, the SSL has the higher TPR.

Another Example:

Example of another SSL and KNN run. This time we took very small amounts of labeled data, Only10 positive and 90 negative samples.

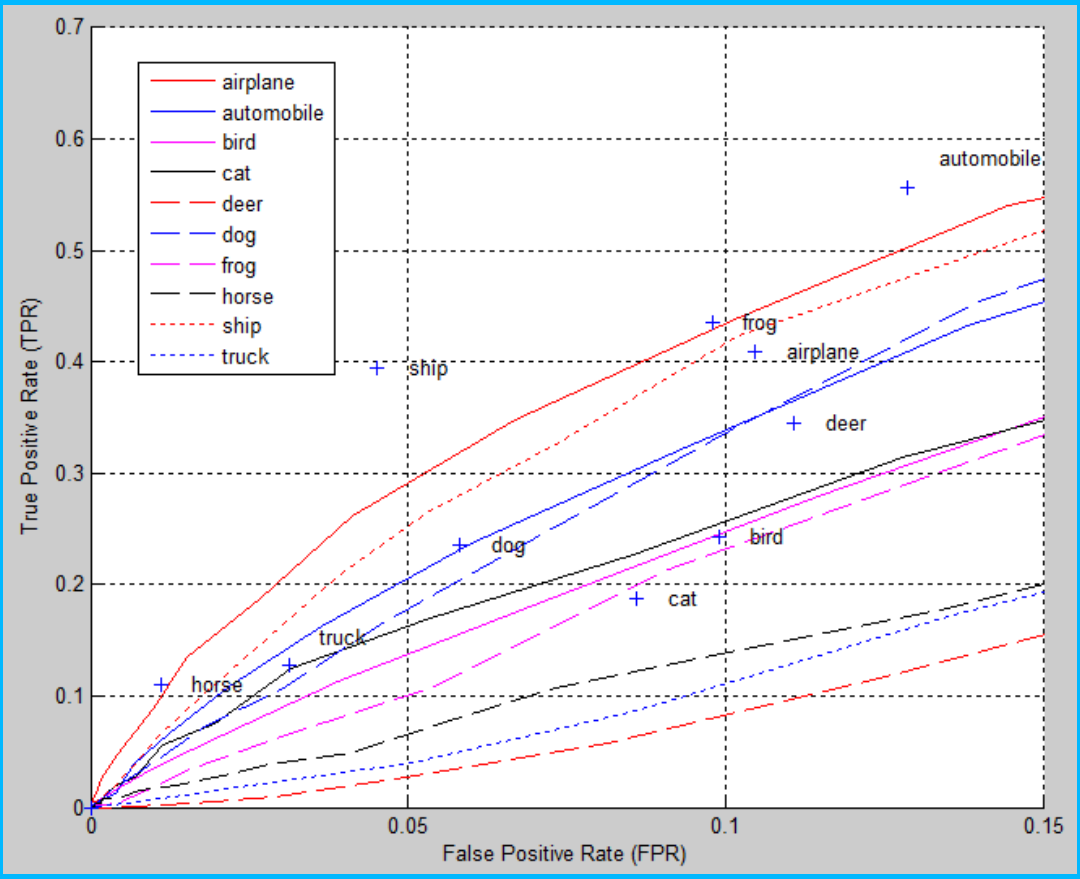

Figure 9: another compare using 10 positive and 90 negative samples. K=5 was used for KNN.

For both algorithms the TPR deteriorated dramatically for most of the classes.

Conclusions

1) Classification algorithm that is based on KNN calculation is not complete.
   It doesn't take into consideration the density and the smoothness of the data, thus it's correctness is damaged.

2) Effectiveness of SSL (based on Eigen-Functions) could be increased, by taking bigger amount of positive and negative samples.

3) Eigen-Function algorithm can retrieve higher positive true percentage if the systems demand allows higher false positive percentage. It is flexible unlike KNN.
   (It can retrieve 90% true positives of "ship" if 0.35 false positive allowed)

4) Eigen-functions algorithm has low complexity than other similar algorithms that it could enhance classification on large databases.

5) Every class of data is different thus it requires different tuning for setting the optimum threshold and choosing the most suitable amount of positive and negative samples. It also depends on the limits of the available computation power. The dilemma could also be between taking 5000 of positive samples and 0 negatives, or taking 2500 positives and 2500 negatives.

6) The SSL algorithm (based on Eigen-functions) has low TPR values, when the allowed FPR is low.

References

1. Rob Fergus, Yair Weiss and Antonio Torralba,
   Semi-Supervised Learning in Gigantic Image Collections

2. Rob Fergus, Yair Weiss and Antonio Torralba,
   Small Codes and Large Image Databases for recognition

3. k-nearest neighbor algorithm,
   http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm

4. Principal component analysis,
   http://en.wikipedia.org/wiki/Principal_component_analysis

5. CIFAR-10 database,
   http://www.cs.toronto.edu/~kriz/cifar.html

6. A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. IJCV, 42:145–175, 2001