Detecting Face Masks - A Ternary Image Classification Problem

Brandon Sams

08Feb2021

Problem

One of the best ways to fight Covid-19 is to simply wear a mask when out in public. This is true because it is a small step that everybody can do. In addition to masking up in the general public, (which ought to be happening anyways), there are locations where masks are required, and also must be work correctly. Common examples include hospitals, but could also include standard retail stores. As of right now, it is responsibility of the employees at that location to detect if a person is wearing a mask, and if they are wearing it correctly.

Luckily, a decent amount of the detection process can, in theory, be handed off to a well-trained machine learning model. Many businesses already have cameras monitoring the interior of said business, so it would be useful if they could be used to help boost public safety, and monitor mask wearing.

However, this problem is not just a matter of mask wearing in general. It should also be noted that while many people do end up wearing a mask, they do so incorrectly. That is, a mask worn correctly must cover the nose and mouth completely, so as to limit the probability of transmission. This changes the problem from a simple binary classifier to one that has 3 classes, with one arguably lying between the two.

Hypothesis

I hypothesize that an image classification model can be trained to detect not only the difference between a masked individual and an unmasked individual, but it can also predict if a person is wearing a mask incorrectly or not.

I also hypothesize that this is able to be accomplished with over 90% accuracy.

Dataset

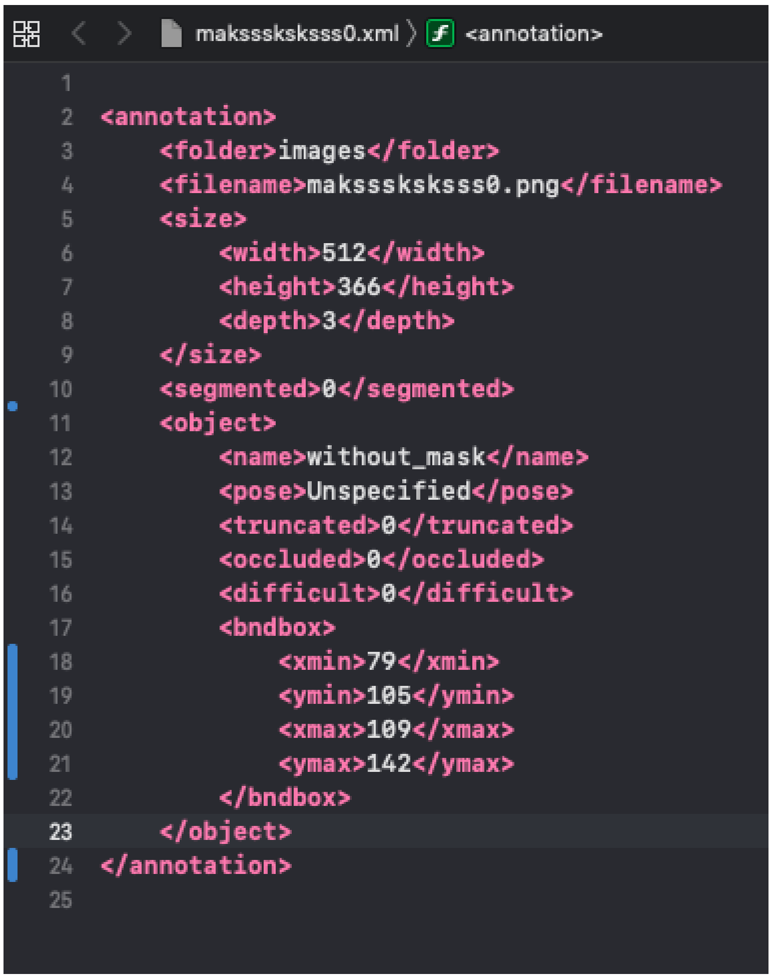
The dataset that will be used for training and testing a model can be found at the following URL:

```
https://www.kaggle.com/andrewmvd/face-mask-detection
```

This dataset contains 853 images, each of which contains images of at least one person’s face. These faces fall into one of three categories:

- mask_weared_incorrect
- with_mask
- without_mask

Each image has an associated annotation file, which is in .xml format. An example is shown here:



```xml

<annotation>
    <folder>images</folder>
    <filename>maksssksksss0.png</filename>
    <size>
        <width>512</width>
        <height>366</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>without_mask</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <bndbox>
            <xmin>79</xmin>
            <ymin>105</ymin>
            <xmax>109</xmax>
            <ymax>142</ymax>
        </bndbox>
    </object>
</annotation>

```

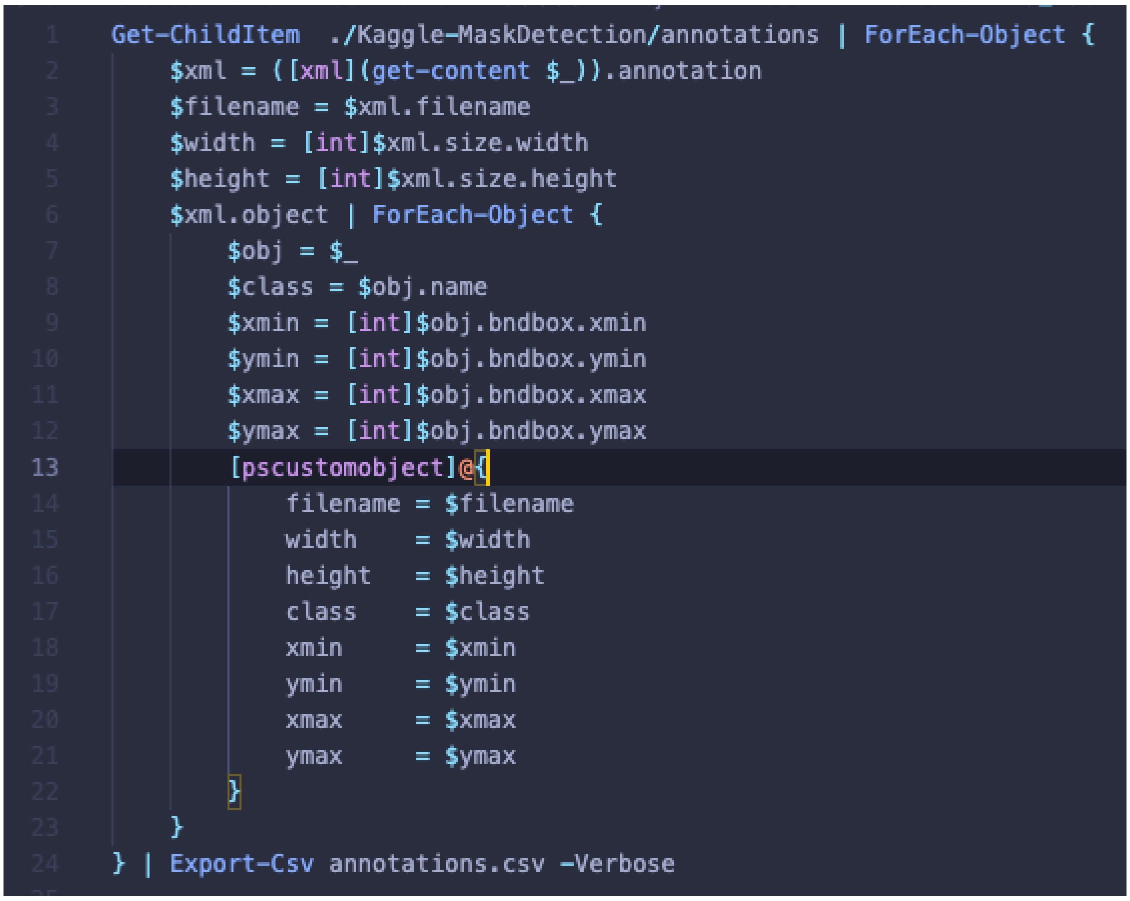
There is a bit of extra information in this dataset that is of little use to us, such as the "segmented" field, or the "pose", "truncated", "occluded", or "difficult" fields. The data that will be useful is:

- filename: which image does this annotation correspond to
- size: dimensions of the source image
- name: classification of the person in the image
- bndbox: tells us what portion of the image contains the face in question.

Given this information, and the images for analysis, the next stage in the data pipeline is to prepare the data for analysis.

Data Preparation

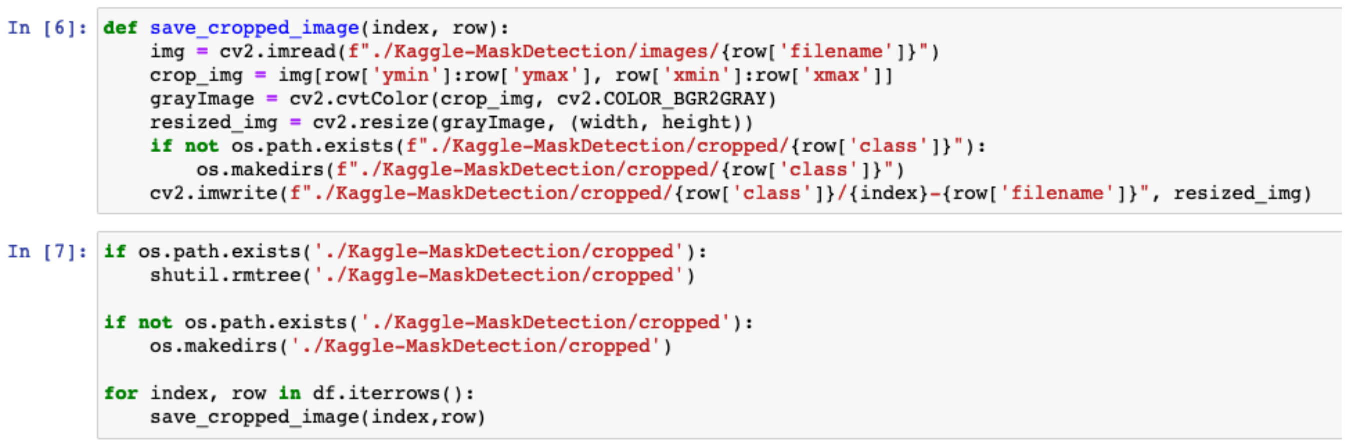
In order to load the annotation data, it was decided that a convenient way to store the data would be a simple .csv file. There are plenty of ways that one may accomplish this task, but for this project, a Powershell script was used. It loops through each annotation file, and within each file, it then loops through each node, creating a new .csv entry with each iteration. It does this by creating a new pscustomobject with specific properties, and sending it down the pipeline.



```
Get-ChildItem  ./Kaggle-MaskDetection/annotations | ForEach-Object {
    $xml = ([xml](get-content $_)).annotation
    $filename = $xml.filename
    $width = [int]$xml.size.width
    $height = [int]$xml.size.height
    $xml.object | ForEach-Object {
        $obj = $_
        $class = $obj.name
        $xmin = [int]$obj.bndbox.xmin
        $ymin = [int]$obj.bndbox.ymin
        $xmax = [int]$obj.bndbox.xmax
        $ymax = [int]$obj.bndbox.ymax
        [pscustomobject]@{
            filename = $filename
            width    = $width
            height   = $height
            class    = $class
            xmin     = $xmin
            ymin     = $ymin
            xmax     = $xmax
            ymax     = $ymax
        }
    }
} | Export-Csv annotations.csv -Verbose
```

After running this step, it was found that the original dataset of 853 images contained 4072 total faces.

Given this csv, the next step of the data cleaning process is to grab the faces from the original images, and save these cropped faces to a separate directory. I decided to use cv2 for this project, as it seems like an excellent python library for image manipulation. I wrote some code that makes a directory if it does not exist already, and saves a cropped and resized version of the original image to that directory.



```
In [6]: def save_cropped_image(index, row):
            img = cv2.imread(f"./Kaggle-MaskDetection/images/{row['filename']}")
            crop_img = img[row['ymin']:row['ymax'], row['xmin']:row['xmax']]
            grayImage = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)
            resized_img = cv2.resize(grayImage, (width, height))
            if not os.path.exists(f"./Kaggle-MaskDetection/cropped/{row['class']}"):
                os.makedirs(f"./Kaggle-MaskDetection/cropped/{row['class']}")
            cv2.imwrite(f"./Kaggle-MaskDetection/cropped/{row['class']}/{index}-{row['filename']}", resized_img)

In [7]: if os.path.exists('./Kaggle-MaskDetection/cropped'):
            shutil.rmtree('./Kaggle-MaskDetection/cropped')

        if not os.path.exists('./Kaggle-MaskDetection/cropped'):
            os.makedirs('./Kaggle-MaskDetection/cropped')

        for index, row in df.iterrows():
            save_cropped_image(index,row)
```

I also went ahead and stripped out the color from the image, as it is likely not necessary for determining the presence of a mask. I could also see it having an impact on the predictions if a person was wearing an uncommonly colored mask.

The images were saved to a local folder, with each class having its own subfolder. This made it quite easy to associate each image with a given class, rather than attempt to associate each image with the csv that was created earlier.

Given this collection of images, I then needed to split it into a training/validation split. I end up using an 80/20 split, respectively.

I then needed to convert each one to a 2-dimensional numpy array, and flatten it into a 1-dimensional vector. This vector will serve as the input for a tensor flow model, so each input vector needs to be the same size.

Model Fitting

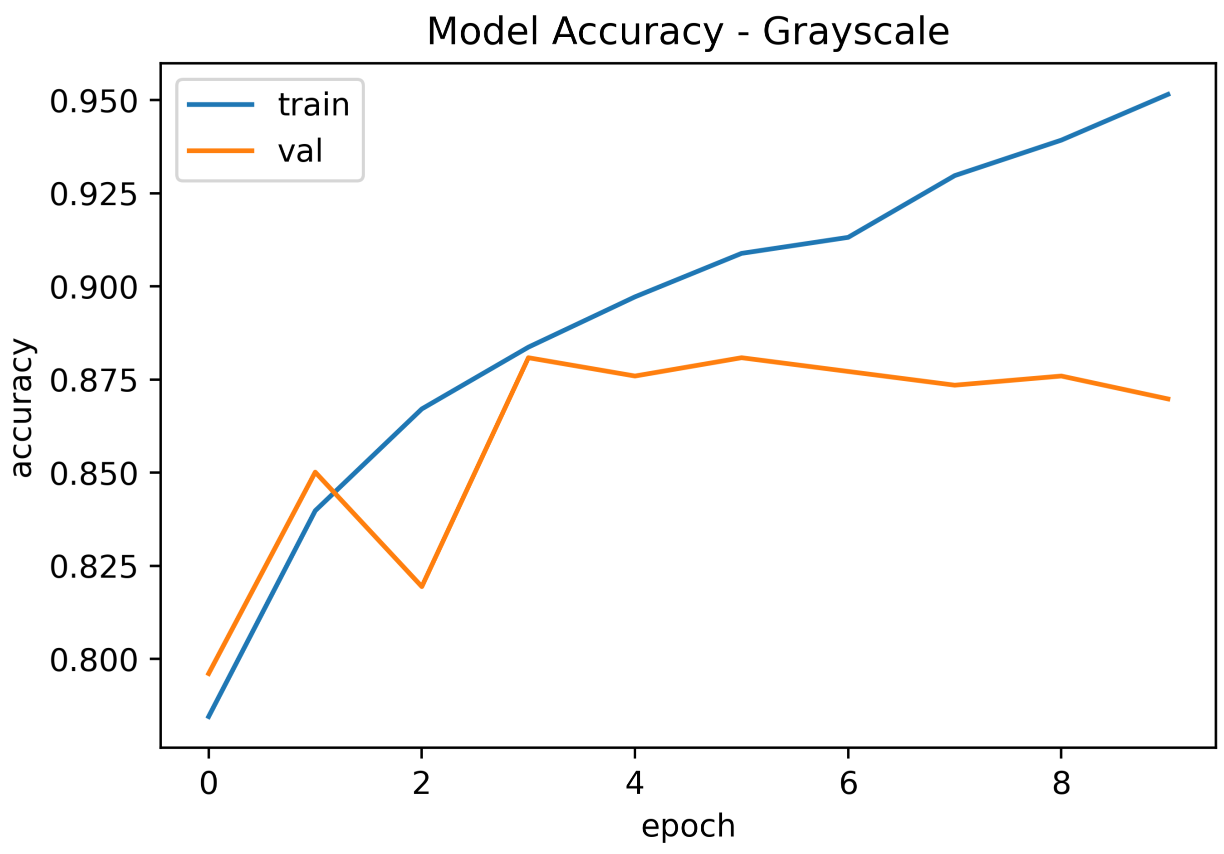
Once the images had been converted into vectors, they also needed to be normalized. The values for each pixel lie between 0 and 255, and they ought to lie between 0 and 1. I decided to do this as the first non-input layer for the model, just to keep the model tidy.

I decided to go with a sequential model, with a collection of 2-dimensional convolutional layers. I also decided to use MaxPooling2D after each layer.

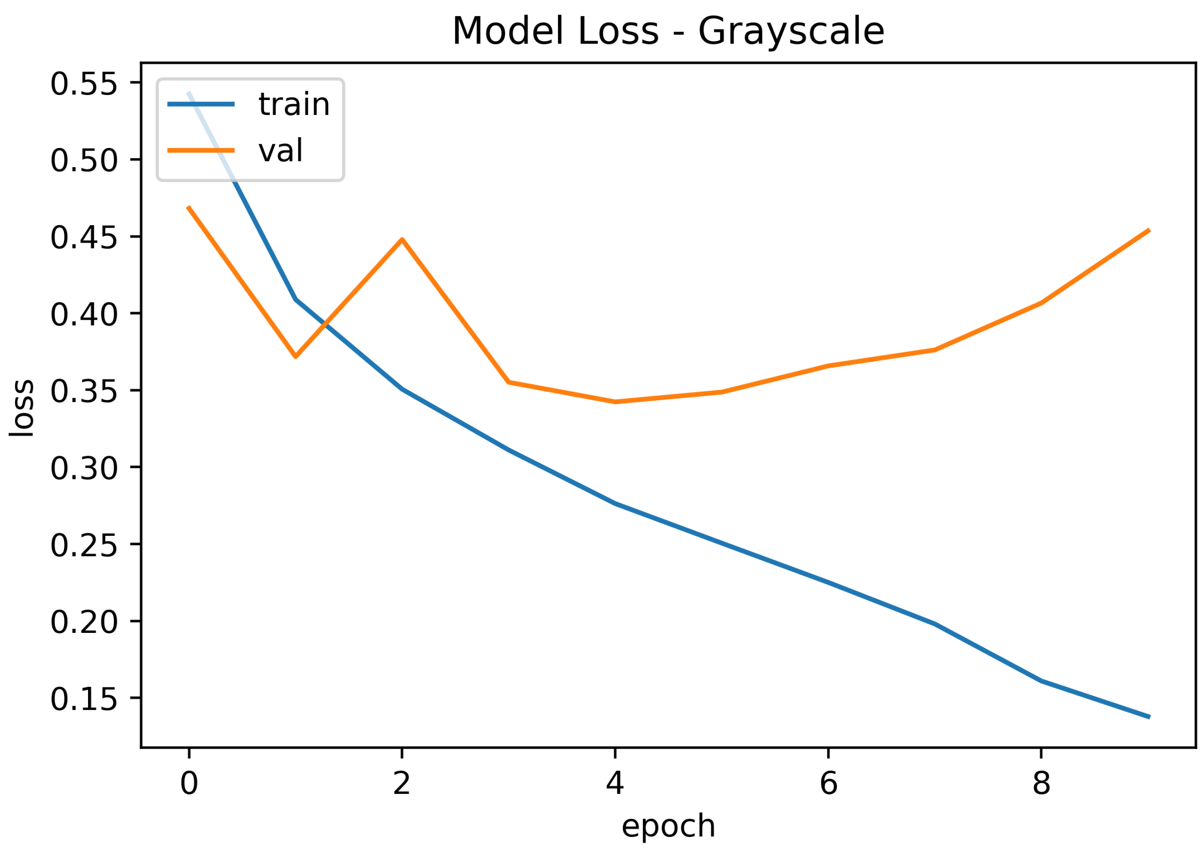
Once this model was compiled, I needed to fit it to the training data I had cleaned and partitioned earlier. I ended up choosing 10 epochs for this process, as the validation accuracy did not appear to be increasing past that point.

Results

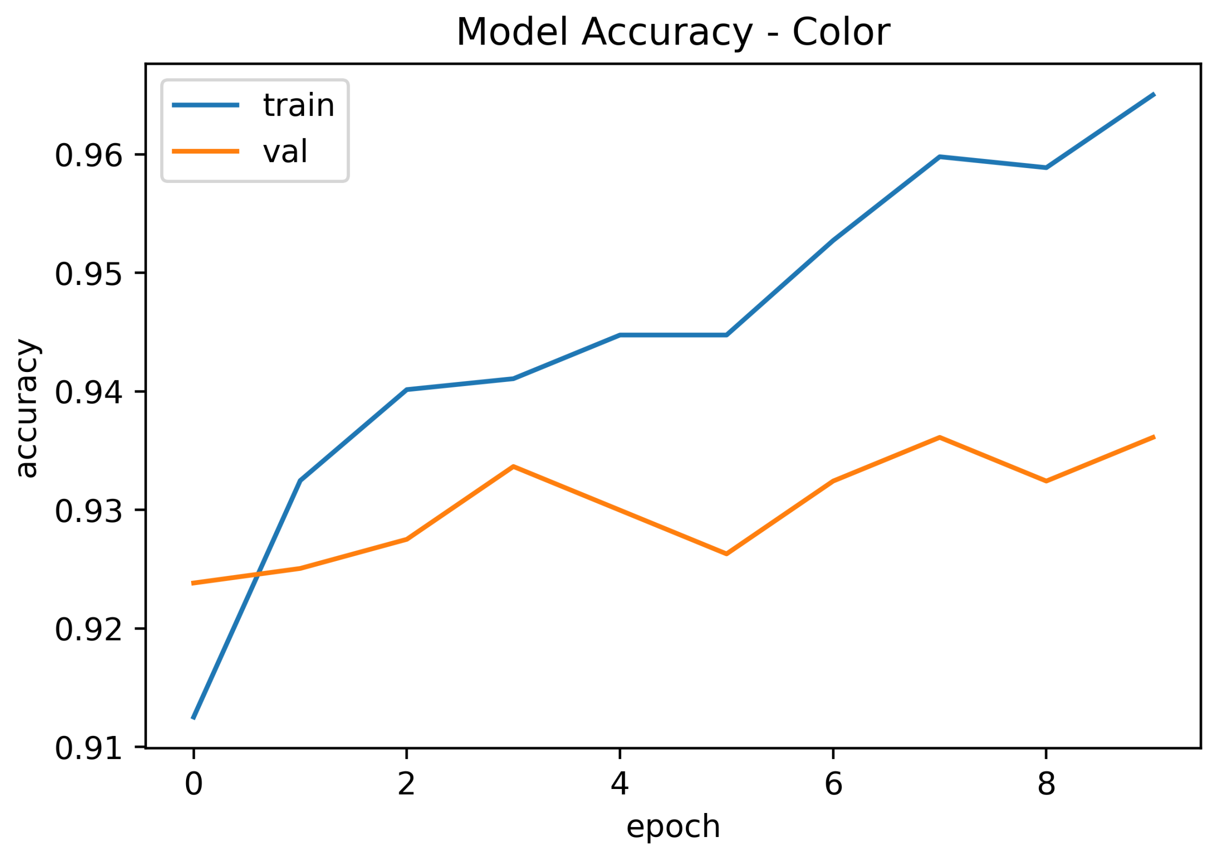
In only 10 epochs, the model was able to tune a model that was 95% accurate on the training data, and 88% accurate on the validation data. I anticipate that running the model for a longer period of time would not have been beneficial, as the validation data's accuracy score was decreasing with additional epochs.

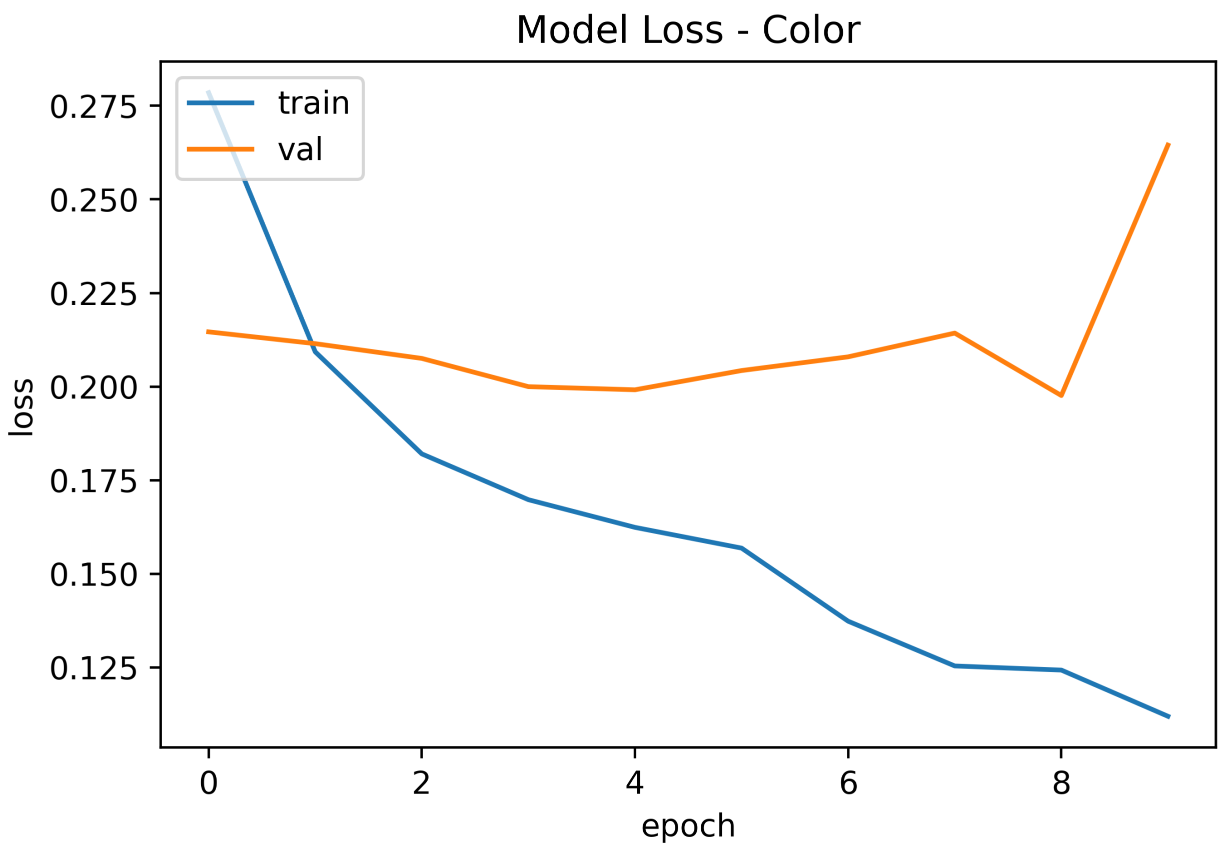


The graph of the loss function tells a similar story, where the training data has a better score than the validation data's associated score. It also appears that there is a critical point where the loss function for the validation data minimizes, and then proceeds to worsen with additional iterations.



These were promising results, but I was curious If somewhere upstream in the data pipeline, I had made a decision that prevented the model from being as optimized as it could have been I decided to re-run the same model, with the same hyperparameters, but this time have the images saved in color. As it turns out, this is a much better result.





This demonstrates that there is plenty of useful information that gets lost when removing color from an image. Particularly in cases like this, keeping the color in the image can improve the quality of the model by several percent, so it is worth keeping it in place.

Conclusion

This work shows that creating a model that can detect if a person is wearing a facemask is not only possible, but a viable strategy for ensuring that masks are being worn, and that they are being worn properly.

Using just 853 source images, a model was able to be trained that was over 93% accurate. With those kinds of results for a relatively small dataset, I anticipate that improved results would be found with additional images.

If this technique were used to verify that masks were being worn correctly in public spaces, I anticipate that public safety would increase as well.

References

Academy, Pink. “Wolrd’s Most Complete Masked Face Recognition Dataset Is for Free.” Medium, 16 July 2020, https://medium.com/the-programming-hub/wolrds-most-complete-masked-face-recognition-dataset-is-for-free-10d780eed512.

```
A large dataset of images of people wearing masks and not wearing masks.
Seems to come from China, as the readme is in Chinese.
```

“Applications in Response to COVID-19: Mask Detection.” Deep Learning, https://blogs.mathworks.com/deep-learning/2020/06/11/mask-detection-using-deep-learning/. Accessed 16 Jan. 2021.

```
A blog post from the mathworks team about their efforts to create a mask
detection model. Uses MatLab model, rather than a traditional Tensorflow
model for training and inference.
```

“COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning.” PyImageSearch, 4 May 2020, https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/.

```
Provides source code for a mask detection algorithm that can function on
images and video streams. Details the entire image processing pipeline,
which includes detection of faces, and the serialization of those faces
into the model for inference.
```

Escarlate, Alberto. “My Quarantine Project: A Real-Time Face Mask Detector Using Tensorflow.” Medium, 16 June 2020, https://towardsdatascience.com/my-quarantine-project-a-real-time-face-mask-detector-using-tensorflow-3e7c61a42c40.

```
Describes the efforts of one person’s quarantine project to build a mask
detection algorithm. Tensorflow is used, but no source code is provided.
```

Face Mask Detection. https://kaggle.com/andrewmvd/face-mask-detection. Accessed 16 Jan. 2021.

```
Data source for face mask images. Contains images of not only
mask/maskless people, but also of subjects who are wearing a mask
incorrectly. Could be used to make a more nuanced classifer.
```

Face Mask Detection in Street Camera Video Streams Using AI: Behind the Curtain | Tryolabs Blog. https://tryolabs.com/blog/2020/07/09/face-mask-detection-in-street-camera-video-streams-using-ai-behind-the-curtain/. Accessed 16 Jan. 2021.

```
Describes a technique for face mask detection that detects human poses,
then human faces, and then detects if a mask is being worn. Starting from
human poses is useful for surveillance footage.
```

“Face Mask Detection with ML/AI on Cisco Industrial Hardware.” Cisco Blogs, 25 Aug. 2020, https://blogs.cisco.com/internet-of-things/face-mask-detection-with-ml-ai-on-cisco-industrial-hardware.

```
A developer at Cisco describes his attempt at making a mask classifer. A
link to a docker image is given.
```

“How to Build a Face Mask Detector Using RetinaNet Model!” Analytics Vidhya, 24 Aug. 2020, https://www.analyticsvidhya.com/?p=69192.

```
Provides a fairly in-depth explanation for the math and architecture
behind RetinaNet, and justifies its use for this application.
```

“Implementing a Real-Time, AI-Based, Face Mask Detector Application for COVID-19.” NVIDIA Developer Blog, 18 Aug. 2020, https://developer.nvidia.com/blog/implementing-a-real-time-ai-based-face-mask-detector-application-for-covid-19/.

```
Provides a developer recipe for how one would build a mask detection model
for use in healthcare facilities.
```

Loey, Mohamed, et al. “A Hybrid Deep Transfer Learning Model with Machine Learning Methods for Face Mask Detection in the Era of the COVID-19 Pandemic.” Measurement, vol. 167, Jan. 2021, p. 108288. PubMed Central, doi:10.1016/j.measurement.2020.108288.

```
Researchers made a model for mask detection. Results are published with
helpful graphs that would be helpful for informing how I would like to
disply the information at the end of this project.
```

Mujtaba, Hussain. “Real-Time Face Recognition| Real Time Face Recognition OpenCV Python.” GreatLearning, 14 Jan. 2021, https://www.mygreatlearning.com/blog/real-time-face-detection/.

```
Describes an implementation of mask detection using the OpenCV library.
This library is useful for face detection in general, but also can help
with mask detection.
```

“(PDF) A FACEMASK DETECTOR USING MACHINE LEARNING AND IMAGE PROCESSING TECHNIQUES.” ResearchGate, https://www.researchgate.net/publication/345972030_A_FACEMASK_DETECTOR_USING_MACHINE_LEARNING_AND_IMAGE_PROCESSING_TECHNIQUES. Accessed 16 Jan. 2021.

```
Describes a mask detection model that was constructed using keras. Several
different types of Neural Networks were used and compared.
```