Classification Optimization Analysis for Facial Expression Recognition from SFEW with Artificial Neural Network

Zhewen Li, u6437091

U6437091@anu.edu.au

Abstract.
Facial expression recognition (FER) is an interesting rather complex problem in artificial intelligence. With the great development in deep learning, growing numbers of well-functioned algorithms have been proved to have achieved excellent performance [1]. Although FER research fields had received a lot of achievements [2], [3], the outcomes of FER in real-world practice are still relatively undesirable. This paper focuses on implementing an alternative neural network approach intended to improve the accuracy for classifying emotions based on the feature data that have been extracted in [4]. Further discussion will include a step-by-step optimization elaboration over the neural network models.

1 Introduction

Artificial Neural Network (ANN) is one of the most popular models in dealing with FER problems, since ANN is capable of pattern recognition and classification of image data [5]. For the past few decades, there have been many successful approaches for improving the performance of the ANN models, but due to the complexity of ANN itself, the problem of ANN model optimization is still challenging in facial expression recognition. It is obvious that the majority of existing methods for multi-class classification consider themselves as a regression issue, where models are trained to fit a binary sequence, and each digit indicates the existence of its corresponding class [6]. The class to which the highest value corresponds is the class in which the input value resides. However, studies have shown the limitation of such criterion since the value of 0.5 as a threshold to distinguish between two classes is not sacrosanct either [7]. Hence, further improvement towards the threshold control is one of the most effective way for model optimization.

The thresholding refers to the technique of image thresholding which is one of the most powerful techniques for image segmentation because of its simple and stable performance. It can be used to distinguish object and background pixels in a digital image [8]. Therefore, setting a threshold and adjusting its scope to make the edges of each class to appear clearer is especially useful in multi-class problem classification.

In this paper, I will present an experiment of trained neural network model which has different criterion combined with a gradual optimization process based on the extracted data from the facial expression images that are established from the previous research [4]. Besides, a comparison between the performance of the network model I used and the network with thresholds control will be discussed in detail as well. In order to get a comparable outcome, I implement the ANN model following the relevant pre-request defined in [4] so as to find the optimal performance that can be achieved under this condition.

2 Dataset

Previous studies have shown that, most FER approaches work well in the well-controlled databases but are usually invalid for the real-world expression [9], as human expressions vary a lot from natural condition to facing the camera. Therefore, I adopted the SFEW database to do experiments upon defined in the previous study, namely Static Expression in the Wild, which is developed by selecting frames from the temporal dataset Acted Facial Expressions in the Wild (AFEW), consisting of close to real world environment extracted from movies. [4] Therefore, the robustness of the trained model to unknown data can be ensured.

The dataset to be used in the experiment will be the top 5 features of an image extracted from [4] with respect to feature extraction methods. Therefore, there will be 675 ×5 image features for a single descriptor in total as the input.

3 Descriptors for Feature Extraction: LPQ & PHOG

Local phase quantisation (LPQ) and pyramid of histogram of oriented gradients (PHOG) are two descriptors that has great performance in extracting features. They're also the descriptors applied in [4]. Also, both of them have been extensively used for static and temporal facial expression analysis which have shown good performance in object recognition as well [4].

4 Preprocessing

Before the actual training, the dataset needs to be preprocessed. To get a better view of the dataset. I split the data into two separate datasets, as the data are extracted by two different algorithms to be compared with each other later. Next, I drop the columns which are irrelevant to the problem, meanwhile, handling the missing value in the dataset by replacing them with the mean value of its column. Then, I normalised the data with standard scalar in sklearn, which transform the distribution of input features to have a mean=0, while standard deviation=1. One of the most important thing is to split dataset into train and test set, so as to maintain the robustness of the model, which is also a way to see if our model can tolerate such random ('wild') data that are unknow to itself.

For splitting dataset, I choose 80% of the overall data as the training set, and the rest of 20% are for testing cases after an optimal model is built. Similarly, I used inbuilt function in sklearn for splitting the dataset. As the model should be modified multiple times for adjusting the parameter to find the optimal setting, a validation set is also required for this purpose. Therefore, I split out another 20% of the training set as a validation set. In general, the distribution of the dataset is showed in table 1.

	Proportion (%)	**Number (features)**
Training set	64	432
Validation set	16	108
Test set	20	135
Total	**100**	**675**

Table 1. Distribution of the dataset after train_test splitting

5 Network Models & Evaluations & Result

A neural network is characterized by (i) Pattern of connections between the neurons (architecture). (ii). Method of determining the weights (iii). Activation function [10]. Based on these instructions, the initial network model I used is a simple modification of multi-layer perceptron (MLP).

```
NN_MLP(
  (input): Linear(in_features=5, out_features=3375, bias=True)
  (hidden1): Linear(in_features=3375, out_features=675, bias=True)
  (hidden2): Linear(in_features=675, out_features=96, bias=True)
  (output): Linear(in_features=96, out_features=7, bias=True)
)
```

It is a 4 layer fully connected neural network with backpropagation method to update weights automatically. In order to increase sparseness, an elementwise activation function rectified linear units (ReLUs) layer can be applied after any convolutional layer. The ReLU layer deals also with the vanishing gradient problem in the error backpropagation phase.[11] Since the problem to be solved is a multi-class classification, which cannot be solved by a definite score of the output to predict the class. Therefore, the model I built will choose the intended class based on the highest probability in seven labels.

After the training process, an output with 7 dimensions will be returned. Each row indicates 7 probabilities towards seven categories. Fig 1 shows a general structure of the network model. There will be 675 × 7 = 3375 neurons in the input layer, 675 neurons in the first hidden layer, 96 neurons in the second, and 7.

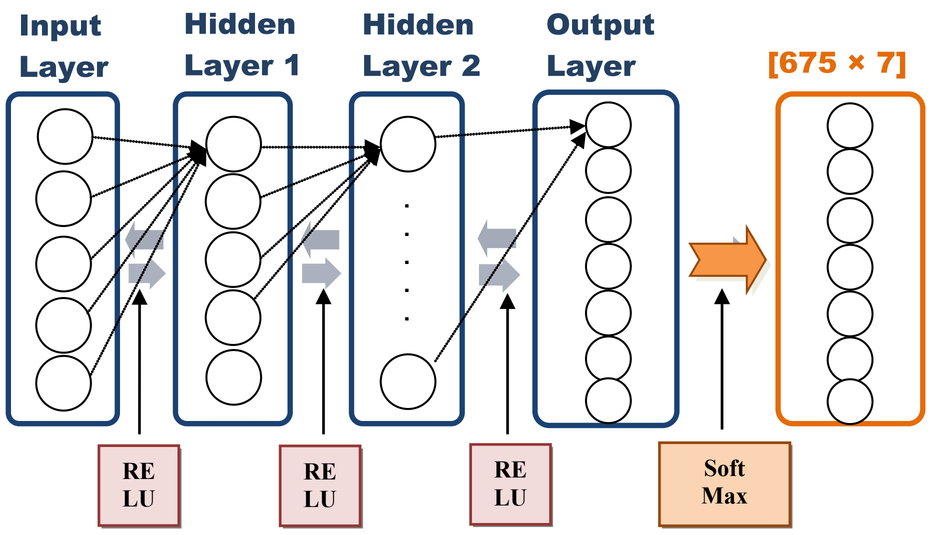


Fig. 1. Initial ANN Structure: 4-Layer NN with RELU

The method is basically focused on the improvement of backpropagation. When training, there are a couple of hyper parameters that should be discussed. For the performance of BP, it is related to the batch size, learning rate, and optimizer. In my following experiments, I will adjust these parameters to find the optimal combination.

I used cross entropy loss as loss function, as it can describe the probability distribution over classes of each input which represent the category as well. For optimizer, I choose SGD at this stage. Starting from a large learning rate to see how the loss will be, then use a standard LR to see the difference.

The evaluation method I used is a manually generated function which after matching with the label, then compute the accuracy and loss with the test set.

The result for classification is showed by train/validation/test accuracy/loss with respect to LPQ and PHOG using the network above is showed in Table 2 below.

Optimizer = SGD, loss_func= CrossEntropyLoss(), LR = 0.6, No_Batch, EPOCH = 1

1st	Train Accuracy	Train Loss	Validation Accuracy	Validation Loss	Test Accuracy	Test Loss
LPQ	0.134	1.947	0.148	1.948	0.148	1.949
PHOG	0.153	1.947	0.167	1.945	0.126	1.930

Optimizer = SGD, loss_func= CrossEntropyLoss(), LR = 0.2, No_Batch, EPOCH = 1

2nd	Train Accuracy	Train Loss	Validation Accuracy	Validation Loss	Test Accuracy	Test Loss
LPQ	0.148	1.945	0.157	1.947	0.126	1.947
PHOG	0.148	1.946	0.157	1.949	0.148	1.947

We can see that without batch learning, the accuracy and loss for train, validate and test set are quite similar. Despite the fact that the result is not good, it is still hard to distinguish which parameter should be adjusted. Besides, the weight is updating in a real slow speed. As the currently most widely used training algorithm for large-scale ML tasks is called Mini-batch Stochastic Gradient Descent (MGD), which improves upon the simplest but powerful Stochastic Gradient Descent (SGD) method. [12] For further experiment, I applied mini-batch gradient descent. In this way, the training set is divided into many small batches, and the losses is calculated for each batch. Therefore, the updating speed of weights will be increased a lot (compared to previous updating after the whole dataset is trained)

6 Optimization

6.1 Model based optimisation:

After several times of experiments, it seems that my model is very likely to overfit as the training loss and validation loss shows no tendency to drop. Therefore, in order to avoid overfitting, I add a dropout layer to automatically drop 20% of the neurons in the network which will lead to overfitting.

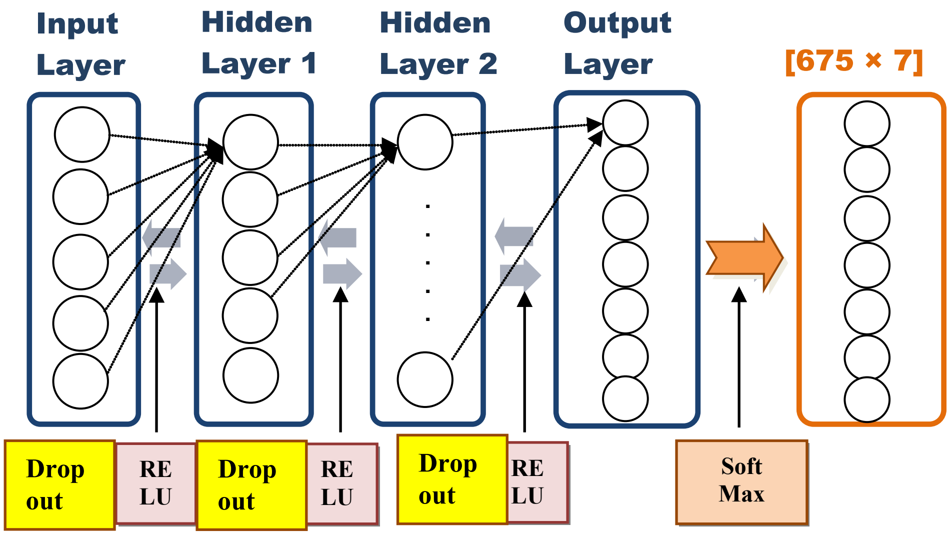


Fig. 2. Final ANN Structure: 4-Layer NN with RELU & Dropout

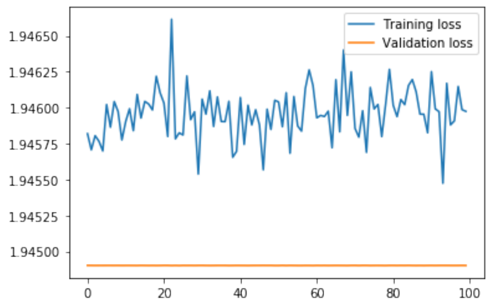
The network structure is showed below, and the following experiments will based on this model.

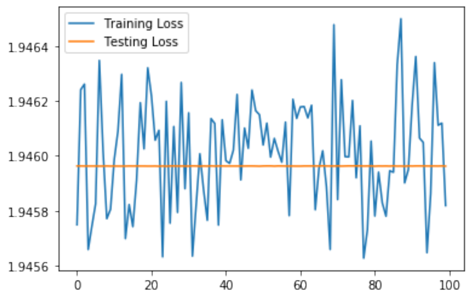
```
NN_MLP(
  (input): Linear(in_features=5, out_features=3375, bias=True)
  (hidden1): Linear(in_features=3375, out_features=675, bias=True)
  (hidden2): Linear(in_features=675, out_features=96, bias=True)
  (output): Linear(in_features=96, out_features=7, bias=True)
  (dropout): Dropout(p=0.2)
)
```

Optimizer = SGD, loss_func= CrossEntropyLoss(), LR = 0.6, Batch_size = 60, EPOCH = 100

LPQ:

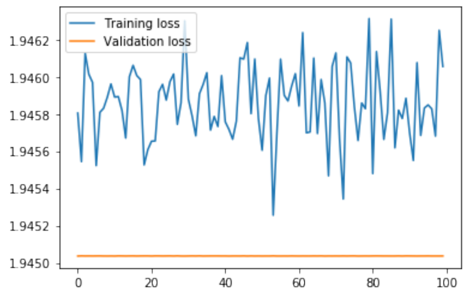
```
Train_loss: 1.946, Train_accuracy: 0.143
validation_loss: 1.945, validation_accuracy: 0.157
Test_loss: 1.946, Test_accuracy: 0.133
```

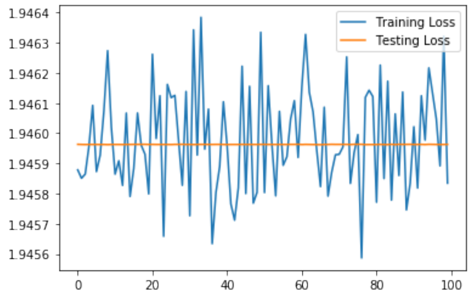




PHOG:

```
Train_loss: 1.946       Train_accuracy: 0.149
Validation_loss: 1.945  Validation_accuracy: 0.157
Test_loss: 1.946,       Test_accuracy: 0.133
```





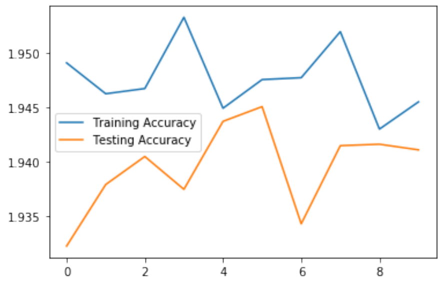
Still setting the initial EPOCH size to very large to make it overfit, so as to see the overall trend of the loss change while set a fast learning rate. We can see that the training loss vary a lot but for validation loss remains no change. Referring to the output it returns, which shows that the accuracy rate is more likely performing random guessing. From the plot we can see that, the data is very likely to be overfitting. By multiple times of on hand experiment, I found that the model should set a very slow learning rate, and the EPOCH size should not be over 10 will get a relative stable result, but the result is still not desirable.

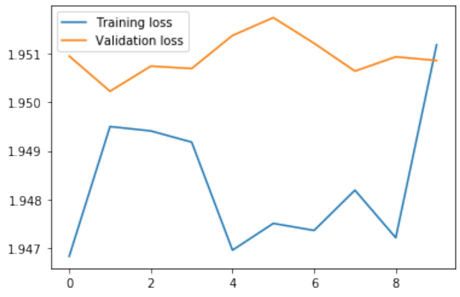
Furthermore, after multiple times of iteration, I find that each model has good performance and bad performance. Thus, I tried to find out the optimal model when training with validation set, and later use this model to fit the test set.

In this case, the optimal result is showed below.

Optimizer = SGD, loss_func= CrossEntropyLoss(), LR = 0.003, Batch_size = 60, EPOCH = 10

```
Average case:
Train_loss: 1.948  Train_accuracy: 0.138
validation_loss: 1.951  validation_accuracy: 0.148
Test_loss: 1.940  Test_accuracy: 0.202
```





7 Thresholding

For the technique of thresholding in [7], I implement based on the previous model. The key point of the thresholding lays at distinguishing boundaries when two classes are very close. In my case, I replace the SoftMax function to sigmoid, and then convert the label to One-hot-encoding. After that, I iterate over threshold, give the rage from [0.520, 0.550]

```
encoder = OneHotEncoder()
 output = net(x.float())
 #print(output)
 y_pred = (output > THRESHOLD).numpy()
 y_logit = encoder.fit_transform(y.reshape(-1,1)).toarray()
 y_logit = torch.LongTensor(y_logit)
 y_p = torch.LongTensor(y_pred)
 print(y_logit)
 print(y_p)
```

threshold	Train accuracy	Validation_accuracy	Test_accuracy
No threshold	0.138	0.148	0.202
0.520	0.142	0.144	0.201
0.530	0.163	0.165	0.210
0.540	0.200	0.224	0.214
0.545	0.210	0.220	0.230
0.550	0.220	0.240	0.250

Therefore, the threshold is actually working!

8 Future work

Give a more detailed implementation over the threshold computation. Modified the model more well as the overfitting problem is still very obvious. Meanwhile, the accuracy improved not that obvious, and the model is not stable enough, as each time the result varies quite much. Besides that, maybe I'll try another network, a more complex network for implementation.

Reference:

[1] L. Li, Z. Ying and T. Yang, "Facial expression recognition by fusion of Gabor texture features and local phase quantization," 2014 12th International Conference on Signal Processing (ICSP), Hangzhou, 2014, pp. 1781-1784.

[2] Rahulamathavan Yogachandran, Phan Raphael CW, Chambers Jonathon A, Parish David J, "Facial Expression Recognition in the Encrypted Domain Based on Local Fisher Discriminant Analysis, "IEEE Transactions on Affective Computing, vol. 4, no. 1, pp. 83-92. 2013.

[3] Li Y, Wang S, Zhao Y, Ii Q, "Simultaneous Facial Feature Tracking and Facial Expression Recognition," IEEE Trans Image Processing, vol. 22, no. 7, pp. 2559- 2573 . 2013.

[4] Abhinav Dhall, Roland Goecke, Simon Lucey, Tom Gedeon, “Static Facial Expression Analysis in Tough Conditions: Data, Evaluation Protocol and Benchmark”,

[5] George F. Hepner, Thomas Logan, Niles Pitter, Nevin Bryant, “Artificial Neural Network Classification Using a Minimal Training Set: Comparison to Conventional Supervised Classification”, Photogrammetric Engineering and Remote Sensing, Vol. 56, No. 4, pp. 469-473. 1990.

[6] A. Zeggada, F. Melgani, Y. Bazi, "A deep learning approach to UAV image multilabeling", IEEE Geosci. Remote Sens. Lett, vol. 14, no. 5, pp. 694-698, 2017.

[7] L.K. Milne, T.D. Gedeon and A.K. Skidmore, “Classifying Dry Sclerophyll Forest
From Augmented Satellite Data: Comparing Neural Network, Decision Tree & Maximum Likelihood”,

[8] L. Wang, H. Duan and J. Wang, "A fast algorithm for three-Dimensional Otsu’s Thresholding method," 2008 IEEE International Symposium on IT in Medicine and Education, Xiamen, 2008, pp. 136-140.

[9] X. Peng, Z. Xia, L. Li and X. Feng, "Towards Facial Expression Recognition in the Wild: A New Database and Deep Recognition System," 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Las Vegas, NV, 2016, pp. 1544-1550.

[10]

[11] A. Zeggada, F. Melgani and Y. Bazi, "A Deep Learning Approach to UAV Image Multilabeling," in IEEE Geoscience and Remote Sensing Letters, vol. 14, no. 5, pp. 694-698, May 2017, doi: 10.1109/LGRS.2017.2671922.

[12] B. Liu, W. Shen, P. Li and X. Zhu, "Accelerate Mini-batch Machine Learning Training With Dynamic Batch Size Fitting," 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 2019, pp. 1-8, doi: 10.1109/IJCNN.2019.8851944.