A REPORT

ON

FACIAL EXPRESSION RECOGNITION USING HOG AND SVMs

BY

ADARSH	2014A7PS062G
JASTI BRAHMARSHI	2014A7PS055P

AT

CENTRAL ELECTRONICS ENGINEERING RESEARCH INSTITUTE, PILANI

A PRACTICE SCHOOL-1 STATION OF

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

JULY 2016

A REPORT

ON

FACIAL EXPRESSION RECOGNITION USING HOG AND SVMs

BY

ADARSH	**2014A7PS062G**	**COMPUTER SCIENCE**
JASTI BRAHMARSHI	**2014A7PS055P**	**COMPUTER SCIENCE**

PREPARED IN PARTIAL FULFILMENT OF PS-1 COURSE

AT

CENTRAL ELECTRONICS ENGINEERING RESEARCH INSTITUTE , PILANI

A PRACTICE SCHOOL-1 STATION OF

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE , PILANI

JULY 2016

ACKNOWLEDGEMENTS

We would like to thank Dr. Santanu Chaudhury (Director, CEERI) and PS Division for giving us the opportunity to work in a top notch research organization like CEERI. We are grateful to Dr. Anil K Saini for guiding us through the project and giving us insights into his own work. We would also like to thank Mr Sumeet Saurav who is our mentor for this project and has constantly been guiding us and helping us get past the roadblocks. We are also grateful to Mr. Pawan Sharma, our PS-I instructor whose constant advice and support throughout the programme has made the PS-1 experience much more fruitful.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI (RAJASTHAN)
Practice School Division

Station : CEERI, Pilani **Centre** : Pilani

Duration : May 23, 2016 to July 15, 2016 **Date of Submission** : July 13,2016

Title of the Project : FACIAL EXPRESSION RECOGNITION USING HOG AND SVMs

ID No./Name(s)/Discipline(s) of the student(s) :

Adarsh	2014A7PS062G	Computer Science
Jasti Brahmarshi	2014A7PS055P	Computer Science

Name(s) and designation(s) of the expert(s) :

Dr Anil K Saini , senior scientist in I.C DESIGN GROUP

Mr Sumeet S Saurav, SRF

Name(s) of the PS Faculty : Mr. Pawan Sharma

Key words: SVMs, Face recognition, Face alignment, LBP, HOG, clandmarks, FEAST, libsvm

Project Areas : Computer Vision ,Image Processing ,Video Processing ,Machine Learning

Abstract: Automatic facial expression recognition is a topic of growing interest mainly due to the rapid spread of assistive technology applications, as human–robot interaction, where a robust emotional awareness is a key point to best accomplish the assistive task. This project aimed to build a facial expression recognition system capable of classifying human expressions in real time using SVMs. Two descriptors for feature extraction were considered- Local Binary Patterns (LBP) and Histogram of Oriented Gradients (HOG). The widely used Radboud Faces Database was used to train the classifiers, and the results were compared and cross validated to get the best trained model for classification. HOG was found to yield better accuracies. Finally, feature selection was performed to reduce the processing time for each frame. An accuracy of 97.17% was achieved on the test set, and the SVM parameters selected yielded a 10-CV accuracy of 99.72%.

Signature(s) of Student(s) Signature of PS Faculty
Date: Date:

TABLE OF CONTENTS

INTRODUCTION

Facial expression is one of the most powerful and immediate means for humans to communicate their emotions, cognitive states, intensions, and opinions to each other. Recent advances in the Computer Vision algorithms equipped with Machine Learning(Pattern Classification) algorithms has opened up wide areas of research in the design of automatic Vision **systems** deploying a blend of these algorithms like Facial Expression recognition system, Object detection system etc. Facial expression recognition has been a widely studied problem, for it has applications from haptic perception for the blind to the ultimate aim of making robot-human interaction more human-like.

In this project, we plan to create a system that takes in video/image and, if it detects a face, classifies the expression into one of 7 classes (or expressions) - neutral, happy, sad, anger, disgust, surprise and fear. The system should be fast enough to process the video in real time, so that instant feedback can be provided to the robot/human receiver.

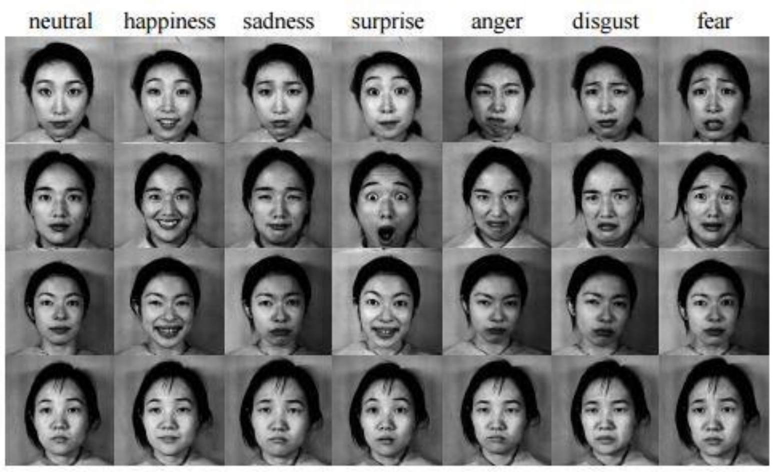


Figure 1. Facial expression database: Examples

The project involves several stages:

- **<u>Preprocessing:</u>** The raw image received cannot directly be used for feature extraction. We have to process it so that the feature extraction algorithms can work on it correctly. Normally, this involves scaling, conversion to grayscale, facial alignment, frontalization, adaboost etc. However, we are not going to use frontalization for this project, as it involves advanced mathematics and complex algorithms.
- **<u>Feature Extraction:</u>** We intend to perform feature extraction using Local Binary Patterns and HOG, as these kinds of texture descriptor has been found to be very effective for facial recognition tasks.
- **<u>Learning the parameters:</u>** Finally, SVMs are used to train the system on the data set obtained after feature extraction. This results in a hypothesis that should be able to, to a certain degree of accuracy, correctly predict the expression given any new image. SVMs were used over neural networks because they are faster and well suited for a real time detection.

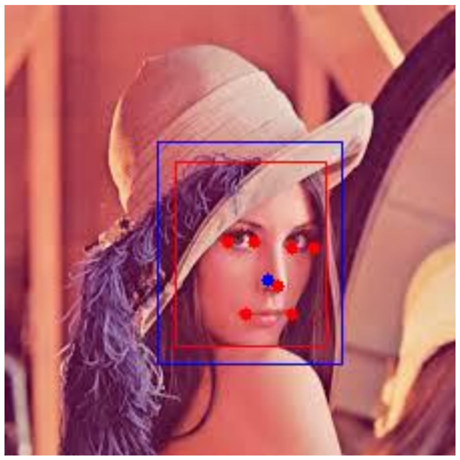
We tested our models on MATLAB, as it is easy to work with and offers higher levels of mathematical abstractions.

PREPROCESSING

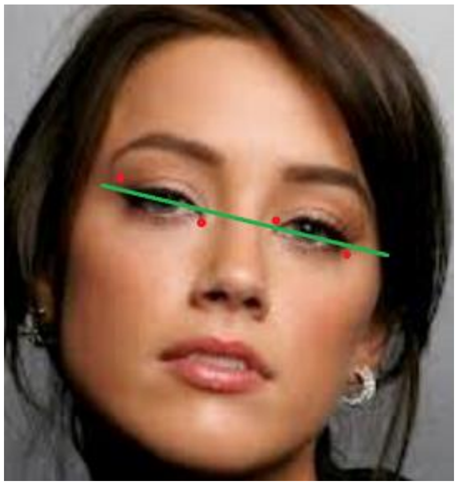
Facial Alignment Algorithms

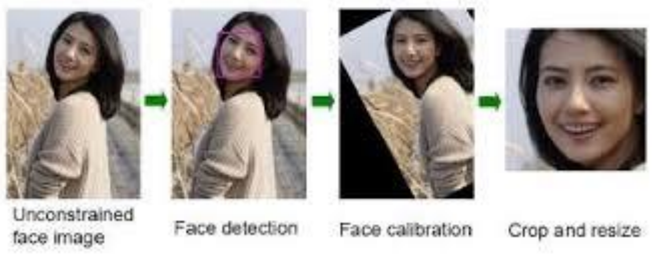
The first thing we have tackled is to align the image properly. If the face is tilted, prediction might not work as well on that image.

We used a library called clandmarks that detects particular landmarks on the face given a particular image. Here's an example of landmarks detected by clandmarks:



We use clandmarks to detect the landmarks at the corners of the eyes. Then, using these landmarks, we detect the inclination of the face by drawing a line through these points obtained. Finally, we rotate the image by an angle equal and opposite to the inclination of the line.





Working With Video Input

The goal of the project is to ultimately work on faces in a video, so we decided to test face alignment on video inputs- both files and the webcam.

MATLAB provides two different kinds of video file handlers- VideoReader and VideoFileReader. Unfortunately, neither of those worked on our Ubuntu 16.04 installation, giving errors that seemed to be open bugs in MATLAB.

So we decided to use mexOpenCV- an open source library package that interfaces OpenCV functions to MATLAB. Also the package contains C++ class that converts between MATLAB's native data type and OpenCV data types. The package is suitable for fast prototyping of OpenCV application in MATLAB, use of OpenCV as an external toolbox in MATLAB, and development of a custom MEX function.

To test video functionality, we used the alignment algorithm on each face in every frame of the video, and saved the cropped and aligned face images to a separate directory.

We also tested the library on webcam, where the system was able to process and align faces in video at 10fps without any noticeable lag.

Image database and its Preprocessing

We used the widely used Radboud Faces Database (RaFD) to train our classifiers. The RaFD is a high quality faces database, which contain pictures of eight emotional expressions. Accordingly to the Facial Action Coding System, each model was trained to show the following expressions: Anger, disgust, fear, happiness, sadness, surprise, contempt, and neutral. A total of 67 models (including Caucasian males and females, Caucasian children, both boys and girls, and Moroccan Dutch males) display the 8 emotional expressions. Each emotion was shown with three different gaze directions and all pictures were taken from five camera angles simultaneously. So, there were a total of 201 (67x3) images for each emotional expression.

Example of RaFD images

Four datasets were created- Aligned & RGB, aligned & grayscale, not_aligned & RGB and not_aligned & grayscale. We used the RGB datasets to extract HOG features, and the grayscale datasets for extracting LBP features. This is the procedure followed while processing the images:

- Detect face bounding box (bbox) in each image using an LBP based facial detector
- Create a new image (I2) by cropping the image to a size slightly larger (20 pixels on each side) than the bbox
- Convert I2 to grayscale and feed to clandmarks for landmark detection and alignment. This produces another RGB image I3, with faces aligned and size same as I2.
- Crop I3 to size of original bbox. Call the cropped image I4.
- Convert I4 to grayscale. Call it I4g(optional/only for LBP)
- Resize I4/I4g to a size of 128x128 pixels. This resized image is your final image to be used for feature extraction. Call it I5.

I5 was saved with the same filename and folder structure as original database.

Further, for the purposes of training classifiers, we divided each of those 4 datasets into two parts, Training set and Test set. Training set is used to train the SVM classifier, while the test set is used to test the model obtained to find out its accuracy. The first 151 images alphabetically were used as Training set for each expression, and the rest 50 images were used as Test set. The entire data was maintained without splitting too in a separate folder named '*both'*.

Thus, the dataset '*RaFD_noalign_rgb'* had three subfolders- *both*, *Training* and *Test*. *Training* contained the 8 expression folders with 151 images each, while *Test* had each expression with 50 images each.

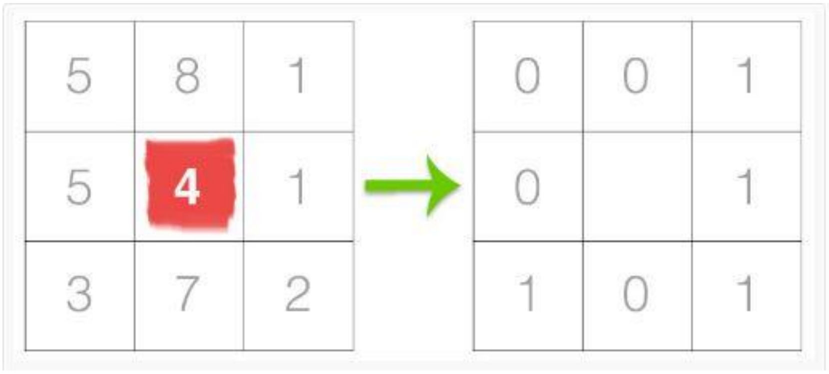
ALGORITHMS USED

Local Binary Patterns

Local binary patterns (LBP) is a type of visual descriptor used for classification in computer vision. LBP is the particular case of the Texture Spectrum model proposed in 1990. LBP was first described in 1994. It has since been found to be a powerful feature for texture classification; it has further been determined that when LBP is combined with the Histogram of oriented gradients (HOG) descriptor, it improves the detection performance considerably on some datasets.

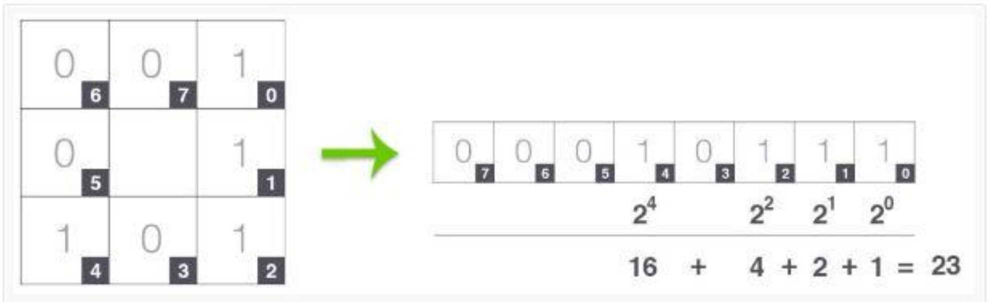
The first step in constructing the LBP texture descriptor is to convert the image to grayscale. For each pixel in the grayscale image, we select a neighborhood of size r surrounding the center pixel. A LBP value is then calculated for this center pixel and stored in the output 2D array with the same width and height as the input image.

For example, let's take a look at the original LBP descriptor which operates on a fixed 3 x 3 neighbourhood of pixels just like this:

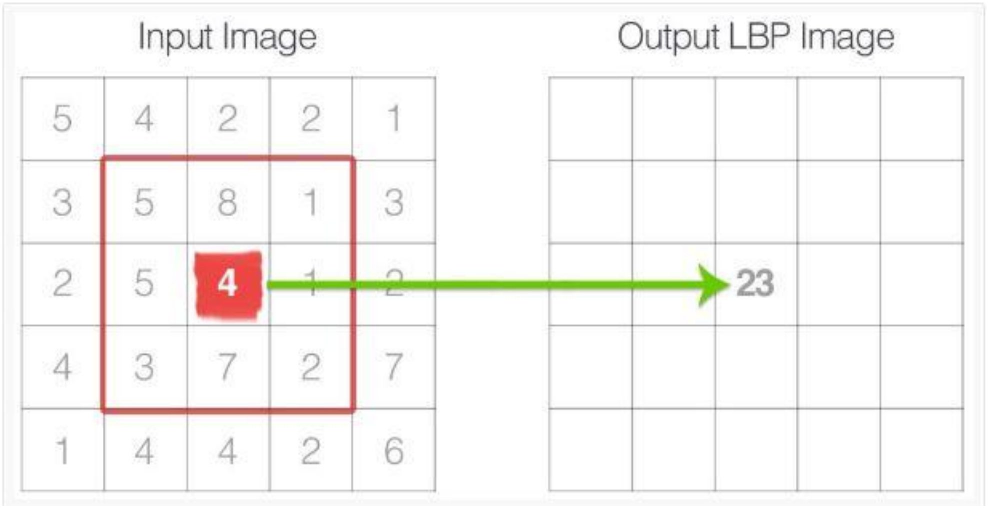


In the above figure we take the center pixel (highlighted in red) and threshold it against its neighborhood of 8 pixels. If the intensity of the center pixel is greater-than-or-equal to its neighbor, then we set the value to 1; otherwise, we set it to 0. With 8 surrounding pixels, we have a total of 2 ^ 8 = 256 possible combinations of LBP codes.

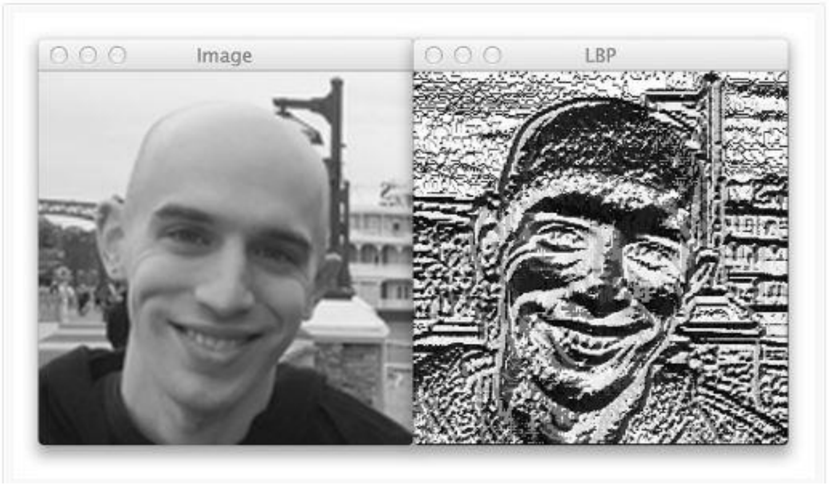
From there, we need to calculate the LBP value for the center pixel. We can start from any neighboring pixel and work our way clockwise or counter-clockwise, but our ordering must be kept consistent for all pixels in our image and all images in our dataset. Given a 3 x 3 neighborhood, we thus have 8 neighbours that we must perform a binary test on. The results of this binary test are stored in an 8-bit array, which we then convert to decimal, like this:



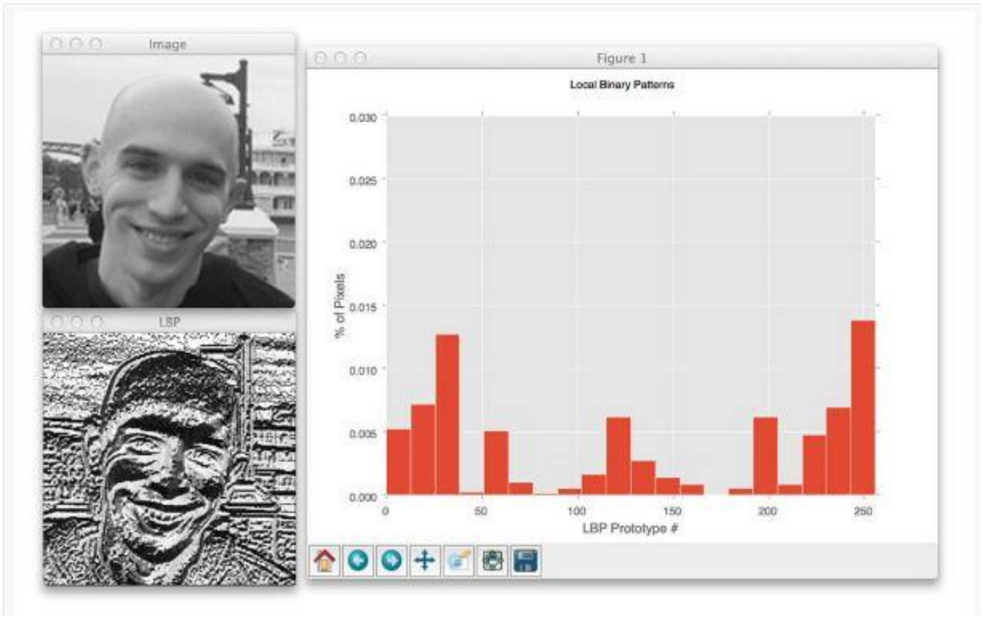
In this example we start at the top-right point and work our way clockwise accumulating the binary string as we go along. We can then convert this binary string to decimal, yielding a value of 23. This value is stored in the output LBP 2D array, which we can then visualize below:



This process of thresholding, accumulating binary strings, and storing the output decimal value in the LBP array is then repeated for each pixel in the input image. Here is an example of computing and visualizing a full LBP 2D array:



The last step is to compute a histogram over the output LBP array. Since a 3 x 3 neighbourhood has 2 ^ 8 = 256 possible patterns, our LBP 2D array thus has a minimum value of 0 and a maximum value of 255, allowing us to construct a 256-bin histogram of LBP codes as our final feature vector:



A primary benefit of this original LBP implementation is that we can capture extremely fine-grained details in the image.

A LBP is considered to be uniform if it has at most two 0-1 or 1-0 transitions. For example, the pattern 00001000 (2 transitions) and 10000000 (1 transition) are both considered to be uniform patterns since they contain at most two 0-1 and 1-0 transitions. The pattern 01010010) on the other hand is not considered a uniform pattern since it has six 0-1 or 1-0 transitions. We shall be using uniform LBP for our feature extraction.

For the purpose of our project, we used extractLBPFeatures function of MATLAB to extract the feature vectors from the dataset. We tweaked the cellSize to find the one that gave the best results with SVM classifiers.

Histogram of Oriented Gradients

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image.

The essential thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing.

The HOG descriptor has a few key advantages over other descriptors. Since it operates on local cells, it is invariant to geometric and photometric transformations, except for object orientation. Such changes would only appear in larger spatial regions. Moreover, as the original authors discovered, coarse spatial sampling, fine orientation sampling, and strong local photometric normalization permits the individual body movement of pedestrians to be ignored so long as they maintain a roughly upright position. The HOG descriptor is thus particularly suited for human detection in images.

Implementation of HOG Descriptor algorithm is as follows:

• Divide the image into small connected regions called cells, and for each cell compute a histogram of gradient directions or edge orientations for the pixels within the cell.

• Discretize each cell into angular bins according to the gradient orientation.

• Each cell's pixel contributes weighted gradient to its corresponding angular bin.

• Groups of adjacent cells are considered as spatial regions called blocks. The grouping of cells into a block is the basis for grouping and normalization of histograms.

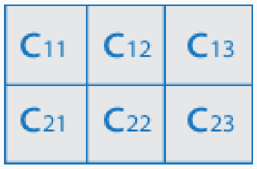
• Normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor (or Feature vector).

We used the extractHOGFeatures function of MATLAB to get the feature vectors. It takes in, other than the image, two important parameters:

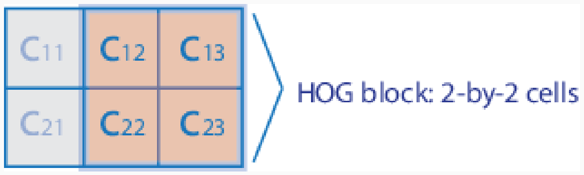
1. CellSize: Size of HOG cell, specified in pixels as a 2-element vector. To capture large-scale spatial information, we increase the cell size, but we may lose small-scale detail.
2. NumBins: Number of orientation histogram bins, specified as positive scalar. To encode finer orientation details, we increase the number of bins. Increasing this value increases the size of the feature vector, which requires more time to process.

Arrangement of Histograms in HOG Feature Vectors

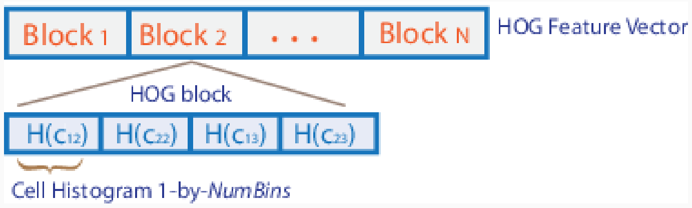
The figure below shows an image with six cells.



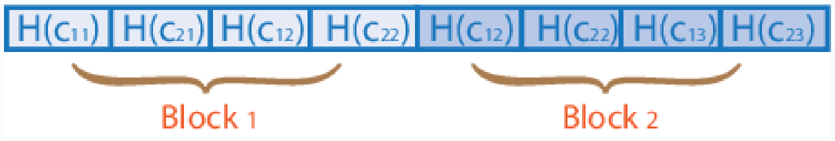
If you set the `BlockSize` to `[2 2]`, it would make the size of each HOG block, 2-by-2 cells. The size of the cells are in pixels.



The HOG feature vector is arranged by HOG blocks. The cell histogram, H(C_{yx}), is 1-by-`NumBins`.



The figure below shows the HOG feature vector with a 1-by-1 cell overlap between blocks.



A visualization showing HOG features superimposed over the original image is shown below:

We need to tweak these parameters to get optimal results on our dataset.

Support Vector Machines (SVMs)

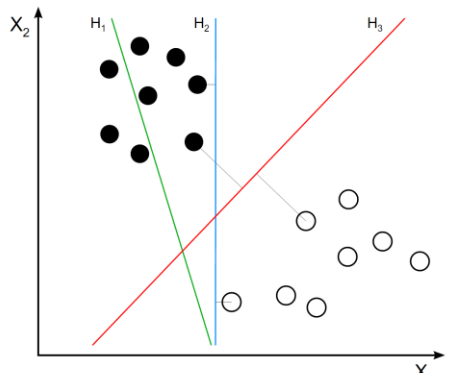
In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

SVMs can be used to solve various real world problems of Uncertainty in Knowledge-Based Systems:

- SVMs are helpful in text and hypertext categorization as their application can significantly reduce the need for labeled training instances in both the standard inductive and transductive settings.
- Classification of images can also be performed using SVMs. Experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback. This is also true of image segmentation systems, including those using a modified version SVM that uses the privileged approach as suggested Vapnik.
- Hand-written characters can be recognized using SVM.
- The SVM algorithm has been widely applied in the biological and other sciences. They have been used to classify proteins with up to 90% of the compounds classified correctly. Permutation tests based on SVM weights have been suggested as a mechanism for interpretation of SVM models. Support vector machine weights have also been used to interpret SVM models in the past. Posthoc interpretation of support vector machine models in order to identify features used by the model to make predictions is a relatively new area of research with special significance in the biological sciences.

SVM are also known as large margin classifiers, since they will find the boundary with the largest separation between classes. In the figure below, SVM will find H3 as the decision boundary instead of H2.



We used the freely available libSVM library to use SVM functions in our code

Feature Extraction and Classification

Getting Feature Vectors for Processed Images

Once the datasets were done, the next step was to feature vectors for them using both HOG and LBP.

For HOG, we considered bin sizes of 9, 12, 15 and 55 and cell sizes of 7, 8, 9, 10 and 11. Subsequent training on SVM classifier and further testing yielded accuracies for different Feature Vectors that led us to conclude that a cell size of 10 and bin size of 15 yielded the best accuracies.

NumBins	**Cell Size**				
	7	**8**	**9**	**10**	**11**
9	97	97	97.25	97.25	96.5
12		97	97.25	97.75	
15			97.25	97.75	96.75
55				97.25	

Accuracies on different cell and bin sizes

Similar procedure was followed for LBP feature vectors and a cell size of 10x10 was found to be optimum here too.

A comparison of accuracies between LBP and HOG feature vectors revealed that HOG was more accurate than LBP (it gave a maximum accuracy of 97.75 compared to LBP's maximum of 96.5). Therefore, we decided to continue training and cross validation with HOG feature vectors only.

Preliminary Test on camera input without cross validation

We tried to see how a model trained on this HOG feature vector performed on test data that was not part of the same dataset. So we tested the trained model on a few random images from the internet as well as on camera input.

While the model did fairly well on googled images, correctly classifying 6 out of 8 of them ('sad' and 'surprise' were misclassified as 'contempt' and 'fear'), the model performed poorly on camera input, predicting 'contempt' more than half of the time, and sometimes failing to recognize even easier emotions like 'happiness'.

Given this result, we decided to remove contempt from the dataset and train the model again. Test accuracies on this aligned and nocontempt model were higher (98.75%) than any other model so far, so we decided to continue with it.

Cross Validating SVM parameters

The trained model was still overfitting, as was evident from the performance on camera input. We performed 10-fold cross validation on the HOG_aligned_nocontempt dataset. It yielded a 10-CV accuracy of 99.72% with c-

=128 and gamma=0.002, which clearly showed that the model was no longer overfitting with these values. Further, training the model using these values and and testing on the Test set resulted in an accuracy of 97.14%.

CONCLUSION

Final Testing

With this new model, the performance on camera input was way better than the preliminary test. There are some instances when the system misclassifies an emotion momentarily. However, in good lighting conditions and a stable input, the system was able to correctly classify all emotions correctly when we tested it.

Problems remaining

While the model is able to correctly classify expressions in images from camera input, the processing time is on the higher side, so that the video output lags a bit. This is because the feature Vector is too large (around 40 MB in size), which in turn is because of redundant or irrelevant features. Feature Selection can be used to improve the performance of the system, both in terms of speed and accuracy.