CS 373 Final Project: Facial Emotion Recognition with Convolution Neural Networks

Dzung Pham

Williams College

dzung.pham@williams.edu

May 17, 2018

Abstract

Facial emotion recognition is a Computer Vision task that can find applications in numerous fields, such as health-care, security, businesses, etc. Given a human(-like) face, the challenge is to predict the expressed emotion out of 7 possible basic emotions. In recent years, numerous Computer Vision problems have been empirically shown to be efficiently tackled with deep learning models. In this project, the author created a Convolution Neural Network model with TensorFlow, combining such techniques as data preprocessing & augmentation, dropout, and batch normalization, and trained it on the FER-2013 dataset from the Kaggle Facial Expression Recognition Challenge. The final test accuracy of the model is 65.33%. *Using the trained model, the author also built a real-time program with OpenCV to predict facial emotions using webcam.*

I. Introduction

Facial emotion recognition is a well-studied problem in the field of Computer Vision. The challenge is to predict the emotion expressed on a human(-like) face from 7 basic emotions, namely: angry, disgust, fear, happy, sad, surprise, and neutral. These emotions are generally considered to be universally recognizable for humans, regardless of culture or ethnicity. However, they can be very difficult for a machine to detect due to very subtle facial movements and micro-expressions.

The ability to automatically and reliably detect facial emotion is remarkably useful to numerous fields, such as psychology studies or treatments involving human emotions, or security practices employed in hospitals or airports, and even business models involving the evaluation of customers' satisfaction, which is critical to success. Traditional machine learning techniques like support vector machines (SVMs) require the explicit design and extraction of features from the data, and are usually not very accurate. Recent developments in deep learning have made it possible to solve Computer Vision tasks much more efficiently, and facial emotion recognition is no exception. The use of Convolution Neural Network (CNN) models in Computer Vision has become increasingly popular thanks to the properties of CNNs, as well as innovations in hardware (GPUs) that made the process of training deep learning models more feasible.

In this project, I investigated the application of CNN models in detecting facial emotion. Basing the design of my model from previous studies and combining recently developed techniques such as batch normalization [6], I managed to train a model that achieves 65.33% test accuracy using the dataset from the 2013 Kaggle Facial Expression Recognition

Challenge (FERC).[1] To put in perspective, the accuracies of the top 5 entries of the competition range from 65.25% to 71.16%. Finally, I created a program using my model to illustrate the application of real-time emotion detection. My work can be found at: https://github.com/vietdzung/facial-emotion-recognition-cnn.

II. Background/Related Work

The use of multi-layer neural networks has been shown to be very useful in some Computer Vision problems as early as 1989 by Lecun [9]. In the last two decades, convolution neural networks have been successfully applied by research institutes and industry companies to numerous Computer Vision tasks, such as object recognition, handwriting classification, image restoration, etc. [10]. Krizhevsky et al [8] demonstrated in 2012 the remarkable success of very large deep convolution neural networks in image classification on the ImageNet dataset, which has 1.2 million images with 1000 classes, combining such innovations as multiple GPUs training and dropout [5].

With regards to Computer Vision tasks concerning facial expressions and features, the work of Gudi [4], on which my model was partly based, shows promising results in applying CNN to facial emotion recognition, achieving an overall accuracy of 67.12% on the FER-2013 test set. The work of Correa et al [1] also provides me with important guidances in designing my model. The current state-of-the-art facial emotion recognition model is by Tang [12], who is also the winner of the 2013 Kaggle FERC. Combining CNNs with the L2-SVM loss function, Tang's model achieves an accuracy of 71.16% on the private test set and 69.4% on the public test set. All top 3 entries of the competition used CNN models, which have better metrics than conventional methods that explicitly extract facial features [2]. A recent survey in 2018 by Ko shows that deep-learning-based models perform better on average than conventional (hand-crafted features) approaches [7].

The book *Deep Learning* by Goodfellow, Bengio, and Courville provides an excellent introduction to deep learning techniques in general and Convolution Neural Networks in particular [3]. The Computer Science Department of Stanford University offers a course titled CS231n,[2] which is entirely dedicated to Visual Recognition with Convolution Neural Networks, and from which I have learned many valuable lessons.

III. Formulation

Facial emotion recognition is a multi-class classification problem. Because of the structure of the input (2D matrix), the problem can be tackled with CNNs. A neural network model is considered a CNN model if it uses at least one convolution layer in its architecture. A typical CNN model is consisted of several convolution stages, followed by some fully connected layers, the last of which has the same number of nodes as the number of classes. This section will briefly describe the basic building blocks of CNNs, as well as other concepts used in my model. More details can be found in [3].

1. Convolution Layer

A convolution layer applies an operation called *convolution* on the input of the layer. For our current problem, the input are 2D images with 1 color channel and can be represented as 2D matrices of real-valued numbers. We define the convolution operation[3] as:

$$\begin{aligned} S(i,j) &= (I * K)(i,j) \\ &= \sum_m \sum_n I(i+m, j+n) K(m,n) \end{aligned}$$

[1] https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

[2] http://cs231n.github.io

[3] Technically, the operation described here is called *cross-correlation*, which is almost the same as convolution.

where I is a 2D matrix representing our input images, i, j are the coordinates on the images, and K is a 2D kernel or filter. We can think of K as a 2D window that are dragged across both axes of our input, and $(I * K)(i, j)$ as a matrix multiplication of K with the corresponding part of the image in position (i, j). The window has a smaller size than the input image, and is dragged some s pixels called *stride*. The size of the output depends on the size of K and s.

After applying K on all portions of I such that K fits within I, the result is a matrix that is smaller than I called *feature map*. For each convolution layer, there are usually multiple different kernels applied in parallel to the input, each detecting some features of the input. The values of the entries of a kernel can be learned.

After applying all convolution operations, the values in the resulting feature maps are passed into an activation function, of which role is somewhat similar to the activation threshold of biological neurons. Modern CNN models typically use *rectified linear units* (ReLU) for convolution layers, which essentially apply a rectifier function on the values of the units of the matrix:

$$f(x) = max(0, x)$$

2. Pooling Layer

A pooling layer typically follows a convolution layer and creates a 2D matrix that replaces each location in the output of the convolution layer with a summary statistics of the neighboring locations [3]. Pooling can help increase the model's invariance to spatial translation of an image's features. We can think of pooling as sliding a window of a certain size across the image by some stride s and applying some operations on the pixels that fall within the window. Like the kernel's stride, the pooling layer's stride also reduces the input size, thereby reducing computation and memory requirement. In many cases, it is useful to add paddings to the output of the pooling layer to make up for the reduced size. The recommended pooling operation is *max pool*, which simply returns the maximum value in a window.

3. Dropout

Dropout is a recently developed regularization technique in neural networks [5]. A dropout layer randomly omits each hidden unit in the representation of an input with some probability p. It has been experimentally shown that dropout can efficiently help prevent models from overfitting, as it reduces the interdependence of hidden units.

4. Batch Normalization

Batch normalization is another recent innovation in neural networks [6]. As training progresses, layer inputs' distribution can change, thus making the training process less efficient, as each layer has to learn to adapt to new input distributions. Batch normalization essentially ensures that the distribution of the layer inputs are consistent for each training batch. It has been demonstrated that batch normalization makes it possible to have higher training rate, thus accelerating the training process.

IV. Approach

1. Dataset

The model was trained and evaluated on the Facial Expression Recognition 2013 dataset (FER-2013) created by Pierre Luc Carrier and Aaron Courville [2]. The dataset consists of 35,887 images queried using Google image search API and hand-labeled with 7 emotions. The resulting images are resized to 48×48 pixels, grayscaled, and stored in `.csv` format. Of the 35,887 images, 28,709 are designated as training samples, 3,589 are public test samples, and 3,589 are private test samples. Figure 1 shows a random sample of several images from the dataset. Note that the images are not all necessarily real human faces, but can be cartoons. A very small proportion of the

dataset is either misclassified or noise.

Figure 1: *Sample images from the FER-2013 dataset*

The emotion distribution of the training set is shown in figure 2. As can be seen, the 7 emotions are not equally distributed. The emotion 'Happy' makes more than a quarter of the training set, while 'Disgust' takes up only 2%. The uneven distribution might affect the final evaluation metrics of models trained on this dataset. I converted the dataset from `.csv` format to numpy[4] arrays for ease of access.

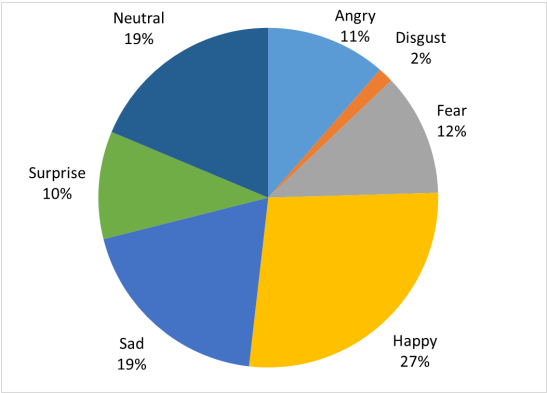

Figure 2: *Emotion distribution of training set*

2. Preprocessing & Augmentation

In order to improve the model, the dataset was preprocessed and randomly augmented during the training process. An experiment in 2017 by Pilatoka demonstrates that preprocessing and augmentation can considerably improve the accuracy of CNN models for facial emotion recognition tasks [11]. The exact preprocessing steps employed are as follow:

- The face in each image is located using the Haar Cascade classifier from the open-source library OpenCV version 3.4.1 for Python.[5] Since OpenCV is not always accurate, if no face is found, the image is simply returned; otherwise, the face is then resized to 48×48 pixels and used instead of the original image.
- For each image, subtract the mean pixel value of that image from all pixels. This is called 'sample-wise zero-centering.'
- Calculate the standard deviation across the entire training set, then scale all images by the calculated value. This is called 'feature-wise standard deviation normalization.'

Data augmentation consists of:

- Randomly flipping an image horizontally.
- Randomly rotating an image by at most 5 degrees, either clockwise or counterclockwise.

Note that preprocessing is done for all training, validation, and test samples, while augmentation is only done for training samples.

3. Model Architecture

I used the library TFLearn,[6] which provides a simple API for Tensorflow,[7] to design my model. The architecture of my model was based on the model used by Gudi [4] and Correa et al [1], with several modifications. Gudi's model is a shallow CNN with 3 convolutional layers, 1 penultimate fully connected layer with 3072 nodes, and a final softmax output layer with 7 nodes for 7 emotions. Initial attempts at replicating the result of Gudi failed to produce any

[4] http://www.numpy.org

[5] https://opencv.org
[6] http://tflearn.org
[7] https://www.tensorflow.org

tangible progress, probably due to not having enough information about hyperparameters as well as insufficient processing power and time. To reduce the long training time, I introduced one max pooling layer after the second convolutional layer, following the work of Correa et al. This has the effect of reducing the dimension of tensors going into the third convolutional layer by a half (18×18 to 9×9). While the training time per epoch was indeed shortened, the model still failed to make any substantial progress.

Because of the proposed properties of batch normalization (discussed in subsection 3 of III), I decided to introduce a batch normalization layer after each convolutional layer and the penultimate fully connected layer, and was able to achieve promising results. To update the gradients, I used stochastic gradient descent (SGD) with momentum to accelerate the learning rate when the model is moving in the right direction. Applying batch normalization throughout the network allowed me to raise the learning rate of the learning function fairly high [6]. The loss function is calculated using categorical cross-entropy. All convolutional layers and the penultimate fully connected layer use ReLU for activation. The final model architecture is illustrated in figure 3.

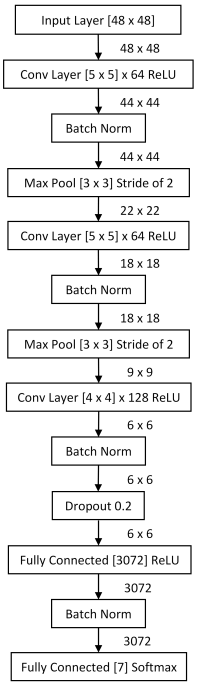

Figure 3: *Final model architecture*

4. Real-time Prediction

To demonstrate the application of my model, I wrote a Python program using the OpenCV library to capture images from webcam and predict the emotion for each face detected. The program can also be used to predict emotions for a single image or all images in a directory, or for a video. For each face, the program draws a green boundary box around the face and the predicted emotion on top, along with a probability (figure 4). The program sometimes fails to detect any faces because it depends on the Haar Cascade classifier provided by OpenCV, and in those cases, no image will be given to the CNN model.

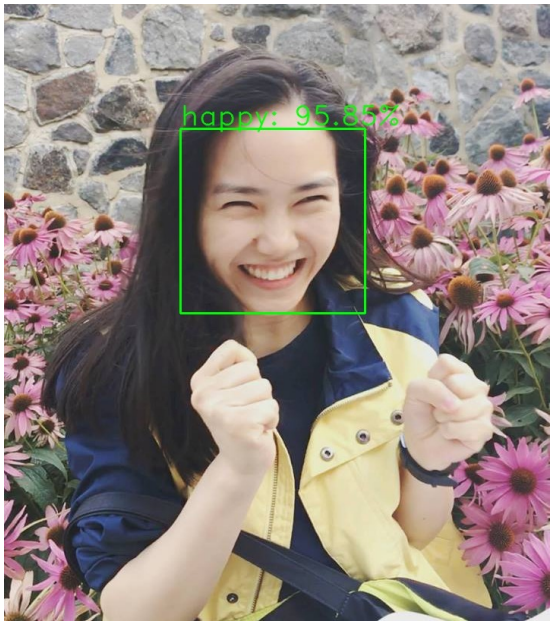

Figure 4: *Sample prediction result.*
The person's face is predicted to be 'Happy', with 95.85% probability

V. Experiment & Analysis

I trained my model on a Unix machine with Intel® Xeon™ E5-2670. The machine did not have the necessary requirements for training with GPU. It took 40 hours to train the model for 80 epochs, or half an hour for each epoch. After some tuning of the hyperparameters using cross-validation, I found that these values for the hyperparameters achieve the most promising result:

- Number of epochs: 80
- Batch size: 50
- Learning rate: 0.01
- Learning rate decay: 0.99
- Learning rate decay step: 250

The model's accuracy on the validation set after epoch 80 is 64.25% (figure 5). As can be seen, from epoch 20 onwards, training accuracies are higher than validation accuracies, indicating that the model is overfitting. Training accuracy appears to be still increasing after epoch 80, while validation accuracy appears to be plateauing. The test accuracy of the final model is 65.33%. Note that random guessing would achieve an accuracy of 14.28%, assuming that the emotions are equally distributed. Relative to the accuracies of the top models in the Kaggle competition, my model would be placed in the top 5.

To better understand the performance of my model for each emotion class, a confusion matrix of the prediction result on the test set is created (figure 6). 'Happy' has the highest accuracy of all emotions (83.2%). Interestingly, even though 'Disgust' constitutes only 2% of the training set, its accuracy is relatively high compared to more well-represented emotions. It is also interesting to note that, 'Angry' is frequently misclassifed as 'Fear' or 'Sad', 'Digust' is frequently misclassified as 'Angry', 'Fear' misclassified as 'Sad', 'Surprise' as 'Fear', and 'Neutral' as 'Sad', indicating possibly substantial similarities between those emotions. The average accuracy for each class is 65.5%, which is higher than that of Gudi's model (59.6%) by almost 6 percent points.

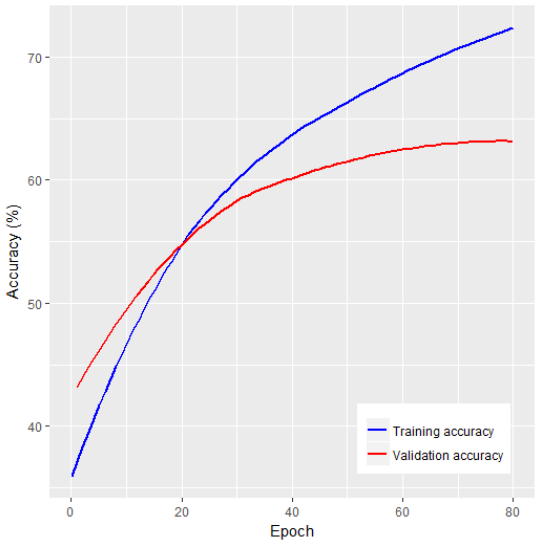

Figure 5: *Smoothed training and validation accuracy.*

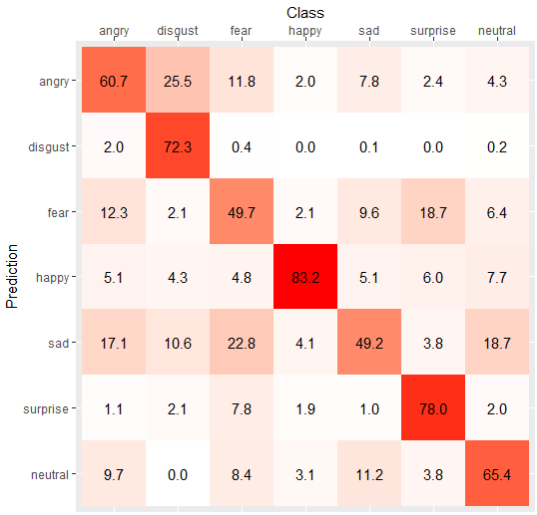

Figure 6: *Confusion matrix for results on test set.*

The graph of the loss value is shown in figure 7. Judging from the shape of the graph, it appears that the loss values are not decreasing very fast. This suggests that the learning rate might be too low, or the learning rate decay is too high. A higher learning rate or a lower learning rate decay might be able to reduce

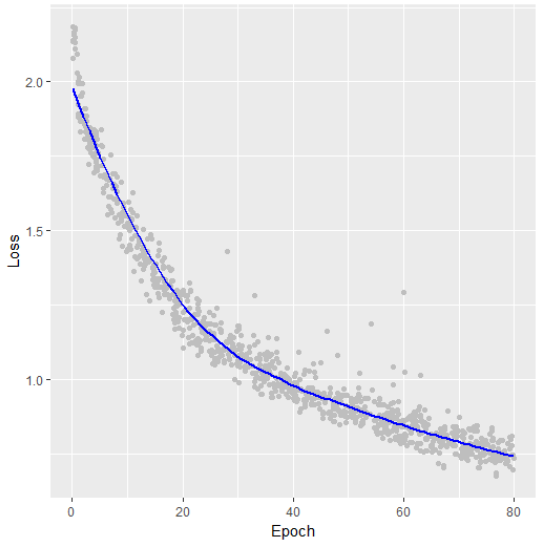

Figure 7: *Loss values during training.*

the loss values faster. However, due to limited computing power and time, these experiments will have to be scheduled in the future.

VI. Conclusions

Like most other Computer Vision tasks, facial emotion recognition can be tackled with Convolutional Neural Networks, which regularly outperform conventional machine learning methods that depend hand-crafted features [7]. Data preprocessing and augmentation techniques can help enrich the dataset and improve the accuracy of CNN models. The use of batch normalization applied throughout the network has been the key to my model's performance.

There are plenty of room for improvements: For start, the number of epoch needed for my model to achieve an accuracy over 60% is fairly large compared to other models on the same task and dataset [4] [1] [12]. The lack of GPUs for training means that more time is needed, fewer models can be tested, model architectures are simpler and shallower, and hyperparameters are less likely to be tuned optimally. Access to more powerful hardware will definitely remedy some, if not all of these issues. A larger and more evenly distributed dataset can help increase the performance of my model and reduce overfitting.

Future experiments with deeper CNN architectures, larger datasets with more poses and viewpoints, and different regularization techniques will be possible with access to sufficient hardware. A different face detection mechanism other than the Haar Cascade classifier provided by OpenCV can also be tried to improve the face detection rate of the real-time application.

VII. Appendix

Some samples from my real-time application:

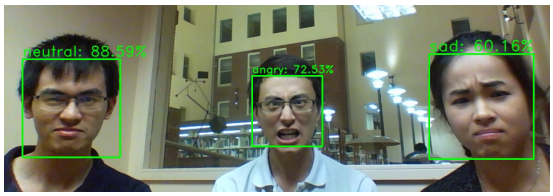

Figure 8: *Real-time prediction with multiple people.*

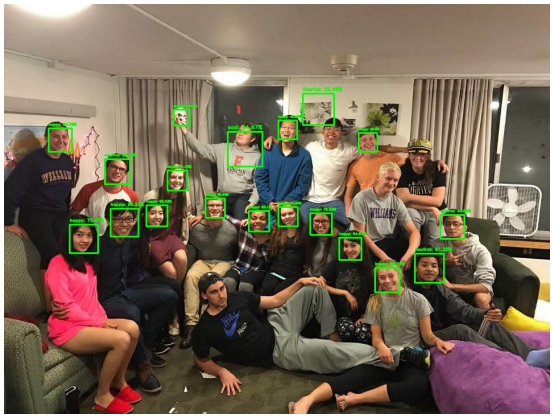

Figure 9: *Real-time prediction with a large group. Some of the faces are not detected, and some non-human objects are classified as faces.*

References

[1] Correa, E., Jonker, A., Ozo, M. and Stolk, R. (2016). Emotion Recognition using Deep Convolutional Neural Networks https://github.com/isseu/emotion-recognition-neural-networks 2, 4, 7

[2] Goodfellow, I., Erhan, D., Carrier, P. L., Courville, A., et al. (2013). Challenges in Representation Learning: A report on three machine learning contests. http://arxiv.org/abs/1307.0414v1. 2, 3

[3] Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning. *MIT Press*. https://www.deeplearningbook.org 2, 3

[4] Gudi, A. (2016). Recognizing Semantic Features in Faces using Deep Learning. *CoRR*. https://arxiv.org/abs/1512.00743v2. 2, 4, 7

[5] Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580v1. https://arxiv.org/abs/1207.0580v1. 2, 3

[6] Ioffe, S., and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, abs/1502.03167. https://arxiv.org/abs/1502.03167v3. 1, 3, 5

[7] Ko, B. (2018). A Brief Review of Facial Emotion Recognition Based on Visual Information. *Sensors*. 2, 7

[8] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. *Proceedings of the 25th International Conference on Neural Information Processing Systems*, Volume 1 (NIPS'12), F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 1. Curran Associates Inc., USA, 1097-1105. 2

[9] LeCun, Y. (1989). Generalization and network design strategies. *Technical Report CRG-TR-89-4*, University of Toronto. 2

[10] Lecun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional Networks and Applications in Vision. *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems:* Nano-Bio Circuit Fabrics and Systems, 253-256. 10.1109/ISCAS.2010.5537907. 2

[11] Pitaloka, D. A., Wulandari, A., Basaruddin, T., and Liliana, D. Y. (2017). Enhancing CNN with Preprocessing Stage in Automatic Emotion Recognition. *Procedia Computer Science*, Volume 116, 523-529. ISSN 1877-0509. https://doi.org/10.1016/j.procs.2017.10.038. 4

[12] Tang, Y. (2013). Deep learning using linear support vector machines. *CoRR*, abs/1306.0239v4. https://arxiv.org/abs/1306.0239v4. 2, 7