Facial Expression Recognition using Convolutional Neural Networks and Data Augmentation

Agrawal, Nehal
Dept. of Computer Science
University of California, Davis
naagrawal@ucdavis.edu

Iyer, Shreenath
Dept. of Computer Science
University of California, Davis
shriyer@ucdavis.edu

Lakhani, Kshitij
Dept. of Computer Science
University of California, Davis
klakhani@ucdavis.edu

Yi, Henry
Dept. of Computer Science
University of California, Davis
henyi@ucdavis.edu

***Abstract*—Our aim for this project was to develop and train a Deep Convolutional Neural Network to perform Facial Expression Recognition on the fer2013 dataset [8]. We derive human expressions from a gray-scale image of 48 by 48 pixels in a range of 7 of the most common facial expressions. We build the CNN in Keras [3] using four convolution layers. Each convolution layer is followed by a batch normalization layer, ReLU activation layer, Max pooling layer and Dropout layer. This is followed by two Fully Connected layers of 256 and 512 neurons respectively. To improve accuracy, we performed a survey of successful CNN architectures to get an intuition about the hyper-parameters and performed hyper-parameter tuning. We performed data augmentation on fer2013 to increase the size of the dataset. This increased the size by 5 folds, bring it close to 180K samples. Our model achieved an accuracy of 78.5% when trained using this augmented data set. We also tested our model with images of our faces, and achieved satisfactory results.**

***Index Terms*—Facial Expression Recognition, Convolutional Neural Networks, Data Augmentation, Convolution, ReLU, Max Pooling, Dropout**

I. Introduction

Facial expression recognition (FER) has been a topic of interest for many years due to its practical importance in numerous fields. To achieve effective human - computer interaction which is comparable to human - human interaction, computers must be able to interact with users naturally. This can be achieved by taking actions based on the feedback received from the user. Facial expressions are a form of nonverbal communication that are an integral part of our everyday communication. Hence facial expression recognition has gained a lot of interest from the computer vision community. Some of the applications include driver fatigue detection, medical treatment, security, automatic counseling systems, lie detection, operator fatigue detection, sociable robotics etc.

Facial expression recognition started as a lab controlled experiment. Traditional methods used hand crafted features to perform recognition. This was primarily due to the limited amount of data and resources available. However, since 2013, competitions such as the FER2013 [9] and EmotiW [5] [6] [7] have provided large enough datasets to train models to perform recognition in wild settings. Additionally, the benefit from using GPUs has motivated the use of deep learning methods to perform predictions which achieve a very high accuracy.

However, there are still some issues with FER. Firstly, to train deep learning models, large amounts of data is required. Existing datasets are not sufficient to train well known deep models that perform really well on other tasks such as object recognition. Table 1 provides a list available datasets for FER. Secondly, expressions are personal attributes and differ from person to person. This high level of variability between subjects poses a challenge.

DATASET	NUMBER OF SAMPLES
JAFFE [23]	213 images
CK+ [4]	593 image sequences
RaFD [18]	1,608 images
SFEW 2.0 [5]	1,766 images
AFEW 7.0 [6]	1,809 videos
BU-3DFE	2,500 images
Oulu-CASIA [33]	2,880 image sequence
MMI [32] [25]	740 images and 2,900 videos
KDEF [2]	4,900 images
RAF-DB [21] [20]	29,672 images
FER-2013 [9]	35,887 images
ExpW [22]	91,793 images
TFD [30]	112,234 images
AffectNet [24]	450,000 images
Multi-PIE [10]	755,370 images
EmotioNet [1]	1,000,000 images

TABLE I
DIFFERENT FACIAL EXPRESSION DATASETS

This project was motivated from the fer2013 [9] Kaggle competition where the task was to categorize each face based on the facial expression, into one of the seven categories (Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral). The winner of this competition achieved an accuracy of 71.161% [31]. The highest accuracy achieved on this dataset is 75.2% [27].

We built and trained a deep CNN on the fer2013 dataset. Our model is made up of four convolution layers, each followed by output batch normalization layer, ReLU activation layer, Max Pooling layer and dropout layer. Our model also has 2 Fully Connected layers of 256 and 512 neurons respectively. In order to improve accuracy and avoid overfitting, we eliminated bad training examples from the dataset and then performed data augmentation. This increased the size of our dataset by 5 folds, making it sufficient to train a deep network and eliminating the issue of insufficient datasets. We were able

to achieve an accuracy of 78.5%, which is the highest accuracy achieved on this dataset so far.

Section II covers our approach in detail. In section III we present our results. We discuss these results in section IV and conclude our findings. That is followed by acknowledgement and references.

II. Methods

This project involves, working with, and manipulating four crucial components to obtain an accurate, robust and user-appealing model, which we explore further -

A. Dataset and Data Augmentation

1) fer2013 dataset: The data consists of 48x48 pixel grayscale images of faces which have been automatically registered, so that the face is more or less centered and occupies about the same amount of space in each image. The data file (in csv format) contains two columns, 'emotion' and 'pixels'. The 'emotion' column contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The 'pixels' column contains a string for each image. The contents of this string are space-separated pixel values in row major order. The data file is parsed such that we obtain two vectors, one which holds the pixel values for all the images and the other, which holds the class values. Below is a split-up of the training, validation and test data (approximately 80:10:10) -

DATA TYPE	NUMBER OF SAMPLES
Training Data	28,709
Validation Data	3,259
Test Data	3,589

TABLE II
SPLITTING DATASET

2) Data Augmentation: In most cases, working with a larger dataset yields a more accurate prediction and a more robust model. Data Augmentation [26] is a popular technique employed to pseudo-increase the size of the original dataset to obtain a new dataset which is much larger in size. We achieve the same by augmenting fer2013 to obtain a final data set comprising 178,852 samples. Below is a split-up of the augmented dataset (approximately 80:10:10) -

DATA TYPE	NUMBER OF SAMPLES
Training Data	143,081
Validation Data	17,884
Test Data	17,887

TABLE III
SPLITTING AUGMENTED DATASET

This increase in size is obtained by carrying out 4 basic operations on each image in fer2013 - i) flipping each image, ii) rotating each image to the right by 15 degrees, iii) rotating each image to the left by 15 degrees and iv) zooming in on each image. Hence, the size of the augmented dataset is approximately five times that of fer2013. Initially, we had increased the dataset to nine times that of fer2013 by using four more operations - shift up, down, left and right by a fixed amount, however, due to resource-constraints we couldn't train our model on this huge dataset. So we limited ourselves to using a 5x augmentation.

B. Architecture

In order to develop a suitable, yet easy to train model, we researched some of the most popular architectures like VGG16 [28], AlexNet [17], ResNet [11] and LeNet [19]. Based on their architectures and our input size of 48x48 pixels, we decided to try out two architectures - Shallow and Deep. The Shallow architecture is used - i) as a stepping stone towards the deep architecture and ii) as a test model for new ideas as it requires lesser training time. The shallow model comprises of two hidden layers - the first hidden layer comprises a Convolutional layer (64 filters of size 3x3 and 'same' type of padding), a ReLu activation layer and a Max Pooling layer (filter size 2x2 and stride of 2), whereas the second hidden layer comprises a Convolutional layer (128 filters of size 5x5 and 'same' type of padding), a ReLu activation layer and a Max Pooling layer (filter size 2x2 and stride of 2). The output of the second hidden layer is flattened out and fed to a Fully Connected layer comprising 256 activation units, which is then followed by a Softmax layer.

Dropout [29] and Batch Normalization [13] techniques were used as apart of each Hidden layer and Fully connected layer. The dropout rate is set to 25% , meaning one in four inputs will be randomly excluded from each update cycle. Batch Normalization normalizes the activations of the previous layer at each batch, i.e. it applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. Dropout helps us to avoid over-fitting and train the network faster. Similarly, Batch Normalization also reduces training time while also, serving as a good input to activation functions (to prevent ReLU from dying out during training)

The Deep architecture, on the other hand, has two additional Hidden layers (shown in Fig. 1) - the first hidden layer of the Deep architecture comprises a Convolutional layer (64 filters of size 3x3 and 'same' type of padding), a ReLu activation layer and a Max Pooling layer (filter size 2x2 and stride of 2) and, the second hidden layer comprises a Convolutional layer (128 filters of size 5x5 and 'same' type of padding), a ReLu activation layer and a Max Pooling layer (filter size 2x2 and stride of 2) whereas, the third and fourth hidden layer comprise a Convolutional layer (512 filters of size 3x3 and 'same' type of padding), a ReLu activation layer and a Max Pooling layer (filter size 2x2 and stride of 2). The output of the fourth layer is flattened out and fed to a Fully Connected layer comprising 256 activation units, the activations of which are fed to a second Fully Connected layer comprising 512 activation units, followed by a Softmax layer . Batch Normalization and Dropout techniques are employed in each Hidden layer and Fully Connected layer, with characteristics similar to those in the shallow model, as discussed above.

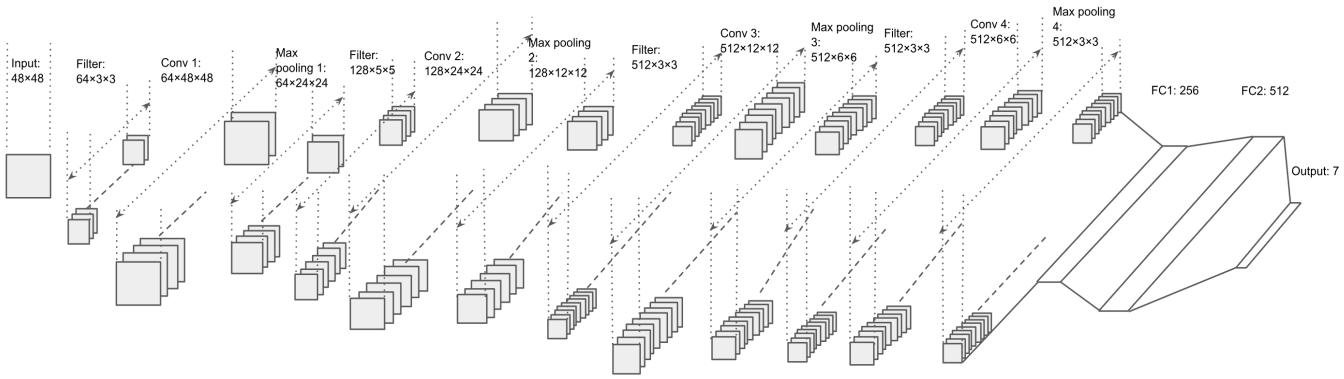

Fig. 1. Architecture for the Facial Expression Recognition using CNNs

C. Training the Model

Taking inspiration from the sources in our literature survey of architectures, and a few trial runs, we decided on choosing the number of Epochs to be 124 and the Batch Size to be 128. Additionally, we decide to use the Adam [16] Optimizer and the Binary Cross entropy loss function to teach our model how to predict facial expressions and categorize them correctly. Our choice of the loss function and the optimizer was driven from the good performance numbers that these methods have yielded in similar previous implementations [33]. In the next section, we delve into how these hyper-parameters (and also the ones mentioned in Section B) are tuned to obtain the best possible accuracy.

D. Hyper-parameter Tuning

For the deep architecture described in the earlier sub-section, we started out with had an accuracy of 63.58% on training data and 64.11% on test data for fer2013.

The objective was to increase this accuracy using hyper-parameter tuning and then use it along with data augmentation to possibly achieve the best possible accuracy. For this, we decided to tune the number of epochs, batch size, convolution filter size, optimizer, output layer activation function and loss function. Unless mentioned otherwise, the default values for these are as follows - Number of epochs = 124, Batch Size = 128, Loss function = Binary Cross Entropy, Output Layer Activation Function = Sigmoid ; the other parameters are set to the default values mentioned in Methods, Sections B and C. The tuning is carried out for both, Shallow and Deep models, however, for brevity reason, we will only show the results for the Deep Model (see Table IV)

From this we deduced that the best possible configuration is entry number four in the table i.e. Epochs = 1200, Batch size = 256, Loss = Categorical Cross Entropy.

E. Extending the Testing Dataset and Using a Web-cam

The images in the fer2013 dataset are gray-scale, 48x48 pixels and are in csv format. The lower resolution makes it easier to reconstruct the original image, but difficult to view the actual image. Furthermore, the data set is absolute and we also needed to validate if the model could detect expressions of faces in images outside the dataset. In order to test the accuracy of our model, we decided to capture photos of ourselves and test the model on our faces.

To begin with, we captured pictures of a single human

HYPER-PARAMETER CHANGE	EFFECTS ON ACCURACY
Default Parameters	Accuracy on Training set : 63.58% Accuracy on Test set : 64.11%
Epochs = 200	Accuracy on Training set : 63.42% Accuracy on Test set : 64.61%
Epochs = 500, Batch size = 256	Accuracy on Training set : 63.69% Accuracy on Test set : 65.44%
Epochs = 1200, Batch size = 256 Loss = Categorical Cross entropy	Accuracy on Training set : 65.39% Accuracy on Test set : 66.035%
Loss = Categorical Cross entropy Conv1 - 64, (3*3), Conv2 - 128, (3*3) Conv3 - 256, (5*5) Conv4 - 512, (7*7) FC1 - 256 FC2 - 1024	Accuracy on Training set : 61.64% Accuracy on Test set : 63.24%
Epochs = 200, Batch size = 256 Output layer = Softmax Loss = Categorical Cross entropy	Accuracy on Test set : 65.14%
Epochs = 512, Batch Size = 256 Optimizer=rmsprop	Accuracy on Test set : 59.38%

TABLE IV
HYPER-PARAMETER TUNING

Fig. 2. Converting an RGB image of arbitrary size to a 48x48 gray-scale image

Fig. 3. Surprised / Fear

model with seven different expressions. However, in order to be consistent with the fer2013 dataset, we first, manually crop the picture to focus on the face and the expression of the person in frame. We maintain a square aspect ratio while cropping so that we can scale it down to a resolution of 48x48 pixels without altering the proportions of the image. Next, we convert this image, which is in RGB format, to its gray-scale format, which is uniform with fer2013's way of representation. Because fer2013 dataset provides a pixel delineation of its training and test data set, we wrote a script that automates the conversion of our gray-scale images to their respective pixel values and stores them as a one dimensional array in a csv format as shown in fig. 2. This creates a dataset that is exactly similar to fer2013's way of representation.

Moreover, to make the implementation an interactive one, we also decided to interface a web-cam to the trained model. This requires detecting a face with the web-cam, capturing a snapshot and pre-processing it to a grayscale 48*48 image to feed it to our model.

F. Merging Classes

One of the reasons why facial expression recognition is a difficult task is due to the similarity in certain expressions, for instance, sometimes, models may classify Surprise and Fear interchangeably (see Fig. 3 - one may easily classify it as Fear instead of Surprise), or Anger and Disgust interchangeably. On observing the dataset, we noticed that images labelled as Disgust made up only 1.5% of the fer2013 dataset. Such a small number of samples may make it difficult to train the model for classifying faces showing disgust, resulting in a lower accuracy.

As Anger and Disgust are similar expressions, we decided to merge both of these and label all images as Anger, thereby, removing the Disgust class altogether (resulting in a total of 6 classes) . On experimenting with this using 124 Epochs and a Batch size of 256 we obtained an accuracy of 50.73% on our test data, which is lesser than what we had with the original 7 classes. Hence, we decided to not go ahead with this idea and reverted back to our original model and dataset.

III. Results

We derive our fer2013 [8] dataset from a Kaggle competition that was released in 2013. The maximum accuracy anyone has achieved on the leader boards is 71.61%, as noted by the competition. We compare our results against an earlier

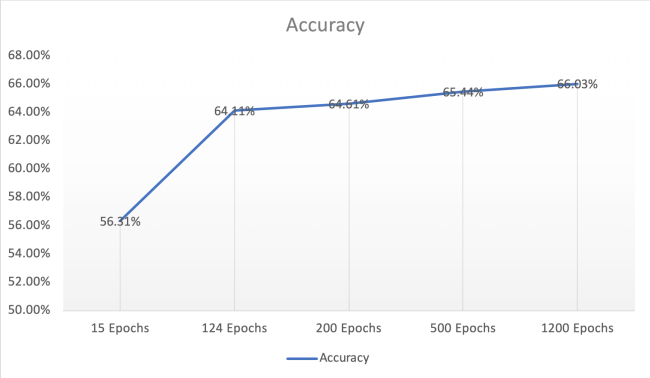

Fig. 4. Accuracy of the model by increasing the number of epochs

implementation of ours which gave an accuracy of 64.11% on the testing dataset. In order to achieve our goal, we ran our experiments on the test data by:

- Modifying the hyper-parameters
- Merging classes
- Augmenting the data set

In the following subsections, we discuss in detail the impacts of each class of the above experiments in terms of performance, robustness and complexity.

A. Accuracy of the model

In this section, we discuss how the three experiments led to different accuracy scores in detail.

1) Modifying the hyper-parameters: For improving the performance of the model, we first modified the hyper parameters of the model to verify its accuracy against the previous performance. We began by increasing the number of epochs in our model from as little as 15 epochs to 64 epochs and eventually 1200 epochs in a gradual manner while keeping the batch size constant at 128 batches. The results of this tuning is noted in fig 4. As seen in the figure, the accuracy of the model is only as good as a random model with a lowly 15 epochs. By increasing the number of epochs to 124 epochs, we were able to assimilate a gain of around 8% in our accuracy scores. However, from here on wards, the gains are fairly limited. An increase in the number of epochs to 200, only gives a minimal boost of 0.5%, and increasing it further to 500 epochs gives an accuracy of 65.44%. Going ahead, increasing the number of epochs to 1200 gives a total accuracy of 66.03%. We stop with our experiments on the size of epochs at this point realizing that the gains we receive isn't cost effective with the training time. Hence, we tried changing the batch size along with the size of epochs in the next part of the same experiment.

By maintaining a constant epoch size of 15 epochs, we ran a smaller model with varying batch sizes and noted the accuracy results that we achieved as seen in fig 4. The model hit an accuracy level of 62.19% with a batch size of just 64. This accuracy is notably greater than the one we received with an epoch size of 15 and a batch size of 124 epochs, which

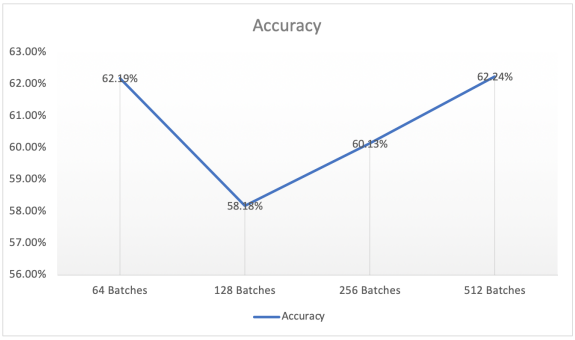

Fig. 5. Accuracy of the model by increasing the number of batches

was nearly 4% under at 48.18%. On further increasing the batch size to 256, we received accuracy levels of 60.13% and by increasing the batch size to 412 batches, the results were surprisingly only slightly greater than that of what we received with a batch size of 64 batches. It was interesting to see the drop in the accuracy of the model with an increase in the batch size in the beginning, and then the gradual climb back up.

Based on our experiments on varying the batch size and epoch size, we were able to conclude that improvements to the overall accuracy of the model is marginal. Hence, we focused our efforts on different experiments.

2) Merging Classes: Our literature study also suggests that the facial expressions of a human that display anger and disgust are very similar. Hence, the neural network can predict either of the two expressions as its output for expressions of disgust or anger. Hence, we decided to combine the two classes and reduce the number of classes by one (from seven to six) to see what results the model yields.

Theoretically, merging these two classes should have improved accuracy. However, the results were contrary to our expectations. We found that by running a model that with 124 epochs and 256 batches with a lesser number of classes yields an accuracy of just 50.7% which was just as good as a random prediction. This was significantly lower than running a model with 7 classes. Hence, we decided to discard this experiment as not important to improving the accuracy.

3) Data Augmentation: In order to train our model, we used 28709 samples and we further used 3589 samples for validation on a deep model with 4 convolutional layers. However, the yields we received was only slightly better than the baseline model. Hence, we decided to augment the original dataset by performing data augmentation on it. We performed 4 kinds of data augmentations viz. i) flipping each image,ii) rotating each image to the right by 15 degrees, iii) rotating each image to the left by 15 degrees and iv) zooming in on each image as described in section II.2. The results of data augmentation in comparison to other methods can be seen in table 5.

It can be seen that the model yields its highest accuracy results when it's augmented and run with an epoch size of 100, a batch size of 128 and *categorical_crossentropy* as the loss function as seen in figure 6. Increasing the batch size to 256 slightly lowers the accuracy of the model.

NO. OF EPOCHS	BATCH SIZE	LOSS FUNCTION	NO. OF AUGMENTATIONS	ACCURACY ON TEST SET
124	128	binary_crossentropy	0	64.13%
200	128	binary_crossentropy	0	64.61%
500	256	categorical_crossentropy	0	65.44%
1200	256	categorical_crossentropy	0	66.03%
100	128	categorical_crossentropy	4	78.51%
100	256	categorical_crossentropy	4	78.22%

TABLE V
COMPARING THE ACCURACY OF DIFFERENT MODELS

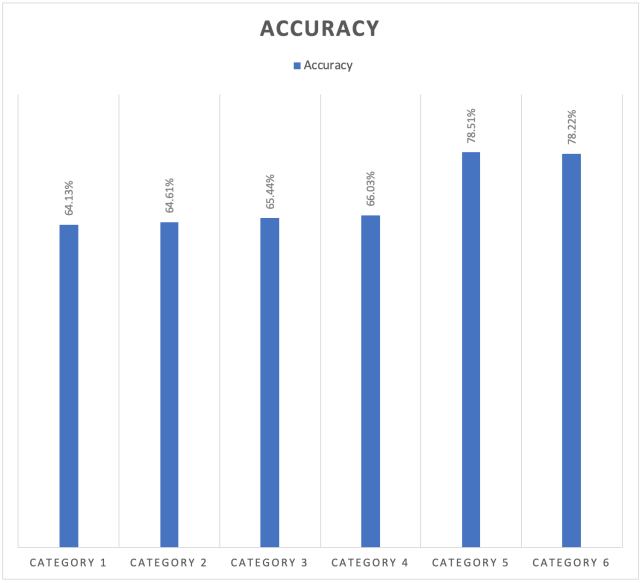

Fig. 6. Accuracy of the model for six categories in table V

B. Implications on and of the Dataset

According to our experiments, data augmentation has had the biggest impact on the accuracy of the model. Because we train our data set on the augmented data set, we essentially get to train our model on a data set that is 5 times larger than the original fer2013 dataset. This facilitates more training for the model and thus, reaches a greater accuracy. Moreover, we have run only 4 kinds of augmentations on the dataset. It is possible to increase the dataset size by carrying out more augmentations. The dataset that we have created by augmentation is equal to the raw dataset size of datasets such as AffectNet. Furthermore, we were able to achieve an improvement of nearly 15% over the baseline model which we started out with. This was achieved by running the model on the augmented data set, which was missing before.

Our results also prove that data augmentation is an effective way of achieving higher accuracy without having to use methods such as transer learning. One of the key drawbacks of the fer2013 dataset is that the input images are 48*48 grayscale images that are provided in a csv format. The small size of the input makes it near impossible to run deeper models such as VGG-16 or GoogLeNet which require much larger input sizes. The smaller sized fer2013 input images would shrink to sizes where feature extraction because highly difficult in such

Fig. 7. Result of the model on our faces

deeper neural networks. Our results show that transfer learning may not even be the most effective way for achieving better results. In many cases, such as this one, the key lies in making the most out of the data at hand.

C. Testing the Model on Additional Faces

To verify how well we understood the dataset and to check the accuracy of our model, we ran our model on our faces to see if we could correctly identify the expressions. Figure 7 shows how well the model works on our faces as well.

Although we weren't able to collect a large number of images, we were still able to correctly predict 5 out of the 7 classes using our enhanced model. This is an improvement over the results we achieved using the baseline model where we could predict only 3 out of the 7 classes. Not only does this cement our understanding of the data set itself, it also shows how we were able to train our model to achieve significantly higher results than before.

IV. Discussion

The big takeaway from this project is the increased accuracy on both, training data (78.47% accuracy) and test data (78.51% accuracy) by embodying Hyper-parameter Tuning and Data Augmentation. The accuracy that we achieve is more than any other implementation that we know of.

The decision to gradually increase the number of epochs was to ensure that the model learns the features better with each step increase in the number of epochs. After a certain number of epochs however, a tendency does exist, for the graph in Fig. 4 to flatten out , or even sometimes reduce, due to over-fitting. Hence, we decided to stick to the 100-150 epoch range, as further computations after that would only result in over-fitting and maybe not give the bang for one's buck. Our implementation achieves the accuracy of 78.51% on test data by using 100 epochs. Ideally, we would have liked to go up to 150 epochs to explore the changes in accuracy, but, we had to restrict ourselves to only a 100 due to resource limitations.

It has been observed that large-batch methods tend to converge to sharp minimizers of the training and testing functions which in turn leads to poorer generalization. In contrast, small-batch methods consistently converge to flat minimizers [15]. Moreover, when we deal with larger number of epochs (in our case, approximately 100) relative to the size of the dataset, the graph in Figure 5 peaks around a batch size of 256 and then further decreases for larger batch sizes, possibly due to the fact that large-batch methods tend to converge to sharp minimizers of the training function. Consequently, training with a smaller batch-size means the generalization of the error is not good enough and it also takes longer time to learn.All in all, we figured out that small batches cause gradient descent to attract to wider basins, while large batches cause gradient descent to attract to narrow basins, which causes higher error when attempting to generalize, because missing the mark in a narrow basin causes higher changes in error. Based on empirical data and the results we obtained, we decide to use 256 as our batch size as it seemed suitable for our dataset size and number of epochs. As literature suggests, to ensure correct learning, we also set the learning rate very low to begin with when using a slightly large batch size [15]

For neural networks, the benefits of a large size dataset is two folds. First, it improves the accuracy of prediction. Second, it prevents a model from over fitting the data. Deep neural networks have very large number of learnable parameters. For these parameters to accurately learn the mapping between the input and output, they need sufficient number of data samples. Therefore, in order to improve the accuracy of our model, we performed data augmentation on the fer2013 dataset. Initially, we performed 8 different types of transformations (flip, rotate left, rotate right, zoom, shift up, shift down, shift left and shift right) per image of the original dataset. This resulted in a dataset that was 9 times the size of the original dataset. However, we did not have resources to train a deep model on a dataset of this size. So we reduced our augmentation to just 4 types (flip, rotate left, rotate right, zoom). This dataset was 5 times bigger. We trained our model on this new dataset while simultaneously tuning parameters. Upon using this augmented dataset, the accuracy of our model jumped from 66% to 78.5%. This clearly signifies the importance of a large dataset.

We also tested our best model with our own images. We got an accuracy of 71.42%.

V. Future Development

To further the development of facial expression recognition, we contrived a new dynamic facial expression recognition method: to use a Long Short Term Memory (LSTM) to sequentially learn from multiple frames of a dynamic facial expression to do facial expression recognition. We believe such method could bring more accurate facial expression recognition since a human's expression is always dynamic with many frames instead of just one; for example, a sad face image may have just been one frame of a rather happy facial expression.

RNN is innately suitable to process sequential learning since it considers the previous state in learning. However, normal RNN has a gradient vanishing problem as the epoch increases. LSTM is a very famous model that solves that problem. LSTM has an input gate to control the flows of input into the current state, an output gate to control the mixture of the current state in the output, and a forgot gate to control how much

the previous state stores in the current state [12]. So, it can store an input for a period of time and thus keeps gradient from falling to zero. Moreover, it has much achievement on natural language processing, which is a similar processing of another type of sequential human expression.

Nevertheless, due to the difficulty in finding enough dynamic facial expression training data and other time limitation, we reserved this proposal to explore it in the future to further improve the accuracy in facial expression recognition and make it more applicable in daily lives.

Acknowledgment

We would like to thank Prof. Illias Tagkopoulos for the inputs he provided us with during his classes that helped us understand many concepts. We would also like to express our gratitude to the authors of the fer2013 [8] dataset, the authors of the CK+ [14] dataset and the authors of the AffectNet [24] for providing us with the facial expression datasets that we could study and run our experiments on.

Author Contributions

We had created a repository on Git where each of us had our own branch, hence we could collaborate in an easier manner and work parallely. Here are the contributions of each author to the project:

1) Nehal Agrawal- Trained model with different configuration of hyper parameters. Trained model after removing bad training data. Performed data augmentation to generate a new data set that was 9 times the size of the original data set. Trained using the new data set to find the best accuracy model.
2) Shreenath Iyer- Created the Git repository for everyone to work on with the initial model. Trained the model with different tuned hyper parameters. Fixed the script that generates the new data set. Automated the conversion of gray-scale values to pixel values and store in a csv format consistent with fer2013. Wrote the script to detect accuracy of a model on our faces.
3) Kshitij Lakhani- Surveyed different architectures. Trained the model with several different hyper parameters. Implemented merged class implementation. Suggested data augmentation as an accuracy increasing parameter. Automated the integration of web-cam to the model. Worked on considering transfer learning as a possibility.
4) Henry Yi- Tried a VGG-19 network on the fer2013 dataset. Made and tried several face image pre-processing techniques and algorithms including illumination normalization, face frontalization, and face extraction. Proposed to use a LSTM to develop dynamic facial expression recognition. Suggested to use a live and interactive application.

   Besides this, each member of the team has contributed in writing the technical report.

   The link to our git repository is here:
   https://github.com/ShreenathIyer/
   facial-expression-recognition

References

[1] Benitez-Quiroz, C. F., Srinivasan, R., and Martínez, A. M. Emotionet: An accurate, real-time algorithm for the automatic annotation of a million facial expressions in the wild. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 5562–5570.

[2] Calvo, M. G., and Lundqvist, D. Facial expressions of emotion (kdef): Identification under different display-duration conditions. *Behavior Research Methods 40*, 1 (Feb 2008), 109–115.

[3] Chollet, F., et al. Keras, 2015.

[4] Courgeon, M., Martin, J.-c., and Jacquemin, C. Users Gestural Exploration of Different Virtual Agents Expressive Profiles (Short Paper). In *AAMAS '08 Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 3* (Estoril, Portugal, 2008), Padgham, Parkes, Müller, and Parsons, Eds., no. Aamas, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1237–1240.

[5] Dhall, A., Goecke, R., Gedeon, T., and Sebe, N. Emotion recognition in the wild. *Journal on Multimodal User Interfaces 10*, 2 (Jun 2016), 95–97.

[6] Dhall, A., Goecke, R., Ghosh, S., Joshi, J., Hoey, J., and Gedeon, T. From individual to group-level emotion recognition: Emotiw 5.0. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction* (New York, NY, USA, 2017), ICMI 2017, ACM, pp. 524–528.

[7] Dhall, A., Goecke, R., Joshi, J., Hoey, J., and Gedeon, T. Emotiw 2016: Video and group-level emotion recognition challenges. In *Proceedings of the 18th ACM International Conference on Multimodal Interaction* (New York, NY, USA, 2016), ICMI 2016, ACM, pp. 427–432.

[8] Goodfellow, I., Erhan, D., Carrier, P.-L., Courville, A., Mirza, M., Hamner, B., Cukierski, W., Tang, Y., Thaler, D., Lee, D.-H., Zhou, Y., Ramaiah, C., Feng, F., Li, R., Wang, X., Athanasakis, D., Shawe-Taylor, J., Milakov, M., Park, J., Ionescu, R., Popescu, M., Grozea, C., Bergstra, J., Xie, J., Romaszko, L., Xu, B., Chuang, Z., and Bengio, Y. Challenges in representation learning: A report on three machine learning contests, 2013.

[9] Goodfellow, I. J., Erhan, D., Carrier, P. L., Courville, A., Mirza, M., Hamner, B., Cukierski, W., Tang, Y., Thaler, D., Lee, D.-H., et al. Challenges in representation learning: A report on three machine learning contests. *arXiv preprint arXiv:1307.0414* (2013).

[10] Gross, R., Matthews, I., Cohn, J., Kanade, T., and Baker, S. Multi-pie. *Image Vision Comput. 28*, 5 (May 2010), 807–813.

[11] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[12] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[13] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[14] Kanade, T., Cohn, J., and Tian, Y.-L. Comprehensive database for facial expression analysis. In *Proceedings of the 4th IEEE International Conference on Automatic Face and Gesture Recognition (FG'00)* (March 2000), pp. 46 – 53.

[15] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. Long short-term memory. *CoRR abs/1609.04836* (2016).

[16] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[17] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[18] Langner, O., Dotsch, R., Bijlstra, G., Wigboldus, D. H. J., Hawk, S. T., and van Knippenberg, A. Presentation and validation of the radboud faces database. *Cognition and Emotion 24*, 8 (2010), 1377–1388.

[19] LeCun, Y., et al. Lenet-5, convolutional neural networks.

[20] Li, S., and Deng, W. Reliable crowdsourcing and deep locality-preserving learning for unconstrained facial expression recognition. *IEEE Transactions on Image Processing PP* (09 2018), 1–1.

[21] LI, S., DENG, W., AND DU, J. Reliable crowdsourcing and deep locality-preserving learning for expression recognition in the wild. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), 2584–2593.

[22] LIU, P., HAN, S., MENG, Z., AND TONG, Y. Facial expression recognition via a boosted deep belief network. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition* (Washington, DC, USA, 2014), CVPR '14, IEEE Computer Society, pp. 1805–1812.

[23] LYONS, M., AKAMATSU, S., KAMACHI, M., AND GYOBA, J. Coding facial expressions with gabor wavelets. In *Proceedings of the 3rd. International Conference on Face & Gesture Recognition* (Washington, DC, USA, 1998), FG '98, IEEE Computer Society, pp. 200–.

[24] MOLLAHOSSEINI, A., HASANI, B., AND MAHOOR, M. H. Affectnet: A database for facial expression, valence, and arousal computing in the wild. *IEEE Transactions on Affective Computing PP*, 99 (2017), 1–1.

[25] PANTIC, M., VALSTAR, M. F., RADEMAKER, R., AND MAAT, L. Web-based database for facial expression analysis. In *Proceedings of IEEE Int'l Conf. Multimedia and Expo (ICME'05)* (Amsterdam, The Netherlands, July 2005), pp. 317–321.

[26] PEREZ, L., AND WANG, J. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621* (2017).

[27] PRAMERDORFER, C., AND KAMPEL, M. Facial expression recognition using convolutional neural networks: State of the art. *CoRR abs/1612.02903* (2016).

[28] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[29] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[30] SUSSKIND, J., ANDERSON, A., AND HINTON, G. E. The toronto face dataset. *U. Toronto, Tech. Rep. UTML TR 1* (2010), 2010.

[31] TANG, Y. Deep learning using support vector machines. *CoRR abs/1306.0239* (2013).

[32] VALSTAR, M. F., AND PANTIC, M. Induced disgust, happiness and surprise: an addition to the mmi facial expression database. In *Proceedings of Int'l Conf. Language Resources and Evaluation, Workshop on EMOTION* (Malta, May 2010), pp. 65–70.

[33] ZHAO, G., HUANG, X., TAINI, M., LI, S. Z., AND PIETIKÄINEN, M. Facial expression recognition from near-infrared videos. *Image Vision Comput. 29*, 9 (Aug. 2011), 607–619.