Tutorial by Nooh Ayub

(SSD for Face Mask Detection Using Pretrained Model)

Objectives:

• Understand Pre-trained SSD model using different backbones (VGG16, Resnet34).

• train this pretrained model for your custom dataset (FaceMask).

• Evaluating the model on face mask test dataset and comparing the performance of different SSD backbones (VGG16, Resnet34)

Here's a step by step breakdown:

Step 1 — Create a directory named "my_ssd_implementation" and inside this directory create a virtual environment:

(a) create a conda environment with python 3.9:

```
conda create --name ssd_custom python=3.9
```

(b) now activate this environment:

```
conda activate ssd_custom
```

Step 2 — Now copy the contents of "project_ssd" inside the parent directory "my_ssd_implementation":

Note: the project folder is present in the zip file

Step 3 — Next install Pytorch 1.13.0 with cuda 11.7 support from the official Pytorch website:

```
pip install torch==1.13.0+cu117 torchvision==0.14.0+cu117 torchaudio==0.13.0 --extra-index-url https://download.pytorch.org/whl/cu117
```

Step 4 —Install the dependencies from the requirements.txt

requirements.txt.txt

```
matplotlib==3.9.3
albumentations==1.0.3
torchmetrics==0.10.3
numpy==1.26.4
tqdm==4.67.1
opencv-python
```

Use the following command when installing the dependencies:

```
pip install -r requirements.txt
```

Step 5— copy the data folder from the zip file and paste it in this parent directory "my_ssd_implementation":

(a) the dataset must be in PASCALVOC-2007 format.

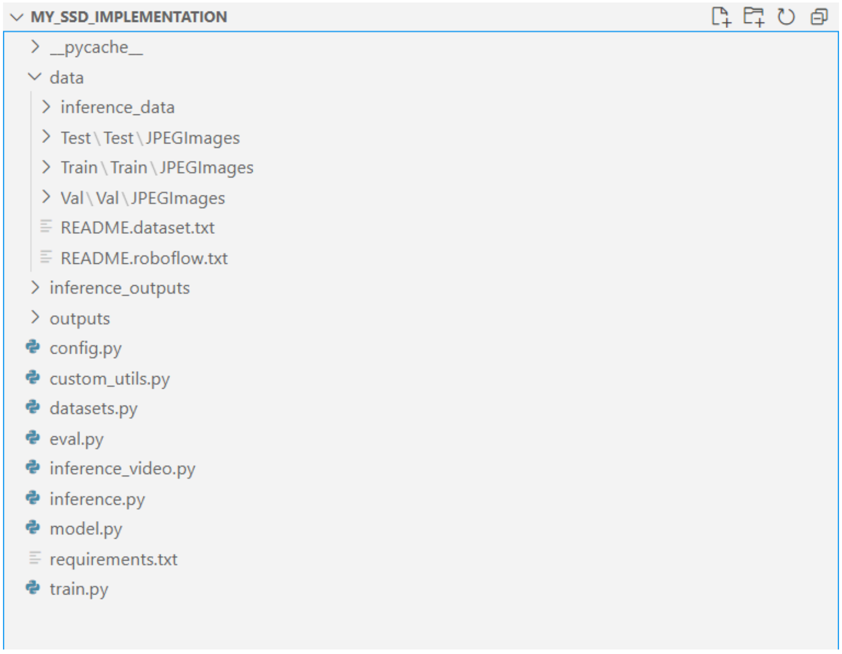
(b) The dataset tree structure is as follows :

```
├── Test
│   └── Test
│       └── JPEGImages [470 entries exceeds filelimit, not opening dir]
├── Train
│   └── Train
│       └── JPEGImages [1888 entries exceeds filelimit, not opening dir]
└── Val
    └── Val
        └── JPEGImages [320 entries exceeds filelimit, not opening dir]
```

(c) The whole Parent directory structure should look like the following:

```
├── data
│   ├── inference_data
│   │   ├── image_1.jpg
│   │   ...
│   │   ├── image_4.jpg
│   │   └── video_1.mp4
│   ├── Test
│   │   └── Test
│   │       └── JPEGImages [470 entries exceeds filelimit, not opening dir]
│   ├── Train
│   │   └── Train
│   │       └── JPEGImages [1888 entries exceeds filelimit, not opening dir]
│   └── Val
│       └── Val
│           └── JPEGImages [320 entries exceeds filelimit, not opening dir]
├── inference_outputs
│   ├── images [239 entries exceeds filelimit, not opening dir]
│   └── videos
│       └── video_1.mp4
├── notebooks
│   └── visualizations_data.ipynb
├── outputs
│   ├── best_model.pth
│   ├── last_model.pth
│   ├── map.png
│   └── train_loss.png
├── config.py
├── custom_utils.py
├── datasets.py
├── eval.py
├── inference.py
├── inference_video.py
├── model.py
└── train.py
```

(d) Final display structure of your workspace:



Step 6—Make the following changes in the config.py according to our custom dataset of Facemask:

(a) Ensure the training and validation paths are correctly set (red rectangle)

config.py ×

config.py > ...

```python
import torch

BATCH_SIZE = 8 # Increase / decrease according to GPU memeory.
RESIZE_TO = 300 # Resize the image for training and transforms.
NUM_EPOCHS = 50 # Number of epochs to train for.
NUM_WORKERS = 4 # Number of parallel workers for data loading.

DEVICE = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

# Training images and XML files directory.
TRAIN_DIR = 'C:\\Users\\ForAI\\OneDrive\\Desktop\\DL task\\my_ssd_implementation\\data\\Train\\Train\\JPEGImages'

# Validation images and XML files directory.
VALID_DIR = 'C:\\Users\\ForAI\\OneDrive\\Desktop\\DL task\\my_ssd_implementation\\data\\Val\\Val\\JPEGImages'

# # Classes: 0 index is reserved for background.
CLASSES = ['__background__', 'with_mask','without_mask','mask_weared_incorrect']


NUM_CLASSES = len(CLASSES)

# Whether to visualize images after crearing the data loaders.
VISUALIZE_TRANSFORMED_IMAGES = False

# Location to save model and plots.
OUT_DIR = 'outputs'
```

(b) In the CLASSES we will be specifying the correct labels for our custom dataset (blue rectangle)

(c) Also make sure to change the "BATCH_SIZE" according to your GPU therefore if your GPU is weak then lower the "BATCH_SIZE" to avoid further errors

Step 7(a)— Next we define the model.py using resnet34 backbone:

(a) Import libraries:

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

from torchvision.models.detection.ssd import (
    SSD,
    DefaultBoxGenerator,
    SSDHead
)
```

(b) Define the model class:

```
def create_model(num_classes=4, size=300, nms=0.45):
    model_backbone = torchvision.models.resnet34(
        weights=torchvision.models.ResNet34_Weights.DEFAULT
    )
    conv1 = model_backbone.conv1
    bn1 = model_backbone.bn1
    relu = model_backbone.relu
    max_pool = model_backbone.maxpool
    layer1 = model_backbone.layer1
    layer2 = model_backbone.layer2
    layer3 = model_backbone.layer3
    layer4 = model_backbone.layer4
    backbone = nn.Sequential(
        conv1, bn1, relu, max_pool,
        layer1, layer2, layer3, layer4
    )
    out_channels = [512, 512, 512, 512, 512, 512]
    anchor_generator = DefaultBoxGenerator(
        [[2], [2, 3], [2, 3], [2, 3], [2], [2]],
    )
    num_anchors = anchor_generator.num_anchors_per_location()
    head = SSDHead(out_channels, num_anchors, num_classes)
    model = SSD(
        backbone=backbone,
        num_classes=num_classes,
        anchor_generator=anchor_generator,
        size=(size, size),
        head=head,
        nms_thresh=nms
    )
    return model
```

Note: make sure the "num_classes" is correctly configured e.g in our case since we have 4 labels hence "num_classes=4"

(c) Next define the main method:

```
if __name__ == '__main__':
    model = create_model(4, 300)
    print(model)
    # Total parameters and trainable parameters.
    total_params = sum(p.numel() for p in model.parameters())
    print(f"{total_params:,} total parameters.")
    total_trainable_params = sum(
        p.numel() for p in model.parameters() if p.requires_grad)
    print(f"{total_trainable_params:,} training parameters.")
```

Make sure you pass the correct number of classes to the model (red rectangle)

Step 7(b)— (optional)→define the model.py using VGG16 backbone:

(a) Import Libraries

```python
import torchvision
from torchvision.models.detection.ssd import SSDClassificationHead
from torchvision.models.detection import _utils
from torchvision.models.detection import SSD300_VGG16_Weights
```

(b) Define the model class:

```python
def create_model(num_classes=4, size=300):
    # Load the Torchvision pretrained model.
    model = torchvision.models.detection.ssd300_vgg16(
        weights=SSD300_VGG16_Weights.COCO_V1
    )
    # Retrieve the list of input channels.
    in_channels = _utils.retrieve_out_channels(model.backbone, (size, size))
    # List containing number of anchors based on aspect ratios.
    num_anchors = model.anchor_generator.num_anchors_per_location()
    # The classification head.
    model.head.classification_head = SSDClassificationHead(
        in_channels=in_channels,
        num_anchors=num_anchors,
        num_classes=num_classes,
    )
    # Image size for transforms.
    model.transform.min_size = (size,)
    model.transform.max_size = size
    return model
```

Note: make sure the "num_classes" is correctly configured e.g in our case since we have 4 labels hence "num_classes=4"

(c) Next define the main method:

```python
if __name__ == '__main__':
    model = create_model(4, 640)
    print(model)
    # Total parameters and trainable parameters.
    total_params = sum(p.numel() for p in model.parameters())
    print(f"{total_params:,} total parameters.")
    total_trainable_params = sum(
        p.numel() for p in model.parameters() if p.requires_grad)
    print(f"{total_trainable_params:,} training parameters.")
```

Make sure you pass the correct number of classes to the model (red rectangle)

Step 8— Next we will be making a couple of changes in the datasets.py:

(a) Import these additional libraries in the script:

```
import matplotlib.pyplot as plt
```

(b) Replace the “visualize_sample” method with the following code:

```
# function to visualize a single sample
def visualize_sample(image, target):
    # Convert from RGB (which OpenCV uses) to BGR (which matplotlib uses).
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Plot the image
    plt.imshow(image_rgb)
    plt.axis('off')  # Hide axes

    # Add bounding boxes and labels
    for box_num in range(len(target['boxes'])):
        box = target['boxes'][box_num]
        label = CLASSES[target['labels'][box_num]]
        plt.gca().add_patch(
            plt.Rectangle(
                (box[0], box[1]),
                box[2] - box[0],
                box[3] - box[1],
                linewidth=2,
                edgecolor='r',
                facecolor='none'
            )
        )
        plt.text(
            box[0],
            box[1] - 5,
            label,
            color='r',
            fontsize=10,
            weight='bold'
        )

    plt.show()  # Display the image with annotations
```

Step 9— Now we will start the training by using the following command in the terminal:

```
\DL task\my_ssd_implementation> python train.py
```

Step 10—Saving the training losses, mAP plots & weights:

(a) After the training has completed the loss curves, mAP score plots and the best weights will be saved in the following "outputs" folder:

```
outputs
  best_model.pth
  last_model.pth
  map.png
  train_loss.png
```

Note: use the "best_model.pth" weights when evaluating the model

Step 11—Next evaluate the model (calculate mAP score) on your test dataset by running the following command in terminal:

```
\my_ssd_implementation> python eval.py
```

Result after the successful execution of eval.py:

```
(ssd_custom) PS C:\Users\ForAI\OneDrive\Desktop\DL task\my_ssd_implementation> python eval.py
Validating
100%|
mAP_50: 65.761
mAP_50_95: 39.558
```

Step 12—Now we will be making some changes in inference.py:

(a) If your dataset has 10 classes then you should have 10 color definitions but since we have 4 classes hence we have defined 4 colors as shown below in the inference.py script:

inference.py ●

inference.py > ...

if we have m classes then we will have m colors as well as shown here

```
)
parser.add_argument(
    '--imgsz',
    default=None,
    type=int,
    help='image resize shape'
)
parser.add_argument(
    '--threshold',
    default=0.25,
    type=float,
    help='detection threshold'
)
args = vars(parser.parse_args())

os.makedirs('inference_outputs/images', exist_ok=True)

#make change here must

COLORS = [[0, 0, 0], [255, 0, 0], [0,255,0], [0,0,255]]

#make change here must

# Load the best model and trained weights.
model = create_model(num_classes=NUM_CLASSES, size=300)
checkpoint = torch.load('outputs/best_model.pth', map_location=DEVICE)
```

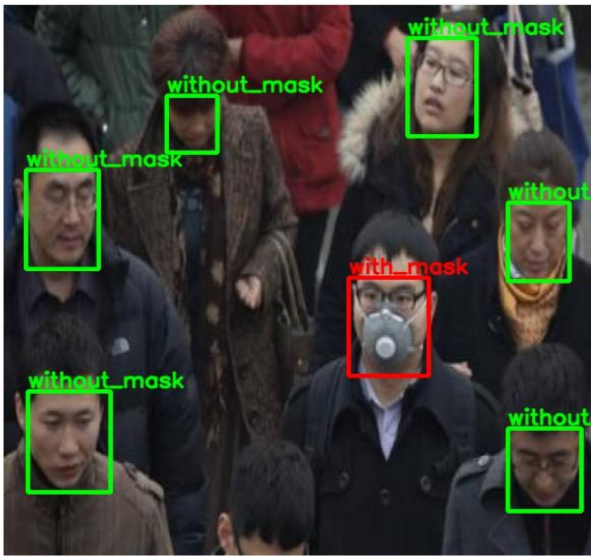
(b) Now run the inference.py by using the following command (make sure you correctly specify the test images path):

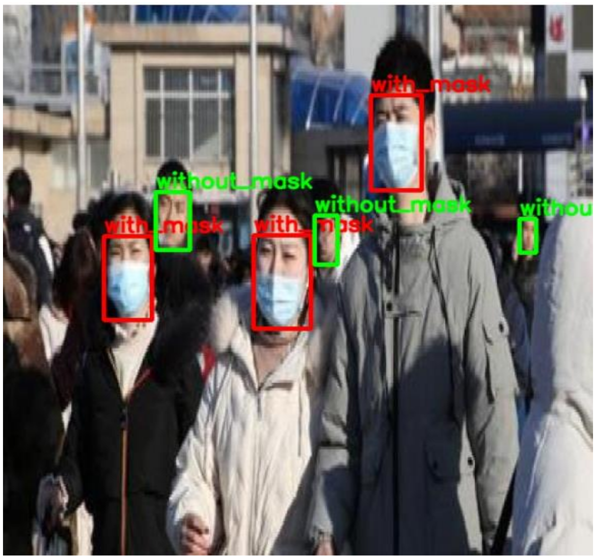
```
python inference.py --input data/Test/Test/JPEGImages/
```

(c) After successful execution of inference.py script the predictions on the test set will be saved in the following directory:

- > data
- ∨ inference_outputs\images
  - maksssksksss2_jpg.jpg
  - maksssksksss4_jpg.jpg
  - maksssksksss17_jpg.jpg
  - maksssksksss31_jpg.jpg
  - maksssksksss53_jpg.jpg
  - maksssksksss64_jpg.jpg
  - maksssksksss67_jpg.jpg
  - maksssksksss70_jpg.jpg
  - maksssksksss82_jpg.jpg
  - maksssksksss88_jpg.jpg
  - maksssksksss99_jpg.jpg
  - maksssksksss110_jpg.jpg
  - maksssksksss116_jpg.jpg
  - maksssksksss126_jpg.jpg
  - maksssksksss153_jpg.jpg

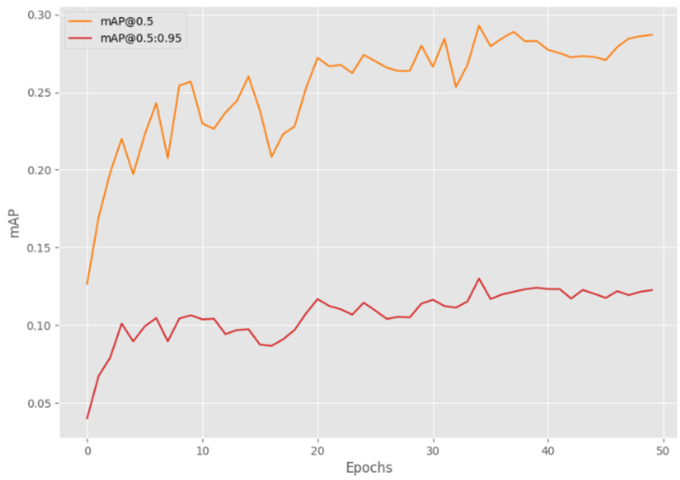
Step 13—Visualize your model predictions by viewing some sample images:

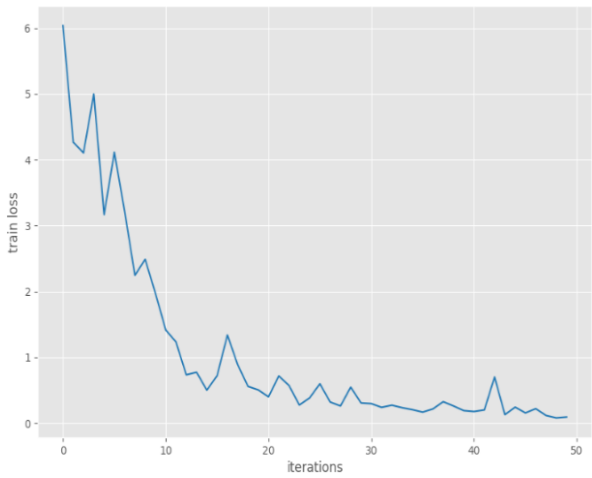




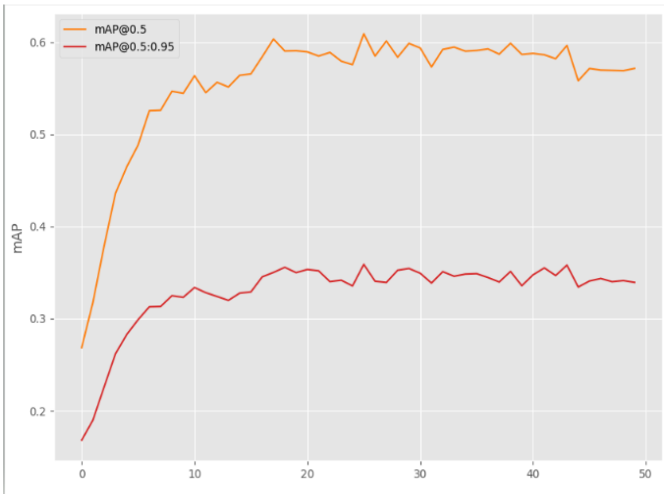
Step 14— Evaluating our SSD model with the VGG16 and resnet-34 backbones:

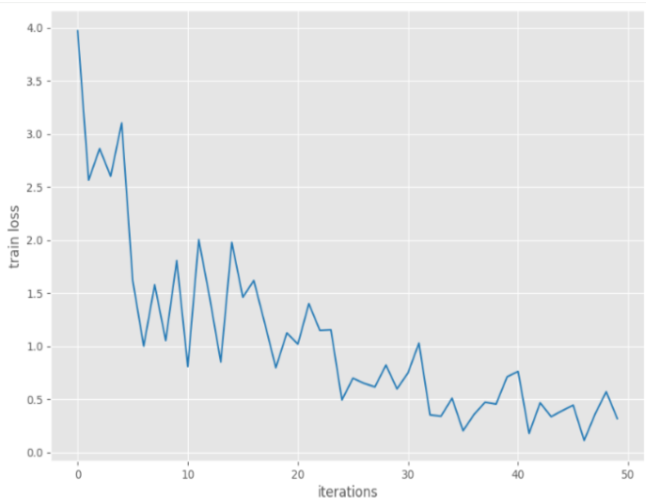
(a) After 50 epochs we got the following mAP and training loss plots for the resnet-34:





(a) Similarly, After 50 epochs we got the following mAP and training loss plots for the VGG16:





Conclusion: *VGG16 scores higher mAP (mAP=65.761) than resnet-34 on this dataset for 50 epochs*

Making Improvements to the PyTorch SSD Model with Custom Backbone

There are a few ways to make the model even better.

- *We can start by training it on more data.*
- *We can also try a larger backbone like ResNet50, ResNet101 which may prove to be a better feature extractor.*
- *Also, we can add FPN (Feature Pyramid Network) which can help the backbone a lot when dealing with small objects.*

Final Notes to avoid Potential errors:

- *When using VGG16 model or resnet-34 model make sure you name your model script as model.py and not model_resnet.py or model_vgg16.py*