Emotion Recognition Based on Different Machine Learning Models

Wenhan Yang, Eric Zhang

Abstract

In our project, we are trying to use different machine learning and deep learning models to solve the multi-class classification problem on emotion recognition. Sample pictures are hardly standardized and have various shapes and color-channels, whereas diverse emotion labels even add difficulties to classification. We dive into the problem of finding the optimal classification model by incorporating each models' architectural features. By cross-comparison, case-based optimally can be deduced, and reflections on model selection can be renovated.

1 Introduction

1.1 Problem Statement

For humans, it's easy to read emotions from our facial expressions: we can immediately tell whether a person is happy or sad when we look at his or her face. However, is it possible for machines to do so? In computer science, we define this kind of task as emotion recognition, where the machine is given a facial image as input, and produce a label from set of emotions as output. Emotion recognition has been widely studied since machine learning and particularly the CNN (Convolutional Neural Network) was introduced.

This emotion classifier also has a lot of real-life applications. For instance, one can incorporate this system with the recommender system. If someone is labeled as 'happy' at certain moment, a music recommender system may promote him some exciting songs instead of sad love songs. Another useful application would be secure driving. If a drived is detected as 'sad' or 'angry' by the system in real time, then it might be helpful to send an security alert. In short, emotion recognition has plenty of use cases.

In this paper, we will develop a emotion recognition classifier based on a dataset, by implementing several classical machine learning algorithms. To be specfic, in section 2, we will introduce the dataset we use for our trained classfier, and give a brief description on our dataset. In section 3, we will introduce different machine learning models we have implemented, including but not limited to the structure of the model, the features or formulations of the model, etc. In section 4, we will discuss the results we obtained and what we have discovered from the results. In section 5, we will reflect on our entire project and propose some future works and directions.

2 Dataset

2.1 Raw Dataset

We use an online facial expression dataset from Professor Brian Lee Yung Rowe who is from CUNY. The dataset contains 13719 images with emotion labels. The dimension of the images are 350 by 350. Most of them are grayscale faces. We also have bad data which are in different dimension or in RGB channels, which we will deal with them in the preprocessing part. The label set contains anger, contempt, disgust, fear, happiness, neutral, sadness, and surprise. The sample input image is shown in *Figure 1.*

2.2 Visualization

In order to have a big picture of what our dataset looks like, we converted all images into gray-scale so that we can express our input data into a matrix form. Then we use it as the input of Principal Component Analysis. We extracted the first two PCs and color our data based on different labels. However, after we run PCA, the clusters are not separated. Hence, we can conclude that our dataset contains complicated information which two PCs cannot represent well. The figure is attached in the appendix since it is too big.

Figure 1: Sample input with label *happiness*

2.3 Preprocessing

By observation, the pictures in the dataset we adopted has various sizes and has various color channels. To ensure that the input is carefully formatted, strategies of resizing and channel matching are used to standardize the input. Images are firstly resized, mostly downsized, into 32x32 pixel size. While we are making the input size smaller, it causes the input pictures to changes resolutions, namely reducing it. This implies the loss in the representative information, and lower-resolution network will take slightly longer to train. Also, ass some pictures are not originally in square shape, converting them into a square will either "stretch" or "compress" the facial expressions, causing biases. In color channel matching, although most of our images are in Grayscale(1-channel), in order to best preserve the information in the rest RGB(3-channel) images, it would be more careful to transfer the Grayscale to RGB than the reverse.

Input format differs among models. For simpler machine learning models such as multinomial logistic regression, the input is 2D-array, sample size by 3072, which means that each row of the array stores a 32x32 color image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image. For ConvNet models, it intakes 4D array with dimension (sample size, 32,32,3), where the 3 in the last dimension implies number of color channels. In terms of data labels, simple machine learning models uses original emotion label as y, whereas in ConvNet models they need to be converted into integers by building a dictionary to record those numbers' corresponding emotion.

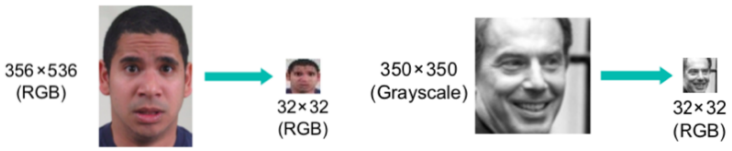

Figure 2: Image Preprocessing Process

3 Solution

3.1 Model List

We used logistic regression, KNN, decision tree, random forest, CNN with VGG, GoogLENET, SEResNet, InceptionV3, and MobileNet. The reason we use the first four models is because our project is a traditional classification problem. We use CNN because of the following two reasons. First, our inputs are images. Second, for this task, we are trying to label emotions which are highly relevant to local features in a face. For example, when we are happy, we are very likely to mouth up. This kind of local feature exactly matches the motivation of CNN, which is to use kernels to learn local informations.

3.2 Training Design

For simple machine learning structures, we use first 500 samples and then all samples, and imported the sklearn packages for existing models. For ConvNets, only first 500 samples are used for lower computational cost on our personal computer. A train-test-split ratio of 2 to 1 is used. For each ConvNet model, we train it for 30 epochs with choice of initial learning rate of 0.001, batch size of 100, and early stopping when the test loss increases for 3 consecutive epochs. We use cross entropy loss as loss function, and continue apply Adam for each epochs to adjust the learning rate.

3.3 Model Design

Below, we will briefly introduce the models which are not introduced in class.

GoogLeNet GoogLeNet is a 22-layer network that incorporates convolution layers, pooling layers, inception layers and worth mentioning a dropout(40%) layer before the linear layer as a regularization technique that is used during training to prevent overfitting of the network. GoogLeNet is a variant of the Inception Network, and such inception architecture study on how an optimal local sparse structure in a convolutional vision network can be approximated and covered by readily available dense components. However, the stacking of Inception layers may drastically increases the computational cost and causes computational blow-up within a few stages. Hence, it leads to another main idea, which is dimension reductions and projections. Simply, it applies 1x1 convolutions for reduction before 3x3 and 5x5 convolutions(*Figure 3*). This specific architecture allows for increasing the number of units at each stage significantly without an uncontrolled blow-up in computational complexity.[1]

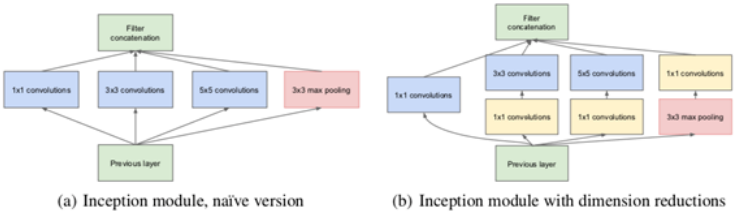

Figure 3: Idea of dimension reductions: applying 1x1 convolutions before larger ones

InceptionV3 One ConvNet model named InceptionV3 inspired by GoogLeNet has made following crucial improvements. Firstly, it factorizes convolutions with larger filter size into either several smaller convolutions or spatially asymmetric convolutions(*Figure 4*) in order to save computation cost. Secondly, the usage of auxiliary classifiers promotes stable learning and quicker convergence. In the training phase, by experiment, application of auxiliary classifiers gives slight promotion in accuracy in late training stages, while being insignificant in early stages. This would also acts as regularizer for the main classifier, giving better performance if auxiliary branch is batch normalized or has a dropout layer. Thirdly, InceptionV3 does grid size reduction by concatenating 2 parallel blocks of convolution and pooling(*Figure 5*), which reduces the computational cost even further while removing the representational bottleneck. The breakdown of convolutions and addition of classifiers generally makes the network deeper and more complex, so InceptionV3 has 42 layers, nearly two times of GoogLeNet's size, and the computation cost is about 2.5 times higher. This large structure is also friendly to low-resolution inputs, though it may take computational cost.[2]

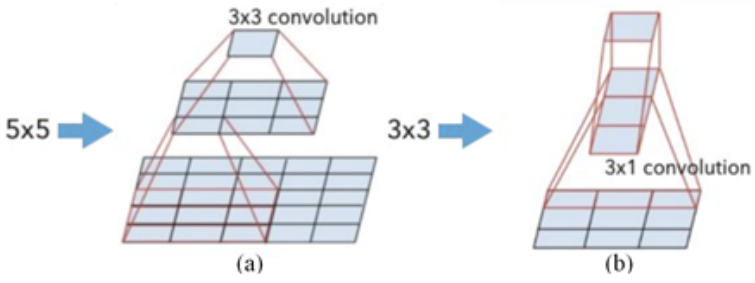

Figure 4: Convolution factorization. (a) 5x5 convolution replaced by two 3x3 convolution, (b) 3x3 convolution replaced by 3x1 convolution with 3 output units

InceptionV3 uses label-smoothing regularization(LSR), which optimizes the usage of cross entropy loss, $l = -\Sigma_{k=1}^{K} log(p(k))q(k)$. While the loss get minimized, the log likelihood of probability distribution of some labels may grow much height than the others, resulting in over-fitting and poor-adaptation because the model becomes too confident in some labels than the other. The motivation is adding a stochastic perturbation to labels. So, this biased preference in training could be solved

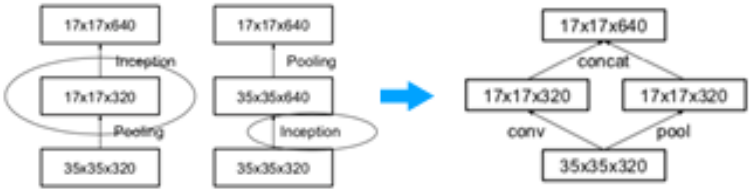

Figure 5: 2 parallel blocks, P and C, concatenation

by replacing ground-truth label distribution with a mixed distribution: $q(k) = (1-\epsilon)\delta_{k,y} + \epsilon u(k)$, where log likelihood is maximized at $q(k) = \delta_{k,y}$, and $u(k)$ is a fixed distribution(e.g. 1/K). Hence, $l = H(q', p) = -\Sigma_{k=1}^{K} log(p(k))q'(k) = (1-\epsilon)H(q,p) + \epsilon H(u,p)$, achieving the desired goal of preventing the largest logit from becoming much larger than all others.[2]

MobileNet MobileNet has much simpler structure. Mainly depth-wise separable convolutions are used to replace standard convolutional filters by depth-wise convolutional filters and point-wise convolutional filters(*Figure 6*), for combining and filtering respectively. Factorization into much simpler convolutions drastically reduces computation and model size, and the resulting low latency is very friendly to common personal computers. The network can also have two hyper-parameters, normally assigned in range (0,1]: width multiplier to makes model even thinner and resolution multiplier to reduce representation. Both hyper-parameters would reduce number of parameters by hyper-parameter squared, and same effect on the computational cost.[3]

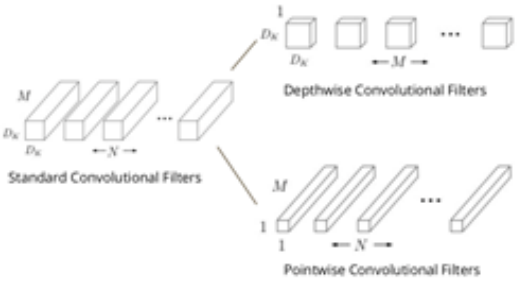

Figure 6: Effective complexity reduction of depth-wise separable convolution in MobileNet

SEResNet SEResNet is a combination of SEnet and ResNet. We first introduce SEnet here. As illstrated in *Figure 7*, we have an input X from inpur or hidden layer. Then, by some transformation, we obtain the result U. Then, what SEnet does is that it tries to learn a weight for each channel in U. Thus, it first does global average pooling on each channel and obtain a vector of length C. Then, it passes the vector to FC, ReLU, FC, and Sigmoid and obtain a vector of same length, but in probability distribution. This is shown by different colors in length C vector in the layer output. Finally, the new X is generated by multiplying each weight on each original channel. For SEResNet, it's simply a combination of SEnet and ResNet, as *Figure 8* shows. We can recall that the idea of ResNet is to add an extra x into the output.[4]

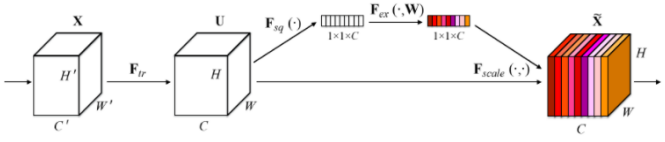

Figure 7: A general SEnet

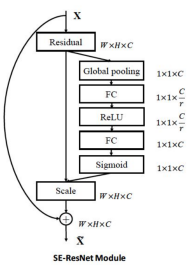

Figure 8: SEResNet

4 Results and Discussion

4.1 Overall Result

Over all test accuracies, the highest is achieved by SEResNet18 at 75.8%, many others has best results around 64%, and the lowest is decision tree, which gives 60.1% test accuracy. Using the first 500 samples will gives better results than using all data in multinomial logistic regression(72.2% vs. 68.4%), as we discovered that all samples include more available labels that will causes possible overfitting in classification. The overall result is illustrated by *Figure 9* and *Figure 10*.

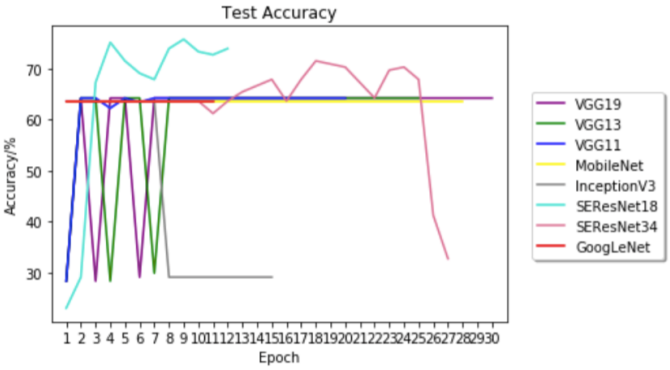

Figure 9: Plot of Overall Results

Model	Best Test Accuracy
Logistic Regression	72.2%
KNN	61.0%
Random Forest	71.5%
Decision Tree	60.1%
VGG11,13,18	64.2%
GoogLeNet	63.6%
InceptionV3	63.6%
MobileNet	63.6%
SEResNet18	75.8%
SEResNet34	71.5%

Figure 10: Table of Overall Results

4.2 Discussion of Different Model Results

We will discuss some well-performed and bad-perfomed models in this section.

Logistic Regression We achieved a very good score on logistic regression. As *Figure 11* shows, most values are on the diagonals of the confusion matrix, which means the classifier achieves high accuracy. We observe some zero entries on diagonal. This is due to the extreme small size of testing data on that class, which is a drawback of our dataset.

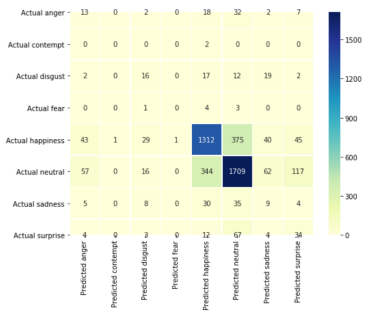

Figure 11: Confusion Matrix of Logistic Regression

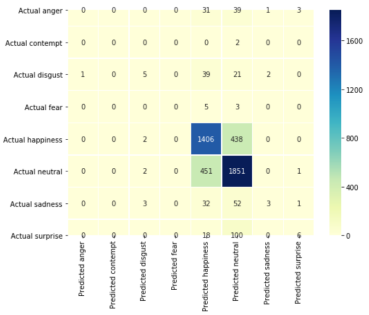

Figure 12: Confusion Matrix of Random Forest

Random Forest We also achieved a good score on random forest. As *Figure 12* shows. It's slightly worse than logistic regression, but much better than single decision tree which makes sense by the essence of random forest.

SEResNet This is the best model of all. The SEResNet learns the weight for each channel in the previous layer, which seems to play a very important role and hence get the best accuracy. We apply early stopping to this model as introduced before. *Figure 13* shows the plot of accuracy.

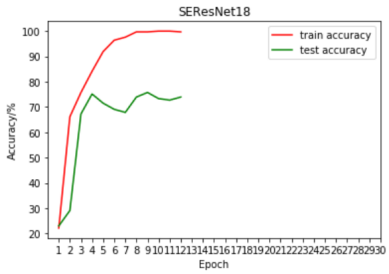

Figure 13: Plot of Accuracy for SEResNet18

GoogLENet, MobileNet, and InceptionV3 From *Figure 9*, both GoogLeNet and MobileNet produces very steady results; the very quick convergence of accuracy within few epochs shows that Adam does fine tuning on learning rate adjustment. Generally MobileNet with width multiplier = 1 runs the quickest amongst all networks, and due to its simpler structure, it could run for more epochs before early stopping than GoogLeNet as overfitting will arrive later. InceptionV3 has very high computational cost on my personal computer, and drops drastically within few epoch after a quick convergence. Due to its prominent depth and complexity, and with our low-resolution input, the computational cost is apparently promoted. The potential reason for overfitting is that over-confidence of model occur when we apply cross entropy loss function. Presumably that our dataset is imbalanced, and thus gives apparent preference in give certain labels. One possible solution is to change the perturbation rate in LSR, which will smooth the imbalanced labeling.

Overall We have shown that logistic regression and random forest gives good performance on the whole large data set, and also give satisfacotory results on small and heavily imbalanced data set(500 data). For more complex models, SEResNet18 seems to be a good choice, when the data set is small, to reach higher plateau of accuracy. MobileNet would allow low latency while assures fair level of accuracy, which is a good choice for quick training even on larger data sets. It would be more careful to use the depth-prominent InceptionV3, specifically using LSR with acceptable perturbation term chosen.

5 Future Works

There are several aspects we can improve on this project.

Data Visualization As discussed, we tried to obtain a good visualization of our data based on PCA. However, two PCs are not enough so we may need to try other dimension reduction techniques.

Dataset The labels are not evenly distributed in our dataset. We have more happiness labels than contempt labels, for instance. Hence, this may cause some issues in learning process. We may need to use bootstrap method to upsample some data to get a more balanced dataset.

Definition of labels Some labels are rather too ambiguous. For example, we sometimes can't tell whether a person is disgust or contempt. Also, 'neutral' label gives some vagueness. These create difficulties for machines to learn the features of certain emotions. We may want to develop more concrete labels and use unambiguous pictures to have a more robust model.

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014. [Online]. Available: https://arxiv.org/abs/1409.4842

[2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015. [Online]. Available: https://arxiv.org/abs/1512.00567

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: https://arxiv.org/abs/1704.04861

[4] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," 2017. [Online]. Available: https://arxiv.org/abs/1709.01507

6 Appendix

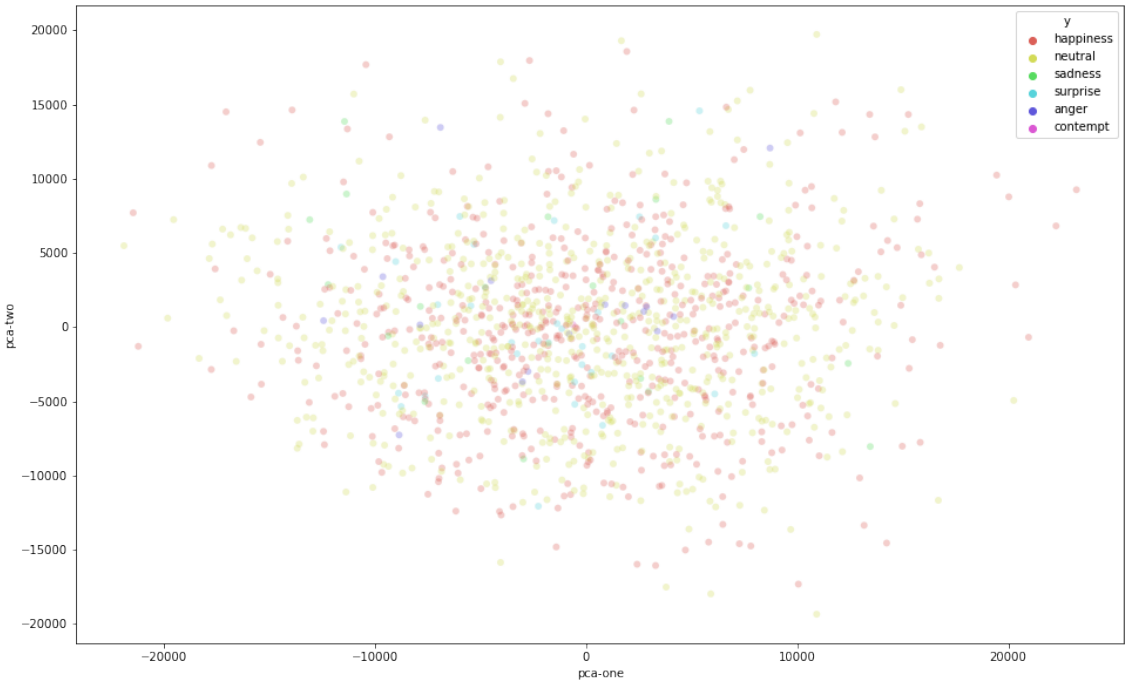

Figure 14: Visualization of dataset using first two PCs