Facial Expression Recognition in the Masked COVID-19 Times

Juan Carlos Meave Ojeda
Stanford University
jcmeave@stanford.edu

Abstract

The COVID-19 pandemic has introduced a new challenge to the task of facial expression recognition. With masks hiding a person's mouth, even humans have a difficult time recognizing facial expressions. In this paper, I report the methodology and results of a CNN algorithm that attempts to detect emotion given an image of a masked faced.

1 Introduction

Facial expression recognition is a popular topic in computer vision. With a quick Google search you can find endless machine learning algorithms that can classify a person's facial expression as a specific emotion with impressive accuracy. In today's world, however, COVID-19 has presented a challenge to facial expression recognition as masks hide a crucial piece of information, the mouth. My goal is to implement an algorithm that is able to classify a person's emotion without seeing their mouth.

2 Dataset

For this project, since I could not find a large enough dataset of masked faces, I am using the FER2013 Dataset from Kaggle that has over 35,000 labeled 48x48 grayscale images of facial expressions[1], and turning it into a dataset of masked faces using MaskTheFace[2], a computer vision-based script that masks faces in images, as in the example in Figure 1 below. MaskTheFace attempts to find a face in an image, and if it finds one it adds a mask to it and saves the image to a new directory, but if it does not find one, it simply skips that image without saving it. This works as another form of preprocessing as some pictures in the dataset are hard to visualize as faces, but the new directory only includes well-formed pictures of faces with masks.

[1] Moorsy, Ahmed. "Facial Expression." *Kaggle*, 11 June 2018, https://www.kaggle.com/ahmedmoorsy/facial-expression.

[2] Anwar, Aqeel. "Aqeelanwar/MaskTheFace: Convert Face Dataset to Masked Dataset." *GitHub*, 2020, https://github.com/aqeelanwar/MaskTheFace#bulk-masking-on-datasets.

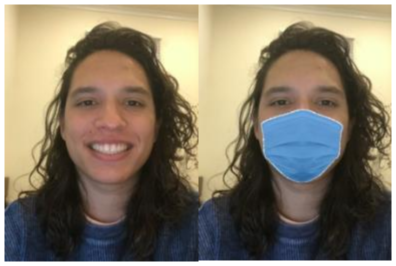


Figure 1. Example of using MaskTheFace to add a mask to an image

3 Methods

As for the algorithm, I am using Ashadullah Shawon's *Facial Expression Detection (CNN)* from Kaggle[3] as my starting point. For the sake of quick iterating, I decided to only use 5000 examples from the dataset, which turned into 3493 after some being discarded by MaskTheFace, and am doing a 90-10 split on training and dev sets. I also made the network shallower, so that the architecture goes Conv2D → Relu → Conv2D → Relu → BatchNorm → Pool → Flatten → Dense → BatchNorm → Relu → Dropout → Dense → Softmax. Using Shawon's original code to train the network, which uses an Adam optimizer and a Categorical Crossentropy Loss function since there are 7 detectable expressions, I saw that the network was able to increase its training accuracy to close to 100% and lower its loss consistently; however, this caused the validation accuracy to decrease and end at around 17% accuracy with the validation loss having consistently increased after each epoch. This meant that the network was overfitting the training set (high variance). To mitigate this, I first tried more regularization to be able to still iterate quickly by incrementally decreasing the batch size from 64 to 32 to 16 to 8 and saw somewhat of an improvement that increased the validation accuracy to a percentage in the low 20s. By introducing more regularization, however, the training set accuracy increased very slowly and by the end of the training, it was in the low 60s, so I increased the learning rate from 0.001 to 0.01. This ended in a training accuracy of about 89% and a validation accuracy of about 29%, with the training loss decreasing steadily and the validation loss slightly increasing, but by less than before.

Now that I had attempted regularization, I wanted to attempt giving the CNN more data. With my starting architecture, however, it would have taken hours to train with the entire dataset. So, to be able to use more data without large increases in time, I decided to use a pretrained layer. I started my model with a pretrained ResNet50 layer, which is the smallest ResNet that came in Keras' applications[4]. I followed this layer with a Dense and Sigmoid Activation to output an array of 7 probabilities, one for each emotion. I trained this model using the same hyperparameters as before, and now I had significantly less variance, but that was with a training accuracy of about 27% and validation accuracy of about 22%. I noticed that the loss was starting very high and rapidly decreasing until it settled at a value that was higher than with the original architecture. For this, I decreased the learning rate from 0.01 back to 0.001, and then to 0.0001. With each decrease, the

[3] Shawon, Ashadullah. "Facial Expression Detection (CNN)." *Kaggle*, 27 Nov. 2019, https://www.kaggle.com/shawon10/facial-expression-detection-cnn/notebook.
[4] Team, Keras. "Keras Documentation: Keras Applications." *Keras*, https://keras.io/api/applications/.

loss started at a lower value and settled at a lower value. This model ended with a training accuracy of 34.3% and a validation accuracy of 33.5%.

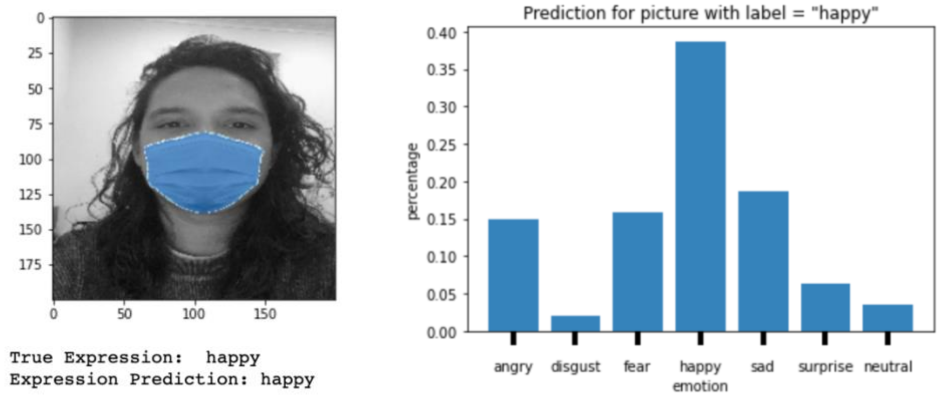
Training Accuracy: 34.3%	Validation Accuracy: 33.5%

Figure 2. Accuracy results of final model

4 Results and Analysis

Believing that this was close to the best accuracy I could get, I wanted to have some metric to see how successful these results were. I tested myself on 50 random pictures from the masked dataset and was able to successfully classify 28 of them (56% accuracy). I was at times, however, able to guess the facial expression from the context of the picture as opposed to features of the person's face. For example, sometimes the picture looked like it was taken casually, in a context where the person should be smiling for the camera and the expression was "happy." Other times, it seemed as if the picture was a mugshot since it was very centered and shot from very straight ahead of the person's face, and I could easily guess "neutral." Then there were the obvious times when someone had their hands up to their cheeks and the expression was "surprise."

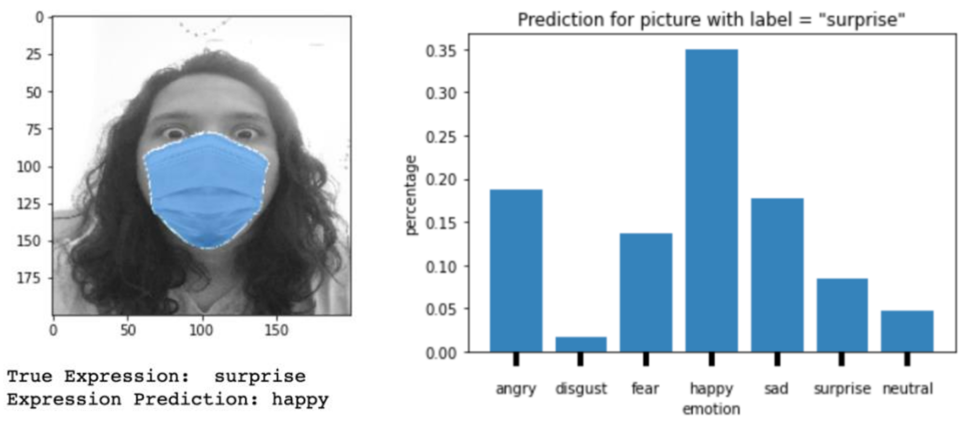
With these observations, I deemed the results of my CNN algorithm sufficient, given that my human performance was not much more impressive. So, I went on to try using my own images and plotting the resulting probabilities. Trying this with the image in Figure 1, the resulting prediction is shown below in Figure 3.



```
True Expression:  happy
Expression Prediction: happy
```

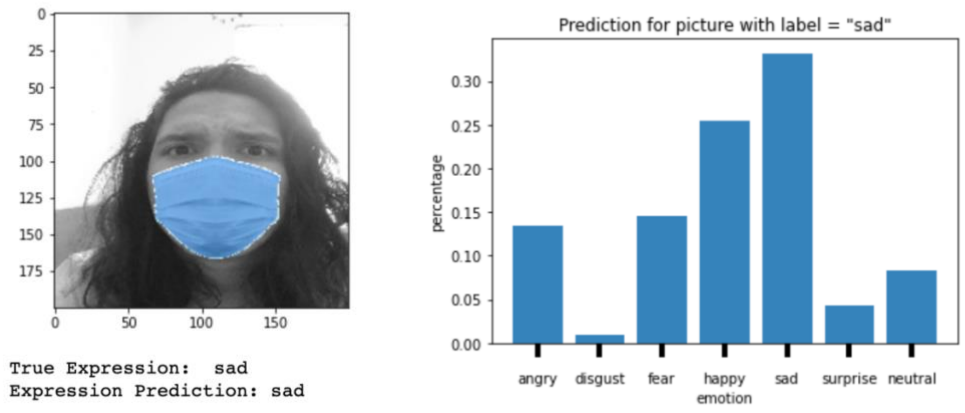
Figure 3. Result of CNN model using my image

One thing I noticed was that oftentimes when the model labeled something incorrectly, it was labelled as "happy," such as in Figure 4, or even when something was labeled correctly, "happy" came as a close second place prediction, such as in Figure 5. At the same time, in most scenarios, "disgust" had a very low probability. I went back and printed counts for each label in the dataset and found that the dataset was very skewed. Of the 25,000 masked images, 7000 were "happy" and only 429 were "disgust" while the rest were in the range of 3000 to 4000. There was not much I could do about the low quantity of "disgust" labels; as for the large quantity of "happy" labels, I decided to test the robustness of my model by reducing the effect of simply always guessing "happy" and getting an accuracy of close to 30%. I did this by setting a limit for how many "happy" examples the model was allowed to use at 3500. This model still was able to achieve a training accuracy of 30.6% and validation accuracy of 29%, showing that the model was robust.



```
True Expression:  surprise
Expression Prediction: happy
```

Figure 4. Incorrect result of CNN model predicting “happy”



```
True Expression:  sad
Expression Prediction: sad
```

Figure 5. Correct result of CNN model with label “happy” coming in second

Moreover, I also tried a quick implementation of a binary classifier for “happy” vs “not happy” using the architecture from the Coursera Keras Tutorial[5] and got results as in Figure 6, with training accuracy of 78.7% and validation accuracy of 63%.

[5] “Keras tutorial – Emotion Detection in Images of Faces” *Coursera*, https://www.coursera.org/learn/convolutional-neural-networks/ungradedLab/mMcLP/keras-tutorial-not-graded.

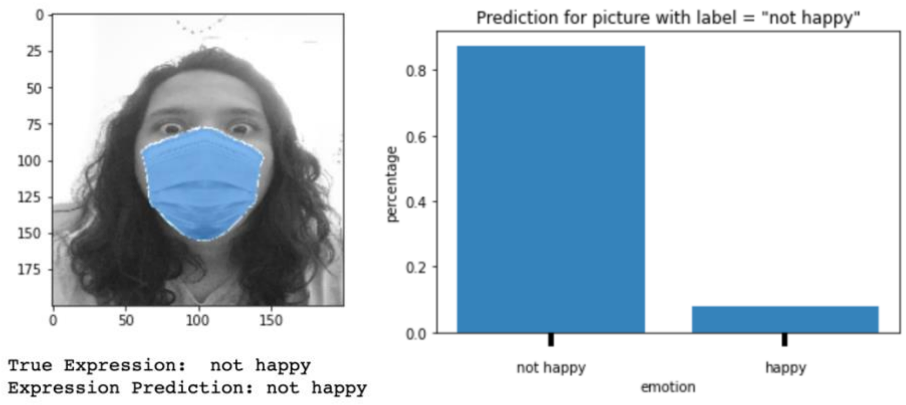


Figure 6. Results of binary classifier model

5 Conclusion and Future Work

After various rounds of hyperparameter tuning and changing the architecture of my CNN, I was able to arrive at a model with which I was satisfied. This model was able to detect masked facial expression with an accuracy of about 30%, when human performance was only at about 50-60%.

If I were to continue working on this project, I would mainly like to focus on improving the dataset. Although FER2013 may work for regular facial expression recognition algorithms, it fell short at times when using it with my model. For one, when implementing the ResNet50 pretrained layer, it required an input with 3 color channels, but all FER2013 images are grayscale with only 1 color channel, so for the model to work I had to keep the mask colorized as in the Figures 3-5, instead of making it grayscale. I could have made the mask grayscale and set the other channels to zero, but I wanted there to be significant color distinction between a person's face and the mask, as is in real life, so I kept it in color, but I would have preferred if the dataset was in color as well. Furthermore, the fact that the images are only 48x48 may be hindering the performance of this model, which in the absence of a persons' full face could benefit from higher resolution images for expression recognition.

In addition to improving the raw dataset, I also would want to improve the preprocessing. I only used one type of mask for this implementation and did not test it on other types of masks. This algorithm could be overfitting this one specific type of mask and have a lower accuracy when detecting facial expressions with other types of masks. So, I would want to make sure than when preprocessing the dataset, I use as many types of masks as possible.

Other than that, when it comes to the actual model, I would want to try both smaller and larger pretrained layers to first increase the training accuracy, then tune the hyperparameters again to bring up the validation accuracy.

References

[1] Anwar, Aqeel. “Aqeelanwar/MaskTheFace: Convert Face Dataset to Masked Dataset.” *GitHub*, 2020, https://github.com/aqeelanwar/MaskTheFace#bulk-masking-on-datasets.

[2] “Keras tutorial – Emotion Detection in Images of Faces” *Coursera*, https://www.coursera.org/learn/convolutional-neural-networks/ungradedLab/mMcLP/keras-tutorial-not-graded.

[3] “Machine Learning in Python.” *Scikit*, https://scikit-learn.org/stable/.

[4] Moorsy, Ahmed. “Facial Expression.” *Kaggle*, 11 June 2018, https://www.kaggle.com/ahmedmoorsy/facial-expression.

[5] Shawon, Ashadullah. “Facial Expression Detection (CNN).” *Kaggle*, 27 Nov. 2019, https://www.kaggle.com/shawon10/facial-expression-detection-cnn/notebook.

[6] Team, Keras. “Keras Documentation: Keras Applications.” *Keras*, https://keras.io/api/applications/.

[7] Team, Keras. “Simple. Flexible. Powerful.” *Keras*, https://keras.io/.

[8] “Tensorflow.” *TensorFlow*, https://www.tensorflow.org/.

[9] “Top 4 Pre-Trained Models for Image Classification: With Python Code.” *Analytics Vidhya*, 23 Dec. 2020, https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/.