CS385 Machine Learning Project1 Report

Lu Yan

June 2019

1 Introduction

In this project, I mainly conduct 4 parts of experiments: image procesing, classifiers building, face detection and feature analysis. I implement some models on my own, and also utilize library including *OpenCV, sklearn, skimage and Keras* to explore more. I will introduce each task in each section below.

2 Preprocessing

Preprocessing is firstly conducted to generate positive and negative samples. This process including 2 steps as specified below.

2.1 Bounding Box

In this step, I firstly read image using OpenCV and its corresponding annotations, which contains the image center's coordinates, radius, as well as rotation angle of each face. According to annotations, I construct bounding boxes of the size

$$Width = (horizontalaxis * 2) * (1 + 1/3)$$

$$Height = (horizontalaxis * 2) * (1 + 1/3)$$

. Note that the bounding box shares the same center with the ellipse in annotations.

Once having decided the positions of bounding box's four edges, I change the RGB value of those pixels. One interesting thing happened here. At the beginning I changed one line or column of pixels to represent each edge. It turned out that the changes are not adequate and I got some bounding boxes missing edges. Then I expended the pixels color changes to 8 lines/columns and obtained satisfying results as green edges showed in fig. 1 and fig. 2.

Images may contain faces on the border. I use the OpenCV function *cv2.copyMakeBorder()* with parameter *cv2.BORDER_REPLICATE* to fill the empty. The make-up width is decided by the size by which the bounding box's edges exceed image border. Considering multiple faces can appear on the border, I perform checking exceeding and make-up for each face. For example, figure 1 below marks make-up using blue lines.

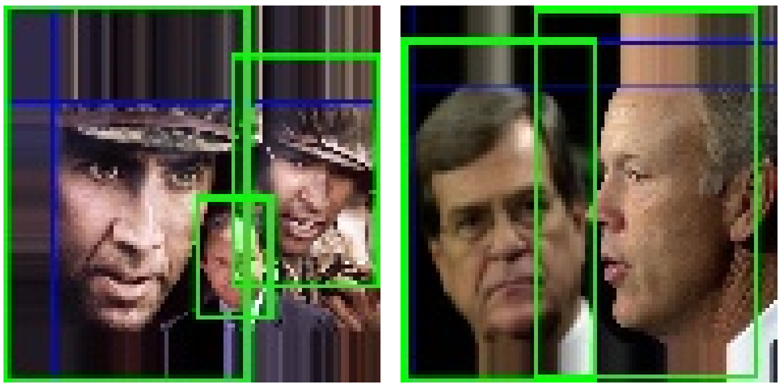

(a) Only 1 face on the border (b) Two faces on the border, make-up twice

Figure 1: Visualization of Make Up

2.2 Create Negative Samples

I generate eight negative images based on each face by sliding the bounding box by 1/3. The figure below 1 shows one example.

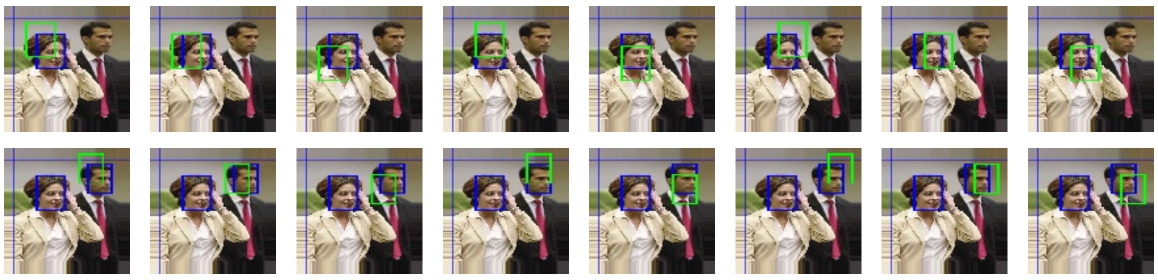


Figure 2: Visualization of the Creating a Negative Sample Process

As required by the assignment file, images cropped along bounding boxes in the first eight folders and created negative samples in the first four folders compose the training set; cropped images in the last two folders and negative images in the last one folder compose the test set. In this way, I obtained 20,683 training samples, in which 4,139 are positive and 16,544 are negative. The test set contains 1,035 positive samples and 4,168 negative samples out of 5,203 images.

3 Extract HOG features

Histogram of Oriented Gradients, also HOG in short, is a widely used image descriptor in Computer Vision. It describes the distribution (histograms) of directions of gradients (oriented gradients). Gradients (x and y derivatives) of an image are useful because the magnitude of gradients is large around edges and corners (regions of abrupt intensity changes) and we know that edges and corners pack in a lot more information about object shape than flat regions. I wrote code of extracting HOG features by referring https://www.learnopencv.com/histogram-of-oriented-gradients/ this website.

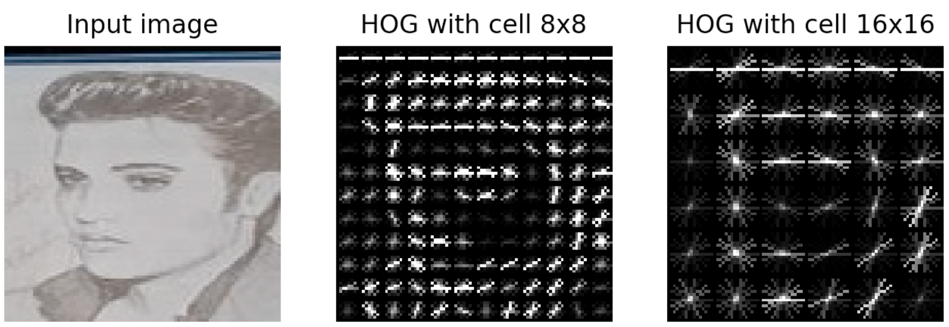


Figure 3: HOG with different cells

3.1 Calculate the Gradient Images

I use the Sobel operator with kernel size 1 in OpenCV to calculate horizontal and vertical gradients. Then *cartToPolar* function is applied to find the magnitude and direction of gradients.

The gradient image removed a lot of non-essential information (e.g. constant colored background), but highlighted outlines. In other words, you can look at the gradient image and still easily say there is a person in the picture.

Since my image has three channels, I choose the magnitude of gradient at a pixel to be the maximum of the magnitude of gradients of the three channels, and the angle to be the angle corresponding to the maximum gradient.

3.2 Calculate Histogram of Gradients in cells

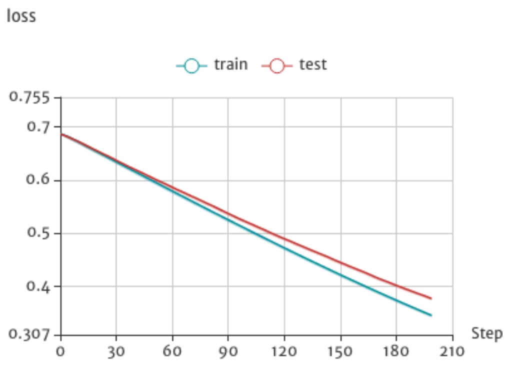
Divide the image into cells achieves more compact representation, i.e., better description since we focus on features of smaller areas. Besides, calculating a histogram over a patch also makes this represenation more robust to noise. Individual graidents may have noise, but a histogram over 88 patch makes the representation much less sensitive to noise.

Traditionally people divide images into 8x8 cells, but the assignment requirements 16x16 cells. I compare the results of different cell size as fig 3 shows (The original image is 2002_08_12_big_img_374). It presents the patch of the image overlaid with arrows showing the gradient — the arrow shows the direction of gradient and its length shows the magnitude. Notice how the direction of arrows points to the direction of change in intensity and the magnitude shows how big the difference is. Although it seems 8x8 cell works better, I still use 16x16 cell for uniformity with others.

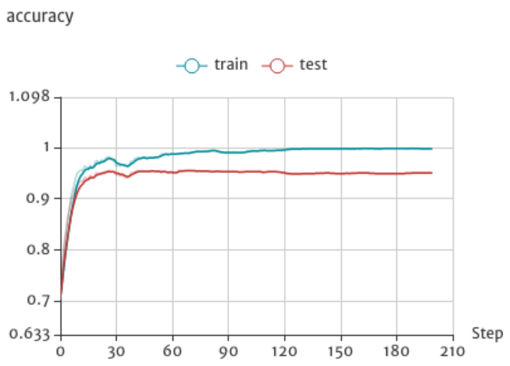
The histogram is essentially a vector of 9 bins (numbers) corresponding to angles 0, 20, 40, 60 ... 160. A bin is selected based on the direction, and the vote, i.e., the value that goes into the bin is selected based on the magnitude.

3.3 2x2 block normalization

Block normalization is applied to make image more robust against lighting. I flatten 9x1 bins of each cell in the 2x2 block into 36x1 vector, and normalize it by dividing its L2 norm. The block moves with the stride 1x1. Thus, I finally get a 5x5x36x1 tensor.



(a) Training and Test Loss



(b) Training and Test Accuracy

Figure 4: Detailed Training Process of Logistic Regression

3.4 results

Concatenate the 5x5x36x1 tensor and generate a 900x1 vector, which is our final result. In this part, I met difficulties when visualizing my gradients, so I use *skimage* API.

4 Create Data Loader

Each image, in both training set and test set is represented by a 900x1 feature vector (HOG results) and a label (1 for positive and 0 for negative). Then I concatenate features of positive samples and negative samples of training and test set into one 20,683x900 matrix and one 5203x900 matrix respectively. Training labels and test labels are stored in two vectors separately with their length the same as the number of samples. and Functions can access those data via a single interface.

5 Classification

I broadly try classification methods, including Logistic Regression, Random Forest, Fisher Model, 4 SVMs, and CNN. Among all the models, CNN and Linear SVM achieve best two performance. I will discuss the details below.

5.1 Logistic Regression

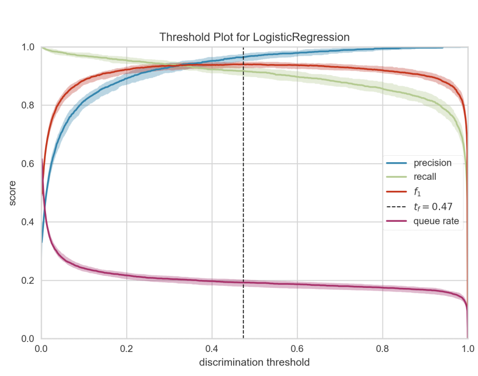
I first wrote Logistic Regression and optimized by mini-batch with batch size 2000. After training for 200 epoches, the result are showed in Table. 1. The detailed training process can be seen in Fig. 4. Basically, xavier initialization makes the model reach about 0.7 initial accuracy. Then the acuuracy jumps in the first 25 epochs and increases smoothly later.

After that, I made deeper exploration about Linear models. For convenience, I used *Sklearn* API in the following experiments.

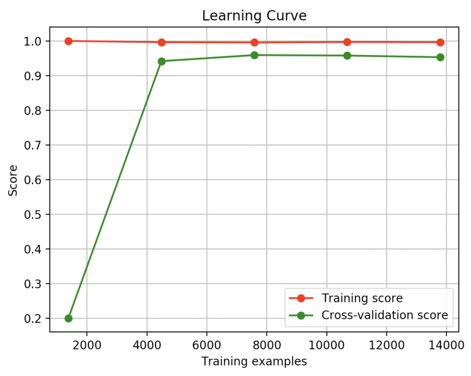
Discrimination Threshold I explore the relationship between prediction metrics (precision, recall, f1-score, queue-rate) of logistic regression model and discrimination threshold (Fig. 5.1. The discrimination threshold is the probability or score at which the positive

	Traing Set	Test Set
No Langiven	0.99	0.948182
With Langiven	0.99	0.950909

Table 1: Logistic Regression With Without Langiven



(a) Metrics Variation with Discrimination Threshold



(b) Training Score Variation with Sample Number

Figure 5: Metrics Variation with Discrimination Threshold

class is chosen over the negative class. Generally, this is set to 50% but the threshold can be adjusted to increase or decrease the sensitivity to false positives or to other application factors. for convience, here I give the definition of the four metrics[4].

- *Precision* An increase in precision is a reduction in the number of false positives; this metric should be optimized when the cost of special treatment is high (e.g. wasted time in fraud preventing or missing an important email).
- *Recall* An increase in recall decrease the likelihood that the positive class is missed; this metric should be optimized when it is vital to catch the case even at the cost of more false positives.
- *F1 Score* The F1 score is the harmonic mean between precision and recall. Optimizing this metric produces the best balance between precision and recall.
- *Queue Rate* The “queue” is the spam folder or the inbox of the fraud investigation desk. This metric describes the percentage of instances that must be reviewed.

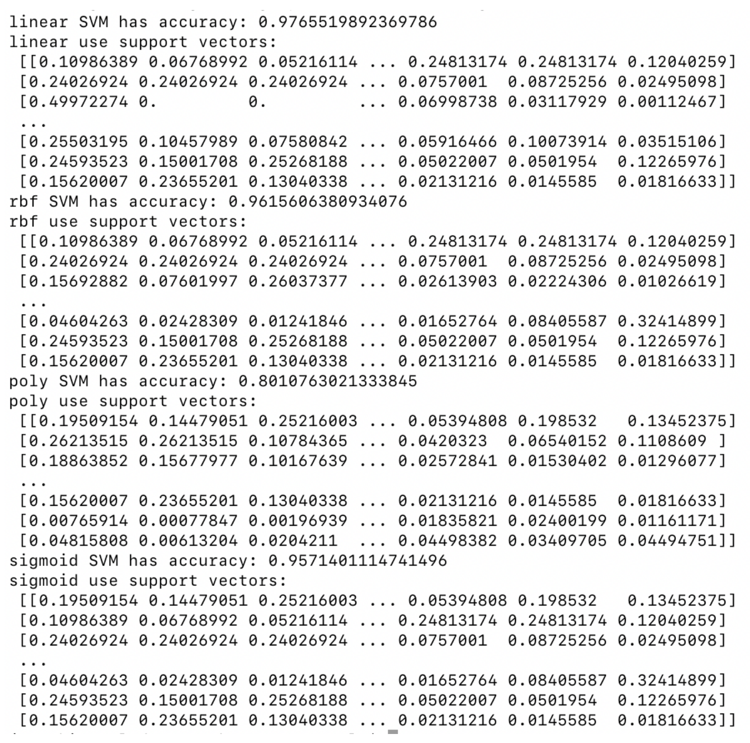
Sample Number Machine Learning is famous for data consuming. The number of total samples in this assignment is large. Do we have enough samples for training an applicable model? Or, are the data way too large to waste training time? To answer these questions, I constructed a random forest model and evaluate its performance with varying data size. The result is showed in Fig. 5.1. We can conclude our datasets are large enough for training.

5.2 Fisher Model

I implement Fisher Model according to LDA part of lecture notes. And below in Fig. 2 shows the results.

intra-class variance	inter-class variance	accuracy
0.00017903	5.15886483e-07	0.924658850663079

Table 2: Results of Fisher Model



```
linear SVM has accuracy: 0.9765519892369786
linear use support vectors:
 [[0.10986389 0.06768992 0.05216114 ... 0.24813174 0.24813174 0.12040259]
 [0.24026924 0.24026924 0.24026924 ... 0.0757001  0.08725256 0.02495098]
 [0.49972274 0.         0.         ... 0.06998738 0.03117929 0.00112467]
 ...
 [0.25503195 0.10457989 0.07580842 ... 0.05916466 0.10073914 0.03515106]
 [0.24593523 0.15001708 0.25268188 ... 0.05022007 0.0501954  0.12265976]
 [0.15620007 0.23655201 0.13040338 ... 0.02131216 0.0145585  0.01816633]]
rbf SVM has accuracy: 0.9615606380934076
rbf use support vectors:
 [[0.10986389 0.06768992 0.05216114 ... 0.24813174 0.24813174 0.12040259]
 [0.24026924 0.24026924 0.24026924 ... 0.0757001  0.08725256 0.02495098]
 [0.15692882 0.07601997 0.26037377 ... 0.02613903 0.02224306 0.01026619]
 ...
 [0.04604263 0.02428309 0.01241846 ... 0.01652764 0.08405587 0.32414899]
 [0.24593523 0.15001708 0.25268188 ... 0.05022007 0.0501954  0.12265976]
 [0.15620007 0.23655201 0.13040338 ... 0.02131216 0.0145585  0.01816633]]
poly SVM has accuracy: 0.8010763021333845
poly use support vectors:
 [[0.19509154 0.14479051 0.25216003 ... 0.05394808 0.198532   0.13452375]
 [0.26213515 0.26213515 0.10784365 ... 0.0420323  0.06540152 0.1108609 ]
 [0.18863852 0.15677977 0.10167639 ... 0.02572841 0.01530402 0.01296077]
 ...
 [0.15620007 0.23655201 0.13040338 ... 0.02131216 0.0145585  0.01816633]
 [0.00765914 0.00077847 0.00196939 ... 0.01835821 0.02400199 0.01161171]
 [0.04815808 0.00613204 0.0204211  ... 0.04498382 0.03409705 0.04494751]]
sigmoid SVM has accuracy: 0.9571401114741496
sigmoid use support vectors:
 [[0.19509154 0.14479051 0.25216003 ... 0.05394808 0.198532   0.13452375]
 [0.10986389 0.06768992 0.05216114 ... 0.24813174 0.24813174 0.12040259]
 [0.24026924 0.24026924 0.24026924 ... 0.0757001  0.08725256 0.02495098]
 ...
 [0.04604263 0.02428309 0.01241846 ... 0.01652764 0.08405587 0.32414899]
 [0.24593523 0.15001708 0.25268188 ... 0.05022007 0.0501954  0.12265976]
 [0.15620007 0.23655201 0.13040338 ... 0.02131216 0.0145585  0.01816633]]
```

Figure 6: Results of SVM

5.3 Support Vector Machine

I tried 4 kinds of SVMs, namely, the Linear SVM, the RBF-SVM, the Poly-SVM and the Sigmoid-SVM. Table 6 are their results.

5.3.1 Analysis

Generally speaking, RBF Kernel is able to project a sample to higher dimensional space and handle non-linear situations. It has less parameters than Sigmoid and Poly Kernel and easier to train. Linear Kernel is a special case of RBF Kernel [5]. However, since our feature size is not large enough for RBF Kernel to operates, Linear Kernel fulfills our object better. Also, when running the code, waiting for RBF results costs most time, about 1 hour in comparison of others' a few seconds. The difficulty of design RBF parameters is revealed in Table 3. With the changing of parameters, performance varies from model to model. Among all 4 models, Linear model has highest robustness. It makes sense because it is easiest and

Param C	Param gamma	Linear SVM	RBF SVM	Ploy SVM	Sigmoid SVM
default	default	0.9765519892	0.961560638	0.8010763021	0.95714011147
1.0	2.0	0.9765519	0.80165289	0.9857774	0.8010763
2.0	2.0	0.9767441	0.80165289	0.9857774	0.8010763
1.0	4.0	0.976552	0.8016528	0.985774	0.801076

Table 3: Results of SVM with Different Parameters

batch size	epoch number	structure	accuracy
no	3	Conv(64, 3, 3)+relu+Conv(32, 3, 3)	0.981933
40	3	same as above	0.981346
no	10	above+MaxPooling(2, 2)	0.98097
no	10	above+Conv(16, 2, 2)	0.98039

Table 4: Trials on the first CNN model Training

Model	Linear SVM	RBF SVM	Poly SVM	Sigmoid SVM	CNN
Accuracy	0.9765519892	0.961560638	0.8010763021	0.95714011147	0.983855

Table 5: Classification Results of Models

insensitive to parameters. RBF and Sigmoid Kernel behaves similarly as expected. Poly Kernel performs the opposite of RBF Kernel and Sigmoid Kernel. This teaches me that some tricks can be applied in designing Kernel Parameters, like libsvm. I leave it as future work.

In the default setting, gamma is 1/num features. As for my data, it is extremely small. In this case, RBF and Sigmoid achieve good accuracy. When setting gamma to 1 or 2, RBF and Sigmoid both has worse results, which may be due to over-fitting. Nonetheless, Poly Kernel outputs better performance, which suggests it may have higher Resistance to large gamma value. Enlarging panelty C can increase Linear SVM's accuracy.

5.4 CNN

I build two CNNs using Keras API. The first one is adapted from Keras example of Mnist and used for HOG features of training and test images input. It achieves 0.983278878 accuracy at most. I also test the relationship between accuracy and network structure, batch size, epoch number and input shape. The results are as Table 4 shows. This suggests a simple model can handle our data. Complicated model, especially with MaxPooling may lose information.

The second takes training and test image directly as inputs. Its structure shows in Fig. 7. I trained for 10 epoches and achieves 0.983855.

6 Detection

In this section, I utilized the combination of 3 methods and 4 models for face detection. The performance of each method is measured by IoU, a widely used metric in object detection defined as:

$$IoU = \frac{Overlap\ area\ of\ detection\ windows\ and\ ground-truth\ windows}{Union\ area\ of\ detection\ windows\ and\ ground-truth\ windows}$$

Larger IoU value represents better detection accuracy. I will first briefly introduce each model and then compare and analyze their results.

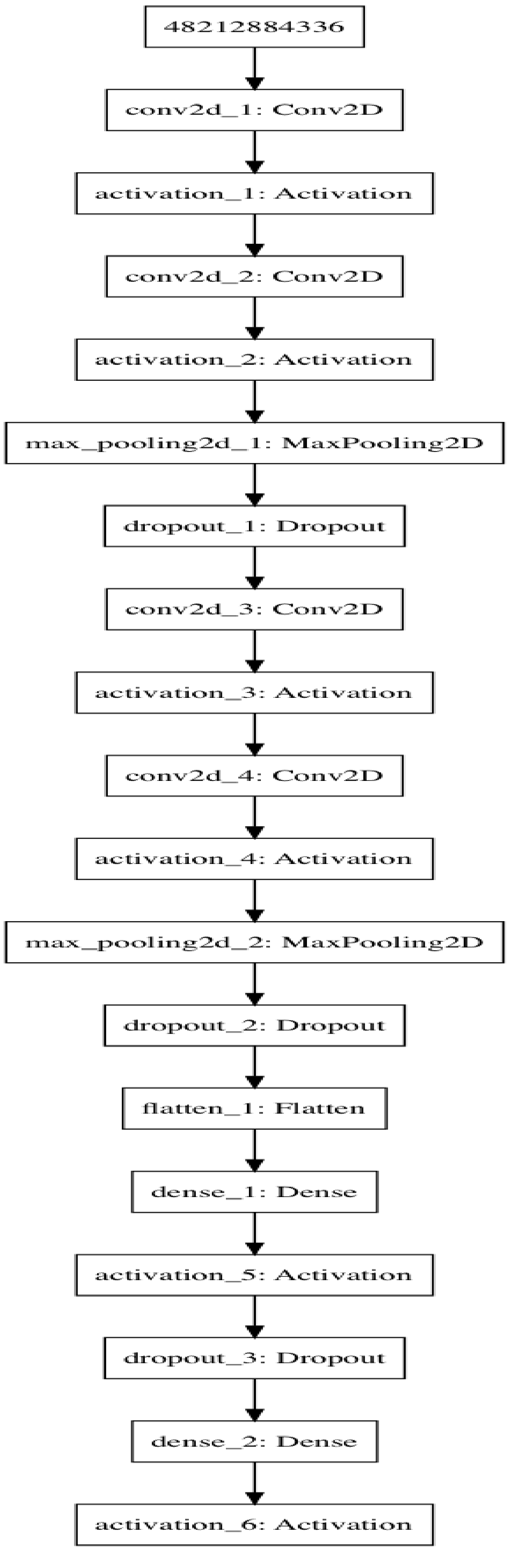


Figure 7: CNN Model for Image Input

6.1 Detection Method

6.1.1 Brute Force

This is the most direct and intuitive way, also the suggested way in assignment file. I implement this approach by moving windows of various size along the whole image. The sizes of windows range from $\frac{1}{10}$ image height/width to $\frac{1}{3}$ image height/width with step $\frac{1}{10}$ image height/width. In practice, this method is extremely time consuming. I spent more than one day to detect images of 30 files using Linear SVM. Besides, the performance is also insatisfactory: the average IoU value of 30 files is about 0.048. Thus, in the following experiments I abandoned this method.

6.1.2 Selective Search

Selective Search is a region proposal algorithm used in object detection [3]. It is based on computing hierarchical grouping of similar regions based on color, texture, size and shape compatibility. Selective Search starts by over-segmenting the image based on intensity of the pixels using a graph-based segmentation method and then iteratively merging the adjacent regions similar to each other until reaching one segmented region covering two objects. At each iteration, larger segments are formed and added to the list of region proposals. Hence it creates region proposals from smaller segments to larger segments in a bottom-up approach.

6.1.3 Skin Detection

When searching online about detection method, this paper attracted my attention[1]. This paper describes a novel detection method which separate faces and background by color. Inspired by this paper, I propose a new approach that combine morphological transformations with skin detection.

Ellipse Detection According to rich statistical analysis of human skin, skin pixels will cluster into an ellipse-shaped distribution when converting image from RGB space to YCrCb space. My intuition in this step is that I can define a standard CrCb ellipse first and then distinguish face pixels after transforming test images. More specifically, a pixel belongs to faces if it lay inside the ellipse after transformation and vice versa. In terms of implementation, I created a mask that is transparent on face pixels and black otherwise. The second image of fig 8 shows the result of covering original images with this mask. The mask has filtered most non-face pixels. However, the pixels passed-through scatter all over the image, which causes difficulty to bound faces in boxes.

Morphological Transformation Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images [2]. I use this method trying to filter noise and obtain a more centered face pixels distribution.

erosion The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object. All the pixels near boundary will be discarded depending upon the size of kernel. It is useful for removing small white noises. In the experiment, I set the kernel size to be 1.

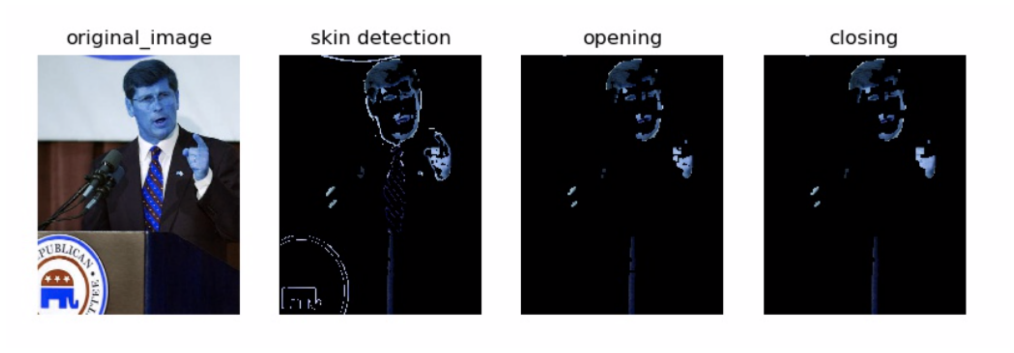


Figure 8: Image Process with Skin Detection

IoU results	Linear SVM	RBF SVM	CNN	Logistic Regression
Selective Search	0.22772262492827303	0.27095302784405473	0.06596	0.22632363908853365
Skin Detection	0.3156385302682977	0.38635427820803275	0.03947	0.3181201476230371

Table 6: Results of Detection: IoU value

dilation This method is opposite to erosion. It increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

- *opening* Opening is just another name of erosion followed by dilation. I conduct opening using kernel of size 1 to cluster white region of mask. Obviously from Fig. 8, after performing this, noise on the lower left corner and top center region has been removed. However, white pixels of the man's hand are still unconnected.

- *closing* closing means dilation followed by erosion. I hope dilation can connect the gaps in the area of the man's hand. This process may bring noise, though. Thus, erosion follows to reduce the negative influence of dilation. The last image of Fig. 8 proves my idea works.

Find Bounding Rectangles After the steps described before, I have removed noise pixels from face area candidate. Next, I use OpenCV's API *findContours* to sketch the contours of skin region. Then another API *boundingRect* finally locates the bounding box. The location information is returned to classifiers.

My method achieves the best performance when detecting faces. Furthermore, it runs much more quickly than the other two methods because its direct way to focus on face candidate.

6.2 Result and Analysis

The IoU results of all combinations are as Table 6 shows.

From the table, we can conclude that RBF SVM have better performance over linear ones. This is in line with our expectations because images are 2-dimensional information. However, CNN has an extremely bad results. I still cannot find the reason yet.

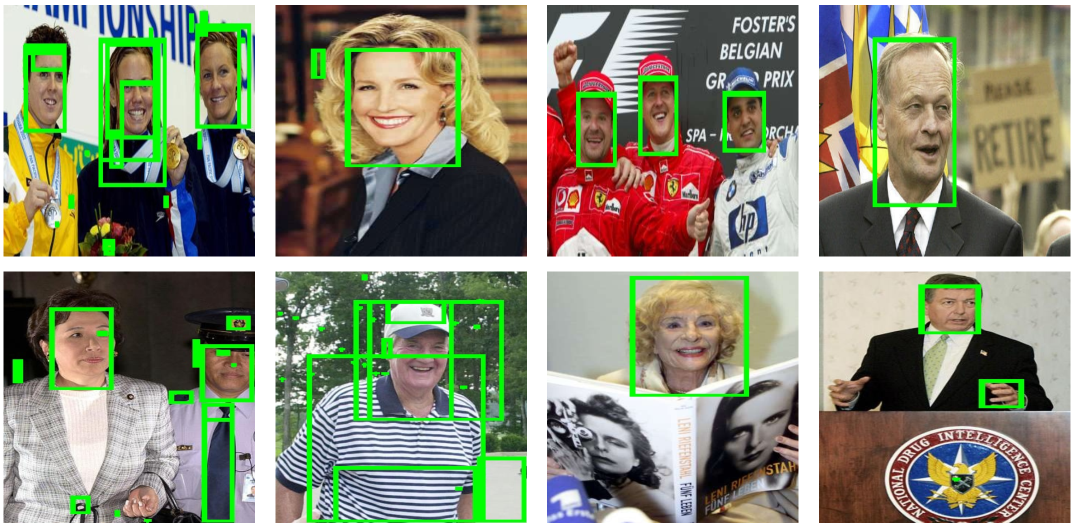


Figure 9: Detection Results: images in the first line are correctly classified; images in the second line are mistakenly classified. The first two images in each line obtains using Selective Search; the last two images in each line obtains using Skin Detection.

Also, Fig. 9 presents examples that are correctly classified in the first line. In the second line are some examples being mistakenly classified by the same model (Linear SVM) and different method. I try to analyze the shortcomings of each method from them.

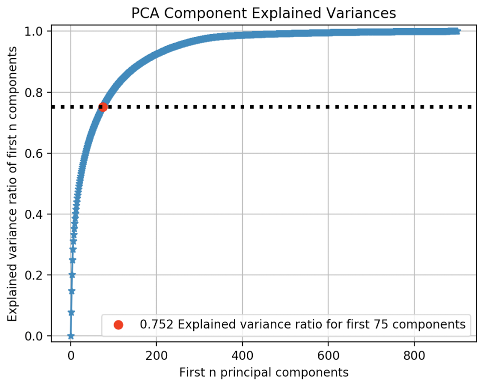
Surprisingly, my method beats famous Selective Search model in this experiment according both the IoU values and visualization results. We can tell from the correctly classified images that Skin Detection separates face regions precisely and is robust to noise (in the last image in the first line, the color of the man's face is similar to that of background). However, Selective Search is more likely to make mistakes. It often picks up unrelated areas or picks up enlarged areas.

One reason could be too much noise in test images as for Selective Search. As mentioned above, Selective Search lays weight on texture. Thus, it encounters noise pixels that has similar texture to faces (for example, in the second image in line 2 of Fig. 9, the man's T-shirt and trees around him distract its attention). Also, Selective Search is a bottom-up approach, which means it cares more about details. Thus, in the first negative image, the button on the woman's coat and the man's tie and shoulder strap, etc. are framed.

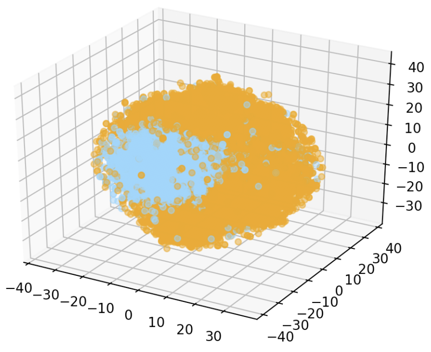
Nonetheless, constrained by the basic idea, it often happens that Skin Detection miss faces colored in black and white or falsely vote for other body parts (as showed in the last two images of the second line).

7 Feature Analysis

Each piece of data in this experiment has 900 features. It is interesting to explore the relationship between these features. I plot Fig. 10 to show linear relations between features using PCA. It suggests 400 components almost cover all the features of data. I use t-SNE to reduce 75 principal components to 3 dimension. In the figure 10(b), the yellow points represents positive samples and the blue points represents negative samples. We can see samples mingle with each other, which suggests using only 3 features even t-SNE,



(a) PCA Components Explained Variances with Components Number



(b) t-SNE Visualization Using 75 Principal Components

Figure 10: Feature Analysis: according to left figure, 75 components can explain 0.75 information; the right figure reduces 75 components to 3 dimension, where the yellow points represents positive samples and the blue points represents negative samples.

a visualization method with higher performance for non-linear situation cannot separate postive and negative samples.

8 The End

In the experiment, I was stuck in traps many times, most because the shape inconsistency. I explored various visualization methods and apply them into the experiment. Furthermore, I accumulate basic knowledge about computer vision and are more familiar with Machine Learning classifiers.

9 Reference

[1] http://www.c-s-a.org.cn/csa/ch/reader/create_pdf.aspx?file_no=20080715flag=1year_id =7quarter_id=
[2] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/ py_morphological_ops.html
[3] Uijlings J R R, Van De Sande K E A, Gevers T, et al. Selective search for object recognition[J]. International journal of computer vision, 2013, 104(2): 154-171.
[4] https://www.scikit-yb.org/en/latest/api/classifier/threshold.html
[5] https://scholar.google.com/scholar?hl=enamp;q=Asymptotic+behaviors+of+support+ vector+machines+with+Gaussian+kernelamp;btnG=amp;as_sdt=1%2C5amp;as_sdtp=