Exploration of Multiple Methods in Solving Facial Expression Recognition Problems
Statistics 561
Sayan Mukherjee

Zichen Miao
Zhi Qiu
Xiaoyan Liu
Zhihe Zhao
Yuhua Cai

May 3, 2020

Abstract

Facial expression analysis is an interesting and challenging problem. It attracts many researchers these years and has been applied in some different fields. In this project, we attempt to examine three machine learning methods including Support Vector Machine Classifier, CNN and GNN on two datasets, FER and CK+. Experiments illustrate that SVM and CNN methods can achieve a good performance. But we try to apply GNN in this task and obtain experimental results. This is our point of the project.

1 Introduction

Facial expression is one of the most powerful, natural and immediate means for human beings to communicate their emotions and intensions. Automatic facial expression analysis is an interesting and challenging problem, and impacts important applications in many areas such as human–computer interaction and data-driven animation[1]. Since it is useful and can be applied in a wide range of fields, much attention has been attracted.

In this project, we use three different methods to explore the implementation of facial expression recognition. The first method is the traditional method: Support vector machine classifier. It is not complicated, just two stages: feature extraction and classification. But the result is not good enough. It achieves a relatively low accuracy. The second method is a normal method, CNN. It has been applied by many researchers[2] and proved to be a high-accuracy way. Last but not least, we choose to use GNN implemented by ourselves, and it still achieves a good result.

2 Explanation of Data

The first dataset is FER 2013, The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The task is to categorize each face based on the emotion shown in the facial expression in to one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).

Train.csv contains two columns, "emotion" and "pixels". The "emotion" column contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image. The "pixels" column contains a string surrounded in quotes for each image. The contents of this string a space-separated pixel values in row major order. test.csv contains only the "pixels" column and your task is to predict the emotion column.

Training set consists of 28,709 examples. The public test set used for the leaderboard consists of 3,589 examples. The final test set, which was used to determine the winner of the competition, consists of another 3,589 examples.

This dataset was prepared by Pierre-Luc Carrier and Aaron Courville, as part of an ongoing research project. They have graciously provided the workshop organizers with a preliminary version of their dataset to use for this contest.

The second dataset is CK+ dataset is an extension of the CK dataset. It contains 327 labeled facial videos, We extracted the last three frames from each sequence in the CK+ dataset, which contains a total of 981 facial expressions. we use 10-fold Cross validation in the extended[3].

3 Solutions

3.1 Solution One: Support Vector Machine Classifier

This part introduces the first pipeline we used in the project. The pipeline we design consists of two stages: Feature extraction and Classification.

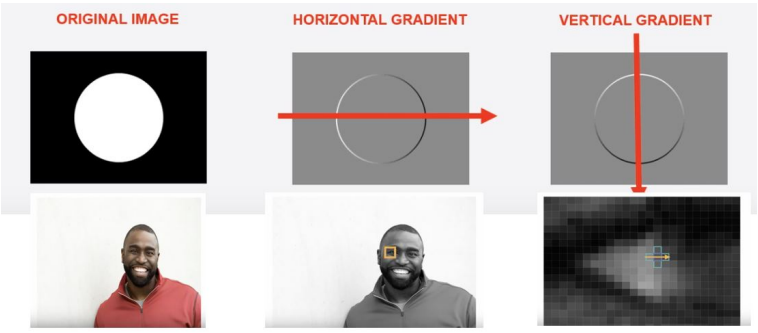

Figure 1: Illustration and an Example of HOG

On the feature extraction stage, histogram of oriented gradient (HOG) [4] is applied as a feature extraction method. Figure.1 is an example of HOG [5]. There are 7 steps for this process which is

shown as below.

1) Grayscale (take the image as a three-dimensional image of x, y, z (grayscale));
2) Gamma correction method is used to normalize (normalize) the color space of the input image; the purpose is to adjust the contrast of the image, reduce the impact of local shadows and changes in lighting, and suppress noise interference;
3) Calculate the gradient (including size and direction) of each pixel of the image; mainly to capture the contour information while further weakening the interference of lighting.
4) Divide the image into small cells (eg. 6 * 6 pixels / cell);
5) Count the gradient histogram of each cell (the number of different gradients) to form the descriptor of each cell;
6) Each several cells form a block (for example, 3 * 3 cells / block), and the feature descriptors of all cells in a block are connected in series to obtain the HOG feature descriptor of the block.
7) The HOG feature descriptors of the image (the target you want to detect) can be obtained by concatenating the HOG feature descriptors of all blocks in the image image. This is the final feature vector available for classification.

After retrieving the extracted features, we feed these vectors into a SVM based classifier.SVM (Support Vector Machine, support vector machine) [6] is a two-class classification model, the basic model is defined as the largest linear classifier in the feature space, the learning strategy of the device is to maximize the interval, and finally it can be transformed into a convex Solve the problem of secondary planning. (Linear support vector machine, nonlinear support vector machine).

The main idea of SVM is to establish a hyper-plane as a decision surface, and the isolation edge between the positive and negative examples is maximized. For a two-dimensional linearly separable case, let H be the classification county that separates the two types of training samples without error, and H1 and H2 are the straight lines parallel to the classification line that have passed the samples closest to the classification line in each type The distance to the lecture classification interval. The so-called optimal classification line requires that the classification line can not only correctly separate the two types, but also maximize the classification interval. In a high-dimensional space, the optimal classification line becomes the optimal classification line.

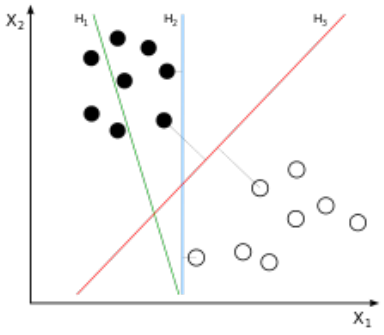

Figure 2: Illustration of SVM

As is illustrated in Figure.2, the feature vector of the instance (take two-dimensional as an example) is mapped to some points in space, which are solid points and hollow points as shown in the figure below, which belong to two different categories. Then the purpose of SVM is to draw a line to distinguish the two types of points "best", so that if there are new points in the future, this line can also make a good classification.
Given a training set of instance-label pairs (x_i, y_i), i = 1..., l where $x_i \in R^n$ and $y \in \{1, -1\}^l$, the support vector machines require the solution of the following optimization problem: In this project,

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned}$$

the SVM+HOG method is just for learning and as baselines, as expected, the Deep Learning based approaches achieve better results which are shown as below parts.

3.2 Solution Two: CNN

CNNs are a class of neural networks that have convolutional layers, which are particularly effective for data that have spatial structures and correlations (e.g. images). It owns the advantage of having fewer parameters to be trained than fully connected networks with the same number of hidden units and thus is worth exploring with regard to the two facial expression data sets. With regard to a convolutional layer, it is similar to a multilayer perceptron (MLP) entirely composing of fully connected layers, which are each a matrix multiply operation (and addition of a bias) followed by a non-linearity such as sigmoid and ReLU. In our CNN model, ReLU is used as the activation function to impose non-linearities. The only difference between MLP and CNN is that the matrix multiply operation is replaced with a convolution operation (in practice a cross-correlation) in CNNs (MLP is a special case of CNN). Moreover, CNNs need not to be entirely composed of convolutional layers and many popular CNN architectures end with fully connected layers.

1) Fully connected layer: In a fully connected layer, the input $x \in \mathbb{R}^{M \times C_{in}}$ is a vector (or a batch of vectors), where M is the minibatch size and C_{in} is the dimensionality of the input. First the input x is multiplied by a weight matrix W and this weight matrix has dimensions $W \in \mathbb{R}^{C_{in} \times C_{out}}$, where C_{out} is the number of output units. Then a bias is added for each output, denoted by $b \in \mathbb{R}^{C_{out}}$. The output $y \in \mathbb{R}^{M \times C_{out}}$ of the fully connected layer is then:

$$y = \text{ReLU}(xW + b)$$

The values of W and b are variables to be learned.

2) Convolutional layer: In a convolutional layer, we convolve the input x with a convolutional kernel (filter), which we also call W, producing output y:

$$y = \text{ReLU}(W * x + b)$$

In the context of CNNs, the output y is often referred to as feature maps. As with a fully connected layer, the goal is to learn W and b. Unlike the input of a fully connected layer, which is $x \in \mathbb{R}^{M \times C_{in}}$, the dimensionality of an image input is 4D: $x \in \mathbb{R}^{M \times C_{in} \times H_{in} \times W_{in}}$, where M is still the batch size, C_{in} is the number of channels of the input (e.g. 3 for RGB), and H_{in} and W_{in} are the height and width of the image. The weight parameter W is also different in a convolutional layer. Unlike the 2-D weight matrix for fully connected layers, the kernel is 4-D with dimensions $W \in \mathbb{R}^{C_{out} \times C_{in} \times H_K \times W_K}$, where H_K and W_K are the kernel height and weight. Values of H_K and W_K is $H_K = W_K = 3$ or 5 vary in different architectures. Convolving the input with the kernel and adding a bias then gives an output $y \in \mathbb{R}^{M \times C_{out} \times H_{out} \times W_{out}}$. If using "same" padding and a stride of 1 in convolution, the output will have the same spatial dimensions as the input: $H_{out} = H_{in}$ and $W_{out} = W_{in}$. For a single member of the minibatch, one convolutional kernel at a time, if considering a stack of C_{out} number of kernels, each of which are 3D ($C_{in} \times H_K \times W_K$), the 3D volume is then slid across the input (which is also 3D: $C_{in} \times H_{in} \times W_{in}$) in the two spatial dimensions (along H_{in} and W_{in}). The outputs of the multiplication of the kernel and the input at every location creates a single feature map that is $H_{out} \times W_{out}$. Stacking the feature maps generated by each kernel gives the 3D output $C_{out} \times H_{out} \times W_{out}$. If repeat this process for all M inputs in the minibatch, a 4D output $M \times C_{out} \times H_{out} \times W_{out}$ will be achieved.

3) Pooling or striding: Pooling and striding are important components in CNN architectures, and they are included for the consideration of dimensionality reduction, translational invariance and receptive field increasement. The two most common forms of pooling are max pooling and average pooling. Both reduce values within a window to a single value, on a per-feature-map basis. Max pooling takes the maximum value of the window as the output value while average pooling takes the mean. For our model, max pooling and average pooling are combined together to build the neural network. With regard to striding, a CNN slides across the input one pixel at a time (stride of 1). It is done to save computation by skipping positions while sliding the convolutional kernel as pixels in an image tend to have high correlation with neighboring pixels. For our model, we are using a stride of 2, skipping calculating 75% of the values of the output feature map, which yields a feature map that's half the size in each spatial direction. Besides, it is worth noticing that while pooling is an operation done after the convolution, striding is part of the convolution operation itself.

3.3 Solution Three: GNN

Though CNN methods have achieve superior performance on FER task, we still want to explore the performance of a new kind of neural network, the Graph Neural Network(GNN). Here, we follow the spectral graph convolution scheme, which includes two popular GNN's: GCN[7] and ChebyNet[8]. View the graph spectral convolutions as the multiplication of the input signal $x \in \mathbb{R}^N$ with convolution kernel $k_\theta = diag(\theta)$ parameterized by $\theta \in \mathbb{R}^N$ in the Graph Fourier domain:

$$x \times k_\theta = U k_\theta U^T x \tag{1}$$

where U is the matrix of eigenvectors of the symmetric normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T$. Then, diagonal components of Λ are eigenvalues of L and $U^T x$ is the graph fourier transform of x. Also, the k_θ as a function of eigenvalues of L, i.e. $k_\theta(\Lambda)$. However, the Eq.1 is computationally expensive, i.e. the eigendecomposion of L and the eigenvector matrix multiplication. To alleviate this problem, Hammond et al.[9] suggested that $k_\theta(\Lambda)$ can be well

approximated by a truncated expansion of Chebyshev polynomials $T_k(x)$ up to $K^t h$ order:

$$k'_\theta(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) \tag{2}$$

where $\tilde{\Lambda}$ is the rescaled, normalized version of λ. $\theta' \in \mathbb{R}^K$ is the Chebyshev coefficients. And polynomials are defined recursively as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0(x) = 1, T_1(x) = x$. Then, the convolution can be approximated by

$$x \times k_\theta \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \tag{3}$$

in which $\tilde{L} = U\tilde{\Lambda}U^T$. This equation is the main equation for ChebNet[8]. Let $K = 1$, the equation above becomes the main equation for GCN[7]:

$$x \times k_\theta \approx \theta(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})x \tag{4}$$

The input to the GNN are facial landmarks and features around them. To be specific, I use deepleanring-based facial aligner[10] to perform face alignment as a type of data normalization, and then extract landmarks on normalized face images. Not all commonly used 68 facial landmarks are necessary for our task. Instead, we choose 15 critical landmark points from them, and crop a 10×10 pixel patch around each point as the input feature. Illustration of facial alignment on FER2013 dataset is shown below.

Figure 3: Illustration of face alignment on FER13

Graph, i.e. the adjacency matrix A built for these 15 landmarks are based on nearest neighbor rule.

4 Experimental Results

Based on FE2R013 and CK+ dataset, there are three models are designed and trained. In FER13, The training set consists of 28709 images ranging from different face emotions, the testing dataset consists 3589 of them. In CK+, there are only about 9 hundred images, so 5-fold cross validation is performed.

4.1 SVM based Classifier

HoG features are extracted according to description above. The final results are shown as below. Training time and evaluation time are also reported.

Dataset	Train Accuracy	Test Accuracy	Training Time	Evaluation Time
FE2R013	28.1	27.4	200.1	23.6

Dataset	Train Accuracy	Test Accuracy	Training Time	Evaluation Time
CK+	38.1	41.9	123	13

4.2 CNN

For CNN, both results of VGG19 model and Resnet18 model on two datasets are shown below.

Models	Test Accuracy
VGG19	73.96
Resnet18	72.61

Table 1: Results of VGG19 and Resnet18 on FER13

Models	Test Accuracy
VGG19	94.76
Resnet18	93.13

Table 2: Results of VGG19 and Resnet18 on CK+

4.3 GNN

Both results of GCN[7] and ChebNet[8] on two datasets are shown below.

Models	Test Accuracy
GCN	60.1
ChebNet	63.2

Table 3: Results of GCN and ChebNet on FER13

Models	Test Accuracy
GCN	90.6
ChebNet	93.4

Table 4: Results of GCN and ChebNet on CK+

5 Conclusions

In this project, we try three different methods on two popular facial expression recognition datasets. The results show that CNN performs best. But the results of GNNs are relatively comparable. The traditional HOG+SVM method still has a lot of potential to improve.

References

[1] C. Shan, S. Gong, and P. W. McOwan, "Facial expression recognition based on local binary patterns: A comprehensive study," *Image and vision Computing*, vol. 27, no. 6, pp. 803–816, 2009.

[2] Z. Yu and C. Zhang, "Image based static facial expression recognition with multiple deep network learning," in *Proceedings of the 2015 ACM on international conference on multimodal interaction*, pp. 435–442, 2015.

[3] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, "The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression," in *2010 ieee computer society conference on computer vision and pattern recognition-workshops*, pp. 94–101, IEEE, 2010.

[4] W. T. Freeman and M.Roth., "Orientation histograms for hand gesture recognition," *Intl.Workshop on Automatic*.

[5] R. Ahmad, "A take on h.o.g feature descriptor." [EB/OL]. `https://medium.com/analytics-vidhya/a-take-on-h-o-g-feature-descriptor-e839ebba1e52` Accessed May 4, 2019.

[6] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[7] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[8] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, pp. 3844–3852, 2016.

[9] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[10] A. Bulat and G. Tzimiropoulos, "How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230,000 3d facial landmarks)," in *International Conference on Computer Vision*, 2017.