Facial Expression Classification

Andrew Spittlemeister

Abstract:

This project consists of training and using convolutional neural networks (CNN) to classify the facial expressions of a human present in the input picture. Various levels of the CNN represent the extraction of different features relevant to the problem; this feature data is flattened and fed into a densely connected deep neural network to perform classification of the facial expression. The data used for training and validation of the CNN consists of 32,298 pre-cropped grayscale images of a human face making one of 7 facial expressions. The three-phase CNN delivers accurate predictions of up to 53% on a validation set of 3,589 sample images.

I. Problem Description

Analysis of human emotion or sentiment has many applications in the fields of psychology, behavioral studies, and most importantly; everyday human interaction. Research into human communication has indicated that facial expressions play a huge role in shaping context and non-verbal inflection of phrases and conversations. Drawing from this, it can be discerned that a significant amount of information regarding a person's mood or sentiment can be established by simply looking at their face, both holistically and with a more detailed and feature-focused analysis.

This project proposes to tackle the problem of automating the process of analyzing human facial expressions and predicting the person’s mood. More specifically, the program is required to take 2D image containing a face as an input, extract relevant feature data from the face in the image, and finally classify the face in the image to a set of predefined emotional states. Ideally, the entire process will be able to be executed in a near-real-time mode of operation which will lend the use of this project’s findings to a broader range of applications.

Given the time frame of the project, the program was developed in the Python language due to its high rate of development time (as well as its current, frequent use in data science and machine learning applications). The goal was to create a near-real-time application will not impede the accuracy of the classification on consumer hardware, but certain concessions can

be made to build an application worth using. Execution of code in the Python language is not inherently fast, so to combat this obstacle various optimization techniques were employed by using some of the existing machine learning and data manipulation libraries (namely Keras, Pandas, and Numpy) [13] [11] [12] to accelerate computation and optimal use of available hardware.

Finally, this problem that I am facing is one where I hope to contribute valuable information to the solution. When coding, testing, and implementing the application, I hope to develop information regarding techniques that are beneficial to the overall solution and techniques that are not. My goal is for this project to elicit concrete evidence on the use of certain methods over others in the realm of facial and emotional recognition. As a result, my code should reflect adequate techniques for extracting this information from the midst of the training and analysis process.

II. Literary Review

The task of trying to understand how the human visual system interprets facial expressions has been challenging the scientific community for decades. Prior to lending this problem to computer scientists, in 1978, a small group of psychologists and researchers led by Paul Ekman developed a framework used to describe any human emotion called the *Facial Action Coding System* (FACS). This well known system originally proposed that there are 44 fundamental actions, called *Action Units,* of individual elements in the human face that contribute to the identification of a specific expression [1]. Furthermore, Ekman expressed that facial emotions could be divided into 7 major categories: Anger, Disgust, Fear, Happiness, Sadness, Surprise, and Neutral. This claim has laid the foundation on which many automated expression analysis systems define their goals and categorize their data [2].

In 1996 Yaser Yacoob and Larry S. Davis proposed that the FACS model is limited in its descriptive potential as its previous applications were solely used with static images. They believed that the information on how the facial features move and deform over time provided a more valuable indicator to which category of emotion was being expressed. To extract relevant information from this newly defined temporal model, Yacoob and Davis devised an algorithm that uses multiple optical flow calculations over various regions of a human face to detect rigid and non-rigid motions. By using sequence of images taken over the time of which an expression is unfolding over the subject's face, the information used to categorize an emotion is formed

from tracking the motion of these regions. The relative motions are captured over 3 periods of the expression's lifetime. By using predefined expression motion data, an emotion with the highest similarity to the incoming data is chosen. Experimental analysis of this method yielded that the optical flow calculations needed to be more resilient to rigid and non-rigid motions of the subject and ultimately needed to employ a more complex model that has the ability to capture more information on the intricate elements of facial actions [3].

Later, in the same year as the publication using optical flow, Yacoob, Davis, and Mark Rosenbloom, put this information from the optical flow to use with a modern and robust classification method. The classification method employed was a radial basis function based neural network classifier. To accentuate the quality of the information that could define each output node of the network, instead of classifying image sequences into emotional categories, the network classifies the temporal information into individual stages of the expression's generation. Stages of an expression that are close to the chosen node's stage have an activation percentage that decreases according to a gaussian distribution as it's temporal distance increases. Thus, the one-hot encoded output vector for expression stages models a gaussian distribution centered at the stage the image is exhibiting. The inputs to the network are derived from the optical flow motion field; the field is decomposed into four binary motion fields (one for each cardinal direction) in which a pixel in the new images are active if its primary direction from the original optical flow image corresponds to this new image's direction. A separate network is trained for each expression. each of these is broken up into three subnetworks that analyze different aspects of the image sequence. The division of the problem into various levels of complexity showed promise but the act of honing in on separate features posed problem in neural network classification when the original subject in the image exhibits significant rigid motion or orientation changes [4].

A more modern approach to gathering spatial relationships from facial images is to train a deep convolutional neural network on a relevant data set. A recent paper, titled *Facial Expression Recognition Based on Deep Evolutional Spatial-Temporal Networks* by Kaihao Zhang (et al.) and a group from the National Laboratory of Pattern Recognition in Beijing, proposed using a standard convolutional deep neural network structure for recognizing and classifying holistic constructs about the face in the image along with using a second network to capture data over multiple frames.

As in the previously mentioned methods, this paper also used image sequences that displayed the evolution of an expression of the subject's face. The network dedicated to

analyzing the change of the expression over time is a part-based hierarchical bidirectional recurrent neural network (PHRNN) [5]. The PHRNN is given data points for certain low level features of the face (such as eyebrows, eyes, mouth, and nose) for each frame of the image sequence. Each of these feature sets is fed into an independent bidirectional recurrent neural network with enhanced memory capabilities. The output data from these independent networks is combined with data from networks whose input data is spatially related to its own. This process is repeated in a hierarchical manner until a single dataset can be obtained that has combined information from all initial sections of the face; this single set is then classified with a densely connected layer. The recurrent nature of the layered network captures information about the movement of features between frames of the sequence.

The second network used to capture spatial relationships of the expression is a four-layered convolutional neural network followed again by a densely connected layer for classification purposes. The first three layers consist of a convolutional, pooling, and normalization sequence that produces an increasing number of filtered images at each stage. Consequently, from the lack of padding and pooling operations, the size of each filtered image shrinks with each layer. The final convolutional layer neglects the pooling and normalization operations and produces a set of 40 2x2 images that are passed into an 80-node densely connected network for classification. Softmax classification output probability vectors from both spatial and temporal networks are input to a custom fusion function for a final decision to be made. This method achieved an accuracy rating of 98.5% on the CK+ database [6] and 81.18% on the MMI database [7], outperforming most other attempts [5].

III. The Data Set

The dataset used in this project for training the convolutional neural network is the *Facial Expression Recognition 2013 Dataset* produced by Kaggle.com [8] and was chosen for its high volume of labelled images as well as its availability to the public. This data set consists of 48x48 grayscale images of human faces expressing one of seven emotions (Anger, Disgust, Fear, Happy, Sad, Surprised, and Neutral). There are 28,709 images that were used for training the network and an additional 3,589 images available for testing the trained model. See examples of faces from this data set in Figure 1.

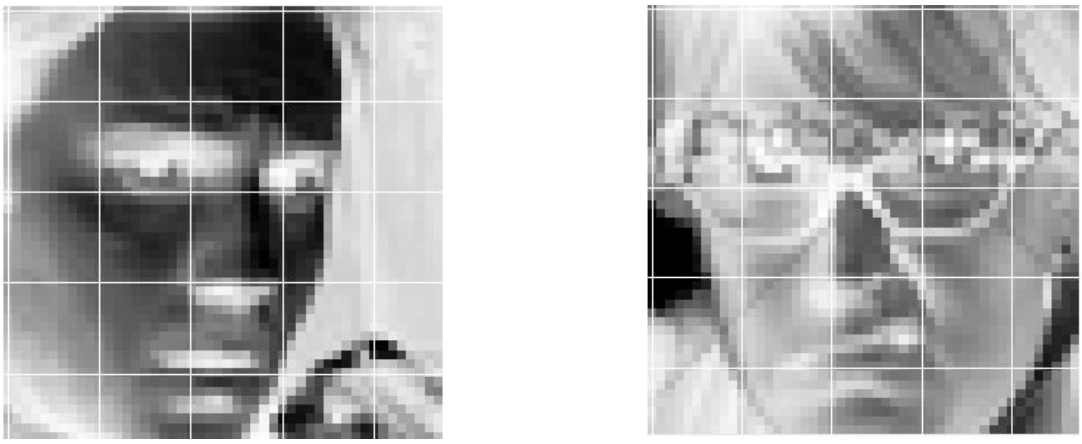

Figure 1: Sample images from the testing and training datasets.

As seen in the figure above, the resolution of these images is very low and for such a complex feature domain that facial expressions encompass, the result is a significant loss in valuable information. Additionally, it should be noted that this dataset does not contain sequences of images of a single subject to represent the unfolding of an expression over time. Unfortunately for this reason, I was unable to perform an analysis on temporal information. However, the images across both training and testing datasets provide a wide variety of faces in different poses, lighting conditions, and camera angles; and feature subjects who have varying ages, skin tones, hairstyles, bone structures, and even eyewear. In an ideal world, the classification model could be trained on a dataset consisting of only faces from the subject it will be used on, but this is impractical and not scalable for many applications. Data exhibiting these variances is very useful when attempting to bring a trained model out of a sanitized setting and into the real world.

The format of this Kaggle dataset took the form of a .csv file wherein each row included 1 tag denoting it for testing or training, 2,304 grayscale pixel values, and a 0-6 valued label denoting the expression. Python's Pandas library was used to parse these images into training and testing dataframes [11]. Visualization and processing these images through the network required batches of the images from the data frames to be converted into a vectorized array structure using Python's Numpy library [12].

IV. The Convolutional Neural Network

In the past a great deal of time and effort was spent on developing effective filters or kernels to convolve with an image matrix to highlight certain information and features about the image. Notable filters derived this way include the Sobel kernel for edge detection and the Gaussian kernel for smoothing an image. Extracting high level features using these simple operators is incredibly hard to accomplish and even harder to develop the kernels to do so. Instead, to extract high level features and classify images according to some highly non-linear function, a convolutional neural network (CNN) is used. The advantage of a CNN is that the elements that make up a filter are learned as weights for a neural network. A CNN is typically split up into two primary sections; the feature extraction layers (using the learned kernels) and the classification or regression layers (typically a densely connected deep neural network).

The feature extraction layers consist of a series of convolutional layers that are occasionally interrupted with a pooling layer. Each of these convolutional layers is defined by core aspects including the number of filters to be learned, size of filters, stride length, and image padding convention [9]. The capacity for the different types of feature information that is attainable is directly due to this type of network's ability to learn many different filters on a single level of image feature resolution. In forward propagation, each convolutional layer produces many output images for a single input image; these are then fed into the next layer to repeat the process. As the processed images pass from layer to layer, the types of features that are extracted become more complex. Pooling layers are used to downsample resulting images from convolutional layers to reduce cost to computation and to gain spatial invariance for the features. From an algorithmic standpoint, at a certain point where the input image has already been processed through convolutional layers, some high-level and complex features may have been learned; therefore it is no longer necessary to know *exactly* where these features are in the image, but simply just some basic spatial or combinational knowledge of itself in relation to other high-level features is adequate [9].

Complex features derived from the convolutional layers are then fed into classification layers that is a simply densely connected deep neural network. Typically, since this network is initially untrained, there is no reason to still use any spatio-relational knowledge between the features as additional information to the dense layers. These features should be on a high enough level where this is less important, and additionally these relational features will be

implicitly learned by the network since the model structure stays fixed through training and testing. As with any other deep neural network, typical optimization techniques such as dropout, random weight initialization, and early stopping can be implemented.

IV.i. The Model

For my model I proposed a version of the previously described CNN with 3 phases of feature extraction followed by a 3 layer densely connected network for classification. In total the network holds 11,911,979 trainable parameters. Figure 2 shows a high level view of the network.

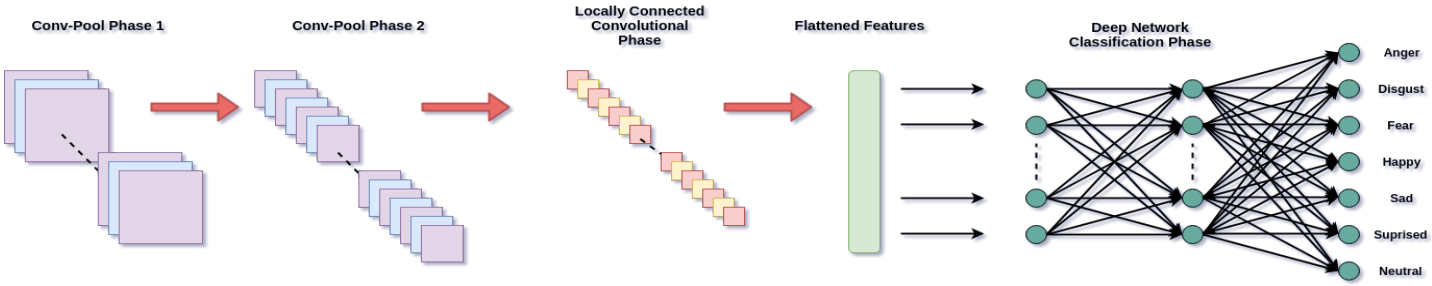

Figure 2: High Level CNN Model

IV.i.i Feature Extraction Phases

Each Convolutional and Pooling Phase consists of three consecutive and uniform convolutional layers followed by a maximum pooling kernel layer. Each of these convolutional layers use 5x5 pixel filters, ReLU activation functions, active image padding, and 40 output filters when in the first phase and 60 output filters when in the second phase.

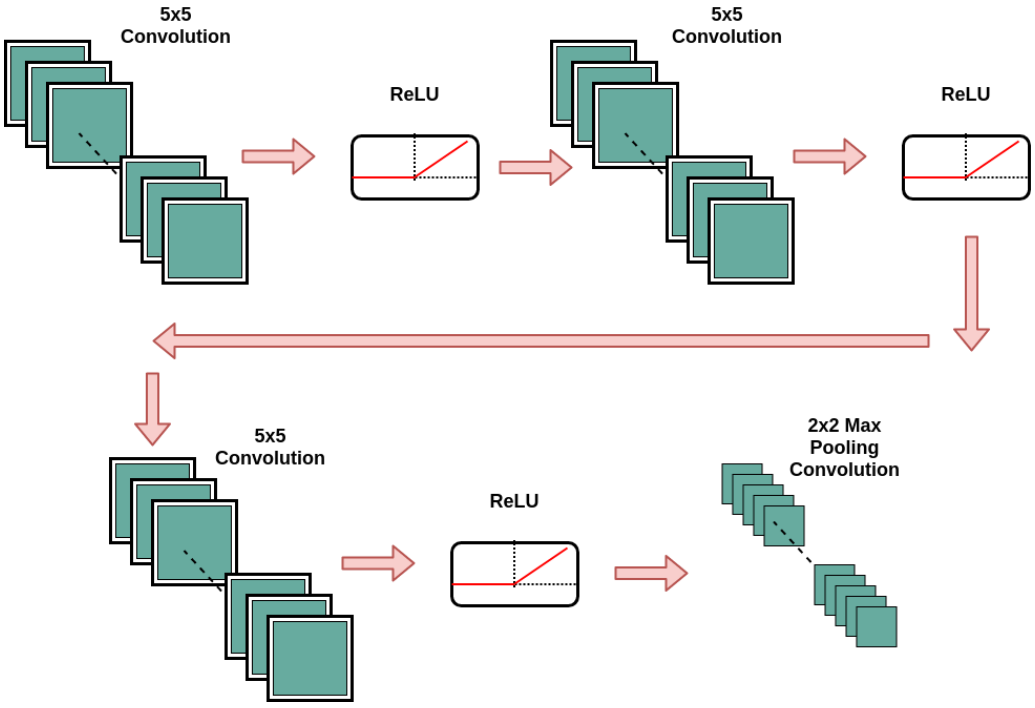

Figure 3: Convolutional and Pooling Phase Network Model

Due to the small dimensions of the original image (recall that the input image is only 48x48 pixels), it was necessary to implement an active padding operation for each convolutional layer such that the size of the images remained constant from input to output. Without this padding, the size of the image shrinks quickly from layer to layer and therefore would not allow for the number of layers that were needed to capture enough data. The 2x2 pixel max pooling convolution layer cuts the size of the images in half for both dimensions to consolidate feature data and reduce computation time for both forward and backward propagation algorithms.

To demonstrate the feature gathering hierarchy that is implicit with this type of model, visualization techniques similar to those proposed in *Visualizing and Understanding Convolutional Networks* by Matthew Zeiler and Rob Fergus [10] were employed. Instead of using custom algorithms to reconstruct the image space from feature data gathered through forward propagation (involving unpooling, rectification of normalized pixels, and reconstruction), intermediate-layer output functions provided by the Keras library [13] were used. Analyzing the output images of a layer can be very beneficial for understanding the function and effectiveness of the layer. The figure below displays all of the output images after a single image is fed through the first convolutional layer of the network.

Figure 4: All 40 Output images from the first layer of the network.

(A) *(B)* *(C)*

Figure 5: (A) Original Image, (B-C) Highlighted output images from the first layer.

As expected the first layer of the network was performing very simple convolutions on the image, highlighting low level aspects. The filter that produced the image in 5.B singled out the background from much of the foreground and occluding objects, giving a clear representation of where the face is in the image. A previously popular technique for accomplishing this task was to reduce the dimension of feature vectors down to features that correspond to axes exhibiting high variance (Principal Component Analysis). When applied to images of faces the results of this process are usually called eigenfaces. However, the convolutional network approach accomplishes this task far more efficiently. The filter producing the image in 5.C highlighted major edges between the background and the target object. Additionally, it was noted that the

number of output images that exhibited little and unintuitive information gain, my original assumption was that the weights for these kernels were functioning inefficiently. But when experimenting with introducing more kernels for the same layer, it became apparent that the amount of information gained had not significantly increased, and in my worst case some of these kernels would rarely activate as if there had been a dropout layer active. In short, the increasing rate of capacity for producing some non-linear function for each convolutional layer would at some point stop fall off for an increase in kernels per layer. This was very useful in determining the size and structure of my network layers to meet feature gathering needs as well as staying within my computational limits. It is also worth noting that this may not be the case for my specific image that was input to this first layer; it is most likely that some kernels did not find features they were trained for in this image.

Figure 6 shows all 40 of the 5x5 convolutional kernels from this first layer after training is complete. This data is somewhat hard to interpret, and is less useful to look at than the output images; but it is useful to know that during training, each of these kernels were trained independently and perform a unique convolution producing unique features. Additionally, viewing the kernel weights after the first layer quickly becomes intractable to show in 2 dimensions.

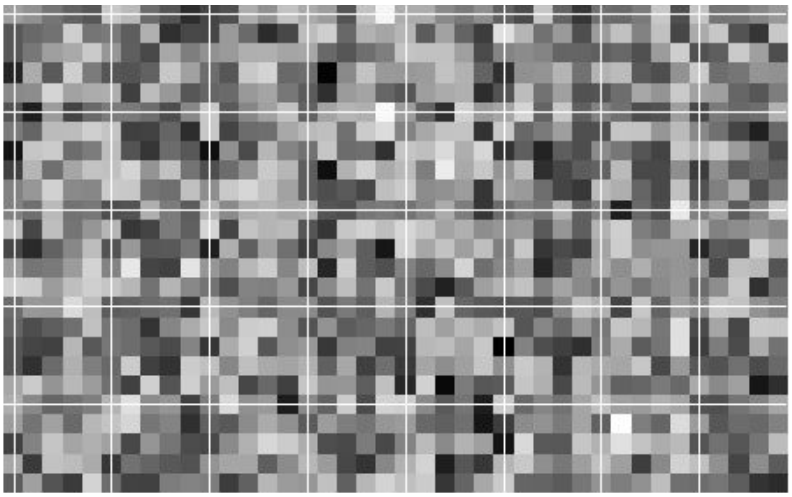

Figure 6: 5x5 Kernels from the first convolutional layer.

After the pooling layer, the output images have not only reduced in size, but have also consolidated feature data as discussed earlier.

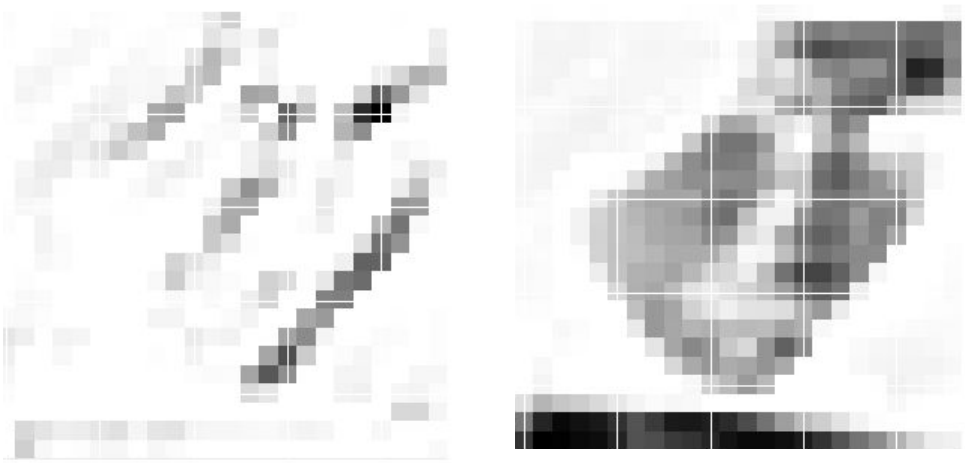

Figure 7: Sample output images from the first max pooling layer.

And eventually, after enough layers of convolution have taken place the data is too compact to discern any useful information that can impact the decisions regarding network structure.

Figure 8: All 60 Output images from the second pooling layer of the network.

The final phase of feature extraction consists of two locally-connected convolutional layers using a 3x3 pixel kernel size each yielding 100 output images. The locally-connected convolutional (LCC) layers are similar to the previously used convolutional layers but differ in one key aspect; for every pixel in one of the output images the kernel that produced it in convolution with the input is unique. Meaning that the kernels for each output image now longer share weights. The data entering the first LCC layer is in the form of 60 channels of a 12x12 pixel image. As seen in Figure 8, the information encoded in a single data point is very complex.

Low level features of the original image are no longer present, and therefore further refinement and downsampling of the data through traditional convolutions do not necessarily produce additional, valuable information. However, what can be inferred from this data are unique local combinational relationships between 2 or more of the high level features present. To accomplish this I used the LCC layers to train a unique kernel that analyzed the relationship between a given pixel feature and its surrounding pixel features. Unfortunately, the Keras machine learning library while having LCC layer functionality does not yet allow for any active padding to be present at the same time; therefore not all the local pixel combinational relationships possible for a given kernel could be analyzed.

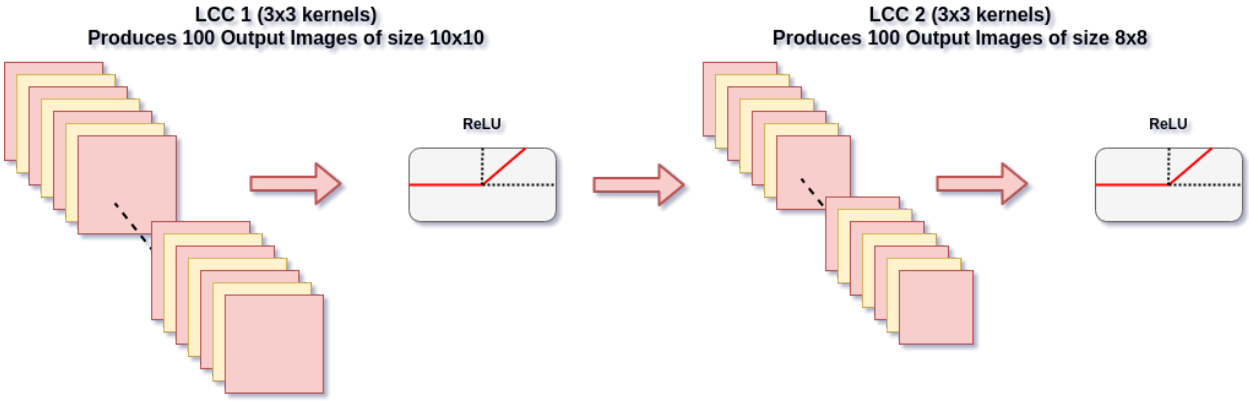

Figure 9: Locally-Connected Convolutional Phase Network Model

This phase yields 100 8x8 pixel images that is then flattened into 6400 data points and fed into the densely connected network for classification.

IV.i.ii Classification Phase

The classification phase is made up of two densely connected layers with 64 nodes each followed by a 7 node output layer. Each hidden layer uses the ReLU activation function while the output layer uses a softmax activation function to produce a valid probability distribution across the 7 emotions. To avoid overfitting to the training data, each node in the hidden layers also utilize a 50% dropout rate during network training. That is, for each backpropagation algorithm executing during training, every weight feeding into these hidden layers has a 50% chance to be neglected when updating before the next training iteration. The full network model including information on output shape for each layer can be found in the appendix.

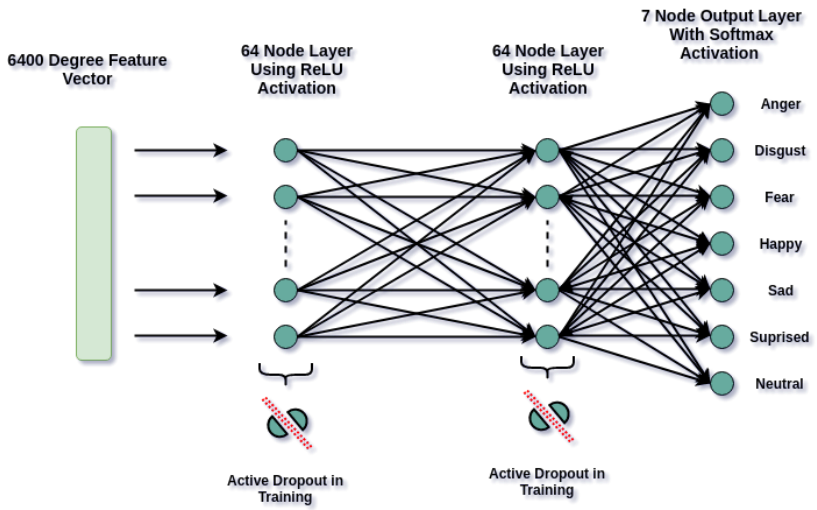

Figure 10: Densely Connected Neural Network in the Classification Phase.

V. Training and Evaluation

During training of my model I defined a loss function using the categorical cross entropy across all 7 classes. This allowed for the network to gauge the accuracy of its predictions during training taking into account the probability distribution across all output nodes. Minimizing this loss metric with respect to each of the 11,911,979 weights in the network during backpropagation was accomplished with the Adam stochastic optimization algorithm. Adam is a modern optimization algorithm that uses adaptive moment estimation during stochastic gradient descent. Adam maintains a unique learning rate for each weighting parameter that are varied based on the first and second moments of the gradients [14]. This optimization method is very useful in computer vision problems such as this where the gradient is sparse.

This project was executed on a Nvidia Tesla K80 GPU that was utilized by the Tensorflow backend of Keras [13]. However, with the large amount of training parameters that were present, I decided to split the training into multiple epochs each using a random batch of samples. My model was trained over 50 epochs each using a mini-batch of 200 training samples. Between training epochs, a secondary subset of random samples was used to validate the current accuracy of the network; the evolution of this validation accuracy is monitored and the network ceased training if this metric stopped improving. Training the network under this hardware environment takes about 6 minutes. After training is complete, I used a full set of 3,589 testing images that had not been used during training to gauge the true accuracy of this model. my model was able to achieve 53.259% accuracy in testing and

69.481% accuracy on training data, yielding an overfit ratio of 0.7665 (which should optimally be 0.5). While this testing accuracy might seem low, it is far better than random guessing (14.3% accuracy). Additionally, in most useful settings that this system can be implemented, a more valid estimate could be easily gained through taking a sequence of images of the same subject and averaging the resulting probability distributions for all emotions.

We also found that it was a common occurrence for the model to confuse similar looking facial expressions such as sad vs. angry and fear vs. surprise. Often when this confusion occurred, the prediction with the next highest probability was the correct response; the table below describes this relationship.

Criteria	Predicted Sad Correct is Angry	Predicted Angry Correct is Sad	Predicted Fear Correct is Surprise	Predicted Surprise Correct is Fear
% 2nd Guess is Correct	76.7%	77.7%	68.3%	94.4%

To put this in perspective, over all incorrect guesses, the percentage of time that the second guess was correct was 36.98%.

VI. Conclusion

This project used a facial expression recognition dataset from Kaggle.com to train a multi-phase convolutional neural network to be able to recognize 7 different emotions that were present in the image dataset. This convolutional neural network utilized a hierarchical feature extraction method using a sequence convolutional and pooling neural network layers to gather high level and complex features about each input image. Locally-connected convolutional layers were used to gain combinational statistics between high-level features. The resulting feature data was fed into a densely connected network to produce a probability distribution for the 7 emotions. In addition to accuracy metrics, the structure of the model was tuned by examining the intermediate output images to examine the types of features that the learned kernels in the convolutional layers were filtering out. My model yielded 53% testing accuracy over 3,589 images.

References

1. P. Ekman and W.V. Friesen, Facial Action Coding System (FACS): *Manual. Palo Alto: Consulting Psychologists Press*, 1978.
2. M. Pantic and L. Rothkrantz, "Automatic analysis of facial expressions: the state of the art," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 12, pp. 1424-1445, Dec 2000.
3. Y. Yacoob and L. Davis, "Recognizing human facial expressions from long image sequences using optical flow," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 6, pp. 636-642, Jun 1996.
4. M. Rosenblum, Y. Yacoob and L. Davis, "Human expression recognition from motion using a radial basis function network architecture," in *IEEE Transactions on Neural Networks*, vol. 7, no. 5, pp. 1121-1138, Sep 1996.
5. K. Zhang, Y. Huang, Y. Du and L. Wang, "Facial Expression Recognition Based on Deep Evolutional Spatial-Temporal Networks," in *IEEE Transactions on Image Processing*, vol. 26, no. 9, pp. 4193-4203, Sept. 2017.
6. P. Lucey, J. Cohn, T. Kanade, J. Saragih, Z. Ambadar, I. Matthews, "The extended Cohn-Kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression", *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, pp. 94-101, Jun. 2010.
7. M. Valstar, M. Pantic, "Induced disgust happiness and surprise: An addition to the mmi facial expression database", *Proc. Int. Conf. Lang. Resour. Eval. Workshop Emotion*, pp. 65-70, May 2010.
8. “Challenges in Representation Learning: Facial Expression Recognition Challenge.” https://www.kaggle.com/c/Challenges-in-Representation-Learning-Facial-Expression-Recognition-Challenge, 12 Apr. 2013. fer2013.csv
9. A. Deshpande “A Beginners Guide To Understanding Convolutional Neural Networks.” *A Beginners Guide To Understanding Convolutional Neural Networks – Adit Deshpande – CS Undergrad at UCLA* (19), 20 July 2016, adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/.
10. M. Zeiler and R. Fergus. “Visualizing and Understanding Convolutional Networks.” *Computer Vision – ECCV 2014 Lecture Notes in Computer Science,* 12 Nov. 2013, pp. 818–833., doi:10.1007/978-3-319-10590-1_53.
11. W. McKinney. “Data Structures for Statistical Computing in Python”, *Proceedings of the 9th Python in Science Conference*, 51-56, 2010
12. T. Oliphant. “A guide to NumPy”, *USA: Trelgol Publishing*, 2006.
13. F. Chollet, “Keras.” Https://Github.com/Fchollet/Keras, 2.1.3, GitHub, www.keras.io.
14. D. Kingman and J. Lei Ba. “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION .” International Conference on Learning Representations, 9 May 2015, pp. 1–15.

Appendix

1) Full Network Model Structure

Layer Type	Output Shape (BatchSize, ...)	Parameters Produced
(2D Convolution)	(None, 48, 48, 40)	1040
(2D Convolution)	(None, 48, 48, 40)	40040
(2D Convolution)	(None, 48, 48, 40)	40040
(Max Pooling)	(None, 24, 24, 40)	0
(2D Convolution)	(None, 24, 24, 60)	60060
(2D Convolution)	(None, 24, 24, 60)	90060
(2D Convolution)	(None, 24, 24, 60)	90060
(Max Pooling)	(None, 12, 12, 60)	0
(Locally Connected 2D Conv)	(None, 10, 10, 100)	5410000
(Locally Connected 2D Conv)	(None, 8, 8, 100)	5766400
(Flatten)	(None, 6400)	0
(Dense)	(None, 64)	409664
(Dropout)	(None, 64)	0
(Dense)	(None, 64)	4160
(Dropout)	(None, 64)	0
dense_6 (Dense)	(None, 7)	455