Final Report

Face Mask Detection with Faster RCNN

Yiming Lai

Problem statement

2020 was an unforgettable year in human history. More than 3 million of lives were lost due to Covid-19. Researchers and scientists are doing everything they can to stop the spread of the virus. One of the easy yet the most important advices from the experts is to wear the face mask and keep social distance in public. In the Bay Area, almost all the buildings, stores or malls now require people wearing the face masks before entering. It is thus convenient to have an automatic model that can detect if people are following the rules or not. In this project, we will build a deep learning model based on Faster RCNN that can do the following (1) detect human faces and draw bounding boxes around the faces (2) identify if those detected people are wearing face mask or not

Data preparation

The dataset contains 853 images. Each image has different number of people in it and they may or may not wear the face masks. Furthermore, some people may not wear the face masks correctly (not covering both the nose and mouth). The figure below shows one of the images from the dataset.

For this project, we use the Faster RCNN and pytorch for the training and the prediction.

In order to feed the dataset into pytorch and train the model, we need prepare the dataset into pytorch pytorch friendly format (the tensor). The workflow for the preparation is listed below:

- The annotation files are in the xml format and we use the BeautifulSoup package to extract the information of the labels and the bounding boxes
- We need at least three features from the annotation files, “boxes”, “labels” and ‘image_id”

```
with open(os.path.join(root,'annotations',annot[idx])) as f:
    data = f.read()
    soup = BeautifulSoup(data,'xml')
    objects = soup.find_all('object')
    num_obj = len(objects)

    # initialize the bboxes and labels
    bboxes = []
    labels = []

    for obj in objects:
        # find the bounding boxes
        xmin = int(obj.find('xmin').text)
        ymin = int(obj.find('ymin').text)
        xmax = int(obj.find('xmax').text)
        ymax = int(obj.find('ymax').text)
        bboxes.append([xmin, ymin, xmax, ymax])

        # define the lables
        if obj.find('name').text == 'with_mask':
            labels.append(1)
        elif obj.find('name').text == 'mask_weared_incorrect':
            labels.append(2)
        else:
            labels.append(3)

    # convert the bboxes and labels into tensor format
    bboxes = torch.tensor(bboxes,dtype=torch.float32)
    labels = torch.tensor(labels,dtype=torch.int64)

    # create the target dictionary for __getitem__ method later in the dataset class
    target = {}
    target['boxes'] = bboxes
    target['labels'] = labels
    target['image_id'] = torch.tensor(idx,dtype=torch.int64)

    return target
```

- Create the customized dataset using pytorch dataset class. The class needs at least two functions, “__getitem__” and “__len__” besides the __init__ function

```
class MaskDataSet(Dataset):
    "Face mask dataset"
    def __init__(self, transforms=None):
        # list of the images
        self.imgs = imgs

        # transform function to be defined later
        self.transforms = transforms

    def __getitem__(self, idx):

        target = get_target(idx)
        img = Image.open(os.path.join(root,'images',imgs[idx])).convert('RGB')

        # normalize the image and convert it to tensor format
        if self.transforms is not None:
            img = self.transforms(img)

        return img, target

    def __len__(self): # this is the necessary method for DataLoader to get the number of samples
        return len(self.imgs)
```

Some additional notes for the customized dataset. Here, we use the pytorch ToTensor() transform to normalized the images. Furthermore, since each image has different dimension in our dataset and each image may have different numbers of objects, we need to use the collate function to organize the batch so that it can be read from DataLoader.

After data preparation, we can simply feed the dataset into the DataLoader for the following batch training. Here we split the images in the 803 training set and 50 test set.

Model training and validation

The Faster RCNN model can be easily created in pytorch by the following code. Here we use the pre-trained ResNet50 as the backbone for the Faster RCNN

```
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
```

```
def get_model_instance_segmentation(num_classes):
    # load an instance segmentation model pre-trained pre-trained on COCO
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

    # get number of input features for the classifier
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    # replace the pre-trained head with a new one
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    return model
```

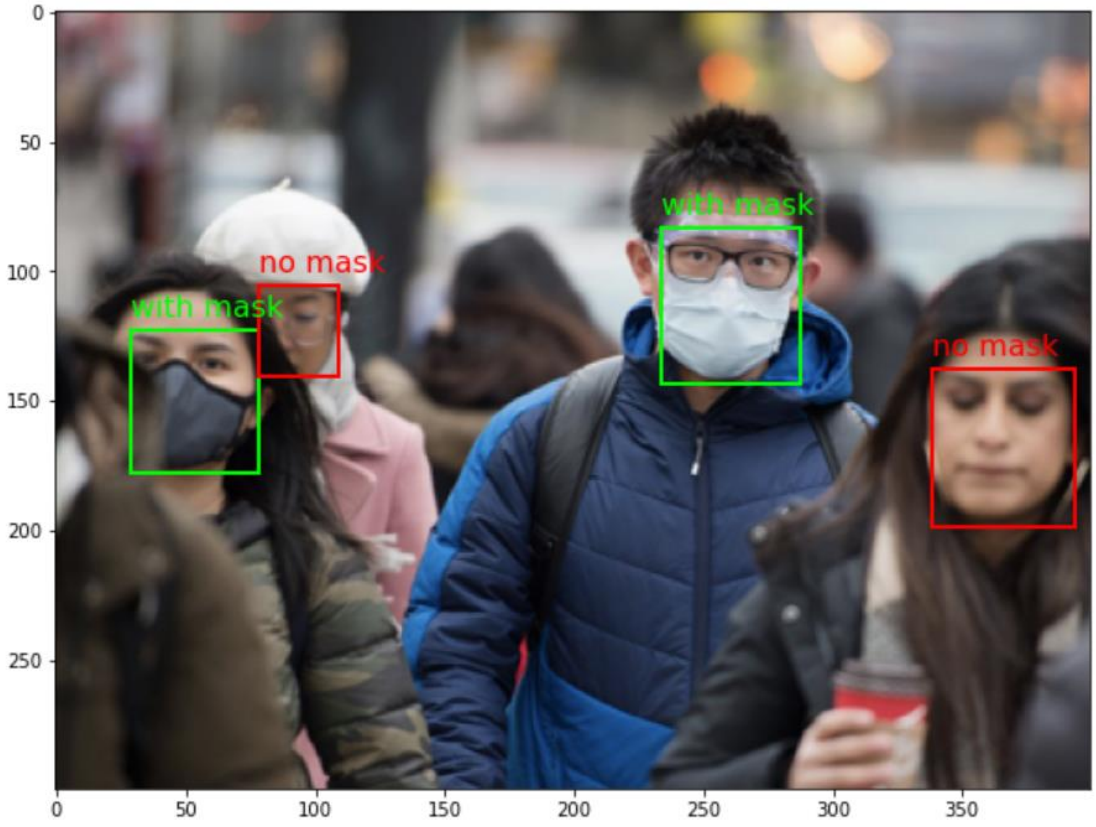
```
model = get_model_instance_segmentation(4)
```

We use mAP (mean average precision) as the metrics for the model performance. The table below shows the mAP for our model on the test set after 30 epochs of training

class	mAP(0.5)	mAP(0.55)	mAP(0.6)	mAP(0.65)	mAP(0.7)	mAP(0.75)	mAP(0.8)	mAP(0.85)	mAP(0.9)	mAP(0.95)	mAP(0.50:0.95)
mask weared incorrect	0.600	0.600	0.600	0.600	0.600	0.600	0.514	0.375	0.042	0.000	0.453
with mask	0.864	0.837	0.815	0.765	0.647	0.524	0.267	0.133	0.049	0.001	0.490
without mask	0.765	0.765	0.671	0.565	0.538	0.415	0.260	0.091	0.011	0.000	0.408

After training, we obtain a mAP of 45.3 for the mask weared incorrect class, 49.0 for with mask class and 40.8 for without mask class.

The figure below shows an example of the test set and the prediction of our model. We can see that our model can detect the face and classify the class correctly.



Future work

After further examination, we notice some common error for our model (1) mis-classify the mask-weared-incorrect class as the with-mask class (2) mis-detect the side face as the full face. To further improve the model performance, we can collect more data for different kinds of situations.