Bidirectional Neural Network and Convolutional Neural Network Used for Faces-Emotion Multi-Classification

Pengqi Wang

Research School of Computer Science,
The Australian National University, Canberra, Australia

U6688509@anu.edu.au

Abstract. In this big data era, a large number of image data promotes the computer vision and neural network development. In this experiment, I will implement a Simple Neural Network, two kinds of bidirectional neural networks (BDNN) and a Convolutional Neural Network (CNN) for multi-classification task based on a specific face emotion dataset, the Static Facial Expressions in the Wild (SFEW) dataset. The classification accuracy results for Simple NN and BDNN are just a little better than random guess and they are totally worse than SFEW dataset paper results [3]. CNN classification accuracy results are almost the same as SFEW dataset paper results. Thus, CNN is better than Simple NN and BDNN for SFEW dataset multi-classification problem. And many detailed and complex works need to be done in the future for CNN based on SFEW face emotion dataset.

Keywords: Face-emotion. SFEW dataset. Feed-forward neural network. BDNN. Autoencoder. CNN. Confusion matrix.

1. Introduction

With the development of neural network and computer vision, it turns out to be possible and very useful to analyze human facial expression images. Face expression can respond face emotion, such as happy, angry, sad and so on. It becomes a more and more popular research area in computer vision. There are many facial expression databases and SFEW dataset is chosen for our experiment. It is closer to our real-world face emotion so that the results can be used for real-world situation research. My task is to classify different face images to the right label emotion classes. It is a multi-classification problem. In this experiment, I will implement a simple feed-forward multi-classification neural network, two kinds of bidirectional multi-classification neural networks (BDNN) and a Convolutional Neural Network (CNN) based on SFEW dataset. The Simple neural network and BDNN will use Local Phase Quantization (LPQ) and pyramid of histogram of oriented gradients (PHOG) descriptors of SFEW image dataset. The CNN will use the raw image dataset. Classification accuracy and confusion matrix are the measure methods to evaluate classification performance. In this experiment, I will compare classification performances for different models (simple NN, BDNN and CNN) based on SFEW dataset. Many performance improvements and parameter fine tunings are also included in this experiment.

1.1 SFEW dataset

SFEW is the Static Facial Expressions in the Wild database, which is a static database selected frames from the temporal database Acted Facial Expressions in the Wild (AFEW) [4]. Images in SFEW are of unconstrained facial expressions, different head poses, large age range, various facial resolution and very close to real world illumination. SFEW facial expression database is close to real world situation compared with other facial expressions databases. Since most facial expressions databases have been recorded in lab-controlled environments, it is better to research on SFEW, which can represent real world conditions more likely. For example, it contains natural head movements and illumination and so on. The dataset contains 7 labels: angry, disgust, fear, happy, sad, surprise and the neutral class. For LPQ and PHOG descriptors dataset used for simple NN and BDNN, they are from 1 to 7. In the experiment, they need to be adjusted from 0 to 6 so that they can be used for Softmax classification problems. For raw images used for CNN, the 7 labels need to be labeled from 0 to 6 first. For the dataset used for simple NN and BDNN, there are 675 image samples and for each image sample, it is represented as 2 descriptors (5 columns for each descriptors). One image sample with 5 naN values, and I turn it to 0 so that it can be used for further training. The inputs for neuron network are 10 columns and the first five columns are Local Phase Quantization (LPQ) descriptors and the last five columns are pyramid of histogram of oriented gradients (PHOG) descriptors. Both of these two descriptors have shown good performance in object recognition, such as invariant to blur and illumination and so on [1]. Thus, it can be a good representation for raw images. For simple NN and BDNN, at the end, I will pick 346 at random for training, and use the rest for testing the classification accuracy.

For CNN, I choose raw image of SFEW dataset as the input. There are 100 angry images, 75 disgust images, 100 fear images, 100 happy images, 100 neutral images, 100 sad images and 100 surprise images. They are all (288,360,3) color images. I first resize them to (256,256,3) and then implement image whitening as the data augmentation methods to improve the robust of the model.

1.2 Define a problem and outline of the investigations

In my experiment, I will research on the classification accuracy for classifying different face-emotion images to 7 emotion label classes based on SFEW datasets. Then I will compare these different models or techniques to find a suitable one for SFEW datasets here. It is a popular area in computer vision, and it is worth researching because it is related to image recognition. There are 4 neural networks built in this experiment for multi-classification problem, which are simple NN, 2 kinds of BDNNs, CNN. The simple NN is just a fully connected neural network. 2 kinds of BDNNs are based on the technique paper [1] and an autoencoder paper [2]. In the dataset paper, it said that the classification accuracy for SFEW is 43.71% for LQ and 46.28% for PHOG. First, I will construct the first simple fully connected neural network and evaluate the performance of classification on SFEW dataset. As for evaluation measure, accuracy and confusion matrix are used. Then, build Bidirectional neural networks (BDNNs) and use them to perform classification on SFEW dataset again. The same measure methods are used. For BDNNs, I implement them in two ways, and they will be discussed later. The input neurons of the neural network are 10. It means for each image sample, all features whether LPQ descriptors or PHOG descriptors become the inputs to the neural network. There are 7 output neurons. By that I mean models will learn the most likely label from the max values of the 7 neurons. For convolutional neural network (CNN), I will use raw images of SFEW dataset as the inputs and in the end, there is a fully connected neural network for classification. Also, accuracy and confusion matrix are used for evaluating performance. Finally, there is a comparison among the simple neural network classification performance, bidirectional neural networks (BDNN) classification performance, convolutional neural network (CNN) classification performance and dataset paper classification performance to judge which technique or model is more suitable for SFEW dataset here. Moreover, there are many ways to improve the performances of different models and they will be discussed later.

2. Method

2.1 Description of BDNN and Autoencoder as an BDNN technique

As inspired by electrical transmission bidirectional, we can use bidirectional neural networks to solve neural network classification problems. If we allow input to output connections, it is possible that more accurate features could be extracted. By that I mean we can use back propagation in both forward and reverse direction to learn weight matrix of the network [5]. However, more epochs are necessary for converge compared with the traditional forward networks [6]. We can train BDNNs as Cluster center finders. It can also be represented that if we train BDNNs between input and output patterns, it can find the representative input vector in the hidden layer. While the traditional forward neural network cannot achieve it. And the representative input vector can enhance the generalization and learning ability of the neural network when we use it for a classification or prediction task followed by a fully connected neural network [1]. We can achieve the same goal by using autoencoder plus a feed-forward classification neural network. In figure 1, you can see the structure of autoencoder plus a classification neural network (the missing arrows are nothing to do with our experiment).

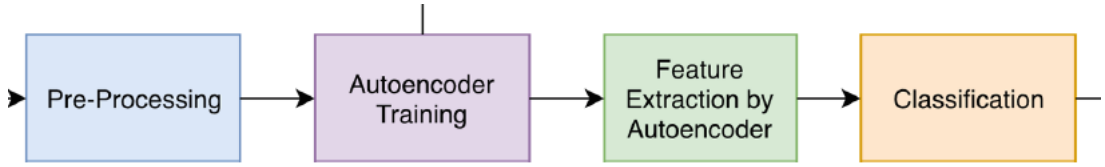

Figure 1. structure of autoencoder + classification [2]

Autoencoder is a good way for both feature learning and visualization. Autoencoder is a sort of compression algorithm, and then un-compress to get the original input. It can achieve the same idea as Bidirectional neural network. An Autoencoder can be divided into 2 parts, an encoder and a decoder. The objective of the Autoencoder is to minimize the difference between input and output (learn output more similar to input) to make the extracted features more robust and enhanced in the center hidden layer. And the encoder part, extracted robust features from the Autoencoder, can be used

for classification followed by a fully connected neural network. Due to treating the robust features from the raw images as a new input to a fully connected classification neural network, it means that more robust image features and less dimensional, so that it can achieve better classification accuracy results.
Autoencoder can deal with many problems. There will be a large amount of inputs sometimes, such as high-quality image. It is hard to learn from such a big input size. Autoencoder can find meaningful features from a large amount of input sizes and also it reduces the dimensions. The encoder part can get the core features from the input and make it easier to perform further learning task(classification/prediction).
I use accuracy and confusion matrix to evaluate bidirectional net performance.
For SFEW dataset, I replace the nan values with 0 and make the label classes from 0 to 6 rather than from 1 to 7 in order for further classification training. I use 10 input neurons for all features and 7 output neurons for seven classes. I implement standard normal distribution to generate initial weights. I use Sigmoid function, batch size=32, Adam as optimizer, cross entropy and MSE loss function, learning rate=0.005, epochs=1000, hidden layers=3, hidden neurons=30.

2.2 Bidirectional net as an BDNN technique

To be honest, this bidirectional net is just a reference of BDNN technique, and it is just used for a double check whether BDNN technique is suitable for SFEW datasets after implementing autoencoder. Inspired by Bidirectional net structure (see Figure 2), I can build 2 networks as for pytorch it is hard to learn and back-propagation in both directions. The first one is for feed-forward learning, back-propagation. The second one is for backward learning, feed-forward-propagation. They are for the same neural network structure but use opposite input output sizes. The input size of the second network is the same size as the output of the first network. The output of the second network is the same size of the input as the first network. Train on the first network several epochs, and then give the trained weights to the second network. Train on the second network several epochs, and then give the trained weights to the first network. Finally, give the trained weights from second network to the first network. After that, we get a well-trained bidirectional neural network. Performing test sets on the tuned first network. It is the direct bidirectional neural network however trained 2 networks actually. And I will not go further about the bidirectional net as it is just a reference to make a double check that whether BDNN technique is suitable for SFEW datasets.
I use accuracy and confusion matrix to evaluate bidirectional net performance.
For SFEW dataset, I replace the nan values with 0 and make the label classes from 0 to 6 rather than from 1 to 7 in order for further classification training. I use 10 input neurons for all features and 7 output neurons for seven classes. I use Sigmoid function, MSE loss function, Adam as optimizer, epochs=500, batch size=16, learning rate =0.005, hidden neurons=30.

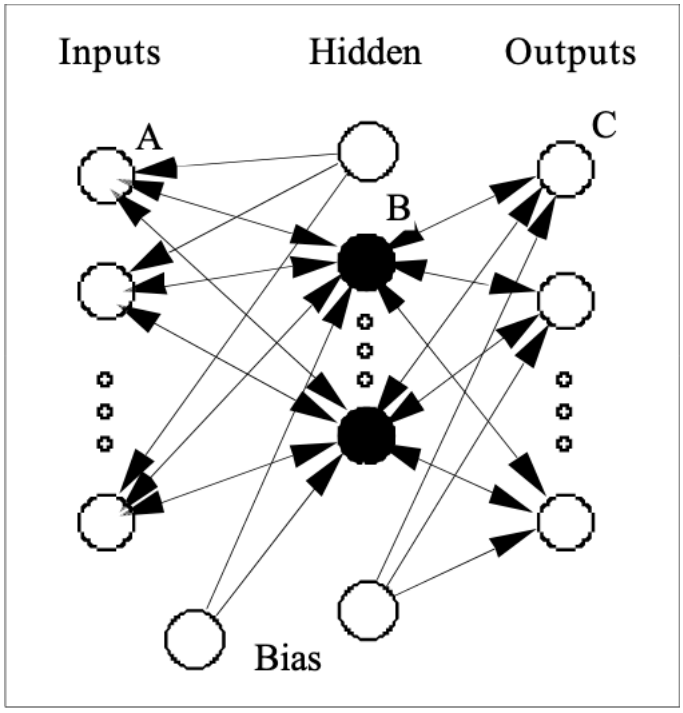

Figure 2. Bidirectional Net Structure [1]

2.3 Convolutional neural network

With the development of big data and improvement of GPU speed, convolutional neural network (CNN) has a high popularity these days. CNN is inspired by the visual pathway and it is used as feature extractors for a deep classification neural network [9]. By that I mean you can think that a CNN is comprised of convolutional layer, pooling layer and fully connected layer. Convolutional layer is the core building block of a CNN. It has a set of learnable filters, which is convolved with the local receptive fields of input data to produce a feature map. Moreover, stride length controls how far local receptive field slides. Padding adds dummy data to the input so that it allows spatial dimensions to be maintained after convolutional layer operation. In figure 3, you can see how filters convolved with input data to generate feature maps. The number of filters decides the number of feature maps. Then, the output of the convolutional layer can be taken to an activation function, the same as the simple neural network.

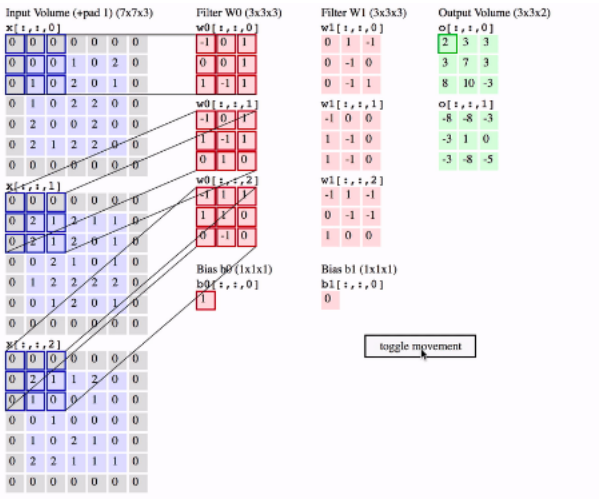

Figure 3. multi-channel convolutions [9]

Next, the results can be fed into the pooling layers. It can down-sample feature maps from the previous convolution layer into a condensed version which is a summary statistic. To be specific, pooling layer is useful for reducing network parameters, size and computational costs [11]. It can also help to prevent overfitting. The most common methods of pooling are max-pooling and average-pooling. There is a fully connected layer at the end of CNN. It is used for classification. In figure 4, you can see the overall structure of CNN. For CNN, it shares filters so that it is an easy job to deal with high dimensional data. It is no need to choose features manually. The only thing you need to do is training filters and weights.

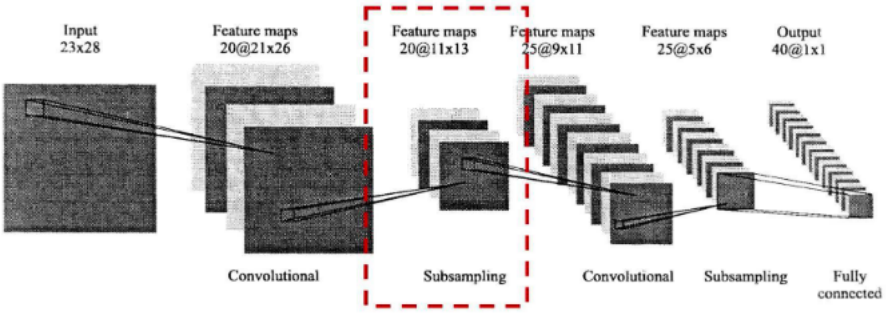

Figure 4. CNN structure [9]

For CNN model, I use SFEW raw images as the input data in this experiment. I implement data augmentation such as resize and image normalization, also mini batch for the input dataset. I also use dropout and batch normalization for CNN. The output size for each convolutional layer is the same as the input to this convolutional layer and the output size for each max pooling layer is the half of the input to max pooling layer. So, for a (3,256,256) input image, after convolutional layers and max pooling layers, there is a (128,16,16) output features. Then feed them to a fully connected neural network with 2 hidden layers, the output is 7 neurons for 7 classes. Optimizer is Adam, loss function is cross entropy, batch size=16, learning rate=0.001, epochs=30. Finally, I use accuracy and confusion matrix to estimate the classification performance.

3. Results and Discussion

I used confusion matrix and accuracy to evaluate the classification performance for these different models. I split SFEW dataset into training set, validation set and testing set. (detailed information of how I split training, validation and testing sets can be seen in part 3.1 and part 3.2.) In Table 1, there are the results for different models. The first two lines are the results from SFEW dataset paper [3]. From this table we can get that based on SFEW dataset, Simple NN, BDNN and CNN are all performing better than random guess as there are 7 classes. However, Simple NN and BDNN (Bidirectional net and Autoencoder + FC) are just above random guess.

Classification accuracy for LPQ:	43.71%
Classification accuracy for PHOG:	46.28%
Simple NN classification accuracy:	20.07%
Bidirectional net classification accuracy:	24.36%
Autoencoder + FC classification accuracy:	25.03%
CNN classification accuracy:	**45.47%**

Table 1. Classification accuracy for different models

Then I had the confusion matrix for BDNN and Simple NN in Figure 5. From here we can know that for some classes, the model can classify most images of this class to the right class. It is better than random guess or not trained model. However, for some classes, it is just like a random guess. I think it means that for these classes, these models have not learned how to classify. For simple NN, it may because of the structure of neural network is too simple to learn valuable weights. For BDNN, it is better to have a one-to-one relation between input and output vectors [1] and we have 10 input and 7 output. It maybe not suitable to use BDNN for SFEW dataset here. Comparing BDNN classification accuracy & simple NN classification accuracy with SFEW dataset paper results, I can say they are not good neural network models for SFEW dataset here.

```
Confusion matrix for BDNN:
tensor([[ 9.,  9., 10.,  8.,  4.,  8.,  4.],
        [ 6.,  8.,  1.,  2.,  4., 11.,  3.],
        [ 5.,  6., 18.,  4.,  2., 11.,  5.],
        [ 4.,  4.,  1., 22.,  4., 12.,  3.],
        [ 2.,  5., 11.,  7.,  7.,  6.,  6.],
        [ 8.,  4.,  4.,  9.,  8., 12.,  2.],
        [ 5.,  9., 15.,  6.,  1.,  9.,  5.]])
```

```
Confusion matrix for Simple NN:
tensor([[13.,  2.,  1.,  6., 11.,  7.,  4.],
        [10.,  2.,  3.,  5., 12.,  8.,  4.],
        [18.,  0.,  8.,  8.,  7.,  4.,  7.],
        [ 8.,  1.,  2., 14.,  8.,  9.,  5.],
        [13.,  1.,  1.,  9., 11., 10.,  5.],
        [12.,  0.,  5.,  5.,  9., 11.,  4.],
        [10.,  3.,  4.,  5.,  7.,  8.,  9.]])
```

Figure 5. Confusion matrix for BDNN and Simple NN

In Figure 6, it shows the confusion matrix for CNN. For most of the classes, it can classify them to the right classes. And in Table 1, CNN can get 45.47% classification accuracy. It is almost the same as SFEW dataset paper results. Thus, I believe that I build a quite good CNN model for SFEW dataset. It is a good way to use raw images as the input to CNN. And detailed discussion can be seen in part 3.2.

```
Confusion matrix for CNN:
tensor([[12.,  2.,  4.,  2.,  5.,  4.,  3.],
        [ 3.,  8.,  1.,  1.,  1.,  1.,  2.],
        [ 4.,  0., 14.,  1.,  2.,  9.,  2.],
        [ 1.,  1.,  2., 10.,  4.,  5.,  4.],
        [ 2.,  1.,  1.,  5., 11., 11.,  4.],
        [ 2.,  1.,  2.,  0.,  4., 19.,  3.],
        [ 1.,  4.,  3.,  4.,  5.,  5.,  7.]])
```

Figure 6. Confusion matrix for CNN

In general, from the results of this experiment, I can say this CNN model is quite suitable for SFEW dataset multi-classification problem. Although Simple NN and BDNN are better than random guess, they are not suitable for SFEW dataset multi-classification problem here.

Part 3.1 and Part 3.2 are detailed experiment information for BDNN and CNN. If you are interested in how I build my models and how I try to improve my models, you can see detailed information here.

3.1 Detailed experiment of BDNN

First, for SFEW dataset, to be honest, I used 5 input neurons for LPQ/PHOG inputs initially the same idea as the dataset paper showed, but I found if I use 10 input neurons for all features, I could get a lower loss and a higher accuracy.
I split SFEW dataset into 2 sets first, picking 346 at random for training set and the rest for testing set as required. Then I split 346 samples training set into 2 parts, 3/4 samples for training and 1/4 samples for validation. I split training set into training set and validation set, as I wanted to tune hyperparameters of the model in validation set to improve the performance as well as made the test set unseen to make the performance more general. Then I implemented 5-fold cross validation on training set and validation set. In this way, I could use training set for training network, validation set for tuning parameters. Once I thought that I have found the best parameters, I combined training set and validation set, making it become the final training set and test on testing set to see the performance in order to make training data as many as possible.
I first implemented autoencoder model with the same hyperparameters as my lab experiments because they have similar numbers of samples and are both built for multi-classification problems. Now, I found that I got a high loss values, low accuracy on training set and also low accuracy on validation set. From that, I thought I was in under-fitting situation. I would not use preventing overfitting skills such as dropout and regularization at that time. First, I guessed that it is due to the default random weight initialization, so I implemented standard normal distribution to generate initial weights. Although loss value was still very high, it helped to improve training accuracy. (For further parameters tuning, it proved that it was necessary to initialize weights rather than default random initialization.)
Then I tried many different activation functions and used ReLU at first. Many neurons would 'die' during training with ReLU activation functions [8]. Thus, I tried to increase my hidden neurons to avoid this problem. It was useful and I got a higher test set accuracy. (Finally, I used Sigmoid function as I thought it was better than ReLU.)
Then I thought that my model was too simple so that it cannot learn from the training set efficiently, and I added 2 more hidden layers and there were 3 hidden layers. It helped a little on validation accuracy. I also tried different batch size. And I decided to use batch size=32 because it gave me lower training set loss values than other settings. I tried many different optimizers and finally I decided to use Adam. I should also mention that the best learning rate was 1e-3. Either higher magnitude or lower magnitude would lead to worse results. Then I implemented a large number of epochs and I found I could get a high accuracy and low loss value on training set. However, low accuracy on test set. Now I thought I was out of the under-fitting situation. I was in the situation of overfitting. Then, I implemented L2 regularization and tried to prevent overfitting. However, still low accuracy on testing set. I also tried dropout method. It is no use.
I tried many different settings of parameters. Whether I could get low performance on training set or high performance on training set (it may overfitting), I cannot get a well performance on test set. The best classification accuracy for BDNN is only about 25%.

3.2 Detailed experiment of CNN

First, I just implemented resizing images to (256,256,3) for raw images of SFEW dataset because it could be better for both classification and running time. Later, I realized it is necessary to take data augmentation technique as there were not many image samples. I read in images and marked labels for each image.
I split raw image dataset into 2 sets, picking 30% at random for testing set and the rest for training set. Then I split training set into 2 parts, 3/4 samples for training and 1/4 samples for validation. I split original training set into training set and validation set, as I wanted to tune hyperparameters of the CNN model in validation set to improve the classification performance as well as to make the test set unseen causing the performance more general. Then I implemented 5-fold cross validation on training set and validation set. In this way, I could use training set for training network, validation set for tuning parameters. Once I thought that I have found the best parameters, I combined training set and validation set, making it become the final training set and then, test on testing set to see the performance in order to make training data as many as possible.
first, I implemented a simple version CNN as a baseline to carry out this multiple classification problem. It has 4 convolutional layers and a fully connected layer with 2 hidden layers. Convolutional layers are used for features extraction and fully connected layer is used for classification. Each feature extraction operation has a convolutional layer, a ReLU function and a max pooling layer. The convolution kernel size is (5,5), stride is (1,1), padding is (2,2). The max pooling kernel size is 2, stride is 2. By that I mean for each convolutional layer, the input size is the same as the output size and for each max pooling layer, the output size is half of the input size. For 4 convolutional layers, we get from 3 channels to 16 channels, from 16 channels to 32 channels, from 32 channels to 64 channels, from 64 channels to 128 channels. (If it is not clear, you can see my code comments and it is clear enough.) Then flat it and feed it a to fully connected layer with 2 hidden layers. The first hidden layer has 1024 neurons and the second hidden layer has 128 hidden neurons. Finally, it has 7 neurons in output layer. I chose Adam as optimizer and cross entropy as loss function. Adam is a popular improvement of gradient descent method and cross entropy is suitable for multiple classification problem. Learning rate is 0.001. if it is too large it is hard to find optimization and if it is too small it takes a long time to train. Epoch is 20 and batch size is 64. I could not get a satisfied classification accuracy even on training set.

Then I used data augmentation technique. By that I mean I used image whitening. It can have effect on dataset and make the model more robust [13]. It makes loss more stabilized and it can also improve accuracy. Next, I implement batch normalization. It uses normalization for each layer in neural network. It can make dataset more uniform, which cannot lead to all neurons are activated or all neurons are not activated [14]. Then I found I can get a nearly 100% classification accuracy on training set however, only about 30% classification accuracy on validation set.
Next, I added dropout layers. During training, dropout layer sampled weight layer parameters in 80% probability. It is used for fully connected layers, ignoring 20% feature detectors and reduce their interactions [15]. After that, I could get about 40% classification accuracy.
Finally, I used batch size=16, epoch=30. Overall, I could achieve about 45% classification accuracy at most.

4 Conclusion and Future Work

In this experiment, I carry out a multi-classification problem based on SFEW face emotions dataset. There are 4 models, Simple neural network, Bidirectional net, Autoencoder + fully connected neural network (the same idea as BDNN) and Convolutional neural network. LPQ and PHOG descriptors SFEW datasets are used for Simple NN and BDNN and raw images SFEW datasets are used for CNN. The classification accuracy for all models is better than random guess. The results of Simple NN and BDNN are just a little better than random guess. And the classification results of CNN are almost the same as SFEW dataset paper classification results. Thus, I can draw a conclusion that CNN is more suitable than Simple NN and BDNN for SFEW datasets multi-classification problem.
Although BDNN is also a good technique, I think it is better to improve CNN in the future and I think the classification accuracy can outperform the SFEW dataset paper results in the future. In the future, it is better to use a variational learning rate in CNN. By that I mean the learning rate can be bigger at first, when validation accuracy increases slowly, decrease learning rate. Secondly, you can build a deeper CNN to learn more features since my CNN model has just 4 convolutional layers. By that I mean ResNet is a good way to implement. It is easy to optimize and helps to solve gradient vanishment. Moreover, you can try pre-train and fine-tuning based on the existed models. It is also a good idea to use evolutionary algorithm to train the model parameters. It is worthwhile to implement all above methods for computer vision and neural network areas.

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/(1995), pp. 1322-1327 vol.3.
2. Yao, Yue, Jo Plested, and Tom Gedeon. 'Deep Feature Learning and Visualization for EEG Recording using Autoencoders', Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11307/(2018), pp. 554-566.
3. Dhall, Abhinav, Roland Goecke, Simon Lucey, et al. 'Static Facial Expression Analysis in Tough Conditions: Data, Evaluation Protocol and Benchmark', (2011), .
4. A. Dhall, R. Goecke, S. Lucey, and T. Gedeon. Acted Facial Expressions in the Wild Database. In Technical Report, 2011. 1, 2
5. Rumelhart, DE, Hinton, GE and Williams, RJ “Learning internal representations by error propagation,” in Rumelhart, DE, McClelland, “Parallel distributed processing,” Vol. 1, MIT Press, 1986.
6. Xiao, Min, Wei Xing Zheng, and Jinde Cao. 'Hopf Bifurcation of an (n+1) -Neuron Bidirectional Associative Memory Neural Network Model with Delays', IEEE Transactions on Neural Networks and Learning Systems, vol. 24/no. 1, (2013), pp. 118-132.
7. Masci, J., Meier, U., Cireşan, D., Schmidhuber, J.: Stacked convolutional auto-encoders for hierarchical feature extraction. In: Honkela, T., Duch, W., Girolami, M., Kaski, S. (eds.) ICANN 2011, Part I. LNCS, vol. 6791, pp. 52–59. Springer, Heidelberg (2011).
8. Eckle, Konstantin, and Johannes Schmidt-Hieber. 'A Comparison of Deep Networks with ReLU Activation Function and Linear Spline-Type Methods', Neural Networks, vol. 110/(2019), pp. 232-242.
9. Jo Plested. CONVOLUTIONAL NEURAL NETWORKS. Retrieved from https://wattlecourses.anu.edu.au/pluginfile.php/2284668/mod_resource/content/2/DL Convolutional Neural Networks.pdf
10. Tom, Connectionist compression. Retrieved from https://wattlecourses.anu.edu.au/pluginfile.php/2221568/mod_resource/content/3/NN_compr.pdf
11. Hosseini, S., Lee, S.H., Kwon, H.J., Koo, H.I. & Cho, N.I. 2018, "Age and gender classification using wide convolutional neural network and Gabor filter", IEEE, , pp. 1.
12. Cai, C., Wang, C., Zeng, Y., Cai, S., Liang, D., Wu, Y., Chen, Z., Ding, X. & Zhong, J. 2018, "Single-shot T 2 mapping using overlapping-echo detachment planar imaging and a deep convolutional neural network", Magnetic resonance in medicine, vol. 80, no. 5, pp. 2202-2214.
13. Lemley, J., Bazrafkan, S. & Corcoran, P. 2017, "Smart Augmentation Learning an Optimal Data Augmentation Strategy", IEEE Access, vol. 5, pp. 5858-5869.
14. Wu, S., Li, G., Deng, L., Liu, L., Wu, D., Xie, Y. & Shi, L. 2019, "L1 -Norm Batch Normalization for Efficient Training of Deep Neural Networks", IEEE Transactions on Neural Networks and Learning Systems, vol. 30, no. 7, pp. 2043-2051.

15. Mendenhall, J. & Meiler, J. 2016, "Improving quantitative structure–activity relationship models using Artificial Neural Networks trained with dropout", Journal of Computer-Aided Molecular Design, vol. 30, no. 2, pp. 177-189.