YOLO v1 FACE MASK DETECTOR

Draft design

Code: https://github.com/zhukson/YOLO-Face-mask-detector

1 Introduction

To ensure every student and teacher have their masks on, a face mask detector is very useful. In this project, I applied one of the most famous object detection models called YOLO (You only look once) for this task. The main idea of YOLO is very straight forward, which is to use the entire image as the input of a 24 layers' Convolutional Neural networks Combined with Linear layers, and the model would directly return the position information of the bounding box and the category which the object within the bounding box belongs to. Since it is a CNN model, the speed of YOLO V1 is very fast. It can process 45 images and predict the according bounding boxes of face masks in only one seconds. Therefore, it satisfied the requirement of 'Real time'.

2 the choice of model

Since there are many good and well performing object detection models, I compared two famous models with each other to see which one is better for this task.

2.1 YOLO V1

The model of yolo v1 is constructed of 24 convolutional layers and 2 fully connection layers. Where convolutional layers is used to extract features, max-pooling layers is used to lower the parameters for the purpose of preventing overfitting and fully connection layers is used to make prediction. The activation function Leaky Relu is combined with each convolutional layer. Different with the Relu activation function, which is to set all negative values into zero, Leaky ReLu multiplied all negative values with a small positive constant $Leaky \in (0, 1]$. By doing so, partial information of the negative values could be preserved. The input of YOLO model is an image with size 3(RGB 3channels) *48*48 and the output are a Tensor with size 30*7*7.

Where 7*7 means the model divide the input image into 7*7=49 grid cell, each grid cell generates 2 bounding boxes. The 30 dimension is composed of 3 parts (5+5+20), where the first two 5 dimension represents the x, y, width, height, and the value of confidence (1 represents there is an object in this grid cell, 0 otherwise). The last 20 dimensions represents the classes. This is achieved by the word-embedding technique one-hot encoding.

In this task, there are three classes in total (wear mask, improperly wear mask, no mask), therefore, the output should be 7*7*(5+5+3).

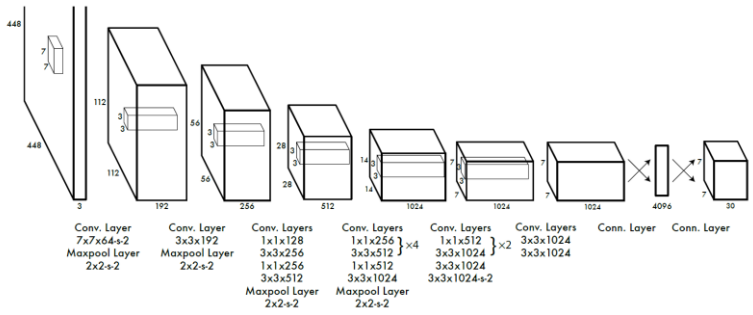

Figure 1: YOLO V1 model

2.2 Faster RCNN:

The main procedure is as follows:

1. put the whole images into CNN model to get the feature maps.

2. feature maps go into rpn and generate anchors with number of (width x height x channel x k). There are two layers in rpn, one is the classification layer and the other is regression layer. Classification layer apply softmax function to get the confidence of positive or negative of anchors. Then the anchors with higher confidence of positive would go into regression layer, and the regression layer would apply bounding regression to the positive anchors and get the proposals.

3.the Proposals and feature maps we got in step 1 would then go into the r-cnn head together. First, they would go into the roi pooling in r-cnn head which would return proposal feature maps based on Proposals and feature maps.

4.Finally, proposal feature maps go into the last two branches of r-cnn head. One is the classification branch, which return the confidence of classes based on fully connected layer + softmax. The other one is the Regression branch which apply bounding box again to get the better bounding box. Here only the bounding Box with the highest confidence would be reserved.

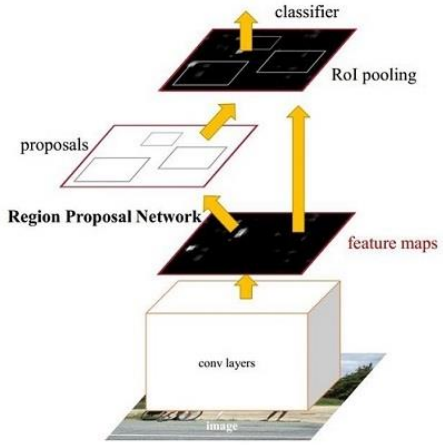

Figure 2: Faster RCNN model

2.3 Conclusion

We can see Faster RCNN is a two-stage model, and its core idea is proposal + classification. Undeniably, this two-stage model have better classification accuracy than YOLO V1. Also, compared with YOLO V1, Faster RCNN performs better when dealing with pictures with many objects adjacent to each other. This is because For YOLO V1, since each grid cell could only generate two bounding boxes and both only responsible for the same single class, if there are two classes of objects appear in the same grid cell, YOLO V1 would fails to correctly produce bounding boxes for both of them.

The disadvantage of Faster RCNN is also obvious, YOLO V1 could process 45 images in one second and satisfied the Real time requirement. However, Faster RCNN could only process 10 images in one second. For this task, detecting whether a student wearing face mask or not doesn't require a very high precision, also the object for each image should not be very much. Therefore, the real time advantage of YOLO V1 should count more than the other two advantages of Faster RCNN.

3 implementation

All the following implementations could be found in the GitHub repository with both the Jupiter notebook version and python project version.

3.1 data preprocessing

Data preprocessing is combined with two parts: image and annotation.

3.1.0 image preprocessing

Here I resized image from the original size into the required input size 448 of YOLO V1 network and do normalization to it by rescale every pixel value into range (0,1). Then save these images into a list.

```
def load_image(self, image_path):
    img = cv2.imread(image_path)    # read image
    img = cv2.resize(img, (self.target_image_size, self.target_image_size))
    return img/255

def image_prepare(self):
    data = os.listdir(self.data_path)
    data_set = []
    data.sort()
    for filename in data:
        img = self.load_image(os.path.join(self.data_path, filename))
        data_set.append(img)
    return data_set
```

3.1.1 annotation preprocessing

The ground truth bounding boxes information is stored in xml files. Therefore, we need to extract them from xml files. The procedure is as follows: **1.** Using ET.parse() to iterate over the label information contained in xml files. **2.**resize the vertex coordinate (This is because the whole image has been resized to 448, the coordinates should also be resized). **3.**calculate the center, width and height of each image based on the new coordinates. **4.** determine which grid cell the center locate at. **5.**create the label list follow the shape of the output of yolo v1(grid cell x, grid cell y, 5*number of boxes + number of classes). The box information follows the shape of (x, y, width, height, confidence value), where x,y is the coordinate of center, confidence value is the confidence whether there is an object in the box. Here what I did is to set the confidence to 1 if an bounding box has been viewed, otherwise it remains 0.

```
    def load_annotation(self, annotation_obj):
        data = np.array(os.listdir(self.data_path))
        original_data_size = cv2.imread(os.path.join(self.data_path, data[0])).shape
        height_ratio = original_data_size[0] / self.target_image_size
        width_ratio = original_data_size[1] / self.target_image_size
        # the label for training, the shape is (cell_size, cell_size, )
        label = np.zeros((self.cell_size, self.cell_size,
                          5 + self.num_cls))  # 1 confidence value + 4 position info + number of classes
        annotation_path = os.path.join(self.annotation_path, annotation_obj)
        tree = ET.parse(annotation_path)
        objs = tree.findall('object')
        for obj in objs:
            bbox = obj.find('bndbox')
            x_min = float(bbox.find('xmin').text) * width_ratio
            x_max = float(bbox.find('xmax').text) * width_ratio
            y_min = float(bbox.find('ymin').text) * height_ratio
            y_max = float(bbox.find('ymax').text) * height_ratio
            # calculate the center of this image
            x_center = (x_min + x_max) / 2
            y_center = (y_min + y_max) / 2
            width = x_max - x_min
            height = y_max - y_min
            # to determine which grid cell the center point locate at
            x_cell = int(x_center * self.cell_size / self.target_image_size)
            y_cell = int(y_center * self.cell_size / self.target_image_size)
            # if the center is out of bound, set the limit
            if x_cell > 6:
                x_cell = 6
            if y_cell > 6:
                y_cell = 6
            if label[x_cell, y_cell, 4] == 1:
                continue
            label[x_cell, y_cell, 4] = 1
            bndbox_new = [x_center, y_center, width, height]
            label[x_cell, y_cell, 0:4] = bndbox_new
            # we have three classes in total, with_mask, without_mask and mask_weared_incorrect.
            cls2index = {'with_mask': 1, 'without_mask': 2,
                         'mask_weared_incorrect': 3}
            # find the according class for the bounding box
            cls_info = cls2index[obj.find('name').text.lower().strip()]
            label[x_cell, y_cell, 5 + cls_info - 1] = 1
        return label
```

3.1.2 Face Mask dataset

Face Mask dataset extends the DataSet class from torch.utils.data. Since I plan to use pytorch for this project, I need to use the DataLoader to load the data. This dataset is the preparation for the DataLoader.

```
from torch.utils.data import Dataset
import torchvision


class mask_data_set(Dataset):
    def __init__(self, is_train=True):
        if is_train:
            self.dataset = data_preprocessing('/content/images_train','/content/annotations_train')
            self.image, self.label = self.dataset.one_stop()
        else:
            self.dataset = data_preprocessing('/content/images_val','/content/annotations_val')
            self.image, self.label = self.dataset.one_stop()

    def __len__(self):
        return len(self.image)

    def __getitem__(self, item):
        return torchvision.transforms.ToTensor()(self.image[item]), torchvision.transforms.ToTensor()(self.label[item])
```

3.2 LOSS function

The LOSS function is constructed by 5 parts:

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2]$$ Center error

$$+\quad\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right]$$ width and height error

$$+\quad\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2$$ obj Condifence error

$$+\quad\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}0_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2$$ noobj Condifence error

$$+\quad\sum_{i=0}^{S^2}1_{i}^{obj}\sum_{c\epsilon classes}(p_i(c)-\hat{p}_i(c))^2$$ classification error

3.2.0 IOU calculation

IoU calculates the ratio of the intersection and union of the "predicted bounding box" and the "real bounding box". Since for each grid cell, there are two bounding boxes generated and there will only be one box responsible for the task. Therefore, IOU is the solution to determine which box should take this responsibility. The formula is as follows:

$$\frac{area(Predicted) \cap area(GroundTruth)}{area(Predicted) \cup area(GroundTruth)}$$

The implementation in python:

```
def calculate_IOU(self, box1,box2):
    left_x1 = box1[0]-box1[3]/2
    left_y1 = box1[1]+box1[2]/2
    right_x1 = box1[0]+box1[3]/2
    right_y1 = box1[1]-box1[2]/2
    rec1 = [left_x1, left_y1, right_x1, right_y1]
    left_x2 = box2[0]-box2[3]/2
    left_y2 = box2[1]+box2[2]/2
    right_x2 = box2[0]+box2[3]/2
    right_y2 = box2[1]-box2[2]/2
    rec2 = [left_x2, left_y2, right_x2, right_y2]
    left_max = max(rec1[0],rec2[0])
    top_max = max(rec1[1],rec2[1])
    right_min = min(rec1[2],rec2[2])
    bottom_min = min(rec1[3],rec2[3])
    #if box1 and box2 are intersecting each other
    if (left_max < right_min or bottom_min > top_max):
        rect1_area = (rec1[2]-rec1[0])*(rec1[3]-rec1[1])
        rect2_area = (rec2[2]-rec2[0])*(rec2[3]-rec2[1])
        area_cross = (bottom_min - top_max)*(right_min - left_max)
        return area_cross/(rect1_area+rect2_area-area_cross)
    else:
        return 0
```

3.2.1 Center error

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2]$$

This function is to calculate the center error between the box responsible for this task and the ground truth label. where λ_{coord} is the weight, S^2 stands for the number of the grid cells, B stands for the number of boxes each grid cell generates, 1_{ij}^{obj} stands for the j bounding boxes in the i grid cells containing object.

3.2.2 width and height error

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\widehat{w_i}} \right)^2 + \left(\sqrt{h_i} - \sqrt{\widehat{h_i}} \right)^2 \right]$$

This function is to calculate the width and height error between the box responsible for this task and the ground truth label.

3.2.3 confidence error

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(C_i - \widehat{C_i} \right)^2 \quad + \quad \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 0_{ij}^{obj} \left(C_i - \widehat{C_i} \right)^2$$

The first function is to calculate the confidence error between the prediction of the box responsible for this task whereas the second one is for the box not responsible for this task(with lower IOU with the ground truth label than the other box).

3.2.4 **classification error**

$$\sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \epsilon classes} (p_i(c) - \hat{p}_i(c))^2$$

This function is to calculate the classification error between the box responsible for this task and the ground truth label. Where c is the class, and p(c) represents the confidence of whether this object belong to c.

3.2.5 **the whole procedure**

The implementation of the whole procedure is as follows:

```
# the whole process
def forward(self, pred, labels):
        # five components of the loss function
        bnd_center_loss = 0   # bounding box center loss
        bnd_wh_loss = 0   # bounding box width and height loss
        obj_confident_loss = 0   # confidence loss for bounding box with object witin
        noobj_confident_loss = 0   # confidence loss for bounding box without object within
        classification_loss = 0   # classification loss
        n_batch = labels.shape[0]
        for i in range(n_batch):
            for m in range(7):
                for n in range(7):
                    #only iterate column with confidence value =1
                    if labels[i, 4, m, n] == 1:
                        print()
                        # first we need to select the bounding box which has
                        # a higher IOU with the ground truth bounding box
                        current_pred = pred[i,:, m, n]
                        current_label = labels[i,:, m, n]
                        box1 = current_pred[0:5]
                        pred_class = current_pred[10:]
                        box2 = current_pred[5:10]
                        iou1 = self.calculate_IOU(box1, current_label)
                        iou2 = self.calculate_IOU(box2, current_label)
                        # print('box1:    {}\n, box2:  {}'.format(iou1,iou2))
                        #if box1 has a higher iou, then box1 would be responsible for this task.
                        if iou1 >= iou2:
                            bnd_center_loss += self.center_loss(box1, current_label)
                            bnd_wh_loss += self.wh_loss(box1, current_label)
                            obj_confident_loss += self.object_confidence_loss(box1, current_label)
                            noobj_confident_loss += self.noobject_confidence_loss(box2, current_label)
                            classification_loss += self.classification_loss(pred_class, current_label)
                        # otherwise, box2 would take this responsibility!
                        else:
                            bnd_center_loss += self.center_loss(box2, current_label)
                            bnd_wh_loss += self.wh_loss(box2, current_label)
                            obj_confident_loss += self.object_confidence_loss(box2, current_label)
                            noobj_confident_loss += self.noobject_confidence_loss(box1, current_label)
                            classification_loss += self.classification_loss(pred_class, current_label)
        total_loss = bnd_center_loss + bnd_wh_loss + torch.sum(obj_confident_loss) + torch.sum(noobj_confident_loss) + classification_loss
        return total_loss
```

3.3 **model**

I didn't use the original yolo model because it is too time consuming to train it. What I did is to use the pretrained convolutional layers of resnet model provided by torchvision which has already been pretrained on the imagenet. Then I combined this resnet model with the last 4 convolutional layers and the rest Linear layers. When it is in training, the parameters of the pretrained resnet model would be freezed and only the parameters of the last several layers of yolo would be trained. I am not sure if it is fine to do that. The implementation of this model is showed as follows:

```python
import torchvision.models as tvmodel
import torch.nn as nn
import torch

class Model(nn.Module):
    def __init__(self):
        super(Model,self).__init__()
        resnet = tvmodel.resnet34(pretrained=True)
        resnet_out_channel = resnet.fc.in_features
        self.resnet = nn.Sequential(*list(resnet.children())[:-2])
        self.Conv_layers = nn.Sequential(
            nn.Conv2d(resnet_out_channel,1024,3,padding=1),
            nn.LeakyReLU(),
            nn.Conv2d(1024,1024,3,stride=2,padding=1),
            nn.LeakyReLU(),
            nn.Conv2d(1024, 1024, 3, padding=1),
            nn.LeakyReLU(),
            nn.Conv2d(1024, 1024, 3, padding=1),
            nn.LeakyReLU(),
        )

        self.Conn_layers = nn.Sequential(
            nn.Linear(7*7*1024,4096),
            nn.LeakyReLU(),
            nn.Linear(4096,7*7*13),
        )

    def forward(self, input):
        input = self.resnet(input)
        input = self.Conv_layers(input)
        input = input.view(input.size()[0],-1)
        input = self.Conn_layers(input)
        return input.reshape(-1, 13, 7, 7)
```

4.Conclusion

Currently, I am still dealing with the training, the result is not satisfied. If the training is done, there would only be two parts remaining for this project. One is to test and draw the predicted bounding boxes onto the images, and the second is to connect the camera demo to realize the real time effect.