ISSN (Online) 2747-0563
Volume 1 Tahun 2020

Identifikasi Penggunaan Masker Menggunakan Algoritma CNN YOLOv3-Tiny

Dicky Giancini[1], Eva Yulia Puspaningrum[2*], Yisti Vita Via[3]

[1,2,3] Informatika, Universitas Pembangunan Nasional "Veteran" Jawa Timur
[1]dicky.giancini.123@gmail.com
[3]yistivita.if@upnjatim.ac.id

*Corresponding author email: evapuspaningrum.if@upnjatim.ac.id

***Abstrak*— Dengan adanya pandemi COVID-19, maka protokol kesehatan seperti menjaga jarak, mencuci tangan dengan sabun secara rutin, dan menggunakan masker merupakan arahan yang diberikan oleh World Health Organization (WHO) untuk mengurangi resiko penyebaran virus COVID-19. Tetapi dengan adanya arahan tersebut, masih ditemukan orang yang tidak menggunakan masker di tempat umum. Munculnya trending *Machine* Learning dan *Deep Learning* menciptakan berbagai riset untuk menemukan metode – metode baru dan arsitektur mutakhir seperti YOLO (You Only Look Once). YOLO merupakan arsitektur detector yang diklaim sebagai "*fastest deep learning object detector*" yang mengorbankan akurasi dengan kecepatan. Dengan menggunakan YOLOv3, kita dapat menciptakan deteksi masker yang *robust* dan presisi untuk mendeteksi apakah seseorang yang tampak pada gambar / kamera bisa dikenali menggunakan masker atau tidak. Tetapi dengan tersedianya YOLOv3 yang memerlukan arsitektur komputer yang berat, maka sistem yang lebih lama akan kesulitan menggunakan arsitektur tersebut. Maka menggunakan YOLOv3-tiny dapat menjadi solusi untuk arsitektur komputer yang lebih lama. Menggunakan *backbone* dari YOLOv3-tiny, kita dapat mendeteksi objek masker dengan arsitektur yang lebih kecil. Untuk memaksimalkan deteksi, maka dilakukan augmentasi data dengan proses *flip, cropping,* dan rotasi sehingga memberikan variasi kepada dataset, sehingga dapat mengurangi *overfitting*. Hasil dari augmentasi data dengan YOLOv3-tiny mencipatakan deteksi dengan akurasi hingga 90% bahkan secara *real-time*.**

Kata Kunci*— Masker, YOLOv3, Darknet, *Deep Learning*, *Object Detector

I. PENDAHULUAN

COVID-19 merupakan jenis penyakit menular yang disebabkan oleh jenis coronavirus yang baru ditemukan. Penyebab virus ini masih belum ditemukan penyebabnya hingga terjadi wabah di Wuhan, Tiongkok, pada bulan Desember 2019 (WHO, 2020). Dengan adanya virus ini maka World Health Organization (WHO) memberlakukan protokol kesehatan untuk mencegah penularan coronavirus lebih luas, seperti menjaga jarak dari 1 meter atau lebih, membiasakan untuk mencuci tangan dengan sabun, dan menggunakan masker ketika diluar rumah. Tetapi kenyataannya masih banyak yang tidak menggunakan masker diluar rumah dan untuk mengetahuinya masih diperlukan pemeriksaan manual. Sehingga menciptakan sistem yang dapat mendeteksi penggunaan masker bisa menjadi opsi untuk peringatan dini.

Dalam beberapa tahun belakangan ini, *Deep Learning* menjadi topik hangat dan semakin digunakan untuk menciptakan sebuah deteksi benda, wajah, dan beberapa jenis lainnya. Beberapa detektor seperti Fast-RCNN, Faster-RCNN dan You Only Look Once (YOLO) menjadi *network* deteksi yang cukup signifikan dan menjadikan evolusi deteksi yang presisi namun ringan dalam beberapa aspek[1].

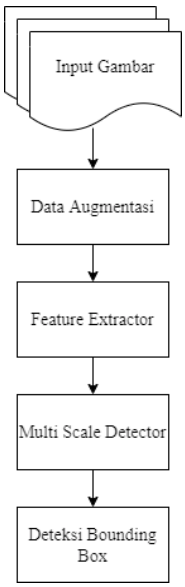
YOLO merupakan algoritma yang menggunakan *convolutional neural network* sebagai deteksi objek. YOLO diklaim sebagai arsitektur yang cepat dan sangat akurat[2]. Walaupun beberapa variabel dapat mempengaruhi keakurasian arsitektur, YOLO dapat menjadi pilihan yang baik untuk deteksi real time dengan memperkecil *loss* dari keakurasian. Gambaran mudah dari YOLO, YOLO dapat melakukan banyak deteksi objek, memprediksi class yang dibuat, dan mengidentifikasi lokasi dari objek tersebut.

YOLO sendiri memiliki beberapa kelemahan seperti kebutuhan arsitektur komputer yang mumpuni dan mutakhir, sehingga pada saat proses training untuk spesifikasi yang lebih lama akan memakan waktu yang sangat lama[1], mengingat ukuran ekstraktor feature yang sangat besar. Maka dari itu dibuatlah rancangan arsitektur yang lebih kecil dinamakan YOLO-tiny. Dari definisinya, versi tiny dari YOLO merupakan arsitektur yang dimana kedalaman *convolutional layers* dikurangi sehingga kita dapat mendapatkan kecepatan latih yang lebih cepat lagi. YOLO-tiny, sebagai arsitektur yang lebih kecil dari YOLO menggunakan layer konvolusi yang lebih sedikit namun dapat menunjukkan deteksi yang cukup[3].

II. METODOLOGI

YOLO adalah arsitektur yang cepat, dan beberapa struktur network yang dirancang merupakan modifikasi dari versi YOLO sebelumnya. Fitur yang diberikan seperti *multi-sclae detection*, fitur ekstraksi yang lebih kuat, dan beberapa perubahan pada *loss function* dengan mengorbankan kecepatan. Pada arsitektur YOLOv3-tiny, tahapan proses deteksi dimulai dengan input image yang akan kita ekstrak fiturnya, kemudian dari hasil ekstraksi fiturnya kita ubah menjadi 2 skala dengan dimensi 13x13 dan 26x26[4]. Berbeda dengan YOLOv3 yang lebih besar, skala yang digunakan yaitu sejumlah 3 dengan masing masing dimensi sebesar 13x13, 26x26 dan 52x52[5].

ISSN (Online) 2747-0563
Volume 1 Tahun 2020



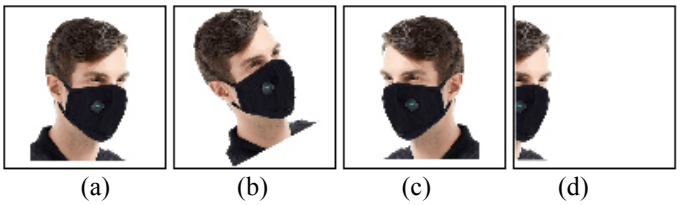
Gbr. 1 Alur YOLOv3-tiny (You Only Look Once), Dari Input Gambar Dilakukan Augmentasi Data, Lalu Ekstraksi Fitur yang Kemudian Dilakukan Deteksi Pada 2 Skala, dan Menggambar Bounding Box

Alur deteksi dari YOLO adalah pertama memasukkan *input* gambar atau dataset yang dimiliki, kemudian dilanjutkan dengan augmentasi data dengan opsi *flip, cropping,* dan rotasi, lalu dilakukan ekstraksi fitur menggunakan backbone Darknet, kemudian dilanjutkan dengan *Multi-Scale Detector* yang nantinya akan dicari deteksi objek menggunakan bounding box.

Pada alur deteksi YOLOv3, semua *bounding box* dan probabilitas dari keseluruhan input dilakukan dengan *convolutional network* tunggal [3], sehingga bisa dikatakan jika YOLOv3 merupakan *Fully Connected Layer* (FCN).

A. *Data Augmentation*

Augmentasi data sering digunakan dalam algoritma *deep learning* seperti YOLO, dimana memiliki fungsi untuk memberikan variasi pada dataset, dan juga untuk mengurangi *overfitting*[6]. Pada umumnya, mengumpulkan data yang cukup merupakan tantangan dari *neural network*[7], sehingga menambah variasi data secara otomatis akan menambah jumlah data yang bisa dilatih. Untuk deteksi objek, augmenasi data yang paling umum digunakan yaitu rotasi, *flip*, dan *cropping*. Pada Gbr. 2, data augmentasi yang dilakukan adalah dengan mengubah data mentah untuk dilakukan rotasi, *flipping,* dan juga *cropping*.



Gbr. 2. *(a) Dataset Penggunaan Masker Mentah, (b) Dataset Penggunaan Masker dengan Rotasi -35°, (c) Dataset Penggunaan Masker dengan opsi horizontal flip, (d) Dataset Penggunaan Masker dengan opsi cropping*

Beberapa tambahan yang dapat diimplementasikan pada augmentasi data, yaitu dengan adanya pemberian fungsi random untuk pengacakan fungsi augmentasi data dimana tidak hanya melakukan rotasi saja, tetapi dapat melakukan kombinasi antara rotasi,*flip,* dan juga *cropping*, sehingga dapat menjadi opsi yang baik dalam menambahkan variasi dataset[6]. Tidak hanya menambahkan variasi, tetapi juga dapat mengurangi dan meminimalisir error pada *network* tersebut[8]. Selain itu, augmentasi data juga dapat meminimalisir *overfitting*, dimana sebuah *network* akan menjadi ketergantungan dengan data yang kita latih, menyebabkan deteksi yang tidak efektif[8].

B. *Darknet-53*

Ekstraksi fitur yang digunakan pada YOLOv3 adalah *Darknet-53.* Berbeda dengan versi sebelumnya menggunakan *Darknet-19* yaitu *network* yang disuplai dengan 19 layer ditambah dengan 11 layer untuk deteksi objek, versi sebelumnya cukup kesulitan dalam mendeteksi objek kecil. Ini disebabkan karena efek dari *downsample* pada *input*. YOLOv2 juga memiliki kelemahan seperti tidak ada *residual block, skip connections,* dan *upsampling*[2]. Kelemahan tersebutlah yang akan ditutupi oleh *Darknet-53* [2].

Model dari *Darknet-53* ini mengkombinasikan *Residual Network* [2] dan dasar ekstraksi fitur dari YOLOv2, yaitu *Darknet-19,* menggunakan layer konvolusi dan *residual* dengan size berturut turut 1x1 dan 3x3. Sesuai dengan Tabel I, definisi dari 1x, 2x, 4x dan 8x adalah bentuk dari perhitungan *residual block* yang dilakukan berurutan dari 1 kali *residual block* layer konvolusi berukuran 1x1 dan layer konvolusi berukuran 3x3, yang kemudian juga dilakukan *skip connections* dari layer konvolusi sebelum terjadinya *residual block* (dalam Tabel I sebelum *block Residual*) menuju ke layer konvolusi setelah *block residual*, apabila perhitungan *residual block* berikutnya tertulis 2x, maka *residual block* dilakukan sebanyak 2x. Pada *Avgpool* tidak terjadi aktivasi[2]. Jika mengkuti pada skala yang diberikan sesuai pada konfigurasi yang diatur, maka skala pada input setelah proses augmentasi data adalah 416x416, dimana pada *multi-scale detector* pada 3 proses *residual block* terakhir (*residual block* 8x, *residual block* 8x, *residual block* 4x pada Tabel I) akan menghasilkan 3 skala detektor untuk arsitektur *Darknet-53* dengan output stride sebesar 32x32, 16x16, dan 8x8.

ISSN (Online) 2747-0563
Volume 1 Tahun 2020

TABEL I
FEATURE EXTRACTOR DARKNET-53

Type	Filters	Size	Output	
Convolutional	32	3x3	256x256	
Convolutional	64	3x3/2	128x128	
Convolutional	32	1x1		1x
Convolutional	64	3x3		
Residual			128x128	
Convolutional	128	3x3/2	64x64	
Convolutional	64	1x1		2x
Convolutional	128	3x3		
Residual			64x64	
Convolutional	256	3x3/2	32x32	
Convolutional	128	1x1		8x
Convolutional	256	3x3		
Residual			32x32	
Convolutional	512	3x3/2	16x16	
Convolutional	256	1x1		8x
Convolutional	512	3x3		
Residual			16x16	
Convolutional	1024	3x3/2	8x8	
Convolutional	512	1x1		4x
Convolutional	1024	3x3		
Residual			8x8	
Avgpool		Global		
Connected		1000		
Softmax				

Sesuai dengan namanya, *Darknet-53* menggunakan sejumlah 53 layer dimana setiap layer diisi oleh *batch normalization* layer dan aktivasi *Leaky ReLU*[2]. Pada Tabel I, seiring bertambahnya jumlah layer konvolusi, jumlah filter akan bertambah sebanyak 2x lipat. Alasan dibalik ini adalah perlunya mencari cangkupan luas dan objek – objek yang dapat dicari. Untuk layer pertama tidak memiliki varian yang dapat dicari, tetapi seiring bertambahnya layer, maka cangkupan yang dicari bisa semakin luas.

C. *YOLOv3-tiny*

Berbeda dengan versi besar dari YOLOv3, YOLOv3-tiny menggunakan arsitektur yang mirip dari YOLOv2 yaitu Darknet-19 dengan perbedaan yaitu tidak terdapat multi-scale detector untuk versi YOLOv2. Melihat pada Tabel II, kita dapat melihat bahwa terdapat sebanyak 23 *layers* dengan melakukan 2 *routing layer* untuk menciptakan *multi-scale detector*. Jika YOLOv3 terjadi *residual block*, maka pada YOLOv3-tiny tidak dilakukan *residual block* untuk mempercepat proses konvolusi layer.

TABEL II
STRUKTUR FEATURE EXTRACTOR YOLOV3-TINY

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	3x3/1	416x416x3	416x416x16
1	Maxpool		2x2/2	416x416x16	208x208x16
2	Convolutional	32	3x3/1	208x208x16	208x208x32
3	Maxpool		2x2/2	208x208x32	104x104x32
4	Convolutional	64	3x3/1	104x104x32	104x104x64
5	Maxpool		2x2/2	104x104x64	52x52x64
6	Convolutional	128	3x3/1	52x52x64	52x52x128
7	Maxpool		2x2/2	52x52x128	26x26x128
8	Convolutional	256	3x3/1	26x26x128	26x26x256
9	Maxpool		2x2/2	26x26x256	13x13x256
10	Convolutional	512	3x3/1	13x13x256	13x13x512
11	Maxpool		2x2/1	13x13x512	13x13x512
12	Convolutional	1024	3x3/1	13x13x512	13x13x1024
13	Convolutional	256	1x1/1	13x13x1024	13x13x256
14	Convolutional	512	3x3/1	13x13x256	13x13x512
15	Convolutional	255	1x1/1	13x13x512	13x13x255
16	YOLO				
17	Route 13				
18	Convolutional	128	1x1/1	13x13x256	13x13x128
19	Upsample		2x2/1	13x13x128	26x26x128
20	Route 19, 8				
21	Convolutional	256	3x3/1	26x26x384	26x26x256
22	Convolutional	255	1x1/1	26x26x256	26x26x255
23	YOLO				

Yang membedakan dari YOLOv2 yaitu pada tahap setelah *layer* ke-16, maka akan dilakukan routing terhadap skala kedua dengan cara merouting *convolutional layer* tersebut

ISSN (Online) 2747-0563
Volume 1 Tahun 2020

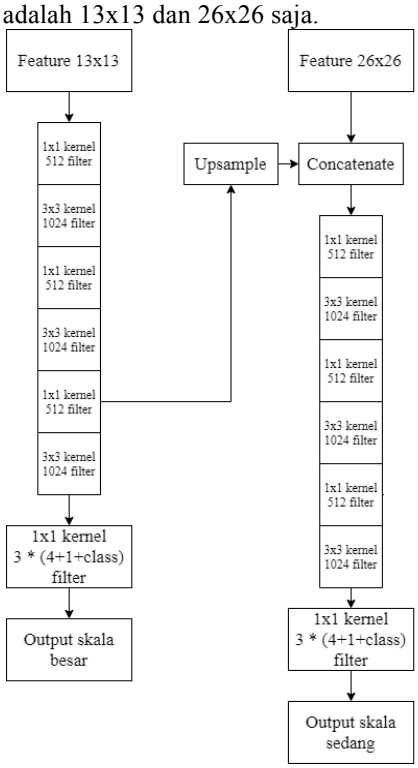
untuk dilakukan *concatenate* sehingga mulai pada *layer* 19 akan didapati skala kedua dari YOLOv3-tiny yaitu 26x26[3]. Sedangkan membandingkan dari Tabel I, struktur awal YOLOv3-tiny dimulai dengan *filter* sejumlah 16, berbeda dengan *Darknet-53* yaitu 32 *filter*.

D. *Multi-scale Detector YOLOv3-tiny*

YOLOv3-tiny menggunakan 2 skala deteksi berbeda untuk memprediksi objek. Rasio yang diberikan adalah 13x13 dan 26x26. Rumus yang diberikan dari versi besar YOLOv3:

$$NxN*[B*(5+C)] \quad (1)$$

Dimana N merupakan rasio fitur, B adalah jumlah *bounding box* pada *cell* yang dapat diprediksi, dan C merupakan jumlah class yang diberikan. Pada YOLOv3 yang telah ditrain pada COCO Dataset, B (jumlah anchor) yang diberikan sejumlah 3, dan C (jumlah class) sejumlah 80, sehingga mendapati ukuran kernel sejumlah NxNx3x255[9]. 5 dapat diambil dari angka 4 *bounding box offset* dijumlahkan dengan *objectness prediction* yaitu 1. Selanjutnya kita ambil peta fitur dari 2 lapisan sebelumnya dan *upsample* sejumlah 2 ×. Metode ini memungkinkan kita mendapatkan lebih banyak informasi semantik yang bermakna dari fitur yang di-upsampled dan informasi yang lebih terperinci dari *feature map* sebelumnya. Namun untuk YOLOv3-tiny, skala yang diberikan adalah 13x13 dan 26x26 saja.



Gbr. 3 *Multi Scale Detector, Feature 13x13 Didapat dari Ekstraksi YOLO Pertama, Kemudian Feature 26x26 Didapat Setelah Upsampling*

Setelah menerima ekstraksi fitur, maka langkah selanjutnya yaitu dengan melakukan *scale detector* dengan awalan menggunakan fitur gambar 13x13 yang nantinya dilakukan proses konvolusi hingga 3 kali, kemudian kita masukkan *detector* YOLO pada output konvolusi yang menghasilkan output dengan skala besar. Kemudian untuk fitur gambar 26x26, kita mengambil konvolusi ke-5 dari fitur sebelumnya yang kemudian kita upsampling sebesar 2x ukuran yang nantinya akan kita *concatenate* dan dilanjutkan dengan konvolusi dan *detector* YOLO, menghasilkan output skala sedang.

E. *Bounding Box*

Ketika memasukkan gambar, sebuah target pada *network* tersebut akan pertama dipilih, kemudian titik tengah gambar tersebut akan ditentukan [10] menggunakan anchor box tepat pada *ground truth*. Jadi, ketika kita memprediksi *bounding box*, kita akan menyesuaikan *box* tersebut sesuai pada *anchor* untuk menyesuaikan target objek, oleh karena itu kita melakukan *offset* [11]. Pada setiap bounding box, YOLOv3-tiny mengestimasi 4 koordinat bounding box dengan simbol (tx, ty, th, tw)[9]. Rumus untuk bounding box adalah:

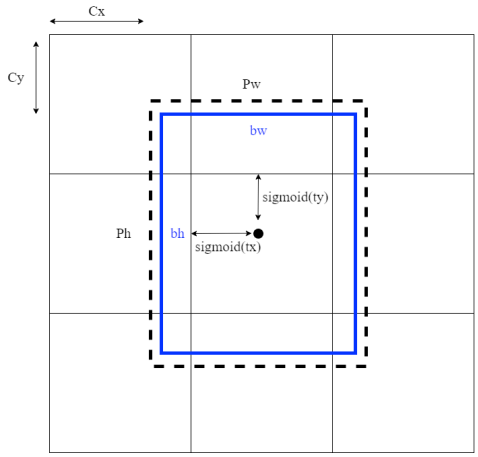
$$bx = \sigma(tx) + cx \quad (2)$$

$$by = \sigma(ty) + cy \quad (3)$$

$$bw = pw.c^{tw} \quad (4)$$

$$bh = ph.c^{th} \quad (5)$$

Dimana (tx,ty) merupakan titik tengah dari *bounding box*, kemudian (tw,th) merupahan ukuran *bounding box*, (pw,ph) merupakan ukuran dari segmentasi *anchor box* [11], dan (cx,cy) merupakan koordinat *offset*. Karena normalisasi, maka nilai dari koordinatnya berkisar diantara 0-1. YOLOv3-tiny mengestimasi nilai objek pada setiap *bounding box* sejumlah 3 *anchor* berdasarkan $[B*(5+C)]$ [2] setiap skala.



Gbr. 4 *Bounding Box Target Object YOLOv3, bw,bh Merupakan Prediksi Bounding Box, pw,ph Merupakan Bounding Box*

ISSN (Online) 2747-0563
Volume 1 Tahun 2020

III. HASIL DAN PEMBAHASAN

Pada bagian ini, kami menggunakan bobot pra-latih yang telah disediakan untuk kami latih ulang, dan network yang kami gunakan adalah YOLOv3-tiny yang memiliki kemiripan dengan backbone Darknet-19. Untuk ukuran input gambar kami menggunakan konfigurasi file YOLOv3-tiny dengan ukuran *network* sebesar 416 x 416. Kami menggunakan konfigurasi nilai bobot pra-latih yang nantinya setelah proses latih menghasilkan *file* bobot yang terlatih untuk digunakan *test detection* pada gambar statis dan *real-time detection.*

A. Dataset

Dataset yang kami gunakan adalah dataset *open source* yang telah disediakan beserta dengan labelingnya yang telah kami ubah menjadi YOLO format. Untuk konfigurasi format label YOLO adalah <object-class> <x_center> <y_center> <width> <height> dimana format <object-class> adalah penomoran objek yang dimulai dari 0 (dapat dilihat berdasarkan konfigurasi nama data objek), kemudian <x_center> <y_center> <width> <height> adalah nilai *float* yang berdasarkan panjang dan lebar dari gambar tersebut. Karena YOLO melakukan 1x1 convolusi gambar maka nilai *float* berjarak antara 0 hingga 1.

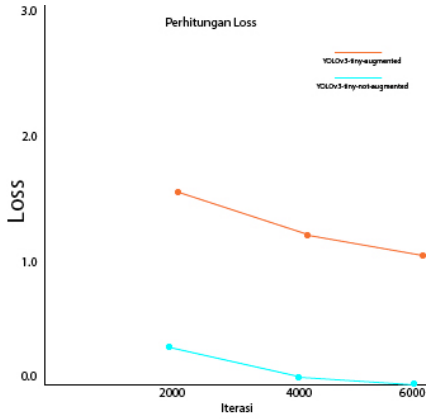
Dataset kami bandingkan antara dataset mentah dengan dataset yang telah diaugmentasikan, pada opsi data augmentasi kami mengubah gambar dengan menambahkan opsi *rotation, random cropping,* dan opsi *flip*. Jumlah dataset kami berkisaran 200 dataset untuk dataset mentah, menjadi 600 dataset beserta augmentasi yang nantinya diiterasi sebanyak 6000 iterasi untuk mengurangi nilai loss yang ditimbulkan setelah nilai bobot final.

B. Spesifikasi Komputer

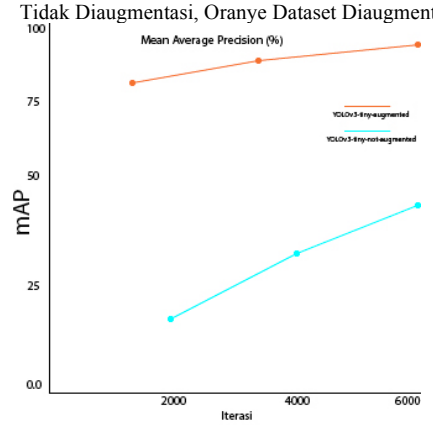
Untuk menjalani proses training beserta data augmentasinya, kami menggunakan arsitektur CPU Intel Core i7-7700HQ, GPU NVIDIA GTX 1050 4GB GDDR5 dengan jumlah core sebesar 640, RAM 16GB 2400Mhz dan HDD Storage SATA 1TB 5400 rpm. Untuk konfigurasi YOLOv3-tiny dapat dilihat pada Tabel II. Untuk mempercepat proses training kami, pada konfigurasi training kita atur mode training untuk berorientasi pada GPU, mengingat jumlah core yang mencukupi untuk proses training yang lebih cepat.

C. Hasil Training

Setelah melakukan proses training, kami mendapati hasil yang sangat signifikan diantara 2 jenis dataset yaitu dataset normal dengan dataset yang telah diaugmentasi. Hasil dari training dapat dilihat pada Gbr. 4. Perbedaan antara 2 jenis dataset ini sangat signifikan mulai dari tingkat loss pada iterasi awal hingga *Mean Average Precision*(mAP).



Gbr. 5.a. Perhitungan *Loss,* Warna Biru Muda Menunjukkan Dataset Tidak Diaugmentasi, Oranye Dataset Diaugmentasi



Gbr. 5.b Perhitungan *Mean Average Precision(mAP),* Warna Biru Menunjukkan Dataset Tidak Diaugmentasi, Oranye Dataset Diaugmentasi

Dari data Gbr. 5 menunjukkan bahwa nilai *loss* dari dataset yang tidak diaugmentasi dimulai dari iterasi ke-2000 sudah menunjukkan nilai loss dibawah nilai 1, bahkan saat menyentuh nilai iterasi ke-6000 sudah mencapai nilai 0.00315 loss pada dataset, sedangkan untuk dataset yang kami augmentasikan, nilai dari iterasi ke-2000 menunjukkan grafik *chart* loss disekitar nilai 1.5, menuju ke iterasi ke-6000 dengan nilai *loss* 1.128.

Namun apabila kita melihat akurasi menggunakan nilai *Mean Average Precision* (mAP), dataset yang tidak diaugmentasi mengalami kesulitan dalam akurasi serendah dibawah 25%, dan mencapai puncaknya pada iterasi ke-6000 disekitar 45% - 50%, dan dari dataset yang kami augmentasikan, dimulai dari iterasi ke-2000 sudah menunjukkan nilai mAP disekitar 77%, dan mencapai puncak pada iterasi ke-6000 disekitar 88%. Perhitungan ini cukup signifikan karena data yang tidak kami augmentasikan tidak kami atur sedikitpun, sedangkan data yang kami augmentasikan, dengan menambahkan opsi *rotation, random cropping,* dan opsi *flip* secara acak, menunjukkan keakurasian data yang sangat signifikan.

ISSN (Online) 2747-0563
Volume 1 Tahun 2020

D. Uji Coba

Data uji coba kami menggunakan 4 jenis uji coba, yaitu uji coba dengan menggunakan masker, tidak menggunakan masker, menggunakan masker dengan memalingkan kepala, dan juga *real-time* detection.

(a) (b) (c)

Gbr. 6. (a) Dataset Augmentasi Menggunakan Masker, (b) Dataset Augmentasi Menggunakan Masker Dengan Memiringkan Kepala, (c) *Real-Time* Deteksi Masker Dengan Augmentasi

(a) (b) (c)

Gbr. 7. (a) Dataset Tanpa Augmentasi Menggunakan Masker, (b) Dataset Tanpa Augmentasi Menggunakan Masker Dengan Memiringkan Kepala, (c) *Real-Time* Deteksi Masker Tanpa Augmentasi

(a) (b) (c)

Gbr. 8. (a) Dataset Augmentasi Tidak Menggunakan Masker, (b) Dataset Augmentasi Tidak Menggunakan Masker Dengan Memiringkan Kepala, (c) *Real-Time* Deteksi Tidak Menggunakan Masker Dengan Augmentasi

(a) (b) (c)

Gbr. 9. (a) Dataset Tanpa Augmentasi Tidak Menggunakan Masker, (b) Dataset Tanpa Augmentasi Tidak Menggunakan Masker Dengan Memiringkan Kepala, (c) *Real-Time* Deteksi Tidak Menggunakan Masker Tanpa Augmentasi

Dari ilustrasi mulai dari Gbr. 6, Gbr. 7, Gbr. 8, Gbr. 9 dapat dianalisa dimana apabila dataset yang menggunakan augmentasi data, maka tampak jelas bahkan deteksi secara *real-time* dapat mendeteksi adanya masker atau tidak dengan tingkat keakurasian yang cukup tinggi, walaupun memalingkan wajah. Sedangkan untuk data yang tidak diaugmentasi, terlihat bahwa untuk gambar statis sekalipun masih belum dapat mendeteksi apakah seseorang pada gambar tersebut menggunakan masker atau tidak, pada proses *real-time*, terdapat deteksi masker atau tidak namun dengan tingkat akurasi yang sangat buruk

TABEL III
HASIL DETEKSI MASKER DENGAN AUGMENTASI DAN TANPA AUGMENTASI

Data	Augmentasi	Akurasi%
Masker	1	98%
Masker	0	Null
Masker 45°	1	81%
Masker 45°	0	Null
Tidak Masker 45°	1	80%
Tidak Masker 45°	0	Null
Tidak Masker	1	97%
Tidak Masker	0	Null
Real-Time	1	100%
Real-TIme	0	29%
Real-Time Tidak Masker	1	100%
Real-Time Tidak Masker	0	55%

Jika dianalisa terhadap Tabel III, secara akurasi untuk dataset yang tidak *real-time*, dengan data augmentasi akurasi yang dihasilkan berjarak antara 98-100%, sedangkan dataset yang *real-time* berakurasi sama yaitu antara 95-100%. Untuk dataset yang tidak diaugmentasi, deteksi yang tidak *real-time* secara mayoritas tidak dapat mendeteksi apapun, sama halnya dengan yang menggunakan masker. Namun saat kami mencoba *real-time*, beberapa kondisi data dapat dikenali apakah menggunakan masker atau tidak namun dengan akurasi yang sangat buruk yaitu berkisar antara 20-60%.

IV. KESIMPULAN

Dari hasil uji coba, dapat diambil kesimpulan bahwa YOLOv3-tiny sangat baik untuk melakukan deteksi objek, pada penelitian ini adalah deteksi masker. Dan untuk mendapatkan hasil yang maksimal, maka diperlukannya sebuah data yang telah diaugmentasikan sebelum proses latih supaya akurasi yang ditimbulkan akan semakin akurat. Walaupun dengan arsitektur komputer yang cukup minim, menggunakan arsitektur YOLOv3-tiny dapat digunakan.

ISSN (Online) 2747-0563
Volume 1 Tahun 2020

REFERENSI

[1] W. Fang, L. Wang and P. Ren, "*Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments*," in IEEE Access, vol. 8, pp. 1935-1944, 2020, doi: 10.1109/ACCESS.2019.2961959.

[2] J. Redmon and A. Farhadi, "*YOLOv3: An Incremental Improvement*" arXiv preprint arXiv:1804.02767, 2018

[3] T. Li, Y. Ma and T. Endoh, "*A Systematic Study of Tiny YOLO3 Inference: Toward Compact Brainware Processor With Less Memory and Logic Gate*," in IEEE Access, vol. 8, pp. 142931-142955, 2020, doi: 10.1109/ACCESS.2020.3013934.

[4] Y. Liu et al., *Research on automatic location and recognition of insulators in substation based on YOLOv3,* in High Voltage, vol. 5, no. 1, pp. 62-68, 2 2020, doi: 10.1049/hve.2019.0091.

[5] H. Zhang et al., *Real-Time Detection Method for Small Traffic Signs Based on Yolov3*, in IEEE Access, vol. 8, pp. 64145-64156, 2020, doi: 10.1109/ACCESS.2020.2984554

[6] L. Huang, W. Pan, Y. Zhang, L. Qian, N. Gao and Y. Wu, "*Data Augmentation for Deep Learning-Based Radio Modulation Classification,*" in IEEE Access, vol. 8, pp. 1498-1506, 2020, doi: 10.1109/ACCESS.2019.2960775

[7] Z. Meng, X. Guo, Z. Pan, D. Sun and S. Liu, "*Data Segmentation and Augmentation Methods Based on Raw Data Using Deep Neural Networks Approach for Rotating Machinery Fault Diagnosis*," in IEEE Access, vol. 7, pp. 79510-79522, 2019, doi: 10.1109/ACCESS.2019.2923417

[8] J. Lemley, S. Bazrafkan and P. Corcoran, "*Smart Augmentation Learning an Optimal Data Augmentation Strategy*," in IEEE Access, vol. 5, pp. 5858-5869, 2017, doi: 10.1109/ACCESS.2017.2696121.

[9] X. Wang, S. Wang, J. Cao and Y. Wang, "*Data-Driven Based Tiny-YOLOv3 Method for Front Vehicle Detection Inducing SPP-Net,*" in IEEE Access, vol. 8, pp. 110227-110236, 2020, doi: 10.1109/ACCESS.2020.3001279.

[10] Z. Liu and S. Wang, "*Broken Corn Detection Based on an Adjusted YOLO With Focal Loss,*" in IEEE Access, vol. 7, pp. 68281-68289, 2019, doi: 10.1109/ACCESS.2019.2916842.

[11] J. Redmon and A. Farhadi, "*YOLO9000: Better, Faster, Stronger*" arXiv preprint arXiv:1612.08242, 2016