Face Mask Detection

A multimodal analysis leveraging multiple datasets

Danya Gordin, Andrew Lemke, William Collins, Ashish Ernest

Table of Contents

Abstract

Face masks have been shown to be essential in mitigating the spread of a variety of diseases and thus proper adherence can be life-saving in certain scenarios. Automated face mask detection would enable communities and organizations to more easily enforce any necessary mask guidelines. Our project developed four different face mask detection algorithms that each leveraged a variety of datasets including both large synthetic datasets of masks digitally superimposed on faces as well as smaller realistic datasets of masks in a variety of positions and locations. The first approach used a face-detection model called MediaPipe to train a CNN from scratch and achieved an accuracy of 85%. The second approach fine-tuned pre-trained CNNs, GoogLeNet and InceptionV3, to achieve 97% accuracy on curated data of cropped faces. The third approach utilized two-stage pre-trained models, R-CNN and VGG-19, to achieve nearly 95% accuracy on real-life images. The final approach used a state-of-the-art computer vision model, YOLOv5, to achieve .96 mAP on a variety of real-life images as well as on real time video feed. We showed that models trained on combined datasets outperform models trained on just one dataset when evaluated on real life data. We also confirmed that the transfer learning approaches greatly outperform any CNNs built from scratch.

Keywords: face mask detection, computer vision, convolutional neural network, synthetic training data, transfer learning, pretrained model

Introduction

Face masks have been proven to be extremely effective at mitigating the rapid transmission of Covid-19[16] and other diseases. Many local and federal governments enacted face mask mandates in response to their effectiveness. With proper mask wearing a constant concern in many situations, the need for a robust mask detection AI would enable communities and organizations to more easily enforce any necessary mask guidelines.

A major obstacle in the development of mask detection AI is the severe lack of large high quality datasets with proper annotations. This "data insufficiency" problem is extremely common in the context of computer vision since datasets that contain large amounts of labeled images can be impractical to generate. While many datasets were created for our specific problem, no one dataset had enough instances or matched the problem domain well. With the unusual circumstances from the disruption of normal life from covid, we can not expect high quality and large datasets in similar situations in the future. The data that available falls into two broad categories:

1. Organic high quality dataset with few instances.
2. Synthetic dataset that do not reflect reality well, but exist have many instances. Synthetic datasets are created by editing pictures of facemasks over normal pictures of faces in an automated way.

Figure 1. On the left, we see an example of the first type of dataset. The right picture is an example of the second kind of dataset. Notice the artifacts and unrealistic nature of the mask.

This divide reflects a fundamental machine learning (ML) balance, bias and variance. Short but realistic data will lead to models with high variance, as there is insufficient data to train a model of appropriate complexity for the problem. Synthetic, but numerous datasets, will lead to models with high bias. They may not generalize well on data that is within the problem domain but may perform well within the limited dataset's domain. One dataset available for our problem

is titled “Face Mask Detection” [5]. It falls into the first category. Each example is a real life picture of a person or persons from a wide variety of settings, portraits to pictures of crowds, and each instance has every face and mask labeled. The high amount of work required to create the dataset limited it to only 853 instances. Contrastingly, the “MaskedFace-Net” [4] dataset contains 133,000 instances, but each is of a face oriented toward the camera with the same surgical blue face mask applied to each image. Each of the 133,000 masks are the same color, and their digital application leaves artifacts which may be detected quickly by a computer vision model, causing good training and test performance, but poor performance when used in any real life context. These datasets often rely on the detection of facial features to automatically apply the mask, meaning only high quality “easy” images can be edited to include a mask. With no one dataset poised to produce a high performing realistic model, leveraging multiple sources of data will help address the “data insufficiency” problem. One option is to combine datasets into a larger one while another option is to use pre-trained models, also known as transfer learning.

Transfer learning helps overcome this limitation by leveraging another model that has previously been trained on massive amounts of more general data. More specifically, transfer learning works by using the new limited data to retrain the latter layers of a network. The early and middle layers transferred from the original network often describe more general image features such as edge detection and shape and thus require much less retraining.

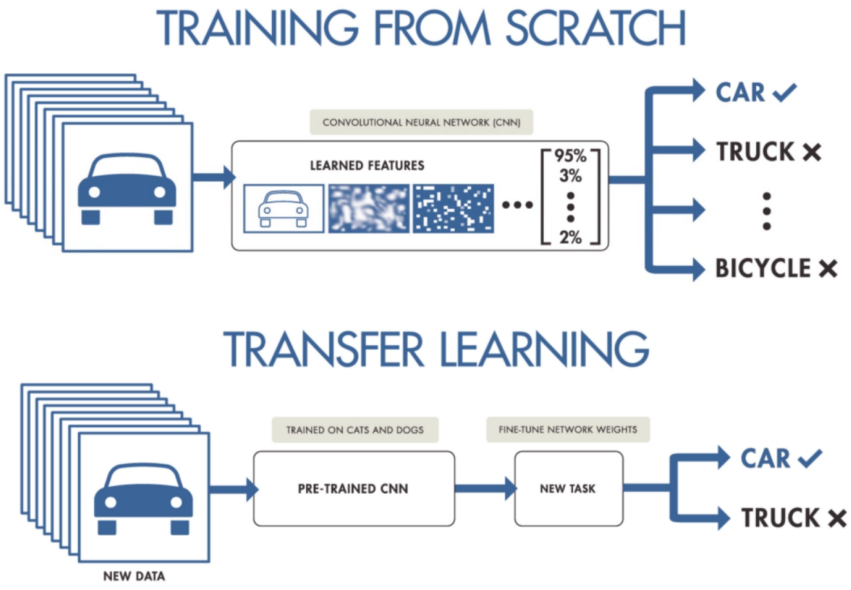

Figure 2. Comparing transfer learning and learning from scratch.

Overall, transfer learning approaches often result in faster training times and better performance metrics while simultaneously requiring significantly less data.

Convolutional Neural Networks (CNN) have long been the go-to machine learning technique for image processing. Various approaches to the problem of face mask detection have been implemented with varieties of CNNs. One such example was reported in the paper "Facial Mask Detection Using Depthwise Separable Convolutional Neural Network Model During COVID-19 Pandemic" [1]. In this paper, Ashgar et al. used a CNN with depth-wise separable convolutions, a form of quantized convolution, which greatly reduces the number of parameters used by the model and allows it to run on mobile apps. YOLO-v3 [2] (You Only Look Once) object detector, MobileNet-v2 [3], and VGG-16 [7] are a few other CNN based architectures which have been studied for use with face mask detection.

Data

Ways to Wear a Mask or Respirator DB (WWMR-DB)

WWMR-DB consists of 1222 images divided into 8 classes covering faces with no mask worn, mask worn correctly, and various ways of wearing a mask incorrectly. The data is partitioned between 42 subjects displaying the various mask wearing styles. There are both surgical masks and cloth masks present. Four different types of masks are worn and the subject's head is angled at 0, 45, and 90 degrees. The images come with annotations giving the bounding box coordinates of the head, the mask, and the face. Of the three datasets we worked with, this one is arguably the highest quality. The images are roughly the same dimensions and feature one unobscured person.

Figure 3. Example of Subject 1 wearing a non-surgical face mask folded on top of the chin with his head at a 45 degree angle.

Face Mask Detection (FMD)

This is a dataset composed of 853 images hosted as part of a Kaggle competition. These images come in all shapes and sizes and have any number of people in them, often partially obscured or blurred in the background, wearing a variety of mask types. Because of this, it was a very difficult set to work with. This dataset also came with annotations giving the bounding box coordinates for each head along with flags indicating if the head is partially obscured. We used this information to crop each head from each image, and omit the head if its dimensions fell below a set threshold.

Figure 4. Example of an image with multiple masked and unmasked people obscured to varying degrees.

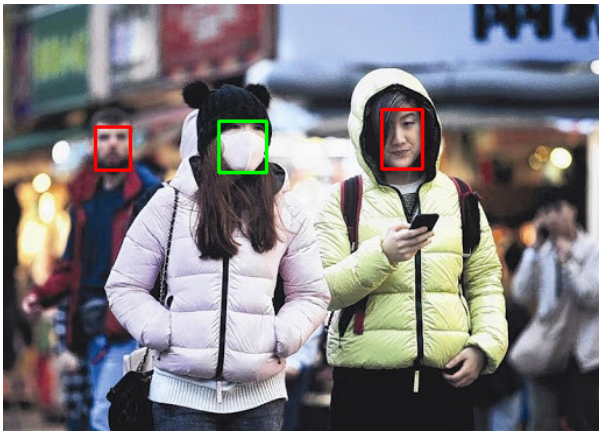

Figure 5. Example with the annotations visualized. The green box is for correctly worn, the red boxes are for not worn. The third class (incorrectly worn) is not shown in this example but is present in the dataset.

MaskedFace-Net (MFN)

This dataset consists of 133,000 face images (1024 x 1024 pixels) with surgical masks synthetically mapped onto them through digital mesh distortion. The mask mapping results are not perfect. There are plenty of telltale distortion artifacts in the masks, and the lighting of the masks is not adjusted to suit the rest of the image. Resampling these images at a lower resolution (e.g. 128x128 pixels) may reduce the prominence of these artifacts. The benefit of this dataset lies in its size. Generally speaking, the more complex and large the machine learning model (as is certainly the case for image recognition models), the more data is needed to train it in order to prevent underfitting. The potential harm of using this full dataset is that the model would overfit on surgical masks with the specific lighting conditions all of these have. For this reason, we need to be careful about balancing our final dataset over the three different sources.

Figure 6. Example of synthetic surgical masks digitally warped in an attempt to realistically map onto a face. The image on the left displays a partially successful mapping although the mask is not looped to the girl's ears. The image on the right is an obvious failure.

MLFW: A Database for Face Recognition on Masked Faces (MLFW)

MLFW is a synthetic dataset of 12,000 images. The type of digital mask varies from image to image, including a variety of surgical-styled and cloth masks. An attempt was made to match the brightness of the image when applying the masks. Each instance is a 112x112 pixel color image. The images are centered and of relatively similar style. The labels are with mask and without mask, and the class balance is 0.75 in favor of masked faces.

Figure 7. Examples of the MLFW dataset.

Methods and Code

Approach 1: Two Stage Approach with Face Detection and CNN from Scratch

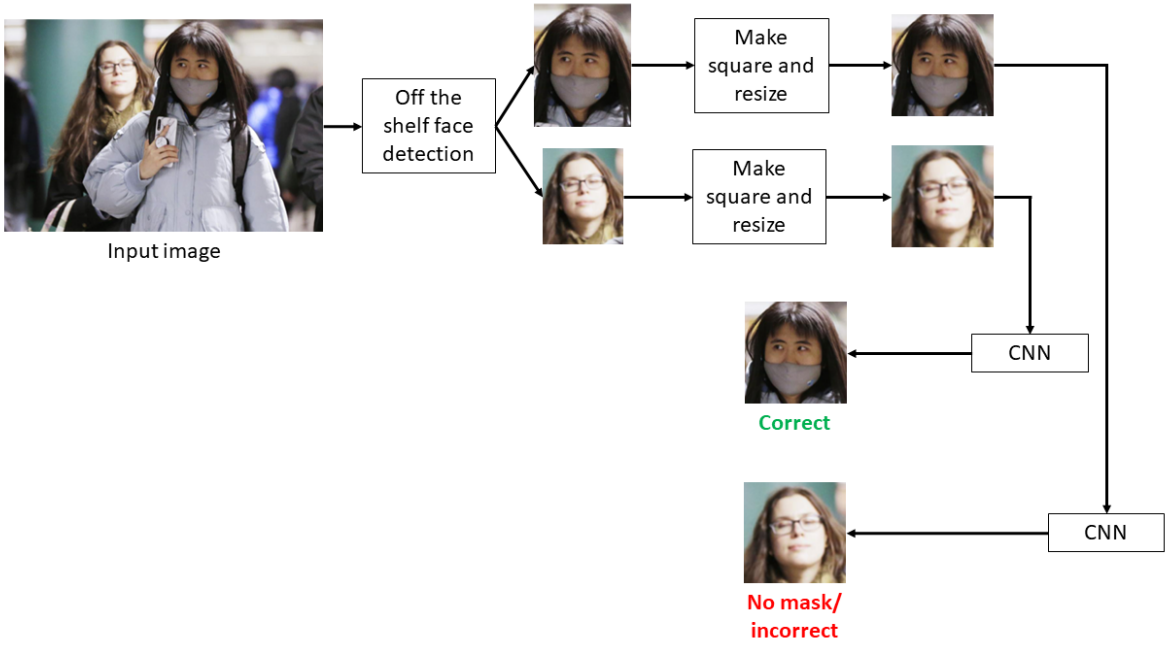

Figure 8. Overview of the 2 stage approach with face detection then CNN.

Using just a CNN as the classifier is not recommended for several reasons. First, the input images have no guarantee of the same aspect ratio. Most mask datasets did not have a common width to height ratio. Cropping could be problematic as there is a chance of cutting out faces that need to be classified. Combining the rare datasets that do have common aspect ratios leads back to the first problem, where datasets are of different aspect ratios, but even if they all did match by coincidence, we could not predict on multiple faces per image, which we need to classify if a particular person is correctly wearing a mask. The third issue is that a CNN that processes entire images would need to require a tremendous amount of trainable parameters, further driving the need for more data, which we don't have of sufficient quality.

These problems motivated the two-stage solution. The first stage is off the shelf face detection software, which identifies faces in images. Then, these face bounding boxes are processed to a standard form. These images are used to train a CNN, forming the second stage. Test images are passed through the face detection algorithm and then processed in the exact same manner before being sent to the CNN for classification.

The choice of face detection software is more critical than initially realized. The first face detection library initially used was Face Recognition [19]. Another popular face detection algorithm is MediaPipe [18]. These two libraries were pitted against each other on a tough dataset of images containing masked, partially masked and unmasked faces. MediaPipe clearly dominated Face Recognition. MediaPipe outperformed Face Recognition by a factor of 2.95 on the entire dataset and by a factor of 3.15 when looking at masked images. MediaPipe was chosen for the first stage of the model. This both helps in test time and with training. As MediaPipe can glean 3 times the number of faces from a dataset, the training data is 3 times larger with this face detection scheme. In test time, MediaPipe would be able to recognize more faces, increasing the total effectiveness of the model.

Face Detector	**No Mask Accuracy**	**Incorrect Mask Accuracy**	**Mask Accuracy**	**Weighted Average**	**Numeric Average**
Face Recognition	19.8%	22.0%	13.3%	14.7%	18.4%
MediaPipe	48.3%	50.4%	41.9%	43.3%	46.9%

Table 1. Face detection algorithms compared.

The output of the face detection algorithm needs to be processed. A square bounding box is created by expanding the narrower of the two dimensions of the box to form a square. The image is expanded in such a way that the center remains fixed. For example, a 90x74 bounding box would become a 90x90 bounding box by growing 8 pixels in each direction along the shorter dimension. This transformation maintains centering on the face. The bounding boxes were quite tight on the face and sometimes missed important details. They were grown by 12.5% in each

dimension. The face bounding square needed to be re-scaled (either up or down) to a fixed dimension for the CNN. A size of 112x112 was chosen.

The CNN will classify the output as correct or not correct. The choice for 2 outcomes (correct mask/not correct mask + no mask) is to not limit the data that can be used for training this model as some datasets only have two labels. Three label datasets can be made into two label dataset by grouping the no mask and not correct mask classes. The CNN is able to train from scratch because the cropped bounding boxes reduce the complexity of the problem in three ways: shape size standardization, reduction of background clutter, and reduction of size.

The outputs of the bounding box crops are faces of similar sizes relative to the image size, which is important as CNNs are good at classification despite translation alterations, but do not have any architectural advantages for size related alterations. The cropped bounding box removes background information that may distract the model as the cropped images almost entirely consist of a face (with or without mask). The size reduction makes training easier and better. When cropping to the bounding box and rescaling, the output is a smaller image of only the important information for the classifier. This keeps the number of trainable parameters low, improving results when working with limited data as overfitting is less likely in a simpler model.

The CNN was constructed in PyTorch and consists of a number of convolutional layers followed by fully connected linear layers. The convolutional layers used channels of 8, 16, 20, 24, and 24 with square kernels of side lengths 10, 10, 10, 12, 12. ReLU activation and max pooling were performed after every convolutional layer. The pooling used a 2x2 window of stride 1 for every pooling layer except the last two, where a stride of 2 was used to reduce the number of nodes entering the fulling connected linear layers. The linear layers are of sizes 1000, 250, 10, and 2, with the final layer being the output layer. The selected loss function was cross entropy loss, after softmax, and the optimizer was stochastic gradient descent (called SGD in PyTorch).

All relevant code can be found in the folder “app/approach1_two_stage” at this link: https://github.com/barrelchester/data-mining.final-project.face-mask-detection

Approach 2: Fine Tuning Pre-trained CNN: GoogLeNet and InceptionV3

A variety of pretrained image networks are available through the torchvision library [13]. The GoogLeNet [8] model and related InceptionV3 [9] models were chosen due to having relatively lower parameter count while still scoring well on the ImageNet [10] classification task. GoogLeNet (Szegedy, et al) is a deep convolutional neural network consisting of 22 layers and 6.6 million parameters which is on the lower side for image models. It's able to function with fewer parameters due to its architecture being composed of "inception modules". These nine layered modules are multiscale feature detectors which serve to progressively reduce the dimensions of the input. In order to use the model for this task, the 1000 class output layer was removed and replaced with a new fully connected layer with 3 output nodes corresponding to the three classes "without mask", "with mask", "mask worn incorrectly".

To fit the GoogLeNet model, the images were resized to 128 pixels and normalized by subtracting the mean and dividing by the standard deviation of the ImageNet data. A learning rate scheduler was used to further reduce the LR upon the failure of the cross validation loss to decrease after several iterations.

The combined dataset was split into train, cross validation (CV), and test sets of sizes 2223, 250, and 500 respectively. Normally a larger percentage of the data would be used for CV and test, but due to the small size of the data set I chose smaller portions.

The model was trained with the Adam optimizer [14] with base learning rate 1e-4. Cross entropy loss was used as the objective function. A minibatch size of 8 was used and a model checkpointing method was used to store the trained model every time the CV test accuracy improved.

The InceptionV3 (Szegedy, et al.) model is closely related to GoogLeNet and has 48 layers and 27 million parameters. This model was trained very similarly to GoogLeNet, but the images needed to be resized to 299 x 299 pixels. Despite having more parameters, this model performed slightly worse on the test data.

All relevant code can be found in the folder "app/approach2_pretrained_cnn" at this link: https://github.com/barrelchester/data-mining.final-project.face-mask-detection/tree/master/app/approach2_pretrained_cnn .

Approach 3: Faster R-CNN and VGG-19

Faster R-CNN

In the field of deep learning, there is a set of ideas called region proposals that have been very influential in computer vision. The Yolo algorithm uses the sliding windows idea; you would take a trained classifier and run it across all of these different windows and run the detector to see if there's an object in the image. You could run the algorithm convolutionally, but one downside is that the algorithm just classifies a lot of the regions where there's clearly no object.

So Russ Girshik, Jeff Donahue, Trevor Darrell, and Jitendra Malik proposed in their paper an algorithm called R-CNN, which stands for regions with convolutional neural networks or regions with CNNs. R-CNN tries to pick only the few important regions to run your ConvNet classifier on. So rather than running your sliding windows on every single section of the image, you instead select just a few windows and run your ConvNet classifier on them.

The regions are selected for proposal by a segmentation algorithm that results in this output on the right, which is used to figure out what regions of interest might be. So, for example, the segmentation algorithm finds a blob in the image, and you might pick that region and run a classifier on it.

Figure 9. Example image for R-CNN. The center image emphasizes that many regions of images are blank. On the right is a segmentation of the image.

You would typically find around 2000 blobs and place bounding boxes around all 2000, then run your classifier on those blobs. The number of blobs can be much smaller than the number of positions on which you could run your ConvNet classifier on when using a sliding window approach.

Now, one downside of the R-CNN algorithm is that it is actually quite slow. So over the years, there have been a few improvements to the R-CNN algorithm. Russ Girshik proposed the fast R-CNN algorithm, and it's basically the R-CNN algorithm but with a convolutional implementation of sliding windows. The original implementation would classify the regions one at a time rather than in parallel as in fast R-CNN. This speeds up R-CNN quite a bit.

It turns out that one of the problems of fast R-CNN algorithm is that the clustering step to propose the regions is still quite slow and so a different group, Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Son, proposed the faster R-CNN algorithm, which uses a convolutional neural network instead of one of the more traditional segmentation algorithms to propose a blob on those regions, and that wound up running quite a bit faster than the fast R-CNN algorithm. Although, most implementations of the faster R-CNN algorithm are usually still quite a bit slower than the YOLO algorithm.

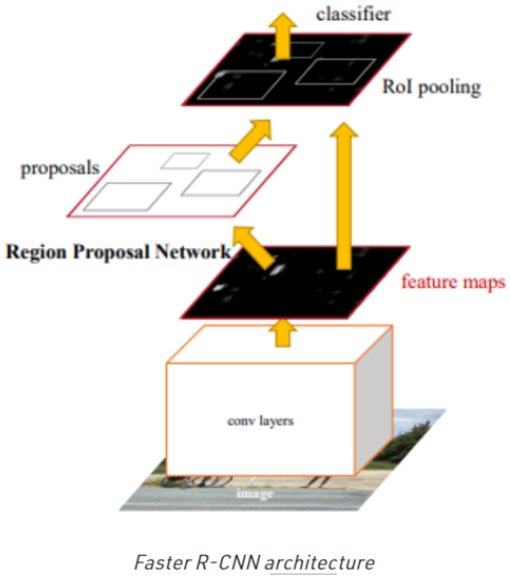

Figure 10. R-CNN architecture diagram.

VGG-19

Before we discuss the VGG-19 architecture, it is important to note that we first crop the face by detecting facial features in the input image and then pass on the cropped face to the neural network to be trained. So how do we detect these features in real time/in an image? The

answer is Haar Wavelets or Haar Features. And the algorithm used is called the Viola-Jones Algorithm.

Haar features are sequences of rescaled square shape functions proposed by Alfred Haar in 1909. They are similar to convolution kernels used for image processing. We will apply these Haar features to all relevant parts of the face so as to detect the human face. These features were trained by giving the algorithm a lot of positive images consisting of faces, and a lot of negative images not consisting of any face.

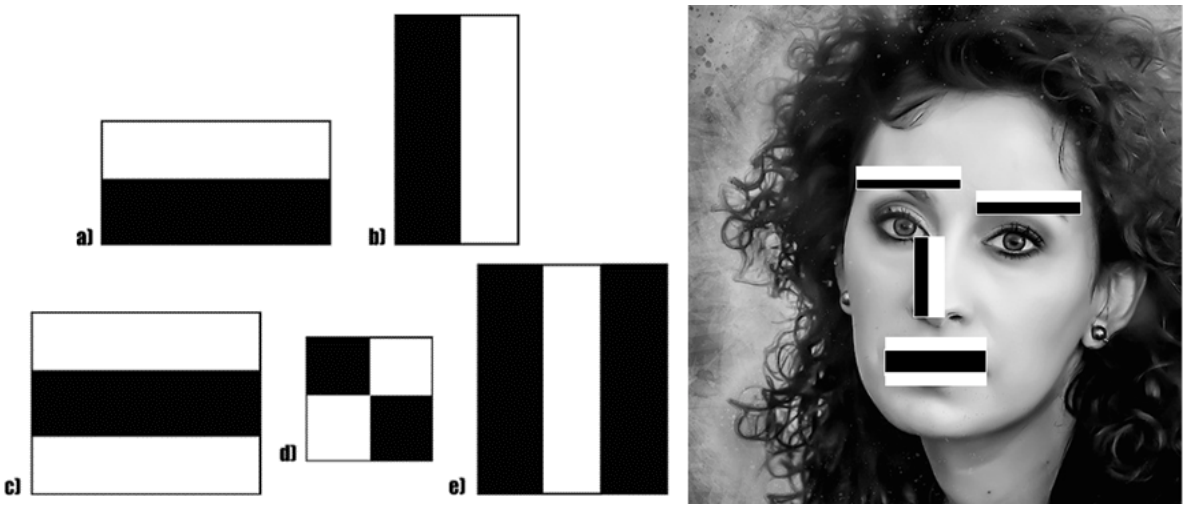

Figure 11. Haar feature filters are shown on the left. Their application to a face is shown on the right.

The objective here is to calculate the sum of all the image pixels lying in the darker area of the Haar feature and the sum of all the image pixels lying in the lighter area of the Haar feature, before finding the differences of the sums as shown in Equation 1. According to Viola-Jones algorithm, to detect Haar-like features present in an image, the below formula should give a result closer to 1. The closer the value is to 1, the greater the chance of detecting Haar features in the image.

$$\Delta = \text{dark} - \text{white} = \frac{1}{n}\sum_{\text{dark}}^{n} I(x) - \frac{1}{n}\sum_{\text{white}}^{n} I(x)$$

Equation 1. Haar feature equation.

Haar Cascade Classifiers are a series of classifiers based on the features explained above that are used to identify objects in an image. Using sliding windows and multiple sets of Haar features (increasing as the number of stages increase) leads to detection of a face. There are a total of 38 stages defined for Viola Jonas Method. Depending upon the sliding windows size, the face location, and the number of features, faces can be detected at a certain intermediate stages.

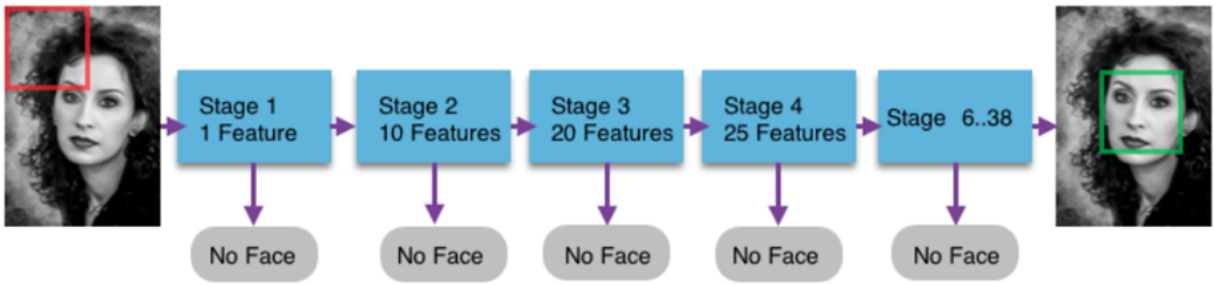

Figure 12. Viola Jones method diagram.

Now, the VGG-19 is a neural network architecture designed to be trained and used on image data. It consists of 19 layers (16 convolution layers, 3 Fully connected layers, 5 MaxPool layers and 1 SoftMax layer) and about a million parameters.

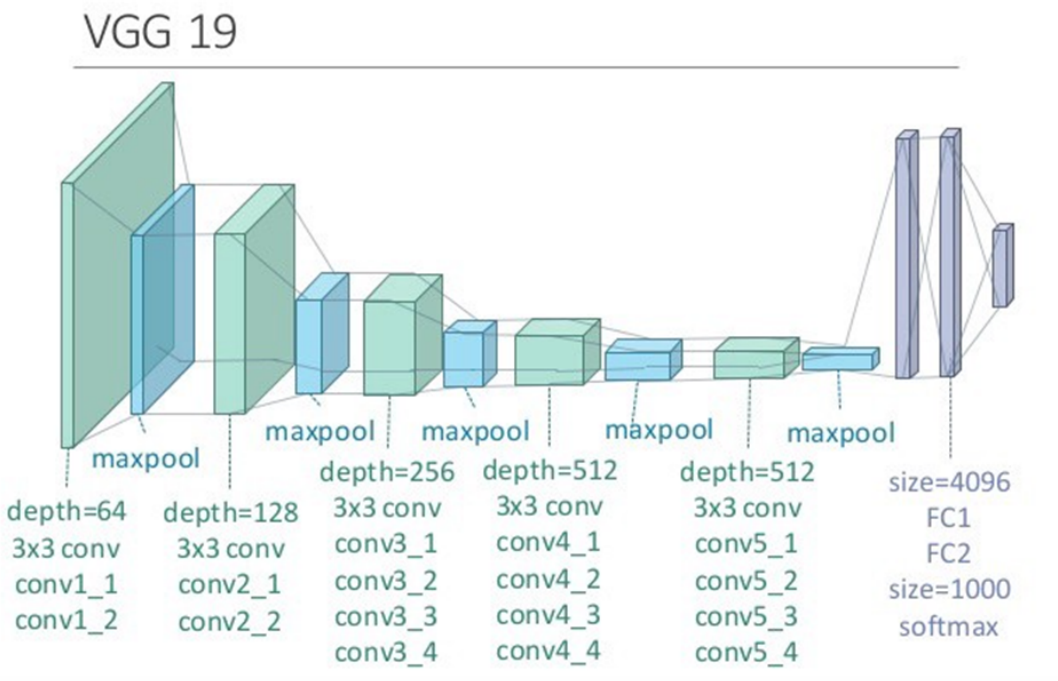

Figure 13. VGG-19 architecture diagram.

AlexNet came out in 2012 and improved on traditional convolutional neural networks; We can consider VGG a successor of AlexNet even though it was created by a different group of researchers at the Visual Geometry Group at Oxford giving this model its name, VGG. It carries and uses some ideas from its predecessors and improves on them, using deep Convolutional neural layers to improve accuracy.

VGG19 is used as a good classification architecture for many datasets. The authors made the models available to the public so they can be used as is or with modification for other similar tasks. VGG19 can be used for transfer learning for tasks such as facial recognition.

All relevant code for this approach can be found in the "app/approach_3_rcnn_vgg" folder at the link below:
https://github.com/barrelchester/data-mining.final-project.face-mask-detection/tree/master/app/approach3_rcnn_vgg

Approach 4: YOLOv5

YOLOv5[17] is a state-of-the-art object detection architecture that is very popular in the field of computer vision due to its speed and accuracy. We ultimately decided to use YOLO because it offered a few key advantages over our other methodologies:

1.) Better performance with the limited image data

2.) Faster processing that would allow for real-time detection.

Brief Introduction to YOLO

You Only Look Once (YOLO) is a real-time object detection system which departs from the traditional two-stage protocol that most other object detection algorithms follow by using an end-to-end fully convolutional neural net that makes predictions of bounding boxes and class probabilities all at once (Figure 14).

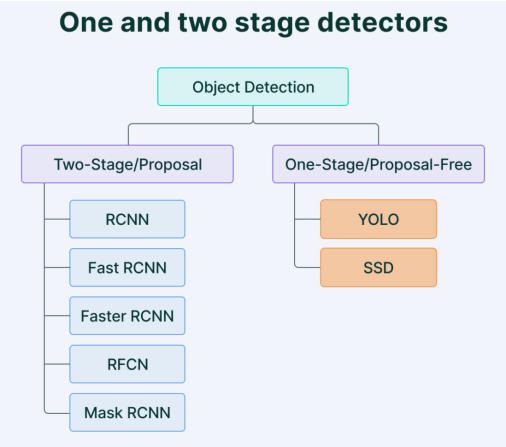

Figure 14. Popular two-stage and single-stage objective detection algorithms.

YOLO essentially works by dividing an image into a grid system, and each grid cell detects objects within itself. More technically, it applies 1x1 detection kernels on feature maps of three different sizes. The resultant feature map (grid cells) has detection attributes that include bounding box coordinates, object scores, and class scores (Figure 15).

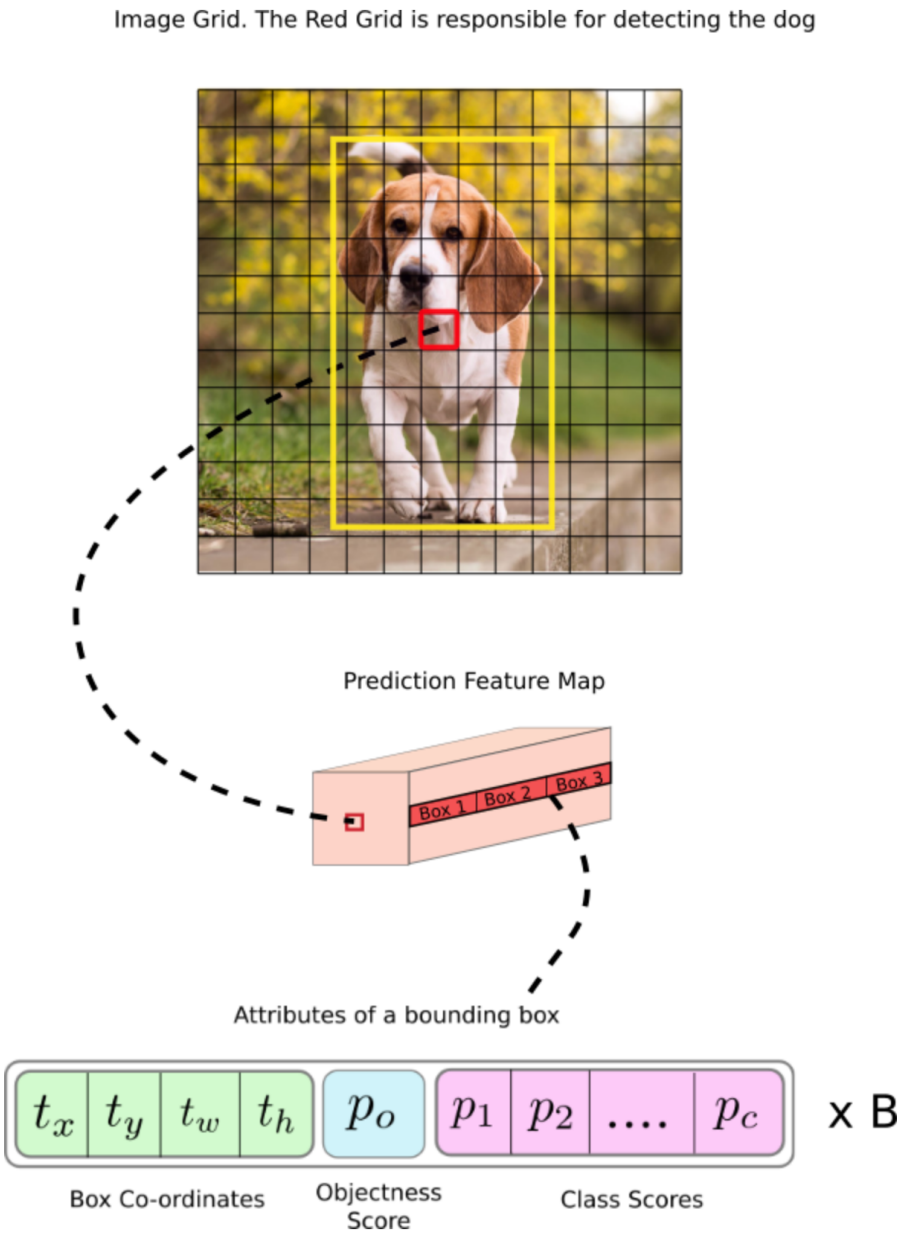

Figure 15. YOLO illustrated example.

Since both detection and recognition occur in each grid cell, multiple cells can predict the same object just with different bounding boxes. In order to identify the best bounding box for each object, YOLO applies a non-maximal suppression approach that suppresses any bounding boxes that have a large IOU (Intersection Over Union) with the highest probability bounding box (Figure 16).

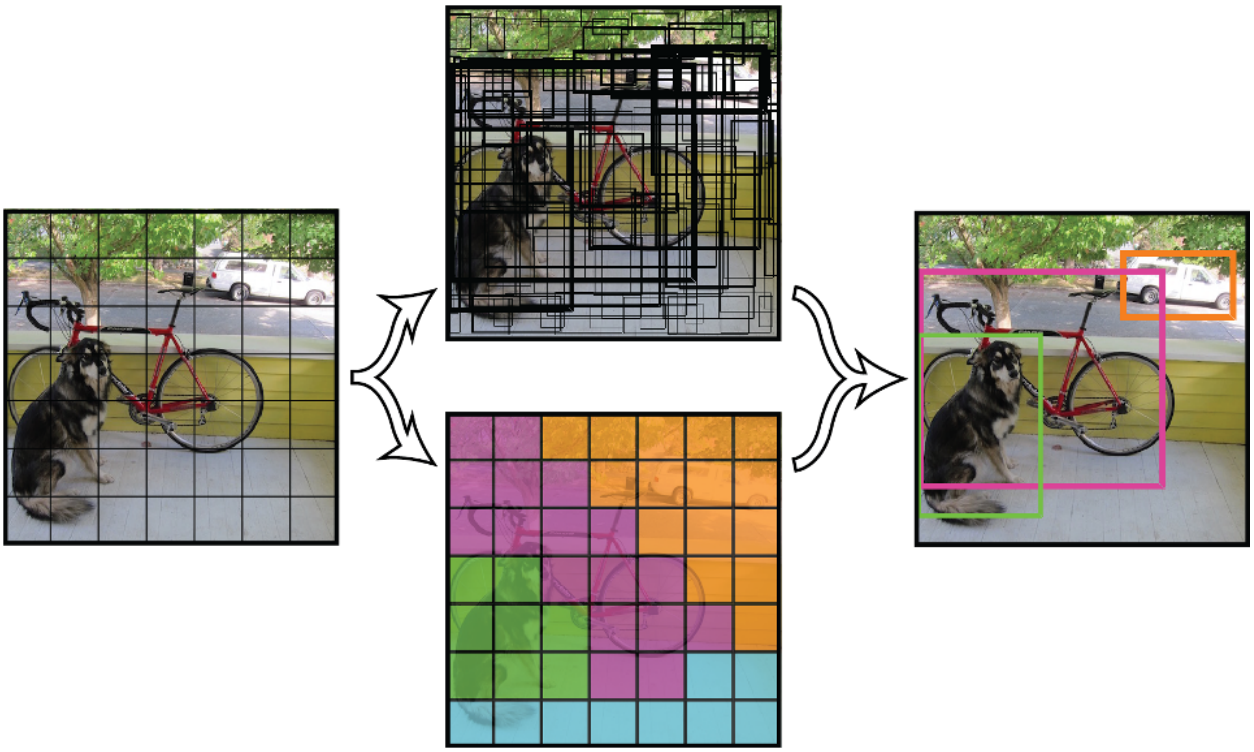

Figure 16. YOLO non-maximal suppression illustration.

Our overall goal was to train YOLOv5 to detect if a person was wearing a mask *correctly* (Class 0) or if a person was wearing a mask *incorrectly* (Class 1). Class 0 required the mask to fully cover the nose and mouth. Class 1 included detecting faces with no masks at all since a bare face wouldn't be allowed in any context that required face masks.

Pre-processing

The majority of the pre-processing involved converting bounding box annotations to YOLO format. Specifically, YOLO annotations are represented by a class value and 4 values describing the bounding box:

[class, x_center, y_center, width, height]

The training was performed using the WWMR-DB and FMD datasets. The WWMR-DB dataset had bounding box annotations that were already in YOLO format which meant we only had to change the class value for each annotation. Of the 8 different classes in the dataset, the images corresponding to the "mask worn correctly" class were assigned Class 0 and the rest of the image classes ("no mask", "mask worn under nose", etc) were assigned to Class 1.

The FMD dataset also had corresponding bounding box annotations for each image, but these were stored in Extensible Markup Language (XML) documents and used the Pascal VOC format. We used a python module (ElementTree XML API) that parses XML data to extract the bounding box data, converted the data to YOLO format, and finally wrote the data into a standard text document (.txt).

There were a total of 1815 images in the two datasets combined. Moreover, there was not a major class imbalance as Class 0 was represented by 3384 objects while Class 1 was represented by 1650 objects. Finally, 80% of the images and their corresponding annotations were assigned to the training set and the remaining 20% were assigned to the validation set.

Training

All training was performed using the default hyperparameters found in the .yaml file. These hyperparameters were originally optimized on the COCO dataset that YOLOv5 was originally trained on.

Due to the computational intensity of this type of training, we elected to use a Google Colab Pro with a P100 GPU. This means we were able to use more aggressive training settings to improve performance. The *image training size* was chosen to be 640 as this was the native resolution COCO was trained on. Since the face objects being detected weren't particularly small, there wasn't a compelling reason to train at a higher resolution. We chose to have the training run for 400 *epochs* to help mitigate overfitting. Finally, our hardware allowed us to use a relatively large *batch size* of 85 to greatly increase performance.

```
!python /content/yolov5/train.py --img 640 --batch 85 --epochs 400 --data data.yaml --weights yolov5s.pt --cache --patience 400
```

Figure 17. YOLO training settings.

Model performance metrics like accuracy and loss were captured in real time by using the experiment tracking tool Wandb. Moreover, this work is easy to reproduce as Wanb saves all the hyperparameters, terminal log, output file, and git state.

All relevant code can be found in the folder "app/approach4_yolo" at this link: https://github.com/barrelchester/data-mining.final-project.face-mask-detection

Results and Discussion

Approach 1: Two Stage Approach with Face Detection and CNN from Scratch

Unlike most testing, which draws from data exactly like the training data, the two stage model was evaluated against a never before seen dataset, the FMD dataset, which very well represents a real world use case. The FMD dataset was processed in the exact same manner as the training data. Labels were drawn from the provided annotated bounding boxes. To evaluate the performance increase of using multiple datasets, we evaluated models trained on individual datasets and a model trained off of all datasets combined.

Dataset	Precision	Recall	f1-score
WWMR	0.05	0.22	0.08
MLFW	0.79	0.80	0.80
All Datasets	0.84	0.82	0.83

Table 2. Performance of two-stage model on unseen dataset.

The models trained on the WWMR and MLFW datasets received a hyperparameter and architecture search, however due to the extensive amount of time (in the order of days) to train the all-dataset model, only one version was trained. Even without an architecture search, the model still outperformed the models trained on only one dataset. It is clear that most of the

performance in the combined model comes examples in the MLFW dataset as its performance was remarkable alone.

The model was then used to put a bounding box on video footage. Each frame was captured, then the face detection algorithm was run. If a face was detected, then it was processed in the fashion described above and sent to the CNN for classification. The results were impressive. Of the face orientations and mask positions tried, only mask under nose in a profile shot was misclassified by the model.

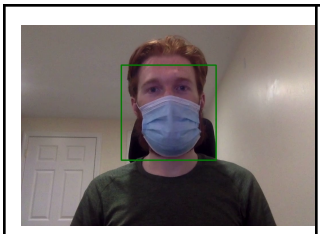	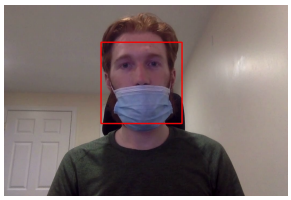	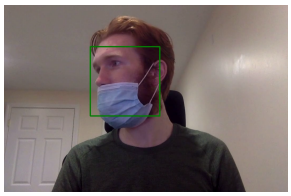	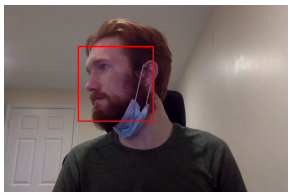
Correctly Classified	Correctly Classified	Incorrectly Classified	Correctly Classified

Figure 18. Frames from video processed and classified by the two-stage CNN model.

Approach 2: Fine Tuning Pre-trained CNN: GoogLeNet and InceptionV3

A. GoogLeNet Results

Due to the lower parameter count of the GoogLeNet model, I was able to train it on a CPU rather than a GPU. At only epoch 4 the CV test accuracy reached its highest score at 97.6%, which was surprisingly quick. The final accuracy of the model on the test set was 97.0%, meaning the model correctly predicted 485 of the 500 test samples. This was an extremely impressive result given the variety of the data and often subtle ways in which a mask could be worn incorrectly. For contrast, in the paper by Ashgar et al. (March 2022) reported state of the art results with 93.12% accuracy on their test set. See the figures below for the training and CV loss curve, classification report details, and confusion matrix.

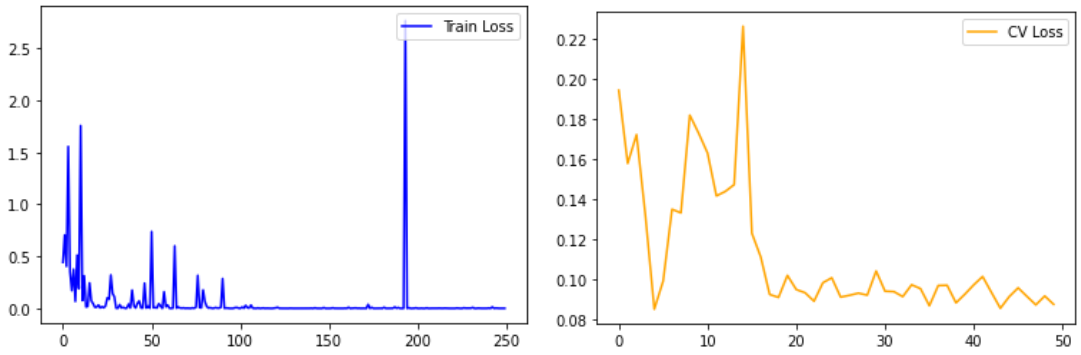

Figure 19. Training and cross validation loss curves.

	precision	recall	f1-score	support
without_mask	0.9586	0.9878	0.9730	164
with_mask	0.9755	0.9755	0.9755	163
mask_weared_incorrect	0.9762	0.9480	0.9619	173
accuracy			0.9700	500
macro avg	0.9701	0.9704	0.9701	500
weighted avg	0.9702	0.9700	0.9699	500

Figure 20. Classification report details for the test set.

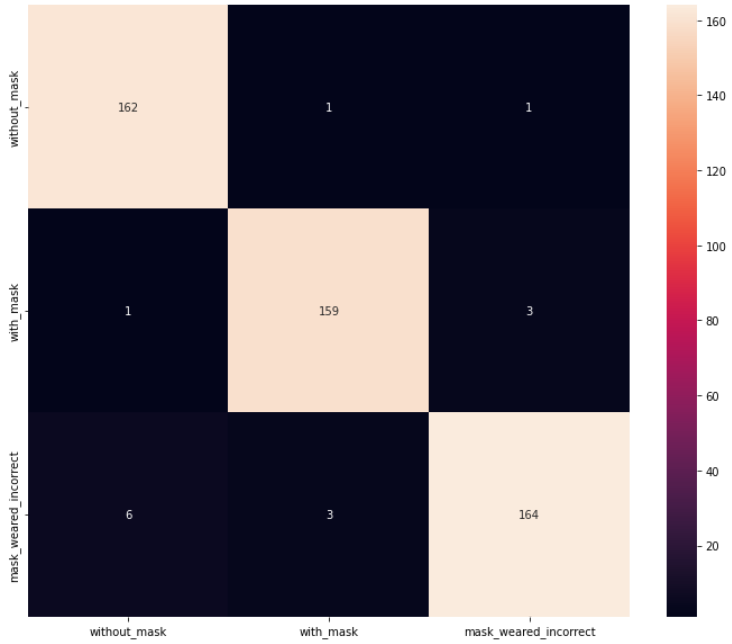

Figure 21. Confusion matrix for the test set.

The 15 images that the model predicted incorrectly were reviewed to look for possible reasons why the mistakes were made. Of the images incorrectly predicted 3 were bad crops/blurry images, 4 were people with masks under their chins that were barely visible and virtually indistinguishable from clothing, and several were questionably labeled.

B. InceptionV3 Results

The InceptionV3 model had slightly worse test results, 96.4% compared to 97%, but still performed very well. When analyzing the images it predicted incorrectly, it was interesting to note that a number of them were the same ones that GoogLeNet also predicted incorrectly. This seems to indicate an issue that could perhaps be resolved through better data selection and pre-processing. I believe this model performed slightly worse due to the need to upscale the images to 299 pixels. When downscaling, some of the noise is naturally reduced so that the network can concentrate on the essential components of the image. This in addition to the higher parameter count makes it easier for this model to overfit on spurious image features.

Approach 3: Faster R-CNN and VGG-19

Faster R-CNN

Given below is a sample image passed through the trained model.

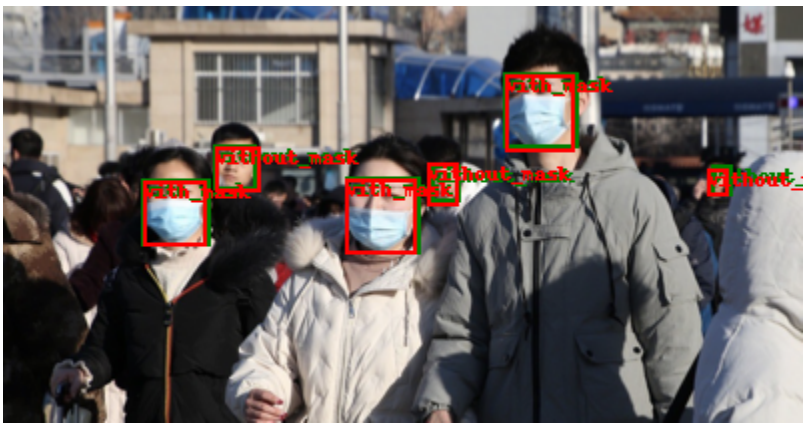

Figure 22. Faster R-CNN classification visualized on a test image.

As can be seen from the image above, our model does a pretty good job at distinguishing between people who have worn the mask properly, people who have not worn the mask properly, and people who have not worn a mask at all.

We have also used the 'coco_eval' toolbox to evaluate our model's performance on the test data. This helps us calculate the precision and recall for each level of IoU (Intersection over union). Intersection Over Union (IoU) is a number that quantifies the degree of overlap between two boxes. In the case of object detection and segmentation, IoU evaluates the overlap of Ground Truth and Prediction region.

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.470
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.757
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.519
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.347
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.694
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.799
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.277
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.518
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.540
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.422
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.760
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.823
```

Figure 23. Performance of R-CNN.

VGG-19

Given below is a sample image passed through the fine-tuned VGG-19 neural network.

Figure 24. VGG-19 classification on a test image.

We have also evaluated the model on test data to achieve an accuracy of 0.9775 and a loss of 0.07 as can be seen from the screenshot below.

```
[ ]  1 # Evaluate model performance on test data
     2 model_loss, model_acc = model.evaluate(test_generator)
     3 print("Model has a loss of %.2f and accuracy %.2f%%" % (model_loss, model_acc*100))

     25/25 [==============================] - 9s 350ms/step - loss: 0.0678 - accuracy: 0.9775
     Model has a loss of 0.07 and accuracy 97.75%
```

Figure 25. VGG-19 performance metrics.

The VGG-19 model performs much better in comparison to the Faster R-CNN algorithm.

Approach 4: YOLOv5

Validation Images

YOLOv5 ended up working remarkably well. This model is able to identify if a person is wearing a mask correctly regardless of the face size, face direction, mask type, or the mask

position. More specifically, we can see that the model is able to correctly classify both forward-facing faces and profiles (Figure 26).

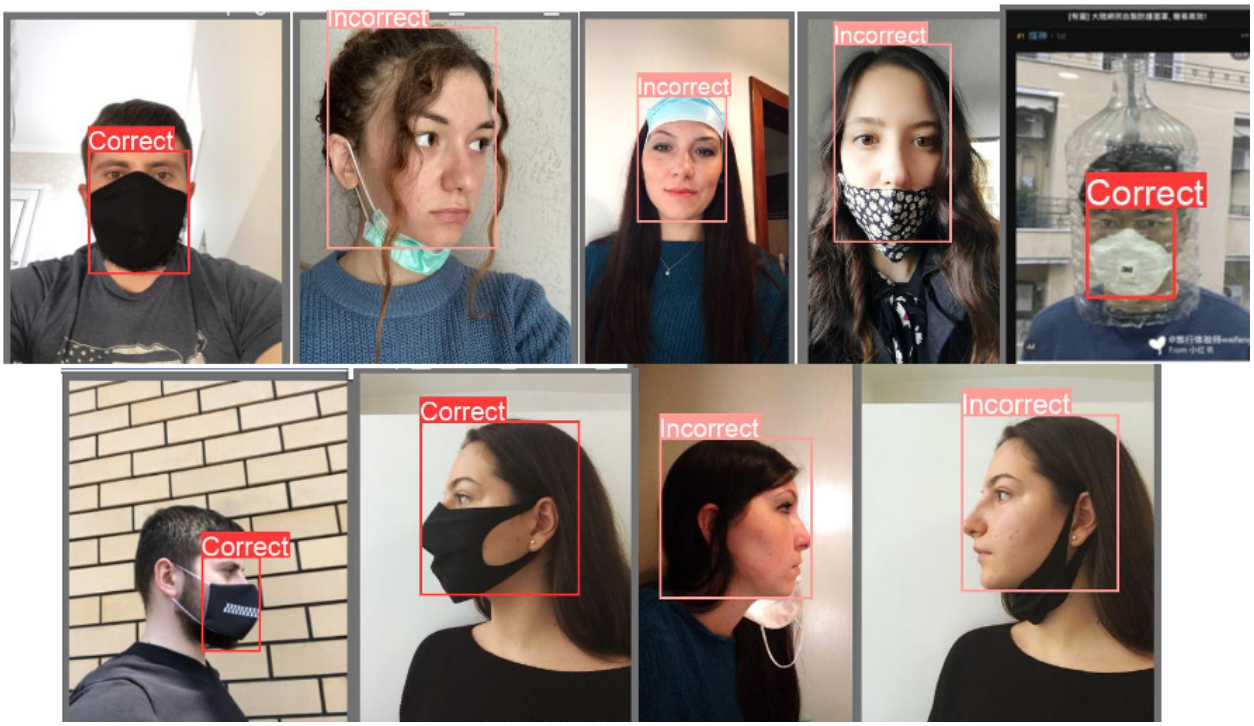

Figure 26. YOLO model classification on test images of a single person.

Furthermore, the model performed very well in images of crowds where there are many faces that are usually quite small (Figure 27).

Figure 27. YOLO model classification on test images of a crowd.

However, it still makes some mistakes, especially in images where there is something covering the face that is not a mask (Figure 28).

Figure 28: YOLO model misclassifications.

Metrics

The metrics verify the strong performance that was apparent from the object detection in the validation images above. The confusion matrix in Figure 9 shows that the model identified 94% of all people correctly wearing their masks and 90% of all people incorrectly wearing their masks. Interestingly, most of the error seems to be coming from false objects being detected in the background. This type of problem could potentially be solved by increasing the image size setting during training as perhaps some splotchy background shapes are being confused for faces.

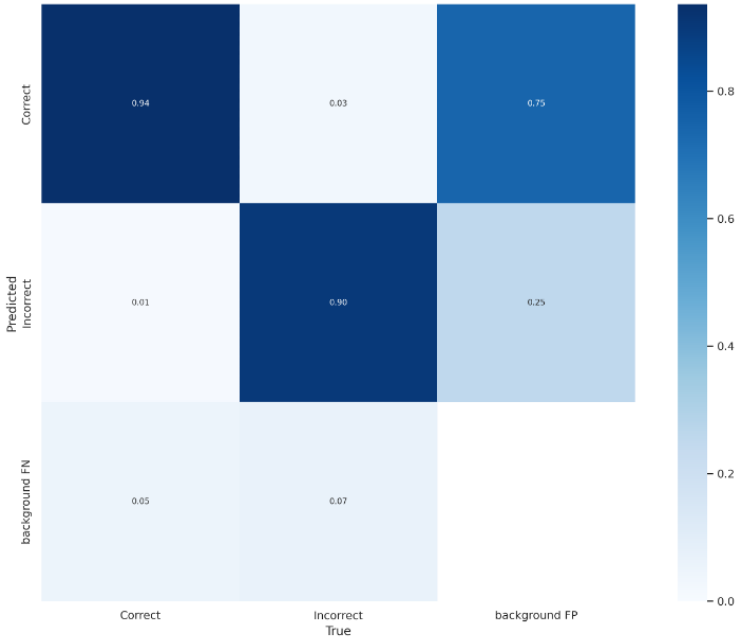

Figure 29. YOLO model performance metrics.

In the field of computer vision, there are a few metrics that are commonly used to evaluate model performance:

1.) **Precision:** Model confidence in making a correct classification for a given class
2.) **Recall:** Model confidence in classifying a given class as that class
3.) **F1 Score:** A measure of the balance between precision and recall.
4.) **Average Precision (AP):** The average of all precision values weighted by their corresponding precision values.
5.) **Mean Average Precision (mAP):** Captures the AP of every class at some IOU values.
    a.) mAP_0.5 is the mAP at an IOU value of 0.5
    b.) mAP_0.5:0.95 is the mAP for IOU values between 0.5 and 0.95. This is generally the most stringent metric as it averages in high IOU requirements.

Using WanB, we were able to experimentally track many of these metrics in real time while the model was being trained (Figure 30). It took about 50 epochs for the model to reach a plateau in precision and recall values. However, the mAP precision was still gradually rising at 400 epochs which suggests the model could have been trained even further without risk of overfitting.

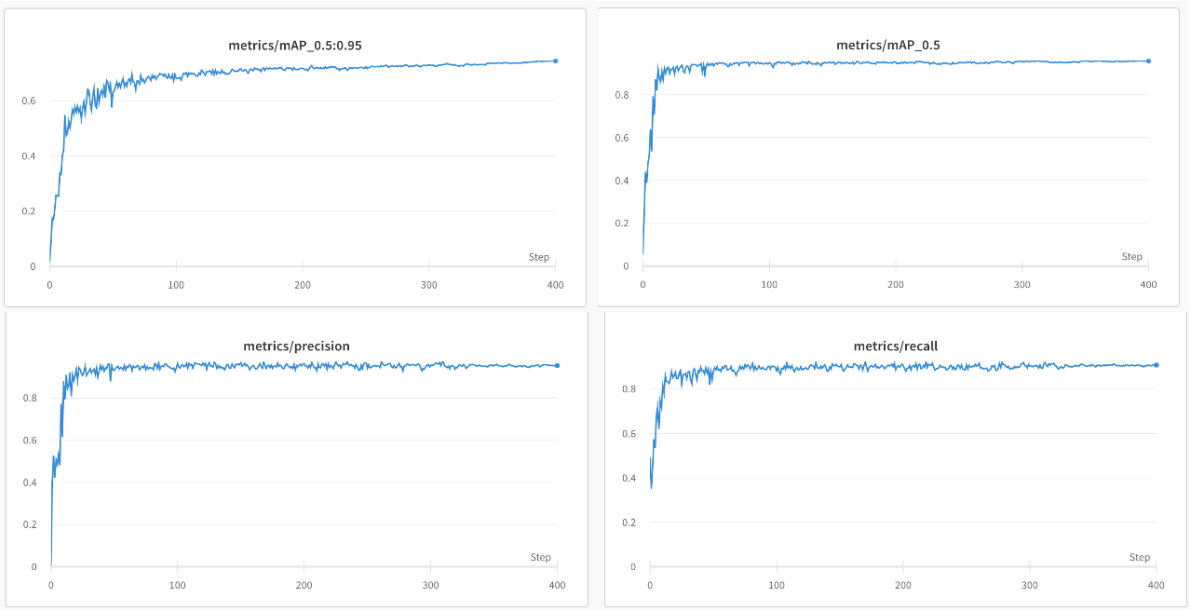

Figure 30. YOLO model MAP, precision, and recall.

The best performing model overall had an F1 Score of 0.93 and a mAP_0.5 of 0.958 as summarized in Figure 31.

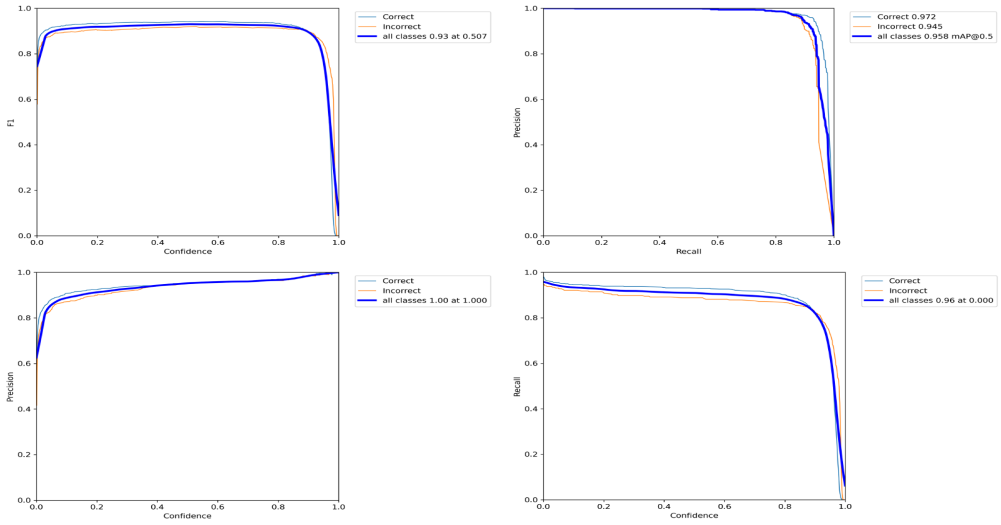

Figure 31. YOLO model performance metrics.

One particularly interesting result that can be gleaned from the Precision-Recall curve is that the people properly wearing face masks (“Correct” class) were more often detected and correctly classified than the people who were not properly wearing face masks (“Incorrect” class). This could be attributed to a few potential reasons. Firstly, the image dataset was slightly imbalanced in favor of the “correct” class giving the neural net more information on that class to train with. Secondly, there is really only one correct way to wear a face mask while there are a multitude of ways to incorrectly wear a face mask. Thus, the “Incorrect” class contains much more inherent variation.

Conclusion and Future Work

The 4 presented approaches each effectively mitigate the data quality and quantity problem by leveraging multiple data sources, whether that source manifests as weights in a pretrained model or simply as the union of several datasets. We have found that the pre-trained models outperform the 2-stage approach, which is one that combines multiple datasets. Pre-trained models are able to offer much more performance as their training observes many more images than the 2-stage model was trained on. Additionally, pre-trained models are highly

optimized and finely tuned using vast computer resources. Our pre-trained face mask detection models were very successful in leveraging this to achieve good results in our problem space without requiring copious amounts of data or compute power.

Future work could examine how much data is needed to train these models for different problems with hopes that the result would inform future modelers on the data requirements for various tasks. This could be furthered by examining if for a set number of image examples, performance from tuned pre-trained models improves in the problem space if those training images come from one dataset or multiple datasets.

References and Acknowledgements

ACKNOWLEDGEMENTS

The following resources were used for this project.

Masked Face Net data from https://github.com/cabani/MaskedFace-Net

WWMR-DB data from

https://ieee-dataport.org/open-access/ways-wear-mask-or-respirator-wwmr-db

Face Mask Detection data from

https://www.kaggle.com/datasets/andrewmvd/face-mask-detection?resource=download

Unmasked GAN Faces from

https://www.kaggle.com/code/theblackmamba31/generating-fake-faces-using-gan/data

MLFW: A Database for Face Recognition on Masked Faces data from

http://whdeng.cn/mlfw/?reload=true

REFERENCES

[1] Asghar, Muhammad Zubair. ‘Facial Mask Detection Using Depthwise Separable Convolutional Neural Network Model During COVID-19 Pandemic’. Frontiers in Public Health 10 (2022): n. pag. Web.

[2] Redmon, Joseph, and Ali Farhadi. ‘YOLOv3: An Incremental Improvement’. 2018. Web.

[3] Sandler, Mark. ‘MobileNetV2: Inverted Residuals and Linear Bottlenecks’. (2018): n. pag. Web.

[4] Adnane Cabani, Karim Hammoudi, Halim Benhabiles, and Mahmoud Melkemi, "MaskedFace-Net - A dataset of correctly/incorrectly masked face images in the context of COVID-19", Smart Health, ISSN 2352-6483, Elsevier, 2020, DOI:10.1016/j.smhl.2020.100144

[5] ‘Mask Dataset’. Make ML. Web.

[6] A. C. Marceddu. R. Ferrero and B. Montrucchio, "Mask and respirator detection: analysis and potential solutions for a frequently ill-conditioned problem," 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), 2022.

[7] Yoshua Bengio, Yann LeCun: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. 2015

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, & Andrew Rabinovich (2014). Going Deeper with Convolutions. *CoRR, abs/1409.4842*.

[9] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. doi:10.48550/ARXIV.1512.00567

[10] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248–255).

[11] Lars Buitinck, et al. "API design for machine learning software: experiences from the scikit-learn project." ECML PKDD Workshop: Languages for Data Mining and Machine Learning. 2013.

[12] Paszke, Adam. ‘Automatic differentiation in PyTorch’. (2017): n. pag. Print.

[13] Falbel D (2022). torchvision: Models, Datasets and Transformations for Images. https://torchvision.mlverse.org, https://github.com/mlverse/torchvision.

[14] Kingma, Diederik P., and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. 2014. Web.

[15] Szegedy, Christian. 'Going Deeper with Convolutions'. 2014. Web.

[16] https://www.pnas.org/doi/10.1073/pnas.2014564118

[17] https://github.com/ultralytics/yolov5

[18] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, & Matthias Grundmann (2019). MediaPipe: A Framework for Building Perception Pipelines. *CoRR, abs/1906.08172.*

[19] https://pypi.org/project/face-recognition/

Annex-A

Andrew Lemke assisted in writing proposal, draft, and final paper, including abstract and introduction. Developed approach 1, the two-stage model from scratch. This entailed evaluating face detection schemes, writing custom image processing code, and developing and training several CNNs from scratch before evaluating them to compare performance.

William Collins focused on preprocessing the various image sets to achieve a standardized form suitable for input to machine learning models. This involved applying bounding box crops, resizing images, standardizing image formats, implementing quality gates, transforming the image data to tensors and standardizing them with respect to the ImageNet mean and standard deviation. He then developed training and testing methods for fine tuning torchvision pretrained models and calculating metrics on the test results.

Ashish Ernest worked on approach 3 which involved training the Faster R-CNN algorithm utilizing the ResNet34 architecture, and fine-tuning the VGG-19 network while using the Viola-Jones algorithm for image pre-processing. Trained and tested both models on the datasets which were mentioned at the beginning of this report. Evaluated the classifiers using the metrics mentioned above.

Danya Gordin assisted in writing the proposal, draft, and final paper including the abstract, introduction, and conclusion. He also developed and wrote Approach 4 which involved pre-processing images/annotations from the WWMR and FMD datasets, training the YOLOv5 model on the combined dataset, validating the model with test images/videos, and evaluating performance metrics using WanB and other tools. He also helped put together and record the powerpoint video presentation.