

EmoSense

Understanding Group Emotions through Individual Facial Analysis

Implementation Process

The basic idea of my model is to detect faces, crop them out, perhaps transform them or filter them out in some way, run facial expression recognition on each face individually, then combine the facial expressions into a general mood.

To implement my model, I worked in reverse. Since the main focus is emotion recognition, I started by focusing on the facial expression recognition models. I settled on HSEmotion after finding that it performed much better than DeepFace and FER on the FER-2013 and FER+ datasets.

Second, I tried to look at super-resolution models, which hold the promise of improving low-resolution images. As a litmus test, I scaled down an image of a woman wearing sunglasses and ran it through the models. Torchsr, which is a general super-resolution model, tested terribly, so I rejected it and looked for face-specific super-resolution models. I found two that looked promising and provided models on GitHub: SCGAN and ELSFace. I got SCGAN working, but while it performed better than torchsr, it still did pretty badly. ELSFace had some bugs in the testing code when I ran it, which I didn't have time to correct, so I did not move forward with it. If I had more time, I would test ELSFace, as it looked very promising in its paper. SCGAN may also prove workable if I had time to test on more faces.

Third, I tested face images vs. resolution to see what effect it had on facial expression recognition. Since the result was pretty strong on the three faces I tested, I decided to make a pixel threshold (of the largest dimension of the face image) for filtering out faces a hyperparameter. At the same time, I added one for bounding box confidence levels.

Fifth, I tested face detection models. The YOLOv8-face model looked promising, which I tested nano, medium, and large versions of, but I also tested Facenet-pytorch (MTCNN algorithm), RetinaFace, and Batch-face (RetinaFace algorithm). I first tested the speed of each model, which immediately led to the ditching of the raw RetinaFace algorithm since scalability is a requirement. For the rest of the models, I compared their speed, false positive rate, false negative rate, and true positive rate (in terms of individual faces detected) using my validation set. I explain the reasoning for choosing the large YOLO model in section 5.5 of the Jupyter notebook.

Sixth, I wrote a class to bring everything together. It takes wrapper functions for the models, filters, transformers, and expression voting methods as input to the constructor, and basically works as a pipeline. The workflow is explained in section 6.1 of the Jupyter notebook.

Differentiating Individual Emotions

To differentiate individual emotions, my model goes through a process similar to that described above, but in reverse.

1. YOLOv8-face detects individual faces in the provided image
2. Using the bounding boxes provided by YOLO, my model crops out the faces
3. The model filters out images based on a minimum pixel threshold (of the largest face image dimension) and a minimum bounding box confidence level.
4. The images are transformed, if any are provided (I didn't get any working).
5. HSEmotion (enet_b2_7 model) takes all the cropped face images and detects facial expressions.
6. The model calculates the overall mood using a given aggregator. By default, it sums the expression probabilities across all the faces in the image and takes the max.

Integrating Existing Technologies

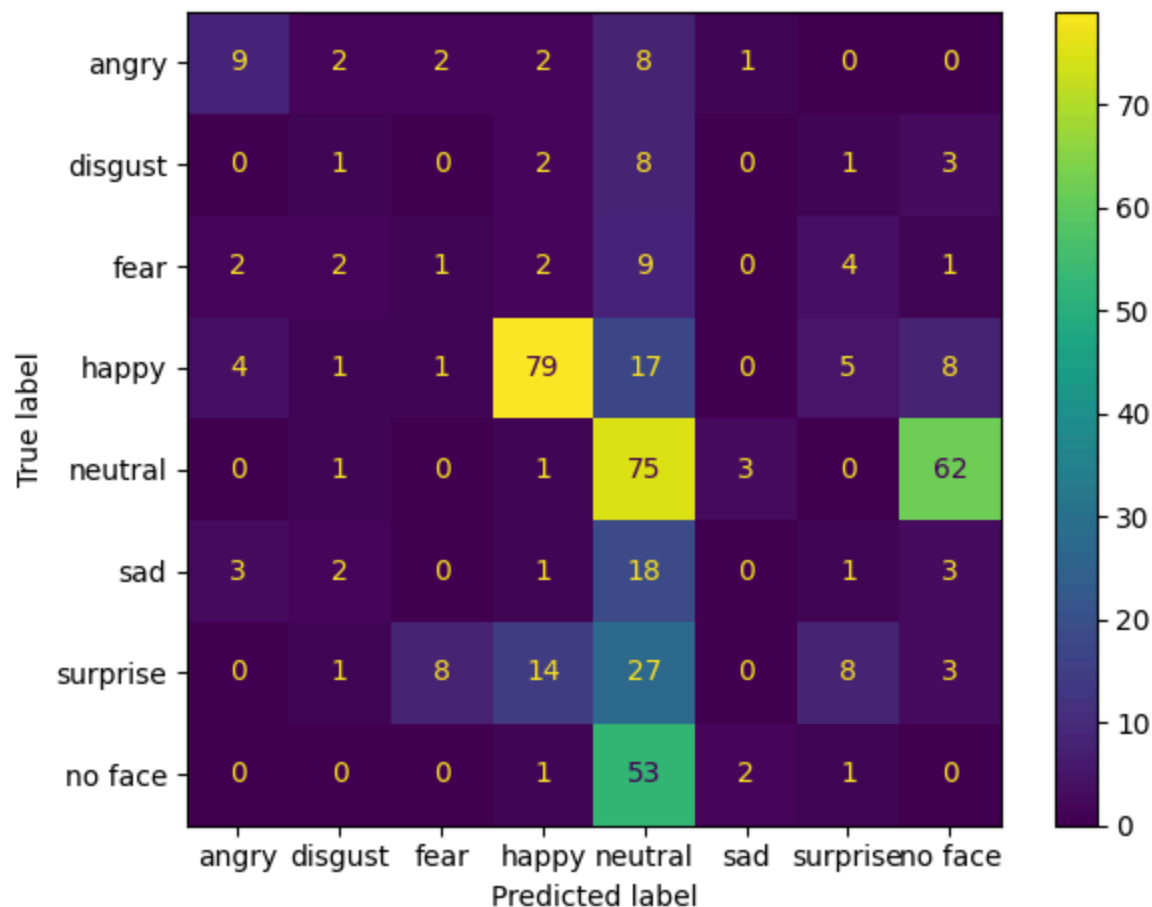
The two primary existing technologies I use are YOLOv8-face and HSEmotion. YOLO is available in the Ultralytics Python package, but I had to download the face detection models from GitHub (<https://github.com/akanametov/yolo-face>), but it's very straightforward to load. HSEmotion is also available as a Python package, but its install isn't perfect. I had to specify the installation of `timm==0.9.7` to get it to work. However, once I specified which model to use, it downloaded it automatically. Both packages support CUDA, meaning they use my GPU with the right parameters, greatly speeding up inference.

If I could have gotten it working to a satisfactory level, super-resolution would have been another technology I would have used. However, given that I was leaning towards using SCGAN or ELSFace, their use was a bit more complicated. I had to download them off of GitHub. Then, because they were not designed to work as packages for other python programs, I had to modify the code. In the case of SCGAN, I had to change the way some modules were loaded.

Preliminary Results

My results did not turn out well. It is difficult to say whether the problem was the labeled data, the model, or both.

On my test set, my mean average precision is about 0.31, which is pretty bad. I include a confusion matrix below, which shows a lot of false positives and false negatives:



My average runtime per image on my test set is 0.23s per picture, which is pretty good. This result suggests that it may be possible to use the model for real-time detection. However, since I ran it on a relatively powerful GPU (RTX 4070Ti), it would need to be scaled down to run on a less powerful machine.

Challenges Faced and Solutions

The primary challenge I faced was finding an accurate facial expression recognition (FER) model. The primary model recommended, DeepFace, performed terribly (34% accuracy) when I tested it on the old FER-2013 test set and only slightly better (37% accuracy) on the re-labeled FER+ test set. I also tried the other common library, FER, with the MTCNN classifier (Haar Cascade simply didn't work) and only achieved 39% accuracy on FER-2013 and 45% accuracy on FER+.

I therefore spent hours searching for better FER models. The one I landed on is HSEmotion, found at github.com/av-savchenko/face-emotion-recognition (note that it requires timm==0.9.7, despite what the documentation says). It does have a package on PyPI, called hsemotion. Note that there are several models available, most with eight emotion classes (contempt being the eighth). I only used seven when labeling my pictures (angry, disgust, fear, happy, neutral, and sad), as I labeled my data expecting to use the DeepFace library, so I use the enet_b2_7 model,

achieving 57% accuracy on the FER-2013 test set 64% accuracy on the FER+ test set; far better than DeepFace and FER. I tried to test the enet_b0_7 model, but it didn't run.

I also faced a challenge with super-resolution, as explained above, and did not manage to get it working to a satisfactory level.

Finally, calculating the mean average precision turned out to be a headache. I spent hours thinking of how to mold the results into a usable structure, which luckily is similar to the structure needed for the confusion matrix. It's rather complex, and I describe it in section 6.7.1.

In the future, super-resolution would be my main focus, testing ELSFace in particular. Also, if we want to be able to run on smaller machines, I would test HSEmotion with a smaller model. I would also look into removing Python overhead, as there are lots of data conversions and manipulations happening in the model that may not be efficient.