Facial Emotion Classification Based on CNN with Bidirectionality

Jiaxu Liu, Wenbo Ge, and Ruikai Cui

Australian National University, Canberra, Australia
{u6920122, Wenbo.Ge, u6919043}@anu.edu.au

Abstract. The SFEW database contains unconstrained facial expressions close to real world. In former research, current machine learning techniques are not robust enough for this uncontrolled environment [2]. Based on the state-of-art model which achieved the best performance in SFEW, we proposed an augmentation method of adding bidirectionality to CNN based on Tom's bidirectional neural network prototype [1], which is the first to combine these two models in literature. We also conducted an experiment of applying decision fusion CNN framework on classifying by training the networks in parallel simultaneously forward and averaging their respective output. In this paper, two algorithms of adding bidirectionality to CNN are proposed, a framework of FER module(ensemble of HOG face detector and CNN with decision fusion and bidirectionality) is introduced and the classification result is listed, compared, and analysed. The final experiment results affirmed that the bidirectional boosting achieves good performance on SFEW benchmark. Furthermore, some future works for precision improvement based on the existing deficiency of current model are presented.

Keywords: SFEW· CNN· Facial emotion recognition· HOG· Decision fusion· Bidirectional· BDNN· Neural network· Classification

1 Introduction

For classification problems, we can apply both traditional machine learning classifiers and deep learning methods including various neural network structures. Generally, these methods perform well on lab-controlled data which means they tend to have high accuracy in prediction. However, expression analysis in close to real world situations is a non-trivial task and requires more sophisticated methods at all stages of the approach.

Nowadays, deep learning techniques have made significant improvement in classification problems comparing to traditional classifiers like SVM or RandomForest. For most cases, CNN(Convolutional Neural Networks) has shown better performance than other kinds of neural networks. The SFEW(Static Facial Expressions in the Wild) database [2] was created by extracting frames from emotional movie clips in the AFEW(Acted Facial Expressions In The Wild) [6] data corpus. Our task of classification was to assign 7 expression labels, namely angry,

disgust, fear, happy, sad, surprise, and neutral to these frames in close-to-real world conditions.

In this paper, we apply data augmentation method for classification similar to Jeon Jinwoo, et al [9] to original SFEW database. Previous research by Tom, et al had demonstrated in their paper that prototypes of BDNN(Bidirectional Neural Network) [1] may be used to enhance the learning and generalisation ability of neural networks, it was also clearly presented what the topology of BDNN is like and how it can be trained by a generalised error back-propagation to provide the capabilities of label classification. Therefore, based on this prototype we propose a method of adding bidirectionality to the fully connected layer of CNN. We also conducted experiments on the decision fusion framework for CNN proposed by Kim, et al [10]. Finally, evaluation toward these models are carried out.

The rest of the paper is organized as follows: Section 2 the method of data preprocessing, augmentation and our CNN architecture is introduced, we also proposed algorithms that combine BDNN with CNN. In Section 3, the experiments and results are presented, and in Section 4 we give the conclusion.

2 Method

2.1 Data Preprocessing & Face Detection

The SFEW database contains unconstrained facial expressions, varied head poses, large age range, occlusions, varied focus, different resolution of face and close to real world illumination [2] that are extracted from AFEW database. The SFEW contains 700 images in 7 classes including Angry, Disgust, Fear, Happy, Neutral, Sad and Surprise, each 100 images per class. Histogram of Oriented Gradient (HOG), which is introduced by Dalal and Triggs in 2005, is invariant to illumination or geometric transformation as facial expression descriptor.

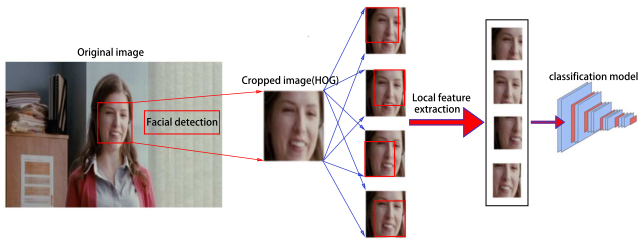

Fig. 1. Data Preprocessing and Augmentation

In our method, we first apply HOG face detection and get the cropped avatar, the image is resized to a resolution of 52x52, we perform data augmentation afterward and get four 48x48 for each cropped image(hence the input size of CNN is 3@48x48 where 3 denotes the RGB channels), thus the dataset is extended by

4 times. We also perform random rotation and flipping on input images during training to enhance the generalisation.

Before inputing into classification model, we do zero-mean normalization(one of the common methods of data pre-processing) for the extended dataset with mean of [0.485, 0.456, 0.406] and standard deviation of [0.229, 0.224, 0.225]. The process above will transform image set into gaussian distribution, i.e. for each RGB channel, $image = (image - mean)/std$. Asccording to convex optimization [12], a set of data that conforms to a specific data distribution is easier to obtain the generalized effect after training.

2.2 CNN Model Architecture

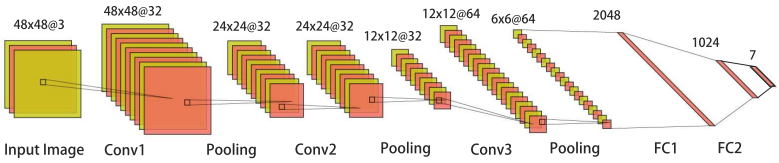

Fig. 2. Our Deep CNN Architecture

The architecture of deep CNN is demonstrated in Figure.2, the deep CNN consist multiple convolutional, pooling and fully-connected layers, each convolutional layer extract features from previous layer and the extracted features are pooled to get spatial invariance.

There are totally three convolutional layers with max-pooling(or average-pooling) layers and two fully-connected layers in our CNN. As its demonstrated in Figure.2, the CNN takes a 48x48 size image as input. The output of last pooling layer is stacked into fully connected layer, and the output FC layers is processed via softmax function which generate the propensity toward 7 classes respectively. To reduce overfitting, dropout layers are introduced, in FC1 and FC2, with dropout probability equal to 0.4. We also introduced batch normalisation, for FC1, the momentum for dynamic mean and std is set to 0.5.

2.3 Bidirectional Neural Network Prototype

A BDNN could be either implemented independently, or merging two mirror FNNs. The point is, the weight for each epoch in training the first network(naming 'model1') is reused in training the second reversed network(naming 'model2'), while inversing the weights(if the weight isn't a square matrix, we use pseudo inverse), and vice versa. The topology of BDNN is demonstrated in Figure.3.

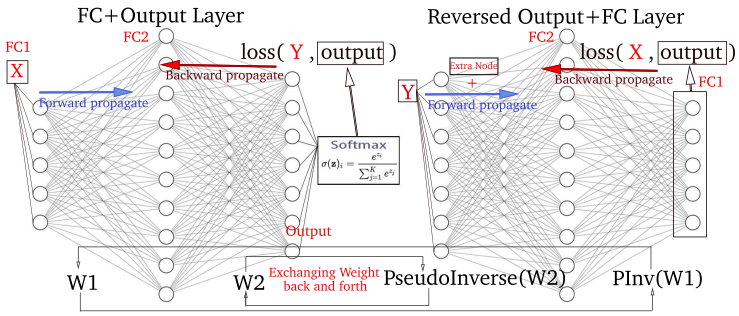

Fig. 3. Topology of Bidirectional Neural Network

More specifically, assume X as the input data ,Y as the output data we want to predict, and $\hat{Y}$ as the predicted data, we optimise the distance between Y and $\hat{Y}$. $\hat{Y}$ could be represented as follows:

$$\hat{Y}_j = \theta_2(\sum W_{ij}^2 S_i) = \theta_2(\sum_j W_{ij}^2 \theta_1(\sum_k W_{ki}^1 X_k)) \tag{1}$$

After completed one forward&backward propagation in model1, for model2, we load Y as an input, note that for SFEW dataset, the label should be encoded and feed into 7 neurons, the strategy we use is quite simple: if label = i, we feed value "1" to the ith neuron, and the others are all fed with "0". Thus reversely, for $\hat{X}$

$$\hat{X}_j = \theta_2(\sum (W^1)_{ij}^{-1} S_i) = \theta_2(\sum_j (W^1)_{ij}^{-1} \theta_1(\sum_k (W^2)_{ki}^{-1} Y_k)) \tag{2}$$

This time, we optimise the distance between X and $\hat{X}$. After the model2 is trained, we then shift the weights of model2 to model1. Back and forth, we train the whole model within both input and output in the dataset.

Note that before we train the network, pre-processing of data is essential in order to make the mapping $f(X) = Y$ and $f^{'}(Y) = X$ one-to-one, which means the mapping should be invertible and thus data should be pre-processed in specific way(In paper [1], 'extra node' is introduced). An extra node will be obtained via function f in later sections.

2.4 Decision Fusion CNN Framework

The Decision Fusion method used in Hierarchical Committee CNN framework is first proposed by Kim et al. in their paper [10] as validation-accuracy-based exponentially-weighted average method. The method described how the outputs of multiple CNNs(hierarchical CNN committee) on the same input are averaged, the weights(significance) of each CNN in the ensemble framework is decided by exponential parameter.

Suppose totally n models are in the framework, their validation accuracy are denoted as k_i where $i \in \{1...n\}$, their output vector(1x1@7) are denoted as v_i where $i \in \{1...n\}, v \in R^7$. We represent the final averaged decision v_{final} as:

$$v_{final} = \frac{\sum_{i=1}^{n}(k_i)^q v_i}{\sum_{i=1}^{n}(k_i)^q} \tag{3}$$

$$PredictedClass = argmax_q\left(\frac{\sum_{i=1}^{n}(K_{unit_i})^q V_{unit_i}}{\sum_{i=1}^{n}(k_{unit_i})^q}\right) \tag{4}$$

where the exponent q is a hyperparameter to determine the significance respective models have. In general, the greater q is, the more specific a model is emphasized. The overall framework is demostrated in Figure.4.

2.5 Two ways of adding bidirectionality to the sample CNN

Compare Kim's method to other methods, i.e. the ensembled decision fusion CNN on SFEW dataset, it is already the state-of-art performance. In our approach, we seek possibilities of combining bidirectional prototype with CNN could further improve the performance, thus in this section, we propose algorithms that add bidirectionality to a sample CNN's fully connected layer.

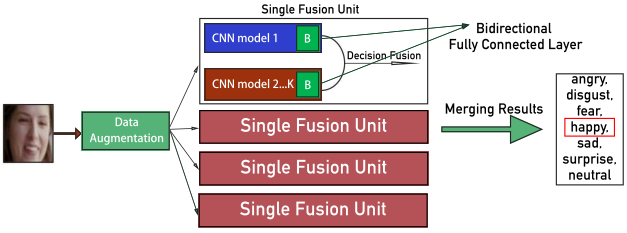

Fig. 4. Decision Fusion Framework with Bidirectional CNN

Intuitively, there are two ways to combine a BDNN with CNN:

1. Train CNN with forward&backward propagation and simultaneously train the fully connected layer like what we did in BDNN, i.e. treating the final stacked output of convolution layers at each epoch as the input of BDNN. Using this method, we will only need to perform the overall training once.
2. Treating the BDNN as a "boosting" method, train the fully connected layer bidirectionally after all the training of CNN is completed. Compared to method 1, this method separate the training of CNN and again its fully connected layer, so overall the fully connected layer is trained twice and we treat final stacked output of convolution layers at final epoch of CNN training as the input of BDNN. In other words, as long as the training for CNN is finished, we will no longer modify it but use it as a feature extractor.

For the first intuition, we propose the following algorithm: In our CNN architecture, three layers(fully connected layers + output layer) are extracted after the back propagation process is completed, afterward, three fully connected layers of CNN in the order of [output→fc2→fc1] are reversely applied to FNN for weight update. For instance, if the expected output is 'happy', we feed in 'happy' into the reversed FNN and generate a 2048(the size of FC1) dimension vector, we compute the cross-entropy loss between the output of FNN and the exact value vector of FC1, then back propagation is proceeded and weights are updated, finally, we assign the updated weight back to the fully connected layers of CNN. The detailed algorithm is presented, naming Algorithm.1 in the appendix.

For the second intuition, we propose the following algorithm: This intuition is simpler to implement in practice. The key point is that we train two models, respectively a CNN and a BDNN one after another but not simultaneously. The detailed algorithm is presented, naming Algorithm.2 in the appendix.

Here are some comments for the two algorithms listed in the appendix: To begin with, FC1 and FC2 denotes the fully connected layers of CNN, OUT denotes the output layer. An m_0 has two outputs, if we feed an image I to m_0, then $m_0(I)_0$ denotes the final output of CNN and $m_0(I)_1$ denotes the middle output(Equivalent to the input of FC layer), i.e. the flattened image after convolution. Function f that computes the value of extra node should be properly defined, we use $mean()$ in our approach. An m_1 denotes a temporary forward neural network, which have the same structure as the FC layer of m_0 but could be in different direction, this FNN could be create as a new one, or directly use the reversed FC layers of CNN model. For all inversion process, we use pseudo inverse instead of general inverse because the weight matrices are not invertible. Thus $w^{-1} \leftrightarrow pinv(w)$ in this case.

3 Experiment and Discussion

In this section, we first compare the difference between two intuition of implementation, see which one is better in performance, we will also compare our decision fusion CNN model to different methods including Deep learning based unconstrained FER models and traditional machine learning classification model. The experiments demonstrate the curve of training loss & validation accuracy of our model for each epoch on classifying seven expression classes.

3.1 Experiments

We first run experiments on comparing two intuitions mentioned above. Recalling section 2.5, based on intuition 1, the CNN and Bidirectional FC layer are trained simultaneously. For intuition 2, we train the CNN firstly, and followed by an extra step that trains Bidirectional FC layer as a 'boosting' for whole model. In our experiment, we perform 500 epochs on CNN training and for intuition 2, we perform an extra 100 epochs training on bidirectional FC layer.

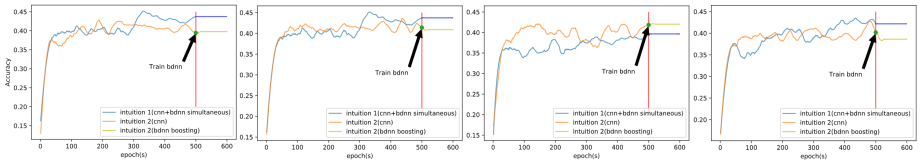

Fig. 5. comparison of the accuracy of intuitions on validation set

Figure.5 demonstrates the accuracy of the two intuitions on validation set where he red line splits the before and after of training bdnn on intuition 2, from these curves we observe that the performance between intuitions does not differ much while for most cases, training models simultaneously(intuition 1) tend to have better accuracy in testing. In the later experiments on comparing different models, we use intuition 1 for all implementation because its overall more computational cheap and provide similar performance to intuition 2.

In our second experiment, we evaluate the performance of single&fusion CNN model with&without bidirectional boosting. To implement a fusion unit, we need to train n multiple separate models(two in our approach), namely from m_1 to m_n, in our approach, we use CNN-max-pooling(m_1) and CNN-average-pooling(m_2) as an example and trains simultaneously with same input image set and combine the prediction via fusion method. Note that this structure could be easily modified by switching the exemplified CNN to different classifiers or adding more image processing nets for decision fusion. The hyperparameter q, mentioned above for decision fusion, is decided by enumerating different values and selected for which provides the maximum performance.

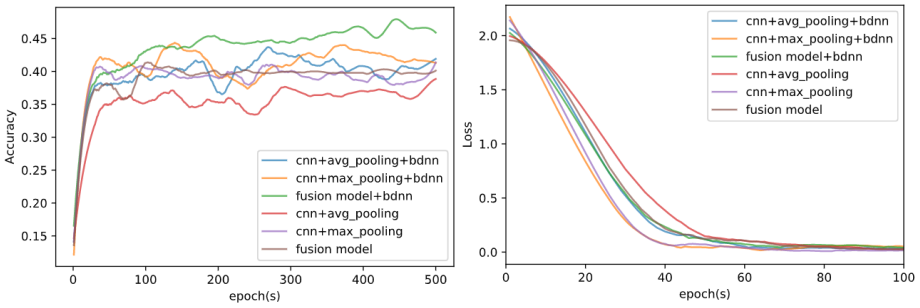

Fig. 6. Validation Accuracy(left) & Training Loss(right, x-axis cut off for clarity) on different model with averaged 5-fold cross validation

We use a 5-fold cross validation with the learning rate of model equal to 0.001 initially with a decay rate of 0.95 every 20 epochs after 50 epochs are done. Further more, we use the batch gradient descent technique with batch size equal to 64, and for all convolution and penultimate layers, we use Rectified

Linear Unit (ReLU) activations for the nonlinearity, finally a softmax activation is applied to the output layer.

The Figure.6 shows the curve of training validation accuracy and the training loss variation on different models and different combinations. We observe that for all models with BDNN boosting, they outperform their original model in most cases. We apply exponential decision fusion to the fusion model with a uniform search of q value in [-150,150], the final accuracy is around 45.21% (Figure.6 green curve), which performs the best among these models. Compared to the 35.93% baseline accuracy of PHOG+LPQ+Linear SVM method provided by the creator of the SFEW dataset [2], it is a significant improvement.

3.2 SPI Baseline for Benchmarking

Based on the SPI protocol, we compute the baseline scores. In Table.1, classwise Precision, Recall and Specifity results of fusion model on the SFEW database is demonstrated, compared to the SFEW paper's accuracy, this result shows a large increment in performance.

Table 1. Classwise Precision, Recall and Specifity results on fusion model

Emotion	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Precision	0.46	0.33	0.5	0.6	0.21	0.5	0.4
Recall	0.38	0.22	0.63	0.6	0.23	0.7	0.33
Specificity	0.89	0.97	0.91	0.88	0.95	0.91	0.94

Also, we provided performance evaluation of several different models in Table.2, the result shows that the bidirectional model somewhat outperforms the original ones, the best bidirectional model with decision fusion network reached a accuracy of 45.21s%, hence we can conclude that the bidirectional model is useful in emotion recognition. In the confusion matrix, the bidirectional models are more uniform in correct prediction(diagonal elements compared to values in same rows and columns) than the other two simple models, which demonstrates that the fused network with bidirectionality boosting could have more balanced weight and better comprehension in features of each classes instead of particular ones.

4 Conclusion and Future Work

The paper proposed algorithms that combine BDNN with CNN architecture and run experiments on ensemble decision fusion framework of CNN that can be generalised into various models. The model focuses on solving the real-world facial emotion recognition task, and its the first to combine BDNN to real-world FER system, the accuracy of our models outperform the baseline model in average.

Our Models	Validation accuracy(%)	From Literature	Validation accuracy(%)
CNN(avg pooling)	38.89	PHOG+LPQ+SVM	35.93
CNN(max pooling)	41.33	Levi's LBP+CNN	44.73
CNN(fusion model)	40.97	Kim's CNN	52.50
CNN(max pooling)+BDNN	41.56		
CNN(avg pooling)+BDNN	42.12		
CNN(fusion)+BDNN	45.21		

Table 2. Comparison of performance(validation accuracy) on different models

In accordance with Tom et al. in their paper [1], bidirectional architecture could potentially enhance the capability of CNN and reduce generalisation error, while in our evaluation, the efficiency of CNN+bidirectional is shown to be more time-consuming while we also observed improvements on performance, the extra node in this paper is generated via mean function, which is probably not optimal for this classification problem and still need further exploration and investigation.

Due to the limitation of computational resource, the entire framework (Figure.4) or larger network is not evaluated in this paper, thus in the future, attempts on different kernel size and number of hidden activations will be carried out. Nevertheless, the variety on the constitution of fusion unit will theoretically raise the performance, thus we plan to combine different feature extractor(for instance, SIFT) as well as different advanced image processing networks(for instance, VGGNet and ResNet) with the fusion unit for better performance. Further more, the data preprocessing method could be improved by making use of landmark detection and different kinds of alignment methods [11], which will also be focused in our future research.

References

1. Nejad, A. F., and T. D. Gedeon. "Bidirectional neural networks and class prototypes." Proceedings of ICNN'95-International Conference on Neural Networks. Vol. 3. IEEE, 1995.
2. 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.
3. Hecht-Nielsen, Robert. "Theory of the backpropagation neural network." Neural networks for perception. Academic Press, 1992. 65-93.
4. C.-C. Chang & C.-J. Lin. LIBSVM: A library for support vector machines, 2001.
5. Ciregan, Dan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification." 2012 IEEE conference on computer vision and pattern recognition. IEEE, 2012.s
6. A. Dhall, R. Goecke, S. Lucey, and T. Gedeon. Acted Facial Expressions in the Wild Database. In Technical Report, 2011
7. V. Ojansivu and J. Heikkil. Blur Insensitive Texture Classification Using Local Phase Quantization. In Proceedings of the 3rd International Conference on Image and Signal Processing, ICISP'08, pages 236–243, 2008.

8. A. Bosch, A. Zisserman, and X. Munoz. Representing Shape with a Spatial Pyramid Kernel. In Proceedings of the ACM International Conference on Image and Video Retrieval, CIVR '07, pages 401–408, 2007.
9. Jeon, Jinwoo, et al. "A Real-time Facial Expression Recognizer using Deep Neural Network." International Conference on Ubiquitous Information Management and Communication ACM, 2016:94.
10. Kim, Bo-Kyeong & Lee, Hwaran & Roh, Jihyeon & Lee, Soo-Young. (2015). Hierarchical Committee of Deep CNNs with Exponentially-Weighted Decision Fusion for Static Facial Expression Recognition. 10.1145/2818346.2830590.
11. V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1867-1874, doi: 10.1109/CVPR.2014.241.
12. Boyd, Stephen, Stephen P. Boyd, and Lieven Vandenberghe. Convex optimization. Cambridge university press, 2004.

Appendix

Algorithm 1 Algorithm based on the first intuition

Input: CNN model m_0, Image I
Output: CNN model

Extend the output layer of m_0 by 1, denote as $ExtraNode$.
for all *epochs* **do**
  Forward&Backward propagation on m_0, update weights(Input: I)
  $w_{1,2,3} \leftarrow$weight(FC1,FC2,OUT)
  $ExtraNode \leftarrow f$(FC1)
  $m_1 \leftarrow$ InSize(OUT); Hidden1Size(FC2); Hidden2Size(FC1); OutSize($m_0(I)_1$)
  Assign w_3^{-1} to W(Input-Hidden1) of m_1(denote as w'_3), w_2^{-1} to W(Hidden1-Hidden2) of m_1(denote as w'_2), w_1^{-1} to W(Hidden2-Output) of m_1(denote as w'_1)
  Forward&Backward propagation on m_1, update weights(Input: $GroundTruth + ExtraNode$)
  $w_{3,2,1} \leftarrow {w'}^{-1}_{3,2,1}$
end for
return m_0

Algorithm 2 Algorithm based on the second intuition

Input: CNN model m_0, Image I
Output: CNN model

Extend the output layer of m_0 by 1, denote as $ExtraNode$.
for all *epochs* **do**
  Forward&Backward propagation on m_0, update weights(Input: I)
end for
$w_{1,2,3} \leftarrow$weight(FC1,FC2,OUT)
$ExtraNode \leftarrow f$(FC1)

$m_1 \leftarrow$InSize($m_0(I)_1$); Hidden1Size(FC1); Hidden2Size(FC2); OutSize(OUT)

Assign w_1 to W(Input-Hidden1) of m_1(denote as ${w'}_1$), w_2 to W(Hidden1-Hidden2) of m_1(denote as ${w'}_2$), w_3 to W(Hidden2-Output) of m_1(denote as ${w'}_3$)

for all *epochs* **do**

  $w_{1,2,3} \leftarrow$weight(FC1,FC2,OUT)

  Forward&Backward propagation on m_1, update weights(Input: $m_0(I)_1$)

  $m_1 \leftarrow$Reverse(m_1)

  Forward&Backward propagation on m_1, update weights(Input: $GroundTruth + ExtraNode$)

  $m_1 \leftarrow$Reverse(m_1)

end for

Replace the FC layer of m_0 by m_1

return m_0