Smile Detection using a Convolutional Neural Network

Vincent Crabtree, Daniel Kobold
Fall 2019

INTRODUCTION

The human range of emotions is quite complex yet can be summarized into positive and negative emotions. Whereas other people can generally identify another's emotion quite accurately in real-time, machine systems must rely on image and video processing to receive information regarding a person's emotional state. In order to narrow the scope of signs of emotions, we will focus on the face, and will specifically use the curvature of the mouth to make a prediction as to the emotion being expressed. By detecting a smile or frown, our system will classify the expression in a facial image as either positive or negative after being trained on a large dataset of images containing faces.

For training and testing purposes, we are using the LFW Crop dataset [11][12], which contains many images of faces labelled as either smiling or non-smiling. This dataset is ideal to our problem due to the format of each image, but our system will include preprocessing functionality that will allow the network to detect smiles or non-smiles in images that are not from the dataset, and may need adjustment prior to being input into the neural network. Given any image, the system will preprocess and then determine whether the image contains a smile or a non-smile, given that a face is present in the image.

To accomplish this task, we are implementing a Convolutional Neural Network (CNN). Preprocessing steps include rescaling the image to the intended square shape to make it easier to filter the image. Recoloring will also be made possible for use with images outside of the LFW Crop dataset which may be in color.

The end goal for this system is to allow more accurate emotion detection using image processing and a convolutional neural network. There are numerous applications for an emotion recognition system in fields such as psychology, criminal justice, and human-machine interaction in general. Within psychology, identifying the general emotion of a person can be used in counseling as well as psychological tests or experiments. In criminal justice, emotion recognition could help with lie-detection during interrogations. Regarding human-machine interaction, it will be important for machines to recognize the emotions a person feels when conversing, just as it is important for humans to do that as we converse with each other.

RELATED WORK

“Emotion Recognition: A Pattern Analysis Approach” written by C.-Y. Chang, et. al. [4], offers a more in-depth look at facial feature extraction, facial recognition, and facial expression recognition.

W. Hu et al. [10] discuss the use of deep CNNs to classify hyperspectral image. S. Lawrence et al. [9] discuss a new technique that combines a convolutional neural-network with a self-organizing map neural-network in order to provide higher accuracy in identifying faces. D Ciresan et al. [5] describes methods by which pattern classification can be greatly improved when used on images of numerical digits. A deep convolutional neural-network (DNN) and multi-column DNN are used, and the authors show that they are very effective for classifications ranging from digits to faces. Unfortunately, these networks are computationally intensive.

A. G. Howard [1] details multiple ways of improving the performance of CNNs by using methods such as adding more transformations to the training dataset and using higher resolution images. Our approach will include some of the techniques described such as extending image crops into extra pixels and manipulating the color of images (namely using grayscale images). Krizhevsky et al. [2] also offers a variety of methods to improve network accuracy including gathering larger datasets, creating more robust models and reducing overfitting through a few techniques. One technique discussed is data augmentation, which involves enlarging the given dataset by adding transformations of the images to the set alongside the original images (while preserving the accuracy of the labels). Another technique is called “dropout” and is achieved by setting the output of each neuron on the fully-connected layer to 0 with probability 0.5.

The study conducted by P. M. Ferreira, et. al. [8] used the idea of separating the emotion recognition process into different components including the “facial parts” component, the “representation” component, and the “classification” component. The “facial parts” component was used for making the relevance map, the “representation” component used the map to model the most relevant facial features, and the “classification” component categorized the model into an emotion category.

H. A. Rowley, et. al. [6] addressed a simplified problem of determining if an image contains a face. A. Verma, et. al. [3], present a new approach that involves identifying regions of an image that contains a face (allowing multiple faces to be detected) and also compared the approach to some of the classical emotion recognition approaches.

J. Kim, et. al. [7] compare a holistic facial feature approach with a geometric approach using the shapes of facial features to determine the emotion on the face. Another contribution by this report was the suggestion of multiple datasets that were considered when identifying a training and testing set.

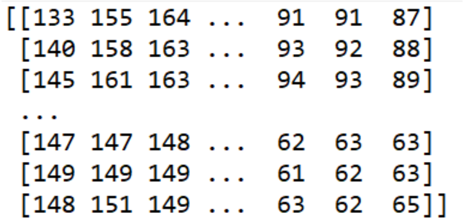
METHODOLOGY

INPUT AND PREPROCESSING

For initial training we are using a portion of the LFW Crop Face Dataset and will use another portion for testing. During this initial period, we have very little preprocessing required, as all the images are in the same format. The preprocessing steps we are using involve the shape and scale of each image. Shaping and scaling are completed by resizing the image. We are using a 64x64 image size as the standard size, in part to allow a minimal amount of preprocessing to be used with the training and testing dataset. To convert the image to grayscale, we are first checking if the RGB values of the image differ from each other and are using convert to change the color if needed.

We first convert the image to an array, which becomes the input to the neural network. The input to this conversion is the image adjusted during preprocessing. An example input image and converted array is shown below.

Figure 1 Input image [11][12] followed by the image with two edge-detection filters applied.



```
[[133 155 164 ...  91  91  87]
 [140 158 163 ...  93  92  88]
 [145 161 163 ...  94  93  89]
 ...
 [147 147 148 ...  62  63  63]
 [149 149 149 ...  61  62  63]
 [148 151 149 ...  63  62  65]]
```

Figure 2 Sample of NumPy array from converted image.

TRAINING SET AND TESTING SET

The dataset we are using to train and test our network on is the LFWcrop Face Dataset. This set consists of over 13,000 images of faces that are either smiling or not smiling, and labeled as such. The images in the dataset are also cropped closely around the face, in order to reduce the amount of noise and redundant data in the picture. Each image is also 64x64 in size, making it easy to keep the code for the input layer consistent. The dataset will be divided into many images for the training set, and a smaller number for the testing set. In the final experiment, 4,000 images were used for training and 500 were used for testing.

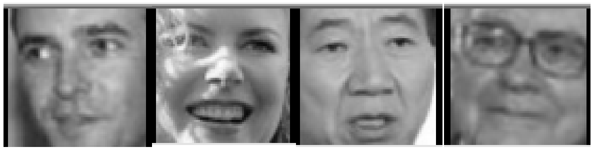


Figure 3 Example images from LFWcrop dataset [11][12].

ALGORITHM

The core structure of our methodology is a simple, single-layer Convolutional Neural Network (CNN). The goal of this CNN is to detect characteristics in the image that are indicative of either the presence or the absence of a smile. This CNN will feed an input image into a convolutional layer that slides filters across the entirety of the image to capture important features. The output of this layer is fed into a pooling layer, which uses a max pooling process in order to down sample the image by removing redundant data, making the image simpler to classify. Next, the image is flattened into a single vector which is passed through a fully-connected layer that uses the soft max activation function to create a probability distribution across the possible outcomes (smile/non-smile) used to classify the image and create a prediction output. In order to train the network, we will calculate the error or loss by comparing the predicted value output by the network to the actual label of the image in order to determine accuracy. Based on the calculated loss, gradients are backpropagated though each layer of the network to update their weights accordingly. This process is repeated for many images until the calculated loss is low enough to consider the network accurate.

The input layer of the CNN is the image file passed to the network, which is converted into an array of values. Since the image is in grayscale, the image will only need to be represented as a two-dimensional array. Each pixel of the image is represented by an intensity value which in this case is an integer ranging from 0 to 255.

The convolutional layer consists of multiple filters (currently two) that are slid across the input image pixel-by-pixel until the entire image has been covered, creating an output image for each filter. Each element of each filter acts as a weight in a normal neural network and is updated during training via backpropagation. Our network currently utilizes 3x3 filters which detect horizontal and vertical lines (edges). It also supports configurations for any number of randomly-initialized filters. We are using valid padding, so no extra 0-valued elements are added to the array in this layer. For each step of convolution, the filter weights are multiplied by the values in the image that they cover, and the results of these multiplications are then summed to form the value in the corresponding element of the output image. The size of the output for this layer is 62x62, times the number of filters used on the image.

The pooling layer works in a similar fashion to the convolution layer, as a pooling filter slides across regions of the image to create the output image. We are using max pooling for our network, so each pooled region is down sampled to the highest value in that region for the output. Our initial version uses a 2x2 pooling filter, with stride of 2. This filter is then applied to every layer of the output from the convolutional layer. The result/output is a set of down-sampled arrays that contain pooled feature maps that are one fourth the size of the convolved arrays.

The final layer of the network is the fully-connected layer which is where the actual classification takes place. Before this can happen, the output from the pooling layer must be flattened into a one-dimensional array with size equal to the dimensions of the max pooling output, multiplied by the number of filters. Next, as in a regular neural network, the input is multiplied by a set of weights and offset by a set of biases. This layer is comprised of fully-connected neurons equal to the size of the flattened input (961*5 filters in this case). The flattened array is passed to the SoftMax activation function to turn the values output by the pooling layer into probability-based classifications. This function is shown in Figure 4 below. This ensures that the output is a probability in the range of 0 to 1 for each provided number and gives the probability distribution across all possible outcomes (smile and not smile in our case). The SoftMax function makes it easy to add more outcomes such as other emotions if desired.

For each value $x_i \; where \; 0 < i < n$:

$$SoftMax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Figure 4 SoftMax Function [7]

The loss function will compare the predicted values (output of entire network) generated by the SoftMax function to the expected values as given by the labels in the dataset. We are using the Cross-Entropy loss function, because it measures loss on values that have probabilities between 0 and 1. Cross entropy loss works by taking the negative natural log of the predicted value for the expected outcome, measuring the discrepancy between the prediction and the actual value. During training, this loss is backpropagated through the network by taking the derivative of the loss with respect to the output of the SoftMax to update its weights, and then sending the derivative of the loss with respect to the input of the SoftMax to propagate to the output of the pooling layer. This process is repeated for each layer to update the weights and improve accuracy.

$$L = -\mathbf{y} \cdot \log(\hat{\mathbf{y}})$$

Figure 5 Cross-Entropy Loss Function [13]

Where y = actual label, y^ = predicted

While training the network we will propagate through the input images one-by-one through each layer of the network until the loss is calculated for that image. Then we will use that loss to start the backward propagation phase of training to update all the weights and biases in each layer of the network. The weights and biases in each layer are updated using the calculated loss. Each iteration, the derivative of the loss is calculated with respect to the weights and biases. This change is multiplied by a learning rate and subtracted from their values, ultimately causing the weights and biases to converge to the proper value over time to predict the outcome correctly.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial out} * \frac{\partial out}{\partial t} * \frac{\partial t}{\partial w}$$
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial out} * \frac{\partial out}{\partial t} * \frac{\partial t}{\partial b}$$
$$\frac{\partial L}{\partial input} = \frac{\partial L}{\partial out} * \frac{\partial out}{\partial t} * \frac{\partial t}{\partial input}$$

Figure 6 Weights, Biases and Input Delta Calculus [13]

*Where L = Loss, w = weights, b = biases, t = totals (input * weights + biases)*

Our proposed algorithm has many similarities to the basic structure used by A. Verma, et. al. [3]. The procedure was expected to be similar, and our planned contribution relates to our method of isolating the mouth as an emotion identifier, and in the resulting comparisons to methods involving more than the mouth. The primary difference is the use of images labeled as either smiles or non-smiles, which should simplify the problem. In addition, many images will be labeled to increase the number of labeled images for use in this and future experiments. We also hope to make other contributions regarding performance, which is evaluated at a later stage in the development process.

Training and Testing

The output of the network is a predicted probability distribution across the possible outcomes (in this case smile and non-smile) given by the SoftMax equation. Then loss is calculated on this distribution using the cross-entropy loss function. The network is trained by iterating through the dataset of images and passing them through the network one-by-one. Each image is propagated through the forward portion of the network in order to calculate loss based on the accuracy of the prediction. The loss is then backpropagated through the layers to update the weights (and biases if applicable). After all of the training images from the dataset have been used to update the weights, a smaller, testing set is sent through only the forward part of the network to record its accuracy.

To test the effectiveness of the network, we vary a number of parameters in the training and testing process to see how they affect the prediction accuracy. The main parameters we vary are the number of training and testing images, to see how accuracy changes with training. The number of convolutional filters and learn rate are also varied in a similar manner. This data gives a better understanding of the best network configuration for our smile detection application.

IMPLEMENTATION

For the implementation of our convolutional neural network, we decided to use the NumPy library for the images, filters and results of each step of the network. We started with some basic edge detection filters which required making a multi-dimensional array of NumPy arrays and then required manually entering the values for the filters. To analyze the images, we used NumPy arrays. In order to convert the image to an array, we first had to set the program to iterate through each file in a folder named "input" by using the change directory command within the OS library. We then opened the image and put the image through the preprocessing function defined in the function.py file.

The only preprocessing required is to convert the image file into a NumPy array. Using the filters and the NumPy image, we completed the convolution operation for each filter individually for use in later layers. The convolution itself involved making an empty feature map array and iterating through the NumPy array and computing the convolution for the value to be placed in the feature map. We used two different functions to allow a list of the feature maps to be returned, which contained each of the feature maps. Next, the RELU function was used to replace the feature map values with the original values, if those values were greater than zero. Like before, a list of the output of the function was returned. This output became the input to the max pooling operation, which returns a smaller NumPy array representing the maximum values of regions that were programmatically iterated to in the max pooling function. These results became the input to the next layer.

Before passing the data to the fully-connected layer, it is flattened to a single dimensional list using the NumPy flatten() function. The fully-connected layer is implemented as a class with a forward method and backward method. The constructor takes the expected input size as a parameter and initializes the weights as a matrix of random numbers the same size as the input, with 2 rows (one for smile weights and one for non-smile weights). The biases are initialized as a 2-element array or zeros. The forward method first multiplies the input by the weights using the NumPy dot() function, then adds the value of the biases to each element. Then, the result is passed through the SoftMax function, which first uses the NumPy exp() function to calculate e raised to the output for each element of the dot product list. This calculation is the numerator for the SoftMax function. The denominator is the sum of all elements in the numerator and is calculated using the NumPy sum() function. The numerator is divided by the denominator and returned as a Python list that contains the predicted probability distribution for each possible outcome (2 in this case for smile and non-smile).

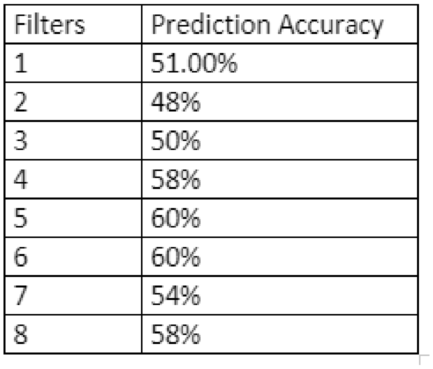
The loss function used is cross-entropy loss, which is after the fully-connected layer and takes the predicted probability distribution as an input. The cross-entropy function is implemented by using the NumPy log() function to take the negative log of the predicted value for the actual outcome. This loss value is then used as the initial input to the fully-connected layer during the backwards phase of the training cycle.

RESULTS

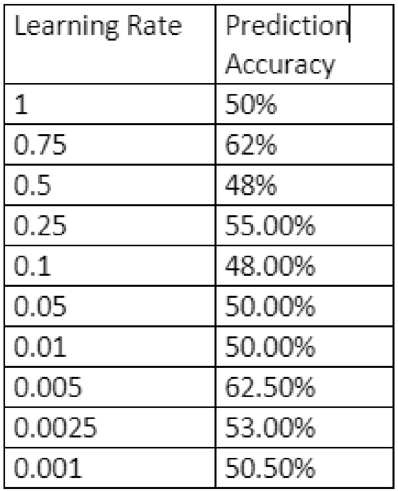
Early testing of the single-layer CNN suggested that the network was still “guessing”, or not properly predicting the presence or absence of a smile in the input image. In order to determine if this was the case, certain key parameters were varied and tested. These included the number of filters, the learning rate, the number of training images, and finally the number of layers in the CNN.

Before detailing the exact results, it is worth noting that repeated tests sometimes offered different results, which makes sense given that the training and testing set were randomly selected from a set of about 1,200 images labelled images. This random selection leads to different adjusted filters being developed during the learning process.

The results are shown in Tables 1, 2, and 3 where the number of filters, learning rate, and number of training images were varied, respectively. Experimentation showed that about five or six filters led to the highest accuracy, but only by a small amount. As for learning rate, the highest accuracy was found for a 0.005 learning rate. As for the number of training images, the accuracy tended to increase as the number of training images increased.

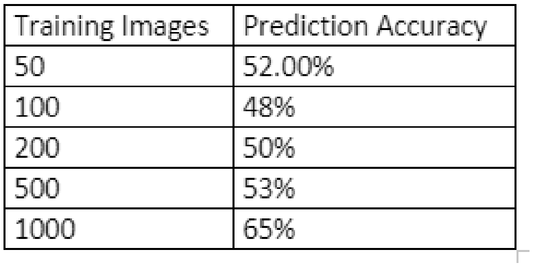


Filters	Prediction Accuracy
1	51.00%
2	48%
3	50%
4	58%
5	60%
6	60%
7	54%
8	58%



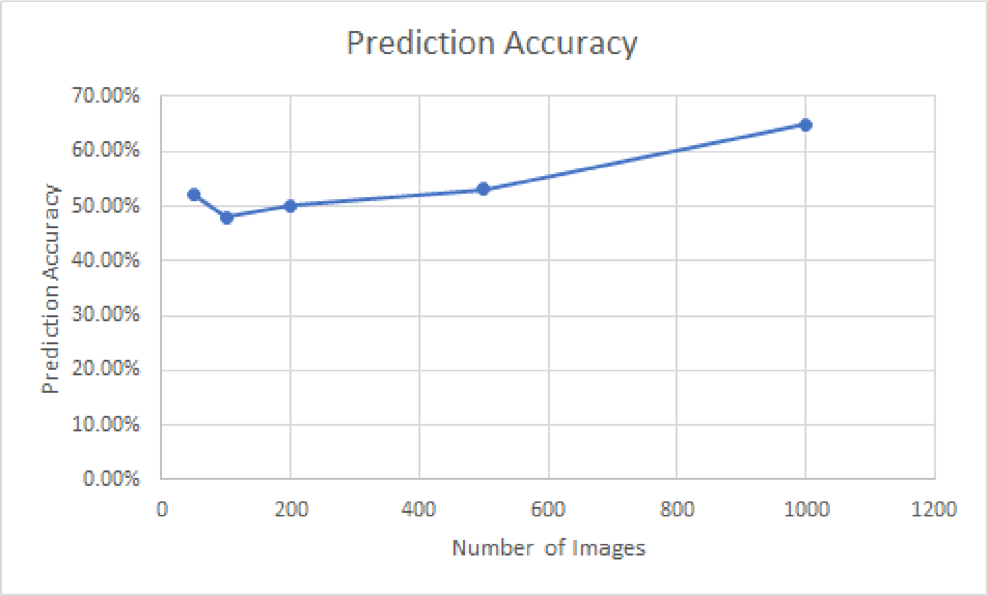
Learning Rate	Prediction Accuracy
1	50%
0.75	62%
0.5	48%
0.25	55.00%
0.1	48.00%
0.05	50.00%
0.01	50.00%
0.005	62.50%
0.0025	53.00%
0.001	50.50%

Tables 1 and 2 Prediction Accuracy for varied configurations (500 training iterations)



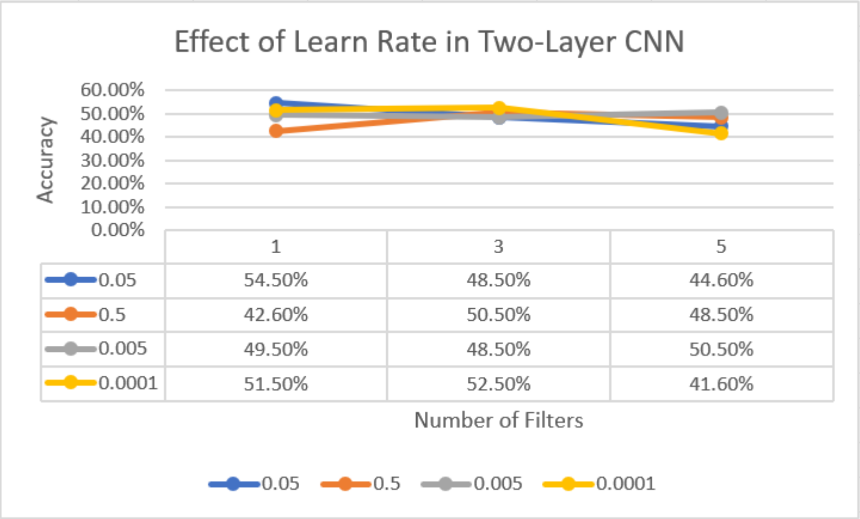
Training Images	Prediction Accuracy
50	52.00%
100	48%
200	50%
500	53%
1000	65%

Table 3 Prediction Accuracy vs. number of training images (500 training iterations)



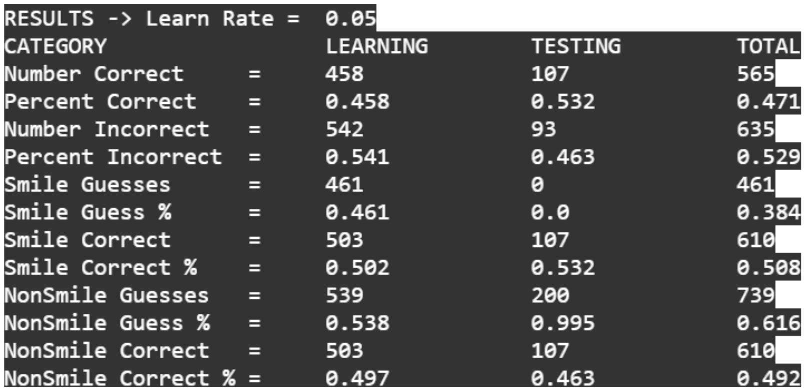
Graph 1 Prediction Accuracy vs. Number of Images

In an effort to increase accuracy, a second layer was added to the convolutional neural network. This involved adding a second instance of the convolution and pooling classes as well as adding the additional forward and backward propagations within the main program. The results of varying the above parameters are shown in the figure below, but in summary were inconclusive. The resulting accuracy percentages on average reflected a random choice. Upon analysis of the output of the program (which shows the number of smile and non-smile guesses), it was found that the network guessed either smile or non-smile 100% of the time for the given run of the program. Given that the data set contains approximately 50% smile images and 50% non-smile images, and that the images are selected at random, it follows that the guesses would converge to about a random guess. As a result, most of the testing covered here involved the single-layer convolutional neural network, as those results are more conclusive.



	1	3	5
0.05	54.50%	48.50%	44.60%
0.5	42.60%	50.50%	48.50%
0.005	49.50%	48.50%	50.50%
0.0001	51.50%	52.50%	41.60%

Graph 2 Accuracy of Varied Learn Rates and Filter Numbers



```
RESULTS -> Learn Rate =  0.05
CATEGORY                 LEARNING         TESTING          TOTAL
Number Correct     =     458              107              565
Percent Correct    =     0.458            0.532            0.471
Number Incorrect   =     542              93               635
Percent Incorrect  =     0.541            0.463            0.529
Smile Guesses      =     461              0                461
Smile Guess %      =     0.461            0.0              0.384
Smile Correct      =     503              107              610
Smile Correct %    =     0.502            0.532            0.508
NonSmile Guesses   =     539              200              739
NonSmile Guess %   =     0.538            0.995            0.616
NonSmile Correct   =     503              107              610
NonSmile Correct % =     0.497            0.463            0.492
```

Figure 7 Sample Output for Two Layer CNN, Five Filters, and Shown Learn Rate

The results of the experimentation on the CNN showed that varying parameters of the network changed the accuracy level, although an increase did not always lead to more accuracy, except in the case of increasing the number of training images. Additionally, increasing the number of layers in the network made the accuracy less conclusive, due to the results showing that using two layers made the network guess one result or the other exclusively. This led to the accuracy measurements naturally converging to 50%.

Although the accuracy is relatively low, the network has shown a small increase in accuracy as the training dataset grows in size. As learning rate and number of filters had only a small impact on the accuracy, and due to the complexity of the images being classified, the network likely needs a higher volume of training data in order to achieve higher accuracy.

Taking this into account, a program was written to allow quick and easy labeling of data to allow a larger training and testing data set. After labeling approximately 3,000 more images, additional testing was completed to determine whether the small number of labeled images was the problem. The results were promising and showed an increase in accuracy. For the additional tests, 4,000 training images and 500 testing images were used. In order to allow a comparison to the previous, smaller dataset, a learning rate of 0.05 was selected due to the 50% accuracy in previous tests. Using this rate, the number of filters was varied to allow multiple tests and both the training and testing accuracies are shown below because of the randomness of the selected images. The results are shown in Figure 4 below. These tests used the one-layer network due to the results of the previous testing.

Filters	Training	Testing
2	0.526	0.541
5	0.557	0.613
10	0.583	0.583

Table 4 Follow-Up Testing with Larger Labeled Dataset (Detailed Above)

The results of this final experiment confirm the theory that the previous accuracies were low at least in part due to the small number of labeled images. Increasing the sizes of the training and testing sets improved the accuracy significantly and adding filters to the new experiment also increased accuracy. While the accuracy is still not very high, the results could theoretically be improved by more manual labeling of images and further experimentation with the network's number of filters and learning rate.

CONCLUSION

Emotion detection is useful across a variety of applications, including criminal investigations and clinical trials in Psychology. One of the most important factors in accurately detecting emotion is whether or not the subject is smiling. There are many approaches to detecting a smile in a given image, often using some sort of vector such as a support vector machine. Another popular approach is using convolutional neural networks. Our project combines a well-tailored dataset with a CNN in an attempt to create an efficient smile detection system that is not overly computationally intensive.

There are many conclusions to be drawn from this project with perhaps the largest being that, in order to reliably use neural networks to solve a problem, a large amount of labeled data is needed. The relatively small portion of the dataset in use that was labeled caused a lower accuracy in the initial results, but by manually increasing the number of labeled images, the results were improved. While the performance contributions that were set as goals were not met, this project contributed to facial expression recognition through labeling many facial images and by detailing a descriptive method for using a convolutional neural network to identify smiles in images. The immediate future work that could be completed includes labeling the rest of the LFW crop dataset (around 8,000 images), completing further testing on the two-layer network, and running more tests in order to determine the optimal number of filters for use in smile recognition.

Further contributions include finding the optimal number of filters and the best learn rate for the smile detection problem. If the observed trend continues, a larger training dataset should continue to increase the accuracy of the system. The combination of well-preprocessed data and the straightforward neural-network architecture gives this system the potential to be an efficient smile detector while staying streamlined and very easy to extend if more features are needed. Due to the use of the SoftMax activation function, it will be easy to add new outcomes to the classifier in order to classify a larger range of emotions if desired.

RELATED WORK REFERENCES

[1] A. G. Howard, “Some Improvements on Deep Convolutional Neural Network Based Image Classification,” *ICLR 2013*, 2013.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *NIPS 2012*, 2012.

[3] A. Verma, P. Singh and J. S. Rani Alex, "Modified Convolutional Neural Network Architecture Analysis for Facial Emotion Recognition," *2019 International Conference on Systems, Signals and Image Processing (IWSSIP)*, Osijek, Croatia, 2019, pp. 169-173.
doi: 10.1109/IWSSIP.2019.8787215

[4] C.-Y. Chang and Y.-C. Huang, “A Subject-Dependent Facial Expression Recognition System,” in *Emotion Recognition: A Pattern Analysis Approach*, First., Hoboken, NJ: John Wiley & Sons, Inc., 2015.

[5] D. Ciresan, U. Meier, and J. Schmidhuber, “Multi-Coulmn Deep Neural Networks for Image Classification,” Feb. 2012.

[6] H. A. Rowley, S. Baluja and T. Kanade, "Neural network-based face detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 23-38, Jan. 1998.
doi: 10.1109/34.655647

[7] J. Kim, B. Kim, P. P. Roy and D. Jeong, "Efficient Facial Expression Recognition Algorithm Based on Hierarchical Deep Neural Network Structure," in *IEEE Access*, vol. 7, pp. 41273-41285, 2019.

[8] P. M. Ferreira, F. Marques, J. S. Cardoso and A. Rebelo, "Physiological Inspired Deep Neural Networks for Emotion Recognition," in *IEEE Access*, vol. 6, pp. 53930-53943, 2018.
doi: 10.1109/ACCESS.2018.2870063

[9] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, “Face Recognition: A Convolutional Nerual-Network Approach,” *IEEE Transactions on Neural Networks*, Jan. 1997.

[10] W. Hu, Y. Huang, F. Zhang, and H. Li, “Deep Convolutional Neural Networks for Hyperspectral Image Classification,” *Journal of Sensors*, Jan. 2015.

DATA SET REFERENCES

[11] C. Sanderson, B.C. Lovell.
Multi-Region Probabilistic Histograms for Robust and Scalable Identity Inference.
ICB 2009, LNCS 5558, pp. 199-208, 2009.

[12] G.B. Huang, M. Ramesh, T. Berg, E. Learned-Miller.
Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments.
University of Massachusetts, Amherst, Technical Report 07-49, 2007.

IMAGE REFERENCES

[13] Victor Zhou, “CNNs, Part 2: Training a Convolutional Neural Network,” *Victor Zhou*, 08- Aug-2019. [Online]. Available: https://victorzhou.com/blog/intro-to-cnns-part-2/.