Deteksi dan Klasifikasi Objek pada Citra Sel Darah menggunakan Metode Berbasis *Deep Learning*

Samatha Marhaendra Putra
19/444071/TK/49267

Table of contents

01 Latar Belakang

Latar Belakang

- Proyek terkait deteksi dan klasifikasi sel darah merah dan sel darah putih pada citra medis
- Penting karena kemampuan dalam pendeteksian sel darah yang cepat dan akurat dapat mendukung dalam pengambilan keputusan/tindakan untuk menyelesaikan permasalahan-permasalahan yang ada di dunia medis (misal: deteksi penyakit)

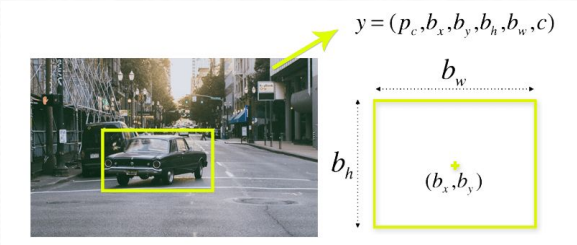
02 Solusi yang Dipakai

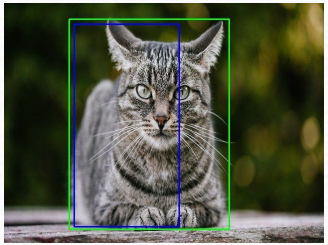
Algoritma YOLO: *Overview*

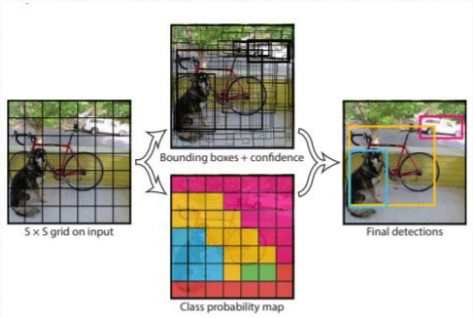
- Algoritma YOLO (You Only Look Once) merupakan algoritma yang mendukung *task* terkait *object detection* dengan menganggap *task* tersebut sebagai permasalahan regresi, yang mana mampu menyediakan probabilitas kelas dan *bounding boxes* yang dideteksi dari gambar.
- Algoritma ini menerapkan *convolutional neural networks* (CNN) untuk melakukan pendeteksian objek.
- Sesuai namanya, algoritma ini hanya memerlukan *single forward propagation* melalui *neural network* untuk mendeteksi objek. Ini berarti, prediksi dari keseluruhan citra dapat dilakukan dengan menjalankan algoritma sebanyak satu kali saja.

Algoritma YOLO: Cara Kerja

- Algoritma YOLO menerapkan tiga teknik berikut:
  - *Residual blocks*
  - *Bounding box regression*
  - *Intersection Over Union* (IOU)

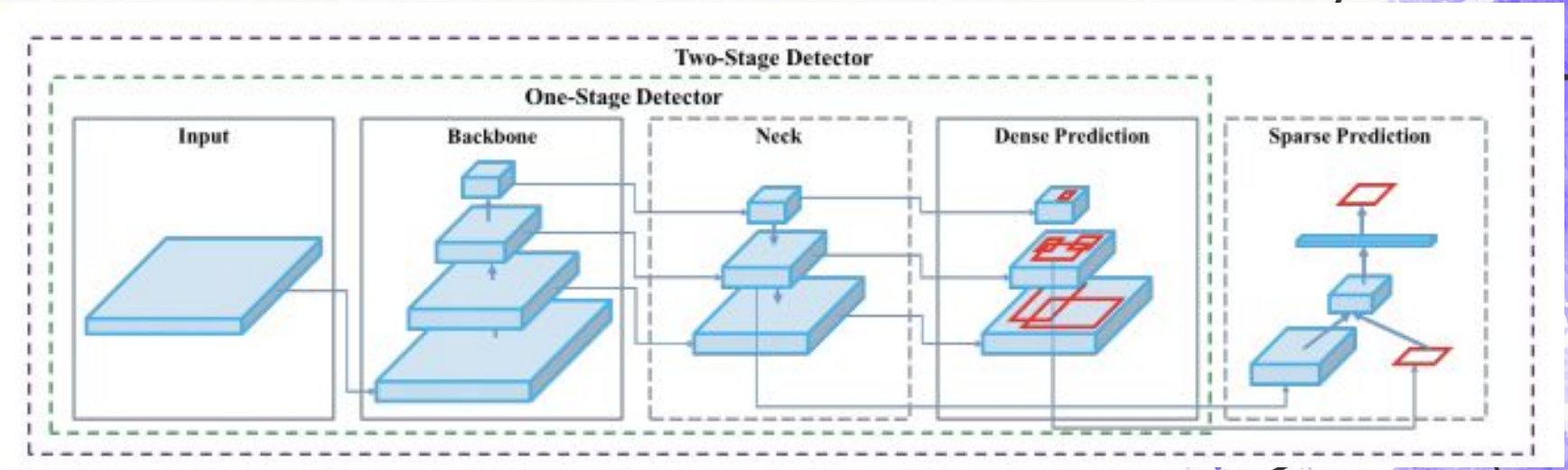






Algoritma YOLO: Arsitektur

3 bagian utama: ***Backbone***, ***Neck***, dan ***Head***

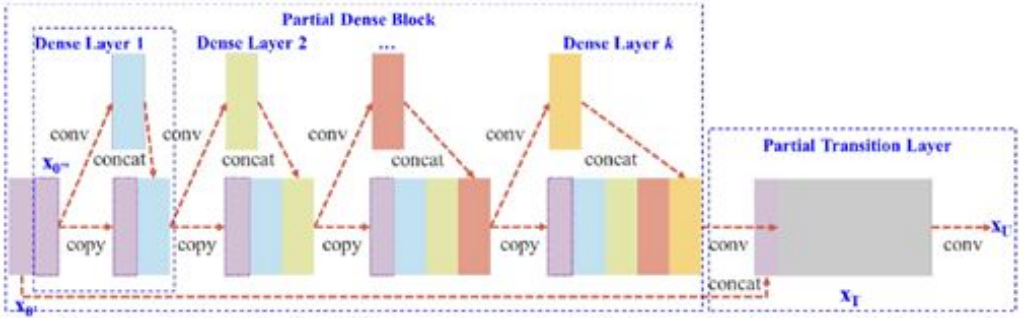


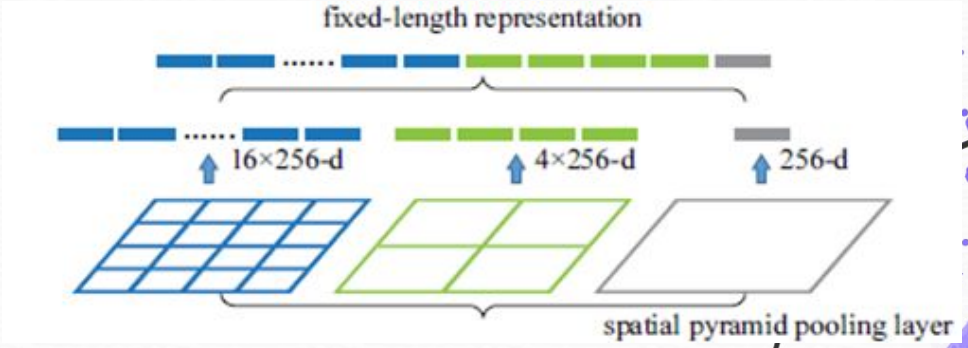
sumber: *YOLOv4 Official Paper*

Algoritma YOLO: Arsitektur

Bagian pertama: ***Backbone***

- Menggunakan **CSPDarknet32** dan **SPP *Layer***

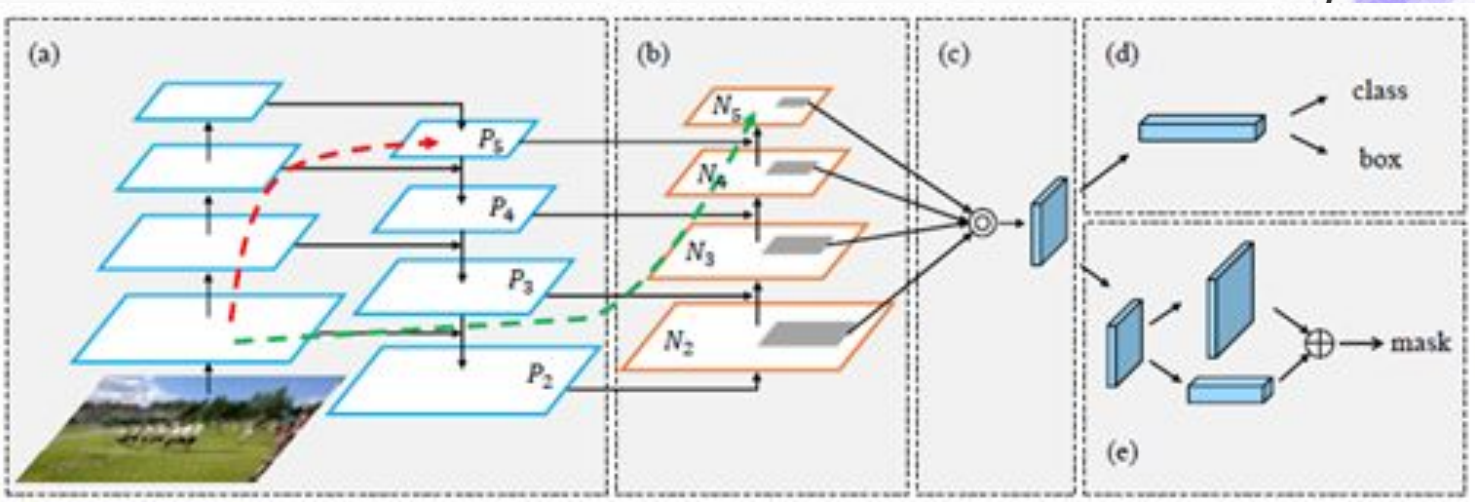




Algoritma YOLO: Arsitektur

Bagian kedua: ***Neck***

- Menggunakan *Path Aggregation Network* (PAN)



Algoritma YOLO: Arsitektur

Bagian ketiga: ***Head***

- Pada bagian ini dilakukan *prediction task*, di mana terdapat pembuatan *bounding boxes* beserta dengan kelas prediksinya.

03 Hasil

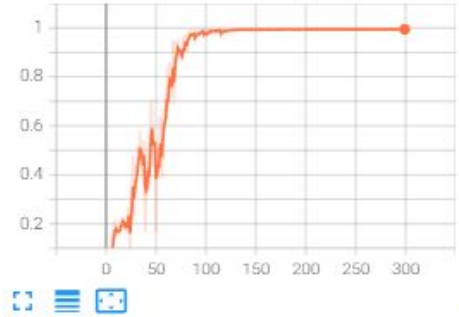
Hasil

```
# training the custom data
%%time
!python yolov5/train.py --img 256 --batch 64 --epochs 100 --data=rbcdet.yaml --cfg=yolov5/models/yolov5s.yaml --name rbcdet_model
```

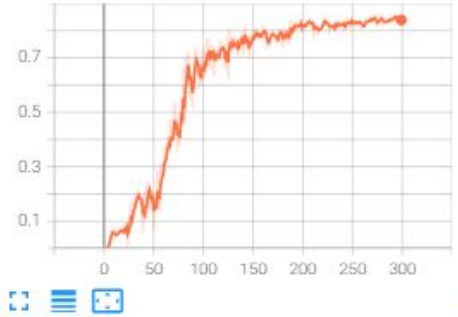
```
Validating yolov5/runs/train/rbcdet_model/weights/best.pt...
Fusing layers...
YOLOv5s summary: 213 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  3.54it/s]
                 all         10        241      0.979      0.983      0.994      0.855
                 rbc         10        231      0.976      0.965      0.993      0.808
                 wbc         10         10      0.982          1      0.995      0.902
Results saved to yolov5/runs/train/rbcdet_model
CPU times: user 4.61 s, sys: 524 ms, total: 5.13 s
Wall time: 8min 10s
```

Hasil

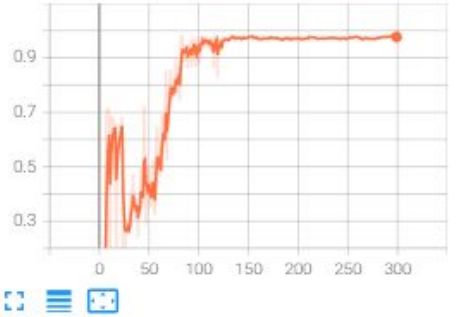
metrics/mAP_0.5
tag: metrics/mAP_0.5



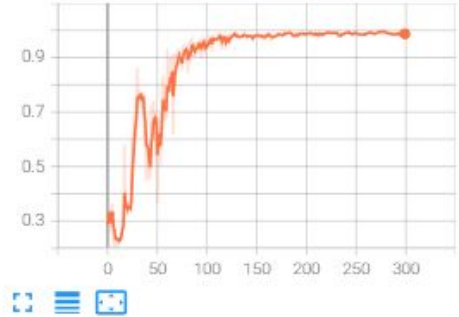
metrics/mAP_0.5:0.95
tag: metrics/mAP_0.5:0.95



metrics/precision
tag: metrics/precision



metrics/recall
tag: metrics/recall



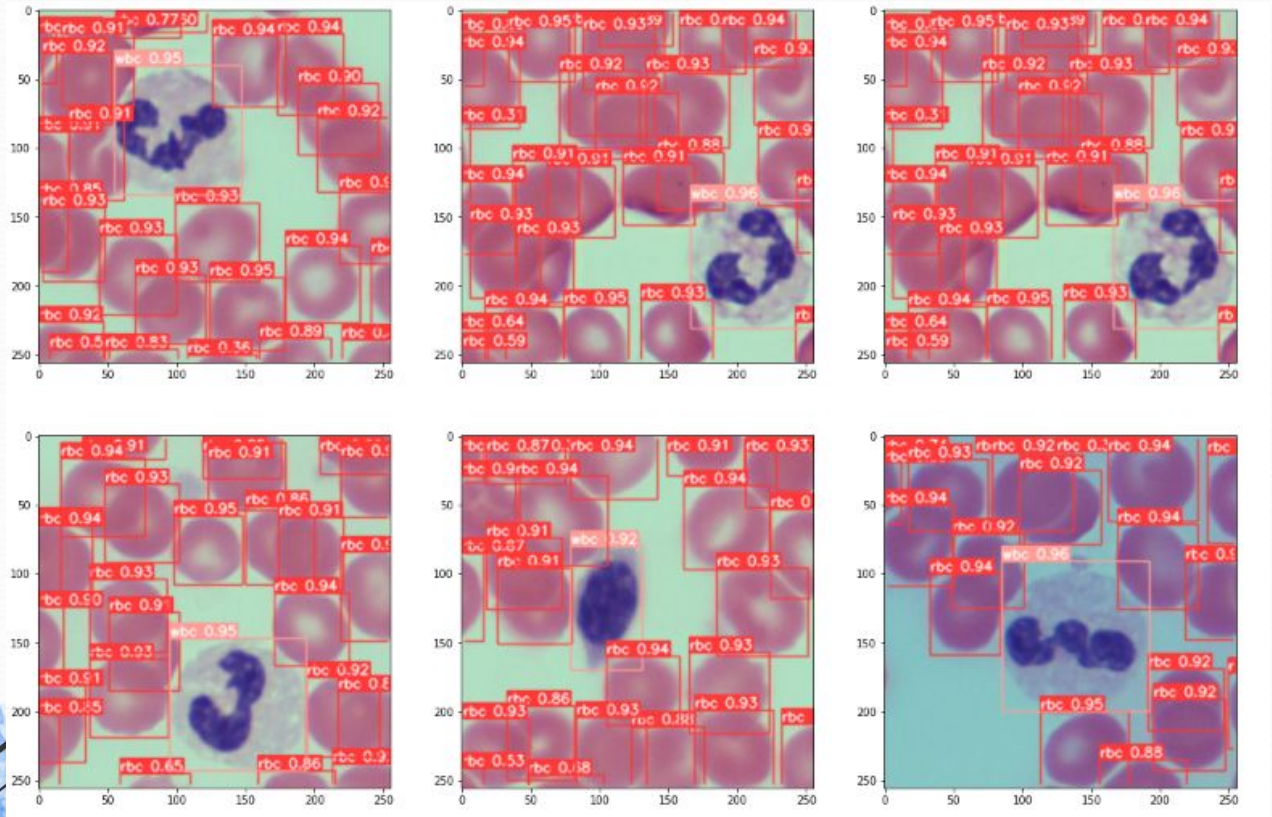
Hasil

```
[ ] # predicting set of images
    !python yolov5/detect.py --source /content/split/images/val/ --imgsz 256 --weights '/content/yolov5/runs/train/rbcdet_model/weights/best.pt' --name '/content/inference/output' --line-thickness 1
```

```
detect: weights=['/content/yolov5/runs/train/rbcdet_model/weights/best.pt'], source=/content/split/images/val/, data=yolov5/data/coco128.yaml, imgsz=[256, 256], conf_thres=0.25, iou_thres=0.45, m
YOLOv5 🚀 v6.1-242-ga80dd66 Python-3.7.13 torch-1.11.0+cu113 CUDA:0 (Tesla T4, 15110MiB)

Fusing layers...
YOLOv5s summary: 213 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
image 1/10 /content/split/images/val/image-22.png: 256x256 25 rbcs, 1 wbc, Done. (0.009s)
image 2/10 /content/split/images/val/image-23.png: 256x256 23 rbcs, 1 wbc, Done. (0.008s)
image 3/10 /content/split/images/val/image-24.png: 256x256 28 rbcs, 1 wbc, Done. (0.008s)
image 4/10 /content/split/images/val/image-25.png: 256x256 26 rbcs, 1 wbc, Done. (0.008s)
image 5/10 /content/split/images/val/image-31.png: 256x256 19 rbcs, 1 wbc, Done. (0.008s)
image 6/10 /content/split/images/val/image-33.png: 256x256 22 rbcs, 1 wbc, Done. (0.008s)
image 7/10 /content/split/images/val/image-35.png: 256x256 26 rbcs, 1 wbc, Done. (0.008s)
image 8/10 /content/split/images/val/image-69.png: 256x256 35 rbcs, 1 wbc, Done. (0.008s)
image 9/10 /content/split/images/val/image-7.png: 256x256 28 rbcs, 1 wbc, Done. (0.008s)
image 10/10 /content/split/images/val/image-91.png: 256x256 25 rbcs, 1 wbc, Done. (0.008s)
Speed: 0.2ms pre-process, 8.2ms inference, 0.9ms NMS per image at shape (1, 3, 256, 256)
Results saved to /content/inference/output
```

Hasil



04 Demo

Tautan Kode Program:
https://colab.research.google.com/drive/1iffS7kgjmsekK5A_gY5vdxAqEaF9Icx-?usp=sharing

Terima Kasih