Design of an Automatic Facial Expression Detector

Jian Liang

An essay presented for the degree

of

M.Math

Applied Mathematics

University of Waterloo

2018/01/26

Contents

1 Abstract

Facial expression recognition (FER) has attracted significant attention in computer vision. It has many applications in real-life, such as human-computer interactions, surveillance, visual reality, video conferencing, customer satisfaction studies. Due to its fast-growing need in the market, many researchers have proposed different methods to improve both accuracy and processing speed. So far, many FER systems can achieve real-time recognition. Facial expression classification has three steps: facial detection, facial alignment, and feature extraction combined with classification. Traditional computer vision finds features and then applies classifiers on top of the feature. However, due to the recent rapid development of deep neural networks, both academic and industrial research are sometimes more likely to combine feature extraction and classification into one network structure. In this paper, we will discuss some effective methods that can be used for each step and will design our own automatic facial expression detector.

2 Introduction

Facial expressions are more powerful than language. You can see through a person by only reading his or her facial expression. A series of facial expression images have the potential to predict whether this person is lying or not. However, the reality is that as humans, we cannot even assure that we would not make any mistakes in reading facial expressions. Many datasets have seven categories: anger, disgust, fear, happiness, sadness, surprise, and neutral. By only reading faces, happiness, sadness, and surprise are relatively easy to distinguish, but the rest are not. Some other datasets that label facial emotions with more than one emotion since people can have complex emotions as well. For example, a person can be both disgusted and angry, which makes the classification task more difficult. In this paper, we focus on performing classification on the FER2013 dataset which contains only one emotional label for each image. FER2013 is a dataset in the Kaggle competition [1], and it contains seven different facial expressions. In the training data, there are 28709 images. Faces in those images are registered which means that the faces are already aligned. Each has a size of 48×48 pixels. The first-place winner on the leaderboard achieved 71% accuracy. A subset of these images are shown in Fig. 1.

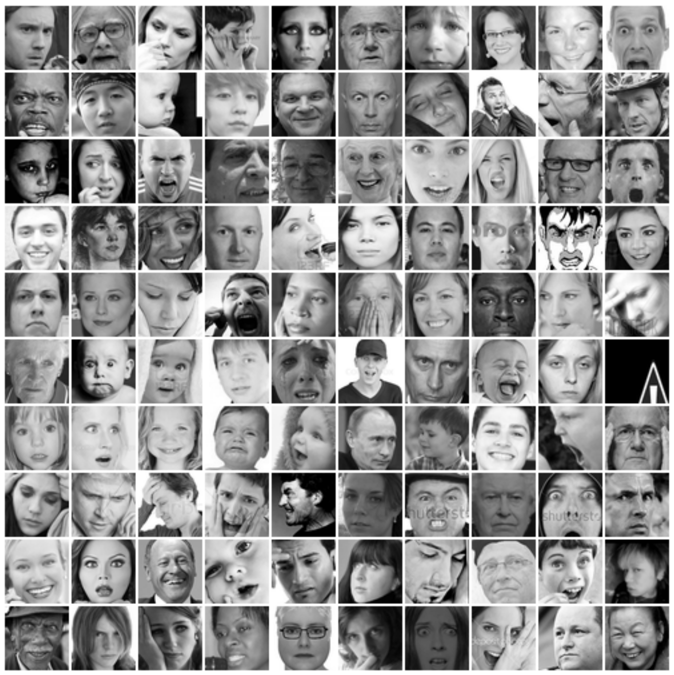


Figure 1: FER2013 data

In the FER2013 dataset, we are working with grey scale images. We usually express a digital image by a $M \times N$ matrix. Customarily, we denote intensity value of a pixel as a_{ij} or $a(i,j)$, $1 \leq i \leq M$, $1 \leq j \leq N$. In an 8 bit-per-pixel image, all elements a_{ij} are integers in the domain of [0,255]. This is stored as UInt8 type. Here, $a_{ij} = 0$ is black and $a_{ij} = 255$ is white.

The dataset states that the faces have been automatically registered which means that the faces are being centered and have roughly the same sizes. However, our goal is to design an automatic detector. We are asked to detect and create multiple registered faces from the original image or video frame. We can do this by using the Viola-Jones face detector method and some face alignment methods. We use face alignment methods to find facial landmarks, such as the tip of a nose, canthi, etc. Next, we want these landmarks to occupy roughly the same sizes within a 48×48 image.

We organize the layout of this essay as follows. Section 3 and 4 deal with facial detection and face alignment, respectively. In Section 5 and 6, we explain feature extraction and classification, respectively. In Section 7, we discuss the deep neural network method that can perform feature extraction and classification. In Section 8, we provide some experimental results. Finally, in Section 9, we draw some conclusions from our observations.

3 Face detection

3.1 Viola–Jones method

Paul Viola and Michael J. Jones [34] developed a face detection method using the so-called "Haar feature and cascading classifier." Even though the Viola-Jones method is more robust in finding the front face than the side face, it is still widely used in the field. This approach has four steps. It uses the Haar feature to find common attributes of faces. The integral image method is then used to calculate block subtractions efficiently. Next, it uses a classifier that can distinguish faces from non-faces. Finally, a cascading classifier is used so that we do not have to scan through an entire image with different size filters.

In general, faces have some common attributes. For example, the eye region is darker than the cheeks, and the nose is brighter than surrounding area. The Haar feature concentrates on the rectangular neighborhood region at a specific location. It accumulates the pixel values from the sub-rectangular regions and then subtracts them from each other. In the paper, we have two, three, and four sub-rectangular Haar features which are shown in Fig. 2. For the two sub-rectangular Haar feature, we have six critical locations that include four corners and two points on the boundary that join two sub-rectangles together. Therefore, we need six, eight, and nine points to indicate the locations of the block within two, three, and four sub-rectangular Haar features, respectively, and we present these positions in red in Fig. 2. More importantly, the Haar feature has different sizes to capture different scales of faces. Therefore, within a 24×24 image, we have 162336 features.

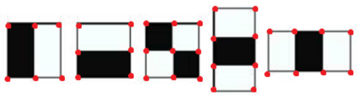
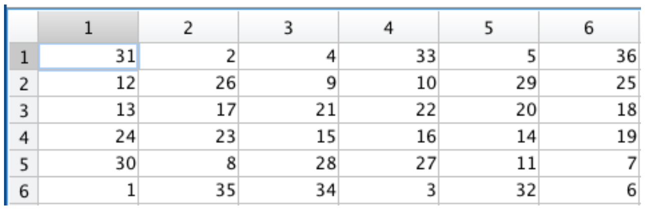


Figure 2: the Haar features [23]

The above computation is very expensive. Viola and Jones purposed an "integral image method" to calculate the sum of pixel values in any rectangular region of an image. Given an image f, the integral image I is defined as follows,

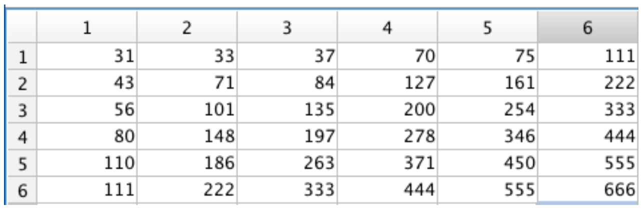
$$I(x,y) = \sum_{1 \le x' \le x} \sum_{1 \le y' \le y} f(x',y') \quad , \tag{1}$$

where (x',y') indicates a pixel location. We show original image f and its corresponding integral image I in Fig. 3 and Fig. 4, respectively. [9].



	1	2	3	4	5	6
1	31	2	4	33	5	36
2	12	26	9	10	29	25
3	13	17	21	22	20	18
4	24	23	15	16	14	19
5	30	8	28	27	11	7
6	1	35	34	3	32	6

Figure 3: An original image f



	1	2	3	4	5	6
1	31	33	37	70	75	111
2	43	71	84	127	161	222
3	56	101	135	200	254	333
4	80	148	197	278	346	444
5	110	186	263	371	450	555
6	111	222	333	444	555	666

Figure 4: The integral image I

In order to compute the integral image more efficiently so that we do not have to use the accumulated summation of the intensity values in every calculation, we record the portion of the integral image and then calculate I as follows,

$$I(x,y) = f(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1) \quad . \tag{2}$$

For example, given $f(2,3) = 17$ in Fig. 3, we want to find $I(2,3)$. From the known values of $I(x,y)$ for $x < 2$ and $y < 3$, we compute, using Eq. (1), $I(2,3) = 17 + 71 + 56 - 43 = 101$. Suppose a, b, c, d stands for the positions of the rectangle in f. To calculate the sum of pixel values of rectangle $abcd$ we assume $a = (x_a, y_a), b = (x_b, y_b), c = (x_c, y_c)$, and $d = (x_d, y_d)$. The sum of pixel values within the rectangle can be expressed as follows,

$$S_{a,b,c,d} = I(x_d, y_d) - I(x_b, y_b - 1) - I(x_c - 1, y_c) + I(x_a - 1, y_a - 1) \quad . \tag{3}$$

For example, referring back to Fig. 3, if we want to find the sum of the pixel within the rectangle $abcd$ such that $a = (3,4)$, $b = (5,4)$, $c = (3,5)$, and $d = (5,5)$. Therefore, $S_{a,b,c,d} = I(5,5) - I(5,3) - I(2,5) + I(2,3) = 450 - 254 - 186 + 101 = 111$. Now, we can find the Haar features efficiently by above method.

AdaBoost [16] is the strong classifier that is able to classify faces and non-faces. It has a series of weak classifiers and a weak classifier can be in any form. Assume that each feature is a high dimensional data point and assume we have n of these data points. We treat these data points as our training data. In this case, the Haar features of a region may or may not contain a face is a data point.

We assign a weight w_i to each data point, setting $w_i = \frac{1}{n}$ initially. Assuming that we want to use J number of weak classifiers h, for each $j \in [1, J]$, we pick the optimal h_j which minimizes the weighted error as:

$$L_j = \frac{\sum_{i=1}^{n} w_i I[y_i \neq h_j(x_i)]}{\sum_{i=1}^{n} w_i} \quad . \tag{4}$$

Here, y is the label and I is the Boolean function. If our prediction $h_j(x_i)$ predicts the output correctly, I produces 1, otherwise 0. In extreme case, $L_j = 0$ indicates the weak classifier h_j is good enough to classify all training data correctly which is not likely if we have sufficient data to train. $L_j = 1$ indicates the otherwise. Next, we define the corresponding weight $\alpha_j = log\frac{1-L_j}{L_j}$ for this particular weak classifier h_j. If L_j is indeed equal to 0, even though, we fail to assign weight; however, we already found the robust classifier. We simply assume L_j is not equal to 0. By assignment weights to these classifiers, we can overweight the weak classifier with lower L_j and underweight the weak classifier perform poorly on training data. So far, we have obtained a weak classifier that can reasonably classify training data as faces or non-faces. We now want to pay more attention to the training data that was not classified correctly by each weak classifier h_j. Therefore, we update the weight of all training data with respect to each weak classifier h_j, putting more weight on the misclassified training data as follows,

$$w_{i,j}^{new} = w_i e^{\alpha_j I[y_i \neq h_j(x_i)]} \quad . \tag{5}$$

We find the next weak classifier and continue this process until we obtain J number of weak classifiers. We can simply assume weak classifier as a simple threshold function. During each iteration, we determine how well each weak classifier performs and then design our final classifier as follows,

$$h(x) = sign\big(\sum_{j=1}^{J} \alpha_j h_j(x)\big) \quad . \tag{6}$$

Finally, we use the cascading method. The idea of cascading is to pay attention to regions that potentially contain faces and removing areas that have no faces at the beginning. We cascade the strong classifiers that trained using AdaBoost. At any level, if the strong classifier rejects a window which contains faces, we reject this window.

4 Face alignment

4.1 Related work

Most face alignment methods can be traced back to the original paper on the so-called active shape model (ASM) [11]. It is sometimes called the "smart snake model" because ASM was developed from an active contour model, sometimes called the "snake model." We will be focusing on the active contour model method since this is the ancestor of many other face alignment methods. Other methods based on ASM include using the active appearance model (AAM) [10] and Constrained Local Model [13]. Following this roadmap, the relatively new algorithms that have been developed are Cascaded Pose Regression [15] and Explicit Shape Regression [8][28].

Here, we also mention other methods such as predicting the facial landmarks by gradient tree boosting and cascade [19], Supervised Descent Method combined with Parameterized Appearance Model [37], and finding face alignment with shape regression [8]. Also, Bulat [7] successfully aligns face using FAN network and was able to transfer a 2D face to a 3D face. Mollahosseini [25] uses bidirectional wrapping for AAM fitting. Zhu [41] uses mixing of trees model to align faces.

4.2 Active Contour Model

Here, we introduce the Active Contour Model, also called the "snake model." Since Kass [18] invented this method in 1987, many segmentation methods other than face alignment were developed based on the snake model. The snake model is particularly useful when we know the approximate shape of the contour. It is a deformation model that is based on the points that roughly outline the object which adjusts the position of these points to minimize an energy function. We can think of ASM as a discrete version of an active contour model. The difference is that instances of ASM are deformed by training data with handcrafted landmarks.

A snake mode has initial control points $v(s) = [x(s), y(s)], s \in [0, 1]$ which are connected head-to-tail to form the initial snake. The energy function has the form:

$$E^*_{snake} = \int_0^1 E_{snake}\big(v(s)\big)ds = \int_0^1 [E_{int}\big(v(s)\big) + E_{image}\big(v(s)\big) + E_{con}\big(v(s)\big)]ds \quad , \tag{7}$$

where

$$E_{int} = (\alpha(s) \mid v_s(s) \mid^2 + \beta(s) \mid v_{ss}(s) \mid^2)/2 \quad , \tag{8}$$

and

$$E_{image} = w_{line}E_{line} + w_{edge}E_{edge} + w_{term}E_{term} \quad . \tag{9}$$

Here, E_{int}, E_{image}, and E_{con} are the internal energy, image forces, and constraint forces, respectively. The external energy is combined with image forces and constraint forces. Our goal is to minimize the energy of the closed snake v.

In the internal energy equation, the first derivative of v indicates the elastic energy and the second derivative of v gives us the bending energy of the snake. The parameters α and β are adjustable. We can think of α as the coefficient for succession, and it penalizes the distance of successive points on the snake. β is coefficient of curvature. It punishes an oscillation of the snake. If we set $\beta = 0$, we allow corners in our snake. We use internal energy to control the internal deformation of the snake, and it can preserve the continuity and smoothness. Internal energy has nothing to do with our object but only the shape of the snake.

Now, suppose we have a facial image I. According to the above equations, we can express the image forces as a weighted linear combination of E_{line}, E_{edge}

and E_{term}.

Generally, we define $E_{line} = I(x, y)$ and the sign of w_{line} to decide whether the snake is attracted to brighter or darker contour. In addition, we want

$$E_{edge} = - \mid \nabla I(x, y) \mid^2 \quad , \tag{10}$$

so that the snake is attracted to the contour that has the larger gradient. As the snake moves toward in the direction of increasing gradient, the external energy decreases. Next, we define

$$E_{term} = \frac{\partial \theta}{\partial n_\perp} = \frac{\partial^2 C / \partial n_\perp^2}{\partial C / \partial n} = \frac{C_{yy} C_x^2 - 2C_{xy} C_x C_y + C_{xx} C_y^2}{(C_x^2 + C_y^2)^{3/2}} \quad , \tag{11}$$

where $\theta = tan^{-1}(C_y/C_x)$, $n = (cos\theta, sin\theta)$, $n_\perp = (-sin\theta, cos\theta)$ and C is the curvature of snake. Finally, E_{con} is optional and it can guide the snake to move under some particular constraint.

Even though the snake model was initially designed for segmentation, many researchers expanded the usage of snake model, and ASM is one of many examples. In ASM [11], the authors prepared hand labeled face images as training data. They labeled critical landmarks and established correspondence for each training example, such that using four landmarks on the mouth, three landmarks on the tip of the nose and many other landmarks forming one high dimension vector. Suppose that n landmarks are labelled on the 2D Cartesian coordinate system, we will have $2n$ dimension vector. The authors denoted these landmarks as feature points. The goal is to scale, rotate, and translate these feature points to fit a new face.

In hand-craft training data, we are not able to label these features points that exactly located on edges and this is the reason why snake model is useful. By connecting these manually labeled features point, we have a contour that indicates the rough shape of the organ such as a mouth. The Active Contour Model can adjust the contour to better fit the shape of the organ and hence, we can update the location of feature points in training data. For example, in Fig. 5, all feature points are away from the object. In this case, the external energy is high and the snake is attracted to higher gradient which is the boundary of the object. Also, in Fig. 6, we display a result of face alignment using ASM.

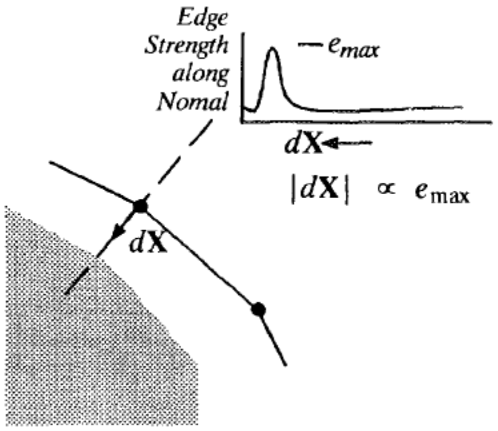


Figure 5: adjust training feature points using snake model

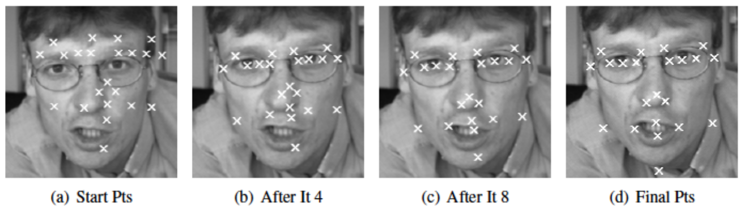


Figure 6: result of face alignment using Active Shape Model [14]

5 Feature extraction

5.1 Related work

Some methods classify emotion using the Local binary patterns (LBP) feature [31, 30]. Many others use the Gabor feature [40]. Wiskott developed a Gabor function-based elastic bunch graph matching (EBGM) to recognize facial images [36]. It was an improvement of Dynamic Link Architecture framework (DLA) by Lades [21]. Zhang's method [39] uses the histogram of Gabor features to recognize faces. These methods showed a significant improvement compared to other techniques for face recognition.

Some other feature extraction methods also perform well. For example, Mollahosseini [24] uses bidirectional warping of Active Appearance Model and IntraFace to extract facial landmarks. IntraFace uses SIFT features for feature mapping. Boughrara [6] presents the use of a biological vision based facial description called perceived facial image and then reduces the dimension using PCA. Kotsia [20] applies a 3D face mask called the "Candide grid" on an image. Wang [35] uses 3D shaded facial range model. Bartlett [4] uses Haar features with cascade. Tian uses [33] both geometric feature and appearance feature.

5.1.1 Local Binary Patterns

Ojala's proposed LBP method [27] is a good method for texture feature finding. LBP focuses on the intensity values of images. The operators of LBP were extended to different shapes. But the main idea is the same. We use the center pixel intensity as the threshold and label the surrounding pixel into a binary pattern. It is very robust to illumination changes since the local region will have the same lighting condition. We lower the pixel intensity value of the original image, and we can see the LBP image in Fig. 7.

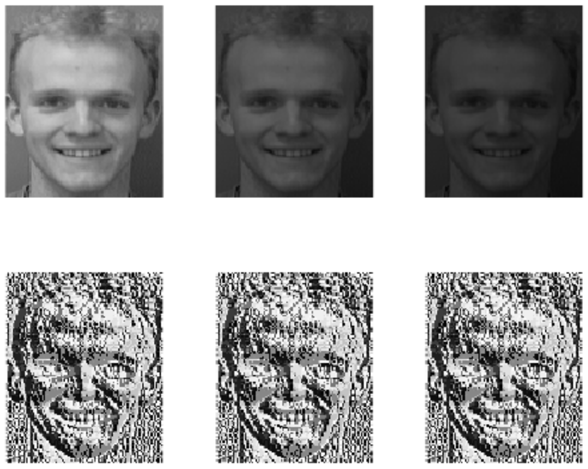


Figure 7: Top: Input images with various lighting conditions. Bottom: The corresponding LBP features.

The basic LBP operator has a size of 3×3 pixels. In this case, we have 8-bit data which contains 256 combinations, and this is called LBP feature. To extend this idea to larger operator sizes, the dimension, i.e., the combinations, of the data will grow exponentially. The authors define an LBP feature as uniform when it contains at most two bit-wise transitions, otherwise, it is defined as circular. By setting the LBP feature this way, we can reduce the dimension from 2^P to $P(P-1)+2$. For the same operator that has the size of 3×3, we reduce features from 256 to 58. Also, the texture feature of images depends on the size of LBP operator.

The authors record all circular patterns as one category and record every combination in a uniform pattern. The reason for doing this is that a uniform pattern contains 90% of useful patterns. Next, the authors define the histogram based on the LBP features such that

$$H_i = \sum_{x,y} I\{f_l(x,y) = i\}, \quad i = 0, ...n-1 \quad , \tag{12}$$

where n is the number of different labels produced by the LBP operator and

$$I(A) = \begin{cases} 1 & A \text{ is true} \quad , \\ 0 & A \text{ is false} \quad . \end{cases} \tag{13}$$

In the facial expression recognition problem, our faces might have different spatial positions. However, we find that finding LBP features for different facial expressions yield no visual differences in our histogram of 59 dimension features. We decided to modify the LBP to make it more robust to spatial changes by dividing an image into different regions. For example, in Fig. 8, instead of having a global LBP histogram, Ahonen [3] uses local LBP histograms. We split 48×48 image into 6×6 blocks, so that each block contains 8×8 pixels. We extract an LBP feature for each block instead of the whole image. Then we concatenate all the histograms to form a whole to obtain a new feature histogram and obtain $6 \times 6 \times 59$=2124 dimension features. It is a local LBP in a sense, since we are fixing local region, yet it contains more spatial information than global LBP features. Also, we can integrate local LBP histogram to obtain global LBP histogram.

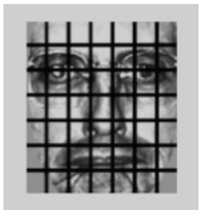


Figure 8: divide an image into blocks

5.1.2 Gabor

Drawbacks of LBP due to orientation can be solved by Gabor filters. Dennis Gabor invented the Gabor filter [17]. A Gabor filter is nothing more than a product of a sinusoidal function and a Gaussian function. This is the equation for a 2D Gabor filter:

$$g(x,y;\lambda,\theta,\phi,\gamma) = \exp\left(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right)\cos\left(2\pi\frac{x'}{\lambda}+\phi\right) \quad , \tag{14}$$

where $x' = x\cos\theta + y\sin\theta$; and $y' = -x\sin\theta + y\cos\theta$. The Gaussian serves as a localized "window." The frequency response is two Gaussian pulse corresponding to the modulated sinusoid. We can visualize these Gabor filters in Fig. 9.

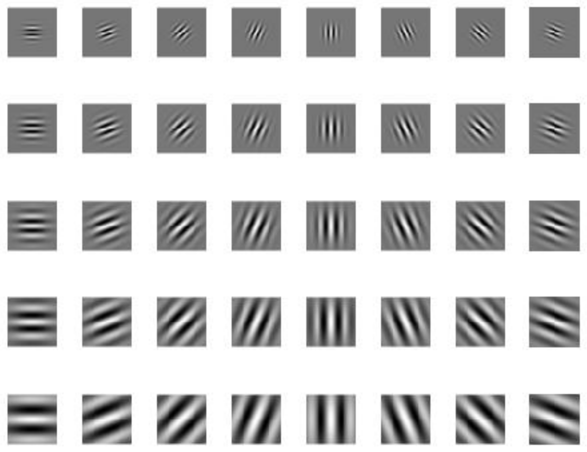


Figure 9: Gabor features

The wavelength λ, aspect ratio γ, and orientation θ govern width, height, and rotation of the Gabor filter, respectively. ϕ is the phase offset. In the above equation, the value of σ is determined by λ and bandwidth b. For a large bandwidth, the envelope increases allowing more stripes appear in the filter. With the small bandwidth, the envelope tightens. We estimate σ as follows,

$$\sigma = \frac{\lambda}{\pi}\sqrt{\frac{\ln 2}{2}\cdot\frac{2^b+1}{2^b-1}} \quad . \tag{15}$$

In Fig. 9, we fix the bandwidth and vary the rest of the parameter to create five different scales and eight different orientations Gabor filters.

6 Classification

6.1 Support Vector Machine

Boser [5] and Cortes [12] initially proposed the idea of support vector machine (SVM). After decades of development of SVM algorithms, we now have hard margin SVM, soft margin SVM and kernel SVM. SVM is famous because it has a strong mathematical foundation and outstanding classification performance. Hard margin SVM can classify the linearly separable problem and cannot tolerate error. Soft margin SVM allows errors to occur at the wrong side of decision boundary, but we always want to minimize error. Kernel SVM is more suitable to handle high dimensional data.

In short, SVM is trying to find a decision boundary that separates two classes, maximizing the distance between the decision boundary and the closest points from each category. The decision boundary is the position that has the maximum uncertainty. We consider the decision boundary as a hyperplane, and there is an infinite number of hyperplanes that can separate different classes of data. We need to find the "best" hyperplane that has the maximum margin. Here, the definition of margin is the distance between the hyperplane and the closest point. SVM is very robust in the two-class problem, also called the one vs. one problem. For multiclass classification problem, also called the one vs. all problem, we can pick one class and assign the rest data as the other class. By doing this, we transfer one vs. all problem to one vs. one problem.

Assume that we can represent the hyperplane in $\mathbb{R}^n$ in the form $\beta^T x+\beta_0 = 0$, where $x, \beta, \beta_0 \in \mathbb{R}^n$. Let the label of the data y_i be either -1 or 1. We then assume $d_i = \beta^T x_i + \beta_0$ and classify on data based on $sign(d_i)$. Let the margin be the distance between hyperplane and the closest point which is equal to $y_i d_i$. Since y_i and d_i have the same sign, their product is always positive. Our goal is to maximize the margin. Next, we normalize d_i as follows,

$$d_i = \frac{\beta^T x_i + \beta_0}{|| \beta ||} \quad . \tag{16}$$

If the point is on the hyperplane $d_i = 0$, otherwise $d_i > 0$. We have the following objective function,

$$\max\left(\min\left(\frac{y_i(\beta^T x_i + \beta_0)}{|| \beta ||}\right)\right) \quad . \tag{17}$$

Assume that we do not have point on the hyperplane, in which case $y_i(\beta^T x_i + \beta_0) > 0$, and $y_i(\beta^T x_i + \beta_0) \geq C$ for some positive C. Now assume that $\beta = \beta/C$ and $\beta_0 = \beta_0/C$. Hence, $y_i(\beta^T x_i + \beta_0) \geq 1$. Therefore, our objective function can be expressed in the following form,

$$\max\left(\frac{1}{|| \beta ||}\right) \quad . \tag{18}$$

We can recast the objective function into L^2 norm form as below,

$$\begin{cases} \min(\frac{1}{2} \mid \beta \mid^2) \\ y_i(\beta^T x_i + \beta_0) \geq 1 \quad . \end{cases} \tag{19}$$

We can solve above optimization problem using the following Lagrangian function,

$$\begin{aligned} L(\beta, \beta_0, \alpha) &= \frac{1}{2} \mid \beta \mid^2 - \sum_{i=1}^{n} \alpha_i [y_i(\beta^T x_i + \beta_0) - 1] \\ &= \frac{1}{2} \mid \beta \mid^2 - \beta^T \sum_{i=1}^{n} \alpha_i y_i x_i - \sum_{i=1}^{n} \alpha_i y_i \beta_0 - \sum_{i=1}^{n} \alpha_i \quad , \end{aligned} \tag{20}$$

where α is the Lagrange multiplier. Taking the derivatives of L with respect to β and β_0 and setting them equal to 0 yields the following equations,

$$\beta = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{21}$$

and

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \quad . \tag{22}$$

We substitute the results into the Lagrangian and obtain the following:

$$L(\beta, \beta_0, \alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \quad . \tag{23}$$

subject to $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$. We can use quadprog function in Matlab to solve the quadratic problem in Eq. (22), Eq. (23) and Eq. (24). The critical step here is to select the support vector to create our classifier. In order for this problem to have a solution, the so-called Karush–Kuhn–Tucker (KKT) optimization conditions must be satisfied. These include stationary, dual feasibility, complementary slackness, and primal feasibility. We are concerned about complementary slackness condition so that we want

$$\alpha_i [y_i(\beta^T x_i + \beta_0) - 1] = 0 \tag{24}$$

and we determine the support vector by satisfying

$$y_i(\beta^T x_i + \beta_0) = 1 \quad , \tag{25}$$

where $\alpha_i > 0$. We then substitute Eq. (22) into

$$y_i(\beta^T x_i + \beta_0) = 1 \tag{26}$$

in order to find β_0.

6.2 Multilayer Perceptron

Rumelhart [29] gave the structure of multilayer perceptron (MLP). The structure of MLP has at least three layers, one input, one output and at least one hidden layer, as illustrated in Fig. 10.

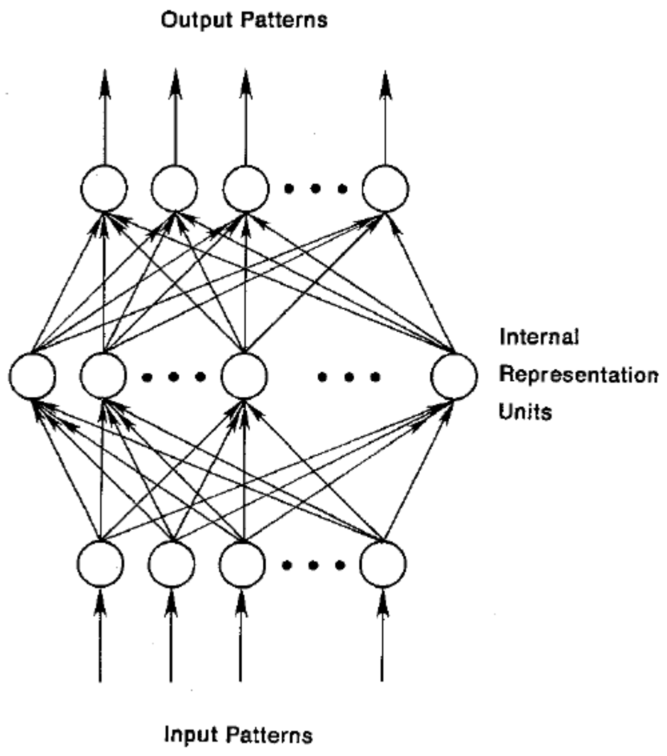


Figure 10: MLP

The number of neurons in the input layer is equivalent to the dimension of inputs. There is nothing in the neuron in input layer but the inputs themselves. In the hidden layers, neurons are fully connected with the neuron from input layer with a set of weights $w1$ and a set of associated biases $b1$. Here, fully-connected means that we have weights between a previous layer's neurons and the next layer's neurons. The number of hidden layers and neurons in hidden layers are hyper-parameters. Suppose we have input vector X, the input of the first hidden layer is output from the input layer which is X as well. The neuron's structure starting in hidden layers are different than the previous neuron from input layer. These neurons have two parts. First of all, they linearly transform the input X with weight and the bias factor. Next, each neuron contains an activation function f such as rectified linear unit (ReLu), sigmoid and tanh.

Hence, we can write the output of the first hidden layer as $f(w_1X + b_1)$, where $w_1 \in w1$ and $b_1 \in b1$.

Finally, after hidden layer, output layer is fully connected to hidden layer as well using $w2$ and $b2$, where $w_2 \in w2$ and $b_2 \in b2$. The number of neurons in the layer is equivalent to the number of classes we have. The structure of neuron in this layer is the same as them in the hidden layer, except for using softmax, denoted as g, which is the activation function. The input of output layer is output of hidden layer $f(w_1X + b_1)$, and the final output is $g(w_2f(w_1X + b_1) + b_2)$.

We have an optimization problem so that we want to find the appropriate w_i and b_i for the classification. We call this process "learning." We use least squared error to measure the loss between our corrected label and predicted label. We can think of it as the following objective function,

$$J(w) = \frac{1}{2}\sum_i (y^i_{correct} - y^i_{predict})^2 \quad . \tag{27}$$

where $y^i_{predict} = g(w_2f(w_1X_i + b_1) + b_2)$ and $y^i_{correct}$ is the label of the training data. We can solve this optimization problem by gradient decent. We randomize all the initial weights and biases and update them using the Chain Rule.

7 Deep neural network

7.1 Related work

Ng [26] fine-tuned AlexNet and VGGNet using FER2013 data set and EmotiW dataset. Tang [32] replaces softmax activation function on the last layer with support vector machine. Liu and Yu [22, 38] use convolutional neural network (CNN) to extract feature and classify emotion.

7.2 Convolutional Neural Network

In a previous section, we discussed the simple structure of neural networks such as the MLP. More recently CNN is one of many structures that perform well in image classification. CNN has great potential to capture the essential features of image inputs. What makes all the pixels in an image meaningful depends on the pattern and order of arrangement. For example, the human face has various objects, such as eyes, lips, nose, etc. Therefore, if the spatial positions of these complex object are different, conventional neural networks will fail to classify the classes since they assign one neuron to each pixel. As a result, the spatial relationships of inputs are not preserved. On the other hand, CNN can capture some level of spatial relationships. This capability is given by the kernel. A larger size of a kernel has more power to capture the spatial relationship but less ability to capture the local details of objects. We will start by observing the structure of CNN in Fig. 11.

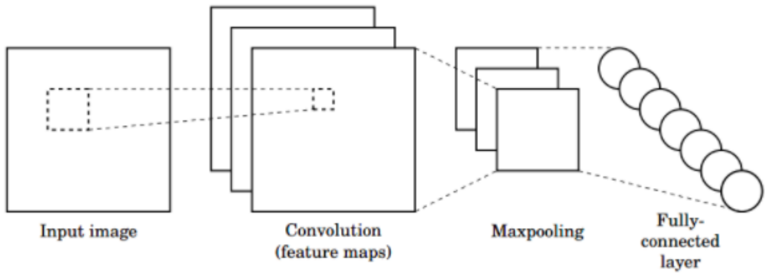


Figure 11: CNN's structure [2]

We will not discuss the fully connected layer here since it is somehow similar to MLP. MLP has the same structure as CNN except that MLP does not have convolutional layers. We will denote the convolutional layer as Conv. All layers are fully connected just like MLP. Good initialization can produce faster convergence and has a lower chance of overfitting. Overfitting means that our model is complex enough to classify training data correctly, but this model is too complex to perform well on test data. Usually, we want to use kernel sizes of 3×3 or 5×5. Of course, we need $3 \times 3 \times 3$ or $5 \times 5 \times 5$ for an RGB image. More neurons in Conv can capture more features, and we can pass through the feature to the next Conv layer. To calculate the output of Conv, we will need to introduce two important concepts, padding and stride. Assume that W and O are the height/length of input and height/length of output, respectively, K is the kernel size, P is the padding size, and S is the stride. We can express the output size of Conv as follows:

$$O = \frac{(W - K + 2P)}{S} + 1 \quad . \tag{28}$$

Fig. 12 shows output after we apply the trained kernel to the input data. The first layer Conv can capture the texture feature of input, and the second layer representation is good enough to detect eye, mouth, and other organs.

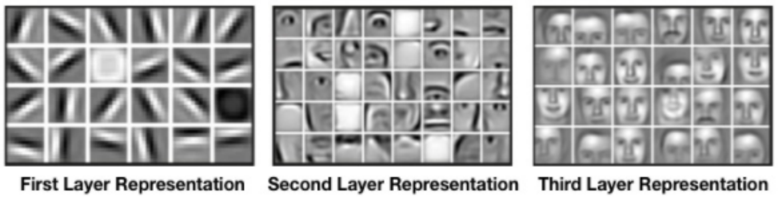

First Layer Representation
Second Layer Representation
Third Layer Representation

Figure 12: CNN's features [2]

The number of parameters of a neural network dataset can easily go above a million. Thanks to today's technology, we have enough computational power to easily calculate mathematical operations in a neural network. Sometimes, however, we still want to save time in order to improve training efficiency. We can do this by using 2×2 polling layer to downsample the size of input to a quarter for a binary image. There are max pooling and average pooling, two different kinds of pooling layers. Max pooling selects the max pixel value, and average pooling takes the average in a window that has a size of 2×2.

8 Experiment

As mentioned earlier, there are 28709 input images in the FER2013 dataset. We decided to divide them into 90% training data and 10% validation data. Our results might be slightly different than the scores on the leaderboard since the competitors in the challenge had 3589 more images to train during public submission, and they submitted the result for the other 3589 images in private test set. In other words, we can achieve at best the same result as the competitors since we have fewer data to train. The purpose here, however, is only to illustrate our method. Therefore, we are not going to apply any data augmentation such as increasing our training data by flipping, rotating, or adding noise to the training images.

We want to try three different features for this classification task. First of all, CNN is a feature extraction method as well as a classification method. It extracts features of an image using the convolutional layers and processes the feature to fully connected layers. The fully connected layers can be used for classification, as is done by MLP. However, CNN has its advantages and disadvantages.

Traditionally, neural networks have some issues include overfitting, vanishing gradient and easily stuck at the local minimum during backpropagation. The concept of overfitting was discussed in Section 7. Conventional methods that prevent overfitting include weight decay method, add noises to training data, early stopping method and add dropout layers. Vanishing gradient means that the parameters stop updating due to extremely small changes calculated by Chain Rule. This will stop the system from further training. We can reduce the effect of vanishing gradient by replacing sigmoid with ReLu. Also, to avoid getting stuck at the local minimum, we can train the network by first using relatively large learning rate and then reduce it. However, extracting CNN features from the deep neural network is quite difficult since we are using Keras and its codes are highly encapsulated. Therefore we are not going to extract CNN feature and apply it on SVM. By using CNN, we use batch size 128, 20 epochs, 2 Conv layers with 32 and 64 kernel with ReLu, respectively, one max pooling layer with 0.25 dropout after it, and stochastic gradient descent with 0.1 learning rate. For the FC layers, we use 128 neurons with 0.5 dropouts. It is worth mentioning that we are going to use the same structure as our MLP for Gabor and LBP to make the results comparable.

Next, we are going to implement LBP and Gabor with the different classifiers. SVM has different kernel methods such as linear, radial basis function (RBF), poly and sigmoid. We will also try MLP using LBP and Gabor features as well. Combining two features has potential to achieve higher accuracy. In Table 1 are shown the results of our experiments with various strategies.

Table 1: experiment results

methods	training accuracy	validation accuracy
$LBP + SVM_{linear}$	0.6898	0.4433
$LBP + SVM_{rbf}$	0.3577	0.3448
$LBP + SVM_{poly}$	0.2518	0.2466
$LBP + SVM_{sigmoid}$	0.2518	0.2466
$Gabor + SVM_{linear}$	0.9978	0.4284
$Gabor + SVM_{rbf}$	0.7968	0.5426
$Gabor + SVM_{poly}$	0.9488	0.5806
$Gabor + SVM_{sigmoid}$	0.2518	0.2466
CNN	0.7175	0.5047
$LBP + MLP$	0.5862	0.4928
$Gabor + MLP$	0.6697	0.5238

Figures 13, 14 and 15 show the accuracy curves of training data and validation data of CNN, LBP+MLP, and Gabor+MLP, respectively. From above table, we calculate the training accuracy by dividing the number of successful classification of training data by the numbers of training data. On the other hand, we calculate the validation accuracy as calculating the training accuracy but using the numbers of successful classification of validation data and the numbers of validation data instead. We should pay more attention to the validation accuracy since training accuracy only tell us how likely this method would be overfitting.

SVM in Sklearn package in Python does not provide the training and validation accuracy update history, and therefore, we only show the final accuracy in the table. For anything that uses the neural network, we can plot the accuracy versus epoch figure as above. The number of epoch means the number of times that we update our prediction model. Plus, to replicate our result, as mentioned earlier, we are using MLP with 128 neurons with 0.5 dropouts as we did in CNN. Also, we are picking the best validation accuracy in 20 epochs' training.

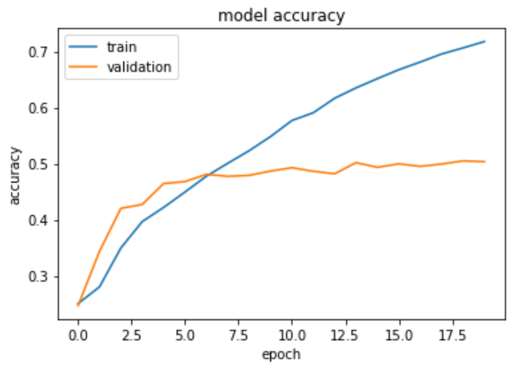


Figure 13: CNN plot

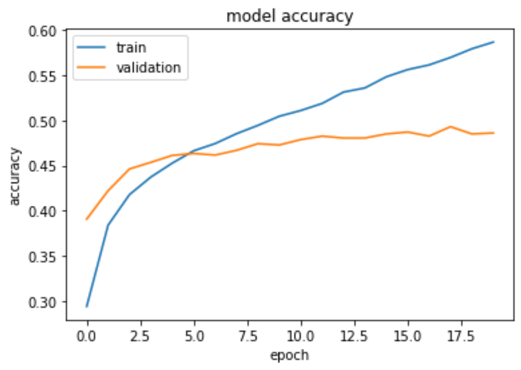


Figure 14: LBP+MLP plot

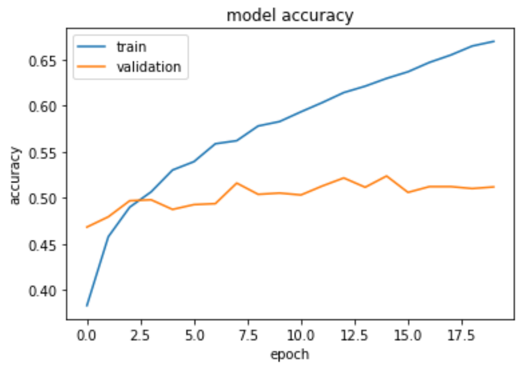


Figure 15: Gabor+MLP plot

As we can observe from the table, overfitting appears almost in every column in our result. So far, Gabor and SVM with poly kernel achieve the best score at 58.06% and Gabor feature, in general, perform better than LBP feature. There is no absolute answer that which is better with classification, SVM or MLP. However, SVM has a strong mathematical foundation while easy to use. MLP, on the other hand, requires a lot of time for training and fine-tuning techniques to achieve the optimal result.

9 Conclusion

Our experimental results on different methods are tabulated in the table above. It can be seen that for the dataset under consideration, Gabor feature and CNN feature perform in better describing facial expression. Gabor with SVM is a bit more accurate. However, by fine-tuning the CNN, it has potential to surpass other methods. On a side note, what exactly is happening in this process is that people are trying to develop a mathematical form to represent and identify human emotions!

References

[1] Challenges in Representation Learning: Facial Expression Recognition Challenge, howpublished = https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data, note = Accessed: 2018-01-26.

[2] Introduction to convolutional neural networks. https://www.vaetas.cz/blog/intro-convolutional-neural-networks/. Accessed: 2017-12-07.

[3] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. Face recognition with local binary patterns. *Computer vision-eccv 2004*, pages 469–481, 2004.

[4] Marian Stewart Bartlett, Gwen Littlewort, Ian Fasel, and Javier R Movellan. Real time face detection and facial expression recognition: Development and applications to human computer interaction. In *Computer Vision and Pattern Recognition Workshop, 2003. CVPRW'03. Conference on*, volume 5, pages 53–53. IEEE, 2003.

[5] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[6] Hayet Boughrara, Mohamed Chtourou, Chokri Ben Amar, and Liming Chen. Facial expression recognition based on a mlp neural network using constructive training algorithm. *Multimedia Tools and Applications*, 75(2):709–731, 2016.

[7] Adrian Bulat and Georgios Tzimiropoulos. How far are we from solving the 2d & 3d face alignment problem?(and a dataset of 230,000 3d facial landmarks). *arXiv preprint arXiv:1703.07332*, 2017.

[8] Xudong Cao, Yichen Wei, Fang Wen, and Jian Sun. Face alignment

by explicit shape regression. *International Journal of Computer Vision*, 107(2):177–190, 2014.

[9] Wikipedia contributors. Summed-area table, 2016. [Online; accessed 30-January-2018].

[10] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active appearance models. *IEEE Transactions on pattern analysis and machine intelligence*, 23(6):681–685, 2001.

[11] Timothy F Cootes, Christopher J Taylor, David H Cooper, and Jim Graham. Active shape models-their training and application. *Computer vision and image understanding*, 61(1):38–59, 1995.

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[13] David Cristinacce and Timothy F Cootes. Feature detection and tracking with constrained local models. In *BMVC*, volume 1, page 3, 2006.

[14] David Cristinacce and Timothy F Cootes. Boosted regression active shape models. In *BMVC*, volume 2, pages 880–889, 2007.

[15] Piotr Dollár, Peter Welinder, and Pietro Perona. Cascaded pose regression. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1078–1085. IEEE, 2010.

[16] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[17] Dennis Gabor. Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, 93(26):429–441, 1946.

[18] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. In *Proc. 1st Int. Conf. on Computer Vision*, volume 259, page 268, 1987.

[19] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1867–1874, 2014.

[20] Irene Kotsia and Ioannis Pitas. Facial expression recognition in image sequences using geometric deformation features and support vector machines. *IEEE transactions on image processing*, 16(1):172–187, 2007.

[21] Martin Lades, Jan C Vorbruggen, Joachim Buhmann, Jörg Lange, Christoph Von Der Malsburg, Rolf P Wurtz, and Wolfgang Konen. Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on computers*, 42(3):300–311, 1993.

[22] Kuang Liu, Mingmin Zhang, and Zhigeng Pan. Facial expression recognition with cnn ensemble. In *Cyberworlds (CW), 2016 International Conference on*, pages 163–166. IEEE, 2016.

[23] Mario Martínez Zarzuela, Francisco Díaz-Pernas, Miriam Antón-Rodríguez, Freddy Perozo, and David González Ortega. Adaboost face detection on the gpu using haar-like features. pages 333–342, 01 2011.

[24] Ali Mollahosseini, David Chan, and Mohammad H Mahoor. Going deeper in facial expression recognition using deep neural networks. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*, pages 1–10. IEEE, 2016.

[25] Ali Mollahosseini and Moohammad Mahoor. Bidirectional warping of active appearance model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 875–880, 2013.

[26] Hong-Wei Ng, Viet Dung Nguyen, Vassilios Vonikakis, and Stefan Winkler. Deep learning for emotion recognition on small datasets using transfer

learning. In *Proceedings of the 2015 ACM on international conference on multimodal interaction*, pages 443–449. ACM, 2015.

[27] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.

[28] Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. Face alignment at 3000 fps via regressing local binary features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1685–1692, 2014.

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[30] Caifeng Shan, Shaogang Gong, and Peter W McOwan. Robust facial expression recognition using local binary patterns. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 2, pages II–370. IEEE, 2005.

[31] Caifeng Shan, Shaogang Gong, and Peter W McOwan. Facial expression recognition based on local binary patterns: A comprehensive study. *Image and Vision Computing*, 27(6):803–816, 2009.

[32] Yichuan Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

[33] Ying-li Tian. Evaluation of face resolution for expression analysis. In *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*, pages 82–82. IEEE, 2004.

[34] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[35] Jun Wang, Lijun Yin, Xiaozhou Wei, and Yi Sun. 3d facial expression recognition based on primitive surface feature distribution. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1399–1406. IEEE, 2006.

[36] Laurenz Wiskott, Norbert Krüger, N Kuiger, and Christoph Von Der Malsburg. Face recognition by elastic bunch graph matching. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):775–779, 1997.

[37] Xuehan Xiong and Fernando De la Torre. Supervised descent method and its applications to face alignment. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 532–539, 2013.

[38] Zhiding Yu and Cha Zhang. Image based static facial expression recognition with multiple deep network learning. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pages 435–442. ACM, 2015.

[39] Baochang Zhang, Shiguang Shan, Xilin Chen, and Wen Gao. Histogram of gabor phase patterns (hgpp): A novel object representation approach for face recognition. *IEEE Transactions on Image Processing*, 16(1):57–68, 2007.

[40] Zhengyou Zhang, Michael Lyons, Michael Schuster, and Shigeru Akamatsu. Comparison between geometry-based and gabor-wavelets-based facial expression recognition using multi-layer perceptron. In *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, pages 454–459. IEEE, 1998.

[41] Xiangxin Zhu and Deva Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2879–2886. IEEE, 2012.