

Deteksi Objek pada Citra X-Ray dari COVID-19 Menggunakan Model YOLOv4 dan YOLOv5



Oleh

Nama: Pandega Abyan Zumarsyah

NIM : 18/424977/TK/46672

Fakultas Teknik

Universitas Gadjah Mada

2021

Intisari

Deteksi objek merupakan bagian dari Computer Vision dengan tujuan menemukan dan mengklasifikasikan objek tertentu yang ada pada suatu citra. Di bidang medis, deteksi objek dapat diterapkan pada citra X-Ray dada sehingga dapat membantu mendeteksi dan melokalisasi penyakit pernafasan, salah satunya yaitu COVID-19.

Pada proyek ini, kami menggunakan model YOLOv4 dan YOLOv5 untuk melakukan deteksi objek pada citra X-Ray dada terkait COVID-19. Berdasarkan pengujian, ukuran citra dan ukuran model tidak terlalu memberikan pengaruh sehingga kami menggunakan yang kecil agar lebih ringan.

Hasilnya secara umum masih kurang baik dengan mAP yang masih berada di angka 0.2. Bagaimana pun juga, proyek ini memberikan berbagai pembelajaran baru terkait penggunaan YOLOv4 dan YOLOv5 pada citra X-Ray dada. Kami mendapati bahwa penggunaan *pre-trained weight* memang mampu membuat model belajar dengan lebih cepat, namun sangat rawan akan terjadinya *over-fitting*. Kami juga mendapati bahwa YOLOv4 mampu mengimbangi YOLOv5 dari segi hasil akhir, meski memang YOLOv5 mampu belajar lebih cepat dan lebih ringan.

Daftar Isi

Intisari.....	2
Daftar Isi.....	3
I. PENDAHULUAN.....	4
A. Latar Belakang.....	4
B. Tujuan.....	4
II. KAJIAN PUSTAKA	5
A. Konsep Dasar Deteksi Objek	5
B. Parameter Ukur Terkait Deteksi Objek	6
C. Model YOLOv4 dan YOLOv5.....	7
D. Implementasi YOLOv4 dan YOLOv5 pada Deteksi Kepala Gandum	8
E. Implementasi YOLO pada Citra X-Ray Dada untuk Deteksi COVID-19	9
III. METODOLOGI	9
A. Alat dan Bahan.....	9
B. Metode.....	11
C. Langkah Uji.....	12
IV. HASIL DAN ANALISIS	13
A. Pengujian Awal	13
B. Pengujian Final	15
V. KESIMPULAN.....	19
VI. DAFTAR PUSTAKA	20

I. PENDAHULUAN

A. Latar Belakang

Deteksi objek merupakan bagian penting dari Computer Vision. Konsep dari deteksi objek adalah menemukan dan mengklasifikasikan objek tertentu yang ada pada suatu citra. Objek yang dikenali bisa sangat beragam, mulai dari manusia dan mobil pada citra sehari-hari, sampai nodul paru-paru pada citra medis. Karenanya, penerapannya juga bisa sangat luas, di antaranya pada industri keamanan, video, olahraga, dan medis. [1] Umumnya, deteksi objek terkait dengan pemberian *bounding box* atau kotak pembatas pada suatu objek tertentu dalam suatu citra. Pada kotak tersebut, terdapat keterangan nama atau label dari objeknya.

Di bidang medis, deteksi objek dapat diterapkan pada citra X-Ray. Dengan deteksi objek, bagian tertentu pada citra yang dianggap bermasalah secara medis dapat dideteksi dan diberi *bounding box*. Penggunaan deteksi objek pada citra X-Ray dapat membantu mendeteksi dan melokalisasi penyakit pernafasan, salah satunya COVID-19.

COVID-19 (coronavirus disease 2019) merupakan penyakit pernafasan baru yang telah menyebar ke seluruh dunia dan bahkan menjadi pandemi. Metode diagnosis yang umum digunakan adalah melalui RT-PCR (real-time reverse-transcription polymerase chain reaction). Karena RT-PCR memiliki sensitivitas yang rendah, citra radiologis juga sering digunakan. Citra yang menjadi gold standard adalah CT, namun X-Ray juga bisa digunakan. [2]

Dalam proyek ini, kami melakukan deteksi objek pada citra X-Ray dari COVID-19 dengan menggunakan model YOLOv4 dan YOLOv5. Kedua model itu merupakan algoritma *state-of-the-art* dalam bidang deteksi objek secara cepat. Akurasiya juga secara umum tidak kalah jauh dibanding algoritma deteksi objek lain yang jauh lebih lambat.

Dataset yang digunakan dalam proyek ini didapatkan dari Kaggle dengan nama *SIIM-FISABIO-RSNA COVID-19 Detection*. Dataset ini terdiri atas ribuan citra X-Ray terkait penyakit COVID-19. Terdapat pula data anotasi atau label *bounding box* untuk menandai abnormalitas terkait COVID-19 yang ada pada citra. [3] Dengan begitu, dataset ini bisa digunakan untuk pengembangan deteksi objek, meski perlu beberapa pengolahan.

Dataset pada Kaggle itu hadir sebagai sebuah kompetisi. Karenanya, sudah ada banyak kode/program deteksi objek yang dibuat untuk dataset itu. Dalam proyek ini, kami tentu banyak belajar dari berbagai kode/program yang sudah ada. Namun, kami juga berusaha memperbaiki dan mengembangkan dari yang sudah ada itu. Pengujian dan perbandingan YOLOv4 dan YOLOv5 sekaligus setahu kami juga belum pernah diterapkan pada dataset tersebut. Dengan itu semua, harapannya kami bisa lebih mendalami bidang deteksi objek ini dan ikut berkontribusi secara lebih luas di masa depan.

B. Tujuan

1. Mengembangkan model deteksi objek untuk mendeteksi kelainan atau abnormalitas pada citra X-Ray terkait COVID-19
2. Membandingkan performa YOLOv4 dan YOLOv5 pada dataset tertentu
3. Memahami konsep dan penerapan model deteksi objek secara umum, termasuk konsep *neural network* yang mendasari model tersebut

II. KAJIAN PUSTAKA

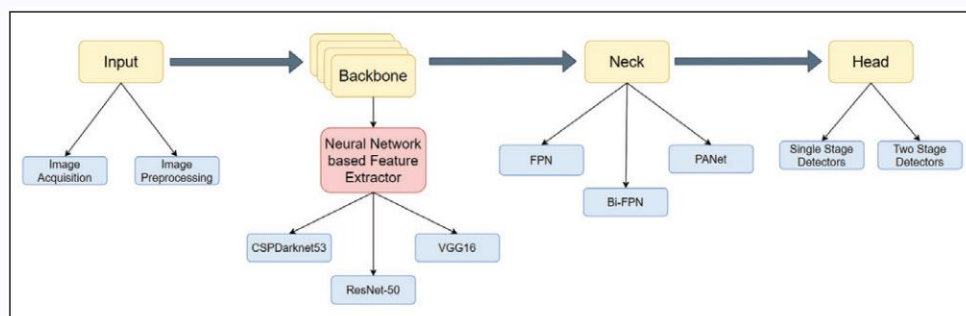
A. Konsep Dasar Deteksi Objek

Konsep dari deteksi objek adalah menemukan dan mengklasifikasikan objek tertentu yang ada pada suatu citra. Umumnya, objek yang didapatkan kemudian diberi *bounding box* atau kotak pembatas. Setiap kotak pembatas memiliki *confidence score* yang menunjukkan seberapa percaya suatu model dengan prediksinya. Jika skornya di bawah batas nilai tertentu, kotaknya tidak akan ditampilkan. [1]

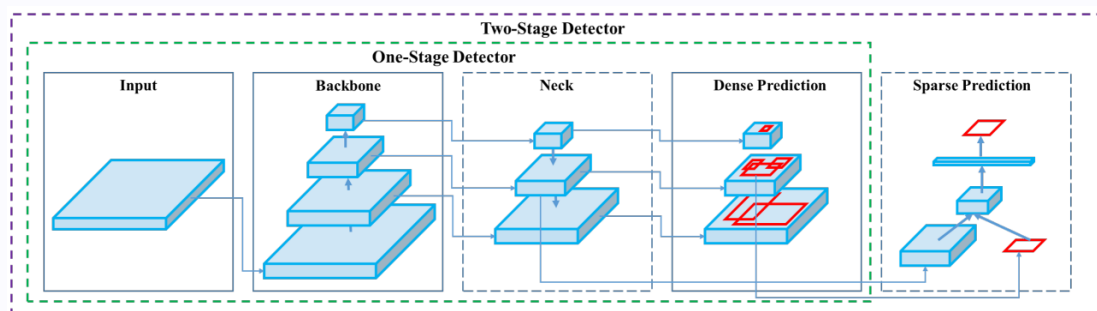
Deteksi objek berbasis CNN memiliki beberapa bagian yang secara umum dapat dibagi menjadi empat: [1][4]

- *The Input*
Ini terdiri atas input yang diterima oleh modelnya, dapat berupa citra statis atau video. Pada bagian ini, dilakukan akuisisi citra dan *pre-processing*.
- *The Backbone*
Ini adalah bagian yang di dalamnya terdapat CNN untuk melakukan ekstraksi fitur dari citra. Bagian ini dapat berisi model CNN seperti ResNet-50, CSPDarkNet53, dan lainnya
- *The Neck*
Bagian ini berisi blok *network* tambahan yang berkaitan dengan diskriminasi fitur dan keandalan dari model. Yang digunakan pada bagian ini di antaranya adalah FPN (*Feature Pyramid Network*) dan PAN (*Path Aggregation Network*).
- *The Head*
Bagian akhir yang mengurus prediksi dan klasifikasi objek. Ini dapat berupa dua tahap (*sparse*) seperti Faster R-CNN atau satu tahap (*dense*) seperti YOLO.

Berkaitan dengan ini, Gambar 1 menunjukkan diagram alir keempat bagian beserta sedikit rinciannya. Sementara itu, Gambar 2 menunjukkan struktur dari keempat bagian.



Gambar 1: *Framework* Model Objek Deteksi dengan Empat Bagian [1]



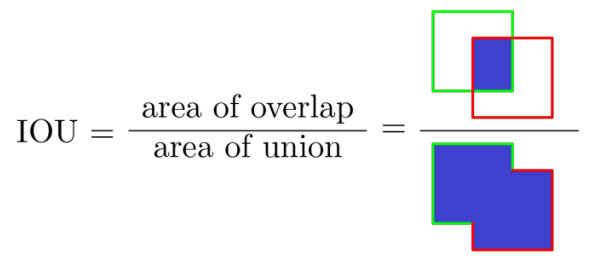
Gambar 2: Struktur Model Objek Deteksi [4]

B. Parameter Ukur Terkait Deteksi Objek

Deteksi objek memiliki berbagai parameter ukur tersendiri seperti IoU, *precision*, *recall*, AP, mAP, dan lainnya. Parameter itu penting untuk menilai performa dari model deteksi objek yang ada. Pada bagian ini, akan dibahas beberapa parameter ukur yang dilibatkan dalam proyek ini.

- IoU (*Intersection over Union*)

Hasil akhir dari deteksi objek di antaranya adalah *bounding box*. Selain *bounding box* hasil prediksi, kita juga memiliki *bounding box* sebenarnya berdasarkan *ground truth*. IoU didefinisikan sebagai rasio antara *intersection* kedua *bounding box* itu dengan *union* nya. Nilainya adalah antara 0 (tidak ada *intersect* sama sekali) atau 1 (*perfectly matched*). Ilustrasi terkait IoU dapat dilihat pada Gambar 3. [5]


$$\text{IoU} = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{blue area}}{\text{combined area of green and red boxes}}$$

Gambar 3: Ilustrasi IoU [5]

- *Precision* dan *Recall*

Precision terkait dengan kemampuan model untuk mengidentifikasi objek yang relevan saja, tidak menghasilkan deteksi berlebihan namun juga tidak kurang. Sementara itu, *Recall* terkait dengan kemampuan model untuk menemukan semua kasus relevan (semua *bounding box* sesuai *ground truth*). Dengan TP, FP, FN berturut-turut sebagai *true positive*, *false positive*, *false negative* maka *Precision* (*Pr*) dan *Recall* (*Rc*) dapat didefinisikan sebagai: [5]

$$Pr = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}} \quad Rc = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}}$$

- AP (*Average Precision*)

Selain menghasilkan *bounding box* dan kelasnya, deteksi objek juga menghasilkan *confidence* dari prediksinya. Itu dapat dimasukkan dalam perhitungan *precision* dan *recall*, yaitu dengan mendefinisikan TP, FP, dan FN sebagai fungsi dari suatu *confidence threshold* τ . Pada dasarnya, jika *confidence* yang didapatkan kurang dari τ , maka prediksinya tidak dianggap. [5]

Ketika τ naik, nilai TP dan FP akan turun sementara FN akan naik. Idealnya, ketika τ diubah-ubah, *precision* akan tetap tinggi ketika *recall* nya naik. Itu membuat kurva *precision* terhadap *recall* memiliki AUC (*area under the curve*) yang tinggi. [5]

Perhitungan AP secara singkat tidak mudah untuk dijelaskan, bahkan mungkin memiliki beberapa metode perhitungan. Meski namanya, *average precision*, perhitungan AP tetap melibatkan nilai *recall* dan tentunya nilai *threshold* τ . [5]

- mAP (*mean Average Precision*)

AP sebelumnya didapatkan secara individual untuk tiap kelas. Untuk mendapatkan gambaran yang lebih umum, kita bisa menggunakan mAP. Dengan C adalah banyak kelas, mAP dapat didefinisikan sebagai: [5]

$$mAP = \frac{1}{C} \sum_{i=1}^C AP_i$$

- mAP dengan *IoU threshold*

Nilai IoU juga sering digunakan sebagai *threshold* ketika menghitung AP atau mAP. Ketika nilai IoU yang didapatkan kurang dari *threshold*, maka deteksinya diabaikan dan tidak teranggap. Di antara mAP yang sering digunakan adalah mAP@.5 yaitu mAP dengan *threshold* IoU sebesar 0.5. Ada juga mAP@.5:.95 yang mana merupakan rata-rata untuk berbagai *threshold* $t = [0.5, 0.55, 0.6, \dots, 0.95]$. [5]

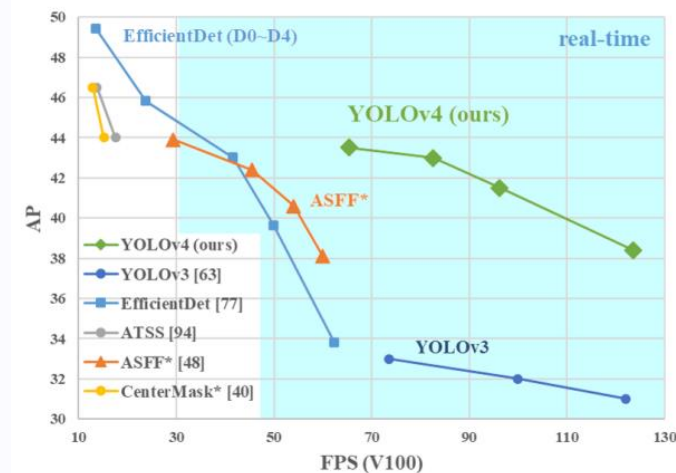
C. Model YOLOv4 dan YOLOv5

YOLO (You Only Look Once) merupakan pendekatan yang relatif baru dalam deteksi objek. Sebelumnya, deteksi objek hanyalah menggunakan klasifikasi untuk melakukan deteksi. Pada YOLO versi pertama, deteksi objek diperlakukan sebagai masalah regresi.

Seiring dengan berjalannya waktu, YOLO berkembang dengan cukup pesat sehingga menjadi makin cepat dan makin akurat. Yang saat ini menjadi *state-of-the-art* dalam deteksi objek adalah YOLOv4 dan YOLOv5.

Dibanding YOLO versi sebelumnya, nilai FPS dari YOLOv4 12% lebih tinggi sementara nilai mAP nya 12% lebih tinggi. YOLOv4 juga cocok untuk komputasi paralel dengan memanfaatkan GPU. [1] Namun, GPU yang digunakan pun tidak perlu yang mahal dan tingkat lanjut, tetapi bisa menggunakan GPU biasa saja. [4]

Terdapat beberapa perubahan dan tambahan pada struktur YOLOv4. Pada bagian *The Neck*, FPN yang sebelumnya digunakan diganti menjadi PAN. YOLOv4 juga menggunakan *Bag of Freebies* dan *Bag of Specials* untuk bagian *backbone* dan detektornya. Semua itu memang terbukti dapat meningkatkan akurasi. [4]



Gambar 4: Perbandingan YOLOv4 dengan model lain [4]

Dengan semua itu, YOLOv4 menjadi model deteksi objek yang cukup ideal. Performanya sangat cepat dengan akurasi yang cukup tinggi. Gambar 4 menunjukkan perbandingan antara YOLOv4 dengan model *state-of-the-art* lainnya. [4] Perlu diketahui juga bahwa peneliti YOLOv4 juga sudah mengembangkan Scaled-YOLOv4. Dengan begitu, YOLOv4 dapat diimplementasikan di berbagai perangkat, mulai dari *embedded* sampai *advanced cloud*, secara optimal tanpa memberatkan. [6]

Kemudian, satu bulan setelah YOLOv4 rilis, peneliti lain datang dengan YOLOv5. Sebenarnya, pengembangan YOLOv5 ini dimulai tidak lama setelah pengembangan YOLOv4 dimulai. Karenanya, proyek YOLOv5 ini dan penamaannya menjadi kontroversi di komunitas. Apalagi, sampai saat ini, paper resmi dari YOLOv5 juga belum ada. [7]

Bagaimana pun juga, YOLOv5 memiliki banyak kelebihan dibanding YOLOv4. Meski teknologi yang digunakan mirip, namun *framework* pengembangannya sangat berbeda. YOLOv5 dikembangkan dengan bahasa Python dan langsung bisa terintegrasi dengan PyTorch. Karena menggunakan PyTorch, pengguna menjadi lebih familiar dan pengembangan lanjut pun menjadi mudah karena komunitas PyTorch sudah besar. Dengan melihat kodenya secara langsung, struktur dari YOLOv5 secara umum: [7]

- *Backbone: Focus Structure, CSP Network*
- *Neck: Blok SPP, PANet*
- *Head: Head* dari YOLOv3 dengan GIoU-loss

Inovasi yang menarik pada YOLOv5 ini adalah terkait *anchor box*. Sebelumnya, *anchor box* ini kurang fleksibel sehingga ketika digunakan pada dataset yang cukup unik, hasilnya kurang memuaskan. Peneliti YOLOv5 pun mendesain agar modelnya juga belajar untuk menentukan *anchor box* yang tepat untuk data tertentu. Dengan semua itu, YOLOv5 terbukti secara umum lebih baik daripada YOLOv4. [7]

D. Implementasi YOLOv4 dan YOLOv5 pada Deteksi Kepala Gandum

Saat ini, bidang agrikultur berkembang pesat dengan bantuan AI, salah satunya untuk identifikasi sifat tanaman. Dengan pengolahan citra berbasis AI, kondisi dari tanaman dapat diidentifikasi dengan lebih mudah. Salah satu tanaman pangan yang banyak ditanam adalah gandum. Bagian dari gandum yang dapat memberikan berbagai informasi terkait kondisinya adalah bagian kepala. Maka, deteksi kepala gandum menjadi salah satu langkah untuk mengidentifikasi kondisi gandum melalui AI. [8]

Penelitian ini menggunakan GWHD (*Global Wheat Head Dataset*) yang memiliki 4700 citra RGB dengan di dalamnya terdapat 190 ribu kepala gandum terlabeli. Dalam penelitian ini, ada berbagai pengolahan sebelum memasukan citranya ke tahap *training*. Terdapat pembersihan data untuk menghindari citra yang kurang baik, pembagian data untuk *test* dan *train*, dan ada pula augmentasi untuk meningkatkan variabilitas data. [8]

Ada dua model yang digunakan, Faster R-CNN dan YOLOv5. Model YOLOv5 dipilih karena konsep *genetic algorithm* untuk membuat *anchor* baru secara otomatis dianggap bermanfaat untuk menghadapi data GWHD. Itu karena GWHD memiliki kotak pembatas yang kecil dan sangat banyak, berbeda dari deteksi objek umumnya. Hasilnya, model YOLOv5 menang mutlak atas model Faster R-CNN. [8]

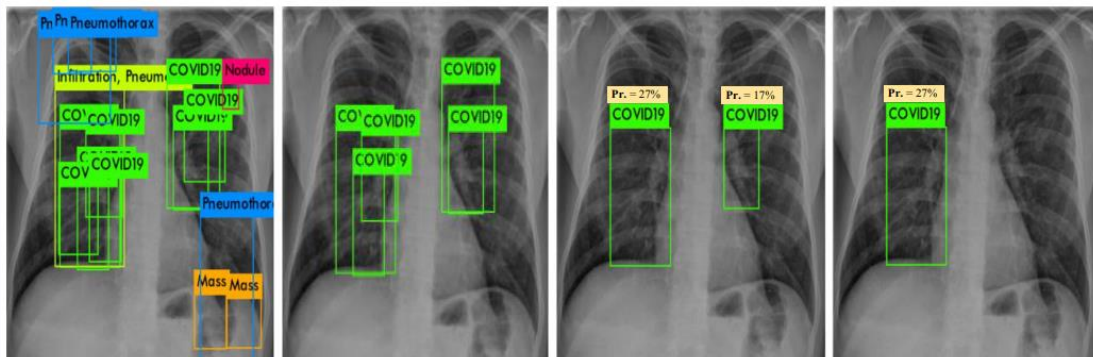
E. Implementasi YOLO pada Citra X-Ray Dada untuk Deteksi COVID-19

Pada penelitian ini, *deep learning* dengan YOLO digunakan untuk melakukan deteksi pada citra X-Ray dada. COVID-19 dibedakan dari delapan penyakit pernafasan lain melalui *multiclass recognition problem*. Metode seperti augmentasi dan *transfer learning* digunakan untuk meningkatkan hasilnya. Terdapat pula *five-fold test* untuk validasi. [2]

Penelitian ini menggunakan dataset ChestX-Ray8 yang berisi citra X-Ray untuk delapan jenis penyakit pernafasan lengkap dengan label kelas dan kotak pembatasnya. Selain itu, digunakan pula dua dataset COVID-19 yang digunakan, yaitu dataset yang telah dikumpulkan Cohen et al dan dataset yang telah dikumpulkan tim riset dari Universitas Qatar. Keduanya tersedia secara publik dengan label kelasnya, namun tidak ada anotasi terkait kotak pembatasnya. [2]

Struktur yang diajukan cukup kompleks, namun dasarnya ada dua tahap. Awalnya, model berbasis YOLO digunakan untuk memprediksi kotak pembatas. YOLO dipilih karena meninjau citra secara keseluruhan, tidak melalui *window* saja. Kemudian, untuk tiap kotak pembatas, dilakukan prediksi untuk menentukan penyakitnya. [2]

Gambar 5 menunjukkan hasil dari penelitian ini. Dari Gambar 5, kita melihat pengaruh *confidence threshold* terhadap banyaknya kotak pembatas yang terlihat. Kita juga melihat bahwa dari satu citra, bisa didapatkan banyak kotak pembatas dengan prediksi penyakit berbeda-beda. Evaluasi yang dijalankan dan parameter yang digunakan pada penelitian ini cukup banyak. Secara umum, hasil yang didapatkan sudah bagus. Akurasi deteksinya mencapai 90,67% sementara akurasi klasifikasinya mencapai 97,40%. [2]



Gambar 5: Hasil Deteksi Objek dengan Variasi *Threshold* [2]

III. METODOLOGI

A. Alat dan Bahan

1. Dataset

Proyek kami menggunakan dataset dari Kaggle yang bernama *SIIM-FISABIO-RSNA COVID-19 Detection*. Dataset ini terdiri atas ribuan citra X-Ray dada terkait penyakit COVID-19. Terdapat pula data anotasi atau label *bouding box* untuk menandai abnormalitas terkait COVID-19 yang ada pada citra. Namun, citranya disimpan dalam format file .dcm atau DICOM (Digital Imaging and Communications in Medicine). [3] Itu memang sudah menjadi standar dalam citra medis, namun kami belum familiar dengannya.

Selain itu, terdapat masalah lain terkait akses ke datasetnya. Untuk bisa mengaksesnya, kami perlu registrasi dan verifikasi. Setelah mencoba berkali-kali, kami tidak bisa verifikasi karena selalu bermasalah ketika mengisi nomor telepon.

Bagaimana pun juga, dataset ini sangat jarang dan sangat berharga untuk dilewatkan begitu saja. Setelah mencari, kami mendapatkan beberapa hasil olahan dari dataset tadi. Ada pihak yang menyediakan seluruh citra dalam bentuk .jpg atau .png biasa, namun tidak menyediakan anotasinya. [9] Di sisi lain, ada pihak yang hanya menyediakan anotasi untuk data *train* saja, bahkan tanpa menyediakan citranya. [10] Dengan menggabungkan keduanya, kami bisa menggunakan data *SIIM-FISABIO-RSNA COVID-19 Detection*.

Meski hanya mendapat data *train* saja, sebenarnya itu sudah cukup banyak sehingga nantinya bisa dibagi menjadi *train-validation-test*. Setelah direduksi, dataset terdiri dari 4598 citra dan 8157 label *bounding box* yang meliputi tiga kelas: ‘atypical’(629 label), ‘indeterminate’(1494), dan ‘typical’(6034). Dari sini, terlihat bahwa datanya memang kurang seimbang.

2. Platform

Kami mengerjakan proyek ini langsung Google Colaboratory sehingga dapat memanfaatkan *resource* seperti RAM, Disk, dan GPU yang cukup mumpuni. Adanya GPU sangat penting karena mampu meningkatkan kecepatan *training* hingga berlipat-lipat.

Meski gratis, penggunaan GPU pada Google Colaboratory memiliki batas tertentu. Jika Google merasa kami telah menggunakan GPU terlalu lama, akses ke GPU akan diputus untuk sementara. Karena tidak banyak hal yang bisa dilakukan, pembatasan itu menjadi rintangan utama yang memperlambat kerja kami dalam mengembangkan model.

3. Model

Kami menggunakan model YOLOv4 dan YOLOv5 dengan melakukan *clone* pada *repository* di Github. Untuk YOLOv5, kami menggunakan *repository* aslinya [10][11] karena memang mudah digunakan. Dia bahkan memiliki fitur seperti koneksi dengan WandB dan Tensorboard yang bisa sangat membantu dalam proses pengembangan.

Untuk YOLOv4, kami tidak menggunakan *repository* asli yang dibangun langsung dari DarkNet, tetapi kami menggunakan *repository* YOLOv4 yang diimplementasikan menggunakan PyTorch. Selain lebih mudah, itu juga agar YOLOv4 bisa dibandingkan dengan YOLOv5 yang juga menggunakan PyTorch. *Repository* yang kami gunakan juga memang didesain agar menyerupai *repository* YOLOv5. Dari *repository* itu, kami juga memilih *branch* [13] yang format modelnya sama dengan format model pada YOLOv5, yaitu format .yaml untuk *config* dan format .pt untuk *weight*. Jika dilihat file .yaml nya, struktur YOLOv4 di sini cukup mirip dengan yang dijelaskan pada Scaled-YOLOv4.[6]

4. Modul Python

Terdapat beberapa modul Python yang kami gunakan. Ada beberapa modul dasar umum seperti numpy, pandas, matplotlib, shutil, dan os. Ada pula beberapa modul dengan fungsi yang lebih khusus seperti OpenCV, scikit-learn, dan yaml. Modul khusus yang paling penting tentu adalah PyTorch yang mana ia menjadi dasar untuk model YOLOv4 dan YOLOv5 yang digunakan.

B. Metode

1. Penyiapan Data

Pada bagian awal, kami mengolah datanya agar bisa digunakan di YOLO. Awalnya, file .csv yang berisi berbagai data terkait citranya, di-load sebagai Dataframe. Di sana, sudah terdapat alamat file citra terkait, kelas, koordinat *bounding box*, dan lainnya.

Pengolahan pertama yaitu menghilangkan data dengan kelas ‘normal’ yang mana dia tidak memiliki *bounding box* sehingga kurang bisa dipakai. Selanjutnya yaitu mengurangi ukuran dataset jika diperlukan. Ini dilakukan terutama ketika masih tahap *trial-error*.

Selanjutnya, dibuat Dataframe yang memuat id citra yang unik saja. Ini penting mengingat satu citra dapat memiliki lebih dari satu *bounding box*. Ini berkaitan dengan kelola file citra yang akan dilakukan. Setelah itu, dilakukan *splitting* untuk membagi data menjadi *train-validation-test* dengan rasio 70:15:15. Salah satu tujuan dari Dataframe dengan id citra unik adalah agar tidak ada citra yang berada di *train* dan *validation* sekaligus, atau kasus serupa itu.

Kemudian, file citra dikelola agar menempati folder tertentu sesuai dengan jenis *split* nya. Citra yang masuk ke *train* akan masuk ke folder *train*. Selain itu, dibuat pula file config dengan format .yaml untuk mencatat lokasi dari foldernya.

Pengolahan selanjutnya adalah pembuatan file .txt untuk mencatatkan nama file citra dan label *bounding box* nya. Kami membuat file train.txt val.txt test.txt yang masing-masing memuat daftar citra yang ada di dalam folder terkait. Selain itu, pada folder tertentu, kami membuat file .txt berisi koordinat *bounding box* dan kelasnya.

Gambar 6 menunjukkan struktur folder sesuai format YOLO. Di dalam images/.../ terdapat file-file citra. Sementara itu, di dalam labels/.../ terdapat file-file .txt yang berisi label *bounding box*, satu file untuk satu citra. Untungnya, struktur folder untuk YOLOv5 dan YOLOv4 sama sehingga tidak perlu mengolah kembali.



Gambar 6: Struktur Folder untuk YOLO

2. Training Model

Bagian ini pada dasarnya sederhana saja, hanya perlu satu baris perintah. Namun sebelumnya, kita perlu memastikan bahwa file config .yaml yang sebelumnya dibuat sudah berada pada lokasi yang tepat. Dari file inilah YOLO akan mengetahui posisi file citra dan file label.

Selain itu, pada YOLOv5, kita juga bisa menggunakan WandB untuk membantu mendokumentasi, mengawasi, dan mengevaluasi proses *training*. Pada program, kita hanya perlu login dan verifikasi saja. Ketika *training* berjalan, WandB akan mencatat berbagai hal secara otomatis. Ini terbukti membantu untuk membandingkan berbagai hal.

Baik untuk YOLOv4 maupun YOLOv5, proses *training* dilakukan dengan menjalankan file `train.py`. Kita bisa memberikan beberapa parameter seperti ukuran citra, ukuran *batch*, banyak *epoch*, *pre-trained weight*, *model config*, dan lainnya. Ketika *training* berjalan, kita bisa mendapatkan beberapa data seperti mAP, penggunaan memori GPU, *precision*, *recall*, dan lainnya.

Ada beberapa jenis model yang akan kami ujikan, semuanya merupakan model YOLO versi ringan atau *small*. Yang pertama adalah model YOLOv4 dan YOLOv5 dengan *pre-trained weight* yang sudah tersedia dalam bentuk file `.pt`. Yang kedua adalah model YOLOv4 dan YOLOv5 yang masih kosong, yaitu dengan menggunakan file `config.yaml`. Yang terakhir, harapannya kami bisa mencoba mengedit file `config.yaml` dan mengujinya.

3. Evaluasi

Evaluasi yang dimaksud di sini terdiri atas tiga hal. Pertama, menampilkan berbagai statistik terkait *training* yang telah dibuat secara otomatis. Ini meliputi kurva P, kurva R, kurva PR, dan statistik lainnya dalam bentuk gambar.

Yang kedua adalah pengujian model yang telah dibuat pada data *test*. Ini dapat dilakukan dengan menjalankan file `test.py` pada YOLOv4 dan `val.py` pada YOLOv5. Di sini, kita juga bisa memberikan beberapa parameter seperti ukuran citra, *confidence threshold*, *IoU threshold*, dan lainnya. Hasilnya adalah nilai parameter ukur seperti *precision*, *recall*, dan mAP. Selain itu, kita juga bisa menjalankan file `predict.py` dengan hasilnya adalah citra yang sudah diberi *bounding box* hasil prediksi.

Yang terakhir adalah membandingkan berbagai statistik dari berbagai pengujian dengan berbagai model. Ini dilakukan dengan menampilkan data-data yang sebelumnya telah disimpan. Bagian ini tentu hanya bisa dilakukan di akhir proyek.

C. Langkah Uji

Pada bagian sebelumnya, kami menjelaskan metode yang sifatnya ideal dan masih rencana. Pada bagian ini, kami menjelaskan pelaksanaan dan berbagai tantangan yang ada.

Ini dimulai dengan mencari dataset yang bisa digunakan. Sejak awal, kami sudah menemukan dataset *SIIM-FISABIO-RSNA COVID-19 Detection*, namun aksesnya tidak mudah. Karenanya, kami perlu mengaksesnya dengan cara lain. Banyak pihak yang mengubah dataset tadi agar lebih mudah diakses, namun kebanyakan hanya menyediakan citranya saja tanpa anotasinya. Meski akhirnya, kami menemukan data anotasinya.

Dataset yang digunakan merupakan dataset untuk kompetisi sehingga sudah ada banyak kode yang tersedia, kebanyakan menggunakan YOLOv5. Kami tentu belajar banyak dari kode itu, ada juga yang kami jadikan acuan utama. []

Namun, modifikasi dari kami juga sangat banyak. Terdapat *splitting* dataset menjadi *train-validation-test* karena pada kode yang asli tidak ada untuk *test*. Selain itu, setelah beberapa kali mencoba *training*, kami mendapati bahwa ternyata kode yang dia buat untuk mengolah koordinat *bounding box* memiliki kesalahan fatal. Kami pun perlu memberikan beberapa penyesuaian baru dan mengulangi *training*.

Bagian yang cukup menyusahkan adalah ketika menambahkan YOLOv4. Itu karena proyek yang menggunakan YOLOv4 jauh lebih sedikit dibanding YOLOv5 sehingga tidak ada banyak referensi. Kami pun belajar cara menggunakannya secara langsung dari *repository* di Github. Di sana, ada berbagai tipe YOLOv4 sehingga kami perlu mencobanya untuk bisa memilih dengan tepat. Kami pun mendapatkan tipe yang cukup sesuai untuk dibandingkan dengan *repository* YOLOv5, meski secara performa dan keandalan sepertinya masih kurang teruji.

Sebelum menjalankan *training* dan evaluasi secara serius, kami beberapa kali menguji kodenya. Ternyata, kode pada YOLOv4 yang ada di *repository* mengalami masalah sehingga ada yang memerlukan *debugging*.

Selain menguji kodenya, kami juga menguji parameter yang tepat. Didapatkan bahwa model yang besar dan citra yang besar tidak terlalu meningkatkan hasil, tetapi komputasinya meningkat pesat. Karenanya, kami tetap menggunakan yang kecil. Perlu diketahui bahwa pengujian-pengujian kecil ini menggunakan dataset yang jauh lebih kecil agar prosesnya lebih ringan dan singkat.

Ketika *training* secara serius, rintangan utamanya adalah keterbatasan *resource* komputasi. Penggunaan GPU pada Google Colaboratory memiliki batas tertentu yang mana aksesnya akan dihentikan sementara jika sudah melewati batas. Penghentian akses ini sudah terjadi berkali-kali pada kami. Apalagi, sebelum proyek ini, kami juga sering menggunakannya untuk berbagai proyek *deep learning* lain sehingga pembatasan pada kami menjadi lebih ketat. Bagaimana pun, tidak ada trik yang bisa dilakukan untuk mendapatkan *resource* komputasi yang lebih baik.

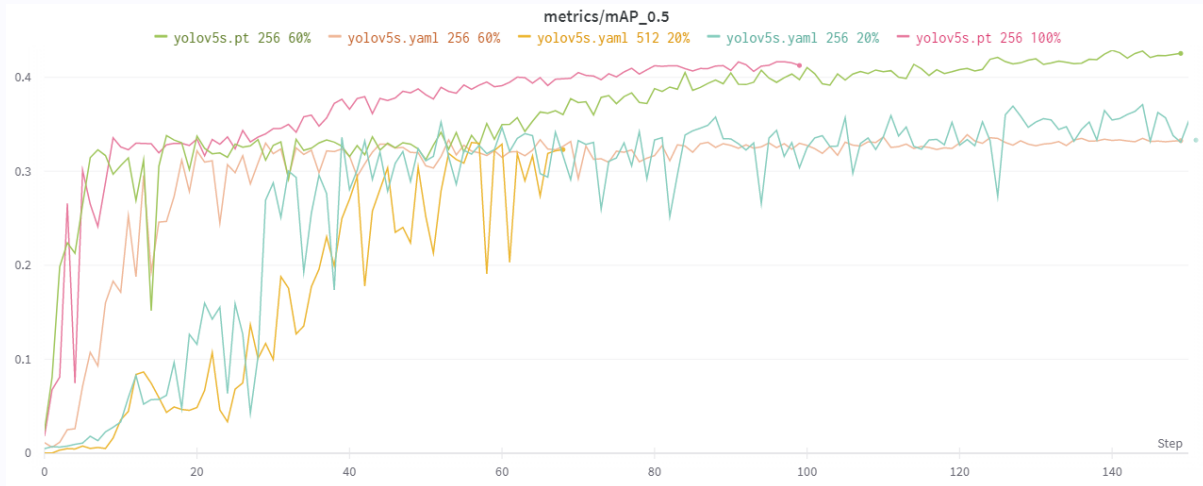
Masalah itu memberikan beberapa konsekuensi penting. Pertama, proses *training* perlu dibuat ringan dan singkat sehingga perlu memilih model yang ringan, citra yang kecil, dan *epoch* yang sedikit. Dengan itu pun, satu kali *training* tetap membutuhkan waktu sekitar 100 menit. Kedua, pembatasan itu membuat kami tidak bisa mencoba banyak model secara maksimal. Ketika proyek ini sudah mencapai *deadline*, akan ada beberapa rencana pengujian yang belum terlaksana. Ketiga, dua hal itu membuat model akhir yang dihasilkan memiliki performa yang jauh dari optimal.

IV. HASIL DAN ANALISIS

A. Pengujian Awal

Bagian ini meliputi berbagai pengujian yang belum terstandar. Bagaimana pun juga, pengujian di sini penting untuk mengambil beberapa keputusan dan melakukan evaluasi. Pengujian awal banyak dilakukan menggunakan YOLOv5 karena memang lebih ringan dan hasilnya bisa tercatat secara otomatis di WandB. Selain yang disampaikan di sini, tentu masih banyak lagi pengujian lainnya, baik untuk YOLOv4 maupun YOLOv5.

Training history dari pengujian awal bagian 1 dapat dilihat pada Gambar 7. Di sini, dilakukan perbandingan terkait tipe model (dengan atau tanpa *pre-trained weight*), ukuran citra (256 dan 512), dan ukuran dataset (dalam persen). Model dengan nama .pt berarti berasal dari *pre-trained weight* sementara yang .yaml berarti berasal dari *config* tanpa *pre-trained weight*.



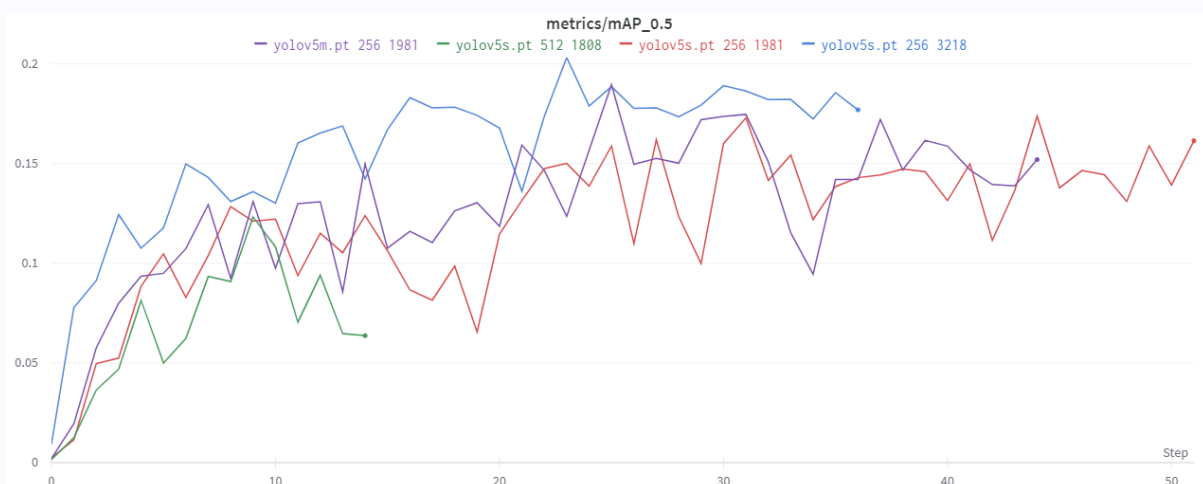
Gambar 7: *Training history* dari pengujian awal bagian 1

Terlihat bahwa model dengan *pre-trained weight* mampu belajar dengan jauh lebih cepat. Di sini, hasil akhir dari model dengan *pre-trained weight* juga relatif lebih baik. Namun, dia memiliki kelemahan yang ternyata baru terlihat di pengujian final.

Selanjutnya, terlihat bahwa ukuran citra yang besar tidak meningkatkan hasilnya. Padahal, peningkatan ukuran citra memberikan dampak yang sangat nyata pada berat dan lamanya komputasi. Kemudian, dari Gambar 7, kita bisa melihat bahwa ukuran dataset memberikan pengaruh. Makin besar ukurannya, secara umum hasilnya lebih baik.

Namun, perlu diketahui bahwa pengujian awal bagian 1 ini kurang valid. Itu karena ternyata terdapat kesalahan dalam pembuatan file-file labelnya. Meski begitu, informasi dan kesimpulan yang kita dapatkan bisa membantu pengambilan keputusan untuk pengujian final.

Terdapat pula pengujian awal bagian 2 yang *training history* nya dapat dilihat pada Gambar 8. Pada pengujian ini, yang divariasi adalah ukuran citra, ukuran dataset, dan ukuran model (m-medium dan s-small). Namun, berbeda dengan pengujian sebelumnya, file label pada pengujian kali ini sudah benar sehingga hasilnya cukup valid. Namun, ini hanya pengujian awal sehingga Epoch nya juga hanya 50 dengan data yang juga kecil.



Gambar 8: *Training history* dari pengujian awal bagian 2

Dari Gambar 8, terlihat bahwa sekali lagi, ukuran dataset yang makin besar berdampak pada hasil yang makin baik. Terbukti pula bahwa peningkatan ukuran citra tidak memberi dampak signifikan bagi hasilnya. Bahkan, di sini hasilnya justru terlihat lebih buruk.

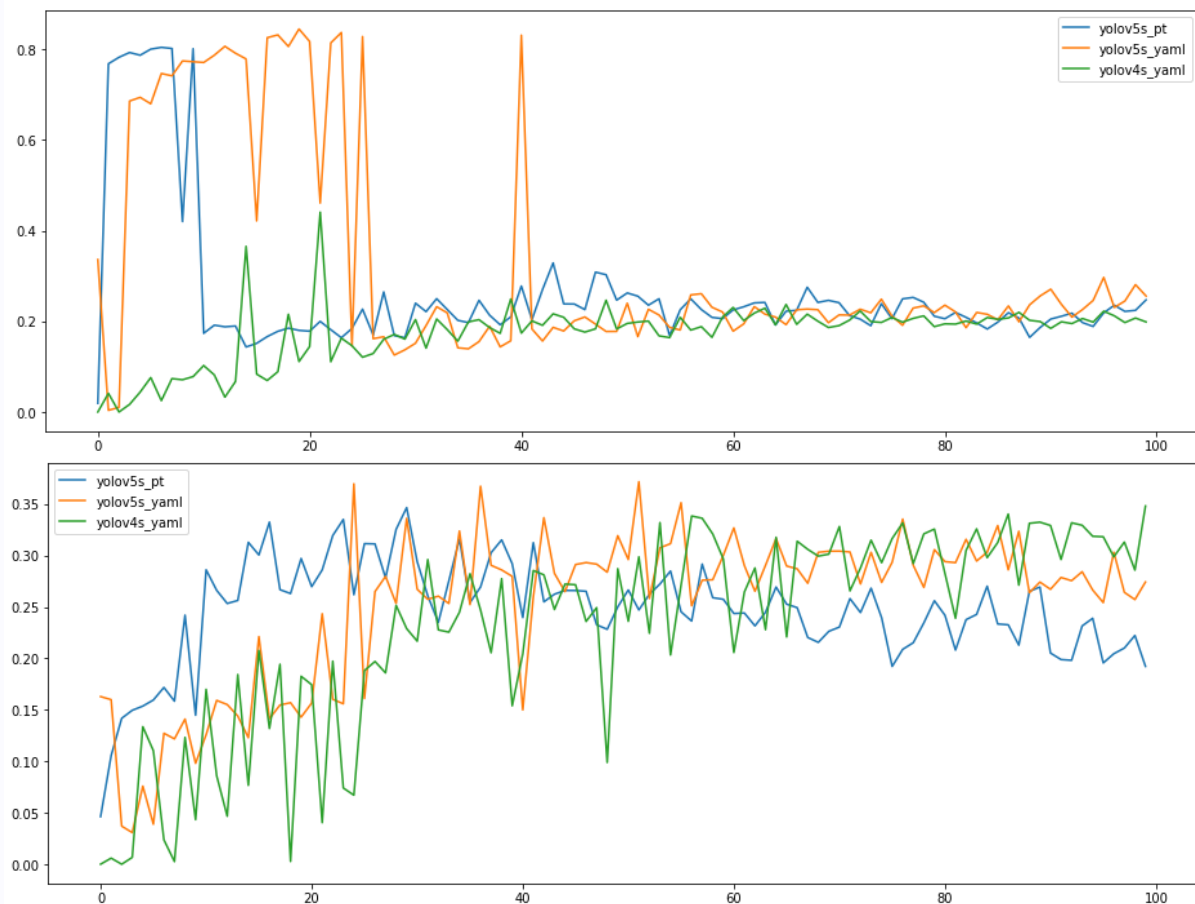
Pengujian ini juga menunjukkan bahwa ukuran model yang lebih besar juga kurang berdampak pada hasilnya. Di sisi lain, seperti citra yang besar, model yang besar membutuhkan komputasi yang berat dan lama.

Dengan demikian, dari pengujian awal ini, kami mengambil beberapa keputusan untuk digunakan di pengujian final: ukuran citra kecil yaitu 256, ukuran model kecil yaitu ukuran s, dan ukuran dataset besar yaitu semuanya.

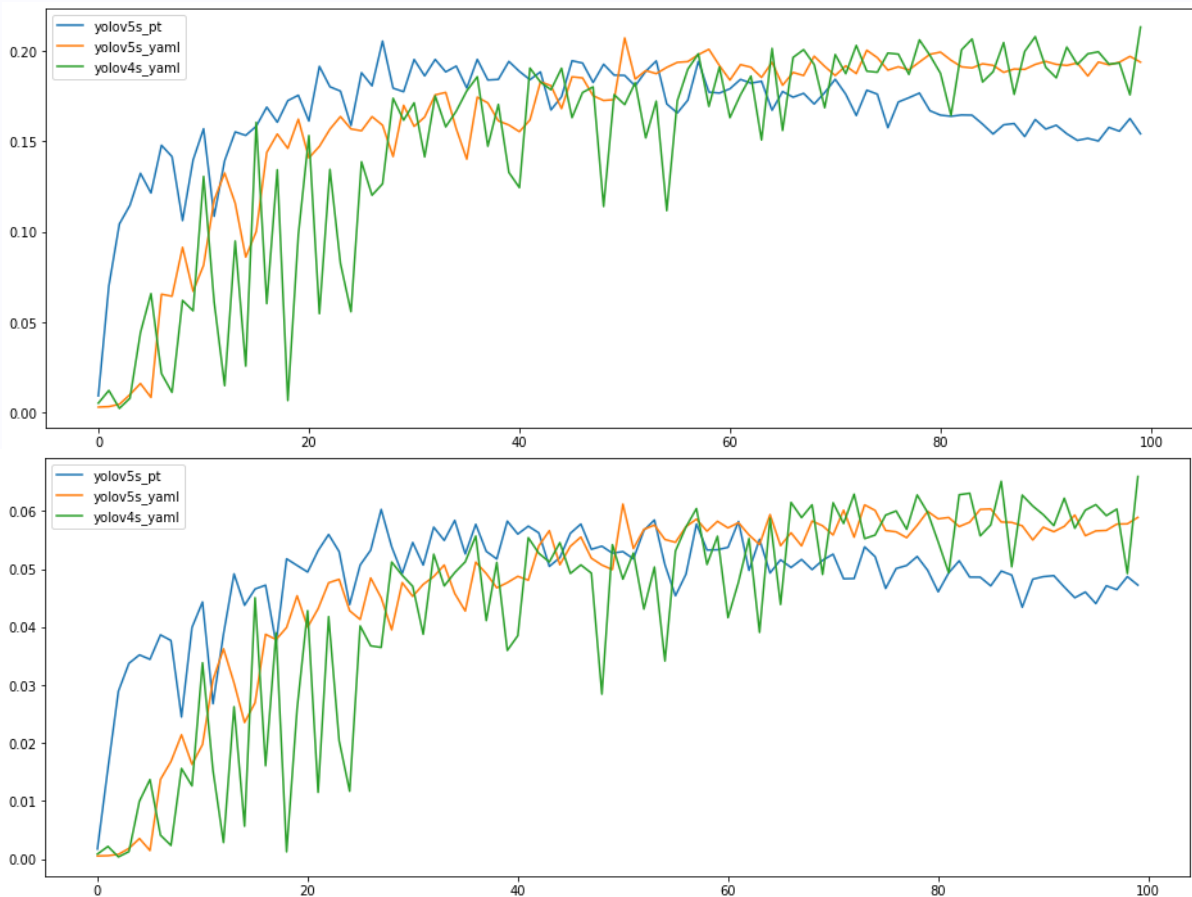
B. Pengujian Final

1. Perbandingan *Training History*

Konsep pengujian final adalah membandingkan beberapa model dengan parameter yang sama sehingga dapat menganalisisnya secara adil. Sebenarnya, ada banyak yang bisa dibandingkan. Bahkan, kami ingin mencoba mengedit struktur YOLO yang ada dalam file .yaml untuk kemudian dibandingkan dengan YOLO yang asli. Namun, karena keterbatasan waktu dan pembatasan *resource* komputasi, hanya tiga model yang teruji.



Gambar 9: *History* Nilai *Precision* (atas) dan *Recall* (bawah) dari Tiga Model



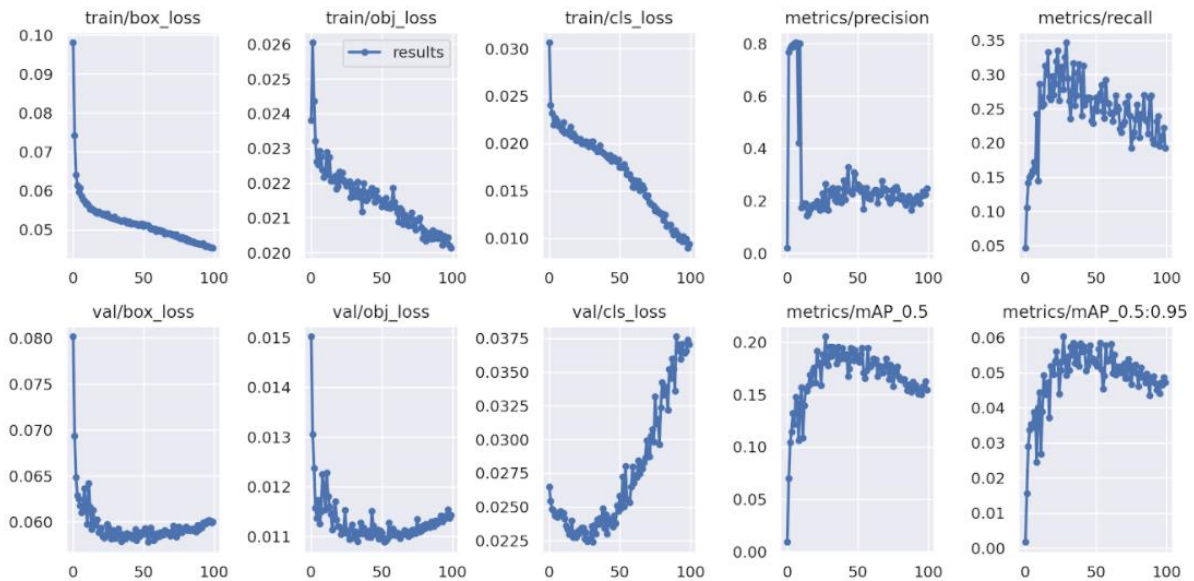
Gambar 10: *History* Nilai mAP@.5 (atas) dan mAP@.5:.95 (bawah) dari Tiga Model

Gambar 9 menunjukkan nilai *precision* dan *recall* dari tiga model: YOLOv5 dengan *pre-trained weight*, YOLOv5 tanpa *pre-trained weight*, dan YOLOv4 tanpa *pre-trained weight*. Sementara itu, Gambar 10 menunjukkan nilai mAP@.5 dan mAP@.5:.95 dari tiga model juga. Untuk semua parameter itu, makin besar tentu makin baik.

Pada bagian awal, kita melihat bahwa YOLOv5 bisa naik lebih cepat daripada YOLOv4, bahkan YOLOv5 dengan *pre-trained weight* naik dengan lebih cepat lagi. Ini berarti, YOLOv5 mampu belajar dengan lebih cepat dibanding YOLOv4. Namun, pada akhirnya, YOLOv4 mampu mengimbangi bahkan sedikit mengalahkan YOLOv5. Ini tentu menarik untuk dianalisis lebih dalam dan diuji lebih lanjut.

Di sisi lain, YOLOv4 cenderung kurang stabil jika dibandingkan YOLOv5. Terlihat bahwa fluktuasinya di nilai *recall* dan mAP lebih signifikan. Yang menarik adalah adanya fluktuasi besar pada YOLOv5 di nilai *precision*. Awalnya bisa sangat tinggi, namun kemudian turun secara drastis sehingga pada akhirnya sama dengan YOLOv4.

Hal paling menarik untuk diperhatikan adalah kurva pada YOLOv5 dengan *pre-trained weight*. Di awal, terlihat dia memberikan hasil dan pertumbuhan yang bagus. Namun di tengah, dia mengalami titik balik yang membuat performanya justru turun. Kami menganggap bahwa *pre-trained weight* yang dia miliki bisa membantu mengenali fitur-fitur di permukaan, namun gagal membantu pada fitur-fitur yang lebih dalam. Ini mengingat *pre-trained weight* dibuat menggunakan citra-citra sehari-hari yang sangat berbeda dengan citra X-Ray terkait COVID-19 ini. Bagaimana pun juga, ini merupakan temuan menarik yang membutuhkan penelitian lebih lanjut.



Gambar 11: *History* dari Pengujian YOLOv5 dengan *pre-trained weight*

Kita bisa melihat Gambar 11 untuk lebih memahami masalah sebelumnya. Seiring dengan meningkatnya Epoch, terlihat bahwa *loss* pada data *train* menurun. Itu karena memang begitulah cara *training* bekerja. Namun, terlihat bahwa *loss* pada data *val* justru meningkat. Bahkan, *cls_loss* meningkat dengan sangat pesat melebihi nilai awal. Di samping hipotesis kami sebelumnya, dari Gambar 11, kami menganggap telah terjadi *overfitting*. Modelnya terlihat bagus pada data *train*, namun mengalami penurunan performa pada data *val*.

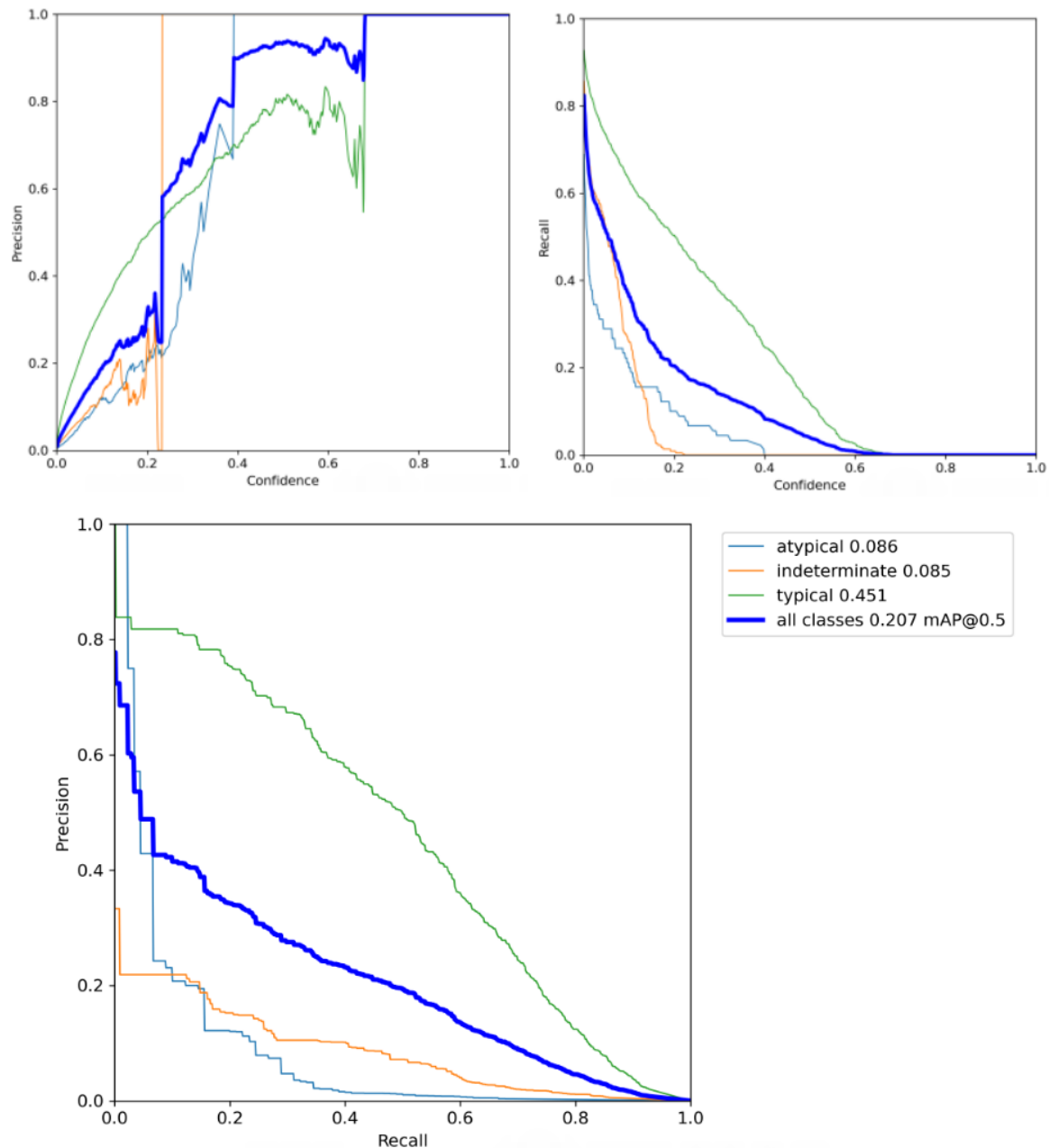
Kemudian, jika kita melihat hasil akhir dari ketiga model, terlihat bahwa nilainya masih sangat rendah. Ketiga model pun memberikan hasil akhir yang cukup mirip. Maka, untuk melihat lebih dalam, pada bagian selanjutnya, kami akan fokus mendalami salah satu model saja tanpa banyak membandingkannya. Model yang dimaksud yaitu YOLOv5 tanpa *pre-trained weight*.

2. Kurva P, R, dan PR

Gambar 12 menunjukkan kurva P, R, dan PR dari model YOLOv5 tanpa *pre-trained weight*. Secara umum, model YOLOv5 dengan *pre-trained weight* juga memiliki kurva yang sama. Sementara itu, pengujian YOLOv4 tidak bisa menghasilkan kurva ini.

Secara teori, *recall* memang akan menurun ketika *confidence threshold* meningkat. Sementara itu, *precision* idealnya selalu tinggi terlepas berapa nilai *confidence threshold* nya. Namun, di sini kita melihat bahwa *precision* nya masih sangat rendah ketika *confidence threshold* nya rendah. Kita juga melihat *precision* nya turun ketika *recall* naik. Yang terakhir ini membuat AUC (*area under the curve*) tidak terlalu besar. Idealnya, kurva PR berupa kurva mendatar di nilai yang besar sehingga AUC nya juga sangat besar dan mendekati 1.

Dari kurva PR pada Gambar 12, terlihat bahwa performa untuk kelas *typical* jauh lebih baik daripada dua kelas yang lain. Ini mungkin berkaitan dengan *class-imbalance* pada datanya. Bisa jadi juga bahwa secara medis kelas *typical* memang lebih mudah dideteksi dibanding dua lainnya.

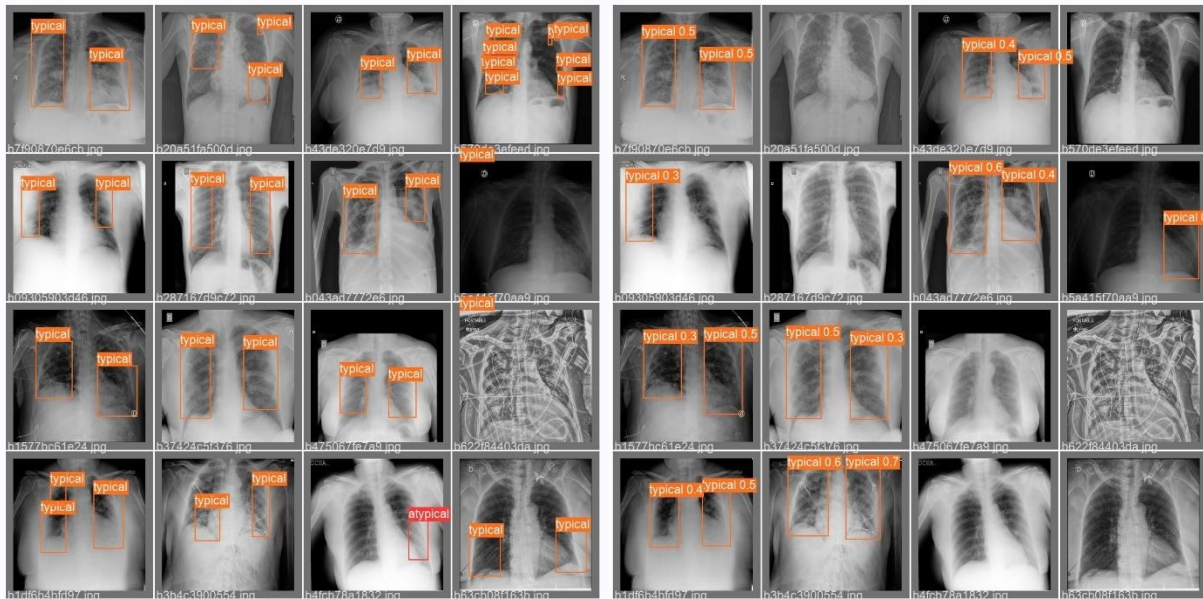


Gambar 12: Kurva P (kiri atas), R (kanan atas), PR (bawah) dari Pengujian YOLOv5 tanpa *pre-trained weight*

3. Perbandingan Hasil Prediksi dan *Ground Truth*

Gambar 13 menunjukkan hasil prediksi dan *ground truth* dari pengujian YOLOv5 tanpa *pre-trained weight* dengan menggunakan data validasi. Terlihat bahwa prediksinya memang masih jauh dari yang seharusnya. Pada beberapa citra, dia memang mampu memprediksi dengan cukup baik. Namun, secara umum, itu masih jauh dari yang seharusnya. Ini membuktikan bahwa performa modelnya memang belum baik.

Gambar perbandingan seperti ini bisa dimanfaatkan oleh ahli (ahli medis, pengolahan citra, atau pun *deep learning*) untuk lebih memahami bagaimana mekanisme pengambilan keputusan dari model. Dengan begitu, harapannya penelitian selanjutnya dapat lebih baik.



Gambar 13: Perbandingan antara *ground truth* (kiri) dan prediksi (kanan)

V. KESIMPULAN

Pada proyek ini, model YOLOv4 dan YOLOv5 digunakan untuk melakukan deteksi objek pada citra X-Ray dada terkait COVID-19. Hasil dari pengujian ini secara umum masih kurang baik dengan mAP yang masih berada di angka 0.2.

Berdasarkan pengujian pada dataset yang ada, ukuran model dan ukuran citra tidak memberikan pengaruh signifikan pada hasil, namun meningkatkan beban dan waktu komputasi. Dari pengujian ini, kami mendapati bahwa penggunaan *pre-trained weight* mampu membuat meningkatkan kecepatan, namun rawan terjadi *over-fitting*.

Dari segi penggunaan, model YOLOv5 lebih unggul karena memiliki fitur menarik seperti koneksi dengan WandB. YOLOv5 juga lebih unggul saat proses *training* karena ringan dan mampu belajar dengan lebih cepat. Dari segi hasil akhir, kami mendapati bahwa YOLOv4 mampu mengimbangi YOLOv5, bahkan terkadang bisa melampauinya.

VI. DAFTAR PUSTAKA

- [1] A. K. Shetty, I. Saha, R. M. Sanghvi, S. A. Save, and Y. J. Patel, "A Review: Object Detection Models," Apr. 2021. doi: 10.1109/I2CT51068.2021.9417895.
- [2] M. A. Al-Antari, C.-H. Hua, J. Bang, and S. Lee, "'Fast deep learning computer-aided diagnosis of COVID-19 based on digital chest x-ray images,'" 2076, doi: 10.1007/s10489-020-02076-6/Published.
- [3] Society for Imaging Informatics in Medicine (SIIM), "SIIM-FISABIO-RSNA COVID-19 Detection," Aug. 2021. <https://www.kaggle.com/c/siim-covid19-detection/overview>
- [4] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," Apr. 2020, [Online]. Available: <http://arxiv.org/abs/2004.10934>
- [5] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva, "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit," *Electronics*, vol. 10, no. 3, 2021, doi: 10.3390/electronics10030279.
- [6] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-YOLOv4: Scaling Cross Stage Partial Network," Nov. 2020, [Online]. Available: <http://arxiv.org/abs/2011.08036>
- [7] D. Thuan, "EVOLUTION OF YOLO ALGORITHM AND YOLOV5: THE STATE-OF-THE-ART OBJECT DETECTION ALGORITHM EVOLUTION OF YOLO ALGORITHM AND YOLOV5: THE STATE-OF-THE-ART OBJECT DETECTION ALGORITHM," 2021.
- [8] M. N. Datta, Y. Rathi, and M. Eliazar, "Wheat Heads Detection using Deep Learning Algorithms," 2021. [Online]. Available: <http://annalsofrscb.ro>
- [9] xhlulu, "SIIM COVID-19: Resized to 256px JPG," May 2021. <https://www.kaggle.com/xhlulu/siim-covid19-resized-to-256px-jpg>
- [10] Ammar Alhaj Ali, "SIIM-COVID-19 Detection Training Labels," Jun. 2021. <https://www.kaggle.com/ammarnassanalhajali/siimcovid19-detection-training-label>
- [11] Glenn Jocher and Ultralytics, "yolov5." Github Repository. [Online]. Available: <https://github.com/ultralytics/yolov5>
- [12] G. Jocher *et al.*, "ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support," Oct. 2021, doi: 10.5281/ZENODO.5563715.
- [13] Wong Kin Yiu, "PyTorch_YOLOv4/u5." Github Repository. [Online]. Available: https://github.com/WongKinYiu/PyTorch_YOLOv4/tree/u5