Proceedings of the Seventh International Conference on Communication and Electronics Systems (ICCES 2022)
IEEE Xplore Part Number: CFP22AWO-ART; ISBN: 978-1-6654-9634-6

Facial Emotion Recognition using DCNN Algorithm

Kurra Santhi Sri
AssociateProfessor, Information Technology
Vignan's Foundation for Science Technology and Research Vadlamudi, India
srisanthi@gmail.com

Namburu Naveen Kumar,Velivela D Satish
Information Technology
Vignan's Foundation for Science Technology and Research Vadlamudi, India
sainaveenkumar.169@gmail.com
devendrasatishvelivela@gmail.com

2022 7th International Conference on Communication and Electronics Systems (ICCES) | 978-1-6654-9634-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCES54183.2022.9835967

I.ABSTRACT

Facial emotion recognition (FER) is critical for human-computer interaction in areas like clinical practice and behavioral description. With the heterogeneity of human faces and kinds of images like various facial poses such as happy, angry, sad, fear, disgust, surprised etc. and lighting, accurate and robust FER by computer models remains a challenge. Deep learning models, particularly Deep Convolutional Neural Networks (DCNNs), have shown great promise among all FER techniques due to their powerful automatic feature extraction and computational of efficiency. On the FER2013 dataset, the highest single-network classification accuracy has been attained in this paper. The VGGNet architecture is used, its hyper parameters are fine-tuned, and different optimization techniques are performed. This proposed model has achieved the state-of-the-art single-network accuracy of 90% on FER2013 without using any additional training data.

I. INTRODUCTION

Identifying expressions that convey basic emotions like fear, happiness, and disgust, among others, is thought as facial emotion recognition. It's useful in human-computer interactions and may be utilized in digital advertising, online gaming, customer feedback evaluation, and healthcare. High emotion identifier accuracy has been achieved in pictures captured under controlled conditions and consistent environments thanks to advances in computer vision, making this a solved problem. Due to high intra-class variation and low inter-class variation, such as changes in facial pose and subtle differences between expressions, challenges persist in emotion identifier under naturalistic terms. Computer vision research is constantly aiming to improve classification accuracy on such problems. Due to their computational efficiency and feature extraction Capability, Deep Convolutional Neural Networks (DCNNs) have shown great promise in image classification. For FER, they are the most widely used deep models. CNN model is used in facial emotion recognition because of its high accuracy. CNNs are very effective in reducing the number of parameters without losing quality.

The goal of this study is to use DCNNs to improve prediction accuracy on FER2013. We use the VGG network to create a series of experiments to check various optimization algorithms and learning rate schedulers. We fine-tuned the model and training hyperparameters to attain state-of-the-art results with a 73.28 percent testing accuracy. This is, to the most effective of our knowledge, the very best single-network accuracy achieved on FER2013 without the utilization of any additional training data. Then, to better understand the network's performance and decision-making process, we create several saliency maps**.**

III.RELATED WORK

DCNNs have shown great potential in image processing since their introduction in the late 1990s. A convolutional layer, a pooling layer, and a fully connected layer are all components of a typical DCNN. As a result, it's good at manipulating static images. However, due to a lack of training data and computing power at the time, the application of DCNNs was restricted. DCNNs became a much more viable tool in feature extraction and image classification after the 2010s, as computing power and the collection of larger datasets increased. Various techniques have been proposed to improve performance even more. To avoid gradient dispersion problems and speed up training, the sigmoid activation function has been replaced by Rectified Linear Unit (ReLU) activation. To down sample the inputs and aid in generalization, various pooling methods such as average pooling and max pooling are used. Dropout, regularization, and data

augmentation are used to avoid overfitting. To prevent gradient vanishing and exploding, batch normalization was developed. There has also been a lot of work put into developing different optimization algorithms that are used in training. Despite the lack of a systematic theoretical guideline for selecting an optimizer, empirical results show that a suitable optimization algorithm can significantly improve the performance of a model. Stochastic gradient descent is the most widely used optimizer (SGD). It's a straightforward method for updating a model's parameters based on the gradient of a single data point. To speed up training, a variety of variations of this algorithm have been proposed. SGD used to find the model parameters that correspond to the best fit between predicted and actual output. AdaGrad scales the learning rate for each network dimension adaptively. The learning rate is drastically reduced by RMSProp. By scaling the learning rate and introducing gradient momentum, Adam combines the benefits of AdaGrad and RMSProp. etc.

One important factor that could affect performance, among many others, is the learning rate. Oscillations around the minima or loss divergence could result from a high learning rate. A low learning rate would significantly slow the model's convergence and could trap it in a non-optimal local minimum. A common technique is to use a learning rate scheduler, which alters the learning rate. Learning rate during training. For instance, time- based decay reduces the learning rate either linearly or exponentially as the iteration number increases. Step decay drops the learning rate by a factor after certain epochs. An adaptive learning rate schedule tries to automatically adjust the learning at based on the local gradients during training. Cosine annealing resets the learning rate periodically and reuses “good weights” during the training process, etc... Cosine annealing also known as stochastic space gradient space decent with restarts helps in accelerating the training of deep neural networks.

IV.EXPERIMENTS

A. Preprocessing, Augmentation and Dataset

When it comes to FER2013 training, we follow the ICML's official training, validation, and test sets. FER2013 includes 35888 images representing seven distinct emotions: anger, neutral, disgust, fear, happiness, sadness, and surprise. According to the competition organizers, Kaggle forum discussion, human accuracy on this dataset is between 65 and 68 percent

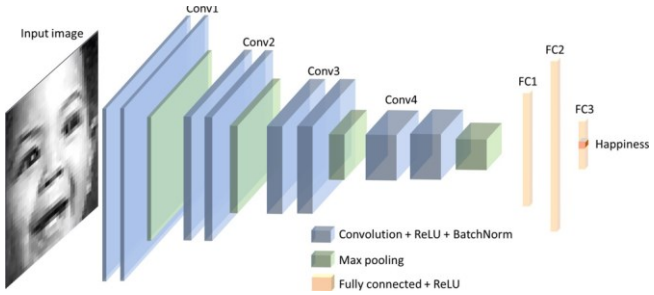

Figure 1: VGGNet architecture.

A face expression image is fed into the model. The four convolutional blocks (Conv) extract high-level features of the image and the fully-connected (FC) layers classify the emotion of the image.

A. TRAINING AND INTERFACE

All of the experiments are run for 300 epochs with the cross-entropy loss optimized. In the sections that follow, we'll change things up a bit. Other parameters should be kept constant, including the optimizer and learning rate schedulers. We use a weight decay of 0.0001 and a fixed momentum of 0.9. To avoid gradient underflow, all experiments are run with gradient scaling. Validation accuracy is used to assess the models, and standard ten-crop averaging is used to test them.

Tuning

We tune our model architecture to maximize performance at first. SGD was used in all of the initial experiments. After our fully connected layers, a grid search is used to determine the best batch size and drop-out rate. Following the optimization of the architecture, we investigate the effects of various optimizers and learning rate schedulers on the performance of our model. We then conducted a final experiment to fine-tune the weights of the trained model and improve its performance.

Optimizer

The goal of the first experiment is to seek out the foremost effective optimizer for training our architecture. SGD, SGD with Nesterov Momentum, Average SGD, Adam, Adam with AMSGrad, Adadelta, and Adagrad are the six algorithms we glance at. Despite the actual fact that several of those algorithms are very similar, understanding how they perform differently during this optimization will help us appreciate the importance of their minor

differences.

This experiment is run in two other ways. We run all algorithms with a set learning rate of 0.001 within the first variation. A grid search was accustomed to determine this learning rate. Within the second variation, we founded a straightforward learning rate scheduler with an initial learning rate of 0.01 and an element of 0.75 reduction if the validation accuracy plateaus after 5 epochs. A grid search was also accustomed determine the parameters of this scheduler. After the initial optimization, all other parameters, like weight decay, momentum, dropout, and batch size, are kept constant.

LR SCHEDULE

The next test will be to determine the best learning rate scheduler. We run the same architecture with 5 different schedulers in this section, using the optimal optimizer determined in the previous section: Reduce Learning Rate on Plateau (RLRP), Cosine Annealing (Cosine), Cosine Annealing with Warm Restarts (CosineWR), One Cycle Learning Rate (OneCycleLR), and Step Learning Rate (StepLR). We also ran a model with a constant learning rate that was determined using a grid search as a baseline. The initial learning rate for all schedulers is 0.01 and their parameters are chosen using a grid search. The rest of the parameters are kept constant.

Despite their similarities, these schedulers perform the learning rate update differently. StepLR and RLRP, for example, both assume that the longer we train, the smaller our step sizes should become. RLRP, on the other hand, monitors the model's current performance before making a learning rate update, whereas StepLR reduces the learning rate after a set number of epochs. A cosine-based learning rate update function is also present in both Cosine Annealing and Cosine Annealing with Warm Restarts. As a result, the learning rate fluctuates between two values. The main difference is that the latter one regularly resets the model's parameters in order to maintain good model weights before proceeding with update steps. These minor variations could have a significant impact on the final performance. Advantages of having more data.

B.VGGNetArchitecture

VGGNet could be a convolutional neural network spec that has been utilized in large-scale image processing and pattern recognition for an extended time [35]. Figure 1 depicts our VGGNet variation. There are four convolutional stages and three fully connected layers within the network. Two convolutional blocks and a max-pooling layer are present in each of the convolutional stages. A convolutional layer, a ReLU activation, and a batch normalization layer conjure the convolution block. Batch normalization is employed to accelerate learning, reduce internal covariance shift, and avoid gradient vanishing or explosion [18]. A ReLU activation follows the activation of the primary two fully connected layers. The classification layer is that the third fully connected layer. Feature extraction, dimension reduction, and non-linearity are all handled by the convolutional stages. The fully connected layers are taught to classify inputs consistent with extracted features.

Tuning

We tune our model architecture to maximize performance at first. SGD was used in all of the initial experiments. After our fully connected layers, a grid search is used to determine the best batch size and drop-out rate. Following the optimization of the architecture, we investigate the effects of various optimizers and learning rate schedulers on the performance of our model. We then conducted a final experiment to fine-tune the weights of the trained model and improve its performance.

Optimizer

The goal of our first experiment is to seek out the foremost effective optimizer for training our architecture. SGD, SGD with Nesterov Momentum, Average SGD, Adam, Adam with AMSGrad, Adadelta, and Adagrad are the six algorithms we glance at. Despite the actual fact that several of those algorithms are very similar, understanding how they perform differently during this optimization will help us appreciate the importance of their minor differences.

This experiment is run in two other ways. We run all algorithms with a set learning rate of 0.001 within the first variation. A grid search was accustomed to determine this learning rate. Within the second variation, we founded a straightforward learning rate scheduler with an initial learning rate of 0.01 and an element of 0.75 reduction if the validation accuracy plateaus after 5 epochs. A grid search was also accustomed determine the parameters of this scheduler. After the initial optimization, all other parameters, like weight decay, momentum, dropout, and batch size, are kept constant.

LRSchedule

The next test will be to determine the best learning rate scheduler. We run the same architecture with 5 different schedulers in this section, using the optimal optimizer determined in the previous section: Reduce Learning Rate on Plateau (RLRP), Cosine Annealing (Cosine), Cosine Annealing with Warm Restarts (CosineWR), One Cycle Learning Rate (OneCycleLR), and Step Learning Rate (StepLR). We also ran a model with a constant learning rate that was determined using a grid search as a baseline. The initial learning rate for all schedulers is 0.01 and their parameters are chosen using a grid search. The rest of the parameters are kept constant. Despite their similarities, these schedulers perform the learning rate update differently. StepLR and RLRP, for example, both assume that the longer we train, the smaller our step sizes should become. RLRP, on the other hand, monitors the model's current performance before making a learning rate update, whereas StepLR reduces the learning rate after a set number of epochs. A cosine-based learning rate update function is also present in both Cosine Annealing and Cosine Annealing with Warm Restarts. As a result, the learning rate fluctuates between two values. The main difference is that the latter one regularly resets the model's parameters in order to maintain good model weights before proceeding with update steps. These minor variations could have a significant impact on the final performance.

Fine Tuning

We then experimented with hyper-tuning our model's final weights to improve its accuracy even more. We reload the parameters and train for a final 50 epochs with a 0.0001 learning rate. Because this is an already trained model, we set this learning rate to keep the update steps small. This ensures that our model's weights are not skewed far away. Cosine Annealing and Cosine Annealing with Warm Restarts are used in this experiment because both of these schedulers slowly oscillate the learning rate back and forth, preventing major weight changes. The second schedule is also advantageous because its warm restarts would mean that the model's weights would be reset to a good location on a regular basis during updates.

The validation set was then combined with the training set in a second variation of this experiment to allow for a larger dataset set when tuning. This larger dataset would provide the model with more samples from which to learn, resulting in improved performance. All other parameters are kept constant, except for the test set. We can confirm two things by running two variations of this experiment. We can confirm the effectiveness of the tuning by using the first variation. Using the second experiment, we can see the advantages of having more data.

V.RESULTS

Optimizer

The impact of optimizers on the model's performance is first investigated and compared. Figure 2 depicts the validation accuracy achieved by our model when using various optimizers. The yellow bars depict the experiment's first variation, which used a constant learning rate, and the orange bars depict the second variation, which used a decaying learning rate. Except for Adadelta, all optimizers have a high validation accuracy of more than 70%. In both experiments, the model using the SGD with Nesterov momentum performs the best, with validation accuracy of 73.2 percent and 73.5 percent, respectively. In addition, we discovered that Adam and its AMSGrad variant outperform Adadelta and Adagrad. Because Adam optimization introduces gradient momentum, it combines the advantages of AdaGrad and RMSProp. Last but not least, All SGD variants outperform all other optimizers on this dataset.

LRSchedule

Following that, we investigate the impact of various learning rate schedulers on our model. Our validation and testing accuracies are shown models. All of the runs in this section use SGD with Nesterov momentum, the best-performing optimizer from the previous section. The first point to mention is that Reducing Learning Rate on Plateau (RLRP) is the most effective method. It has a testing accuracy of 73.06 percent and a validation accuracy of 73.59 percent. To our knowledge, this already outperforms the previous state-of-the-art single-network performance.

Because the testing accuracies we're reporting are solely for public benchmarking, we'll focus on validation accuracy for the next set of comparisons. Some of the other schedulers are outperformed by the constant learning rate (OneCycleRL and StepLR). This could be because OneCycleLR is typically designed for fast training with higher learning rates, which may not be applicable on FER2013.

The results of Cosine Annealing and Cosine Annealing with Warm Restart are comparable. When StepLR and RLRP are compared, they both gradually reduce the learning rate to a minimum. RLRP performs better because it monitors current performance before deciding when to reduce the

learning rate, rather than reducing the learning rate in a systematic manner

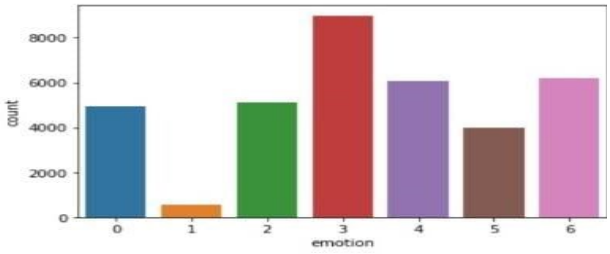

Fig 2: how many images uploaded based on emotions

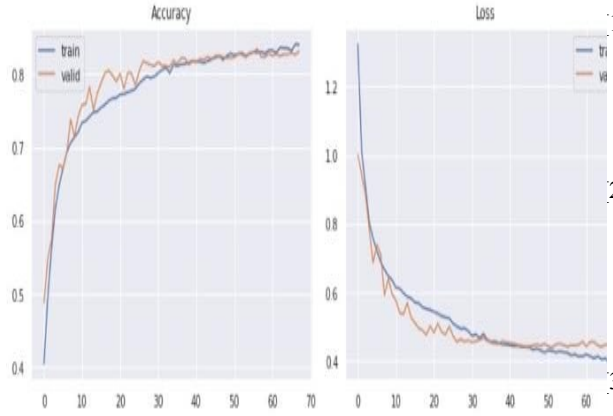

FIG 3: Accuracy and loss of trained data

Confusion Matrix

On the FER2013 testing set, Figure 4 shows the final model's confusion matrix. The model correctly classifies the emotions "happiness" and "surprise." It, on the other hand, makes the most errors when determining the difference between "disgust" and "anger." Next, the low classification accuracy in "disgust" and "fear" can be attributed to the fact that the original training set contains fewer samples. The misclassification of "fear" and "sadness" could be due to the dataset's inter-class similarities

FIG 4: confusion matrix

VI. CONCLUSION

Using a VGGNet, this paper achieves single-network state-of-the-art classification accuracy on FER2013. All hyper parameters are fine-tuned to achieve an optimized model for facial emotion recognition. Different optimizers and learning rate schedulers are investigated, and the best initial testing classification accuracy achieved is 73.06 percent, which exceeds all previous single-network accuracies. We also use Cosine Annealing to fine-tune our model and combine the training and validation datasets to increase the classification accuracy to 73.28 percent. To improve our performance in facial emotion recognition, we plan to use FER2013 to investigate different image processing techniques and ensembles of different deep learning architectures

VII. REFERENCES

[1] M. S. Bartlett, G. Littlewort, I. Fasel, and J. R. Movellan, "Real Time Face Detection and Facial Expression Recognition: DevelopmentandApplicationstoHumanComputerInteraction," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2003, vol. 5, doi: 10.1109/CVPRW.2003.10057.

[2] F. Abdat, C. Maaoui, and A. Pruski, "Human-computer interaction using emotion recognition from facial expression," in *Proceedings - UKSim 5th European Modelling Symposium on Computer Modelling and Simulation, EMS 2011*, 2011, doi: 10.1109/EMS.2011.20.B. Fasel and J. Luettin, "Automatic facial expression analysis:Asurvey,"*PatternRecognition*,vol.36,no.1.2003,doi: 10.1016/S0031-3203(02)00052-3.

[3] E. Sariyanidi, H. Gunes, and A. Cavallaro, "Automatic analysis of facial affect: A survey of registration, representation, and recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 6. 2015, doi: 10.1109/TPAMI.2014.2366127.

[4] N. Mehendale, "Facial emotion recognition using convolutiona lneural networks (FERC),SNAppl.Sci" vol.2,no. 3, 2020, doi:10.1007/s42452-020-2234-1.

[5] V. Tümen, Ö. F. Söylemez, and B. Ergen, "Facial emotion recognitiononadatasetusingConvolutionalNeuralNetwork,"in *IDAP 2017 - International Artificial Intelligence and Data ProcessingSymposium*,2017,doi:10.1109/IDAP.2017.8090281.

[6] D.K.Jain, P.Shamsolmoali,andP.Sehdev,"Extended deep neural network for facial emotion recognition,"*PatternRecognit. Lett.*, vol. 120, 2019, doi:10.1016/j.patrec.2019.01.008.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, 2017, doi:10.1145/3065386.

[8] O. Gervasi, V. Franzoni, M. Riganelli, and S. Tasso, "Automating facial emotion recognition,"*WebIntell.*,vol.17,no. 1, 2019, doi:10.3233/WEB-190397.

[9] M. M. Taghi Zadeh, M. Imani, and B. Majidi, "Fast Facial emotion recognition Using Convolutional Neural Networks and Gabor Filters," in *2019 IEEE 5th Conference on Knowledge*

[10] *Based Engineering and Innovation, KBEI 2019*, 2019, doi: 10.1109/KBEI.2019.8734943.

[11] E. Pranav, S. Kamal, C. Satheesh Chandran, and M. H. Supriya,"FacialEmotionRecognitionUsingDeepConvolutional Neural Network," in *2020 6th International Conference on Advanced Computing and Communication Systems, ICACCS 2020*, 2020,

doi:10.1109/ICACCS48705.2020.9074302.

[12] I. J. Goodfellow *et al.*, "Challenges in representation learning: A report on three machine learning contests," *Neural Networks*, vol. 64, 2015, doi:10.1016/j.neunet.2014.09.005.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient- based learning applied to document recognition,"Proc.IEEE,vol. 86, no. 11, 1998, doi:10.1109/5.726791.

[14] G.E.Dahl,T.N.Sainath,andG.E.Hinton,"Improving deep neural networks for LVCSR using rectified linear units and dropout," in ICASSP, IEEE International Conference on Acoustics,SpeechandSignalProcessing-Proceedings,2013,doi: 10.1109/ICASSP.2013.6639346.