Title: Detecting Defects in Steel

Name of Student: KARTIK JOSHI

Student IDs: 17BLC1187

Please send your Word file and your final power point slides at 2020/07/03 to [andrewlin2011@gmail.com](mailto:andrewlin2011@gmail.com) and cc to [fityanul@telkomuniversity.ac.id](mailto:fityanul@telkomuniversity.ac.id)

Index

1. Abstract

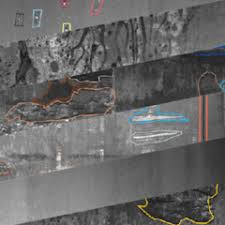
Steel Defect Detection is one field where even manual inspection is very difficult and tend to be inefficient. To make this better and efficient, there is a lot of work done to make this process automated. Prime motive of our work is to detect the location and type of the defects. This is very crucial in improving the benchmark standards of manufacturing and improving the whole process of steel mining and production. MMdetection offers a variety of modules and methods for the task for both Detection as well as segmentation. With features like modular design, multiple frameworks, it is a good fit for our problem at hand. In several tasks MMdetection has outperformed Detectron, the de facto standard present at the time. We attempt to find a solution with the multi stage methods along with the other modules. The main advantage presented by MMDetection is the mechanism to easily implement different pipelines. We establish a benchmark to enhance on further and the results obtained are motivating to experiment with adding augmentation techniques.

1. Introduction and Motivation

Steel Manufacturing has played important role right from the industrial revolution times. Steel Mining and Production is a delicate process involving multiple steps requiring high precision. Manual Inspection has a scope of error making the process inefficient. The automation process is expected to enhance the process. Over the years, there has been a surge in the amount of data that is available to process. Steel Manufacturing giants like Severstal have taken the responsibility of publishing large scale dataset for the same. They have come up with a dataset consisting of 14369 images (12568 train, and 1801 test). The objective is to come up with 4-ary classification. Each image is categorized into 4 different class id. The problem can be thus decomposed into two independent challenges, namely, (a) Defect Type Detection and (b) Defect Localization.

Dataset Information:

Training Dataset contains 12568 images of steel surfaces. Out of which 6666 images have some defects belonging to class id (1,2,3,4) and the other 5902 images have no defect in them. Below image has all the kinds of defects that is present in the steel surface dataset.



Source : Severstal: Steel Defect Detection, Kaggle.

Motivation behind this project was the complexity of the process. Class 1, Class 2 defect is very much similar to the point that manual inspection is prone to error. This very situation is a critical problem to be addressed. With earlier approaches involving Machine Learning Algorithms, it was evident that the solution to the problem could indeed lie in using these more often. Previous works dealing with Encoder-Decoder network approach gave a good reference to start with and helped in getting a better understanding of the complexity of the project. We leverage the modular approach of MMDetection and try to experiment with various pipelines for a detailed comparison in terms of the metric value.

1. Network Architecture

MMdetection has many individual components to finetune the complete network. Following are the individual components that form the MMdetection network:

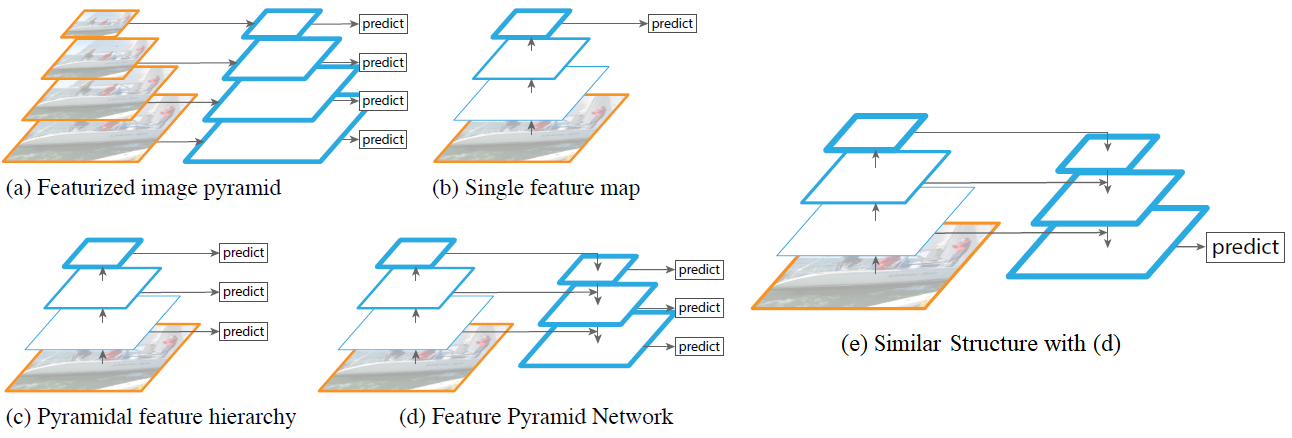
Backbone: Images are converted to feature maps here. We mainly used ResNet, ResNext as our backbone for the task.

ResNet 50: ResNet Architecture has solved the problem of vanishing gradients very wisely. By introduction of identity skip connections, ResNet allows the network to be deeper without compromising on the gradient scale despite repetitive multiplication of gradients. This has been groundbreaking for several computer vision tasks. 

Source: Deep Residual Network for Image Recognition

ResNext 101: ResNext is a smart variant of ResNet, that inculcates principle from Inception model. The idea is to split and then perform the transformation. But ResNext brings a simple approach to deal with large deep networks, by introducing cardinality.

Neck: Connects the Backbone with the heads by adding a set of refinements over the transformed feature maps. We used Feature Pyramid Networks along with the Region Proposal Network for the defect detection.



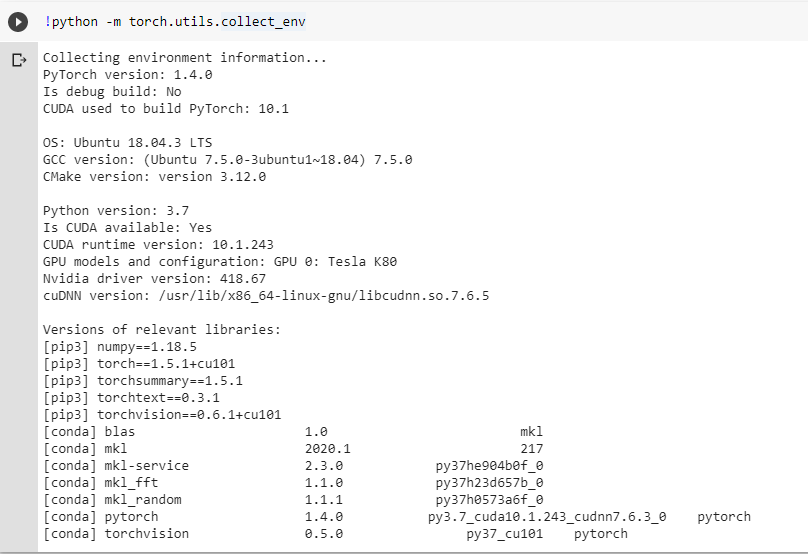
Source: Feature Pyramid Networks for Object Detection

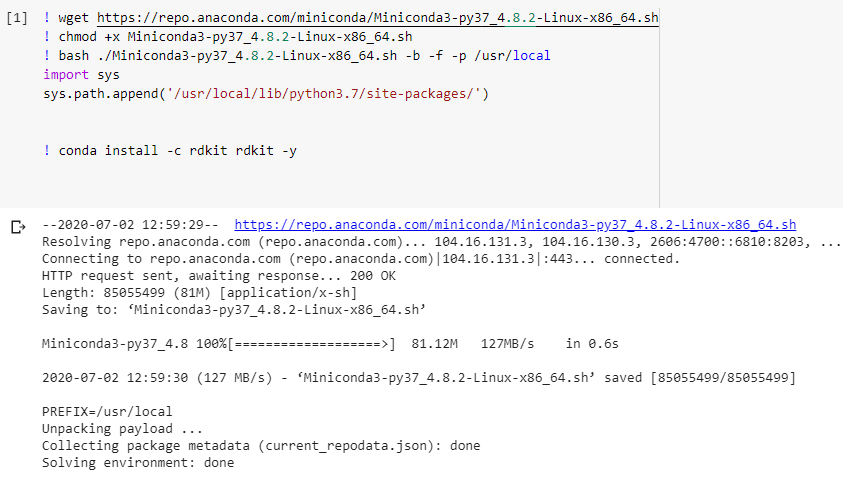
DenseHead: It identifies the feature maps for the presence of objects and denotes by bounding box. We used RPNHead for our project.

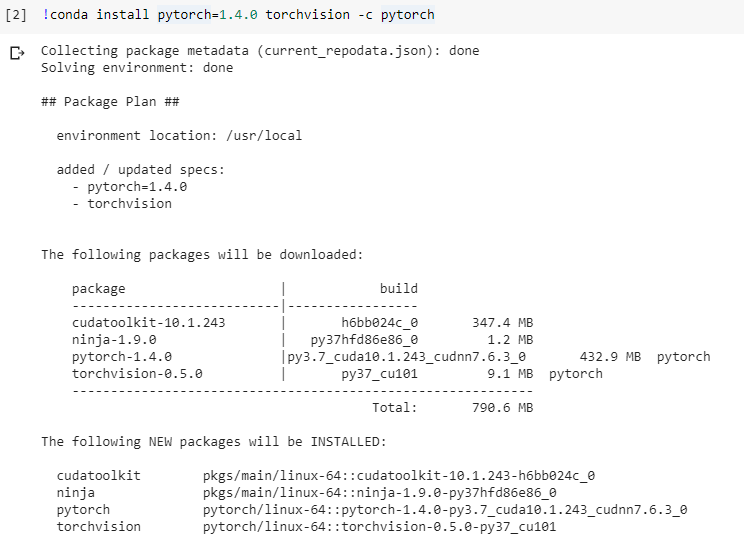
ROIExtractor: As the name suggests, this network extracts the region of interest. ROI Extractor network is responsible to extract those areas of steel surface where there is a probability of a defect

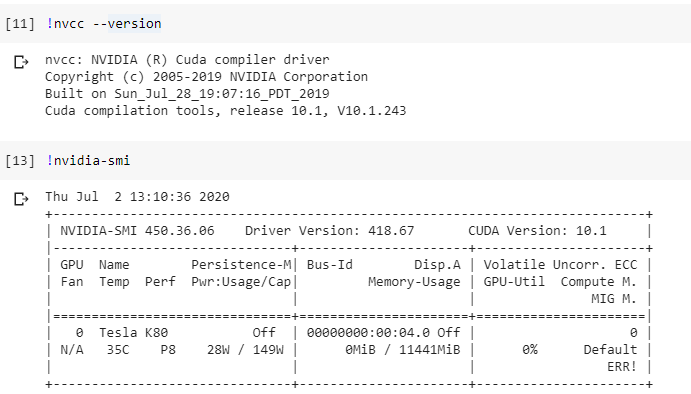
ROI Head: This component takes inputs from the ROI Extractor and performs the detection task. For our project it detects the type of defect present in the steel surface image by proposing a bounding box for each kind of defect where the probability of the class of the defect is high.

1. System Configuration



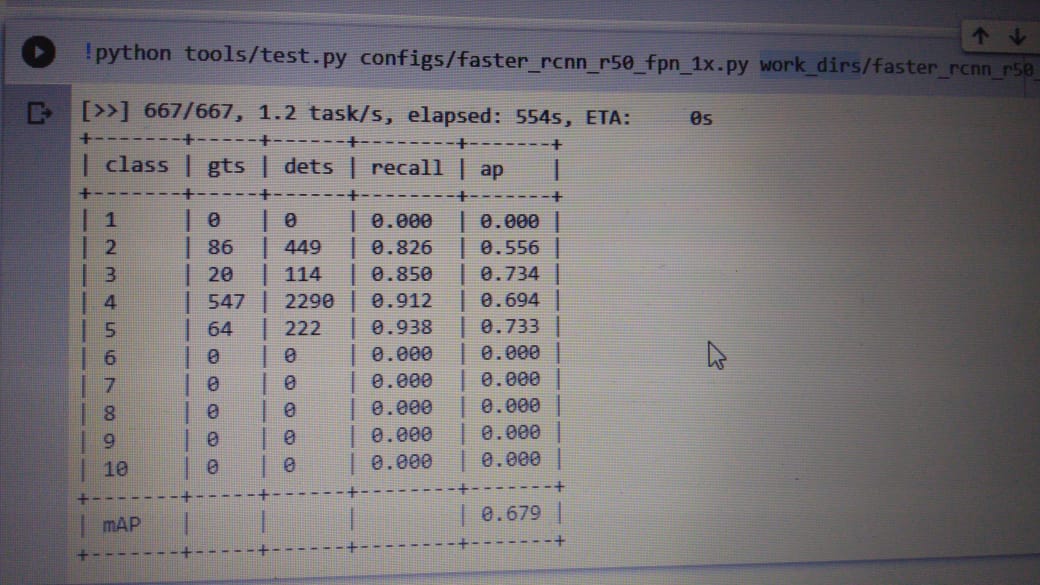






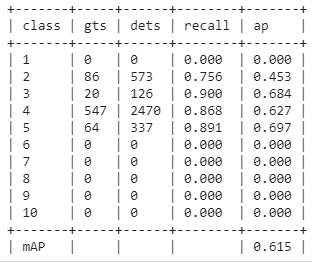
1. Experimental Results

5.1 Faster RCNN with ResNet50, FPN\_1x



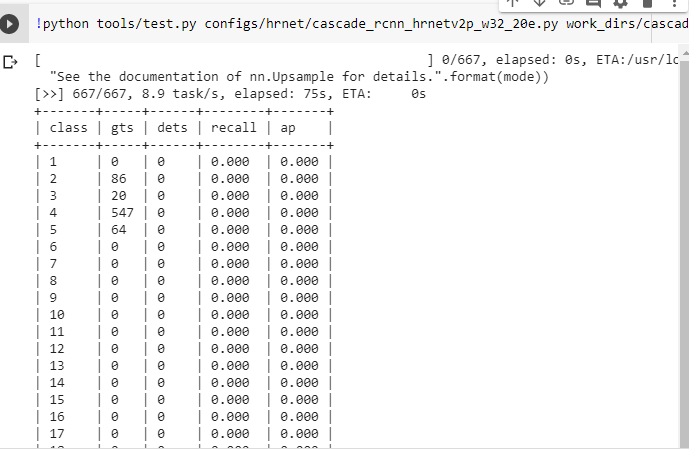
After 20 epochs, the mAP value obtained 67.9%

* 1. cascade\_rcnn\_x101\_64x4d\_fpn\_1x  
      with 10 epochs



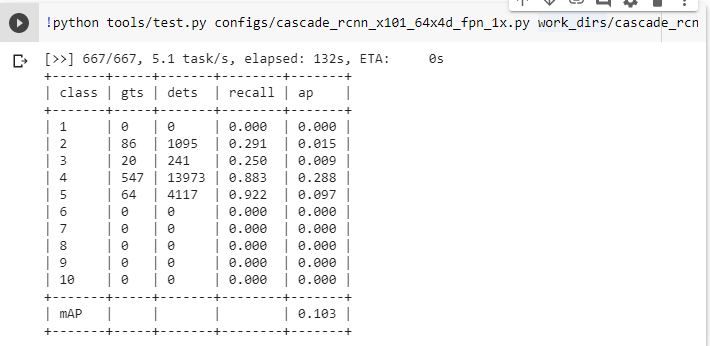
The performance after 10 epochs did not improve significantly. With highest mAP of 62.8.

5.3 hrnet/cascade\_rcnn\_hrnetv2p\_w32\_20e



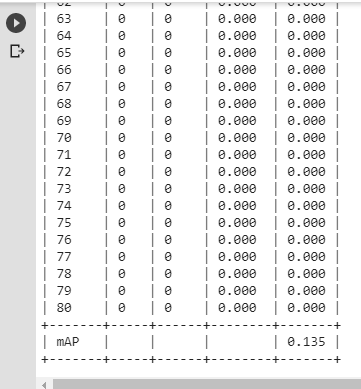
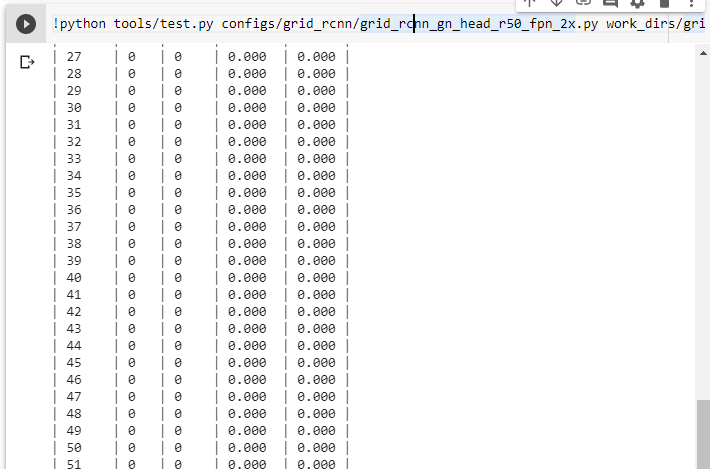
Since it is a Detection problem, this model collapsed and didn’t pick up well.

5.4 Cascade\_rcnn\_101

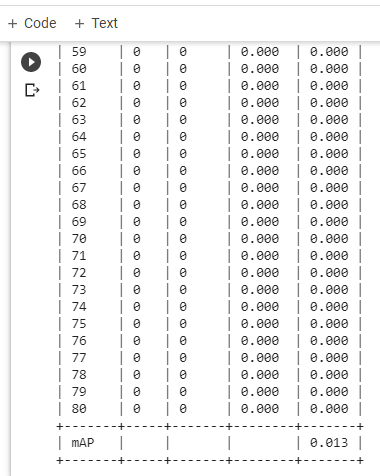
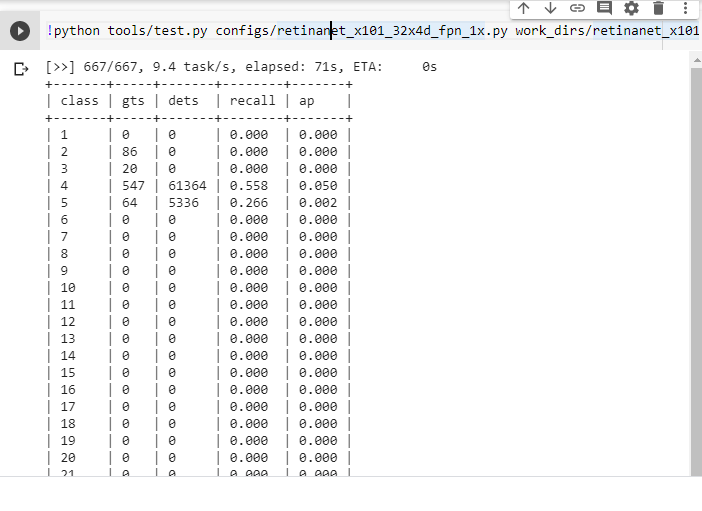


Very slow increase in the mAP value so this model was discarded as it didn’t add much mAP value.

5.5 Grid\_rcnn (discarded due to 0.135 mAP)



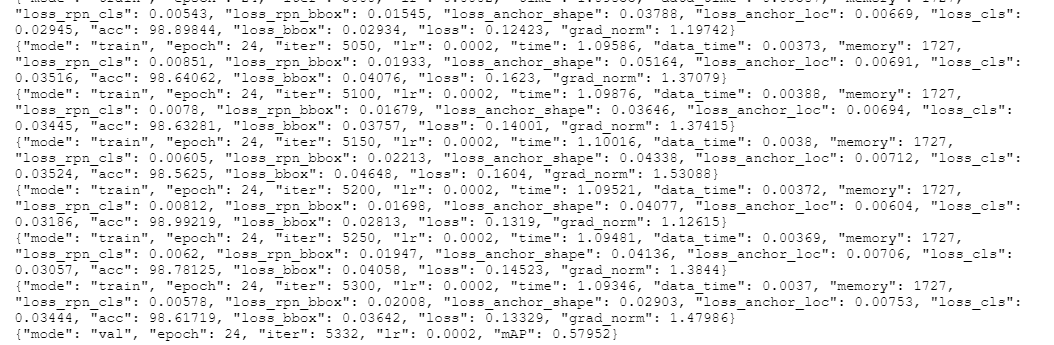
5.6 Retinanet\_101,32x4d,fpn\_1x



0.013 – not good enough to go ahead

5.7 Faster RCNN with multiscale and Guided Anchoring

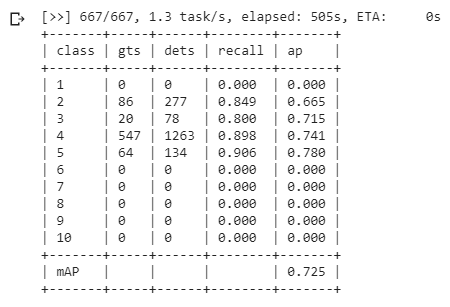
* Ran for 24 epochs.
* After epoch#12, introduced Guided Anchoring
* After epoch#16, introduced Multiscale
* Increased Anchor Stride to 96
* #img\_per\_gpu = 1



57.9 train mAP, and 62.3 test mAP value

5.8 Cascade RCNN with Guided Anchoring, Multiscale

* #img\_per\_gpu = 1
* Anchor Stride – unchanged
* Train mAP Value



1. Possible Extensions

For further enhancement, Augmentation techniques will play a crucial role. Changing the Flip Ratio didn’t improve much on the mAP value. But, the Albumentation seems to be promising and worth going on that direction.

Another approach could be to change the optimizer, as professor had suggested, I further studied SGD-R and the results are motivating to try for our project as well.

Deformable Convolution is a promising approach so we can further enhance the detection by adding deformable convolution particularly at deeper stages.

And finally, since we now know about the different backbones that are effective we can also try to custom design the architecture involving optimal configurations.

1. Conclusions

We experimented with the different configurations offered by MMDetection. Out of which Cascade RCNN was found to be the best among others. And rightly so, as it is a multi stage detector, refining the representations at each stage. Due to Colab GPU limits, we could not train for much epochs, but we studied all the hyperparameters involved like img\_per\_gpu, pos\_neg\_ub, Flip Ratio apart from the learning rate, optimizer, beta and gamma.

1. References
   1. MMDetection: Open MMLab Detection Toolbox and Benchmark
   2. Deep Residual Learning for Image Recognition
   3. Evaluating ResNext Model Architecture for Image Classification
   4. Feature Pyramid Networks for Object Detection
   5. Faster R-CNN: Towards Real – Time Object Detection with Region Proposal Networks
   6. SGDR - Stochastic Gradient Descent with Warm Restarts
   7. Albumentations : fast and flexible image augmentations
   8. Deformable Convolution Networks ([arXiv:1703.06211](https://arxiv.org/abs/1703.06211))
   9. Cascade R-CNN : High Quality Object Detection and Instance Segmentation
   10. Faster R-CNN **(**1506.01497)

# 

Appendix

1. Cascade RCNN

model = dict(

type='CascadeRCNN',

num\_stages=3,

pretrained='open-mmlab://resnext101\_32x4d',

backbone=dict(

type='ResNeXt',

depth=101,

groups=32,

base\_width=4,

num\_stages=4,

out\_indices=(0, 1, 2, 3),

frozen\_stages=1,

norm\_cfg=dict(type='BN', requires\_grad=True),

),

neck=dict(

type='FPN',

in\_channels=[256, 512, 1024, 2048],

out\_channels=256,

num\_outs=5),

rpn\_head=dict(

type='GARPNHead',

in\_channels=256,

feat\_channels=256,

octave\_base\_scale=8,

scales\_per\_octave=3,

octave\_ratios=[0.5, 1.0, 2.0],

anchor\_strides=[4, 8, 16, 32, 64],

anchor\_base\_sizes=None,

anchoring\_means=[.0, .0, .0, .0],

anchoring\_stds=[0.07, 0.07, 0.14, 0.14],

target\_means=(.0, .0, .0, .0),

target\_stds=[0.07, 0.07, 0.11, 0.11],

loc\_filter\_thr=0.01,

loss\_loc=dict(

type='FocalLoss',

use\_sigmoid=True,

gamma=2.0,

alpha=0.25,

loss\_weight=1.0),

loss\_shape=dict(type='BoundedIoULoss', beta=0.2, loss\_weight=1.0),

loss\_cls=dict(

type='CrossEntropyLoss', use\_sigmoid=True, loss\_weight=1.0),

loss\_bbox=dict(type='SmoothL1Loss', beta=1.0, loss\_weight=1.0)),

bbox\_roi\_extractor=dict(

type='SingleRoIExtractor',

roi\_layer=dict(type='RoIAlign', out\_size=7, sample\_num=2),

out\_channels=256,

featmap\_strides=[4, 8, 16, 32]),

bbox\_head=[

dict(

type='SharedFCBBoxHead',

num\_fcs=2,

in\_channels=256,

fc\_out\_channels=1024,

roi\_feat\_size=7,

num\_classes=11,

target\_means=[0., 0., 0., 0.],

target\_stds=[0.1, 0.1, 0.2, 0.2],

reg\_class\_agnostic=False,

loss\_cls=dict(

type='CrossEntropyLoss', use\_sigmoid=False, loss\_weight=1.0),

loss\_bbox=dict(type='SmoothL1Loss', beta=1.0, loss\_weight=1.0)),

dict(

type='SharedFCBBoxHead',

num\_fcs=2,

in\_channels=256,

fc\_out\_channels=1024,

roi\_feat\_size=7,

num\_classes=11,

target\_means=[0., 0., 0., 0.],

target\_stds=[0.05, 0.05, 0.1, 0.1],

reg\_class\_agnostic=False,

loss\_cls=dict(

type='CrossEntropyLoss', use\_sigmoid=False, loss\_weight=1.0),

loss\_bbox=dict(type='SmoothL1Loss', beta=1.0, loss\_weight=1.0)),

dict(

type='SharedFCBBoxHead',

num\_fcs=2,

in\_channels=256,

fc\_out\_channels=1024,

roi\_feat\_size=7,

num\_classes=11,

target\_means=[0., 0., 0., 0.],

target\_stds=[0.033, 0.033, 0.067, 0.067],

reg\_class\_agnostic=False,

loss\_cls=dict(

type='CrossEntropyLoss', use\_sigmoid=False, loss\_weight=1.0),

loss\_bbox=dict(type='SmoothL1Loss', beta=1.0, loss\_weight=1.0))

])

# model training and testing settings

train\_cfg = dict(

rpn=dict(

ga\_assigner=dict(

type='ApproxMaxIoUAssigner',

pos\_iou\_thr=0.7,

neg\_iou\_thr=0.3,

min\_pos\_iou=0.3,

ignore\_iof\_thr=-1),

ga\_sampler=dict(

type='RandomSampler',

num=256,

pos\_fraction=0.5,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=False),

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.7,

neg\_iou\_thr=0.3,

min\_pos\_iou=0.3,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=256,

pos\_fraction=0.5,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=False),

allowed\_border=-1,

pos\_weight=-1,

center\_ratio=0.2,

ignore\_ratio=0.5,

debug=False),

rpn\_proposal=dict(

nms\_across\_levels=False,

nms\_pre=2000,

nms\_post=2000,

max\_num=300,

nms\_thr=0.7,

min\_bbox\_size=0),

rcnn=[

dict(

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.5,

neg\_iou\_thr=0.5,

min\_pos\_iou=0.5,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=512,

pos\_fraction=0.25,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=True),

pos\_weight=-1,

debug=False),

dict(

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.6,

neg\_iou\_thr=0.6,

min\_pos\_iou=0.6,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=512,

pos\_fraction=0.25,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=True),

pos\_weight=-1,

debug=False),

dict(

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.7,

neg\_iou\_thr=0.7,

min\_pos\_iou=0.7,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=512,

pos\_fraction=0.25,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=True),

pos\_weight=-1,

debug=False)

],

stage\_loss\_weights=[1, 0.5, 0.25])

test\_cfg = dict(

rpn=dict(

nms\_across\_levels=False,

nms\_pre=1000,

nms\_post=1000,

max\_num=1000,

nms\_thr=0.7,

min\_bbox\_size=0),

rcnn=dict(

score\_thr=0.05, nms=dict(type='nms', iou\_thr=0.5), max\_per\_img=100))

# dataset settings

dataset\_type = 'CustomDataset'

data\_root = 'data/'

img\_norm\_cfg = dict(

mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to\_rgb=True)

train\_pipeline = [

dict(type='LoadImageFromFile'),

dict(type='LoadAnnotations', with\_bbox=True),

dict(type='Resize',multiscale\_mode='range',img\_scale=[(1333, 640), (1333, 960)], keep\_ratio=True),

dict(type='RandomFlip', flip\_ratio=0.5),

dict(type='Normalize', \*\*img\_norm\_cfg),

dict(type='Pad', size\_divisor=32),

dict(type='DefaultFormatBundle'),

dict(type='Collect', keys=['img', 'gt\_bboxes', 'gt\_labels']),

]

test\_pipeline = [

dict(type='LoadImageFromFile'),

dict(

type='MultiScaleFlipAug',

img\_scale=[(1333, 640), (1333, 960)],

flip=False,

transforms=[

dict(type='Resize',multiscale\_mode='range', keep\_ratio=True),

dict(type='RandomFlip'),

dict(type='Normalize', \*\*img\_norm\_cfg),

dict(type='Pad', size\_divisor=32),

dict(type='ImageToTensor', keys=['img']),

dict(type='Collect', keys=['img']),

])

]

data = dict(

imgs\_per\_gpu=1,

workers\_per\_gpu=1,

train=dict(

type=dataset\_type,

ann\_file=data\_root + 'train\_80\_mmdetection.pkl',

img\_prefix=data\_root + 'train/',

pipeline=train\_pipeline),

val=dict(

type=dataset\_type,

ann\_file=data\_root + 'val\_10\_mmdetection.pkl',

img\_prefix=data\_root + 'train/',

pipeline=test\_pipeline),

test=dict(

type=dataset\_type,

ann\_file=data\_root + 'test\_10\_mmdetection.pkl',

img\_prefix=data\_root + 'train/',

pipeline=test\_pipeline))

evaluation = dict(interval=1, metric='mAP')

# optimizer

optimizer = dict(type='SGD', lr=0.001, momentum=0.9, weight\_decay=0.0001)

optimizer\_config = dict(grad\_clip=dict(max\_norm=35, norm\_type=2))

# learning policy

lr\_config = dict(

policy='step',

warmup='linear',

warmup\_iters=500,

warmup\_ratio=1.0 / 3,

step=[8, 11])

checkpoint\_config = dict(interval=1)

# yapf:disable

log\_config = dict(

interval=64,

hooks=[

dict(type='TextLoggerHook'),

# dict(type='TensorboardLoggerHook')

])

# yapf:enable

# runtime settings

total\_epochs = 20

dist\_params = dict(backend='nccl')

log\_level = 'INFO'

work\_dir = './work\_dirs/cascade\_rcnn\_x101\_32x4d\_fpn\_1x'

load\_from = None

resume\_from = None

workflow = [('train', 1)]

1. faster\_rcnn\_x101\_32x4d\_fpn\_1x

# model settings

model = dict(

type='FasterRCNN',

pretrained='open-mmlab://resnext101\_32x4d',

backbone=dict(

type='ResNeXt',

depth=101,

groups=32,

base\_width=4,

num\_stages=4,

out\_indices=(0, 1, 2, 3),

frozen\_stages=1,

norm\_cfg=dict(type='BN', requires\_grad=True),

style='pytorch'),

neck=dict(

type='FPN',

in\_channels=[256, 512, 1024, 2048],

out\_channels=256,

num\_outs=5),

rpn\_head=dict(

type='RPNHead',

in\_channels=256,

feat\_channels=256,

anchor\_scales=[8],

anchor\_ratios=[0.5, 1.0, 2.0],

anchor\_strides=[4, 8, 16, 32, 64],

target\_means=[.0, .0, .0, .0],

target\_stds=[1.0, 1.0, 1.0, 1.0],

loss\_cls=dict(

type='CrossEntropyLoss', use\_sigmoid=True, loss\_weight=1.0),

loss\_bbox=dict(type='SmoothL1Loss', beta=1.0 / 9.0, loss\_weight=1.0)),

bbox\_roi\_extractor=dict(

type='SingleRoIExtractor',

roi\_layer=dict(type='RoIAlign', out\_size=7, sample\_num=2),

out\_channels=256,

featmap\_strides=[4, 8, 16, 32]),

bbox\_head=dict(

type='SharedFCBBoxHead',

num\_fcs=2,

in\_channels=256,

fc\_out\_channels=1024,

roi\_feat\_size=7,

num\_classes=11,

target\_means=[0., 0., 0., 0.],

target\_stds=[0.1, 0.1, 0.2, 0.2],

reg\_class\_agnostic=False,

loss\_cls=dict(

type='CrossEntropyLoss', use\_sigmoid=False, loss\_weight=1.0),

loss\_bbox=dict(type='SmoothL1Loss', beta=1.0, loss\_weight=1.0)))

# model training and testing settings

train\_cfg = dict(

rpn=dict(

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.7,

neg\_iou\_thr=0.3,

min\_pos\_iou=0.3,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=256,

pos\_fraction=0.5,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=False),

allowed\_border=0,

pos\_weight=-1,

debug=False),

rpn\_proposal=dict(

nms\_across\_levels=False,

nms\_pre=2000,

nms\_post=2000,

max\_num=2000,

nms\_thr=0.7,

min\_bbox\_size=0),

rcnn=dict(

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.5,

neg\_iou\_thr=0.5,

min\_pos\_iou=0.5,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=512,

pos\_fraction=0.25,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=True),

pos\_weight=-1,

debug=False))

test\_cfg = dict(

rpn=dict(

nms\_across\_levels=False,

nms\_pre=1000,

nms\_post=1000,

max\_num=1000,

nms\_thr=0.7,

min\_bbox\_size=0),

rcnn=dict(

score\_thr=0.05, nms=dict(type='nms', iou\_thr=0.5), max\_per\_img=100)

# soft-nms is also supported for rcnn testing

# e.g., nms=dict(type='soft\_nms', iou\_thr=0.5, min\_score=0.05)

)

# dataset settings

dataset\_type = 'CustomDataset'

data\_root = 'data/'

img\_norm\_cfg = dict(

mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to\_rgb=True)

train\_pipeline = [

dict(type='LoadImageFromFile'),

dict(type='LoadAnnotations', with\_bbox=True),

dict(type='Resize', img\_scale=(1333, 800), keep\_ratio=True),

dict(type='RandomFlip', flip\_ratio=0.5),

dict(type='Normalize', \*\*img\_norm\_cfg),

dict(type='Pad', size\_divisor=32),

dict(type='DefaultFormatBundle'),

dict(type='Collect', keys=['img', 'gt\_bboxes', 'gt\_labels']),

]

test\_pipeline = [

dict(type='LoadImageFromFile'),

dict(

type='MultiScaleFlipAug',

img\_scale=(1333, 800),

flip=False,

transforms=[

dict(type='Resize', keep\_ratio=True),

dict(type='RandomFlip'),

dict(type='Normalize', \*\*img\_norm\_cfg),

dict(type='Pad', size\_divisor=32),

dict(type='ImageToTensor', keys=['img']),

dict(type='Collect', keys=['img']),

])

]

data = dict(

imgs\_per\_gpu=1,

workers\_per\_gpu=1,

train=dict(

type=dataset\_type,

ann\_file=data\_root + 'train\_80\_mmdetection.pkl',

img\_prefix=data\_root + 'train/',

pipeline=train\_pipeline),

val=dict(

type=dataset\_type,

ann\_file=data\_root + 'val\_10\_mmdetection.pkl',

img\_prefix=data\_root + 'train/',

pipeline=test\_pipeline),

test=dict(

type=dataset\_type,

ann\_file=data\_root + 'test\_10\_mmdetection.pkl',

img\_prefix=data\_root + 'train/',

pipeline=test\_pipeline))

evaluation = dict(interval=1, metric='mAP')

# optimizer

optimizer = dict(type='SGD', lr=0.001, momentum=0.9, weight\_decay=0.0001)

optimizer\_config = dict(grad\_clip=dict(max\_norm=35, norm\_type=2))

# learning policy

lr\_config = dict(

policy='step',

warmup='linear',

warmup\_iters=500,

warmup\_ratio=1.0 / 3,

step=[16, 19])

checkpoint\_config = dict(interval=1)

# yapf:disable

log\_config = dict(

interval=50,

hooks=[

dict(type='TextLoggerHook'),

# dict(type='TensorboardLoggerHook')

])

# yapf:enable

# runtime settings

total\_epochs = 20

dist\_params = dict(backend='nccl')

log\_level = 'INFO'

work\_dir = './work\_dirs/faster\_rcnn\_x101\_32x4d\_fpn\_1x'

load\_from = None

resume\_from = None

workflow = [('train', 1)]

NOTE: These are the config files of the 2 most optimal models among other that were implemented.