Towards Facial Expression Recognition: Neural Networks with Bimodal Distribution Removal and Convolutional Neural Networks

Ruitao Leng[1]

[1] Research School of Computer Science
Australian National University
U6777647@anu.edu.au

Abstract. Facial expression recognition (FER) is an important part of human-computer interaction and can be used for educational, medical and technical purposes. The success of FER relies on classifying facial features into different expression categories. In this paper we perform FER with two types of features: hand-crafted features and features learned from convolutional neural networks (CNNs). Between the two, we find end-to-end CNNs can achieve 1.63 times higher accuracy on SFEW dataset. For hand-crafted features, we propose a baseline neural network (NN) to classify LPQ and PHOG descriptors, by changing various hyperparameters. Then bimodal distribution removal (BDR) is applied to the baseline NN as an outlier removal approach. For automatically learned features, we experiment with both self-designed CNNs and transfer learning techniques, getting a best model with 44.74% overall accuracy.

Keywords: machine learning, classification, facial expression, SFEW dataset, outliers, neural network, bimodal distribution removal, deep learning, CNN, transfer learning

1 Introduction

Facial expression recognition (FER) is an active research direction in the field of computer vision and artificial intelligence. It is a classification problem of mapping face images to expression categories. A facial expression usually involves movements of the muscles beneath facial skin, happening during daily activities or under the stimuli of certain emotions [3]. It can provide rich information about mood, personality and temper, affective state and mental activity [4]. Understanding human facial expressions is an important step towards understanding human behavior in human computer interaction. FER techniques are also widely used in fatigue driving detection, smart homes, face and visual speech synthesis, video conferencing, security and education [1,5,6,7]. However, to achieve accurate FER, algorithms need to overcome variability of human faces like gender, orientation, color, posture, as well as environmental differences like lightening condition, background, shade, etc. Thus, the extraction and classification of robust features is required.

Traditionally, features are manually extracted from images: either global features, where features are extracted from the entire face; or local features, where features are obtained from local face parts [12]. Previous researchers use eigenfaces to represent global features [13], where the eigenvalues of grayscale images form a compact representation. In 1999, Donato [4] compared different methods including principal component analysis (PCA), independent component analysis (ICA), and Gabor-wavelet [14] descriptors. The best performance comes from ICA and Gabor-wavelet. However, these global features are computationally expensive. They are mostly applied to images from controlled lab environments [12] and hence not robust to environment changes. On the other hand, local features can resolve these problems. Typical examples are local feature analysis (LFA) [15] and local binary pattern (LBP) [16]. Local phase quantization (LPQ) is an extension of LBP and has more robustness against blur [1]. Recently, pyramid of histogram of oriented gradients (PHOG) [17] has also been applied to FER [1].

Then, these features are categorized using classifiers. A typical classifier is SVM [1], which performs non-linear classification. But SVM is not suitable for large datasets where the number of patterns is greater than the number of features. Furthermore, there are no probability outputs for SVMs either. By contrast, feedforward neural networks (NN) shows good potential in solving these issues [9]. However, since NN is a non-parametric estimator where no prior knowledge about the decision boundary is assumed, the drawback of non-parametric statistics also apply to NN [18]. Such drawback is referred to as bias-variance dilemma. For a general estimator, bias appears when the assumed estimator structure differs from the actual structure. High bias will cause the model to ignore the connection between training samples and target, hence result in underfitting. Variance is generated when the estimator focuses on patterns of a particular dataset and cannot generalize well. So, high variance will make the estimator overfit. The dilemma states that it is difficult to find a model that both captures enough details from the finite training data and can generalize well to unseen test data.

One way to solve the bias-variance trade-off issue is outlier removal [18], where the initial variance in training samples are reduced, making it easier to obtain an average model that can generalize to test data. Outlier removal aims to delete the noisy training patterns which cause dramatic weight changes during back propagation. Traditional methods include absolute criterion, least median squares and least trimmed squares [19]. These methods suffer from either oscillation or slow convergence. A better solution is to use bimodal distribution removal (BDR) [2]. At the early stage of training, the frequency distribution of training errors of all samples forms a two-modal shape. The modal with higher mean indicates outlier candidates. By removing such outlier candidates, the network finally learns a solution that forces all patterns to have lower errors.

With the rise of deep learning, especially convolutional neural networks (CNNs), FER can be solved using end-to-end training with higher accuracy [8,10,11] than traditional extraction-classification methods. For these approaches, CNNs directly take face images as input and give the probabilities of each expression class by automatically learning features of different levels. Recently, the evolution of deep learning techniques such as max-pooling [22], batch normalization [23] and residual learning [24] further improves CNNs' performance.

In this paper, we investigate FER with both hand-crafted features and end-to-end CNNs. We first use NN to classify manually extracted LPQ & PHOG descriptors. Different configurations of the NN structures are explored to demonstrate the effectiveness of NN. Furthermore, BDR is applied to a baseline NN, aiming to investigate its influence on network performance. Then we train a CNN with 6 convolutional layers using raw dataset images. Due to the limited size of dataset, we also experiment with transfer learning techniques using pretrained Resnet18, Resnet50 and Resnet101 [25].

2 Method

2.1 Dataset, train/test split and preprocessing

This paper is based on the Static Facial Expressions in the Wild (SFEW) dataset [1]. The SFEW dataset is developed from the Acted Facial Expressions in the Wild (AFEW) dataset, which contains video clips of human expressions from movies. SFEW consists of frames extracted from the AFEW dataset. The advantage of using this dataset is that those movie frames mimic facial expression under real-world circumstances, having imperfect illumination, different face poses, varied resolution, etc. This way the evaluated FER system can demonstrate its performance in nearly realistic environment.

The SFEW dataset has a total number of 675 unconstrained facial images, with ground truth labels of seven classes: angry, disgust, fear, happy, sad, surprise and neutral. As shown in **Fig.1**, the seven classes are equally numbered except 25 patterns are missing for "disgust". This makes the dataset unbalanced. SFEW requires strictly person independent (SPI) protocol, where we randomly choose 50% of the data to put in set 1 and make the remaining data set 2. Any model should first train on set 1 and test on set 2, then reverse the process (i.e. 2-fold cross validation). The class distribution after train/test split is also shown in **Fig.1**.

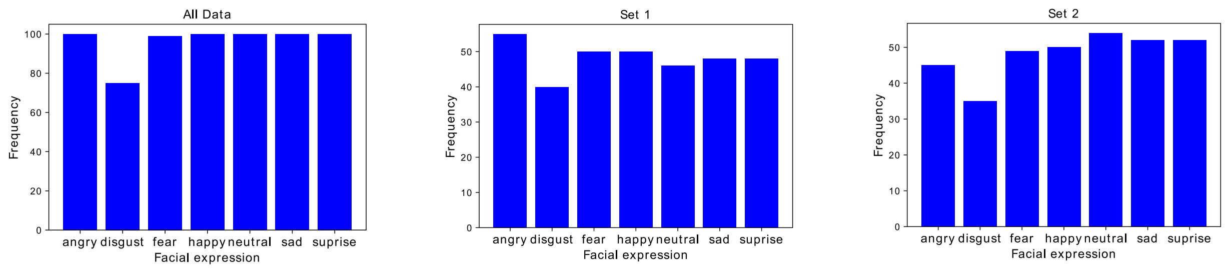

Fig. 1. Class frequency distribution of SFEW. Left figure shows the original dataset. Middle and right figure shows the split datasets according to SPI protocol. 2-fold cross validation should be performed on set 1 and set 2.

For experiments with hand-crafted features. Dataset preprocessing follows the method stated in [1]. First, Viola-Jones detector is applied to images to find the bonding boxes of faces. Then LPQ and PHOG descriptors are computed on the cropped faces. To remove redundant features and hence reduce problem complexity, PCA is applied to the extracted features, leaving top-5 principal components for LPQ and PHOG respectively. The resulted dataset is a 675 by 11 matrix, with 10 columns corresponding to features and an extra column for target values. Feature normalization is a typical method of increasing network performance. To normalize, the mean and variance of each feature column are shifted to 0 and 1 respectively. The effectiveness of normalization will be further discussed in results and discussion section. The original SFEW target values are integers from 1 to 7. They are changed to 0 to 6 because all experiments are done with Pytorch, which expects target values to start from 0.

For CNN experiments. We use RGB images as network inputs. We load them as PIL images, with three channels' intensity values in [0, 1]. As for our self-designed CNN, we apply the same transformation to training and test images: resizing from original size 720×576 to 160×160, further normalizing each channel by subtracting mean and dividing standard deviation (both set to 0.5). As for the transfer learning models, we follow the method in [25]. Since Pytorch pretrained models with ImageNet, we need to use the same preprocessing routine so that the models can be transferred to our problem. We resize all images to 224×224 and normalize using the mean and standard deviation of ImageNet provided in [25]. For training images only, we perform data augmentation by randomly flipping images horizontally. This step is to increase the diversity of training data and help the network learn robust features.

2.2 Evaluation metric

All models' performance is evaluated using the SPI protocol, as required by [1]. Based on confusion matrix, the overall accuracy, class-wise precision, recall and specificity are calculated using the following equations. After 2-fold cross validation, the final results are averaged over two test sets.

$$\text{Overall accuracy:=}\frac{true\ positive+true\ negative}{true\ positive+false\ positive+false\ negative+true\ negative} \quad \textbf{(1)}$$

$$\text{Precision:=}\frac{true\ positive}{true\ positive+false\ positive},\quad \text{Recall:=}\frac{true\ positive}{true\ positive+false\ negative},\quad \text{Specificity:=}\frac{true\ negative}{true\ negative+false\ positive} \quad \textbf{(2)}$$

Of the four statistics, overall accuracy is used primarily to determine the performance of neural networks, the other three are reported in a detailed comparison with SFEW baseline [1].

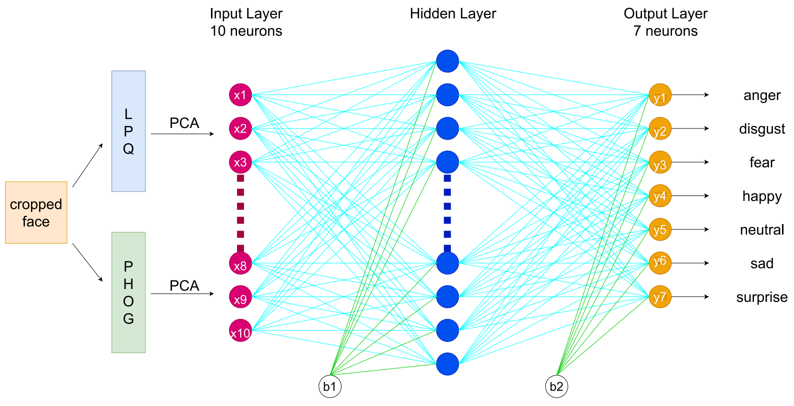

Fig. 2. Pipeline of FER using hand-crafted features. For the neural network, each input x_i is 1 of 10 feature values. Each output y_j is the probability that the pattern belongs to class j.

2.3 Feedforward neural network

The pipeline of FER using hand-crafted features is shown in **Fig.2**. The classification task is conducted using feedforward neural network (particularly referring to multilayer perceptron-MLP in this case). A simple one hidden layer NN is adopted. To maintain enough amount of feature information, for each sample in the dataset, 5 LPQ features and 5 PHOG features are grouped together as a 10-dimensional vector. Thus, the input layer of neural network has 10 neurons. The output layer has 7 neurons, each refers to a facial expression. As a rule of thumb for a multi-class classification problem, each output value y_j is the probability that the input sample belongs to class j, obtained from the softmax function:

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \textbf{(3)}$$

where z_k is the input value of the kth output neuron. Then input is thus assigned to the class with largest y_j value. On the other hand, hidden layer uses ReLU as activation function:

$$f(x) = \max(0, x) \quad \textbf{(4)}$$

Although vanishing gradient of traditional sigmoid function is not significant for our single-hidden-layer network, using ReLU is more computational efficient since we do not need to compute exponentials.

Training a NN is to minimize a loss function between target value and network output. In accompany with the softmax function, this paper uses cross-entropy loss:

$$L(y,t) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\ln(y_{nk}) \quad \textbf{(5)}$$

The network weights are updated using gradient descent, where gradients are calculated by back propagation. In Pytorch, gradients are calculated automatically by auto-differentiation. To get better performance, network hyperparameters need to be carefully designed. Learning rate, number of hidden neurons and optimizer are determined through experiments. By changing one of them and keeping others the same, we can investigate the effect of each individual hyperparameter. A baseline neural network is proposed combining optimal hyperparameters. The baseline network is then used as a contrast to demonstrate the effect of BDR.

2.4 Bimodal distribution removal (BDR)

As originally proposed by Slade and Gedeon [2], BDR provides a method of detecting and removing outliers, as well as a natural stopping point for the training process. During training, the frequency distribution of errors is recorded for all training samples every 50 epochs. At early stage of training, the distribution will shift from a large-variance Gaussian to a bimodal distribution, given that significant outliers exist in the dataset. The two peaks are separated using the mean value $\overline{\delta_{ts}}$ of the error distribution of all training samples. Samples with larger error than $\overline{\delta_{ts}}$ are taken as a group of outlier candidates. When removing outliers, there is a concern that outliers cannot be removed too quickly. Otherwise the size of training set will decrease dramatically and result in overfitting. BDR uses the selecting criterion that samples with error greater than:

$$\overline{\delta_{ss}} + \alpha\sigma_{ss} \quad \text{where } 0 \le \alpha \le 1 \tag{6}$$

will be removed. Here $\overline{\delta_{ss}}$ is the mean value of errors of the outlier candidate group, σ_{ss} is the corresponding standard deviation. The variance ν_{ts} of error distribution of all training samples is used as a control handle. In [2], BDR starts when $\nu_{ts} \approx 0.1$ and is repeated every 50 epochs. BDR stops when $\nu_{ts} < 0.01$, and training process is also stopped at this time. However, it is worth investigating whether these values are universally applicable or are specifically tailored for the particular dataset and loss function. In this paper, experiments are designed on top of previously mentioned baseline NN, studying the influence of above mentioned parameters, as well as the overall effectiveness of BDR in the proposed FER system.

2.5 Convolutional neural network (CNN)

CNN is a special kind of neural networks where neurons are arranged in 3D volumes. Such spatial architecture makes it good at processing images inputs. Meanwhile, since for each filter, the weights are shared across the input volume, the total number of network parameters will not explode for large images. To perform FER in an end-to-end manner, we start by designing a simple CNN shown in **Fig.3**. Our network is inspired by the early layers of VGG [20] network, where only small 3×3 receptive field is used. Our network can be divided into 3 blocks, each having a conv-conv-pool structure, with depth increasing from 32 to 128. For each convolution layer, both padding and stride are 1, so that the width and height of input and output volumes remain unchanged. Again, we use ReLU as the activation function to prevent vanishing gradients. The activations of each convolution layer are standardized in mini-batches during training, using batch normalization [23], which has been proven to help deep neural networks converge faster. The input is down-sampled 3 times using 2×2 max-pooling, to reduce network parameters. We also apply dropout to reduce overfitting, with the idea that muting neurons can cut unnecessary connections in the network and reduce network complexity. During training, after each pooling layer and between final fully connected layer, neurons are randomly muted with probability p. We use the same cross-entropy loss function as the feedforward NN. Details of hyperparameters are shown in results section.

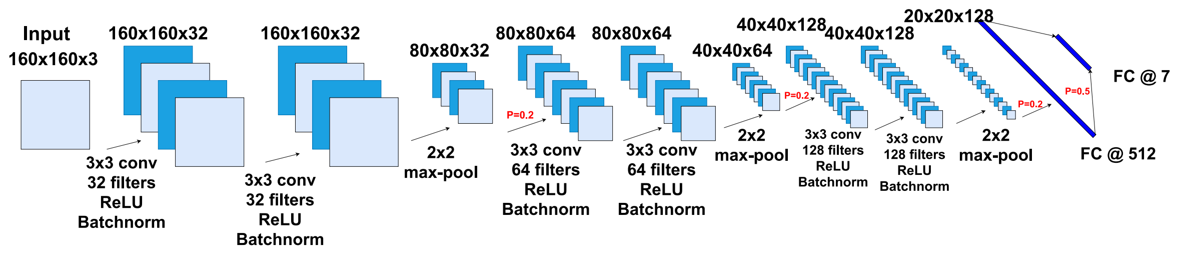

Fig. 3. Structure of the self-designed CNN. It has 6 convolution layers with 3×3 kernels, each followed by batch normalization and ReLU activation. The down-sampling is performed by 2×2 max-pooling. The linear classifier is composed by two fully connected layers

2.6 Transfer learning

Training a deep CNN from scratch requires a large dataset, which is not always realistic. As in our case the dataset only contains 675 images, making it hard to train a CNN with random initialization. An alternative approach is to use a network pretrained on a larger dataset as either a fixed feature extractor or weight initialization. Such approach is called transfer learning. For transfer learning, we use the state-of-the-art residual network [24] as backbone. For each residual module, besides weight layers, an identity mapping is used to pass the input to the output. So that when the network grows deeper, the error can still be backpropagated instead of vanished, since in worst case unnecessary layers will learn to be identity mappings. Such characteristics enable the training of very deep CNNs. Pytorch's Resnet family was pretrained on ImageNet, which contains 1.2 million images with 1000 classes. We perform transfer learning on the Resnet18, Resnet50 and Resnet101. The numbers indicate the layers in the network.

Generally, people use pretrained networks as fixed feature extractors when the target dataset is small but similar to the original dataset. Since the similarity between SFEW dataset and ImageNet is hard to decide, we experiment with both feature extractor and finetuning cases. For fixed feature extractor, we replace final fully connected layer of Resnet with a new layer, which has the same number of inputs but the output number is changed to 7. Then all previous layers are frozen and only the new layer's weights are updated during training. For fine tuning, we replace the final layer in the same way, except that during training weights of previous layers are also updated.

Since our dataset is small, we start training with small learning rate because otherwise learning too quick from this dataset will result in severe overfitting. We schedule the learning rate with exponential decay towards the end of training. Details are shown in results section.

3 Results and Discussion

2.1 Experiments on baseline neural network

The goal of these experiments is to choose optimal hyperparameters and determine a baseline NN that is suitable for this facial expression classification task. Because the dataset is relatively small (with total 675 samples), 2-fold cross validation is adopted. The dataset is randomly partitioned to 2 parts with each part containing half number of the data. A NN is first trained on one set and predicts on the other. Then another NN is trained on previous test set and predict on previous training set. The accuracy, given by equation **(1)**, is the average of the two NNs. This method ensures no data is wasted. One special thing to notice is that NNs do not give identical results for multiple runs, because each time the weights are randomly initialized. Therefore, to get convincing results, each NN is trained 10 times. So, for each set of hyperparameters configuration shown below, the accuracy and loss results are the average of 2×10 different NNs.

Learning rate. Learning rate is the primary concern when designing a neural network. It controls how much the weights are updated each step, in response to the gradient of loss function. If the learning rate is too small, the network needs long time to train; if the learning rate is too large, the parameters will oscillate violently and cannot reach the optimum. To determine the appropriate learning rate for this task, experiments are done with a NN with the structure of **Fig.2**. The number of hidden neurons is set to 10. For each learning rate, training cross-entropy loss is plotted against epochs, shown in **Fig.4 (a)**. When learning rate is too large (lr = 1), the loss gets stuck at around 2. When the learning rate is too small (lr = 0.001), the loss decreases along epochs but very slowly. The reasonable learning rate for this task is around 0.005 to 0.05, where the loss decreases smoothly.

Number of hidden neurons. The number of hidden neurons decides how many dimensions are used to encode the hidden features. Experiments are conducted with hidden units ranging from 2 to 300. For each number, the NN is trained on learning rates 0.005, 0.01 and 0.05 to further determine the best learning rate. The test accuracy after cross validation is shown in **Fig.4 (b)**. The NNs are trained for 2000 epochs. It can be seen that a small number of hidden neurons gives bad results. This indicates that the NN cannot distinguish facial expressions based on hidden features of low dimension. When the number grows larger, the accuracy also increases. But too many hidden units will make the model rather complex, which requires more time to train and can also result in overfitting. So, for the proposed FER system, 200 hidden units are chosen, which can give good accuracy and avoid too complex model. The corresponding learning rate is 0.01.

Optimizer. Pytorch integrated multiple optimizers for weight update. **Fig.4 (c)** shows the comparison between normal stochastic gradient descent and Adam, with the 200-hidden-neuron network. Both optimizers are based on gradient descent. It can be observed that SGD is easily trapped with local minimum, while Adam can overcome this difficulty because it combines the advantages of RMSprop and SGD with momentum [21]. Using Adam can reduce training time as well as increase network performance.

Normalization. To study the effect of normalization for the SFEW dataset, both training samples and test samples are normalized in the same manner: for each feature column, the feature value subtracts the column mean and then it is divided by the column's standard deviation. This way all features are forced to a distribution with 0 mean and variance 1. The effectiveness of normalization is shown in **Fig.4 (d),** using the same hyperparameters, the NN needs much less time to converge when dealing with normalized training data. However, it should be noticed that in real world situations, normalization is hard to achieve when the dataset is too large, or when the test set and training set are from different probability distributions. So, in this paper both normalized and unnormalized data are studied to mimic real world cases.

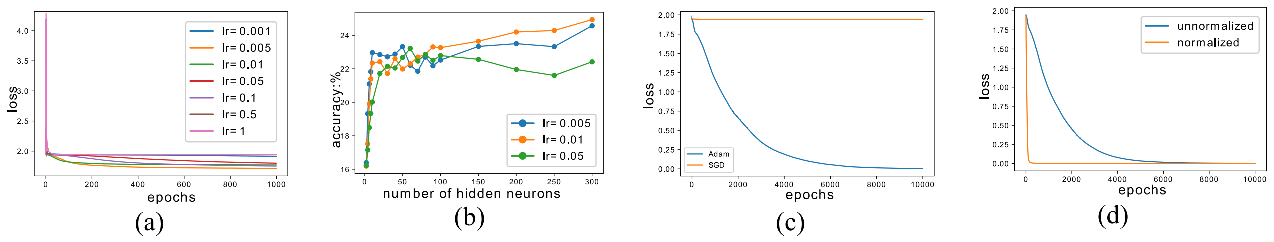

Fig. 4. Experiments with the normal feedforward NN. (a) shows the influence of learning rate. (b) investigates the appropriate number of hidden neurons. (c) studies the difference between Adam and SGD optimizer. (d) addresses the effectiveness of normalization.

With the results from above experiments, the baseline NN is decided to have 10 input neurons, 200 hidden neurons with ReLU activation and 7 output neurons with softmax. The loss function is cross-entropy. The optimizer is Adam, with learning rate equals to 0.01. The baseline NN's detailed performance on both normalized and unnormalized data is reported in **Table 1**. We can see that using normal feedforward NN can have 5-7% more overall accuracy than the SVM based results proposed in [1]. This shows the effectiveness of NN in such classification task.

2.2 BDR with unnormalized dataset

The aim of these experiments is to study the effect of BDR on unnormalized dataset. The implementation of BDR on a normal NN is closely relevant to the loss distribution of training samples. So, before using BDR to remove outliers, the behavior of baseline NN's distribution over training epochs is studied. The results are shown in **Fig.5**.

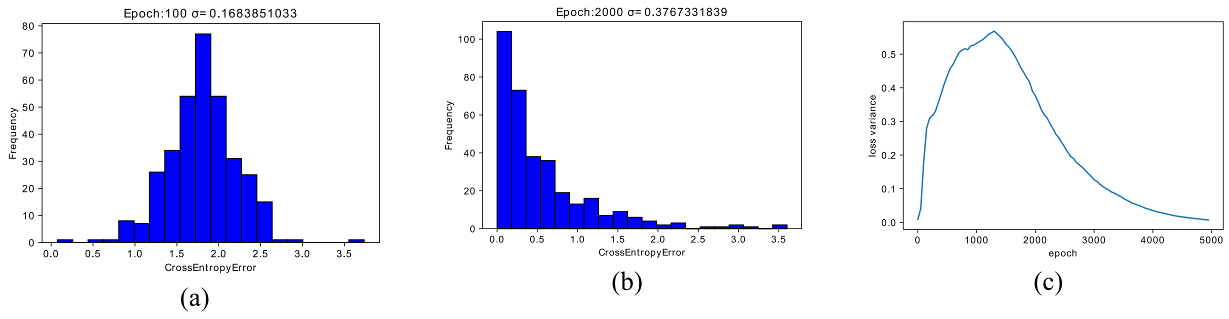

Fig. 5. Error distribution of the baseline NN. (a) error distribution at epoch 100. (b) error distribution at epoch 2000. (c) variance of loss distribution over epochs.

It can be observed in **Fig.5 (a)** that at the early stage of training period, the cross-entropy error distribution can be estimated by a normal distribution. From **Fig.5 (c)** it can be observed that initial variance is quite small. This can be explained as follows: the network weights are randomly initialized with small values. At this point the network cannot distinguish between inputs from different classes, no matter what the input is, the network will map it to a vector containing small values. This way the error of samples from the entire training set will be close to each other, resulting in only one peak. Again because the weights are random, so the distribution will look like Gaussian. When the network begins to learn different features, the weights begin to be optimized, this will cause the error distribution to diverge and therefore result in increasing variance. When the weights are optimized to a certain extent, the network can distinguish inputs from different classes, all the errors are forced to move towards zero. So the loss variance will decrease again. **(b)** shows the error distribution at epoch 2000. At this stage it is evident that the network has learned from training data, since the variance has already decreased. Most errors are located around zero, with only a few at the far end of x axis. This situation can be loosely considered as a bimodal distribution. However, the two modals are not ideally separated.

The original BDR starts when loss variance is smaller than 0.1. Since the error distribution's behavior is not the same as stated in [2], this start trigger is no longer applicable. Through inspection, for the baseline NN, a reasonable point to start is at around 2000 epochs, where the variance starts to fall and the two modals are formed. The implementation of BDR is shown in **Fig.6**. From epoch 2000 onwards, BDR is applied every 50 epochs. This interval allows the network to learn from the new training set. It can be seen from **Fig.6 (c)** and **Fig.6 (d)** that after only 4 rounds of BDR, the variance of loss distribution has dropped below 0.01, and loss itself has decreased towards zero. So, for this task, it is reasonable to use variance<0.01 as a stopping point for BDR. The results indicate that BDR provides a strong force to make the network converge. Using BDR can greatly reduce training time.

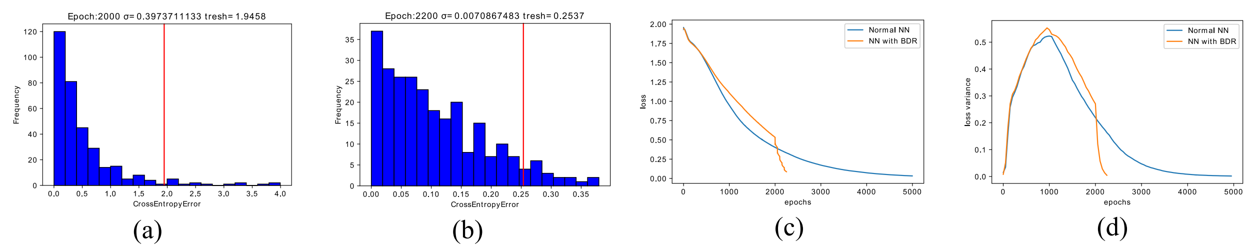

Fig. 6. Apply BDR to baseline NN. (a) BDR starts at epoch 2000, the red line is the threshold calculated from equation **(6)**, samples on the right side of the red line will be removed as outliers. (b) after 4 rounds of BDR, the variance of loss distribution is reduced to 0.0071. (c) loss with and without BDR. (d) loss variance with and without BDR

Besides start and stop point, another parameter to be configured is α in equation **(6)**. Theoretically, this parameter controls how 'strict' BDR is. If α is close to zero, the threshold will be smaller, in this case BDR will be stricter since more samples will be removed as outliers. If α is close one then vice versa. However, it is found through experiment that it barely has impact on the result of BDR for this dataset. As shown in **Fig.7**, although α varies from 0.1 to 1, BDR always terminate within 200 epochs, which corresponds to 3 or 4 runs. Which means the number of removed outliers will be close. To remain as many samples as possible, α is chosen to be 1 for this task.

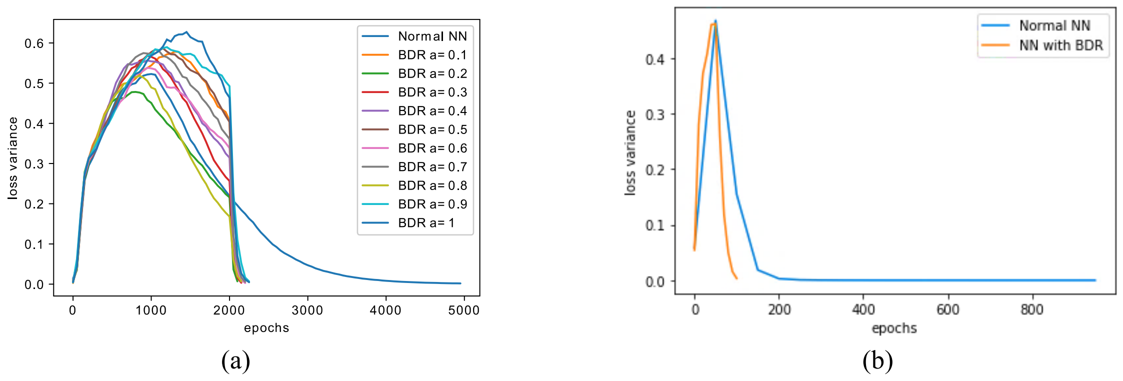

Fig. 7. (a) The impact of parameter α on BDR results with unnormalized dataset. It can be spotted that varying α from 0.1 to 1 has no major impact on the stop time of BDR. (b) Loss variance of the normalized dataset. Note that the variance drops much than the unnormalized case.

With the above configurations, BDR is applied to the baseline NN. Results are shown in **Table 1**. The overall accuracy is only slightly better than the baseline NN with unnormalized data. This is because BDR cannot effectively detect outliers when are significantly distinguishable from valid samples. As shown in **Fig.6 (a)**, those samples on the right side but close to the red line are actually 'slow learners' instead of outliers, which should not be removed. After 4 rounds of BDR, about 130 out of 337 training samples are removed. A large portion of these are actually valid training sample. This will cause the training set to be smaller and increase overfitting, reducing classification accuracy. Such overfitting forms a trade-off with the benefit of removing outliers, that's why there is no significant increase of network performance.

2.3 BDR with normalized dataset

On the normalized SFEW dataset, BDR follows the same approach as in the unnormalized dataset, but with different parameters. Compare the blue line in **Fig.6 (d)** and **Fig.7(b)**, we can find the variance of loss distribution drops much faster with the normalized dataset. This is because by normalizing all features to have the same mean and standard deviation, the classification problem essentially becomes easier. Thus, the network can effectively learn using less epochs. Under such condition, the start point of BDR is set to be epoch 50, and the interval of each run is reduced from 50 to 10. The stopping point and α remains the same as the unnormalized case. The results are shown in **Table 1**. Again, we can observe a slight increase in overall accuracy than the baseline NN.

Overfitting problem. It can be observed from **Fig.3(d)** that with either normalized or unnormalized data, the training loss eventually converges to zero. In this case the training accuracy becomes 100%. But the test accuracy is rather low, with the highest case 27.45%, which is obtained using NN and BDR on normalized data. This is because although the network is fully capable of learning and distinguishing the features, these LPQ and PHOG features are not descriptive enough for facial expressions. So the network performs much poorly on the test set.

Table 1. The results of FER using hand-crafted features / end-to-end models. Each result is obtained by 2-fold cross validation

		anger	disgust	fear	happy	neutral	sad	surprise
Models using hand-crafted LPQ and PHOG features:								
SFEW baseline [1]	Precision	0.17	0.15	0.20	0.28	0.22	0.16	0.15
	Recall	0.21	0.13	0.18	0.29	0.21	0.16	0.16
	Specificity	0.48	0.66	0.64	0.51	0.61	0.60	0.66
	Average acc				**19%**			
NN (unnormalized data)	Precision	0.30	0.17	0.28	0.26	0.18	0.29	0.20
	Recall	0.34	0.17	0.26	0.22	0.18	0.33	0.20
	Specificity	0.86	0.90	0.88	0.89	0.86	0.87	0.85
	Average acc				**24.24%**			
NN with BDR (unnormalized data)	Precision	0.36	0.20	0.27	0.28	0.21	0.29	0.20
	Recall	0.27	0.18	0.33	0.30	0.27	0.27	0.15
	Specificity	0.92	0.90	0.83	0.87	0.82	0.89	0.89
	Average acc				**25.52%**			
NN (normalized data)	Precision	0.35	0.17	0.31	0.28	0.18	0.28	0.28
	Recall	0.34	0.19	0.28	0.24	0.19	0.31	0.28
	Specificity	0.89	0.88	0.89	0.89	0.86	0.87	0.86
	Average acc				**26.48%**			
NN with BDR (normalized data)	Precision	0.35	0.18	0.33	0.34	0.20	0.31	0.23
	Recall	0.37	0.16	0.29	0.25	0.28	0.34	0.22
	Specificity	0.88	0.91	0.89	0.92	0.81	088	0.86
	Average acc				**27.45%**			
End-to-end models:								
Self-designed CNN	Precision	0.44	0.42	0.40	0.35	0.35	0.43	0.32
	Recall	0.51	0.45	0.46	0.32	0.29	0.38	0.22
	Specificity	0.83	0.92	0.88	0.89	0.90	0.91	0.92
	Average acc				**37.03%**			
Resnet50 (fine tune)	Precision	0.50	0.46	0.57	0.48	0.30	0.53	0.36
	Recall	0.45	0.39	0.53	0.44	0.36	0.56	0.41
	Specificity	0.92	0.95	0.93	0.91	0.85	0.91	0.87
	Average acc				**44.74%**			

2.4 Experiments with CNNs

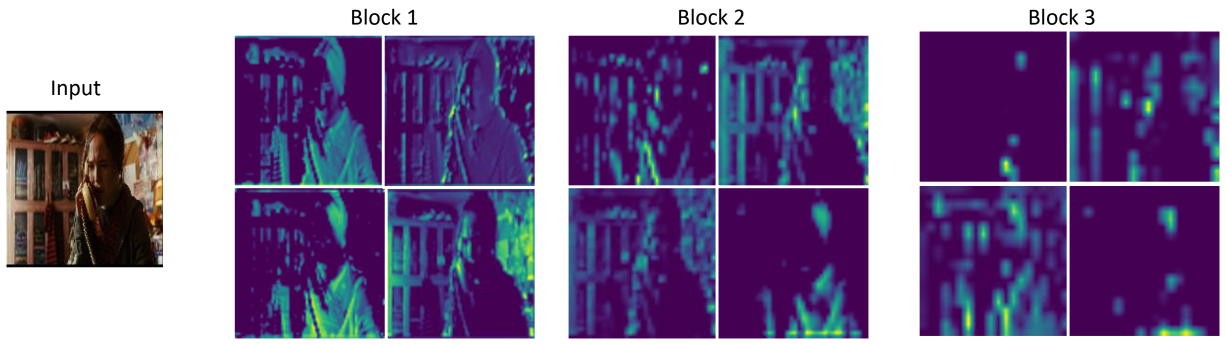

Fig. 8. The activation maps of self-designed CNN. Each "Block" refers to a conv-conv-pool structure shown in **Fig.3.** The activation maps are randomly chosen from the output volume of the corresponding layer, and rescaled to the same size for visualization.

Self-designed CNN. The self-designed CNN is trained on a GTX2070 GPU, using the preprocessing method stated in section **2.1**. To reduce GPU memory usage, the batch size for both training and evaluation is set to 8. Based on a rule of thumb, we use 0.001 as learning rate. We use Adam optimizer and train the CNN for 50 epochs. Since 2-fold cross validation is required, we trained two models and the averaged results are shown in **Table 1.** The overall accuracy 37.03% is much higher than NN methods. To understand the functionality of our CNN, we visualize the layer activations in Fig.8. Since we use ReLU, the activations look sparser when the network goes deeper, because ReLU zeros out negative-valued activations. We can see at early stage of the CNN (Block 1), the network focuses more on large-scale features, the contours and edges are highlighted. At late stage (Block 3), the network focus more on local features, which only correspond to a small region in the input image. In this way, the network learns features that are appropriate for our FER task itself.

Transfer learning. We experiment with 3 pretrained networks of increasing depth: Resnet18, Resnet50 and Resnet101 [25]. The networks are trained on GTX2070, using batch size 8. The optimizer is also Adam. Since the networks are larger than our self-designed CNN, we train these ones for 150 epochs instead of 50. We schedule the learning rate as follows: the initial learning rate is 0.001, it is reduced to 1e-4 at epoch 100 and further reduced to 1e-5 at epoch 120. The aim is to let the network learn less from the dataset as epochs grow, because the important features should have already been learned and only small changes are needed to the weights. Theoretically overfitting can be reduced since the network will not become too adapted to the dataset. Using the method stated in section **2.6**, we train all three networks for both fixed feature extractor and fine tuning.

The overall accuracy is shown in **Table 2**. As we can see, for all three models, using them as feature extractor is worse than fine tuning. This essentially proves that the SFEW dataset does not closely resemble ImageNet. Since ImageNet contains photos of various objects, humans, animals etc., but SFEW only contains screenshots of human expressions from movies scenes. So, when used as feature extractor, the pretrained network cannot extract effective features for expression classification. When fine tuned on SFEW, the best model is Resnet50, which gives 44.74% overall accuracy, better than our self-designed CNN trained from scratch. It means transfer learning is an effective approach to deal with small datasets.

The loss and accuracy curves are shown in **Fig.9**. It shows that the deeper the network, the longer time is required for loss to drop. We can also see overfitting is a main source of error, since in late epochs, the loss curves diverge dramatically but accuracy remains stable. All three models are overfitting, because of the limited size of SFEW dataset. Resnet101 has the most severe overfitting, because deeper network can easily memorize training patterns without the ability to generalize.

Table 2. Compare fixed feature extractor and fine tuning. Each result is obtained by 2-fold cross validation

	Accuracy	
	Fixed feature extractor	Fine tune on SFEW
Resnet18	29.03%	42.97%
Resnet50	34.07%	44.74%
Resnet101	34.67%	41.93%

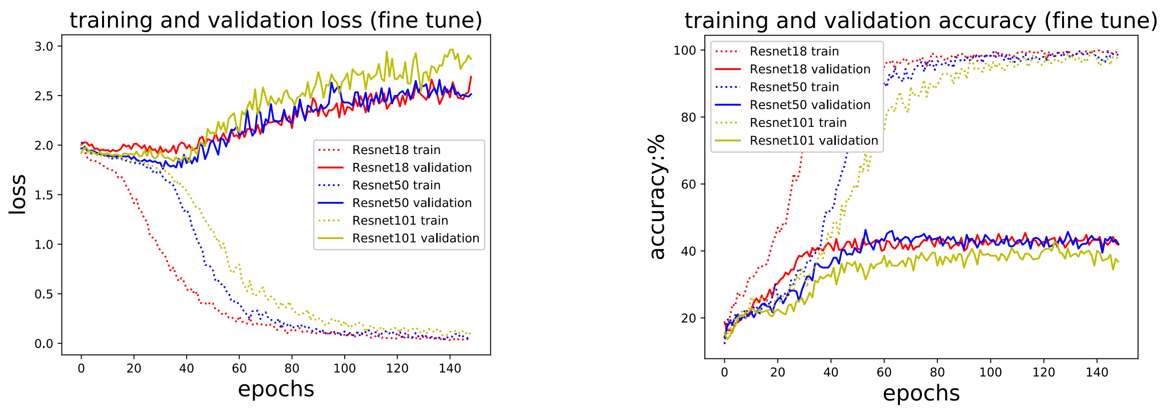

Fig. 9. The loss and accuracy curves during training. The Resnet models are fine tuned on SFEW. (Since 2-fold cross validation is used, this figure only shows the case of train on set 1 and test on set 2.)

Compare to NN with hand-crafted features. As shown in **Table 1**, both the self-designed CNN and fine-tuned Resnet50 have better accuracy than NN models. This shows that automatically learned features form a better representation of facial expressions, thus the final fully connected classifiers of CNNs can more successfully distinguish between different classes. Meanwhile, CNNs are more robust to class imbalance of the training set. There are 25 samples missing for 'disgust' in the dataset. For NN with hand-crafted features, the reported precision for 'disgust' class is lower than the others. Such problem is resolved in CNN models. However, due to the data-driven nature of CNNs, overfitting is still the main issue when CNNs are trained on a small dataset.

4 Conclusion and Future Work

This paper compares two types of facial expression recognition: NNs with hand-crafted LPQ and PHOG features; CNNs trained in end-to-end manner. Various experiments are done to compare the performance of these two methods.

For NN with hand-crafted features, it is discovered that improper learning rate will hinder network training. By varying the number of hidden units, the optimal choice for this task is found to be 200. To achieve fast convergence, Adam optimizer is better than SGD. Meanwhile, we can observe a noticeable accuracy improvement when using the normalized dataset. Using BDR to remove outliers can improve NN's performance with both unnormalized and normalized dataset. Although the increase is minor because BDR wrongly removes many valid samples and results in a trade-off with the benefit of outlier removal.

In comparison, CNNs can learn features of different scales in different stages of the network. Such features are more robust to inputs' diversity, enabling CNNs to give better FER accuracy. For a small dataset, transfer learning is an effective approach to obtain a well-performing model. In our SFEW dataset, fine tuning Resnet50 gives the best result among all experimented models. However, overfitting still remains an issue when transferring large network to small datasets.

Since in our experiments BDR removes valid samples as outliers, future work may adjust BDR's outlier decision criterion, so that it can also work well when the outliers are not significantly distinguishable from valid samples. When training the CNNs, we use a simple augmentation method by randomly flipping the images. Future researchers may investigate effective augmentation approaches to enlarge the training set and hence reduce overfitting. Another point is that we transfer models trained from ImageNet, which is not closely related to our SFEW dataset. Further work may pretrain CNNs on large-scale facial recognition datasets and transfer to SFEW dataset to achieve better performace.

References

1. Dhall, A., Goecke, R., Lucey, S., Gedeon, T. (2011, November). Static facial expressions in tough conditions: Data, evaluation protocol and benchmark. In 1st IEEE International Workshop on Benchmarking Facial Image Analysis Technologies BeFIT, ICCV2011.
2. Slade, P., & Gedeon, T. D. (1993, June). Bimodal distribution removal. In International Workshop on Artificial Neural Networks (pp. 249-254). Springer, Berlin, Heidelberg.
3. De, A., Saha, A. and Pal, M.C., 2015. A human facial expression recognition model based on eigen face approach. Procedia Computer Science, 45, pp.282-289.
4. Donato, G., Bartlett, M.S., Hager, J.C., Ekman, P. and Sejnowski, T.J., 1999. Classifying facial actions. IEEE Transactions on pattern analysis and machine intelligence, 21(10), pp.974-989.
5. Hunke, M. and Waibel, A., (1994, November). Face locating and tracking for human-computer interaction. In Proceedings of 1994 28th Asilomar Conference on Signals, Systems and Computers (Vol. 2, pp. 1277-1281). IEEE.
6. Siddiqi, M.H., Ali, R., Idris, M., Khan, A.M., Kim, E.S., Whang, M.C. and Lee, S., 2016. Human facial expression recognition using curvelet feature extraction and normalized mutual information feature selection. Multimedia Tools and Applications, 75(2), pp.935-959.
7. Pantic, M. and Patras, I., 2006. Dynamics of facial expression: recognition of facial actions and their temporal segments from face profile image sequences. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 36(2), pp.433-449.
8. Xie, S. and Hu, H., 2017. Facial expression recognition with FRR-CNN. Electronics Letters, 53(4), pp.235-237.
9. Zhang, Z., Lyons, M., Schuster, M. and Akamatsu, S., (1998, April). Comparison between geometry-based and gabor-wavelets-based facial expression recognition using multi-layer perceptron. In Proceedings Third IEEE International Conference on Automatic face and gesture recognition (pp. 454-459). IEEE.
10. Liu, K., Zhang, M. and Pan, Z., 2016, September. Facial expression recognition with CNN ensemble. In 2016 international conference on cyberworlds (CW) (pp. 163-166). IEEE.
11. Li, Y., Zeng, J., Shan, S. and Chen, X., 2018. Occlusion aware facial expression recognition using CNN with attention mechanism. IEEE Transactions on Image Processing, 28(5), pp.2439-2450.
12. Siddiqi, M.H., Ali, R., Khan, A.M., Park, Y.T. and Lee, S., 2015. Human facial expression recognition using stepwise linear discriminant analysis and hidden conditional random fields. IEEE Transactions on Image Processing, 24(4), pp.1386-1398.
13. Chakrabarti, D. and Dutta, D., 2013. Facial expression recognition using eigenspaces. Procedia Technology, 10(4), pp.755-761.
14. Lyons, M.J., Budynek, J. and Akamatsu, S., 1999. Automatic classification of single facial images. IEEE transactions on pattern analysis and machine intelligence, 21(12), pp.1357-1362.
15. Li, S.Z., Hou, X.W., Zhang, H.J. and Cheng, Q.S., (2001, December). Learning spatially localized, parts-based representation. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001 (Vol. 1, pp. I-I). IEEE.
16. Shan, C., Gong, S. and McOwan, P.W., 2009. Facial expression recognition based on local binary patterns: A comprehensive study. Image and vision Computing, 27(6), pp.803-816.
17. Bosch, A., Zisserman, A. and Munoz, X., (2007, July). Representing shape with a spatial pyramid kernel. In Proceedings of the 6th ACM international conference on Image and video retrieval (pp. 401-408).
18. Geman, S., Bienenstock, E. and Doursat, R., 1992. Neural networks and the bias/variance dilemma. Neural computation, 4(1), pp.1-58.
19. Joines, J.A. and White, M.W., (1992, June). Improved generalization using robust cost functions. In [Proceedings 1992] IJCNN International Joint Conference on Neural Networks (Vol. 3, pp. 911-918). IEEE.
20. Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556..
21. Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
22. Ranzato, M.A., Huang, F.J., Boureau, Y.L. and LeCun, Y., 2007, June. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In 2007 IEEE conference on computer vision and pattern recognition (pp. 1-8). IEEE21. Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
23. Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
24. He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
25. Pytorch Resnet: https://pytorch.org/hub/pytorch_vision_resnet/
26. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. and Fei-Fei, L., 2009, June. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.