```
# Cloning the Monk Object Detection repository
!git clone 'https://github.com/Tessellate-Imaging/Monk_Object_Detection.git'
# Installing requirements
! cd Monk_Object_Detection/1_gluoncv_finetune/installation && cat requirements_colab.txt | xargs -n 1 -L 1 pip install
```

```
Cloning into 'Monk_Object_Detection'...
remote: Enumerating objects: 10565, done.
remote: Counting objects: 100% (40/40), done.
remote: Compressing objects: 100% (30/30), done.
remote: Total 10565 (delta 18), reused 24 (delta 10), pack-reused 10525
Receiving objects: 100% (10565/10565), 260.88 MiB | 36.24 MiB/s, done.
Resolving deltas: 100% (4710/4710), done.
Updating files: 100% (8428/8428), done.
xargs: warning: options --max-args and -L are mutually exclusive, ignoring previous --max-args value
Collecting xmltodict
  Downloading xmltodict-0.13.0-py2.py3-none-any.whl (10.0 kB)
Installing collected packages: xmltodict
Successfully installed xmltodict-0.13.0
Collecting mxnet-cu100
  Downloading mxnet_cu100-1.9.0-py3-none-manylinux2014_x86_64.whl (354.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 354.0/354.0 MB 2.1 MB/s eta 0:00:00
Requirement already satisfied: numpy<2.0.0,>1.16.0 in /usr/local/lib/python3.10/dist-packages (from mxnet-cu100) (1.25.2)
Requirement already satisfied: requests<3,>=2.20.0 in /usr/local/lib/python3.10/dist-packages (from mxnet-cu100) (2.31.0)
Collecting graphviz<0.9.0,>=0.8.1 (from mxnet-cu100)
  Downloading graphviz-0.8.4-py2.py3-none-any.whl (16 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.20.0->mx
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.20.0->mxnet-cu100) (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.20.0->mxnet-cu
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.20.0->mxnet-cu
Installing collected packages: graphviz, mxnet-cu100
  Attempting uninstall: graphviz
    Found existing installation: graphviz 0.20.1
    Uninstalling graphviz-0.20.1:
      Successfully uninstalled graphviz-0.20.1
Successfully installed graphviz-0.8.4 mxnet-cu100-1.9.0
Collecting gluoncv
  Downloading gluoncv-0.10.5.post0-py2.py3-none-any.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 6.0 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from gluoncv) (1.25.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gluoncv) (4.66.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from gluoncv) (2.31.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from gluoncv) (3.7.1)
Collecting portalocker (from gluoncv)
  Downloading portalocker-2.8.2-py3-none-any.whl (17 kB)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from gluoncv) (9.4.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from gluoncv) (1.11.4)
Collecting yacs (from gluoncv)
  Downloading yacs-0.1.8-py3-none-any.whl (14 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from gluoncv) (1.5.3)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from gluoncv) (6.0.1)
Collecting autocfg (from gluoncv)
  Downloading autocfg-0.0.8-py3-none-any.whl (13 kB)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (from gluoncv) (4.8.0.76)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->gluoncv) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->gluoncv) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->gluoncv) (4.48.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->gluoncv) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->gluoncv) (23.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->gluoncv) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->gluoncv) (2.8.2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->gluoncv) (2023.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->gluoncv) (3.3
```

```
#fixed version of tqdm output for Colab
!pip install --force https://github.com/chengs/tqdm/archive/colab.zip
#IGNORE restart runtime warning, it is indeed installed
#missing a few extra packages that we will need later!
!pip install efficientnet_pytorch
!pip install tensorboardX
```

```
Collecting https://github.com/chengs/tqdm/archive/colab.zip
  Downloading https://github.com/chengs/tqdm/archive/colab.zip
     - 91.8 kB 1.8 MB/s 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: tqdm
  Building wheel for tqdm (setup.py) ... done
  Created wheel for tqdm: filename=tqdm-4.28.1-py2.py3-none-any.whl size=47874 sha256
  Stored in directory: /tmp/pip-ephem-wheel-cache-5rjk2rg6/wheels/65/77/d5/d5ddeac992
Successfully built tqdm
Installing collected packages: tqdm
  Attempting uninstall: tqdm
    Found existing installation: tqdm 4.66.2
    Uninstalling tqdm-4.66.2:
      Successfully uninstalled tqdm-4.66.2
ERROR: pip's dependency resolver does not currently take into account all the package
huggingface-hub 0.20.3 requires tqdm>=4.42.1, but you have tqdm 4.28.1 which is incom
panel 1.3.8 requires tqdm>=4.48.0, but you have tqdm 4.28.1 which is incompatible.
prophet 1.1.5 requires tqdm>=4.36.1, but you have tqdm 4.28.1 which is incompatible.
spacy 3.7.2 requires tqdm<5.0.0,>=4.38.0, but you have tqdm 4.28.1 which is incompati
Successfully installed tqdm-4.28.1
Collecting efficientnet_pytorch
  Downloading efficientnet_pytorch-0.7.1.tar.gz (21 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-package
Building wheels for collected packages: efficientnet_pytorch
  Building wheel for efficientnet_pytorch (setup.py) ... done
  Created wheel for efficientnet_pytorch: filename=efficientnet_pytorch-0.7.1-py3-non
  Stored in directory: /root/.cache/pip/wheels/03/3f/e9/911b1bc46869644912bda90a56bcf
Successfully built efficientnet_pytorch
Installing collected packages: efficientnet_pytorch
Successfully installed efficientnet_pytorch-0.7.1
Collecting tensorboardX
  Downloading tensorboardX-2.6.2.2-py2.py3-none-any.whl (101 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.7/101.7 kB 1.7 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: protobuf>=3.20 in /usr/local/lib/python3.10/dist-packa
Installing collected packages: tensorboardX
Successfully installed tensorboardX-2.6.2.2
```

Let's get some data!

The best part about Roboflow is the efficient management of your datasets. Upload you dataset and you will recieve a fresh curl code to ouput it in whatever augmented and annotated format you need.

```
import cv2
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle


!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="dPPsOldBngjDaeVqSjOm")
project = rf.workspace("yolo-otbw9").project("weapon-detection-o4mdd")
dataset = project.version(11).download("coco")
```

```
Collecting roboflow
  Downloading roboflow-1.1.19-py3-none-any.whl (70 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 70.2/70.2 kB 922.4 kB/s eta 0:00:00
Collecting certifi==2023.7.22 (from roboflow)
  Downloading certifi-2023.7.22-py3-none-any.whl (158 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 158.3/158.3 kB 3.6 MB/s eta 0:00:00
Collecting chardet==4.0.0 (from roboflow)
  Downloading chardet-4.0.0-py2.py3-none-any.whl (178 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 178.7/178.7 kB 12.0 MB/s eta 0:00:00
Collecting cycler==0.10.0 (from roboflow)
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
Collecting idna==2.10 (from roboflow)
  Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.8/58.8 kB 8.3 MB/s eta 0:00:00
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-packag
Collecting opencv-python-headless==4.8.0.74 (from roboflow)
  Downloading opencv_python_headless-4.8.0.74-cp37-abi3-manylinux_2_17_x86_64.manylin
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.1/49.1 MB 11.6 MB/s eta 0:00:00
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-pack
Collecting python-dotenv (from roboflow)
  Downloading python_dotenv-1.0.1-py3-none-any.whl (19 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from r
Collecting supervision (from roboflow)
  Downloading supervision-0.18.0-py3-none-any.whl (86 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 86.7/86.7 kB 9.9 MB/s eta 0:00:00
Requirement already satisfied: urllib3>=1.26.6 in /usr/local/lib/python3.10/dist-pack
Collecting tqdm>=4.41.0 (from roboflow)
  Downloading tqdm-4.66.2-py3-none-any.whl (78 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.3/78.3 kB 6.2 MB/s eta 0:00:00
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.10/dist-packag
Collecting requests-toolbelt (from roboflow)
  Downloading requests_toolbelt-1.0.0-py2.py3-none-any.whl (54 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.5/54.5 kB 7.9 MB/s eta 0:00:00
Collecting python-magic (from roboflow)
  Downloading python_magic-0.4.27-py2.py3-none-any.whl (13 kB)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
Requirement already satisfied: defusedxml<0.8.0,>=0.7.1 in /usr/local/lib/python3.10/
Requirement already satisfied: scipy<2.0.0,>=1.10.0 in /usr/local/lib/python3.10/dist
Installing collected packages: tqdm, python-magic, python-dotenv, opencv-python-headl
  Attempting uninstall: tqdm
    Found existing installation: tqdm 4.28.1
    Uninstalling tqdm-4.28.1:
      Successfully uninstalled tqdm-4.28.1
  Attempting uninstall: opencv-python-headless
    Found existing installation: opencv-python-headless 4.9.0.80
    Uninstalling opencv-python-headless-4.9.0.80:
      Successfully uninstalled opencv-python-headless-4.9.0.80
  Attempting uninstall: idna
    Found existing installation: idna 3.6
    Uninstalling idna-3.6:
      Successfully uninstalled idna-3.6
  Attempting uninstall: cycler
    Found existing installation: cycler 0.12.1
    Uninstalling cycler-0.12.1:
      Successfully uninstalled cycler-0.12.1
  Attempting uninstall: chardet
    Found existing installation: chardet 5.2.0
    Uninstalling chardet-5.2.0:
      Successfully uninstalled chardet-5.2.0
  Attempting uninstall: certifi
    Found existing installation: certifi 2024.2.2
    Uninstalling certifi-2024.2.2:
      Successfully uninstalled certifi-2024.2.2
ERROR: pip's dependency resolver does not currently take into account all the package
lida 0.0.10 requires fastapi, which is not installed.
lida 0.0.10 requires kaleido, which is not installed.
lida 0.0.10 requires python-multipart, which is not installed.
lida 0.0.10 requires uvicorn, which is not installed.
Successfully installed certifi-2023.7.22 chardet-4.0.0 cycler-0.10.0 idna-2.10 opencv
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in Weapon-Detection-11 to coco:: 100%|██████████| 302

Extracting Dataset Version Zip to Weapon-Detection-11 in coco:: 100%|██████████| 7501
```

```
#let's take a look at our directory
#notice the data came down in train, valid, test, splits - this is pre set during the dataset upload process
%ls
```

```
Monk_Object_Detection/  sample_data/  Weapon-Detection-11/
```

```
#let's take a peak in train
#jpg images and some coco json annotations
%cd Weapon-Detection-11/
```

```
/content/Weapon-Detection-11
```

```
%ls train
```

```
000000-12_jpg.rf.253ad1e0439fbd32f301df7646cc0232.jpg
000000-12_jpg.rf.2f893887b7b683655f060fc36fa8db4c.jpg
000000-12_jpg.rf.f9f2851daacdc145229e07d845cf2dbf.jpg
00-1_jpg.rf.4c74549418d4f6f83d5bfdef1f6c4d6e.jpg
00-1_jpg.rf.72ef9bd773e31c45f63919a58d8d4646.jpg
00-1_jpg.rf.c272e2f121f932996a19b595442c3664.jpg
002-44-2-_jpg.rf.2961665f6d4011d5b9849987eb2c1116.jpg
002-44-2-_jpg.rf.7b1d61cc02bf77704ee3b4e979131e70.jpg
002-44-2-_jpg.rf.f3c51db9294c51de61bd3d4635a7f983.jpg
006-2-_jpg.rf.5396fd51a1663a8c5ac5bf608c5b59c6.jpg
006-2-_jpg.rf.be2adf7580e39fa4803e8b75cd935d8a.jpg
006-2-_jpg.rf.df2102c81b5c79d043750961d49cea90.jpg
008-JPG-2-_jpg.rf.27df279e6a4e6682c8414c8f574f9796.jpg
008-JPG-2-_jpg.rf.60cc7ef189986b7bb7eb8d0125444e35.jpg
008-JPG-2-_jpg.rf.dcb1e036e6ca75d4d348a3035c957ede.jpg
0129231733c-scaled_jpg.rf.5b02f8d0a6e0932ca71c92a7fac74279.jpg
0129231733c-scaled_jpg.rf.7961926feb9864fb4103dbefd2506670.jpg
0129231733c-scaled_jpg.rf.955eed4a75f5cb2efbb1ec64eb74c22d.jpg
014917-017868_krytac-emg-fn-p90-smg-aeg_ttg_jpg.rf.268a643f07d2939aac13011970aa8897.jpg
014917-017868_krytac-emg-fn-p90-smg-aeg_ttg_jpg.rf.6125fed5f83c3d4f889cde4cf1f540ff.jpg
014917-017868_krytac-emg-fn-p90-smg-aeg_ttg_jpg.rf.d9f5ed8b3bd5b80cb3921b4d2f00f457.jpg
017553-021594_krytac-emg-fn-p90-smg-aeg-alpine-limited-edition_ttp_jpg.rf.1ec185660bd64381809de00d5c11a983.jpg
017553-021594_krytac-emg-fn-p90-smg-aeg-alpine-limited-edition_ttp_jpg.rf.8ac60a283831b2f4f2a49333f0b2b52c.jpg
017553-021594_krytac-emg-fn-p90-smg-aeg-alpine-limited-edition_ttp_jpg.rf.93272dddb351727b150d8dbe6a85af2c.jpg
0-1_jpg.rf.4431a548ee73e2415eaa298177cc8d99.jpg
0-1_jpg.rf.ce14c5a3fb0d1c7f77fed08d1157d391.jpg
0-1_jpg.rf.f88c7f3c9f6654a9a16e6af8a7ef5a08.jpg
04486641311bbfb3149c18d0ab931e9b662de738_jpg.rf.319c5d5dd5647feac8eff6ef75b9a09b.jpg
04486641311bbfb3149c18d0ab931e9b662de738_jpg.rf.acf4c7b92490d5adfc38b4eabf41677f.jpg
04486641311bbfb3149c18d0ab931e9b662de738_jpg.rf.b4bc23a4fb34d27721fe134b04d50d46.jpg
0b91cbffdf82b186052dec1b3104c4c2_t_jpg.rf.2681ee8d829475f20527bf78050b93d0.jpg
0b91cbffdf82b186052dec1b3104c4c2_t_jpg.rf.db3d7d1e5dff30198ac1fae478f8c640.jpg
0b91cbffdf82b186052dec1b3104c4c2_t_jpg.rf.fdc980ee20ee853e23a4ac467211f93b.jpg
0bc6c33e24d0bf36afd8fcc8ba76cdd1_jpg.rf.2638f7820fe1ad434ea4195db7a08dcd.jpg
0bc6c33e24d0bf36afd8fcc8ba76cdd1_jpg.rf.85000bdf02153b42dc8c04e662ace9cc.jpg
0bc6c33e24d0bf36afd8fcc8ba76cdd1_jpg.rf.c22f27c96a730cd02046b4c9fa267819.jpg
0blcvygr_jpg.rf.a435408519638d68922ab4003ff35dc3.jpg
0blcvygr_jpg.rf.c19d933ebeb67fd837194c4a8f76883e.jpg
0blcvygr_jpg.rf.f4402e0bbcc29bac890a44b8f77ce3be.jpg
0-Suppressed-M240B-1024x683_jpg.rf.20695cd598711ba0ce8678c0c0cce1c1.jpg
0-Suppressed-M240B-1024x683_jpg.rf.86007991585608f49970fb3cb016b648.jpg
0-Suppressed-M240B-1024x683_jpg.rf.b3963ee6ba4acd06de046a8b80bec078.jpg
1000_F_119462557_doJrBmYTJGEOcU9jmU0ZFsyGWCpNJGMn_jpg.rf.54278393f6cd1667faa0ceeda9046653.jpg
1000_F_119462557_doJrBmYTJGEOcU9jmU0ZFsyGWCpNJGMn_jpg.rf.61f3c6708280208d804dc80d0b33ac86.jpg
1000_F_119462557_doJrBmYTJGEOcU9jmU0ZFsyGWCpNJGMn_jpg.rf.98119cc38a817e235b72760e3b73f781.jpg
1000w_q953_jpg.rf.08bcbe8362a432c2177dc0fb4486f135.jpg
1000w_q953_jpg.rf.3424e9282f9d1e90a60d3f97e857f4f1.jpg
1000w_q953_jpg.rf.bb04f8327147ffb21e25b7de6da7b63b.jpg
1000w_q95_jpg.rf.20e3e5e31ee323b28e2acf415271a5ce.jpg
1000w_q95_jpg.rf.c1eea225f250d7cc8f62dd34355bbfb8.jpg
1000w_q95_jpg.rf.decf5b5c9fa6e0b0d04958d0150859e0.jpg
1005-01-412-3129_jpg.rf.4defa65bc10b25cda122fcbef596c8f0.jpg
1005-01-412-3129_jpg.rf.a65acbad6559c372c554523197ade8b3.jpg
1005-01-412-3129_jpg.rf.f059aee0669fb33179da83e7f95c5dc0.jpg
1024px-FN_SCAR_H_PR_jpg.rf.cdeea72f939551050156498c36c5511e.jpg
1024px-FN_SCAR_H_PR_jpg.rf.d2af1a8a09e13bc5f122945b27aa4cda.jpg
1024px-FN_SCAR_H_PR_jpg.rf.f7460b2476fca4316df3feb36f5e2dd8.jpg
```

```
#in the next three cells, we move the data into a structure that the image detection library will be expecting
#but no file data manipulation is necessary
#images can also be segmented into class folders, but we combine all classes here
!mkdir Weapons
!mkdir Weapons/annotations
!mkdir Weapons/Annotations
!mkdir Weapons/Images
```

```
%cd ./
```

```
/content/Weapon-Detection-11
```

```
%cp train/_annotations.coco.json Weapons/annotations/instances_Images.json

%cp train/*.jpg Weapons/Images/
```

Training

In this section we set up the efficientDet-d0 model from backbone and train to our custom case

```
import os
import sys
sys.path.append("Monk_Object_Detection/4_efficientdet/lib/");
```

```
!pip install pycocotools
```

```
Requirement already satisfied: pycocotools in /usr/local/lib/python3.10/dist-packages (2.0)
Requirement already satisfied: setuptools>=18.0 in /usr/local/lib/python3.10/dist-packages (from pycocotools) (67.7.2)
Requirement already satisfied: cython>=0.27.3 in /usr/local/lib/python3.10/dist-packages (from pycocotools) (3.0.8)
Requirement already satisfied: matplotlib>=2.1.0 in /usr/local/lib/python3.10/dist-packages (from pycocotools) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (0.10.0
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (4
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (1
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (1.25.2
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (23
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (9.4.0
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (3
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from cycler>=0.10->matplotlib>=2.1.0->pycocotools) (1
```

```
import os
import torch
import numpy as np

from torch.utils.data import Dataset, DataLoader
from pycocotools.coco import COCO
import cv2


class CocoDataset(Dataset):
    def __init__(self, root_dir, img_dir="images", set_dir='train2017', transform=None):

        self.root_dir = root_dir
        self.img_dir = img_dir
        self.set_name = set_dir
        self.transform = transform

        self.coco = COCO(os.path.join(self.root_dir, 'annotations', 'instances_' + self.set_name + '.json'))
        self.image_ids = self.coco.getImgIds()

        self.load_classes()

    def load_classes(self):

        # load class names (name -> label)
        categories = self.coco.loadCats(self.coco.getCatIds())
        categories.sort(key=lambda x: x['id'])

        self.classes = {}
        self.coco_labels = {}
        self.coco_labels_inverse = {}
        for c in categories:
            self.coco_labels[len(self.classes)] = c['id']
            self.coco_labels_inverse[c['id']] = len(self.classes)
            self.classes[c['name']] = len(self.classes)

        # also load the reverse (label -> name)
        self.labels = {}
        for key, value in self.classes.items():
            self.labels[value] = key

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

    def __getitem__(self, idx):

        img = self.load_image(idx)
        annot = self.load_annotations(idx)
        sample = {'img': img, 'annot': annot}
```

```
        sample = {'img': img, 'annot': annot}
        if self.transform:
            sample = self.transform(sample)
        return sample

    def load_image(self, image_index):
        image_info = self.coco.loadImgs(self.image_ids[image_index])[0]
        path = os.path.join(self.root_dir, self.img_dir, self.set_name, image_info['file_name'])
        img = cv2.imread(path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # if len(img.shape) == 2:
        #     img = skimage.color.gray2rgb(img)

        return img.astype(np.float32) / 255.

    def load_annotations(self, image_index):
        # get ground truth annotations
        annotations_ids = self.coco.getAnnIds(imgIds=self.image_ids[image_index], iscrowd=False)
        annotations = np.zeros((0, 5))

        # some images appear to miss annotations
        if len(annotations_ids) == 0:
            return annotations

        # parse annotations
        coco_annotations = self.coco.loadAnns(annotations_ids)
        for idx, a in enumerate(coco_annotations):

            # some annotations have basically no width / height, skip them
            if a['bbox'][2] < 1 or a['bbox'][3] < 1:
                continue

            annotation = np.zeros((1, 5))
            annotation[0, :4] = a['bbox']
            annotation[0, 4] = self.coco_label_to_label(a['category_id'])
            annotations = np.append(annotations, annotation, axis=0)

        # transform from [x, y, w, h] to [x1, y1, x2, y2]
        annotations[:, 2] = annotations[:, 0] + annotations[:, 2]
        annotations[:, 3] = annotations[:, 1] + annotations[:, 3]

        return annotations

    def coco_label_to_label(self, coco_label):
        return self.coco_labels_inverse[coco_label]

    def label_to_coco_label(self, label):
        return self.coco_labels[label]

    def num_classes(self):
        return len(self.classes)


def collater(data):
    imgs = [s['img'] for s in data]
    annots = [s['annot'] for s in data]
    scales = [s['scale'] for s in data]

    imgs = torch.from_numpy(np.stack(imgs, axis=0))

    max_num_annots = max(annot.shape[0] for annot in annots)

    if max_num_annots > 0:

        annot_padded = torch.ones((len(annots), max_num_annots, 5)) * -1

        if max_num_annots > 0:
            for idx, annot in enumerate(annots):
                if annot.shape[0] > 0:
                    annot_padded[idx, :annot.shape[0], :] = annot
    else:
        annot_padded = torch.ones((len(annots), 1, 5)) * -1

    imgs = imgs.permute(0, 3, 1, 2)

    return {'img': imgs, 'annot': annot_padded, 'scale': scales}


class Resizer(object):
    """Convert ndarrays in sample to Tensors."""

    def __init__(self, common_size=512):
```

```
        self.common_size = common_size;

    def __call__(self, sample, common_size=512):
        common_size = self.common_size;
        image, annots = sample['img'], sample['annot']
        height, width, _ = image.shape
        if height > width:
            scale = common_size / height
            resized_height = common_size
            resized_width = int(width * scale)
        else:
            scale = common_size / width
            resized_height = int(height * scale)
            resized_width = common_size

        image = cv2.resize(image, (resized_width, resized_height))

        new_image = np.zeros((common_size, common_size, 3))
        new_image[0:resized_height, 0:resized_width] = image

        annots[:, :4] *= scale

        return {'img': torch.from_numpy(new_image), 'annot': torch.from_numpy(annots), 'scale': scale}


class Augmenter(object):
    """Convert ndarrays in sample to Tensors."""

    def __call__(self, sample, flip_x=0.5):
        if np.random.rand() < flip_x:
            image, annots = sample['img'], sample['annot']
            image = image[:, ::-1, :]

            rows, cols, channels = image.shape

            x1 = annots[:, 0].copy()
            x2 = annots[:, 2].copy()

            x_tmp = x1.copy()

            annots[:, 0] = cols - x2
            annots[:, 2] = cols - x_tmp

            sample = {'img': image, 'annot': annots}

        return sample


class Normalizer(object):

    def __init__(self):
        self.mean = np.array([[[0.485, 0.456, 0.406]]])
        self.std = np.array([[[0.229, 0.224, 0.225]]])

    def __call__(self, sample):
        image, annots = sample['img'], sample['annot']

        return {'img': ((image.astype(np.float32) - self.mean) / self.std), 'annot': annots}
```

```
import torch
import torch.nn as nn
import numpy as np


class BBoxTransform(nn.Module):

    def __init__(self, mean=None, std=None):
        super(BBoxTransform, self).__init__()
        if mean is None:
            self.mean = torch.from_numpy(np.array([0, 0, 0, 0]).astype(np.float32))
        else:
            self.mean = mean
        if std is None:
            self.std = torch.from_numpy(np.array([0.1, 0.1, 0.2, 0.2]).astype(np.float32))
        else:
            self.std = std
        if torch.cuda.is_available():
            self.mean = self.mean.cuda()
            self.std = self.std.cuda()

    def forward(self, boxes, deltas):

        widths = boxes[:, :, 2] - boxes[:, :, 0]
        heights = boxes[:, :, 3] - boxes[:, :, 1]
        ctr_x = boxes[:, :, 0] + 0.5 * widths
        ctr_y = boxes[:, :, 1] + 0.5 * heights

        dx = deltas[:, :, 0] * self.std[0] + self.mean[0]
        dy = deltas[:, :, 1] * self.std[1] + self.mean[1]
        dw = deltas[:, :, 2] * self.std[2] + self.mean[2]
        dh = deltas[:, :, 3] * self.std[3] + self.mean[3]

        pred_ctr_x = ctr_x + dx * widths
        pred_ctr_y = ctr_y + dy * heights
        pred_w = torch.exp(dw) * widths
        pred_h = torch.exp(dh) * heights

        pred_boxes_x1 = pred_ctr_x - 0.5 * pred_w
        pred_boxes_y1 = pred_ctr_y - 0.5 * pred_h
        pred_boxes_x2 = pred_ctr_x + 0.5 * pred_w
        pred_boxes_y2 = pred_ctr_y + 0.5 * pred_h

        pred_boxes = torch.stack([pred_boxes_x1, pred_boxes_y1, pred_boxes_x2, pred_boxes_y2], dim=2)

        return pred_boxes


class ClipBoxes(nn.Module):

    def __init__(self):
        super(ClipBoxes, self).__init__()

    def forward(self, boxes, img):
        batch_size, num_channels, height, width = img.shape

        boxes[:, :, 0] = torch.clamp(boxes[:, :, 0], min=0)
        boxes[:, :, 1] = torch.clamp(boxes[:, :, 1], min=0)

        boxes[:, :, 2] = torch.clamp(boxes[:, :, 2], max=width)
        boxes[:, :, 3] = torch.clamp(boxes[:, :, 3], max=height)

        return boxes


class Anchors(nn.Module):
    def __init__(self, pyramid_levels=None, strides=None, sizes=None, ratios=None, scales=None):
        super(Anchors, self).__init__()

        if pyramid_levels is None:
            self.pyramid_levels = [3, 4, 5, 6, 7]
        if strides is None:
            self.strides = [2 ** x for x in self.pyramid_levels]
        if sizes is None:
            self.sizes = [2 ** (x + 2) for x in self.pyramid_levels]
        if ratios is None:
            self.ratios = np.array([0.5, 1, 2])
        if scales is None:
            self.scales = np.array([2 ** 0, 2 ** (1.0 / 3.0), 2 ** (2.0 / 3.0)])

    def forward(self, image):
```

```
        image_shape = image.shape[2:]
        image_shape = np.array(image_shape)
        image_shapes = [(image_shape + 2 ** x - 1) // (2 ** x) for x in self.pyramid_levels]

        all_anchors = np.zeros((0, 4)).astype(np.float32)

        for idx, p in enumerate(self.pyramid_levels):
            anchors = generate_anchors(base_size=self.sizes[idx], ratios=self.ratios, scales=self.scales)
            shifted_anchors = shift(image_shapes[idx], self.strides[idx], anchors)
            all_anchors = np.append(all_anchors, shifted_anchors, axis=0)

        all_anchors = np.expand_dims(all_anchors, axis=0)

        anchors = torch.from_numpy(all_anchors.astype(np.float32))
        if torch.cuda.is_available():
            anchors = anchors.cuda()
        return anchors


def generate_anchors(base_size=16, ratios=None, scales=None):
    if ratios is None:
        ratios = np.array([0.5, 1, 2])

    if scales is None:
        scales = np.array([2 ** 0, 2 ** (1.0 / 3.0), 2 ** (2.0 / 3.0)])

    num_anchors = len(ratios) * len(scales)
    anchors = np.zeros((num_anchors, 4))
    anchors[:, 2:] = base_size * np.tile(scales, (2, len(ratios))).T
    areas = anchors[:, 2] * anchors[:, 3]
    anchors[:, 2] = np.sqrt(areas / np.repeat(ratios, len(scales)))
    anchors[:, 3] = anchors[:, 2] * np.repeat(ratios, len(scales))
    anchors[:, 0::2] -= np.tile(anchors[:, 2] * 0.5, (2, 1)).T
    anchors[:, 1::2] -= np.tile(anchors[:, 3] * 0.5, (2, 1)).T

    return anchors


def compute_shape(image_shape, pyramid_levels):
    image_shape = np.array(image_shape[:2])
    image_shapes = [(image_shape + 2 ** x - 1) // (2 ** x) for x in pyramid_levels]
    return image_shapes


def shift(shape, stride, anchors):
    shift_x = (np.arange(0, shape[1]) + 0.5) * stride
    shift_y = (np.arange(0, shape[0]) + 0.5) * stride
    shift_x, shift_y = np.meshgrid(shift_x, shift_y)
    shifts = np.vstack((
        shift_x.ravel(), shift_y.ravel(),
        shift_x.ravel(), shift_y.ravel()
    )).transpose()

    A = anchors.shape[0]
    K = shifts.shape[0]
    all_anchors = (anchors.reshape((1, A, 4)) + shifts.reshape((1, K, 4)).transpose((1, 0, 2)))
    all_anchors = all_anchors.reshape((K * A, 4))

    return all_anchors
```

```
import torch
import torch.nn as nn


def calc_iou(a, b):

    area = (b[:, 2] - b[:, 0]) * (b[:, 3] - b[:, 1])
    iw = torch.min(torch.unsqueeze(a[:, 2], dim=1), b[:, 2]) - torch.max(torch.unsqueeze(a[:, 0], 1), b[:, 0])
    ih = torch.min(torch.unsqueeze(a[:, 3], dim=1), b[:, 3]) - torch.max(torch.unsqueeze(a[:, 1], 1), b[:, 1])
    iw = torch.clamp(iw, min=0)
    ih = torch.clamp(ih, min=0)
    ua = torch.unsqueeze((a[:, 2] - a[:, 0]) * (a[:, 3] - a[:, 1]), dim=1) + area - iw * ih
    ua = torch.clamp(ua, min=1e-8)
    intersection = iw * ih
    IoU = intersection / ua

    return IoU


class FocalLoss(nn.Module):
    def __init__(self):
        super(FocalLoss, self).__init__()

    def forward(self, classifications, regressions, anchors, annotations):
        alpha = 0.25
        gamma = 2.0
        batch_size = classifications.shape[0]
        classification_losses = []
        regression_losses = []

        anchor = anchors[0, :, :]

        anchor_widths = anchor[:, 2] - anchor[:, 0]
        anchor_heights = anchor[:, 3] - anchor[:, 1]
        anchor_ctr_x = anchor[:, 0] + 0.5 * anchor_widths
        anchor_ctr_y = anchor[:, 1] + 0.5 * anchor_heights

        for j in range(batch_size):

            classification = classifications[j, :, :]
            regression = regressions[j, :, :]

            bbox_annotation = annotations[j, :, :]
            bbox_annotation = bbox_annotation[bbox_annotation[:, 4] != -1]

            if bbox_annotation.shape[0] == 0:
                if torch.cuda.is_available():
                    regression_losses.append(torch.tensor(0).float().cuda())
                    classification_losses.append(torch.tensor(0).float().cuda())
                else:
                    regression_losses.append(torch.tensor(0).float())
                    classification_losses.append(torch.tensor(0).float())

                continue

            classification = torch.clamp(classification, 1e-4, 1.0 - 1e-4)

            IoU = calc_iou(anchors[0, :, :], bbox_annotation[:, :4])

            IoU_max, IoU_argmax = torch.max(IoU, dim=1)

            # compute the loss for classification
            targets = torch.ones(classification.shape) * -1
            if torch.cuda.is_available():
                targets = targets.cuda()

            targets[torch.lt(IoU_max, 0.4), :] = 0

            positive_indices = torch.ge(IoU_max, 0.5)

            num_positive_anchors = positive_indices.sum()

            assigned_annotations = bbox_annotation[IoU_argmax, :]

            targets[positive_indices, :] = 0
            targets[positive_indices, assigned_annotations[positive_indices, 4].long()] = 1

            alpha_factor = torch.ones(targets.shape) * alpha
            if torch.cuda.is_available():
                alpha_factor = alpha_factor.cuda()

            alpha_factor = torch.where(torch.eq(targets, 1.), alpha_factor, 1. - alpha_factor)
```

```
        focal_weight = torch.where(torch.eq(targets, 1.), 1. - classification, classification)
        focal_weight = alpha_factor * torch.pow(focal_weight, gamma)

        bce = -(targets * torch.log(classification) + (1.0 - targets) * torch.log(1.0 - classification))

        cls_loss = focal_weight * bce

        zeros = torch.zeros(cls_loss.shape)
        if torch.cuda.is_available():
            zeros = zeros.cuda()
        cls_loss = torch.where(torch.ne(targets, -1.0), cls_loss, zeros)

        classification_losses.append(cls_loss.sum() / torch.clamp(num_positive_anchors.float(), min=1.0))


        if positive_indices.sum() > 0:
            assigned_annotations = assigned_annotations[positive_indices, :]

            anchor_widths_pi = anchor_widths[positive_indices]
            anchor_heights_pi = anchor_heights[positive_indices]
            anchor_ctr_x_pi = anchor_ctr_x[positive_indices]
            anchor_ctr_y_pi = anchor_ctr_y[positive_indices]

            gt_widths = assigned_annotations[:, 2] - assigned_annotations[:, 0]
            gt_heights = assigned_annotations[:, 3] - assigned_annotations[:, 1]
            gt_ctr_x = assigned_annotations[:, 0] + 0.5 * gt_widths
            gt_ctr_y = assigned_annotations[:, 1] + 0.5 * gt_heights

            gt_widths = torch.clamp(gt_widths, min=1)
            gt_heights = torch.clamp(gt_heights, min=1)

            targets_dx = (gt_ctr_x - anchor_ctr_x_pi) / anchor_widths_pi
            targets_dy = (gt_ctr_y - anchor_ctr_y_pi) / anchor_heights_pi
            targets_dw = torch.log(gt_widths / anchor_widths_pi)
            targets_dh = torch.log(gt_heights / anchor_heights_pi)

            targets = torch.stack((targets_dx, targets_dy, targets_dw, targets_dh))
            targets = targets.t()

            norm = torch.Tensor([[0.1, 0.1, 0.2, 0.2]])
            if torch.cuda.is_available():
                norm = norm.cuda()
            targets = targets / norm

            regression_diff = torch.abs(targets - regression[positive_indices, :])

            regression_loss = torch.where(
                torch.le(regression_diff, 1.0 / 9.0),
                0.5 * 9.0 * torch.pow(regression_diff, 2),
                regression_diff - 0.5 / 9.0
            )
            regression_losses.append(regression_loss.mean())
        else:
            if torch.cuda.is_available():
                regression_losses.append(torch.tensor(0).float().cuda())
            else:
                regression_losses.append(torch.tensor(0).float())

    return torch.stack(classification_losses).mean(dim=0, keepdim=True), torch.stack(regression_losses).mean(dim=0,
                                                                                                             keepdim=True)
```

```
import torch.nn as nn
import torch
import math
from efficientnet_pytorch import EfficientNet as EffNet
# from src.utils import BBoxTransform, ClipBoxes, Anchors
# from src.loss import FocalLoss
from torchvision.ops.boxes import nms as nms_torch


def nms(dets, thresh):
    return nms_torch(dets[:, :4], dets[:, 4], thresh)


class ConvBlock(nn.Module):
    def __init__(self, num_channels):
        super(ConvBlock, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(num_channels, num_channels, kernel_size=3, stride=1, padding=1, groups=num_channels),
            nn.Conv2d(num_channels, num_channels, kernel_size=1, stride=1, padding=0),
            nn.BatchNorm2d(num_features=num_channels, momentum=0.9997, eps=4e-5), nn.ReLU())

    def forward(self, input):
        return self.conv(input)


class BiFPN(nn.Module):
    def __init__(self, num_channels, epsilon=1e-4):
        super(BiFPN, self).__init__()
        self.epsilon = epsilon
        # Conv layers
        self.conv6_up = ConvBlock(num_channels)
        self.conv5_up = ConvBlock(num_channels)
        self.conv4_up = ConvBlock(num_channels)
        self.conv3_up = ConvBlock(num_channels)
        self.conv4_down = ConvBlock(num_channels)
        self.conv5_down = ConvBlock(num_channels)
        self.conv6_down = ConvBlock(num_channels)
        self.conv7_down = ConvBlock(num_channels)

        # Feature scaling layers
        self.p6_upsample = nn.Upsample(scale_factor=2, mode='nearest')
        self.p5_upsample = nn.Upsample(scale_factor=2, mode='nearest')
        self.p4_upsample = nn.Upsample(scale_factor=2, mode='nearest')
        self.p3_upsample = nn.Upsample(scale_factor=2, mode='nearest')

        self.p4_downsample = nn.MaxPool2d(kernel_size=2)
        self.p5_downsample = nn.MaxPool2d(kernel_size=2)
        self.p6_downsample = nn.MaxPool2d(kernel_size=2)
        self.p7_downsample = nn.MaxPool2d(kernel_size=2)

        # Weight
        self.p6_w1 = nn.Parameter(torch.ones(2))
        self.p6_w1_relu = nn.ReLU()
        self.p5_w1 = nn.Parameter(torch.ones(2))
        self.p5_w1_relu = nn.ReLU()
        self.p4_w1 = nn.Parameter(torch.ones(2))
        self.p4_w1_relu = nn.ReLU()
        self.p3_w1 = nn.Parameter(torch.ones(2))
        self.p3_w1_relu = nn.ReLU()

        self.p4_w2 = nn.Parameter(torch.ones(3))
        self.p4_w2_relu = nn.ReLU()
        self.p5_w2 = nn.Parameter(torch.ones(3))
        self.p5_w2_relu = nn.ReLU()
        self.p6_w2 = nn.Parameter(torch.ones(3))
        self.p6_w2_relu = nn.ReLU()
        self.p7_w2 = nn.Parameter(torch.ones(2))
        self.p7_w2_relu = nn.ReLU()

    def forward(self, inputs):
        """
            P7_0 -------------------------- P7_2 -------->

            P6_0 ---------- P6_1 ---------- P6_2 -------->

            P5_0 ---------- P5_1 ---------- P5_2 -------->

            P4_0 ---------- P4_1 ---------- P4_2 -------->

            P3_0 -------------------------- P3_2 -------->
        """
```

```
        # P3_0, P4_0, P5_0, P6_0 and P7_0
        p3_in, p4_in, p5_in, p6_in, p7_in = inputs
        # P7_0 to P7_2
        # Weights for P6_0 and P7_0 to P6_1
        p6_w1 = self.p6_w1_relu(self.p6_w1)
        weight = p6_w1 / (torch.sum(p6_w1, dim=0) + self.epsilon)
        # Connections for P6_0 and P7_0 to P6_1 respectively
        p6_up = self.conv6_up(weight[0] * p6_in + weight[1] * self.p6_upsample(p7_in))
        # Weights for P5_0 and P6_0 to P5_1
        p5_w1 = self.p5_w1_relu(self.p5_w1)
        weight = p5_w1 / (torch.sum(p5_w1, dim=0) + self.epsilon)
        # Connections for P5_0 and P6_0 to P5_1 respectively
        p5_up = self.conv5_up(weight[0] * p5_in + weight[1] * self.p5_upsample(p6_up))
        # Weights for P4_0 and P5_0 to P4_1
        p4_w1 = self.p4_w1_relu(self.p4_w1)
        weight = p4_w1 / (torch.sum(p4_w1, dim=0) + self.epsilon)
        # Connections for P4_0 and P5_0 to P4_1 respectively
        p4_up = self.conv4_up(weight[0] * p4_in + weight[1] * self.p4_upsample(p5_up))

        # Weights for P3_0 and P4_1 to P3_2
        p3_w1 = self.p3_w1_relu(self.p3_w1)
        weight = p3_w1 / (torch.sum(p3_w1, dim=0) + self.epsilon)
        # Connections for P3_0 and P4_1 to P3_2 respectively
        p3_out = self.conv3_up(weight[0] * p3_in + weight[1] * self.p3_upsample(p4_up))

        # Weights for P4_0, P4_1 and P3_2 to P4_2
        p4_w2 = self.p4_w2_relu(self.p4_w2)
        weight = p4_w2 / (torch.sum(p4_w2, dim=0) + self.epsilon)
        # Connections for P4_0, P4_1 and P3_2 to P4_2 respectively
        p4_out = self.conv4_down(
            weight[0] * p4_in + weight[1] * p4_up + weight[2] * self.p4_downsample(p3_out))
        # Weights for P5_0, P5_1 and P4_2 to P5_2
        p5_w2 = self.p5_w2_relu(self.p5_w2)
        weight = p5_w2 / (torch.sum(p5_w2, dim=0) + self.epsilon)
        # Connections for P5_0, P5_1 and P4_2 to P5_2 respectively
        p5_out = self.conv5_down(
            weight[0] * p5_in + weight[1] * p5_up + weight[2] * self.p5_downsample(p4_out))
        # Weights for P6_0, P6_1 and P5_2 to P6_2
        p6_w2 = self.p6_w2_relu(self.p6_w2)
        weight = p6_w2 / (torch.sum(p6_w2, dim=0) + self.epsilon)
        # Connections for P6_0, P6_1 and P5_2 to P6_2 respectively
        p6_out = self.conv6_down(
            weight[0] * p6_in + weight[1] * p6_up + weight[2] * self.p6_downsample(p5_out))
        # Weights for P7_0 and P6_2 to P7_2
        p7_w2 = self.p7_w2_relu(self.p7_w2)
        weight = p7_w2 / (torch.sum(p7_w2, dim=0) + self.epsilon)
        # Connections for P7_0 and P6_2 to P7_2
        p7_out = self.conv7_down(weight[0] * p7_in + weight[1] * self.p7_downsample(p6_out))

        return p3_out, p4_out, p5_out, p6_out, p7_out


class Regressor(nn.Module):
    def __init__(self, in_channels, num_anchors, num_layers):
        super(Regressor, self).__init__()
        layers = []
        for _ in range(num_layers):
            layers.append(nn.Conv2d(in_channels, in_channels, kernel_size=3, stride=1, padding=1))
            layers.append(nn.ReLU(True))
        self.layers = nn.Sequential(*layers)
        self.header = nn.Conv2d(in_channels, num_anchors * 4, kernel_size=3, stride=1, padding=1)

    def forward(self, inputs):
        inputs = self.layers(inputs)
        inputs = self.header(inputs)
        output = inputs.permute(0, 2, 3, 1)
        return output.contiguous().view(output.shape[0], -1, 4)


class Classifier(nn.Module):
    def __init__(self, in_channels, num_anchors, num_classes, num_layers):
        super(Classifier, self).__init__()
        self.num_anchors = num_anchors
        self.num_classes = num_classes
        layers = []
        for _ in range(num_layers):
            layers.append(nn.Conv2d(in_channels, in_channels, kernel_size=3, stride=1, padding=1))
            layers.append(nn.ReLU(True))
        self.layers = nn.Sequential(*layers)
        self.header = nn.Conv2d(in_channels, num_anchors * num_classes, kernel_size=3, stride=1, padding=1)
        self.act = nn.Sigmoid()
```

```
    def forward(self, inputs):
        inputs = self.layers(inputs)
        inputs = self.header(inputs)
        inputs = self.act(inputs)
        inputs = inputs.permute(0, 2, 3, 1)
        output = inputs.contiguous().view(inputs.shape[0], inputs.shape[1], inputs.shape[2], self.num_anchors,
                                          self.num_classes)
        return output.contiguous().view(output.shape[0], -1, self.num_classes)


class EfficientNet(nn.Module):
    def __init__(self, model_name):
        super(EfficientNet, self).__init__()
        model = EffNet.from_pretrained(model_name)
        del model._conv_head
        del model._bn1
        del model._avg_pooling
        del model._dropout
        del model._fc
        self.model = model

    def forward(self, x):
        x = self.model._swish(self.model._bn0(self.model._conv_stem(x)))
        feature_maps = []
        for idx, block in enumerate(self.model._blocks):
            drop_connect_rate = self.model._global_params.drop_connect_rate
            if drop_connect_rate:
                drop_connect_rate *= float(idx) / len(self.model._blocks)
            x = block(x, drop_connect_rate=drop_connect_rate)
            if block._depthwise_conv.stride == [2, 2]:
                feature_maps.append(x)

        return feature_maps[1:]


class EfficientDet(nn.Module):
    def __init__(self, num_anchors=9, num_classes=20, compound_coef=0, model_name="efficientnet-b0"):
        super(EfficientDet, self).__init__()
        self.compound_coef = compound_coef

        self.num_channels = [64, 88, 112, 160, 224, 288, 384, 384][self.compound_coef]

        if(self.compound_coef == 0 or self.compound_coef==1):
          self.conv3 = nn.Conv2d(40, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv4 = nn.Conv2d(80, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv5 = nn.Conv2d(192, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv6 = nn.Conv2d(192, self.num_channels, kernel_size=3, stride=2, padding=1)
          self.conv7 = nn.Sequential(nn.ReLU(),
                                     nn.Conv2d(self.num_channels, self.num_channels, kernel_size=3, stride=2, padding=1))
        elif(self.compound_coef == 2):
          self.conv3 = nn.Conv2d(48, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv4 = nn.Conv2d(88, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv5 = nn.Conv2d(208, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv6 = nn.Conv2d(208, self.num_channels, kernel_size=3, stride=2, padding=1)
          self.conv7 = nn.Sequential(nn.ReLU(),
                                     nn.Conv2d(self.num_channels, self.num_channels, kernel_size=3, stride=2, padding=1))
        elif(self.compound_coef == 3):
          self.conv3 = nn.Conv2d(48, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv4 = nn.Conv2d(96, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv5 = nn.Conv2d(232, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv6 = nn.Conv2d(232, self.num_channels, kernel_size=3, stride=2, padding=1)
          self.conv7 = nn.Sequential(nn.ReLU(),
                                     nn.Conv2d(self.num_channels, self.num_channels, kernel_size=3, stride=2, padding=1))
        elif(self.compound_coef == 4):
          self.conv3 = nn.Conv2d(56, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv4 = nn.Conv2d(112, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv5 = nn.Conv2d(272, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv6 = nn.Conv2d(272, self.num_channels, kernel_size=3, stride=2, padding=1)
          self.conv7 = nn.Sequential(nn.ReLU(),
                                     nn.Conv2d(self.num_channels, self.num_channels, kernel_size=3, stride=2, padding=1))
        elif(self.compound_coef == 5):
          self.conv3 = nn.Conv2d(64, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv4 = nn.Conv2d(128, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv5 = nn.Conv2d(304, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv6 = nn.Conv2d(304, self.num_channels, kernel_size=3, stride=2, padding=1)
          self.conv7 = nn.Sequential(nn.ReLU(),
                                     nn.Conv2d(self.num_channels, self.num_channels, kernel_size=3, stride=2, padding=1))
        elif(self.compound_coef == 6):
          self.conv3 = nn.Conv2d(72, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv4 = nn.Conv2d(144, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv5 = nn.Conv2d(344, self.num_channels, kernel_size=1, stride=1, padding=0)
          self.conv6 = nn.Conv2d(344, self.num_channels, kernel_size=3, stride=2, padding=1)
```

```
      self.conv7 = nn.Sequential(nn.ReLU(),
                                 nn.Conv2d(self.num_channels, self.num_channels, kernel_size=3, stride=2, padding=1))
    elif(self.compound_coef == 7):
      self.conv3 = nn.Conv2d(80, self.num_channels, kernel_size=1, stride=1, padding=0)
      self.conv4 = nn.Conv2d(160, self.num_channels, kernel_size=1, stride=1, padding=0)
      self.conv5 = nn.Conv2d(384, self.num_channels, kernel_size=1, stride=1, padding=0)
      self.conv6 = nn.Conv2d(384, self.num_channels, kernel_size=3, stride=2, padding=1)
      self.conv7 = nn.Sequential(nn.ReLU(),
                                 nn.Conv2d(self.num_channels, self.num_channels, kernel_size=3, stride=2, padding=1))



    self.bifpn = nn.Sequential(*[BiFPN(self.num_channels) for _ in range(min(2 + self.compound_coef, 8))])

    self.num_classes = num_classes
    self.regressor = Regressor(in_channels=self.num_channels, num_anchors=num_anchors,
                               num_layers=3 + self.compound_coef // 3)
    self.classifier = Classifier(in_channels=self.num_channels, num_anchors=num_anchors, num_classes=num_classes,
                                 num_layers=3 + self.compound_coef // 3)

    self.anchors = Anchors()
    self.regressBoxes = BBoxTransform()
    self.clipBoxes = ClipBoxes()
    self.focalLoss = FocalLoss()

    for m in self.modules():
        if isinstance(m, nn.Conv2d):
            n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
            m.weight.data.normal_(0, math.sqrt(2. / n))
        elif isinstance(m, nn.BatchNorm2d):
            m.weight.data.fill_(1)
            m.bias.data.zero_()

    prior = 0.01

    self.classifier.header.weight.data.fill_(0)
    self.classifier.header.bias.data.fill_(-math.log((1.0 - prior) / prior))

    self.regressor.header.weight.data.fill_(0)
    self.regressor.header.bias.data.fill_(0)

    self.backbone_net = EfficientNet(model_name)

def freeze_bn(self):
    for m in self.modules():
        if isinstance(m, nn.BatchNorm2d):
            m.eval()

def forward(self, inputs):
    if len(inputs) == 2:
        is_training = True
        img_batch, annotations = inputs
    else:
        is_training = False
        img_batch = inputs

    c3, c4, c5 = self.backbone_net(img_batch)
    p3 = self.conv3(c3)
    p4 = self.conv4(c4)
    p5 = self.conv5(c5)
    p6 = self.conv6(c5)
    p7 = self.conv7(p6)

    features = [p3, p4, p5, p6, p7]
    features = self.bifpn(features)

    regression = torch.cat([self.regressor(feature) for feature in features], dim=1)
    classification = torch.cat([self.classifier(feature) for feature in features], dim=1)
    anchors = self.anchors(img_batch)

    if is_training:
        return self.focalLoss(classification, regression, anchors, annotations)
    else:
        transformed_anchors = self.regressBoxes(anchors, regression)
        transformed_anchors = self.clipBoxes(transformed_anchors, img_batch)

        scores = torch.max(classification, dim=2, keepdim=True)[0]

        scores_over_thresh = (scores > 0.05)[0, :, 0]

        if scores_over_thresh.sum() == 0:
```

```
                return [torch.zeros(0), torch.zeros(0), torch.zeros(0, 4)]

            classification = classification[:, scores_over_thresh, :]
            transformed_anchors = transformed_anchors[:, scores_over_thresh, :]
            scores = scores[:, scores_over_thresh, :]

            anchors_nms_idx = nms(torch.cat([transformed_anchors, scores], dim=2)[0, :, :], 0.5)

            nms_scores, nms_class = classification[0, anchors_nms_idx, :].max(dim=1)

            return [nms_scores, nms_class, transformed_anchors[0, anchors_nms_idx, :]]


if __name__ == '__main__':
    from tensorboardX import SummaryWriter
    def count_parameters(model):
        return sum(p.numel() for p in model.parameters() if p.requires_grad)

    model = EfficientDet(num_classes=80)
    print (count_parameters(model))
```

```
 Loaded pretrained weights for efficientnet-b0
 4499798
```

```
import os
import argparse
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision import transforms
# from src.dataset import CocoDataset, Resizer, Normalizer, Augmenter, collater
# from src.model import EfficientDet
from tensorboardX import SummaryWriter
import shutil
import numpy as np
from tqdm.autonotebook import tqdm


class Detector():
    '''
    Class to train a detector

    Args:
        verbose (int): Set verbosity levels
                        0 - Print Nothing
                        1 - Print desired details
    '''
    def __init__(self, verbose=1):
        self.system_dict = {};
        self.system_dict["verbose"] = verbose;
        self.system_dict["local"] = {};
        self.system_dict["dataset"] = {};
        self.system_dict["dataset"]["train"] = {};
        self.system_dict["dataset"]["val"] = {};
        self.system_dict["dataset"]["val"]["status"] = False;

        self.system_dict["params"] = {};
        self.system_dict["params"]["image_size"] = 512;
        self.system_dict["params"]["batch_size"] = 8;
        self.system_dict["params"]["num_workers"] = 3;
        self.system_dict["params"]["use_gpu"] = True;
        self.system_dict["params"]["gpu_devices"] = [0];
        self.system_dict["params"]["lr"] = 0.0001;
        self.system_dict["params"]["num_epochs"] = 10;
        self.system_dict["params"]["val_interval"] = 1;
        self.system_dict["params"]["es_min_delta"] = 0.0;
        self.system_dict["params"]["es_patience"] = 0;


        self.system_dict["output"] = {};
        self.system_dict["output"]["log_path"] = "tensorboard/signatrix_efficientdet_coco";
        self.system_dict["output"]["saved_path"] = "trained/";
        self.system_dict["output"]["best_epoch"] = 0;
        self.system_dict["output"]["best_loss"] = 1e5;



    def Train_Dataset(self, root_dir, coco_dir, img_dir, set_dir, batch_size=8, image_size=512, use_gpu=True, num_workers=3):
        '''
        User function: Set training dataset parameters

        Dataset Directory Structure

                    root_dir
                       |
                       |------coco_dir
                       |         |
                       |         |----img_dir
                       |                |
                       |                |------<set_dir_train> (set_dir) (Train)
                       |                         |
                       |                         |---------img1.jpg
                       |                         |---------img2.jpg
                       |                         |---------.........(and so on)
                       |
                       |
                       |         |---annotations
                       |         |----|
                       |              |--------------------instances_Train.json  (instances_<set_dir_train>.json)
                       |              |--------------------classes.txt


                - instances_Train.json -> In proper COCO format
                - classes.txt          -> A list of classes in alphabetical order
```

```
        For TrainSet
         - root_dir = "../sample_dataset";
         - coco_dir = "kangaroo";
         - img_dir = "images";
         - set_dir = "Train";


        Note: Annotation file name too coincides against the set_dir

    Args:
        root_dir (str): Path to root directory containing coco_dir
        coco_dir (str): Name of coco_dir containing image folder and annotation folder
        img_dir (str): Name of folder containing all training and validation folders
        set_dir (str): Name of folder containing all training images
        batch_size (int): Mini batch sampling size for training epochs
        image_size (int): Either of [512, 300]
        use_gpu (bool): If True use GPU else run on CPU
        num_workers (int): Number of parallel processors for data loader

    Returns:
        None
    '''
    self.system_dict["dataset"]["train"]["root_dir"] = root_dir;
    self.system_dict["dataset"]["train"]["coco_dir"] = coco_dir;
    self.system_dict["dataset"]["train"]["img_dir"] = img_dir;
    self.system_dict["dataset"]["train"]["set_dir"] = set_dir;


    self.system_dict["params"]["batch_size"] = batch_size;
    self.system_dict["params"]["image_size"] = image_size;
    self.system_dict["params"]["use_gpu"] = use_gpu;
    self.system_dict["params"]["num_workers"] = num_workers;

    if(self.system_dict["params"]["use_gpu"]):
        if torch.cuda.is_available():
            self.system_dict["local"]["num_gpus"] = torch.cuda.device_count()
            torch.cuda.manual_seed(123)
        else:
            torch.manual_seed(123)

    self.system_dict["local"]["training_params"] = {"batch_size": self.system_dict["params"]["batch_size"] * self.system_dict["loc
                                                    "shuffle": True,
                                                    "drop_last": True,
                                                    "collate_fn": collater,
                                                    "num_workers": self.system_dict["params"]["num_workers"]}

    self.system_dict["local"]["training_set"] = CocoDataset(root_dir=self.system_dict["dataset"]["train"]["root_dir"] + "/" + self
                                                     img_dir = self.system_dict["dataset"]["train"]["img_dir"],
                                                     set_dir = self.system_dict["dataset"]["train"]["set_dir"],
                                                     transform = transforms.Compose([Normalizer(), Augmenter(), Resizer(common_
                                                     )

    self.system_dict["local"]["training_generator"] = DataLoader(self.system_dict["local"]["training_set"],
                                                                 **self.system_dict["local"]["training_params"]);


def Val_Dataset(self, root_dir, coco_dir, img_dir, set_dir):
    '''
    User function: Set training dataset parameters

    Dataset Directory Structure

                root_dir
                   |
                   |------coco_dir
                   |         |
                   |         |----img_dir
                   |                 |
                   |                 |------<set_dir_val> (set_dir) (Validation)
                   |                          |
                   |                          |---------img1.jpg
                   |                          |---------img2.jpg
                   |                          |---------.........(and so on)
                   |
                   |
                   |         |---annotations
                   |         |----|
                   |              |--------------------instances_Val.json  (instances_<set_dir_val>.json)
                   |              |--------------------classes.txt


         - instances_Train.json -> In proper COCO format
```

```
          - classes.txt          -> A list of classes in alphabetical order


          For ValSet
           - root_dir = "..sample_dataset";
           - coco_dir = "kangaroo";
           - img_dir = "images";
           - set_dir = "Val";

           Note: Annotation file name too coincides against the set_dir

      Args:
          root_dir (str): Path to root directory containing coco_dir
          coco_dir (str): Name of coco_dir containing image folder and annotation folder
          img_dir (str): Name of folder containing all training and validation folders
          set_dir (str): Name of folder containing all validation images

      Returns:
          None
      '''
      self.system_dict["dataset"]["val"]["status"] = True;
      self.system_dict["dataset"]["val"]["root_dir"] = root_dir;
      self.system_dict["dataset"]["val"]["coco_dir"] = coco_dir;
      self.system_dict["dataset"]["val"]["img_dir"] = img_dir;
      self.system_dict["dataset"]["val"]["set_dir"] = set_dir;

      self.system_dict["local"]["val_params"] = {"batch_size": self.system_dict["params"]["batch_size"],
                                                 "shuffle": False,
                                                 "drop_last": False,
                                                 "collate_fn": collater,
                                                 "num_workers": self.system_dict["params"]["num_workers"]}

      self.system_dict["local"]["val_set"] = CocoDataset(root_dir=self.system_dict["dataset"]["val"]["root_dir"] + "/" + self.system
                                                  img_dir = self.system_dict["dataset"]["val"]["img_dir"],
                                                  set_dir = self.system_dict["dataset"]["val"]["set_dir"],
                                                  transform=transforms.Compose([Normalizer(), Resizer(common_size = self.system_dict

      self.system_dict["local"]["test_generator"] = DataLoader(self.system_dict["local"]["val_set"],
                                                               **self.system_dict["local"]["val_params"])


#efficientnet-b0;
#efficientnet-b1;
#efficientnet-b2;
#efficientnet-b3;
#efficientnet-b4;
#efficientnet-b5;
#efficientnet-b6;
#efficientnet-b7;
#efficientnet-b8;
def Model(self, model_name="efficientnet-b0", gpu_devices=[0], load_pretrained_model_from=None):
      '''
      User function: Set Model parameters

      Args:
          gpu_devices (list): List of GPU Device IDs to be used in training

      Returns:
          None
      '''
      if(not load_pretrained_model_from):
          num_classes = self.system_dict["local"]["training_set"].num_classes();
          coeff = int(model_name[-1])
          efficientdet = EfficientDet(num_classes=num_classes, compound_coef=coeff, model_name=model_name);

          if self.system_dict["params"]["use_gpu"]:
              self.system_dict["params"]["gpu_devices"] = gpu_devices
              if len(self.system_dict["params"]["gpu_devices"])==1:
                  os.environ["CUDA_VISIBLE_DEVICES"] = str(self.system_dict["params"]["gpu_devices"][0])
              else:
                  os.environ["CUDA_VISIBLE_DEVICES"] = ','.join([str(id) for id in self.system_dict["params"]["gpu_devices"]])
              self.system_dict["local"]["device"] = 'cuda' if torch.cuda.is_available() else 'cpu'
              efficientdet = efficientdet.to(self.system_dict["local"]["device"])
              efficientdet= torch.nn.DataParallel(efficientdet).to(self.system_dict["local"]["device"])

          self.system_dict["local"]["model"] = efficientdet;
          self.system_dict["local"]["model"].train();
      else:
          efficientdet = torch.load(load_pretrained_model_from).module
          if self.system_dict["params"]["use_gpu"]:
              self.system_dict["params"]["gpu_devices"] = gpu_devices
              if len(self.system_dict["params"]["gpu_devices"])==1:
```

```
                os.environ["CUDA_VISIBLE_DEVICES"] = str(self.system_dict["params"]["gpu_devices"][0])
            else:
                os.environ["CUDA_VISIBLE_DEVICES"] = ','.join([str(id) for id in self.system_dict["params"]["gpu_devices"]])
            self.system_dict["local"]["device"] = 'cuda' if torch.cuda.is_available() else 'cpu'
            efficientdet = efficientdet.to(self.system_dict["local"]["device"])
            efficientdet= torch.nn.DataParallel(efficientdet).to(self.system_dict["local"]["device"])

        self.system_dict["local"]["model"] = efficientdet;
        self.system_dict["local"]["model"].train();



def Set_Hyperparams(self, lr=0.0001, val_interval=1, es_min_delta=0.0, es_patience=0):
    '''
    User function: Set hyper parameters

    Args:
        lr (float): Initial learning rate for training
        val_interval (int): Post specified number of training epochs, a validation epoch will be carried out
        es_min_delta (float): Loss detla value, if loss doesnn't change more than this value for "es_patience" number of epochs, t
        es_patience (int): If loss doesnn't change more than this "es_min_delta" value for "es_patience" number of epochs, trainin
```