Facial Expression Recognition with Intensity Normalized Images and CNNs

Sean Levorse
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623
sml2565@rit.edu

Abstract

Facial expression recognition is one way that a computer can process human emotion. This paper examines one way in which CNNs can be used to categorize images of faces into the emotions that they express. One of the problems faced in facial expression recognition is that the same expression can look very different. Lighting, facial hair, skin color, and other factors can make it difficult to categorize images. To combat the issue of lighting and skin color, the images in this project undergo a little bit of preprocessing to normalize the intensity of the images. The processed images are fed into a convolutional neural network. The network is trained using data from the 2013 Kaggle Challenge for Facial Expression Recognition. The dataset contains 28,709 pre-labeled images for training and 3,589 of each public and private validation images. After training, the resulting training accuracy was 0.5294 and the validation accuracy was .4717.

1 Introduction

This project approaches the problem of characterizing emotions based on an image of a person's face. Facial expression recognition involves taking a picture of a face and determining what emotion that person is expressing based on the image. Facial expression recognition can be used to improve the interaction between computers and humans. Understanding emotion can help a program make decisions about how to communicate with a user. This is particularly applicable in robotics and artificial intelligence. To break down facial expression recognition, the project uses seven emotions classifications provided by the labels in the dataset. These emotions can be used as categories to build a classifier around. A convolutional neural network can be used to take in images and produce a classification into one of the seven categories that represents an emotion. This method is used over other newer methods due to constraints with the project's 2 week schedule, no access to a GPU, and no prior experience with machine learning for computer vision.

One problem generally encountered when attempting facial expression recognition is localization. Faces can be anywhere in the image. Another common problem with images used for facial expression recognition is the impact of lighting and skin color. These can shift the values of pixels in the entire image to be higher or lower than pictures of the same expression, which can throw off training. This project solves the localization issue by using a dataset that contains images centered around the face. To limit the effects of light and skin color, the images are preprocessed by normalizing the intensities.

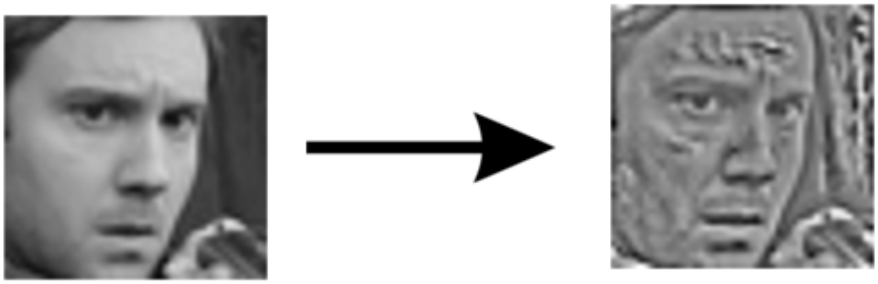

Figure 1: An image from the dataset on the left run through the normalization process output on the right

2 Past related work

As the dataset comes from a Kaggle Challenge that was presented in 2013, there is already existing work in the field. Much of the recent research seems to be about addressing the localization problem when it comes to recognizing expressions on a face that could be anywhere in an image. The paper by Lopes et al. addresses this in their paper Lopes et al. [2017]. They discus how merely localizing the face is not enough to isolate it. The problem is that faces can be in any rotation which can pose an issue. Their solution is to locate the centers of the eyes and use those to rotate all images to a common orientation.

Along those same lines, the paper by Valstar et al. focuses on other orientation problemsValstar et al. [2017]. They bring up how most facial expression recognition research uses posed images that are taken head on with decent light. This presents a problem when the system needs to analyze expressions in more extreme scenarios. They propose a challenge for facial expression recognition systems to function using nine different angles and to surpass the benchmark data they collected.

More relevant to this dataset is the paper "Multiclass Learning With Partially Corrupted Labels." The Kaggle dataset contains some mislabeled images amongst many correctly labeled images. This paper proposes a way to deal with this. They consider mislabeled data points to be a type of noise within the data Wang et al. [2017]. They suggest using a technique called importance reweighting to limit the impact noisy labels have on the weights. Their technique works for Support Vector Machines, which are a potential alternative to the softmax classifiers used in this paper's model.

The paper, by Liu et al. describes how even though the process for creating a facial expression recognition system is pretty well defined, there is still a large need for studies that try out and benchmark different combinations of systems. In general, facial expression recognition involves first feature learning, then feature selection, then classifier selection. Mixing and matching different techniques for each phase is cumbersome and requires an in depth study for each combination. They suggest an architecture for iterating through each of the three stages to assist in this search Liu et al. [2014].

3 Proposed approach

The approach this network takes starts by normalizing the image intensities using the same intensity normalization discussed in Lopes et al.Lopes et al. [2017]. The image is filtered using $x' = \frac{x-\mu_{mhgx}}{\sigma_{mhgx}}$. Here, x is the pixel in the source image, x' is the pixel in the normalized image, μ_{mhgx} is the Gaussian average of a 7x7 neighborhood around the pixel, and σ_{mhgx} is the standard deviation of that neighborhood. The resulting images are invariant to light and skin tone. Figure **??** shows the effect the normalization function has on images.

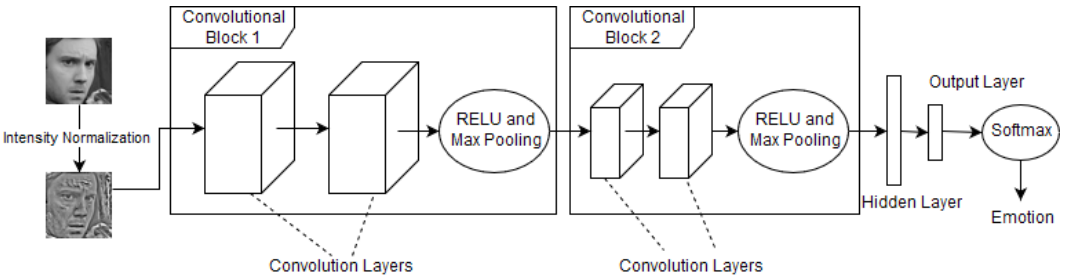

Figure 2: The model of the CNN used to recognize facial expressions

The dataset comes from the 2013 Kaggle Challenge for Facial Recognition in the form of a CSV file. Each line includes an integer emotion label, the 48x48 image data flattened into one dimension, and a usage. Some of the images in the dataset are not labeled correctly. The images are loaded and passed into the normalization function. The normalized images are divided into Training, Public Validation, and Private Validation, based on the usage in dataset. For this project, the Public Validation images were used as validation data during training, while the Private Validation were used to test the prediction after training. To save processing time, the normalized images are stored in a serialized file, along with all of the other data from the CSV to preserve the orders. The normalized training images are then fed into a convolutional neural network.

The CNN is created using a Keras Sequential model. The model is divided into 2 convolution blocks, and one fully connected hidden layer with an output layer. The first convolution block has two identical convolutional layers. Each of these layers uses 14 filters and a kernel size of 5x5. The first feeds into the second which feeds into a relu activation layer. This convolution block is pooled using max pooling to bring the input size down to 24x24x1 for the next convolution block. This also has two identical convolution layers, each with 8 filters and a kernel size of 7x7. Again, the convolutional layers feed into a relu activation layer which is max pooled, reducing the input down to 12x12x1. Now, the output is flattened feature vector that is passed through a deep net with one fully connected hidden layer. The hidden layer has 144 nodes that feed into 7 output nodes. Between the hidden layer and the output layer, there is a dropout layer with a dropout of 0.5. The Keras model is compiled to use the categorical crossentropy loss function and the rmsprop optimizer. The output of the network is run through a softmax function to determine which of the emotions the picture is most expressing. Figure **??** presents the model in graphical form.

To train the model, all 28,709 normalized training images are fed into the network over 20 epochs using the Keras fit function. The weights are stored in the 'weights.h5' file and can be loaded into the model after execution. For prediction, any input image must be 48x48 pixels in greyscale. First the image must be normalized, which can be done using the normalizeImage function in helper.py. Then, the normalized image can be run through the network with the Keras predict function. The output is an array that can then be passed through the softmax function in helper.py.

4 Experiments and Results

This model is a result of some experimentation and research. The first model generated only had one convolutional layer and one output layer. It took in the images as provided by the data set. The model was also compiled using stocastic gradient dissent instead of rmsprop. The highest training accuracy this model achieved was 0.192 with validation accuracy at 0.155, barely better than randomly choosing.

In an attempt to increase the accuracy, the next iteration increased the number of layers to 2 convolutional layers and one fully connected hidden layer before the output layer. This did a little better with training accuracy up to 0.231 and validation accuracy at 0.210.

After reading the paper by Lopes et al., image intensity normalization was added to see if the lack of normalization was causing a problemLopes et al. [2017]. This model resulted in a slight increase in accuracy. At this point training accuracy was 0.258 and validation accuracy was 0.214.

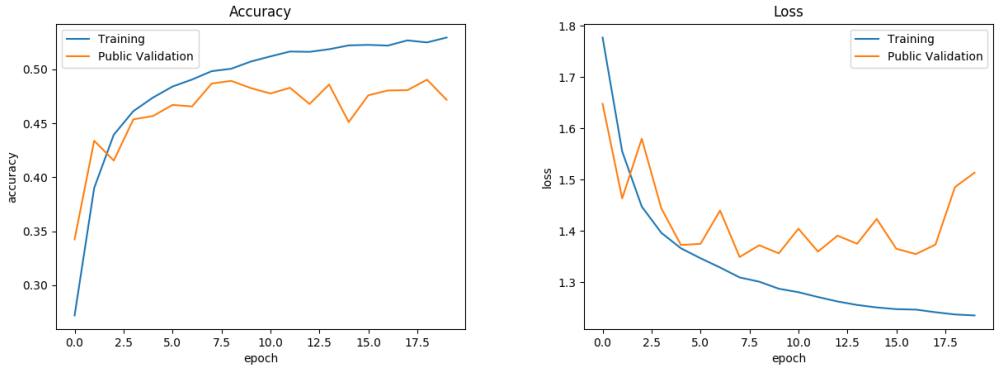

Figure 3: Left: Training and Validation accuracies over 20 epochs, Right: Training and Validation losses over 20 epochs

Next, the activation layers were switched from relu to sigmoid. However, this brought the training accuracy back down to 0.228. Next, the training time was increased from 10 epochs to 50. This brought the model's training accuracy up to .324 and validation accuracy of 0.260. However, this severely increased the runtime, so for testing, 10 epochs were used until a better model could be found.

While reading Chollet's article on Keras's blog, it was noted how using different parameters for the optimizer when compiling the model can changed how the model learnsChollet. Additionally, the model used in the article influenced the choice to have two convolutional blocks instead of one and to add the drop layer. After playing around with several optimizers, the rmsprop optimizer seemed to work the best. This brought the training accuracy up to 0.387 for just 10 epochs with validation accuracy at 0.341. At this point, each epoch averaged 290 seconds. Drawing from earlier conclusions, the number of epochs was raised to 50. Unfortunately, this might be too much data for Keras to process as the training loss became NaN after 30 epochs which dropped the training accuracy to 0.141 percent, approximately the same as randomly guessing. So the final model uses 20 epochs to ensure that this scenario doesn't occur.

5 Evaluation methods

The final training accuracy was 0.529 with a validation accuracy of 0.4717. Figure 3 shows the training and validation accuracies and losses throughout the training session.

5.1 Results and discussions

From the tests that were run, it is apparent that not all convolutional neural networks work the same, and there are a lot of changes and tweaking that can be done to improve the accuracy of a neural network for a given dataset. For this dataset, using multiple convolutional layers and multiple convolution blocks seemed improve the accuracy. Additionally, relu appeared to work better than sigmoid as an activation function. Increasing the number of epochs raises how high the accuracy will reach after training, however this only works to a certain degree. Performing a little preprocessing can help with training. Specifically, for this dataset, normalizing the images before feeding them into the network increased the training accuracy. The biggest factor for the model in this paper was the optimization function. Using the rmsprop optimizer was the only way to get the training accuracy over 50 percent. However, this seems to come at a tradeoff where the validation loss is inconsistently high as training goes on. This could be due to overfitting.

6 Conclusion and future work

From going through the process of setting up Keras and TensorFlow to creating a model, training it, and testing it, the machine learning process for computer vision has a lot of variables that may

impact the accuracy of the model. Without proper resources, testing a model takes a long time. Normalizing all of the images in the entire dataset took the CPU it was running on 75 minutes. And that had to be fed into the Keras model. It made sense to save as much information as possible so that the program could be run the fewest amount of times. This also lead to creating a lot of interfaces for saving and loading processed information to save on time for tweaking the system further down the line.

As a result of the limited timeframe and limited resources, there are several considerations for the future of the project. There was not enough time to try out different regulizers and their impact on accuracy. The kernel sizes within the convolution layer were never tested, and it would be interesting to see how different kernel sizes change the accuracy. There was not a chance to explore alternative classifiers, such as support vector machines which could implement the importance reweighting. Also, could kernels that are more vertical or more horizontal instead of square be more relevant to faces as they tend to be symmetrical? Could pretraining with a more accurate dataset help this less accurate dataset? With more time, these could be tested out with the existing model.

References

Francois Chollet. Building powerful image classification models using very little data. URL `https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html`.

Ping Liu, Shizhong Han, Zibo Meng, and Yan Tong. Facial expression recognition via a boosted deep belief network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

André Teixeira Lopes, Edilson de Aguiar, Alberto F. De Souza, and Thiago Oliveira-Santos. Facial expression recognition with convolutional neural networks: Coping with few data and the training sample order. *Pattern Recognition*, 61(Supplement C):610 – 628, 2017. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2016.07.026. URL `http://www.sciencedirect.com/science/article/pii/S0031320316301753`.

Michel F. Valstar, Enrique Sánchez-Lozano, Jeffrey F. Cohn, László A. Jeni, Jeffrey M. Girard, Zheng Zhang, Lijun Yin, and Maja Pantic. FERA 2017 - addressing head pose in the third facial expression recognition and analysis challenge. *CoRR*, abs/1702.04174, 2017. URL `http://arxiv.org/abs/1702.04174`.

R. Wang, T. Liu, and D. Tao. Multiclass learning with partially corrupted labels. *IEEE Transactions on Neural Networks and Learning Systems*, PP(99):1–13, 2017. ISSN 2162-237X. doi: 10.1109/TNNLS.2017.2699783.