Emotional Face Recognition
Using a Two Stage Model Architecture

Andrew Hinson
Georgia Institute of Technology
ahinson7@gatech.edu

Christopher Austin
Georgia Institute of Technology
caustin48@gatech.edu

Abstract

This paper explores the concept of identifying a person's emotional state based on their facial expression. To accomplish this, we used a two model approach so we could finely tune each model's purpose. We first utilized a pre-trained face detection model to locate an individual's face from an image. We then fed this extracted face data into a second custom model to classify the person's emotional state. After completing our experiments, our results show we can successfully locate an individual's face in an image and correctly classify their emotional state using this two model architecture.

1. Introduction/Background/Motivation

1.1. Emotional Classification

The problem we were interested in solving was, could it be possible to correctly label an individual's emotional state given a still image. This problem first had to be broken down into two discrete steps: (1) use a pre-trained face detection model to extract and individual's face from an image; (2) given the data from the first model, feed it into our custom model for classifying emotional states. Our main objective was to take a publicly available data source with images of faces in different scenarios and be able to place a bounding box around the face in the image and then correctly classify the emotion based on the truth label, which was predetermined in the data source.

1.2. Commonly Used Approaches

Based on our research, we found a mixture of architectures for solving this problem. Some architectures used a similar approach by using two separate models for detecting a face in an image and one for classifying the emotional state [3]. Some architectures used a single model for this to accomplish this task [5]. Given our research, we found having a two stage deep neural network approach provided a more robust solution. As you will see in some of our experiment results below, we found that solutions not utilizing two deep neural network models were more error-prone.

1.3. Possible Applications

The successful implementation of this architecture could help further research in the area of emotional classification and potentially be applied in multiple domains. For example, one of the domains we could see this being applied in is airport security. We envision a system to detect people's emotional state while in an airport and help security identify individuals who may pose a risk. Another potential domain could be a smart phone application which detects a user's emotional state to help identify individuals who may want to harm themselves. By using the two deep neural network approach, the solution could be robust enough to be implemented in an endless number of applications.

1.4. Data

For the face detection model we used 500 images (355 MB) from the publicly available WIDER FACE dataset found here. Further details on the dataset can be found in the corresponding research paper [4]. The images we used for our experiments were a subset of the WIDER FACE dataset used for validation. Since our goal was to only focus on a single individual's emotional state, the data was pruned to remove images containing more then one person. The data also contained instances of individuals in a wide variety of backgrounds and situations, ages, sexes, and races. All of our selected images were in RGB format and of various image sizes. The images also contained bounding box data that was used as a truth label to gather metrics on.

The emotional dataset was another publicly available dataset found on Kaggle. There were approximately 5500 images (56 MB). All of the images were 224x224 in size and in gray scale. The images were pre-processed before the input layer by scaling each image to 128x128 to save memory and all pixels were normalized. The data was pre-categorized into 8 distinct classes (separated into individual directories): anger, contempt, disgust, fear, happiness, neutrality, sadness, and surprise. The images contained real

human expressions and varied in how the category was displayed (i.e. a female displaying anger is visibly different from a male displaying anger). The individuals shown in the images also ranged in age from a young baby to an older adult, difference races, and different sexes.

2. Approach

2.1. Face Detection Solution

For the pre-trained face detector we chose two possible solutions: a Multi-task Cascaded Convolutional Neural Network(MTCNN) neural network and a Haar Cascade detector. The Haar Cascade model is a classical algorithm that uses boosting with weak learners in order to detect lines and edges of faces. The MTCNN is a deep neural network that uses convolution in order to find the bounding box for each face in an image.

The version of the Haar Cascade detector we used was from a pre-trained model that was loaded in using opencv in python. This model can be found here. The MTCNN model is a pytorch model that can be found on github. This MTCNN uses a bounding box regression branch and a classification branch to find faces in images.

2.2. Emotional Classification Solution

The next part of the problem was to have an emotional classification model take in the cropped image from the face detector and classify the emotional state into one of the eight possible emotional labels. The reason we chose this approach was because we realized early on you could not classify a face without knowing where the face was on the image. We wanted the network to focus on learning the facial features and not the image as a whole. Another advantage is the classifier can take in less data so the model can be smaller and take less resources.

We developed two models that took inspirations from other common state of the art models that were trained on the Kaggle dataset. These models were not pre-existing models but were built from the ground up using some of the techniques from VGG and ResNet.

2.2.1 VGG

Our version of the VGG model took inspiration from the original version talked about in the research paper on large scale image recognition [2]. The model was created in Pytorch. We wanted to increase the number of filters as the network got deeper and then flatten the feature maps at the end in order to classify through a linear layer.

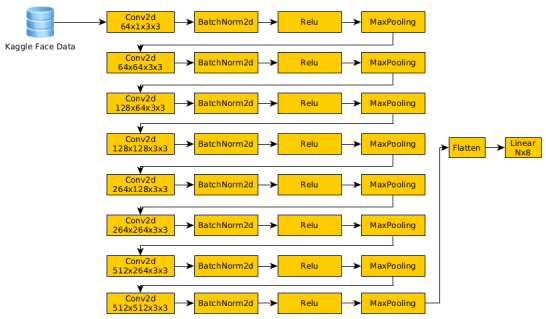

Figure 1. Custom VGG Architecture

The model starts by taking the input images that have been normalized and resized to (128x128). The key to our model was to create many feature maps and extract the most relevant information as we travel down the network. Our kernel sizes grew from 64 to 512 over the course of the network. Max Pooling was used to extract the largest pixel values. This helped give feature maps with the most important information in that filter. Our model was different in that it was smaller in size and contained less linear layers than the original model. The reason for these decisions was due to GPU memory constraints and we found minimal to no gains in adding more linear layers.

2.2.2 ResNet

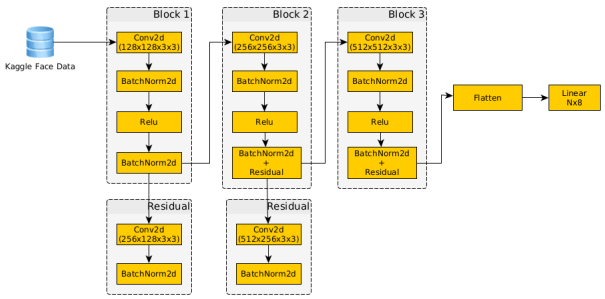

Figure 2. Custom ResNet Architecture

The residual blocks of this architecture help to reduce the vanishing gradients. We created a more compact version in order to preserve GPU memory. We also found this compact version to perform well enough to avoid making a larger model. This model was built from the ground up using Pytorch.

ResNet uses residual blocks in order to help the model learn the features. These residual blocks get added back into certain layers along the network. This helps to preserve the gradients and alleviate the vanishing gradient problem. These techniques would allow for more accurate results. Given the architecture of ResNet and VGG we wanted to investigate how each one would perform on our dataset.

2.2.3 Loss Function and Optimizer

Both models are trying to classify an object into 8 classes therefore Categorical Cross-Entropy was used as the loss function. For training the best optimizer used was Adam. The Adam optimizer combines many of the best properties from other optimizer functions. It includes RMSProp and momentum. It stores the learning rate and the weighted average of the momentum.

2.3. Anticipated Problems

One of the biggest challenges was finding a labeled dataset for the emotional classification. Many datasets did not have quality images or were not labeled. The Kaggle dataset fulfilled our needs but had a shortcoming of only 5500 images and many of the labeled images could have belonged to multiple categories (i.e. in one instance an anger looked like a surprise face).

Another anticipated problem was not knowing if our pre-trained face detection model would perform well enough to suit our needs. We tried to mitigate this problem by running a small subset of the data through the detector to see if it would label the bounding boxes.

2.4. Problems Encountered

The emotional dataset out of the box was not enough to train a good classifier and caused our model to overfit. Due to this small dataset, data augmentation was introduced to create a larger dataset that would allow for better model generalization.

We assumed the Haar Cascade model would not perform as well as a deep learning model; however, it failed to identify a large majority of the 500 faces from the WIDER dataset. Because of this problem our emotional classifier model could not perform on the output of the Haar Cascade model.

3. Experiments and Results

3.1. Face Detection

We tested both the Haar Cascade and MTCNN models on the WIDER dataset to see which model gave us more accurate bounding boxes. The metric used to determine the accuracy of the bounding boxes was IoU (Intersection of Union).

$$IoU = \frac{AreaOfOverlap}{AreaOfUnion}$$

The more the bounding boxes overlapped with the truth box the more accurate the model's prediction was.

We calculated three types of Intersection of Union (IoU) thresholds using the WIDER dataset on both the Haar Cascade and MTCNN models. The three IoU thresholds used were: IoU^{50}, IoU^{75}, and IoU^{85}. The numbers in the IoU values are the thresholds for showing the model found the correct bounding box (i.e. the model must have an intersection with the truth bounding box by at least 50, 75, or 85 percent). From these metrics, we then computed the accuracy of the model.

3.2. Haar Cascade Detector

Recall from above we chose to use a pre-trained face detection model to locate and extract an individual's face from a given image. For this evaluation, we looked at the accuracy of the model using the IoU thresholds (found in Table 1) and visually inspected some of the images that performed poorly. This model performed poorly on all of the images we were using for our experiments (see Figure 3 below for an example).

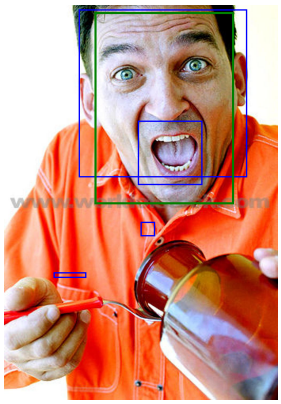

Figure 3. Haar Cascade Face Detector

As seen from the figure above, the blue boxes, which were predictions from the Haar Cascade detector, did not fully encapsulate the truth value, shown in the green box. The Haar Cascade also detected multiple bounding boxes instead of detecting just a single bounding box around the actual face (multiple predicted bounding boxes by the Haar Cascade detector were on the individual's shirt). Based on our research from the article on Haar Cascades [1], our results from our experiment are not unexpected due to the model being unable to learn the detailed feature maps that a deep neural net can extract.

3.3. Multi-task Cascaded Convolutional Network (MTCNN)

Since the Haar Cascade detector did not perform as desired, we chose a Multi-task Cascaded Convolutional Network (MTCNN) as the model for our face detection part of the architecture. As with the Haar Cascade model we

gathered the same IoU threshold metrics to see how it performed.

Table 1. MTCNN vs Haar Cascade

Model	Accuracy at IoU^{50}	Accuracy at IoU^{75}	Accuracy at IoU^{85}
Haar Cascade	0.07303	0.01668	0.00090
MTCNN	0.55555	0.42328	0.19753

As seen from the Table 1 above, the MTCNN outperformed the Haar Cascade detector in each IoU category. These results fortified to us that using a deep neural net model was a superior choice for the face detection portion of our architecture. You can see from Figure 4 below how much more accurate the MTCNN model was with its bounding box versus the Haar Cascade detector shown previously.

Figure 4. MTCNN Face Detection

The MTCNN also only displayed one bounding box over the face versus the Haar Cascade dector from Figure 3 above which detected multiple faces some of which weren't in the facial region of the image.

3.4. Emotional Classifier Model

The emotional classifier model experiments were initially broken down into two distinct experiments: testing a VGG like model and testing a ResNet like model. Both experiments included training and validating on the Kaggle dataset. We collected accuracy, precision, and recall metrics for each model. Each model was then tuned to provide the best results possible.

We built both the VGG and ResNet models and used an 80/20 train/test data split. Table 2 shows our initial parameters used for testing and Table 3 shows our initial results before any hyperparameter tuning.

Table 2. VGG vs ResNet Initial Hyperparameters

Model	Batch Size	Learning Rate	Epochs
VGG	128	0.001	10
ResNet	128	0.001	10

Table 3. VGG vs ResNet

Model	Accuracy	Precision	Recall
VGG	0.42732	0.31573	0.31606
ResNet	0.36221	0.31181	0.29040

After obtaining the initial results, we noticed both models performed very similarly to each other. To further investigate which model would be the one we would select, we performed hyper parameter tuning for each model to determine which one would perform the best. We tuned the batch size, number of epochs, and learning rate for each model. Table 4 shows the final hyperparameters selected for each model and Table 5 shows the best results we were able to obtain.

Table 4. VGG vs ResNet Final Hyperparameters

Model	Batch Size	Learning Rate	Epochs
VGG	64	0.0001	40
ResNet	128	0.001	40

Table 5. VGG vs ResNet Tuned

Model	Accuracy	Precision	Recall
VGG	0.50665	0.46764	0.44636
ResNet	0.46764	0.42523	0.40312

As seen from the above table, after tuning the VGG model performed the best overall. Based on these results, we chose to move forward with the VGG model in our Emotional Classifier part of the architecture.

3.4.1 Data Augmentation

Although VGG performed the best after tuning, overall, both models still performed poorly. We tried several methods in order to improve the overall performance. We first tried using dropout after each activation function in the model. We varied this value from 0.3-0.7 and the models performed much worse than our baseline performance. We also tried to increase the number of training cycles to see if the model could generalize better, but we were still unable to improve the accuracy.After further investigation, we came to the conclusion the limited amount of data we had to train on as most likely the culprit. With the lack of avail-

able labeled data, we decided to use data augmentation in an attempt to improve our metrics.

For the data augmentation, we used 3 types of image augmentation: horizontal flipping, Gaussian blurring, and enhancing contrast. Horizontal flipping was used because it allowed an easy way to double our data while allowing our model to have better spacial localization. Gaussian blurring introduced noise into the data and allowed the model to learn the features of the images that help correctly classify it instead of using parts of the image that were unimportant. Finally, enhancing the contrast helped us improve some of the images were the facial features weren't as well defined (some appeared white-washed). Using these methods, we were able to take our initial dataset from approximately 5500 samples to approximately 22000 samples. Table 6 shows our final metrics obtained from the tuned VGG model using the augmented data (we also included ResNet in the metrics to see how it improved as well).

Table 6. VGG vs ResNet Augmented Data

Model	Accuracy	Precision	Recall
VGG	0.82111	0.82515	0.79217
ResNet	0.81900	0.80759	0.78760

We obtained learning curves to verify our model was generalizing well (we included ResNet as well as a means for comparison) shown in Figure 7 below. From the learning curves, we can see our model did over fit slightly but generalized well enough to produce acceptable results.

3.5. Connecting the Two Models Together

After completing our analysis on the facial detection MTCNN and the VGG model for the emotional classifier, we tied the two models together by feeding the detected face from the MTCNN into the VGG emotional classifier. The data that was fed to the VGG emotional classifier from the MTCNN was cropped to only contain the face that was inside the bounding box. We ran our experiment on a sub sample of the augmented WIDER dataset. We then computed our overall accuracy for the completed two model architecture which included being able to detect the face correctly in the image and classifying the emotion. Our ending accuracy was approximately 80.3%. Figures 5 and 6 show some examples of images that were correctly classified and incorrectly classified.

Figure 5. Correctly Classified Image

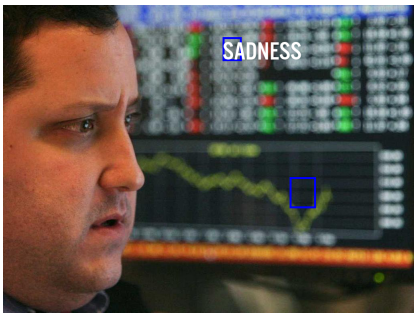

Figure 6. Incorrectly Classified Image

3.6. Successes and Failures

Overall, with an accuracy of 80.3%, our overall two model architecture performed well. However, you can see from Figure 6 above, we still had issues with detecting faces properly and also classifying some of them. Part of the reason Figure 6 wasn't classified correctly was due to the orientation of the individual's face. We believe with more data with different facial orientations, the model would most likely have learned these features and been able to detect/classify the faces better. However, we were able to prove using the two model approach, one for detecting faces and one for classifying emotions, was a suitable solution.

We also believe we could have seen higher success with better labeled data. After investigating some of the images in the Kaggle dataset used to classify the emotional model, we found images labeled "sad" that were obviously "happy." This compromised our model's ability to generalize well given the obvious bad data. In the end, we feel we were successful and given additional good data, we could improve our results even further.

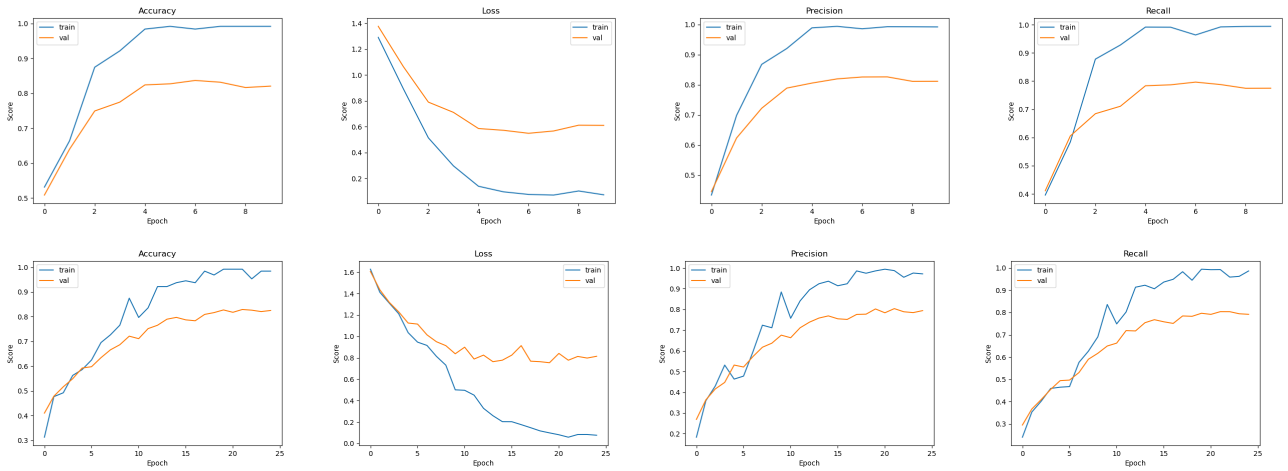

Figure 7. VGG (top row) vs ResNet (bottom row) Performance Using Augmented Data

4. Future Work for Improvement

The MTCNN needs to be fined tuned to better detect faces at unique angles. This could be done via transfer learning this way you can keep the weights and model architecture of the current existing model. As part of expanding our abilities work need to be done in detecting multiple faces and classifying all of the detected faces in the image. A lot of work needs to be done in finding better data for emotional classification. The current dataset has too many errors in the labeling.

Another big area to work on is using this on a video feed. Using a video and classifying people in real time would be a more applicable use case. Using a video feed introduces more complexities with memory and frame rate. Working with videos would require either smaller models in order to run the video at a high and stable frame rate or increase the compute available.

5. Work Division

See Table 7 for a breakdown of the distribution of work.

References

[1] Aditya Mittal. Haar cascades, explained. http://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d, 2020. 3

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. https://arxiv.org/pdf/1409.1556.pdf, 2015. 2

[3] Gaganjot Singh. Facial emotion classification using deep learning. https://medium.com/analytics-vidhya/facial-emotion-classification-using-deep-learning-d08dd02a2d38, 2019. 1

[4] Shuo Yang, Ping Luo, Chen Change Lay, and Xiaoou Tang. Wider face: A face detection benchmark. https://arxiv.org/pdf/1511.06523.pdf, 2015. 1

[5] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng, and Yu Qiao. Joint face detection and alignment using multi-task cascaded convolutional networks. https://arxiv.org/ftp/arxiv/papers/1604/1604.02878.pdf, 2016. 1

Student Name	Contributed Aspects	Details
Christopher Austin (caustin48)	Data collection, Emotional Classification Model, Hyper Parameter Tuning, Testing and Analysis, Report Generation	Searched and located existing datasets for emotional classification. Implemented and tested ResNet for the emotional classifier. Tuned Hyper params for models Analyzed graphs and output metrics from the models. Contributed Equally to generating the final report.
Andrew Hinson (ahinson7)	Implemented Detectors, Setup Training Pipeline, Report Generation, Pre-Processed data, Data Collection	Implemented and tested detectors. Trained VGG model. Build pipeline for easy testing of models. Contributed Equally to the final report. Developed Pre-processing for data. Helped gather data for the detector.

Table 7. Contributions of team members.