Teknika 17 (2): 447 – 458

JURNAL TEKNIKA
ISSN: 0854-3143
e-ISSN: 2622-3481
Journal homepage: http://jurnal.polsri.ac.id/index.php/teknika
Journal Email: teknika@polsri.ac.id

Deteksi Objek di Atas Konveyor Peraga dengan EfficientDet dan Raspberry Pi

Enas Dhuhri Kusuma* [1], Tariq Fitria Aziz [2], Sujoko Sumaryono [3]

[1,2,3]Dept. Teknik Elektro dan Teknologi Informasi, Fakultas Teknik, Universitas Gadjah Mada, Yogyakarta, Indonesia
*Email Penulis Korespondensi: enas@ugm.ac.id

Abstrak

Detektor objek untuk memperagakan sistem konveyor merupakan kebutuhan penting dalam edukasi mahasiswa teknik agar tidak terjadi gap pengetahuan industri. Meskipun demikian, ketersediaan alat peraga tersebut di laboratorium masih belum terpenuhi sehingga perancangan dilakukan untuk memenuhi kebutuhan tersebut. Detektor ini akan menggabungkan perangkat Raspberry Pi dengan model dari keluarga EfficientDet yang dilatih ulang melalui transfer learning. Implementasi rancangan produk dilakukan dengan framework TensorFlow. Model yang dihasilkan adalah varian ringan dari model yang diberi nama Efficientdet-Lite. Deteksi dilakukan terhadap objek yang memiliki warna berbeda dan bergerak dengan kecepatan tertentu di konveyor peraga. Dari hasil pengujian, rancangan yang dibuat mampu memenuhi spesifikasi: akurasi 0,282-0,502 mAP, konsumsi energi 6,5 Watt-jam, dan frame rate 1-7 FPS. Pengujian juga menunjukkan bahwa perbaikan kinerja dapat terjadi apabila kecepatan objek diperlambat, dan warna objek dibuat kontras terhadap background. Ukuran parameter model tidak menjamin kinerja deteksi terbaik.

Kata kunci— *EfficientDet, Raspberry Pi, Deteksi objek, Visi komputer, Automasi manufaktur*

Abstract

Object detectors for conveyor system trainers are very important tools in engineering education, especially to avoid gaps with industrial knowledge. However, in most cases, university laboratories sometimes lack trainer devices availability. Hence, action to fulfill the need of those trainers should be initiated. This detector will combine Raspberry Pi device with EfficientDet family model which is trained by transfer learning method. Product design implementation has been done with TensorFlow framework. The generated model is the lite version of EfficientDet-Lite model. Detection is performed to objects with different colors and move with arbitrary speed in the conveyor trainer. On the results, proposed system can achieve specifications as follows: 0.282 mAP, energy consumption 6.5 Wh, and framerate 1-7 FPS. The proposed system also shows that performance enhancement can be achieved if the object speed is slowed down, and contrast between object color and background is sharpened. Parameter sizes don't guarantee best detection performance.

Keywords— *EfficientDet, Raspberry Pi, Object detection, Computer vision, Manufacture automation*

1. PENDAHULUAN

Sebagai bagian dari teknologi kecerdasan buatan (*artificial intelligence*) dan pembelajaran mesin (*machine learning*), visi komputer telah diaplikasikan ke dalam berbagai kebutuhan mulai dari pengawasan lalu lintas hingga mengukur pertumbuhan tanaman [1,2]. Selain itu, visi komputer juga merupakan penyokong aktivitas sistem manufaktur cerdas (*smart manufacturing system*) [3]. Dengan sensor yang tersebar secara luas, visi komputer dapat membantu proses pengawasan dan pengumpulan data dari objek-objek dalam proses manufaktur secara waktu-nyata (*real-time*) dan dalam skala yang masif. Hal ini, pada akhirnya, akan membantu sistem manufaktur mencapai otonomi. Apabila mencapai otonomi, sistem manufaktur diharapkan akan mengalami peningkatan produktivitas dan fleksibilitas yang menguntungkan pelaku industri.

Pemanfaatan visi komputer di industri sayangnya tidak diajarkan di Laboratorium Schneider Electric Universitas Gadjah Mada karena tidak adanya alat peraga. Meskipun demikian, dengan maraknya penelitian di bidang ini, peneliti menjadi tertarik untuk mendesain sistem tersebut untuk kebutuhan praktikum di laboratorium. Sebagai alat peraga, sistem yang didesain akan jauh lebih sederhana dibandingkan yang ada di industri. Sistem ini diharapkan mampu mendeteksi objek mainan di atas konveyor berjalan sehingga dapat membantu edukasi calon insinyur yang akan berkecimpung di industri manufaktur.

Deteksi objek dilakukan dengan perangkat berdaya rendah dan memiliki keterbatasan komputasi yaitu Raspberry Pi. Untuk menjalankan deteksi, pelaksana akan memanfaatkan model pembelajaran mesin dari keluarga EfficientDet [4]. Dalam implementasi model, TensorFlow adalah salah satu framework yang menyediakan model terlatih (*pre-trained model*) [5]. Model kemudian dilatih ulang melalui metode transfer learning dengan memakai TensorFlow Lite Model Maker API. Hasilnya adalah varian EfficientDet yang bernama EfficientDet-Lite. Pelatihan model dilaksanakan di Google Colaboratory dengan memakai dataset yang telah disiapkan. Setelah model di-deploy ke Raspberry Pi, deteksi dilakukan terhadap objek dengan warna kontras dan objek dengan warna sama seperti konveyor (*background*).

2. METODE PENELITIAN

2.1 Detail Implementasi

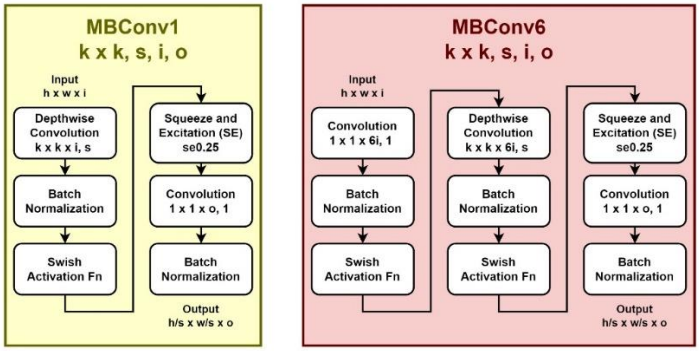
2. 1.1 Arsitektur Model EfficientDet-Lite

EfficientDet merupakan model deteksi yang dikembangkan untuk perangkat dengan kapabilitas komputasi terbatas. Model ini dibangun di atas jaringan EfficientNet. Jaringan ini dapat diatur ukurannya dengan metode khusus bernama *compound scaling* [6]. Dalam metode tersebut, jaringan akan dimodifikasi kedalaman, lebar, dan resolusinya sesuai dengan faktor skala tertentu. Dari *compound scaling*, EfficientNet memiliki delapan varian yaitu EfficientNet-B0 hingga EfficientNet-B7. Semakin tinggi angka di belakang nama model, semakin besar parameter model, dan akurasi yang diberikan semakin tinggi. Meskipun demikian, komputasi yang dibutuhkan juga semakin berat. EfficientNet dibentuk oleh blok jaringan yang bernama MBConv [7]. MBConv dibentuk oleh operasi konvolusi, *depthwise convolution*, fungsi aktivasi *swish*, *batch normalization*, dan optimisasi *squeeze-and-excitation* [8, 9, 10].

Peta fitur dari EfficientNet kemudian dimasukkan ke dalam jaringan *Bi-directional Feature Pyramid Network* (BiFPN) [4]. Jaringan BiFPN dimanfaatkan untuk menggabungkan peta fitur dari dua arah: atas-bawah (*top-down*) dan bawas-atas (*bottom-up*). Dalam EfficientDet, jaringan ini juga dapat diatur skalanya dengan metode *compound scaling*. EfficientDet termasuk ke dalam kategori *single-shot detector*. EfficientDet menjadi model populer karena diklaim mampu mencapai kinerja *state-of-the-art* (SOTA) dengan biaya komputasi terbatas [4].

Jaringan EfficientNet-B0 merupakan varian paling kecil dari EfficientNet. Jaringan ini

juga menjadi jaringan dasar yang skalanya akan diubah dengan *compound scaling* untuk menghasilkan varian lain. Oleh karena itu, EfficientNet-B0 disebut sebagai jaringan *baseline*. Arsitektur dari jaringan ini dapat diamati dari Tabel 1. Dari tabel tersebut, perbedaan antara MBConv1 dan MBConv6 dapat dilihat pada Gambar 1. Perbedaan dari kedua blok adalah MBConv1 tidak memiliki fase ekspansi yang ditandai operasi konvolusi 1×1, *batch normalization*, dan *swish*. Selain itu, MBConv6 memiliki enam kali lipat jumlah kanal dibandingkan MBConv1. Pada EfficientDet, lapisan prediksi di tingkat kesembilan tidak akan dipanggil.



Gambar 1. Blok MBConv Pembentuk EfficientNet

Tabel 1. Arsitektur Jaringan Efficientnet-B0

Tingkat	Operator	Resolusi	Kanal (*c*)	Pengulangan (*n*)
1	Conv3×3	224×224	12	1
2	MBConv1, 3×3	112×112	16	1
3	MBConv6, 3×3	112×112	24	2
4	MBConv6, 3×3	56×56	40	2
5	MBConv6, 3×3	28×28	80	3
6	MBConv6, 3×3	28×28	112	3
7	MBConv6, 3×3	14×14	192	4
8	MBConv6, 3×3	7×7	320	1
9	Conv1×1, Pooling, FC	7×7	1280	1

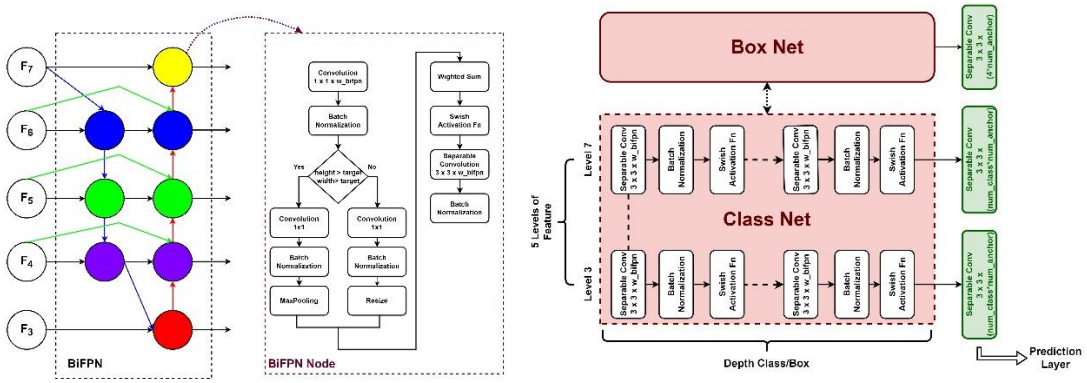
Lapisan BiFPN dapat diamati pada Gambar 2(a). Dari gambar tersebut, BiFPN dibentuk oleh operasi konvolusi, *batch normalization*, aktivasi *swish*, *resize* atau *pooling*, *weighted sum*, dan *separable convolution*. *Separable convolution* adalah konvolusi khusus yang membagi pemetaan berdasar kanal sebelum disatukan dengan konvolusi 1×1 [11]. Operasi *pooling* dan *resize* akan menyamakan ukuran peta fitur dari tingkat berbeda. Sementara itu, peta fitur ini akan dijumlahkan dengan *weighted sum* sehingga diperoleh tiur agregat. Apabila F_i^{td} menyatakan fitur dari aliran atas ke bawah BiFPN sementara F_i^{bu} menunjukkan fitur bawah ke atas, serta $w_i \in (0,1)$ menyatakan bobot, maka *weighted sum* dapat dinyatakan oleh (1) dan (2).

$$F_i^{td} = Conv\left(\frac{w_1 \cdot F_i^{in} + w_2 \cdot Resize\left(F_{i+1}^{in}\right)}{w_1 + w_2 + \epsilon}\right) \quad (1)$$

$$F_i^{out} = F_i^{bu} = Conv\left(\frac{w_i' \cdot F_i^{in} + w_2' \cdot F_i^{td} w_3' \cdot Resize\left(F_{i-1}^{out}\right)}{w_1' + w_2' + w_3' + \epsilon}\right) \quad (2)$$

Fitur dari BiFPN akan dipetakan oleh lapisan prediksi kelas dan kotak batas. Lapisan ini, seperti ditunjukkan Gambar 2(b), terdiri dari *separable convolution*, *batch normalization*, dan *swish* untuk setiap tingkat fitur. Perbedaan ditemukan antara lapisan prediksi kotak batas dan

kelas yaitu jumlah kanal fitur. Lapisan kotak batas memiliki jumlah kanal sama dengan empat kali *anchor/default bounding box* sementara kanal lapisan kelas akan mengikuti jumlah kelas yang ada.



(a) (b)

Gambar 2. Lapisan (a) BiFPN dan (b) Prediksi Kelas dan Kotak Batas EfficientDet

Tabel 2. Perbandingan Arsitektur Model Efficientdet-Lite

Model	***Backbone***	***Image Size*** (r_{input})	***Num Filter*** (w_{bifpn})	***Num BiFPN*** (d_{bifpn})	***Num Head Conv*** ($d_{class/box}$)
Lite0	EfficientNet-Lite0	320×320	64	3	3
Lite1	EfficientNet-Lite1	384×384	88	4	3
Lite2	EfficientNet-Lite2	448×448	112	5	3
Lite3	EfficientNet-Lite3	512×512	160	6	4
Lite4	EfficientNet-Lite4	640×640	224	7	4

Apabila semua bagian digabungkan, arsitektur lengkap EfficientDet dapat diamati pada Gambar 3. Arsitektur tersebut akan diubah dengan *compound scaling* sehingga diperoleh delapan varian EfficientDet yaitu EfficientDet-D0 hingga EfficientDet-D7. Meskipun demikian, varian tersebut bukan varian yang digunakan dalam penelitian.

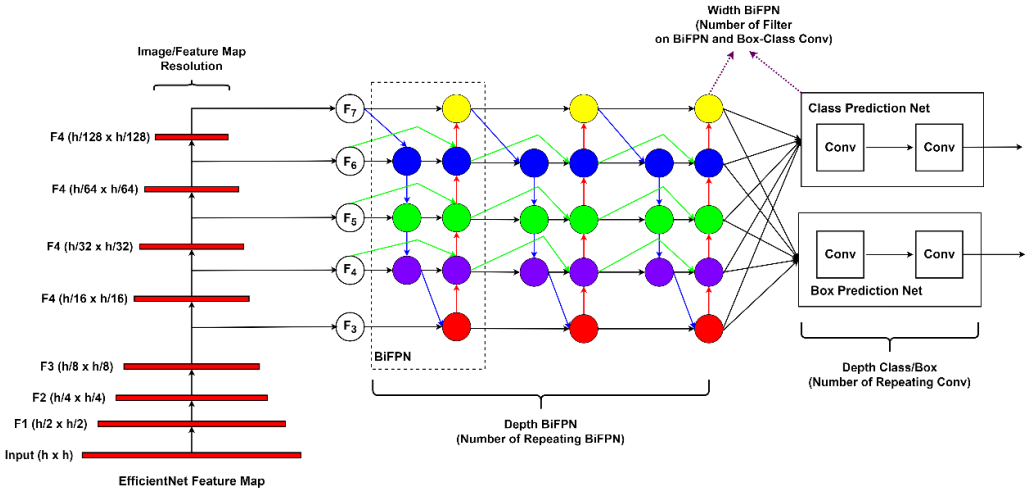
Penelitian ini memakai EfficientDet-Lite, yaitu varian yang dikembangkan dengan *framework* TensorFlow Lite. Arsitektur dari varian ini memiliki skala yang ditunjukkan oleh Tabel 2. Dari tabel tersebut, EfficientDet-Lite memakai jaringan tulang belakang bernama EfficientNet-Lite. Jaringan tersebut adalah varian TensorFlow Lite untuk EfficientNet. Perbedaan varian ini dengan varian EfficientNet biasa adalah jaringan MBConv tidak akan dioptimisasi oleh lapisan *squeeze-and-excitation*. Perbedaan lain adalah seluruh fungsi aktivasi pada EfficientDet akan diganti dari swish menjadi fungsi ReLU6 [12]. Resolusi, kedalaman, dan lebar dari EfficientDet-Lite juga dituliskan sehingga pengaruh terhadap arsitektur pada Gambar 3 dapat diketahui. EfficientDet-Lite dalam *framework* TensorFlow adalah model yang telah dilatih (*pre-trained*) dan duji dengan *dataset* MS-COCO [13].

2. 2.2 Persiapan Dataset

Penelitian ini memakai *dataset* khusus yang disiapkan dari nol. *Dataset* dibuat dengan melalui tiga tahapan: pengumpulan data, *augmentation*, dan pelabelan. Pengumpulan data dilakukan dengan kamera 13 MegaPixels dari gawai Samsung J7 Prime. Data citra diolah dalam proses *augmentation*. Proses ini bertujuan untuk meningkatkan keberagaman data dan memastikan data mengikuti standar yang sama. Proses *augmentation* dilakukan pada Google Colaboratory dengan memanfaatkan *framework* TensorFlow dan Keras.

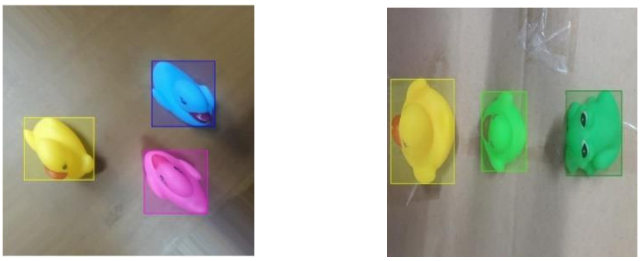
Setelah proses *augmentation*, data citra diberi label dengan Label Studio. Label Studio merupakan perangkat lunak sumber terbuka yang dapat dipakai dalam pelabelan berbagai data,

mulai dari citra, audio, dan seterusnya. Data yang telah diberi label dalam Label Studio dapat diekspor dalam berbagai format baku. Untuk data citra, format yang disediakan mengikuti berbagai kompetisi atau model dalam visi komputer seperti PASCAL VOC, MS-COCO, dan YOLO. Dalam penelitian ini, dataset yang telah diberi label diekspor dalam berkas ZIP dengan mengikuti format PASCAL VOC XML [14].



Gambar 3. Arsitektur Lengkap Model Deteksi EfficientDet

Setelah proses pelabelan, dua *dataset* yang masing-masing terdiri dari 181 dan 238 citra diperoleh. Sampel dari kedua dataset dapat diamati pada Gambar 4. Dataset akan dimanfaatkan untuk mempersiapkan model sehingga dapat mendeteksi dalam dua skenario. Skenario tersebut adalah deteksi objek berdasarkan warna. Pada skenario pertama, objek akan mendeteksi objek dengan warna kontras dengan konveyor. Untuk skenario kedua, objek yang dipilih perlu berwarna mirip dengan *background* yaitu konveyor.



(a) (b)

Gambar 4. Sampel *Dataset* untuk Skenario: (a) Warna Kontras dan (b) Warna Mirip

2. 2.3 Pelatihan Model

Model dilatih di lingkungan Google Colaboratory dengan sumber daya komputasi dari Google Cloud Platform. Google memberikan opsi untuk melatih model dengan GPU sehingga pelatihan menjadi cepat. Pelatihan dilaksanakan dengan TensorFlow Lite Model Maker API. API ini adalah API Python khusus dalam *framework* TensorFlow untuk melatih ulang model *pre-trained* melalui *transfer learning* kemudian mengonversi model tersebut ke format TFLite. Berkas TFLite merupakan format penyimpanan model dari TensorFlow untuk membuat model menjadi lebih ringan dan dapat di-*deploy* ke perangkat komputasi terbatas seperti *single board computer* (SBC) dan perangkat Android.

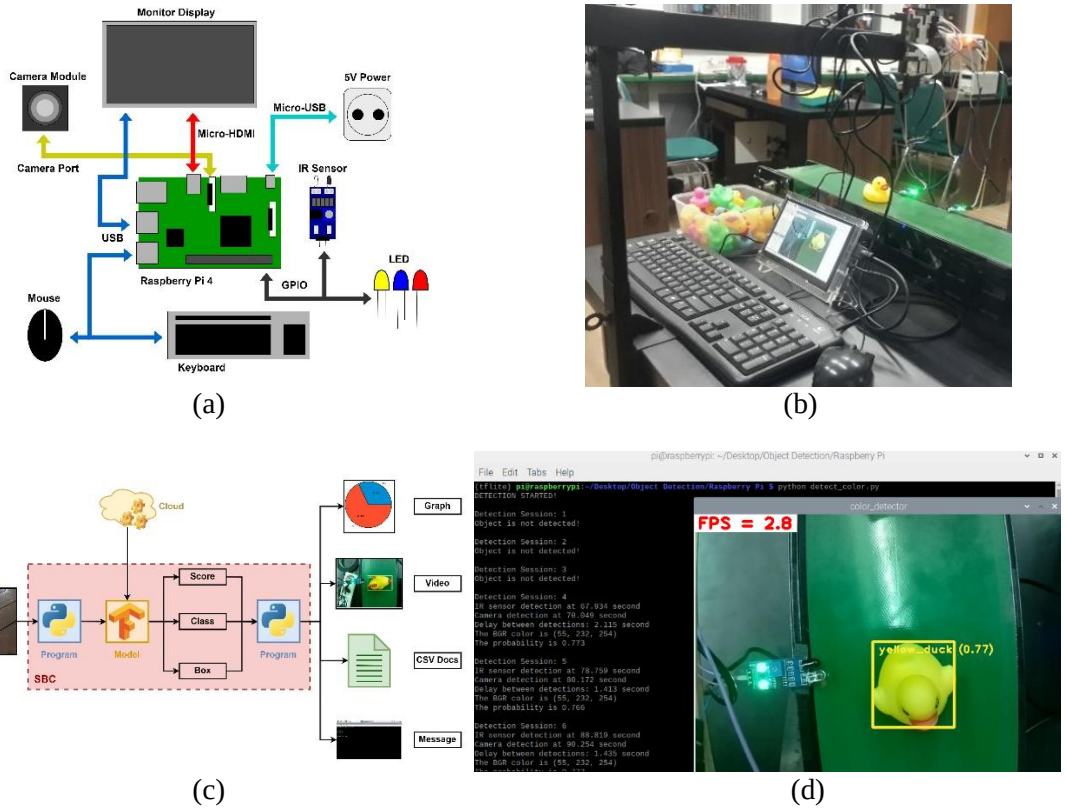
Model dilatih dengan *dataset* yang disiapkan. Setelah pelatihan, model TFLite tidak dikenakan optimisasi dengan proses kuantisasi tambahan sehingga model yang dihasilkan adalah model TFLite *default* dengan kuantisasi 32-bit *floating point*. Selamat pelatihan, *optimizer* yang dimanfaatkan adalah *Stochastic Gradient Descent* dengan fungsi objektif *Huber loss* dan *focal loss* [15]. *Huber* dan *focal loss* masing-masing berperan sebagai fungsi *loss* dari prediksi kotak batas dan kelas. Model ini akan dievaluasi dengan metrik dari MS-COCO.

Tabel 3. Parameter, Ukuran, dan Akurasi EfficientDet-Lite

Model	Jumlah Param	Ukuran (KB)	mAP Kontras	mAP Mirip
EfficientDet-Lite0	3.239.711	4.342	0,315	0,282
EfficientDet-Lite1	4.241.703	5.799	0,476	0,365
EfficientDet-Lite2	5.236.415	7.220	0,400	0,393
EfficientDet-Lite3	8.328.993	11.456	0,429	0,385
EfficientDet-Lite4	15.110.223	20.068	0,502	0,426

Hasil dari eveluasi dapat diamati pada Tabel 3. Dari tabel tersebut, jumlah parameter EfficientDet-Lite berada di kisaran 3 juta hingga 15 juta parameter tergantung varian. Ukuran berada pada rentang 4 MB hingga 20 MB. Sementara itu, akurasi dari model TFLite berada pada rentang 0,282 mAP hingga 0,502 mAP dengan menyesuaikan varian serta skenario deteksi.

2. 2.4 Deployment



Gambar 5. (a) Diagram Perangkat Keras ke (b) Realisasi dan (c) Diagram Perangkat Lunak ke (d) Keluaran Program

Model yang dilatih kemudian di-*deploy* ke perangkat Raspberry Pi 4. Sebelum deployment dilaksanakan, Raspberry Pi 4 perlu dipasang dengan Raspberry Pi OS versi Buster. Setelah itu, perangkat periferal, modul kamera, sensor, dan LED perlu dihubungkan dengan Raspberry Pi. Perangkat-perangkat ini berfungsi untuk menerima masukan dari pengguna,

mengakuisisi citra, mengecek keberadaan objek, dan mengirimkan hasil deteksi ke pengguna. Diagram dan realisasi *setup* perangkat keras untuk menjalankan deteksi objek dapat diamati pada Gambar 5(a) dan Gambar 5(b).

Setelah perangkat keras siap, program integrasi adalah hal selanjutnya yang perlu dirancang. Program ini akan menyatukan dan mengolah hasil prediksi dari model EfficientDet-Lite menjadi keluaran yang diterima oleh pengguna. Keluaran yang dimaksud adalah visualisasi dan pesan deteksi serta visualisasi kinerja deteksi. Integrasi model EfficientDet-Lite dalam program deteksi Raspberry Pi dapat diamati pada Gambar 5(c) dan Gambar 5(d).

Di sini, program dikembangkan dengan menfaatkan bahasa pemrograman Python. Pada Gambar 5(c), program akan menerima masukan citra. Citra kemudian diolah oleh program deteksi sebelum dimasukkan ke model. Pengolahan citra dijalankan dengan pustaka OpenCV. Dalam menjalankan prediksi di SBC, program akan memanfaatkan modul TFLite Support dari TensorFlow. Modul ini memiliki metode untuk mendefinisikan model dan memanggil model tersebut untuk melakukan deteksi. Keluaran dari metode deteksi terdiri dari skor probabilitas, kelas, dan koordinat kotak batas dalam piksel. Keluaran deteksi akan diolah oleh program sehingga diperoleh grafik hasil deteksi, pesan di terminal Raspberry Pi, visualisasi objek, dan berkas *comma separated value* (CSV). Berkas CSV ini berisi data waktu tunda, *frame rate*, nilai *boolean* deteksi/tidak terdeteksi, prediksi label, dan label sesungguhnya dari objek. Keluaran dari program deteksi dapat diamati pada Gambar 5(d). Dari gambar tersebut, objek yang bergerak di konveyor dan berada dalam *Field of View* (FoV) kamera akan divisualisasikan beserta kotak batas dan prediksi label. Selain itu, pada terminal Raspberry Pi, pesan deteksi akan dapat diamati oleh pengguna.

2. 3 Persiapan Pengujian

2. 3.1 Metrik Evaluasi

Dalam menguji kinerja deteksi, metrik yang umum digunakan pada bidang pembelajaran mesin terdiri dari precision, recall, F1-score. Selain itu, model juga akan dibandingkan menurut waktu tunda deteksi, frame rate, dan waktu inferensi.

- ***F1-score:*** Metrik ini memberikan rangkuman nilai *precision* dan *recall* deteksi. Apabila *F1-score* semakin tinggi, maka *precision* dan *recall* juga tinggi. Nilai dari *F1-score* dihitung dengan (5).

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5}$$

- **Waktu Tunda dan *Frame Rate:*** Waktu tunda dihitung dari selisih antara waktu deteksi sensor IR dengan waktu deteksi kamera (model) dalam detik. Sensor IR mampu menemukan keberadaan objek secara *real-time* sementara model akan mengalami penundaan deteksi. Kecepatan deteksi juga diukur dari *frame rate* sekuens citra. Nilai *frame rate* memiliki satuan *frame per second* (FPS).

2. 3.2 Instrumen Pengukuran

Dalam mengukur konsumsi daya dan energi deteksi, instrumen yang dipakai adalah USB Doctor Keweisi. Instrumen ini mampu menampilkan perubahan arus dan tegangan dalam Ampere (A) dan Volt (V). Cara pemakaian instrumen ini adalah dengan menghubungkan catu daya ke porta masukan USB dan menghubungkan porta keluaran USB ke porta daya Raspberry Pi.

Meskipun demikian, karena instrumen ini tidak menghasilkan berkas *log*, pencatatan nilai dilakukan secara manual. Hasil pengukuran instrumen akan direkam dengan gawai dan nilainya akan dicatat satu per satu untuk setiap detik. Setelah itu, nilai arus dan tegangan yang tercatat akan dikalikan untuk memperoleh nilai daya terukur seperti pada (6). Nilai rata-rata daya untuk seluruh hasil pengukuran adalah nilai akhir dari konsumsi daya deteksi. Nilai rata-rata ini

dapat dikalikan dengan lama waktu deteksi dalam jam sehingga dihasilkan konsumsi energi dalam Watt-jam.

$$P = V \cdot I \quad (6)$$

$$W = \bar{P} \cdot t \quad (7)$$

2. 3.3 Skenario Pengujian

Pengujian akan dilakukan dengan mengikuti Tabel 4. Pada tabel tersebut, model memiliki lima alternatif: EfficientDet-Lite0 hingga Lite4. Sementara itu, kecepatan memiliki tiga alternatif: rendah (*low*), sedang (*medium*), dan tinggi (*high*). Secara detail, kecepatan rendah, sedang, dan tinggi masing- masing memiliki rata-rata 0,07533 m/s, 0,253 m/s, dan 0,4105 m/s. Sementara itu, skenario deteksi untuk warna mirip (*similar*) dan kontras (*contrast*) masing-masing memiliki tiga kelas objek. Pengujian akan dilakukan dengan kombinasi skenario (kolom) dan model/kecepatan (baris). Kombinasi ini, yang ditunjukkan oleh lokasi sel, menunjukkan satu kasus deteksi.

Tabel 4. Kasus Deteksi dalam Pengujian

EfficientDet		**Similar**			**Contrast**		
		yellow duck	**green frog**	**green duck**	**yellow duck**	**blue duck**	**pink duck**
Model	**Lite0**	✓	✓	✓	✓	✓	✓
	Lite1	✓	✓	✓	✓	✓	✓
	Lite2	✓	✓	✓	✓	✓	✓
	Lite3	✓	✓	✓	✓	✓	✓
	Lite4	✓	✓	✓	✓	✓	✓
Velocity	**Low**	✓	✓	✓	✓	✓	✓
	Medium	✓	✓	✓	✓	✓	✓
	High	✓	✓	✓	✓	✓	✓

Dalam satu kasus deteksi, model akan dilewatkan di konveyor sebanyak 100 kali sehingga diperoleh 100 data deteksi dalam berkas CSV. Data ini akan dikumpulkan menurut model, kecepatan, atau skenario yang sama sehingga diperoleh data agregat. Data agregat ini akan divisualisasikan dan dianalisis sehingga dapat diketahui pengaruh dari model, kecepatan, dan skenario deteksi.

3. HASIL DAN PEMBAHASAN

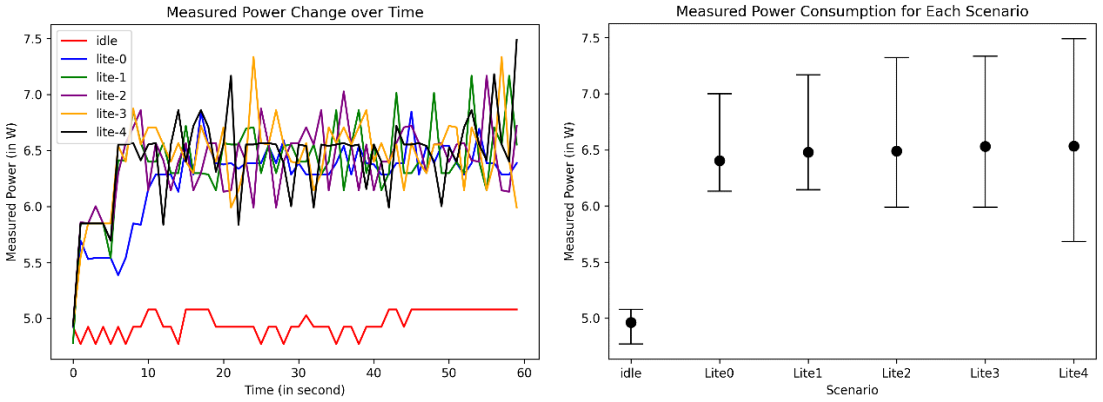
3.1 Konsumsi Daya Deteksi

Pengujian ini dan seterusnya dijalankan di Raspberry Pi 4. Di sini, konsumsi daya deteksi untuk setiap model EfficienetDet-Lite akan diamati. Hasil pengujian konsumsi daya dapat diamati pada Gambar 6. Pada gambar tersebut, rentang konsumsi dan perubahan konsumsi daya terhadap waktu dapat dilihat. Konsumsi daya deteksi dari model EfficientDet-Lite0 hingga EfficientDet-Lite4 akan dibandingkan dengan konsumsi data pada kondisi *idle*.

Dari Gambar 6(a), kondisi *idle* membutuhkan daya yang cenderung tetap yaitu sekitar 5 Watt. Ketika deteksi dilakukan, peningkatan akan terjadi dari 5 Watt menuju sekitar 6,5 Watt. Dengan demikian, deteksi akan menyebabkan peningkatan konsumsi daya sebesar 1,5 Watt. Temuan lain dari Gambar 6(a) adalah peningkatan daya terukur dari EfficientDet-Lite0 yang lebih lama untuk mencapai kondisi stabil (*steady state*). Hal ini menunjukkan bahwa EfficientDet-Lite0 membutuhkan daya yang kecil saat memulai deteksi sebagai akibat jumlah parameter model yang sedikit. Dengan parameter sedikit, komputasi yang dibutuhkan juga minimal, sehingga alokasi CPU ke model akan diberikan secara bertahap. Akibatnya, konsumi daya untuk komputasi juga akan meningkat secara bertahap.

Pengaruh alokasi CPU untuk komputasi model terhadap konsumsi daya juga dapat menjelaskan fenomena pada Gambar 6(b). Pada gambar tersebut, konsumsi daya untuk setiap

model memiliki rata-rata yang hampir sama di kisaran 6,5 Watt. Meskipun demikian, rentang nilai maksimum-minimum akan semakin besar untuk model yang lebih besar. Rata-rata yang sama disebabkan setiap model akan memperoleh alokasi CPU yang sama yaitu 4 Thread CPU. Sementara itu, perbedaan rentang dapat disebabkan oleh fenomena *Multithreading*. Pada fenomena ini, program deteksi akan berbagi sumber daya dengan proses lain di Raspberry Pi. Model yang lebih ringan akan lebih mudah memperoleh alokasi sehingga komputasi lebih stabil. Sementara itu, model yang besar alokasinya akan berubah-ubah untuk menyedikan alokasi ke proses lain sehingga komputasi menjadi kurang stabil. Kestabilan komputasi dapat diamati dari rentang konsumsi daya model. Pada kondisi *idle*, dengan kebutuhan komputasi minimal, alokasi mudah dilakukan sehingga komputasi lebih stabil dan rentang konsumsi daya juga kecil.



(a) (b)

Gambar 6. (a) Perubahan Konsumsi dan (b) Rentang Konsumsi Daya Deteksi pada Raspberry Pi 4

3.2 Perubahan Waktu Tunda dan Frame Rate

Pengujian selanjutnya dilakukan terhadap waktu tunda dan *frame rate* deteksi. Hasil pengujian ini dapat diamati pada Gambar 7. Pada gambar tersebut, peningkatan ukuran model akan menyebabkan penurunan *frame rate* dan peningkatan waktu tunda deteksi. Dari model Lite0 ke Lite4, *frame rate* akan turun dari kisaran 7 FPS menjadi kisaran 1 FPS. Secara lebih detail, *frame rate* turun dari rata-rata 7,2 FPS menjadi 0,668 FPS. Sementara itu, waktu tunda akan meningkat dari rata-rata 0,795 detik menjadi 8,791 detik. Hal ini menunjukkan bahwa model yang lebih besar akan memiliki biaya yang juga lebih besar dari sisi kecepatan deteksi.

Dari grafik waktu tunda, rentang nilai maksimum-minimum juga berubah untuk setiap model. Model yang lebih besar akan memiliki rentang yang lebih besar juga. Hal ini dapat berhubungan dengan kestabilan komputasi model yang sebelumnya dijelaskan dan berhubungan denga alokasi *multithreading* CPU.

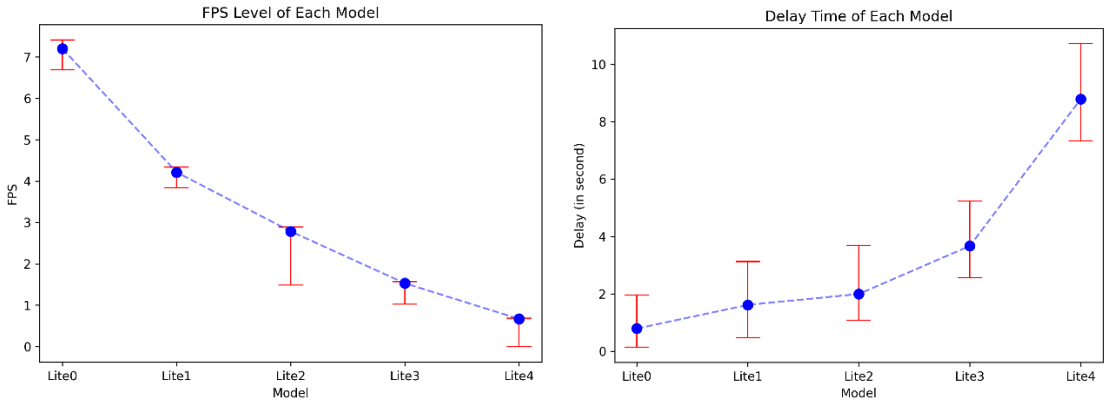
3.3 Perubahan Kinerja terhadap Kecepatan Objek

Objek yang dideteksi adalah objek yang bergerak dengan kecepatan tertentu di konveyor. Pengaruh kecepatan terhadap kinerja deteksi menjadi hal yang perlu ditentukan pengaruhnya sehingga dapat menjadi dasar pertimbangan pengguna. Hasil pengujian deteksi terhadap perubahan kecepatan objek dapat diamati pada Gambar 8(a). Pengujian di sini menggunakan model EfficientDet-Lite2 untuk skenario deteksi warna kontras dan mirip.

Pada Gambar 8(a), apabila kecepatan objek diubah dari kecepatan rendah ke tinggi, maka secara umum, kinerja akan mengalami penurunan. Kinerja yang diukur di sini adalah *precision*, *recall*, dan *F1-score*. Penurunan kinerja terjadi baik untuk objek berwarna mirip maupun

berwarna kontras. Meskipun demikian, penurunan pada objek berwarna mirip dengan *background* akan lebih besar dibandingkan objek berwarna kontras. Objek berwarna mirip mengalami penurunan F1-score sebesar 0,439 sementara objek kontras akan turun sebesar 0,187.

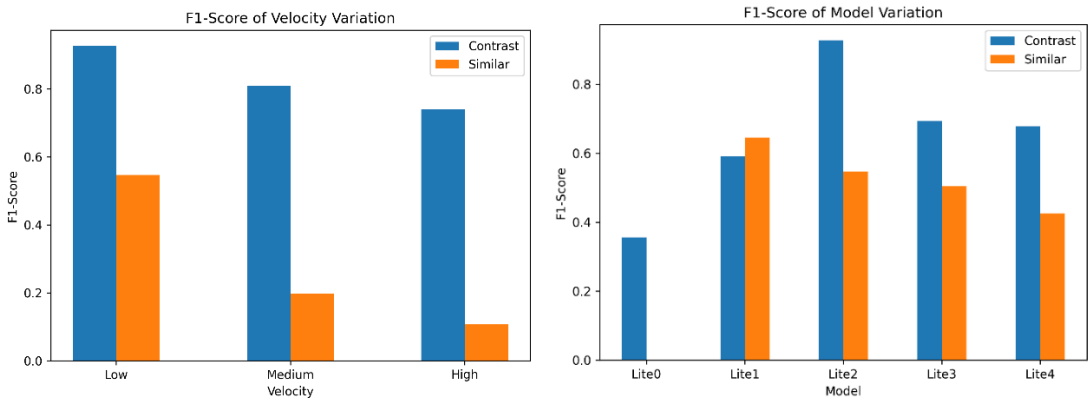
Penurunan kinerja ini dapat disebabkan oleh *blur* pada citra akibat objek berkecepatan tinggi. Kamera dan model deteksi yang mengakuisisi citra secara lambat dapat menyebabkan efek tersebut pada citra yang diprediksi. Selain blur, objek juga akan berubah bentuk menjadi lebih lonjong. Perubahan pada citra ini dapat mempengaruhi perilaku model dalam prediksi sehingga opsi terbaik adalah membuat kecepatan objek sekecil mungkin.



Gambar 7. Perbandingan *Frame Rate* dan Waktu Tunda Deteksi

3.4 Perubahan Kinerja terhadap Warna Objek

Hal terakhir yang diuji adalah pengaruh warna objek terhadap kinerja deteksi. Hasil pengujian ini dapat diamati pada Gambar 8. Pada gambar tersebut, secara umum, objek dengan warna kontras memiliki kinerja deteksi yang lebih baik dibandingkan objek berwarna mirip dengan konveyor. Pengecualian terjadi pada model EfficientDet-Lite1 tetapi kinerja model tersebut tidak jauh berbeda untuk kedua skenario yaitu dengan perbedaan nilai 0,054. Bukti lain dari penurunan kinerja juga ditemukan di bagian sebelumnya. Ketika kecepatan ditingkatkan, objek berwarna mirip akan mengalami penurunan signifikan.



(a) (b)

Gambar 8. Perbandingan *F1-Score* untuk (a) Kecepatan Berbeda dan (b) Model Berbeda

Kinerja deteksi secara umum lebih baik pada objek berwarna kontras dapat disebabkan oleh fitur warna yang lebih mudah dibedakan oleh model. Warna pada citra dapat diketahui langsung dari nilai matriks citra dan distribusinya. Warna tidak membutuhkan konvolusi lebih

lanjut untuk dapat dikenali model. Oleh karena itu, pemilihan objek dengan warna kontras adalah opsi terbaik untuk menjalankan deteksi.

Temuan lain dari pengujian ini adalah EfficientDet-Lite2 pada deteksi warna kontras memiliki *F1-score* tertinggi, yaitu 0,926, meskipun bukan model terbesar. Apabila akurasi ini dihubungkan dengan jumlah parameter, waktu tunda, dan frame rate, EfficientDet-Lite2 dapat dinyatakan sebagai model paling optimal selama pengujian. Hal ini menunjukkan bahwa jumlah parameter tidak menjamin kinerja deteksi terbaik.

Model pembelajaran mesin dibentuk oleh data, *hyperparameter*, dan algoritma. Meskipun algoritma dan *hyperparameter* sama, data yang berbeda dapat menyebabkan kinerja berbeda. Di sini, data yang digunakan untuk melatih dapat mengalami ketidak-seimbangan kelas (*class imbalance*) setelah pengacakan akibat tidak digunakannya *random seed generator*. Penambahan *seed* akan membuat pengacakan memiliki urutan data yang sama sehingga mencegah data dengan kesamaan kelas berkumpul di satu urutan. Dengan demikian, perbaikan model dapat dilakukan dengan memperbaiki representasi data.

4. KESIMPULAN

Penelitian ini menghasilkan detektor objek yang bergerak di atas konveyor peraga. Dengan EfficientDet-Lite sebagai model dan Raspberry Pi 4 sebagai perangkat keras, beberapa temuan diperoleh: peningkatan kecepatan objek dapat menurunkan kinerja deteksi, pengunaan warna kontras dengan konveyor akan meningkatkan kinerja, dan jumlah parameter dan ukruan model tidak menjamin kinerja terbaik. Pengunaan model yang lebih besar akan meningkatkan biaya berupa waktu tunda, konsumsi daya, dan penurunan *frame rate*. Oleh karena itu, model paling optimal yang dihasilkan penelitian adalah EfficientDet-Lite2 dengan F1-score 0,926, kisaran konsumsi energi 6,5 Watt, dan kisaran frame rate 2,7 FPS. Dari penelitian ini, peningkatan kinerja deteksi dapat dilakukan dengan menambahkan kuantisasi setelah pelatihan, pemakaian hardware accelerator, atau memperbarui model secara berkala.

5. SARAN

Penelitian lebih lanjut dapat dilakukan terhadap model deteksi ringan lain, seperti model dari keluarga YOLO, sehingga dapat diketahui perbandingan kinerja model berbeda terhadap objek bergerak. Pengujian kinerja sebaiknya memanfaatkan perangkat keras, instrumen, dan metrik yang sama sehingga perbandingan mudah dilakukan. Topik lain yang menarik adalah mengamati pengaruh kuantisasi pasca-pelatihan terhadap kinerja model deteksi.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tim Redaksi Jurnal Teknika Politeknik Negeri Sriwijaya yang telah memberi memberi kesempatan, sehingga artikel ilmiah ini dapat diterbitkan.

DAFTAR PUSTAKA

[1] Karim, A. S., and A'isy, L. R., 2019, *Sistem Informasi Lalu Lintas di Kota Bandar Lampung Berbasis CCTV*, *Teknika*, No.1, Vol.13, https://doi.org/10.5281/zenodo.3461326

[2] Putra, R. B., and Saputra, K., 2022, *Sistem Pengukur Tinggi Tanaman dengan Computer Vision dan Raspberry PI*, *Teknika*, No.1, Vol.16

[3] Mittal, S., Khan, M. A., Romero, D., and Wuest, T., 2019, *Smart manufacturing: Characteristics, technologies and enabling factors*, *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, No.5, Vol.233,

https://doi.org/10.1177/0954405417736547

[4] Tan, M., Pang, R., and Le, Q. v., 2020, *EfficientDet: Scalable and Efficient Object Detection*, *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10778–10787, https://doi.org/10.1109/CVPR42600.2020.01079

[5] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X., 2016, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, *ArXiv (Cornell University)*, https://doi.org/10.48550/arXiv.1603.04467

[6] Tan, M., and Le, Q., 2019, *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th International Conference on Machine Learning*, Vol.97, 6105–6114, https://proceedings.mlr.press/v97/tan19a.html

[7] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C., 2018, *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4510–4520, https://doi.org/10.1109/CVPR.2018.00474

[8] Ioffe, S., and Szegedy, C., 2015, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, *International Conference on Machine Learning*, 448–456.

[9] Ramachandran, P., Zoph, B., and Le, Q. v., 2017, *Searching for Activation Functions*, *arXiv (Cornell University)*, https://doi.org/10.48550/arXiv.1710.05941

[10] Hu, J., Shen, L., and Sun, G., 2018, *Squeeze-and-Excitation Networks*, *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7132–7141. https://doi.org/10.1109/CVPR.2018.00745

[11] Chollet, F., 2017, *Xception: Deep Learning with Depthwise Separable Convolutions*, *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1800–1807, https://doi.org/10.1109/CVPR.2017.195

[12] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H., 2017, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, *ArXiv (Cornell University)*, https://doi.org/10.48550/arXiv.1704.04861

[13] Lin Tsung-Yi and Maire, M. and B. S. and H. J. and P. P. and R. D. and D. P. and Z. C. L., 2014, *Microsoft COCO: Common Objects in Context*, In T. and S. B. and T. T. Fleet David and Pajdla (Ed.), *Computer Vision – ECCV 2014*, Springer International Publishing.

[14] Everingham, M., van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A., 2010, *The Pascal Visual Object Classes (VOC) Challenge*, *International Journal of Computer Vision*, No.2, Vol.88, 303–338, https://doi.org/10.1007/s11263-009-0275-4

[15] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollar, P., 2020, *Focal Loss for Dense Object Detection*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, No.2, Vol.42, 318–327, https://doi.org/10.1109/TPAMI.2018.2858826