Using a Soft-Margin Support Vector Machine for Face Detection

Rohan Agarwal, Annabel Buckfire, Nick Paras, Leon Sasson

Introduction

Most people would say that they can tell whether something is a face or not with just a quick glimpse, without thinking at all. And yet, being able to program computers to identify faces in images has proven to be a tremendously hard problem. In this project, we aim to use methods learned in class to apply supervised machine learning models, as well as computer vision concepts and methods, to develop a functional face detection model.

Methodology

Overview

Our method, like most machine learning solutions, starts with a relatively large data set containing images with both faces and no faces. Images are then converted to grayscale and features are extracted using a Histogram of Oriented Gradients (HoG) method (see *Preprocessing and Feature Extraction*). These HoG features are then vectorized and stacked columnwise to form the matrix that our learning model will take as an input. With the data matrix and the known labels (e.g face or no face), our system then uses a perceptron as our first supervised learning model. We then use those learned coefficients as the initial solution and perform 3-fold Cross-Validation to tune the lambda regularizer in the Soft SVM model. We choose the optimal lambda by picking the one with the lowest Cross-Validated testing error. All of our models finds the optimal model coefficients by performing a hessian descent on the soft SVM cost function.

After the model is trained and the optimal coefficients are known, our system is ready to classify new images and find faces in them. Since images are rarely just a face, our system uses a sliding-window algorithm to scan through and image and test whether every sub-patch is a face or not using the trained coefficients. We do this for multiple patch sizes to account for the fact that resolution and relative size of faces often varies from image to image.

Training Data Set

During the process of creating a functional classifier we tried using several datasets. The first dataset we used was the Caltech 10,000 Web Faces data set[1], but we found that the data was not cropped - sometimes there were several faces in an image or actions in the background.

Subsequently we tried using the Yale Face Database B[2], but we found that these faces were *too* cropped. The result of this was that when testing on faces with images, the classifier would not work. There weren't enough subjects, and the cropping meant that the model was not robust to space around a face. The dataset also contained only forward-angled faces.

[1] http://www.vision.caltech.edu/Image_Datasets/Caltech_10K_WebFaces/
[2] http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html

Our final dataset of choice was given to us by our advisor. The faces were cropped to heads and included space around the face, as well as faces at different angles and under different lighting conditions. With this dataset we were able to achieve considerably better results.

Pre-Processing and Feature Extraction

The process of creating our final training data took place in several steps:

1) Importing the faces into MATLAB
2) Creating a Histogram of Oriented Gradients (HoG) feature set
3) Generating non-face images

In order to import the images into MATLAB, the imread function was used along with the rgb2gray function to convert from three channels to a single grayscale channel. We then used the *imresize()* function in MATLAB to convert all the images to 28x28.

At this point, the Histogram of Oriented Gradients was created using the vlfeat package[3] in MATLAB. The process of creating a HoG is to first divide the image into batches of neighboring pixels called cells, then computing edge lengths for every cell with pixel contributions, and finally normalizing cells by grouping adjacent cells into blocks and computing the histograms. Note that the HoG output is in 31 channels, and can greatly increase the number of features if the cell size isn't chosen well. Based on this we chose a cell size of 7x7. Therefore, the final dimension of the HoG features is (28/7)x(28/7)x31 = 496.

Our rationale for using the HoG is that it has several advantages: firstly, it serves an an accurate edge detector as the shape of an image is described using a distribution of edges directions. Secondly, normalizing the histogram blocks creates invariances to changes in lighting conditions. Further, the method of creating the cells does preserve some of the feature locality, but less strictly than just using normalized pixel intensity values.

Once this process is complete for the face images, we generate non-face images and apply the same process. Non-face images are generated by taking arbitrary patch samples from a set of 16 images that contain objects or scenes that are associated with faces but don't directly contain them. Examples include clothes, streets, or classrooms. The sampling process code was adapted from a textbook[4]. After both the facial and non-facial images are loaded, and processed with HoG, we add bias terms and class labels to the data to make it ready for classification.

Creating a Classifier

The underlying classification model that we use to detect faces is a Soft Margin Support Vector Machine. Our implementation of the Soft Margin SVM uses a hessian descent as the iterative optimization to find the optimal weights that minimize the cost function. Note that the Hessian of the soft margin SVM is actually a subgradient as shown in Chapter 9 of the class text and reproduced below :

$$\nabla^2 f(\mathbf{x}) = 2\mathbf{D}_{S(\mathbf{x})}\mathbf{D}_{S(\mathbf{x})}^T + 2\lambda\mathbf{U}.$$

[3] © VLFeat, see end of report for license
[4] http://www.naturalimagestatistics.net/

Since Soft Margin SVM allow observations to be mislabeled, the cost function also includes a penalty parameter, lambda, to control the tradeoff between a small margin and mislabeled observations. To choose lambda, we perform a 3-fold cross validation with lambda in the set {0, 0.001, 0.01, 0.1, 1}, and choose the one that chooses the weights that minimize the training error. We then fit the model with the best lambda to all of our training data, and use it for testing.

Finding faces in new images

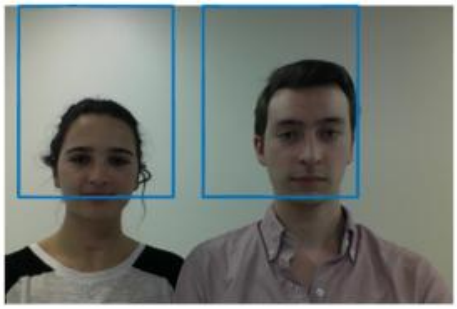
Since most images are not just a single face, like our training data set, we need to use a method to look within an image for a face. To do this, we use a sliding-window algorithm that, conceptually, goes across the image extracting patches and using the trained classifier to test whether each patch is a face or not. The algorithm starts by scanning the input image horizontally and vertically, and extracting patches using multiple different patch sizes. For each of these patches, it performs the same feature extraction and vectorization, that is, each patch gets resized to the appropriate size (we used 28x28) and the same Histogram of Gradient method is applied.

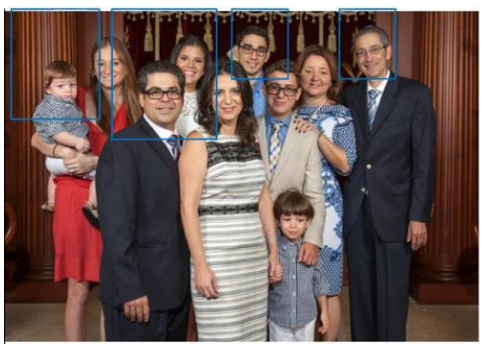
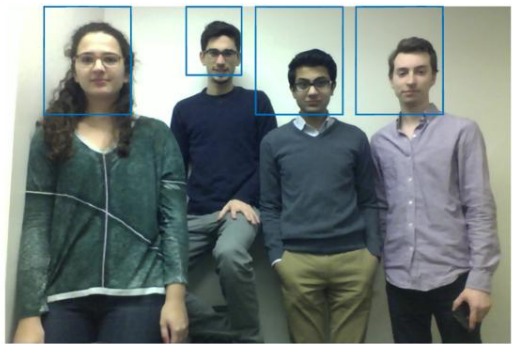
Subsequently, the data is now classified using the trained output of the Soft Margin SVM to determine whether any given patch is a face or not. Since often multiple adjacent patches are classified as a face, our approach is to:

1) Draw a box around the first patch detected
2) For all subsequent patches, if a patch is *not* classified as a face then a box isn't drawn. But if a patch is classified as a face, a box is only drawn if the patch doesn't already intersect with the contents of a previously drawn box

Demo

The demo is comprised of running 14 photos taken from Facebook, Google followed a photo of the four of us. Please see the video at http://youtu.be/9mA_VJ2cCtU and photos:



Results and Discussion

As can be seen in the demo file FINALDEMOSCRIPT.m, available on our github repository, the misclassification error on the training set was 0.003983, and the misclassification error on a testing set (20% of the size of training, however the model was *not* trained using this data) was 0.005417. This suggests that the model while clearly not perfect, was also not substantially overfit. As can be seen in the photos in the Demo section above, our algorithm was fairly successful at locating faces in new photos. It was however, very sensitive to the aspect ratio of the window used in *window_slide.m* function, which took a while to manually tune and still isn't perfect. Further, *window_slide.m* and *drawBoundingBoxes.m* do not handle the box plotting very well in cases where there are a lot of faces closely grouped together, see Limitations and Next steps for more information.

Limitations and Next Steps

As discussed above, we achieved promising results using our methodology. That said, there is always room for improvement, from improving the data set used, to using better features and/or a more efficient optimization method. While the data set of faces we used was large, it was largely well-lit, which proved to be a challenge later on when testing our models on a variety of skin colors in addition to images with many shadows. We also learned that a good data set of non-faces was important, particularly to avoid faces being detected in areas with high grainy-ness but otherwise flat features. To fix these problems, we increased the size of our non-face dataset to include more images with a grainy quality and flat features. We would have liked to have used even more data than the 10000 faces and 50000 non-faces that we did use, but our computers did not have enough memory to perform the matrix operations for a larger sample.

Further, computation time was challenging because it didn't allow us to perform a reasonable size K-fold cross-validation along with a large range of penalty (Lambda) values. In addition, we only tested using one predictive model, soft margin SVM, due to the scope of the project. Going forward, it would be critical to test and cross-validate against other models such as Logistic Regression and Neural Networks. Doing this would be key to determine which model performs the best better for the specific case of face recognition.

Another limitation is that the model has issues detecting faces that are not proportioned to the rest of the photo in a specific way. If a photo was too closely cropped around a face, there are detection problems. On the other end of the spectrum, if the other features in a photo took up much more space than the face, or if the face was proportionally very tiny in the photo, the model cannot detect the face. Finally, the boxes drawn around captured faces are the first and smallest boxes possible. This prevents all the faces or the whole image from being combined into one big box. However, one of the current limits is that since the boxes drawn are the smallest possible boxes, there is a tendency to sometimes cut off ears, chins and faces that are very close together.

Going forward, many of these limitations could be addressed by using better computers (ones that can handle more photos) and by using advanced optimization methods to make the

algorithms faster and more efficient. More specifically, we would like to scale up to use both more face images and more non-faces. In particular, we would want to use more non-face images of walls, curtains, and hair with no faces.

Finally, our window sliding function could be improved to more dynamically address the issues of different sized images and the sensitivity of the model to the aspect ratio of the sliding window used to take image patches. Perhaps there is some way to employ a variation of a binary search methodology rather than manual tweaking and explicit enumeration.

Conclusion

The goal of this project was to implement the Machine Learning algorithms we learned in class to solve the problem of face detection in images. During the course of pursuing this project of creating a face classifier, we were not only exposed to the power of machine learning but also its shortcomings. We were able to see that pictures of anything can be expressed simply in terms of a matrix of numbers which can subsequently be manipulated and analyzed. On the other hand we also found ourselves limited by the data we used, the computers we used, and even the parameters we chose. Despite these limitations we were able to create a real-world application that we feel is a cohesive representation of what machine learning can do.

Bibliography and License Information

1. http://www.vision.caltech.edu/Image_Datasets/Caltech_10K_WebFaces/

2. http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html

4. http://www.naturalimagestatistics.net/