Face detection using SSD networks

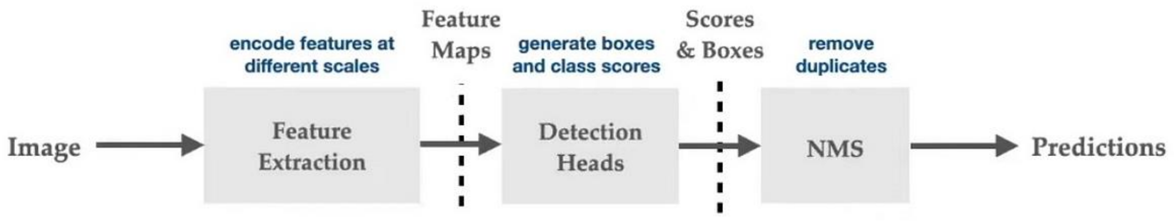
(by Ahmed Jlidi)

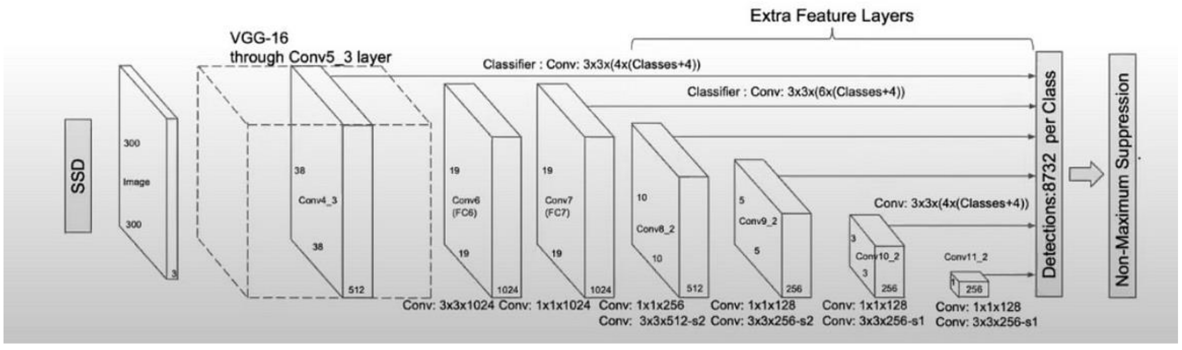
Introduction:

With the rise of deep learning, object detection models such as SSD, YOLO, R-CNN can be used for many applications. Face detection can be implemented for a variety of uses including pedestrian tracking, face recognition, mask detection (whether someone is wearing a mask or not), emotion and gender classification and much more. Source code is available: Face_detection_github

Choosing the network architecture:

For this goal, SSD architecture (Single Shot detection) is going to be used. It works by creating bounding boxes and classifying objects at the same time unlike R-CNNs. Our network is going to be **ssd300_vgg16** by torchvision models. This network uses part of the vgg16 model as its backbone. It adds an additional convolution layer for more feature extraction, the anchor generator which is used to generate bounding boxes, and the classification head to classify the resulting bounded image. More info: Pytorch ssd300_vgg16

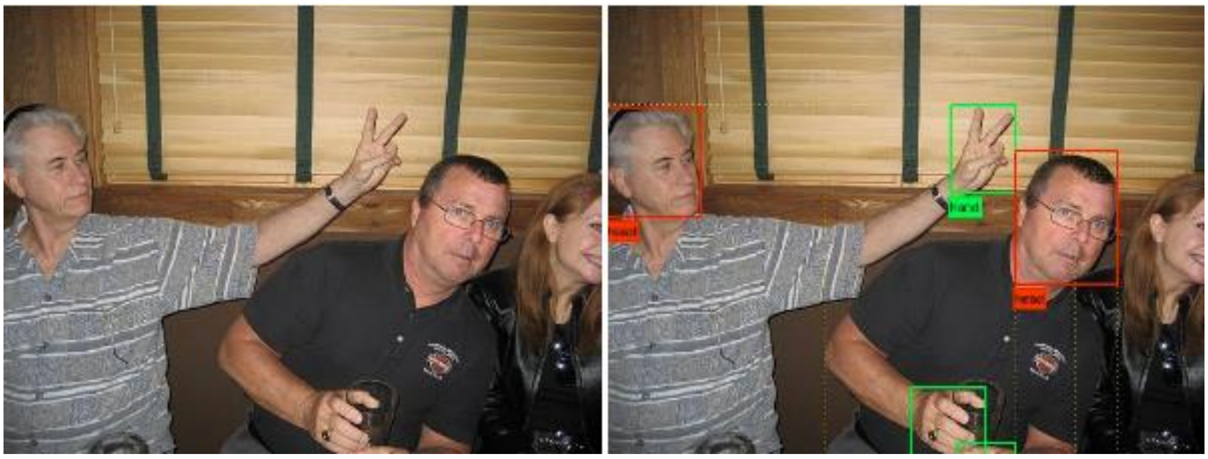




Choosing the dataset:

There are several datasets online for the purpose of face detection, but choosing the right one is important as first, the model will learn based on them. Second, the data needs to be in a compatible format for the ssd300_vgg16. The dataset should contain both images and annotations for the objects. For this, the **Voc_2011-person layout** is going to be used. Although the dataset is made to detect human parts which are (head, hands, feet), it still can be used for face detection purposes.

Oxford Pascal Voc



Configuring the model:

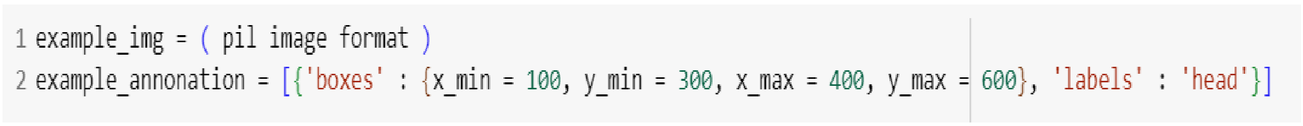
Pytorch offers a pre-trained version of ssd300_vgg16 on the coco dataset. However, it requires fine-tuning to work with the voc dataset. Specifically, the classification head needs to be modified such that it outputs 3 channels (head, hands, feet).

```python
model = torchvision.models.detection.ssd300_vgg16(pretrained=True)
def fine_tune_model(model, num_classes, size, device):
    #Changing the classification head to the number of classes
    in_channels = _utils.retrieve_out_channels(model.backbone, (size,size ))
    num_anchors = model.anchor_generator.num_anchors_per_location()
    model.head.classification_head = SSDClassificationHead(
        in_channels=in_channels,
        num_anchors=num_anchors,
        num_classes=num_classes,

    )
    model.transform.min_size = (size,)
    model.transform.max_size = size
    model = model.to(device)
    return model
model = fine_tune_model(model, 4, 300, device)
```

Pre-processing data:

The voc dataset contains multiple folders. We are interested in the Imageset and JPEGImages folders. The first contains the xml annotations for the layout and the second contains the actual images. The annotations processed using xml tree, passed to a dictionary and get the ‘parts’ key extracted to another dictionary. On the other hand, the images require no pre-processing and can be stored in a list for faster extraction.

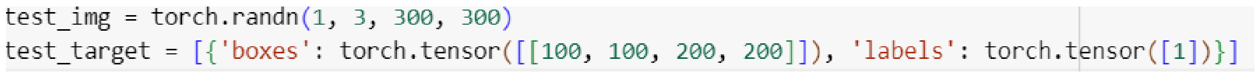


```
example_img = ( pil image format )
example_annonation = [{'boxes' : {x_min = 100, y_min = 300, x_max = 400, y_max = 600}, 'labels' : 'head'}]
```

hand, the images require no pre-processing and can be stored in a list for faster extraction.

Data augmentation:

Before passing the data to the model. The images and annotations are transformed into tensors. Additionally, images need to be resized to (Batch Num, channels, 300, 300) where Batch_num is the number of samples in batch per epoch, channels is the number of color channels, in our case 3 channels (RGB).



```
test_img = torch.randn(1, 3, 300, 300)
test_target = [{'boxes': torch.tensor([[100, 100, 200, 200]]), 'labels': torch.tensor([1])}]
```

Training set-up and Meta-parameters:

- Loss function: When model is set to train mode, The loss is calculated implicitly without the need to declare a loss function

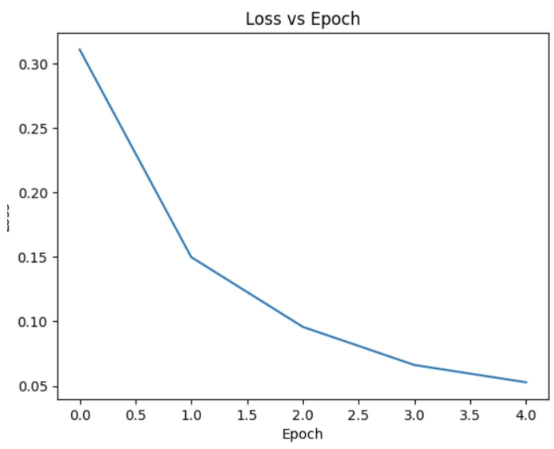
- Optimizer: Adam optimizer with learning rate of 0.0001. Choosing a high learning rate can cause exploding gradient.

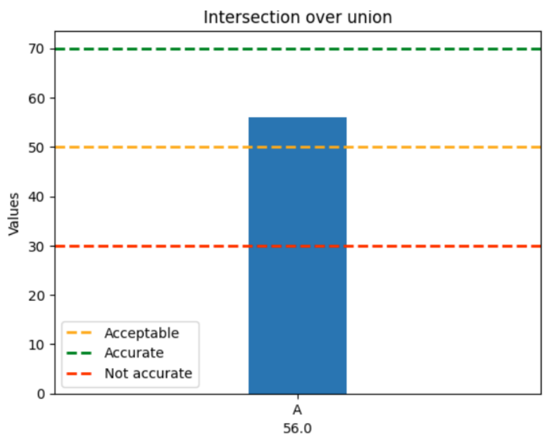
- Number of epochs: 20 epochs processing a batch of 16 samples

- Device: Nvidia T4 gpu

Validation and testing:

For accuracy evaluation, traditional methods cannot be used. Instead, an algorithm that calculates the correct area covered by the predicted bounding box will be used





Finally, we can test the model on sample images. It seems to perform well with some mistakes here and there, but overall, the performance is satisfying