Visual Recognition
Group Project 1

Seelam Lalitha(IMT2017027), Ananya Appan(IMT2017004),
Swasti Shreya Mishra(IMT2017043)

March 2020

1 VLAD (vs BOVW) on CIFAR-10

1.1 Introduction

Image Classification is an important area of computer vision that can classify an image according to its visual content. It has application in many fields. BOVW, VLAD are some of the basic algorithms that are used in image classification.

1.2 BOVW

Bag of Visual Words is commonly used in image classification. This is similar to NLP's BOW. The main idea of BOVW is to represent an image as features. We use SIFT. SURF or any feature extraction algorithm to extract key points and descriptors. Descriptors are the representation of features in the image.

Next, we make clusters from descriptors using k-means. We can consider the centroid of each cluster as a visual word. The main idea of BOVW is to convert the list of descriptors into a histogram of visual words. We compute euclidean distance and assign the descriptor to its closest visual word(centroid). We then train and test our model using the histogram using KNN.

1.2.1 Accuracies on BOVW

SURF without hessian threshold, with k = 50 clustes of kmeans gives an accuracy of 0.15080527086383602, with KNN classification.

1.3 VLAD

Similar to BOVW, Vector of Locally Aggregated Descriptors or VLAD is also used in image classification. BOVW is simply counting number of descriptors which belong to a particular visual word, plotting a histogram out of it and thus representing an image in the form of a vector. VLAD is an extension of this concept.

1. First, we extract features using feature extraction algorithms like SIFT, SURF, ORB etc.

2. Similar to BOVW, we make clusters from descriptors using k-means. In our case, we used minibatch k means as it faster compared to k-means

3. In VLAD, we match each descriptors to its corresponding cluster using euclidean distance. For each cluster we store sum of differences of all descriptors that correspond to the respective cluster

4. Assume C = {c_1, c_2, ... c_k} be the clusters obtained from k-means or minibatch k-means. The mathematical representation of storing sum of differences of all the descriptors is as given below

   $v_{ij} = \sum_{x|x=NN(c_i)}(x_j\text{-}c_{ij})$ where,

   v is a 2D array containing VLAD descriptors for an image
   vij is the element in i^{th} row at j^{th} position
   x_j is the j^{th} dimension of descriptor x that belongs to the image
   c_{ij} is the j^{th} dimension of centroid c_i where c_i is the cluster that the descriptor x belongs to The vector v is subsequently normalized

The primary advantage of VLAD over BOVW is that we add more discriminative property in our feature vector by adding the difference of each descriptor from the mean in its voronoi cell. This first order statistic adds more information in our feature vector and hence gives us better discrimination for our classifier. This also points us to other improvements we can by adding higher order statistics to our feature encoding as well as looking at soft assignment,i,e. assigning each descriptor multiple centroids weighed by their distance from the descriptor.

1.3.1 Accuracies on VLAD

- **SURF with a hessian threshold of 400, with k = 50 clusters on kmeans gives the following accuracies with KNN classification:**

Model	**Accuracy**
VLAD	0.2382511105915361
PCA + VLAD	0.2271
Resized image + VLAD	0.24269347673602992
Gray Scaled image + VLAD	0.25344867898059387
Gray Scaled + resized image + VLAD	0.24713584288052373

- **SURF with a hessian threshold of 400 and with k = 100 clusters on kmeans gives the following accuracies with KNN classification**

Model	**Accuracy**
VLAD	0.2447977554360533
Resized image + VLAD	0.23731587561374795
Gray Scaled image + VLAD	0.23731587561374795
Gray Scaled + resized image + VLAD	0.23076923076923078

- **SURF with k = 50 without hessian threshold gives the following accuracies with KNN classification**

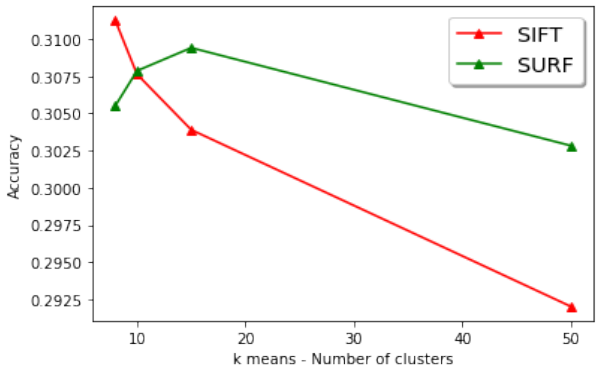
Model	Accuracy
Resized image + VLAD	0.24814888933360016
Gray Scaled + resized image + VLAD	0.2448714099869909

- **SURF with k = 10 clusters of kmeans and pca gives 0.2687 accuracy with KNN classification**
- **SIFT with k = 50 and 100 clusters on kmeans gave 0.2382511105915361 with KNN classification**
- **ORB with k = 10 and pca : 0.2284370622373424 with KNN classification**
- **We have also tried the classification using SVM instead of KNN**

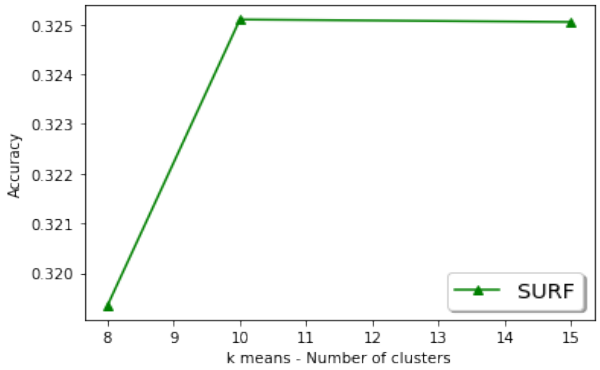
Model	Accuracy
VLAD	0.4181
VLAD + PCA	0.3453
VLAD + RBF Kernel with PCA	0.2322

1.3.2 Experiments and Observations

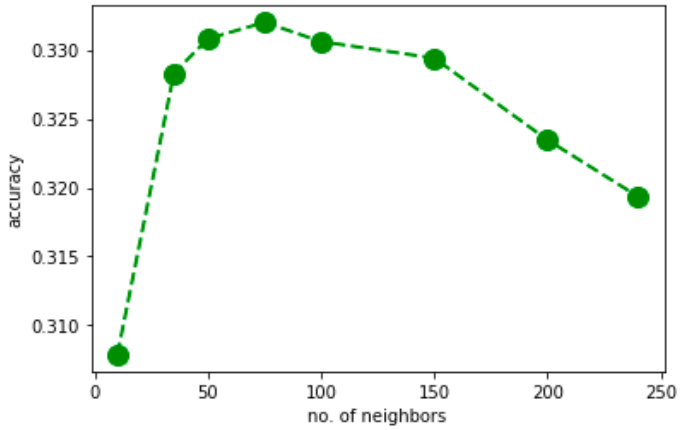
The following are the accuracies we got after trying out different values for number of clusters in k means as well as trying out different descriptors with VLAD. The graphs are plotted based on accuracies produced by the code which doesn't import data from keras and uses the images after extracting it from the ciphar-10 pickle file.



We can infer from the above graph that, when we increase the number of clusters while using the SIFT descriptor, the cross validation accuracy reduces. This might be due to overfitting by the model. Whereas, in SURF it gradually increases and then decreases. This might be due to the fact that SURF descriptors are an approximation of SIFT descriptors. So, increasing the number of clusters increases the complexity of the model and the model in turn fits the training data better. But, eventually, even that results in overfitting of the training data and the cross validation score reduces.



The above graph is plotted for different values of number of clusters used in the kmeans algorithm. The number of nearest neighbour is set to 240. We can see that, there is an overall increase in the cross validation score compared to the previous graph as in the previous graph the number of nearest neighbour was set to 10. We can also infer that the peak occurs somewhere around number of clusters = 10.



The above graph is plotted for different values of number of neighbors used in the kmeans algorithm. The number of clusters was fixed at a value of 10. As we can see, the accuracy increases along with the number of clusters until 75, after which it again starts to decrease. The reason behind this could be that if we consider too few neighbors, we may be overfitting. On number of neighbors until a point helps as it brings in more variance. However considering too many neighbors will bring in too much variance.

We also tried performing image classification using SVM instead of KNN. We obtained the following accuracies.

1. SVM without PCA : 0.4181
2. SVM with PCA : 0.3453

3. SVM using RBF kernel with PCA : 0.2322

We can infer the following from the above observations.

1. Because SVM gave us better accuracies than KNN, the dataset is somewhat linearly separable. The fact that SVM using an RBF kernel drastically reduced the accuracy also indicates the same.

2. Since the accuracy reduced on using PCA with SVM, it shows that SVM performs better on higher dimensional datasets.

The above are just speculations on our part. Since SVM inherently chooses "important" points to support it (support vectors), it is already trying to "prune" the dataset. Perhaps on further trying to reduce dimensionality using PCA, we have lost out on more information.

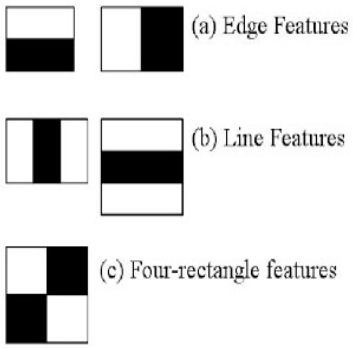
2 Face Recognition

2.1 Pre-Processing

2.1.1 Face and Eye Detection using Haar cascade

Introduction : Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones. Here, we use Haar cascade classifiers for face and eye detection.

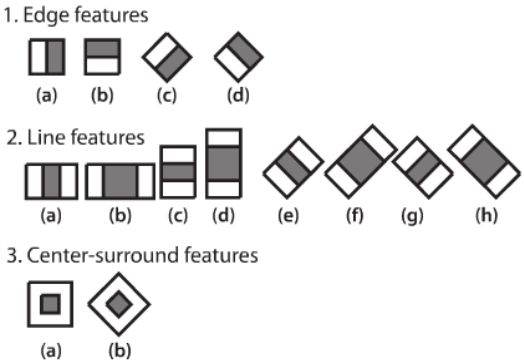
Haar Cascade Classifiers : For face detection, we will work with a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. We use the following to extract features.



Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle. Boosting (AdaBoost) is used to reduce computational complexity.

In an image, most of the image is non-face region. For this they introduced the concept of Cascade of Classifiers.The image is broken down into windows. The features are grouped into different stages of classifiers and applied one-by-one on each window. If a window fails the first stage,it is discarded. The window which passes all stages is a face region.

OpenCV's Haar Cascade Classifiers : OpenCV's algorithm is currently using the following Haar-like features which are the input to the basic classifiers



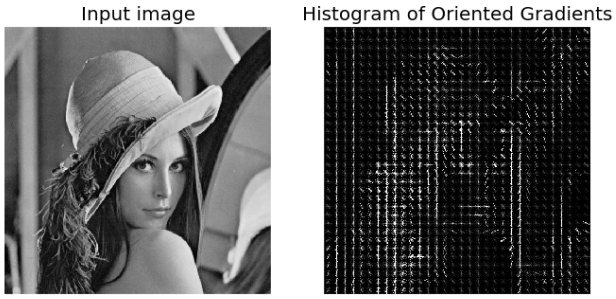
The CascadeClassifier function is used on grayscale images with the following parameters.

- image : Contains image on which detection is done.
- scaleFactor : Determines by how much the image size is reduced at each scale.
- minNeighbors : How many neighbors each candidate rectangle should have to retain it.
- minSize : Minimum possible object size.
- maxSize : Maximum possible object size.

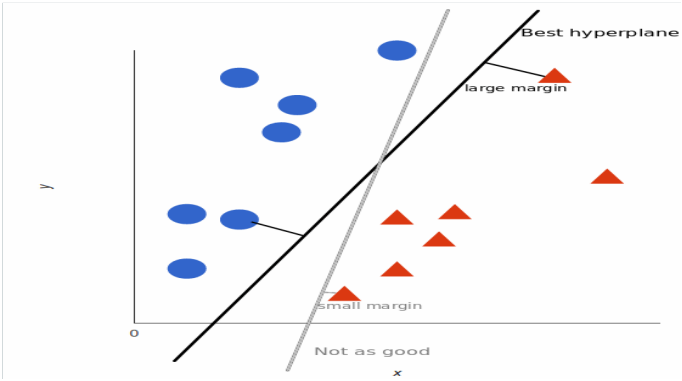
2.1.2 Face Detection using Dlib

Introduction : The first step towards extracting features of the face is creating a bounding box, or a region of interest (ROI), around the face. This was done using Dlib's front face detector. This works by using Histogram of Gradients to extract features, and an SVM (Support Vector Machine) for subsequent face detection.

Histogram of Oriented Gradients : In the HOG feature descriptor, the histograms of directions of gradients are used as features. These gradients are useful as they are larger around the corners and edges and are thus, helpful in shape and region detection.



Support Vector Machine (SVM) : A support vector machine takes the above features as points and outputs the hyperplane that best separates them. This line is the decision boundary, which maximizes the distance from the nearest point of both regions. The points to one side of the line will be classified as features of a face, while the other side will not.



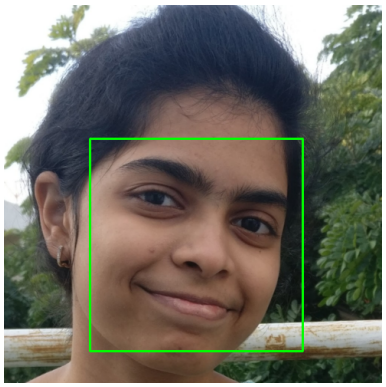
Dlib's Frontal Face Detector : This model is built out of 5 HOG filters – front looking, left looking, right looking, front looking but rotated left, and a front looking but rotated right. The dataset used for training, consists of 2825 images which are obtained from LFW dataset and manually annotated. The Pros and Cons are as follows.

Pros :

- Fastest method on CPU
- Works very well for frontal and slightly non-frontal faces
- Works under small occlusion

Cons :

- The major drawback is that it does not detect small faces as it is trained for minimum face size of 8080.
- The bounding box often excludes part of forehead and part of chin at times.
- Does not work for side face and extreme non-frontal faces



2.1.3 Facial Landmark Detection

Introduction : This was done using Dlib's Facial Landmark Detector. This is used to detect important facial structures such as the mouth, left and right eyebrows and eyes, the nose and jawline. This has been implemented using an Ensemble of Regression trees.

Regression Tree : A Regression trees are a variant of decision trees, designed to approximate real-valued functions, instead of being used for classification methods. It is built through a process called binary recursive partitioning, which splits the data into partitions iteratively. The split is done to minimize the sum of squared deviations in both the partitions. In order to prevent overfitting, pruning is done.

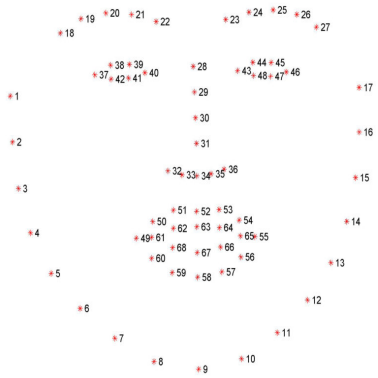
Ensemble Methods : Ensemble Methods used with regression trees include bagging, boosting and random trees.

Dlib's Facial Landmark Detector : The pre-trained facial landmark detector inside the dlib library is used to estimate the location of 68 (x, y)-coordinates that map to facial structures on the face.

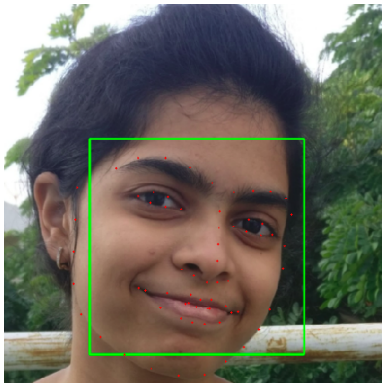
This method is trained using the following :

- A training set consists of manually labeled facial landmarks on an image, specifying (x, y)-coordinates of regions surrounding each facial structure.
- Priors, the probability on distance between pairs of input pixels.

Given this training data, an ensemble of regression trees are trained to estimate the facial landmark positions directly from the pixel intensities themselves , without any feature extraction.



Applying this on our image, we get the following landmark positions (Drawn as small red circles) .



2.1.4 Image Scaling

Using the landmark positions obtained above, we are able to detect the eyes and measure the distance between them. Thus, to ensure that the distance between them in the scaled image is 128px, we multiply by 128 and divide by the distance obtained to get the scale.

2.1.5 Image Rotation

We want the image to be aligned along the horizontal, in case it's tilted. This is done by the following.

dY = rightEyeCenter[1] - leftEyeCenter[1]
dX = rightEyeCenter[0] - leftEyeCenter[0]
angle = np.degrees(np.arctan2(dY, dX)) - 180

This is because if the line joining the eyes is aligned along the horizontal, the face will also be aligned.

On performing scaling and rotation , we get the following image.

2.1.6 Outlining the Face

An ellipse is drawn around the face mainly using the landmarks positions of the jawline. This is done by drawing an ellipse as follows.

- The centre of the ellipse is taken as the midpoint of the left most and right most points in the jawline.
- The major axis is taken as half the distance between the left most point and the right most point.
- The minor axis is taken as the distance between the centre of the ellipse and the bottom most point in the jawline.

The final outlined image is shown below.

2.2 Face Recognition

Introduction: Face recognition is an easy task for humans. Even three day old babies are able to distinguish between known faces. So how could it be difficult for a human? Face recognition based on the geometric features of a face is probably the most intuitive approach to face recognition. There are several approaches for recognizing a face. The algorithm can use statistics, try to find a pattern which represents a specific person or use a convolutional

neural network. In this project we mainly explore **Eigenfaces (PCA)** and compare it with **Fisherfaces (LDA)** and **Local Binary Patterns Histogram (LBPH)**.

2.2.1 Eigenfaces

Eigenfaces is a method for performing facial recognition based on a statistical approach. Here, we extract the principal components which affect the variation in the images the most. It is based on Principal Component Analysis (PCA) for reducing the number of dimensions while preserving the most important information.
The steps followed for training are as follows:

1. We take pre-processed grayscale images for training. (In this step we use the images generated from the previous pre-processing step where the size of each image is 256 x 256 pixels.) These grayscale images are flattened to get a 1-D vector (65536 x 1). We then stack all the images together to form a matrix.

2. Now, in order to reduce the dimensionality we need to project all training samples into the PCA subspace.For this follow the following steps:

   - We take the average face vector i.e. sum(all image vectors)/$number$(images)
   - We subtract the average face vector from every image from the image stack matrix.
   - Calculate the covariance matrix for the image stack matrix i.e. AA^T. But the size of this matrix is very huge and computing eigen vectors and eigen values for it will be very computationally expensive. So, we calculate A^TA instead.
   - Find **eigen vectors** and **eigen values** for the above covariance matrix.
   - We then pre-multiply the eigen vectors of A^TA with A to get eigen vectors of AA^T.

3. We choose k best eigen vectors such that $k << number\ of\ images$. We find the weights for each image and store it. (Here we transform the image matrix by projecting all the training samples into the PCA subspace.)

The steps followed for testing are as follows:

1. We first normalize the test image i.e. $test\ image\ vector - average\ face\ vector$.

2. We find the weights for the image (transform it by projecting it into the PCA subspace).

3. Find the nearest neighbor between the projected training images and the projected test image. This is done by calculating the error between the weights of the test image and weights of all images in the training set. We have used *Euclidean distance* here which is also known as the $L2$ $norm$.

The ℓ^2-norm (also written "l^2-norm") $|\mathbf{x}|$ is a vector norm defined for a complex vector

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

by

$$|\mathbf{x}| = \sqrt{\sum_{k=1}^{n} |x_k|^2},$$

4. Choose the image of training set which has minimum error.

Results:
With k = 100:
Rank 1 accuracy with scatter matrix eigen vectors: 0.5339805825242718
Rank 1 accuracy with covariance matrix eigen vectors: 0.5339805825242718

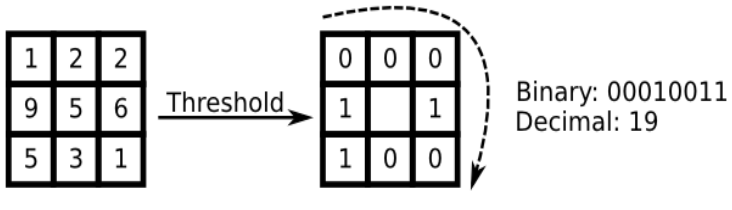
Top 3 accuracy with scatter matrix eigen vectors: 0.6310679611650486
Top 3 accuracy with covariance matrix eigen vectors: 0.6213592233009708

Top 10 accuracy with scatter matrix eigen vectors: 0.7572815533980582
Top 10 accuracy with covariance matrix eigen vectors: 0.7572815533980582

2.2.2 Comparison of PCA with LDA and LBPH

Fisherfaces
The Linear Discriminant Analysis performs a class-specific dimensionality reduction. In order to find the combination of features that separates best between classes the LDA maximizes the ratio of between-classes to within-classes scatter, instead of maximizing the overall scatter. The main idea being: same classes should cluster tightly together, while different classes are as far away as possible from each other in the lower-dimensional representation.

Local Binary Patterns Histograms
The basic idea of Local Binary Patterns is to summarize the local structure in an image by comparing each pixel with its neighborhood. Take a pixel as center and threshold its neighbors against. If the intensity of the center pixel is greater-equal its neighbor, then denote it with 1 and 0 if not. You'll end up with a binary number for each pixel, just like 11001111. So with 8 surrounding pixels you'll end up with 2^8 possible combinations, called Local Binary Patterns or sometimes referred to as LBP codes. The first LBP operator described in literature actually used a fixed 3 x 3 neighborhood just like this:



Results:
With PCA Face Recognizer:
Rank 1 accuracy: 0.6699029126213593
Top 3 accuracy: 0.7766990291262136
Top 10 accuracy: 0.883495145631068

With LDA Face Recognizer:
Rank 1 accuracy: 0.6699029126213593
Top 3 accuracy: 0.7087378640776699
Top 10 accuracy: 0.7669902912621359

With LBPH Face Recognizer:
Rank 1 accuracy: 0.7864077669902912
Top 3 accuracy: 0.8543689320388349
Top 10 accuracy: 0.941747572815534

Observations: PCA performs better in case where number of samples per class is less. Whereas LDA works better with large dataset having multiple classes. Class separability is an important factor while reducing dimensionality. LBPH is better than PCA and LDA for small training data set.

2.3 References

1. OpenCV Face Recognition Tutorial: `https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html`
2. Face Recognition using eigenfaces technique (Deval Shah): `https://medium.com/@devalshah1619/face-recognition-using-eigenfaces-technique-f221d505d4f7`
3. OpenCV haar classifier tutorial: `https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html`
4. DLib's Facial Landmark Detection tutorial : `https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/`
5. Face alignment and scaling tutorial : `https://www.pyimagesearch.com/2017/05/22/face-alignment-with-opencv-and-python/`