CSE 676 Project
Emotion Recognition from Facial Expressions

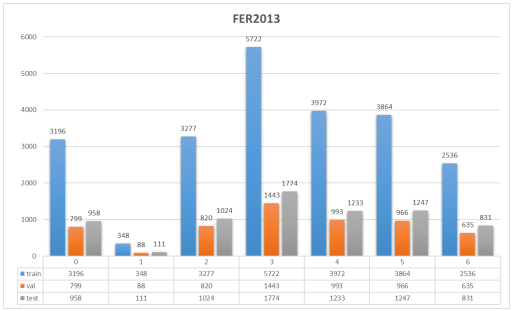
Author(s): Haohua Feng(haohuafe@buffalo.edu), PinKuan Hsieh(pinkuanh@buffalo.edu)

Abstract

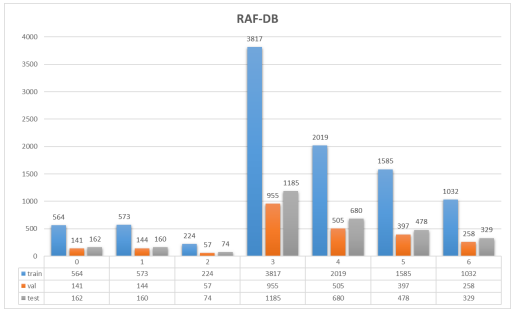
The project's objective is to train a CNN (Convolutional Neural Network) model capable of accurately distinguishing human facial expressions. Our target is to achieve an accuracy as high as possible with trying combination of different approaches. The ultimate aim is to implement this model in user interface design and mental health monitoring applications.

1 Dataset

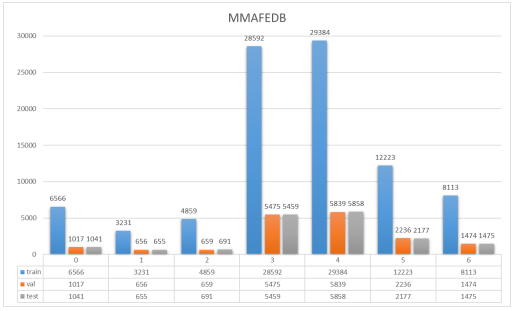
The dataset we used for this project are RAF-DB[1], MMAFEDB[2] and FER2013[3]. All of them are obtain from `www.kaggle.com`. All dataset contains 7 classes, [angry, disgust, fear, happy, neutral, sad, surprise]. Dataset FER2013 and RAF-DB only contain training set and testing set, without validation set. We split the training set into two sets by the ratio of 8 : 2, become the training set and the validation set. Below is the distribution of all 3 dataset:



(a) FER2013



(b) RAF-DB



(c) MMAFEDB

Figure 1: data distribution of all 3 dataset

1.1 Problematic of dataset

Initially we use only the dataset of FER2013, in the first experiment of testing classic model (AlexNet, VGG16), we found out that the test accuracy is not good enough. And, the papers[4][5] we used for reference pointed out there are some images are incorrectly labeled or without a face in it, which leads to lower accuracy, we also proof it in our experiment 1(see Figure.2). Therefore we decide to use additional dataset (RAF-DB, MMAFEDB) to see if we can relief the problem. RAF-DB is another well known dataset, but unfortunately, we contact the provider of RAF-DB for permission to access, we never get reply. We obtain the copy of RAF-DB from kaggle, but it is an in-completed dataset that distributed by other user, missing a lots of samples.
Refer to figure.1, all 3 training set we used have the problem of imbalance. Insufficient amount of samples in some class can lead to low accuracy of predicting those classes. We will try to use data augmentation to deal with imbalance problem.

2 Model Description

In this project, we use four models and the modified versions of them.

- AlexNet
- VGG16
- DCNN (model that use in Paper)
- custom CNN (a custom model by referring models above)

2.1 Experiments to select the best model

The first experiment we done is to find out the baseline by using the classic algorithms (AlexNet, VGG16). In this experiment, we modified AlexNet and VGG16 into different versions, such like changing the activation functions(ReLU, Sigmoid, Tanh), optimizer (SGD-Momentum=0.9, ADAM, AMSGRAD), number of neurons in fully connected layers (256 or 4096). By using the set up in Table.1 with input size of (224*224) for VGG16, and (227*227) for AlexNet, we have the result:

Figure 2: Exp1: validation accuracy baseline of AlexNet & VGG16

According to the result in Figure.2, excluding the worst combination (Alex_256_sigmoid_SGD), after all model gets converge, the lower bound is approximate 58%, and the upper bound is approximate

Table 1: setup of runing baseline experiment on AlexNet and VGG16

dataset(train,test)	learning rate	batch size	loss function	epoch	input size
FER2013	0.0001	32	CrossEntropyLoss	200	Alex(227), VGG16(224)

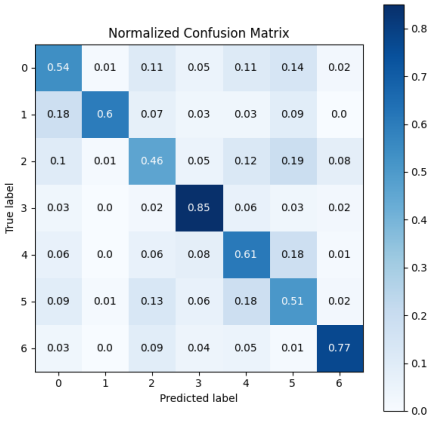


Figure 3: confusion matrix of Alex_256_ReLU_Adam on test set

64%. The best among all is Alex_256_ReLU_Adam, reaches the validation accuracy of 64.6%. The accuracy is not good enough.

From the confusion matrix in Figure.3, apart from classes 3 and 6, the accuracy of predictions of other classes are not good at all. This could be due to the problems of imbalance data distribution and incorrect labeling that pointed out by the paper[4]. Through this experiment, it was discovered that using the ADAM optimizer leads to more efficient model convergence and a marginally higher validation accuracy. Consequently, we have chosen to employ it as the optimizer in subsequent experiments to achieve faster convergence and timely results.

2.2 DCNN Algorithm

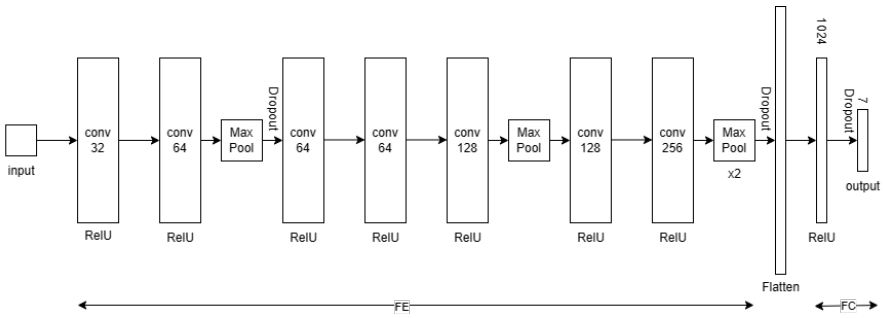


Figure 4: DCNN Algorithm (vanilla, V0)

DCNN is the model we borrow from the paper[4]. Just like the AlexNet and VGG Algorithms, it has two big blocks, Feature Extraction Layers and Fully Connected Layers. We use the same set up in experiment 1 to train and test the accuracy on DCNN.v0, and the result is similar to the result of the two algorithm above. The validation accuracy is about 63%, and the confusion matrix is also similar to Figure.3. In the paper[4], the authors use data augmentation to increase the size of training set, to get a higher test accuracy above 70%. We also tried the same approach by doing data augmentation (rotation, horizontal flip, translate, shear, random crop. They will be discussed more in the later section). The amount of training sample be come 6x the original amount (original + 5 kinds of augmentations). However, we did not able to replicate the result the authors produced, only 67% in validation accuracy.

We also test the DCNN algorithm on RAF-DB, the result is better than FER2013. With 73% on both validation and test accuracy. The confusion matrix (Figure.5) of it looks better than FER2013.

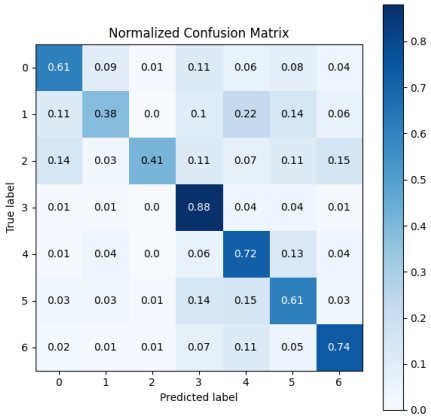


Figure 5: confustion matrix of DCNN train and test on RAF-DB

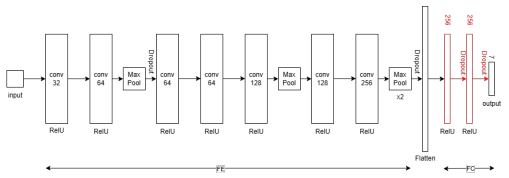
2.3 Conclusion based on the above experiments

1. FER2013 is a challenged dataset, we decide to use RAF-DB for further experiment.
2. To address the issue of low accuracy for certain classes due to imbalanced data distribution, as depicted in Figure5, we do data augmentation on classes 1 and class 2 of RAF-DB, combining data augmentation and the original RAF-DB training set as one to train the model.
3. Using ADAM as the optimizer with learning rate of 0.0001 is a good option.
4. Use early stop mechanism to stop training early if validation accuracy unchanged for 10 epochs. DCNN and custom CNN have larger amount of parameters, use 200 epoch as the standard will waste time on training a model that is already convergence.
5. Depend on the algorithm of DCNN and custom CNN, using the 64*64 as the input size can perform better.

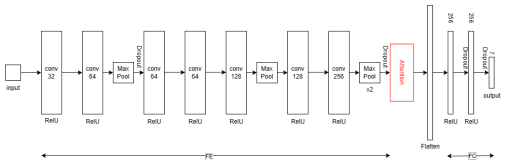
Table 2: setup for further experiments (DCNN, custom CNN)

dataset(train)	learning rate	batch size	loss function	epoch	image size
RAF-DB(aug12)	0.0001	32	cross entropy loss	early stop	64*64

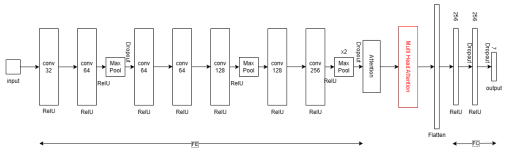
2.4 Modified version of DCNN



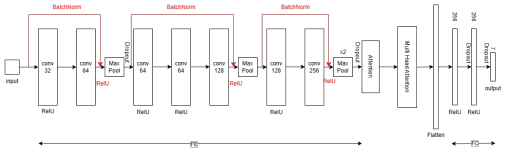
(a) v0+: narrow down neurons in FC, add 1 FC layer



(b) v1: "Squeeze and Excitation Attention" after FE

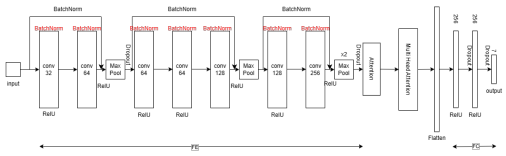


(c) v2: "Multi-Head Attention" after "SE Attention"

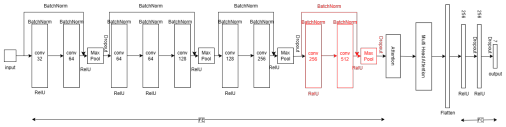


(d) v3: Add Residual Connections in FE blocks

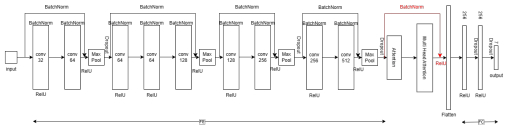
Figure 6: DCNN v0.1 - v3



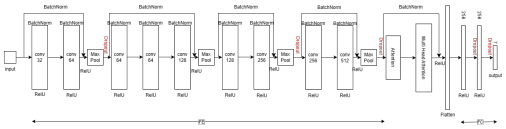
(a) v3+: Add Batch Normalize



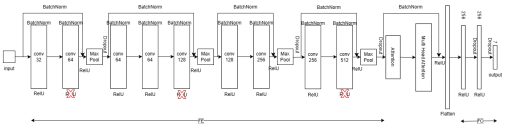
(b) v4: Add 2 more convolution layers with residual connection



(c) v5: Add Residual Connection on Attention Blocks



(d) v6: dropout probability from '0.5' to '0.65'



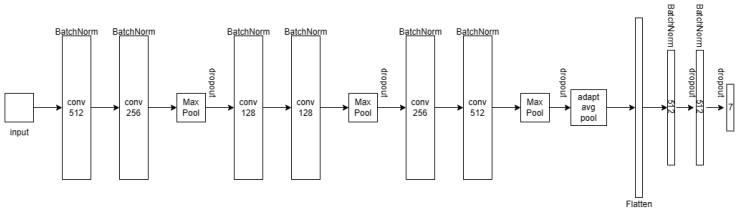
(e) v7: Remove some Activation function

Figure 7: DCNN v3.1 - v7

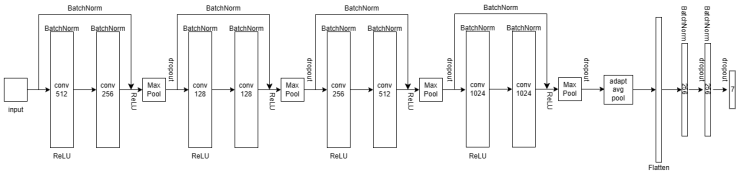
Table 3: Test accuracy of DCNN v0 - v7 (%)

v0	v0+	v1	v2	v3	v3+	v4	v5	v6	v7
75.84	76.76	76.27	75.09	77.80	78.03	78.84	79.85	80.83	79.62

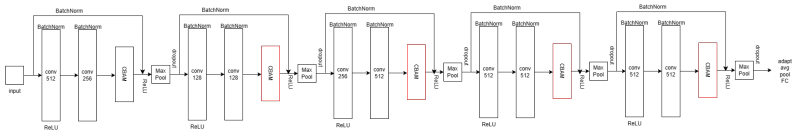
2.5 Custom CNN and its modified versions



(a) v0: vanilla



(b) v7.2: no attention blocks



(c) v8: Add CBAM attention

Figure 8: Custom CNN, v0, v7.2, v8

Similar approach from section 2.4 applied to custom CNN algorithm (Figure.8(a)). There are 2 additional versions have different kinds of structure than structure use in section 2.4. Shown as above ((b)v7.2 and (c)v8).

Table 4: Test accuracy of Custom CNN v0 - v8 (%)

v0	v1	v2	v3	v4	v5	v6	v6.1	v7	v7.2	v8
73.27	74.34	74.28	78.84	77.96	78.65	79.40	80.21	79.69	80.41	79.95

2.6 Best model(s)

The best 3 model from Table.3 and Table.4 is DCNNv6 (80.83%), custom CNNv7.2 (80.41%) and custom CNNv6.1 (80.21%). We also have done experiments by applying other optimization algorithm on these 3 model, custom CNNv7.2 is the best of 3 after applied (see section 4 for more details).

3 Loss Function

The first 3 loss functions came out our heads are mean square error (MSE), negative log likelyhood (NLL), and cross entropy loss (CEL). These 3 we had taught in this course. MSE is not commonly used in classification problem, typically use in regression problem. As for the other NLL and CEL, they can use in classification problem. Before the experiment 1, we've done a little experiment on choosing loss functions. We compare these two loss functions by training the models Alex_256_ReLU_Adam and Alex_4096_ReLU_Adam on dataset FER2013. CEL is just way better than NLL in result (Figure.9). Beside this, in pytorch library, the label smoothing mechanism is

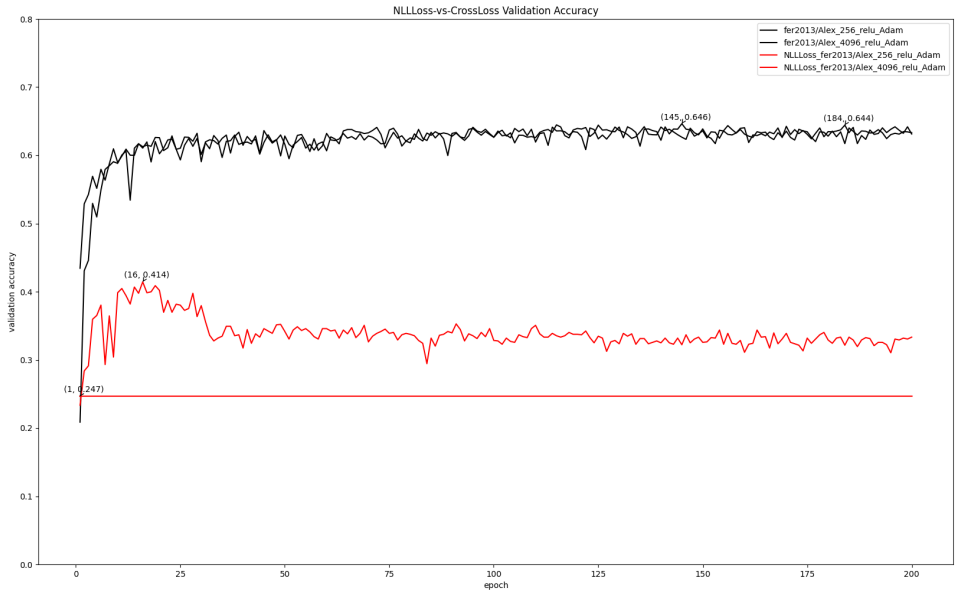


Figure 9: Loss function comparison, NLL vs CEL

available in nn.crossentropyloss(). We also want to try it out to see if there is any improvement by using this mechanism, label smoothing.

4 Optimization Algorithm

As mentioned in section 2.6, we selected the best 3 model for further optimize experiments.

4.1 Learning Rate Scheduler

LR scheduler is the first optimization algorithm we tried. To enhance training efficiency and increase the likelihood of converging to achieve better accuracy, we adjusted the learning rate (LR) scheduler. Now, the LR is halved every 2 epochs, provided there is no improvement in accuracy.

This adjustment is aimed at optimizing the training process for more effective results. As result (Figure.10), CustomCNNv7.2 is improved a lot by comparing the other two. We use LR scheduler for further experiments.

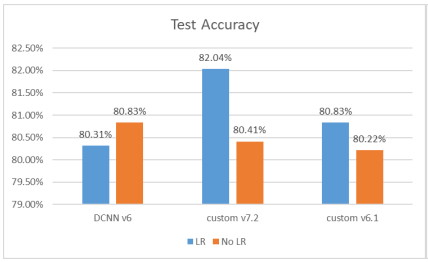


Figure 10: LR scheduler

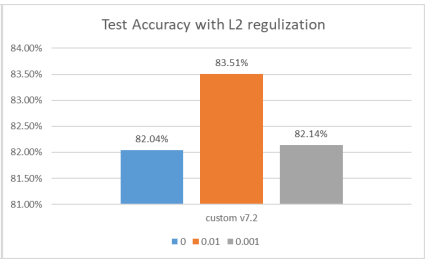


Figure 11: L2 = [0, 0.01, 0.001]

4.1.1 L2 regularize (Figure.11)

Apply L2 regularization on ADAM optimizer. As result, L2 = 0.01 is the best option for the model.

4.2 Adjust weight base on test accuracy of each class(Figure.12)

Adjust the weight of classes during computing loss function. Weight is adjusted based on the test accuracy of classes. It seems no improvement in this set up.

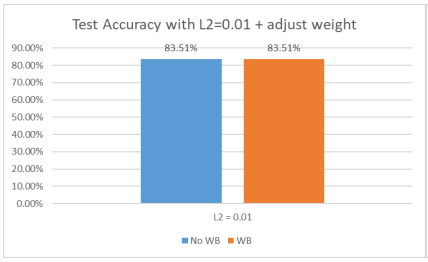


Figure 12: weight balance

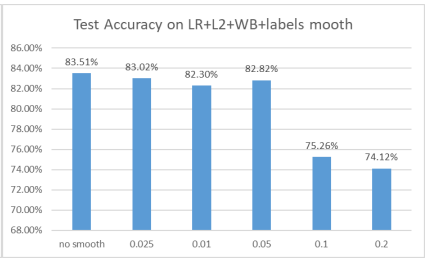
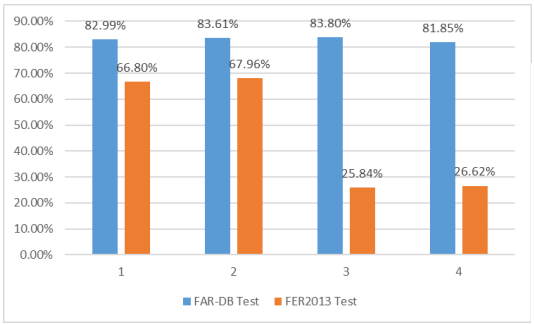


Figure 13: Label Smoothing

4.3 Label Smoothing (Figure.13)

According to the result, label smoothing with any value is not helping in our experiment.

4.4 Data augmentation and Combined Training Set



1. RAF + (AutoAug on RAF label 1,2) x5
   + FER + (AutoAug FER label 1,2) x 5
2. RAF + (AutoAug on RAF label 1,2) x5
   + FER + (AutoAug FER label 1,2) x 5
   + weight balance base on accuracy
3. RAF + (AutoAug on RAF label 1,2) x 10
4. RAF + (salt_pepper noise on RAF label 0,3,4,5,6)
   + (Aug on RAF label 1,2)

Figure 14: Test accuracy on 4 dataset combinations

We also try few different combinations of datasets with data augmentations (using auto-augmentation from transform library). There are so many combinations, we couldn't try all of them. It is very time

consuming just by training on one combination, just by combined RAF-DB and FER213 without augmentation, it already takes nearly an hour on my machine to finish if early stop is applied. Above (Figure.14) is the test accuracy result of trying different combinations of it. The best of all is the second, mix with with both RAF-DB and FER2013 dataset, plus 2 times of auto augmentation on sample of label 1 and 2 in both dataset, then update the parameter of the model by classes weights base on the accuracy on the result of combination 1. Refer back to section 4.2, we infer that adjust weight for updating parameter may improve more in a larger training set.

5 Metrics and Experimental Results

At first, we choose validation accuracy as the metric to compare results. For final optimization result, we choose test accuracy instead of validation accuracy as the metric, because we believe the test set may have more unseen features than validation set. Since the validation set is separated from the original training set, some of the the samples may contain similar features as the separated training set. Most of the comparisons and experimental result are listed in other sections above. Please refer to it for more details.

6 Contribution

Project Github repository: `https://github.com/HaohuaFeng/CSE676_project`

Table 5: Project Contribution

name	contribution
Haohua Feng (60%)	Design the algorithm, Trying different combination of approach to improve the model accuracy, Tweek hyper parameter, Process data, Doing experiment on GPU, Report, PPT
PinKuan Hsieh (40%)	Design the algorithm, Process data, Collect dataset, Analyze results, Report, PPT

References

[1] RAF-DB: https://www.kaggle.com/datasets/shuvoalok/raf-db-dataset

[2] MMAFEDB: https://www.kaggle.com/datasets/mahmoudima/mma-facial-expression?rvi=1

[3] FER2013: https://www.kaggle.com/datasets/msambare/fer2013

[4] : Ozioma Collins Oguine and Kanyifeechukwu Jane Oguine and Hashim Ibrahim Bisallah and Daniel Ofuani (2022) Hybrid Facial Expression Recognition (FER2013) Model for Real-Time Emotion Classification and Prediction

[5] : Jia Le Ngwe and Kian Ming Lim and Chin Poo Lee and Thian Song Ong (2023) PAtt-Lite: Lightweight Patch and Attention MobileNet for Challenging Facial Expression Recognition

[6] Sanghyun Woo and Jongchan Park and Joon-Young Lee and In So Kweon (2018) CBAM: Convolutional Block Attention Module

[7] Attention block implementation: `https://zhuanlan.zhihu.com/p/563549058`

[8] Algorithm structure visualizations is create using `https://www.drawio.com/`.