Oyu Gantumur and Shakh Saidov

CS 414. Machine Learning

Final Project Executive Report

Recognizing and Identifying Human Emotions in Various Environments

In this final project, we wanted to explore the capabilities of machine learning in the domain of face detection and emotion recognition. We searched and found a dataset in Kaggle that contained images of faces with diverse postures and angles, but mostly looking straight at the camera [1]. Our goal was to train a model using this dataset and then apply our model on handpicked real-life images that contain multiple people in various positions and scenarios. We conducted this experiment to observe how the model would act under a new environment and how accurate the results would be.

The main dataset used in our project consisted of 35000 grayscale images that were grouped into 7 categories of emotions: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. We chose this dataset based on its size, its consistency and completeness. We were tasked to create a self-sustaining notebook and make the project run seamlessly without local downloads, so we decided to upload all of the dataset files to our shared Github repository. Since the extracted csv file was too big to upload at once, we split the dataset into 15 chunks and pushed the files to our remote repository. Then, we scoured the Internet in order to find an example code to start off with. There were many solutions with varying accuracy results, and we decided on the code written by Alpel Temel as it had a relatively high validation accuracy of 58% [2]. The target was to find a way to reconfigure the model in order to increase the accuracy of the model.

The model itself is a sequential model that utilizes convolutional neural networks (CNNs), which is perfect for this project as CNNs are regarded to be very accurate at image recognition and classification. We used 2D convolution layers, where the kernel performs element wise multiplication and summation of results into a single output. For our activation function, we utilized the ReLU function in our hidden layers, as it circumvents the vanishing gradient problem, results in a better performance and converges at a faster rate than other activation functions. We have also discussed in class that the ReLU sometimes performs better than the hyperbolic tangent functions, and we have also found that to be the case in the process of hyperparameter tuning. We added batch normalization in order to stabilize the inputs, which we believe helped the model's output at the end. Also, we added max pooling to remove invariances, making the model detect the presence of features rather than its exact location. The specific change we made was to remove average pooling and only utilize max pooling because we believe the former wasn't extracting important features and was causing the accuracy to be lower. Since we wanted to eventually test the model on images of tilted faces in different angles, we found max pooling to be crucial in the model's accuracy. The output layer is implemented with the softmax function because our project is a multiclass classification problem - given the goal is to label each face with one of the seven emotions - and the softmax is best suited for handling multiple classes. The softmax function computes the probabilities of the image belonging to each of the seven emotion classes, and the sum of all probabilities add up to one. And the class with the highest probability is chosen to label the image. Given the nature of our project being a multiclass classification problem, we decided that softmax was the most appropriate and efficient activation function to be used for our output layer.

After we trained our model, we wanted to apply the model in recognizing emotions in real-life images with multiple people in various scenarios. Before performing any face detection, we first converted the image to grayscale to reduce its noise and improve computational efficiency as it is easier to identify faces in grayscale than in color. For face detection, we used the pre-trained *haarcascade* classifier built in OpenCV. This algorithm uses edge or line detection features proposed by Viola and Jones. This model is based on the sum of pixels for various facial features - such as eyes are darker than the nose and cheeks regions, and eyes are darker than the bridge of the nose - and is trained with a lot of positive images with faces and negative images with no face [3]. We chose to use the OpenCV haarcascade classifier because it is fast and requires less computing power, making it suitable for our limited time and resource [4]. It can also detect faces in a wide range of orientations and scales, making them versatile for a variety of situations and scenarios. However, we have later learned that the downside of the haarcascade model is that it produces many false positives. In order to reduce the number of false positive, we decided to make the filtering process a bit more strict: we increased the minNeighbors to three so false positives from the background are removed; and we modified the scaleFactor to 1.3 so the larger faces, or the main faces in the frame, can be detected faster and more accurately [5]. These changes run the risk of missing some faces, but we think eliminating false positives is more important than missing a few true positives in terms of efficiency and accuracy.

After the model was made optimal we searched for images that contained groups of people in various positions. We wanted our model to be able to detect faces in irregular angles and their expressions correctly. After finding the development dataset, which comprised 10 such images, we ran the model. The model's ability to label facial expressions was on par with the test result, which maintained around 62%. Looking at the results we did notice that there is a case of overfitting happening, which could be remedied by adding more data and reducing the number of features. The main issue that we faced in the process of conducting this research was the facial recognition system by Computer Vision library which would sometimes miss human facial features or falsely detect background areas as human faces, as we explained in detail above. This project could benefit from further research and development where we either find the optimal way of facial recognition or create a model that is fed with datasets filled with images of human faces in various angles and positions.

References

1. Research Prediction Competition. "Challenges in Representation Learning: Facial Expression Recognition Challenge." *Kaggle*, 12 Apr. 2013, www.kaggle.com/competitions/challenges-in-representation-learning-facial-expression-recognition-challenge/data
2. Temel, Alper. "Fer2013 (with Keras)." *Kaggle*, 27 Sept. 2020, www.kaggle.com/code/alpertemel/fer2013-with-keras
3. "Face Detection Using Haar Cascades." *OpenCV*, https://docs.opencv.org/3.4/d2/d99/tutorial_js_face_detection.html
4. Jaiswal, Abhishek. "Object Detection Using Haar Cascade: Opencv." *Analytics Vidhya*, 27 Sept. 2022, www.analyticsvidhya.com/blog/2022/04/object-detection-using-haar-cascade-opencv/#:~:text=Haar%20cascade%20is%20an%20algorithm,%2C%20buildings%2C%20fruits%2C%20etc
5. Selvaraj, Natassha. "Face Detection with Python Using Opencv Tutorial." *DataCamp*, 17 Apr. 2023, www.datacamp.com/tutorial/face-detection-python-opencv