Definition

Project Overview

In this project we will deal with object detection, one of the areas of computer vision that has evolved a lot with the advent of deep learning. Our goal is to build an object detection system that recognizes whether a person is wearing a mask or not and whether the mask is properly worn.

Problem Domain

Computer vision is an area of computer science that allows computers to extract information from digital images, videos, and visual inputs. If artificial intelligence allows computers to think, then that power added to that of computer vision will allow computers to make decisions based on what they see.

There were several approaches in the area of object detection, from the use of primitive shapes such as squares, circles etc… to form figures, to the systems we have today that use large neural networks to perform inferences, but one that stood out a lot in academia industry was the Yolo framework, a high-level object detection framework that abstracts the enormous complexity of building an object detection system and allows developers to focus on data quality which is often an issue.

Project Origin and Dataset

The origin of this project came from Kaggle, a platform dedicated to data science competitions and has a huge repertoire of datasets, and among these datasets one that stood out a lot in recent times was Face Mask Detection, due to the situation current in the world.

Project Statement

Our goal is to detect people and know if they are wearing a mask correctly, incorrectly or not wearing a mask, regardless of the number of people or the environment, the algorithm must be able to detect a face and then classify it for mask use.

For this, we will use our proposed dataset and train a pre-trained network, so that it is possible to achieve good results despite the small amount of data, in the end we expect the

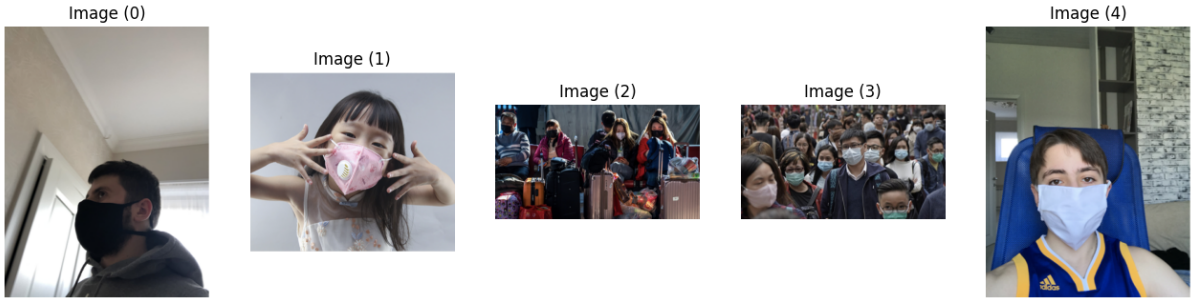
model to be able to recognize the face of anyone in any environment. and classify it with high accuracy since the dataset is of good quality.

Metrics

The metric that will be used to measure the performance of our problem is the COCO-style mAP, it is the most used metric in object detection problems, since we need to consider the category and location of the object in question.

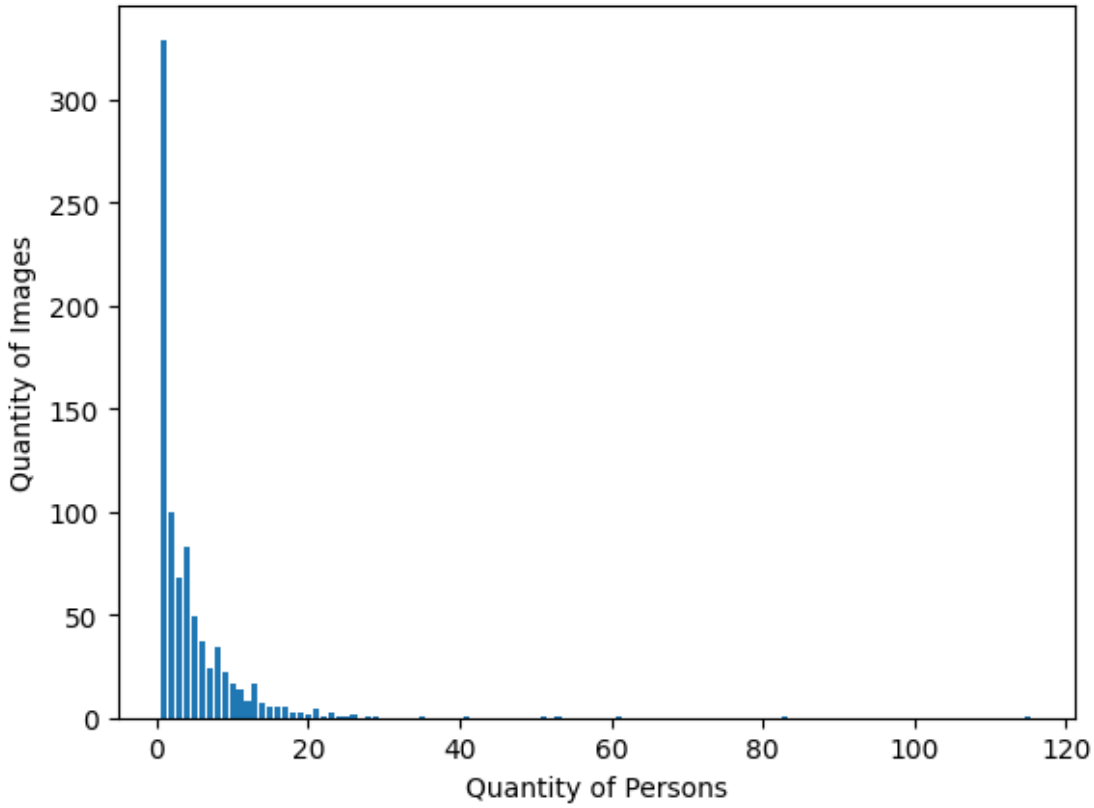
Analysis

Data Exploration and Visualization



We can see that we have a diversity of image types, people alone, from the front, from the side, crowds, selfies, this is a good sign because it indicates that in a real world situation our model can generalize well once the data is fine distributed.

Our dataset contains a total of 853 images, and the presence of 4072 people in these images, where they are distributed in 717 without a mask, 3232 with a mask and 123 with masks dressed incorrectly, a considerable imbalance that can hinder the classification of images dressed incorrectly.



Also, looking at the number of people per image, we can see that 329 images contain only one person, most images have between 0 and 10 people and few images contain more than 20 people or crowds.

Image with the largest number of people.

Algorithms and Techniques

Several algorithms and techniques are used during the problem, from data extraction to metrics calculation.

A very important library that we will use is **Pillow**, it is an open source Python library to open, manipulate and save images in different formats, as we need to open the image and convert it to an array so that the computer can interpret it, it will be very important in that project.

We will also use the **XML** library, since the information of each image such as position and class is in an XML file, we need this library to be able to extract this information.

We will also use Pytorch, the most important within our libraries, it is a framework that allows training neural networks at a high level and contains several tools for data manipulation, which we will use in most of the project.

Benchmark

As a benchmark we used some work done on top of our same dataset, the project in question was the **Mask and social distancing detection using VGG19**, the pre-trained VGG19 network was used, a much more complex network than the one we will use.

Unfortunately, the only metric used by most of the experiments was accuracy, just checking if the class of the predicted is the same as the original, however it is also important to know if the position of the predicted bounding box also matches the original, it is necessary to have the location and category relationship.

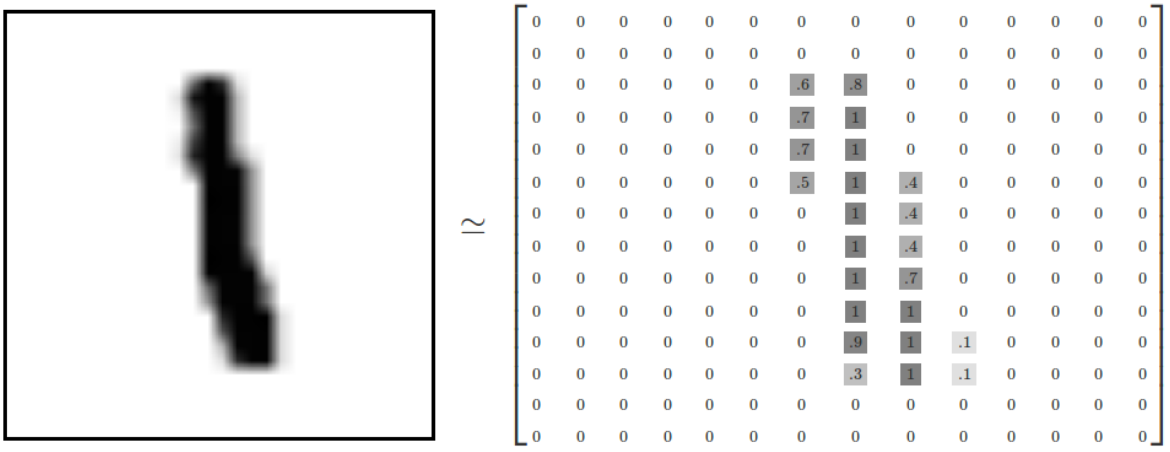
As our goal is a model that can reliably underclass classes and at low latency, we want the model to have an accuracy above 80% and an inference time below 0.5s on CPU.

Implement

Data processing

As we are dealing with images, it is necessary to do some preprocessing before we train the model, as the model expects to receive an array of numbers that represent the images.

First we read the image using Pillow and convert it to rgb, then we use pytorch's Transforms module to transform the image to a tensor.



During training we also perform data augmentation, which is a technique in which we apply transformations to the images so that the model sees different views of the image and can generalize better, this is usually a technique that brings great improvements in the model inference, in our case only we do a horizontal flip.

We also need to convert our XML files to a format that the model understands, as it contains the position information of faces and label. We use Python's XML library, extract this information which is unstructured and convert it into a python dictionary format.

Implementation

First we start implementing the Dataloader, it is the class that will read and transform the data, this uses all the techniques we discussed in the pre-processing part to transform an image into an array of numbers for the model. Then we define which model we are going to use, we instantiate the pre-trained mobilenet_v2 model, and we also instantiate an AnchorGenerator, The job of the anchor generator is to create (or load) a collection. of bounding boxes to be used as anchors, after we put everything together and instantiate our FastRCNN model

In the main scope of the model we put all these helper methods together, first reading the dataset, and dividing it into training and testing, we took a subset of 50 images to be the test dataset.

After instantiating the data loader, we load our pre-trained model and set Stochastic gradient descent as loss function and learning rate, momentum and weight decay as hyperparameters.

After that, we configure the number of epochs and at each epoch the model is trained, has its weights adjusted and is evaluated using the chosen metrics.

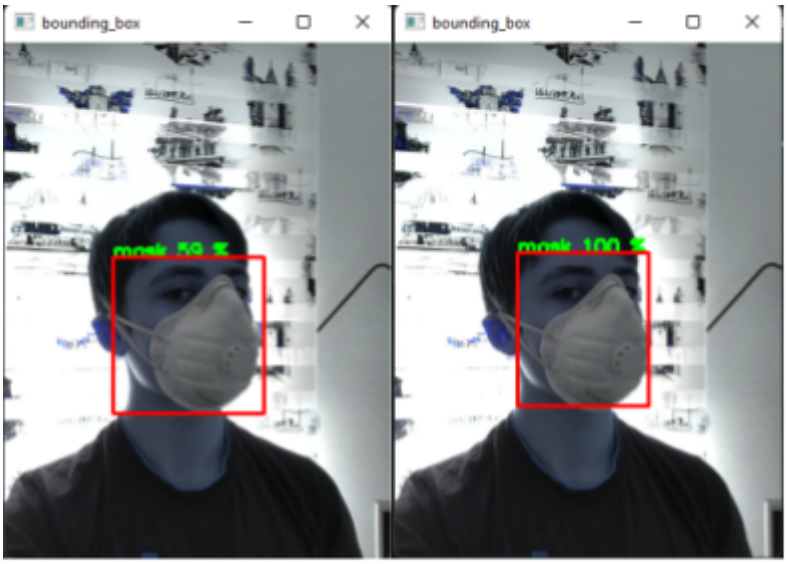
Refinement

Our initial model used the pre-trained mobilnet_v2 network, it is a small network and made for inference on edge devices, it is excellent for its fast response time and inference with good results, but if the objective is not fast inference and excellent accuracy, it is more ideal

to use more robust networks, we chose to use mobilenet_v3_large, which is from the mobilenet family, so it will still have a quick inference, but because it has a greater number of parameters, it will also have a gain in accuracy.

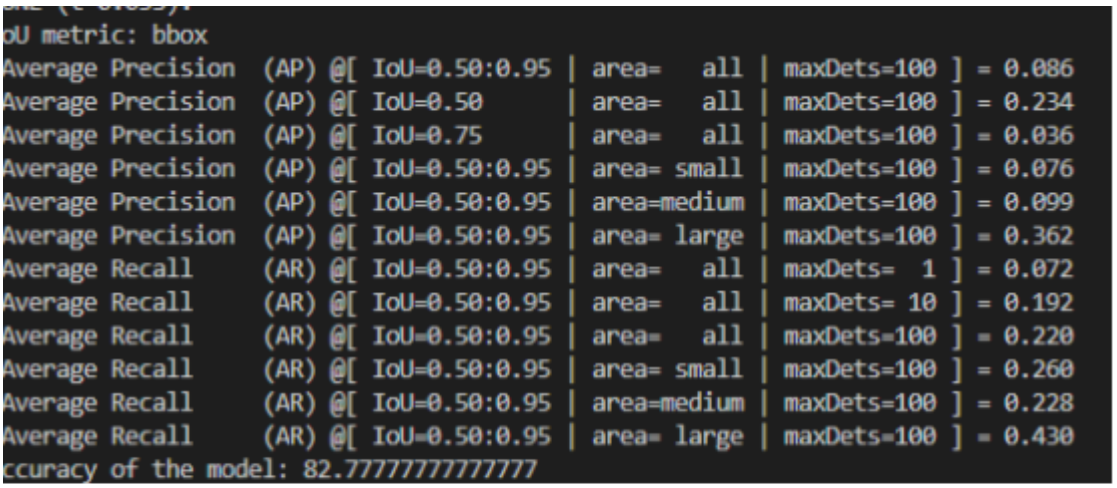
We also changed our optimizer, despite the SGD achieving good results, in academic and real world studies it has already been shown that the Adam optimizer achieves even better results, so it was our choice in refinement.

We can compare the first model with the second and see that confidence has increased from 59% to 100%, a significant increase.

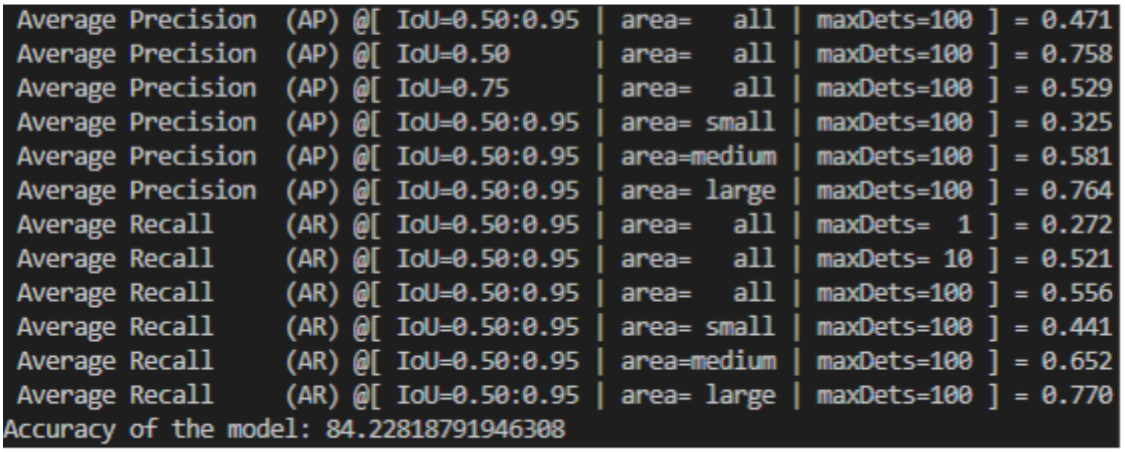


mobilenet_v2 mobilenet_v3_large

We can also look at how the metrics improved after refinement, we had an Average Precision of 0.086 and Average Recall of 0.072, after refinement we had an Average precision of 0.471 and an average recall of 0.272, which is also a significant increase in model quality.



before refinement



after refinement

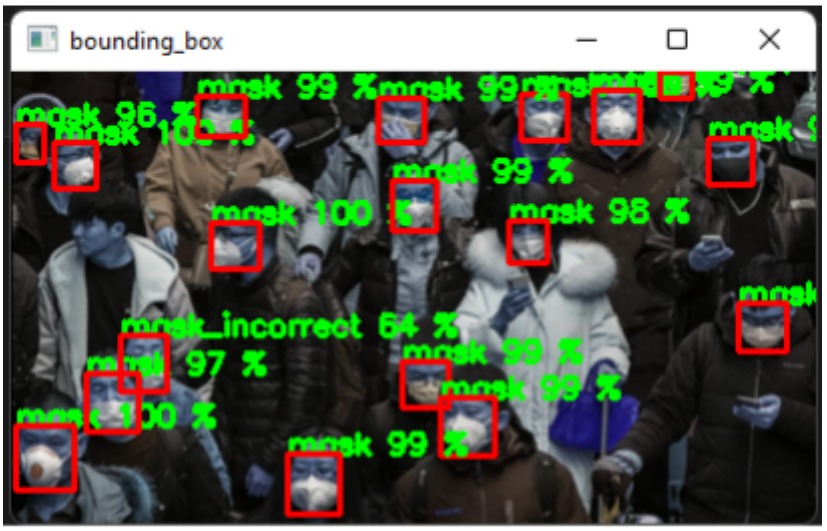
Results

Model Evaluation and Validation

After 30 epochs it was possible to find a COCO-style mAP of 47.1%, and an accuracy of 84.2%, this is a good value considering that we are using a model whose focus is to obtain good results with a low latency, and we consider both categories and locality.

We can assess the power of the model in crowd images, unlike the first model that recognized with low confidence and didn't catch many cases, the final model is able to recognize with high confidence even parts of faces that are hidden.

model 1



model 2

Conclusion

Justification

Analyzing the model that is being compared to ours, 98.43% accuracy was achieved in training and 98% accuracy in test, training the model in 20 epochs.

```
Epoch 20/20
9/9 [==============================] - 3s 306ms/step - loss: 0.0591 - accuracy: 0.9843
```

```
model.evaluate_generator(test_generator)
```

```
[0.06896067410707474, 0.9800000190734863]
```

Our model had an accuracy of 84.2%, training the model in 10 epochs, it may seem like a bad result, however we use a network with fewer parameters thinking about the problem of inferring in real time with low latency, it is a problem that often occurs called speed-accuracy trade off.

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.471
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.758
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.529
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.325
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.581
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.764
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.272
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.521
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.556
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.441
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.652
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.770
Accuracy of the model: 84.22818791946308
```

To calculate the inference time, we instantiate a dataset with 200 images and obtain the following statistics:

```
model.eval()
device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
from time import time
start = time()
with torch.no_grad():
    for data, target in dataset_test:
            prediction = model([data.to(device)])
end = time()
print("Dataset size", len(dataset_test))
print("Time: ", end - start)
print("Inferece time per Image: ", (end - start)/len(dataset_test))
```

✓ ✓ 11.9s

```
Dataset size 200
Time:  11.918375730514526
Inferece time per Image:  0.05959187865257263
```

While our desired accuracy slightly surpassed our established threshold, our desired latency fared much better than expected, showing that the solution would fit well in a real-time system with low latency.