California Polytechnic State University

CPE 428 - Computer Vision

Technical Report

Facial Emotion Recognition Using Deep Convolutional Neural Networks

By Lemar Popal and Archit Bose

June 12, 2020

Contents

1 Introduction

The ability to recognize facial expressions from images enables many applications in human-computer interaction such as human-like robotics, driver fatigue surveillance, and medical treatment. Training a computer to detect a person's facial emotions is an interesting problem because it cannot easily be solved using a set of predetermined rules. Instead, we must train a neural network to determine the rules that classify which emotion a subject in the image is feeling. It's also a difficult problem because faces are complex and dynamic, and thus the difference between emotions such as "happy" and "surprised" are quite subtle. There also does not exist a large database of training images and classifying images real-time is difficult because emotions in general are not very exaggerated.

1.1 Aims and Objectives

In this project we evaluate different types of Convolutional Neural Networks (CNNs) on their ability to accurately classify 7 emotions: Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral. We try three different datasets, FER2013, Expression-in-the-Wild (ExpW), and a custom combination of the two.

We also propose a real-world application of the trained network for online education. During the COVID-19 pandemic, education has largely moved online. Some educators have had difficulty creating online course content that is engaging and effective for student learning. By detecting student's attentiveness and emotions during a recorded lecture, educators can identify which parts of their video are most effective or most confusing for students.

1.2 Real World Application

A real world application for this would be to check if a person is distracted or not. In online classes and online work meetings, it is very difficult to stay focused and in order for good communication with the presenter to occur, it is important for the speaker to know when

their audience is distracted or not. It is an interesting problem to be able to detect when a person is distracted online because it would require an eye tracking mechanism to see and analyze when a view is looking at the screen.

Figure 1: *A graphic of a Zoom meetings*

2 Background

2.1 Academic Papers

Over the past several years there has been active research in the field of facial emotion recognition. A paper by Li and Deng (Li & Deng, 2020) gives a useful survey of what has been done so far in this research area. It also describes the three main steps that are common in facial emotion recognition, pre-processing, deep feature learning, and deep feature classification, and summarizes the widely used algorithms at each step and the existing state-of-the-art best practices in the field. This paper helped orient us in which direction we should begin so we could achieve good results.

One of the most well-known emotion recognition competitions was the ICML 2013 Challenges in Representation Learning: Facial Expression Recognition Challenge which included

the FER2013 dataset. There were many entries, with the best model achieving about 71% accuracy (Tang, 2013). Tang used a convolutional neural network and replaced the softmax activation with a linear support vector machine and showed that they were able to achieve significant gains in accuracy on the FER2013 dataset. Following this competition, there have been many attempts to get better results using the same dataset, with varying methods and test accuracy results of up to 75.2% (Li & Deng, 2020).

Another paper (Duncan & Shine, 2016) attempted to detect images in real time using both clean, laboratory images ($> 90\%$ accuracy) and non-clean images but with much less successful results. They found that any deviation from laboratory conditions caused the accuracy to fall significantly, but the authors suggested that heavier pre-processing of the images (such as adjusting brightness to the same level on all images) and image data augmentation may have improved test accuracy.

Another paper (Pramerdorfer & Kampel, 2016) we referenced throughout the project achieved an accuracy of about 75% on the FER2013 dataset using an ensemble of modern deep CNNs without requiring auxiliary training data.

2.2 Blogs and GitHub Repositories

A blog post by Priya Dwivedi gives some useful code snippets on how to implement facial detection and gives a link to a GitHub repository with code to train a neural network from scratch (Dwivedi, 2016). We used this code as a starting point because it provided a simple Keras model that used image augmentation and achieved about 58% result accuracy with little work.

We also used blog posts from Machine Learning Mastery (Brownlee, 2020) and PyImageSearch (Rosebrock, 2019) that provided code on how to perform transfer learning with pre-trained models.

One github repository by Antoine Lamé gives some useful code as to how to use the DLib library to implement a pupil tracker from a webcam and would display the pupil's location

of where the pupil is on a frame and perform some light analysis as to where the person was looking.

3 Data and Methods

3.1 Data

We used three datasets primarily: FER2013, Expression-in-the-Wild (ExpW), and a custom combination of the two. Both original datasets consist of images collected "in the wild", in contrast to a laboratory setting where emotions are exaggerated and lighting, pose, and subjects are controlled.

Images from FER2013 were collected by the Google image search API and resized to 48x48 pixels after manually labelling and centering the face in the image. It contains 7 distinct emotions: Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral. The training set consisted of 28,709 images and the validation and test sets both contained 3,589 images for a total of 35,887 images.

Images from ExpW were collected in a similar fashion to FER2013, except images are colored contain multiple faces per image. It also contained the same set of 7 emotions. After cropping out faces and resizing each image to 64x64, the training set consisted of 71,176 images and the validation and test sets both contained 9,077 images for a total of 89,331 images.

To combine the two datasets into my own custom one, I resized the FER2013 images to 64x64 and converted the ExpW images to greyscale and combined the training, validation, and test sets respectively to get a total of 124,671 images.

3.2 Face Detector

Because our datasets consisted of images of cropped faces, we needed a way to detect faces within in an image before sending it through the neural network for classification. We used

a face detector called MTCNN, which stands for Multi-task Cascaded Convolutional Neural Networks for Face Detection. The architecture underneath is a bit complicated, but the model is easy to use as we can just pass it a frame and it returns a bounding box around the faces in the image.

3.3 Gaze Tracking

Gaze Tracking works by using Python's DLib library which uses facial landmarks to mark areas of an image where certain components of a face is detected. It first localizes the face in an image and the detects the key facial structures on the face ROI. An example of how the facial landmarks look like are shown in Figure 1. We used this library because it had pre-built eye and pupil classes that could be used and we changed the horizontal ranges of the eye tracker in order to get detection when someone is looking in a certain direction. We also added conditions to detect when a person was not looking at the camera to detect if the person was distracted from looking at the screen. If we had to make further changes to the library. It would be good idea to add a calibration feature where the computer detects a person's eye and stores the configurations to make it easier to track the eye of a specific people because each eye is different and tracks differently.

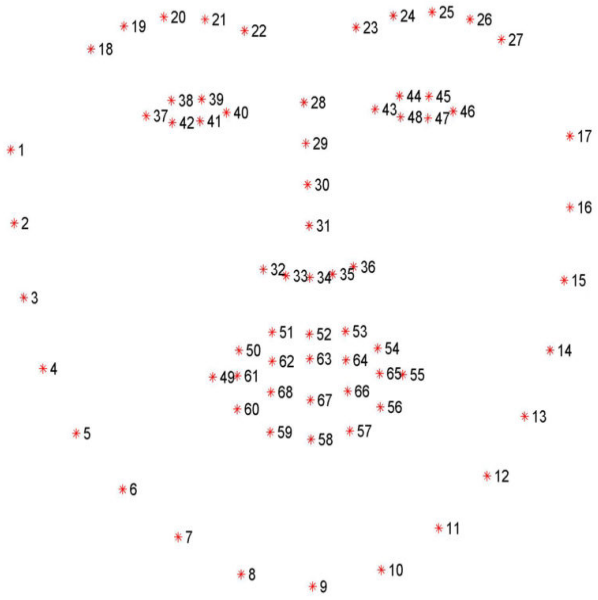


Figure 2: *An image of facial landmarks*

3.4 Convolution Neural Networks

We used primarily used three types of Convolutional Neural Networks (CNNs) that were available in Keras Applications:

- VGG16 trained from scratch (~15 trainable parameters)
- ResNet152V2 trained from scratch (~60 million trainable parameters)
- ResNet152V2 pre-trained on ImageNet and fine-tuned for emotion classification (~60 million trainable parameters)

We chose these three CNN architectures because they have proven to be effective at classification, with VGG achieving a top-5 accuracy (meaning the correct class was within the top-5 classes the model predicted) of 90.1% on the ImageNet dataset (contains thousands of classes). The ResNet architecture improves in VGG16 by achieving a 94.2% top-5 accuracy, but because of the depth of the network, takes much longer to train. We wanted to experiment to see if a deeper network with more trainable parameters would lead to better results.

We also wanted to experiment to see if starting with a pre-trained network on ImageNet and fine-tuning for emotion classification would lead to better results.

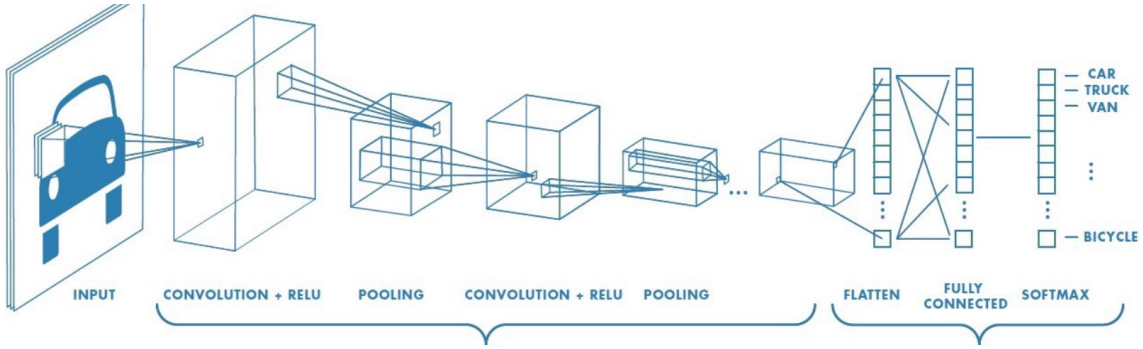


Figure 3: *An image of a Convolutional Neural Network*

4 Evaluation

During training the network was evaluated on a separate validation set and we monitored two statistics: accuracy and f1-score. Model weights were saved after each epoch where validation loss decreased.

Accuracy was a useful statistic to measure how well our network was doing over the entire validation set. However, due to some classes having more training images than others, the network could be good at predicting certain classes while getting others completely wrong. The f1-score is a better metric because it penalizes models that consistently predict certain classes.

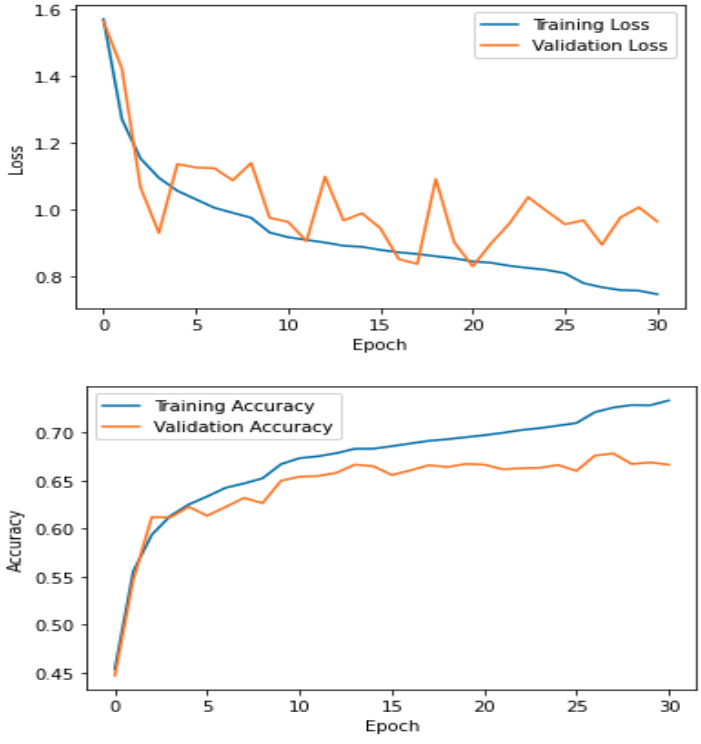
For example, consider a dataset with 2 classes. Class 1 is 90% of the training data, Class 2 is the remaining 10%. If the model always predicts Class 1, the accuracy would be 90%. At first glance the model is doing great, but it will never predict Class 2. Accordingly, the f1-score would be 0. Similar to accuracy, f1-scores closer to 1 are better.

5 Results

5.1 Experiment 1 - VGG16 trained from scratch

We started the first experiment by training a VGG16 network from randomly initialized weights. We used a small learning rate of 0.0001 and batch size of 256. We used the Image Data Generator in Keras to create additional training data by applying shears, shifts, zooms, and horizontal flips to images. Images were preprocessed using the VGG preprocessing function available in Keras.

After only 2 epochs, the VGG network was at 65% accuracy and had an f-beta score of 0.5347. After 7 epochs, it achieved an accuracy of 70% and an f-beta score of 0.6320. In subsequent epochs the model started over fitting because the training accuracy neared 100% while the validation accuracy stopped increasing.



5.2 Experiment 2 - Added Gaze Tracker

In this experiment, the eye motion tracker was added to detect whether or not a person was distracted. It took longer to figure out how to implement the code so this feature was pushed back to be a part of the second experiment. This prototype of a gaze tracker improved the system by adding a different algorithm for facial detection than the initial experiment. By using the DLib library to initially landmark facial features and facial detection, it ensured that only the relevant frames for emotion detection were being passed into the neural network to be processed which in turn reduced the load on the feature matching for emotion detection. Another iteration that was added was the ability to put recorded videos into the program and detect gaze and emotions throughout the video which allowed us to test the results against known metrics for higher accuracy.

5.3 Experiment 3 - ResNet152V2 trained from scratch

Given the promising results from Experiment 1, we investigated if a deeper network with more trainable parameters would improve accuracy and f1-score. All hyperparameters were kept the same as Experiment 1. Images were preprocessed using the provided preprocessing function for ResNet models.

After 6 epochs, the model was at 65% accuracy and f1-score of 0.6616. Unfortunately, the model did not improve further despite lowering the learning rate using the learning rate scheduler. The model began to overfit and the validation accuracy and f1-score decreased significantly.

5.4 Experiment 4 - ResNet152V2 pre-trained on ImageNet and fine-tuned for emotion classification

Since the model performance in Experiment 3 wasn't as accurate as VGG16, we wanted to rerun Experiment 3 using the pre-trained weights from ImageNet. Instead of training

an entire network from scratch, the first few layers in the model can be used for feature extraction (and their weights will not be changed) and the final few layers can be fine-tuned for emotion classification. The same hyperparameters and image preprocessing were used as Experiment 3.

After 6 epochs, the model only achieved an accuracy of 48% and f1-score of 0.31 before beginning to over fit. Attempts to change the learning rate and other hyperparameters did not yield much better results.

5.5 Final Results

In a Python script, we put together the gaze detector and neural network predictor using the model weights from Experiment 1. The webcam reads in a frame, the face detector crops out the face from the image, the gaze detector finds the pupils and orientation of the eyes, and finally the network predicts the emotion. The location of the pupils, orientation of the eyes, and predicted emotion were shown on the image as text.

Upon evaluating the script overall, we noticed that the code performed the eye tracking very efficiently but it took time for the image to propagate through the network and properly detect an emotion. We could speed up the performance of the emotion detector by using a different type of network called a MobileNet. These networks usually have a lot less parameters with a minimal performance drop. These networks are designed to run on resource constrained devices such as mobile phones. Alternatively, not every frame needs to be passed through the network. If we only predicted 1 frame per second, this would likely be enough for emotion detection.

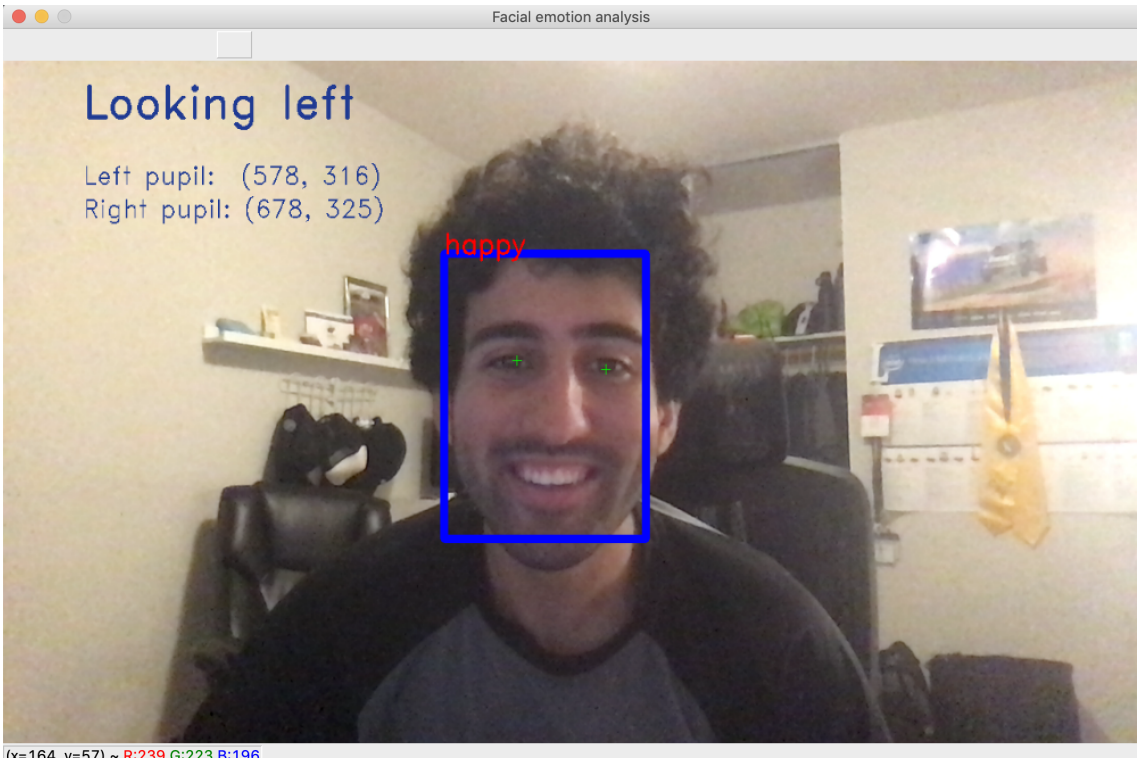


Figure 4: *Emotion Detector Application*

6 Outlook

The expectation of this project was to be able to create a neural network that could be used to detect emotions on a person's face through a camera and create an application that educators could use to evaluate the effectiveness of their recorded lecture videos by measuring student attentiveness and emotions. We were able to successfully develop a prototype of an application that used an eye motion tracker and a neural network in unison to detect facial features and analyze the features in order to detect emotions. We were able to learn how to build and train our own neural network and also learned how to use the python DLib library for facial detection and use these tools to develop our own application.

To continue improving this project, we would like to add features to the application such that after a lecture video it outputs a log file of timestamps with associated student emotions and attentiveness. Then, another application could aggregate those log files and educators

can see the distribution of student emotions throughout the video. Another addition to the project would be to run the application with multiple people being present on the screen or on a zoom call and to be able to track more than one person's emotions on the screen.

Furthermore, we could investigate why model performance decreased using larger networks and with pre-trained weights.

Additionally, we could perform hyperparameter (learning rate, batch size, activation function, dropout, etc.) tuning using a random search method or Bayesian optimization. This would significantly increase training time but likely lead to better results.

We could also look at obtaining larger datasets with more images. For example, AffectNet contains over 1 million images collected in-the-wild. This would give our model more variety and likely better results.

Lastly, another option is to ensemble our network with other emotion recognition models. For example, if we can detect keypoints on a face, such as the location of the mouth, eyebrows, and eyes, we can use that information along with the network to make a more informed decision about the predicted emotion.

References

Brownlee, J. (2020, May 20). Multi-label classification of satellite photos of the amazon rainforest.. Retrieved from https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to

Duncan, D., & Shine, G. (2016). Facial emotion recognition in real time..

Dwivedi, P. (2016, April 3). Face detection, recognition and emotion detection in 8 lines of code!. Retrieved from https://towardsdatascience.com/face-detection-recognition-and-emotion-detection-in-

Li, S., & Deng, W. (2020, March). Deep facial expression recognition: A survey. *IEEE Transactions on Affective Computing*. Retrieved from http://dx.doi.org/10.1109/TAFFC.2020.2981446

Pramerdorfer, C., & Kampel, M. (2016). Facial expression recognition using convolutional neural networks: State of the art.. Retrieved from https://arxiv.org/abs/1612.02903

Rosebrock, A. (2019, June 3). Multi-label classification of satellite photos of the amazon rainforest.. Retrieved from www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/

Tang, Y. (2013, 06). Deep learning using linear support vector machines. Retrieved from https://arxiv.org/abs/1306.0239