BAB III
METODOLOGI PENELITIAN

3.1. Subjek dan Objek Penelitian

Pada penelitian ini subjek yang akan diteliti adalah citra gigi yang terdampak karies gigi yang didapatkan dari *scraping* di *google image* dengan kata kunci "*Dental Caries*". Objek penelitian yang diambil ialah hasil deteksi citra yang terdampak karies gigi.

3.2. Alat dan Bahan Penelitian

Dalam penelitian pengembangan *object detector* untuk mendeteksi karies gigi dibutuhkan alat dan bahan. Di antaranya adalah perangkat lunak (*software*), perangkat keras (*hardware*), dan dataset.

3.2.1. Perangkat Lunak

Perangkat lunak (*software*) yang akan digunakan selama proses penelitian ini sebagai berikut:

Tabel 3. 1 Spesifikasi Software.

Software	Kegunaan
Sistem Operasi Windows 10 Home Single Language 64-bit	Sistem operasi
Bahasa Pemrograman Python 3.7	Bahasa pemrograman
Anaconda	*Manager environment dan library*
Github	*Version Control*
Tensorflow 2.0	*Library machine learning*
Keras	*Library machine learning*
Matplotlib	*Library ploting*
cv2	*Library pre-proceessing*
Object Detection API	*Library object detection*
Jupyter Notebook	*Text Editor*

3.2.2. Perangkat Keras

Perangkat keras (*hardware*) yang digunakan untuk menyelesaikan penelitian ini sebagai berikut:

Tabel 3. 2 Spesifikasi Hardware.

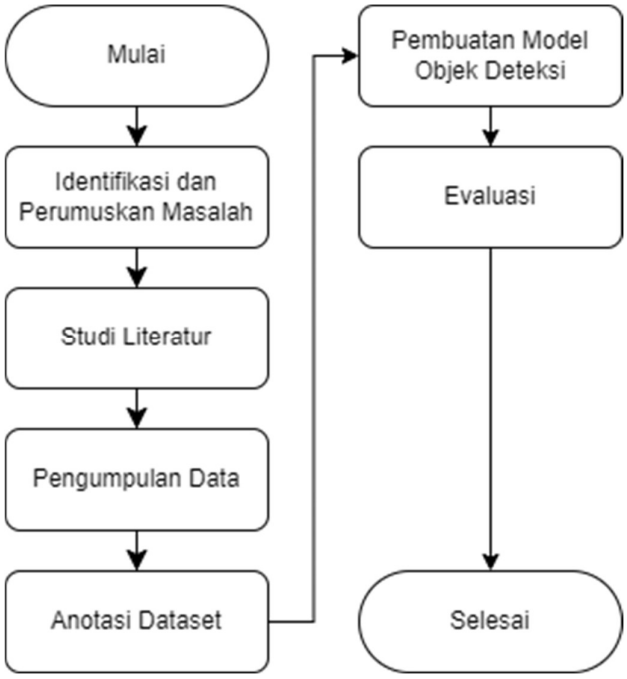
Hardware	Spesifikasi
Laptop	*Legion 5 Pro 16ACH6H*
RAM	16 GB
Memori	SSD 1 TB
Kartu Grafik	*NVIDIA GeForce RTX 3060 Leptop GPU*
Processor	*AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz*

3.2.3. Bahan

Bahan pada penelitian ini adalah data berupa citra mulut yang terdapat gigi yang terdampak karies gigi. Di mana setiap citra akan dianotasi/labeli lokasi kariesnya dan kemudian akan digunakan untuk melatih model *pre-trained*.

3.3. Diagram Alir Penelitian

Alir pada penelitian ini di gambarkan pada Gambar 3.1. Tahapan penelitian yang dilakukan pada penelitian ini dimulai dengan identifikasi masalah yang di ikuti oleh perumusan masalah. Kemudian data-data yang diperlukan di kumpulkan berupa citra gigi yang terdampak karies gigi. Citra-citra tersebut kemudian di anotasikan/labeli agar dapat di gunakan untuk melatih model agar dapat melakukan deteksi karies gigi. Objek detektor dirancang setelah dataset siap. Model objek deteksi dibuat menggunakan *object detection API* dengan menggunakan model *pre-trained Faster R-CNN ResNet50 dan ResNet101*. Setelah model deteksi dibuat, akan dilakukan pengujian terhadap model untuk melihat apakah model mempunyai performa yang baik. Atas dasar temuan penelitian, rekomendasi kemudian dibuat.



Gambar 3. 1 Diagram Alir Penelitian

3.3.1. Identifikasi dan Perumusan Masalah

Langkah pertama dalam melakukan suatu penelitian adalah mengidentifikasi suatu masalah yang akan dijadikan topik penelitian. Dalam penelitian ini, peneliti membahas tingginya prevalensi penyakit gigi di Indonesia terutama untuk penyakit karies gigi. Kedua, masalah dirumuskan dalam bentuk masalah utama yang dapat dipecahkan dan alasan yang mendasari penelitian ini perlu dilakukan. Di mana masalah yang ditemukan ialah butuhnya sistem deteksi yang dapat digunakan untuk membantu praktisi gigi dalam mendiagnosis karies gigi. Hal ini dilakukan dengan melakukan pembuatan model objek deteksi yang dapat mendeteksi karies gigi menggunakan *transfer learning model pre-trained ResNet50* dan *ResNet101*.

3.3.2. Studi Literatur

Proses studi literatur merupakan salah satu tahapan yang perlu dilakukan oleh peneliti pada penelitian. Di mana studi literatur dilakukan dengan tujuan mencari referensi dari penelitian terdahulu guna dijadikan acuan penelitian, agar solusi yang dibuat nanti tidak sama tetapi dapat menjadi pengembangan dari penelitian sebelumnya. Pada studi literatur penelitian ini beberapa bidang ilmu yang

berhubungan dengan penelitian akan di pelajari. Referensi literatur dapat berasal dari skripsi, buku, jurnal, dan internet, dengan teori yang digunakan berhubungan dengan:

1. Penyakit gigi dan mulut
2. *Machine Learning*
3. *Deep Learning*
4. *Object Detection*
5. *CNN*
6. *Faster R-CNN*
7. *ResNet*
8. *Bounding Box Regression*

3.3.3. Pengumpulan Data

Data yang digunakan selama penelitian adalah citra gigi yang terdampak karies gigi. Karies gigi dipilih karena merupakan penyakit gigi dengan prevalensi tertinggi di Indonesia[35]. Data tersebut dikumpulkan dari bulan Oktober 2022 hingga saat ini. Sejauh ini dataset yang dikumpulkan melalui *scraping* di *google image* dengan kata kunci "*Dental Caries*" dan dari situs klinik-klinik gigi.

Gambar 3. 2 Sample dari dataset.

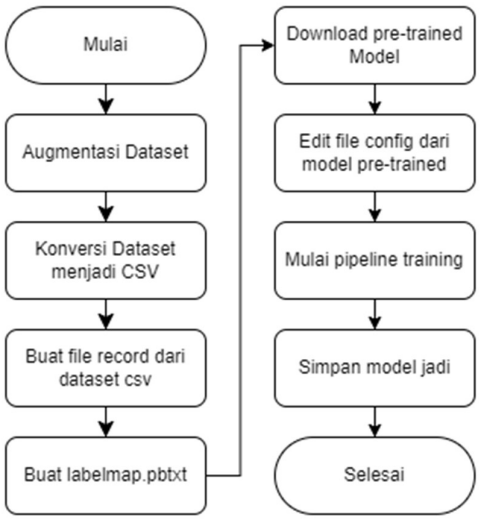
3.3.4. Anotasi Data

Data yang sudah dikumpulkan kemudian dianotasi di mana tiap citra di dataset diberikan label. Dalam tugas pendeteksian objek, pemberian anotasi pada dataset melibatkan penandaan kotak pembatas di sekitar objek yang diminati di dalam citra dan memberikan label untuk setiap objek. Proses ini membantu melatih model untuk mengidentifikasi dan melokalisasi objek tertentu. Tiap citra dapat memiliki lebih dari satu objek di dalamnya. Proses anotasi pada dataset dilakukan

dengan bantuan tenaga ahli kesehatan gigi dengan pengalaman praktik dua tahun yang merupakan lulusan dari Universitas Jenderal Soedirman, dokumentasi proses *labeling* dapat dilihat pada Lampiran 17 dan Lampiran 18. Tenaga ahli membantu dengan menandai lokasi karies pada citra mulut. Kemudian peneliti akan memberikan *bounding box* pada citra tersebut. Pada proses anotasi peneliti menggunakan aplikasi pihak ketiga bernama *labelimg* untuk mempercepat proses anotasi. Keluaran dari *labelimg* adalah file *xml* yang berisi informasi seperti titik x dan y dari karies, tinggi citra, lebar citra, kategori, dan lokasi filenya. Anotasi dalam *labelimg* dapat dilihat pada Gambar 3.3.

Gambar 3. 3 Gambar Karies gigi yang sudah di anotasi.

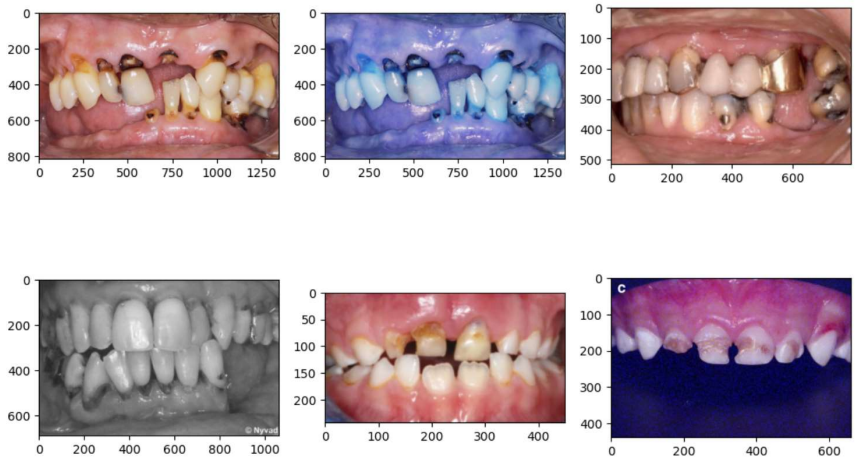
3.4. Pembuatan Model Objek Deteksi



Gambar 3. 4 Alir pembuatan model objek deteksi.

3.4.1 Augmentasi Dataset

Dataset yang di gunakan memiliki beberapa kekurangan di antaranya ialah kurangnya variasi dan jumlah citra. Untuk menangani kekurangan tersebut akan di lakukan augmentasi degan tujuan menambah jumlah citra dan variasi citra. Augmentasi yang di lakukan ialah menerapkan filter-filter pada citra. Filter-filter tersebut ialah: *gaussian-blur, noise, sharpen, BGR,* dan *grayscale.* Skrip augmentasi dapat dilihat pada Lampiran 10. Hasil augmentasi dataset dapat dilihat pada Gambar 3.5.



Gambar 3. 5 Sample dataset sesudah augmentasi.

3.4.2 Konversi Dataset menjadi CSV

Dataset yang sudah di augmentasi kemudian akan di konversi menjadi file CSV. File anotasi berformat *.xml* ini akan di *parsing* dengan tujuan mengekstrak informasi-informasi penting dari dalamnya. Informasi penting tersebut ialah *filename*, *width*, *height*, *class*, *xmin*, *ymin*, *xmax*, dan *ymax*. Proses konversi ini dilakukan dengan memanggil skrip "*xml_to_csv.py*", dapat dilihat pada Lampiran 11. Skrip ini melakukan *parsing* pada file *xml* dan menyimpan informasi-informasi relevan kedalam *list* kemudian *list* tersebut akan di simpan kedalam format CSV.

3.4.3 Buat File *Record* dari Dataset CSV

Untuk memulai *pipeline* pelatihan *object detection API* harus ada file *record*. File *record* adalah dataset yang akan dibaca oleh model selama proses pelatihan. File *record* memiliki format file biner yang berisi data pelatihan terserialisasi atau

dapat dibilang berisikan informasi-informasi penting atau fitur yang akan di pelajari oleh model yang akan dilatih. File *record* menyimpan *metadata* seperti *bounding box* berlabel atau anotasi bersama dengan data citra nya. Beberapa dari *metadata* bounding box di antaranya ialah *width*, *height*, *class*, *xmin*, *ymin*, *xmax*, dan *ymax.*

File *record* didapatkan dari menkonversi file CSV menjadi file *record*. Pada tahap ini juga *xmin*, serta *xmax* akan di bagikan dengan *width*, dan *ymin*, serta *ymax* akan dibagikan dengan *height* agar mendapatkan nilai yang mewakili posisi relatif dari lebar dan tinggi objek atau *bounding box*. Konversi dari CSV ke file record dilakukan dengan memanggil skrip "dataset_to_tfrecord.py", skrip dapat dilihat pada Lampiran 12. Skrip ini melakukan pengelompokan berdasarkan citra, kemudian akan membuat pemetaan fitur untuk tiap citra di dataset. Selain itu pada tahap ini dataset akan dibagi menjadi dataset untuk training dan testing dengan rasion 9:1.

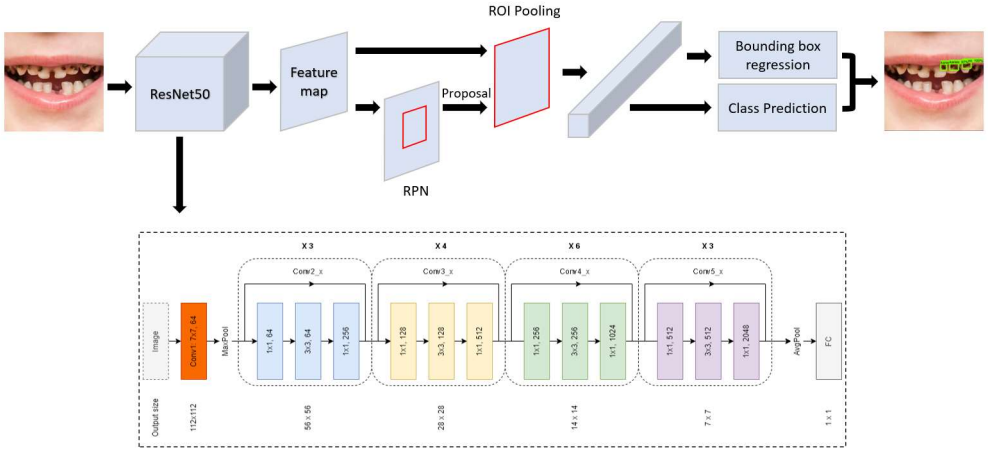
3.4.4 Membuat *Labelmap*

Label map adalah sebuah file yang berguna untuk mendefinisikan pemetaan antara kelas id dan nama kelas. Karena pada penelitian ini hanya terdapat satu kelas saja yaitu karies maka hanya ada satu id di dalam *labelmap*. Label map memiliki format pbtxt. Untuk syntax dan konfigurasi pada *labelmap* dapat dilihat di Lampiran 1.

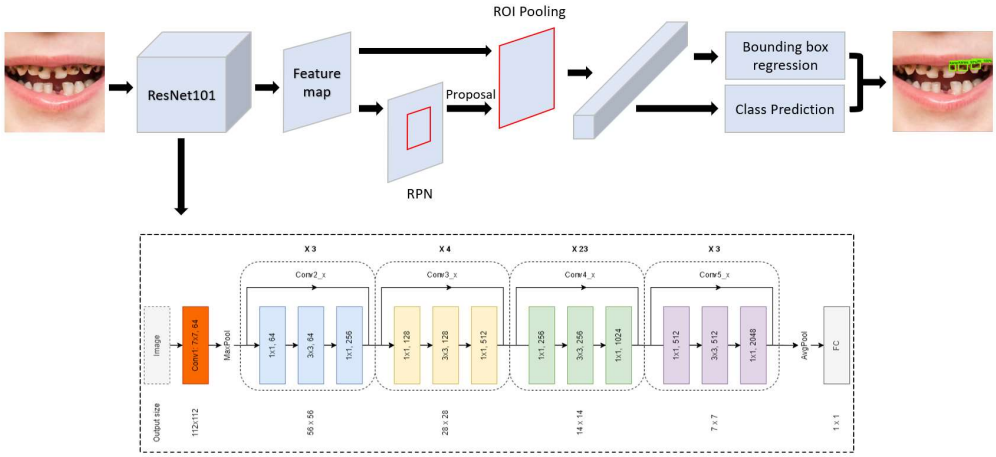
3.4.5 *Unduh pre-trained* Model

Dalam penelitian ini model *pre-trained Faster R-CNN ResNet50 V1 640x640* dan *Faster R-CNN ResNet101 V1 640x640* akan digunakan, model tersebut didapatkan dari kolekesi model *pre-trained* yang disediakan oleh *tensorflow object detection API*[81]. Sedangkan model *pre-trained* sendiri adalah model *machine learning* yang telah dilatih pada dataset yang besar untuk tugas tertentu, seperti klasifikasi citra, deteksi objek, pemrosesan bahasa alami, atau pengenalan suara. Dengan tujuan untuk mempelajari pola dan fitur umum dari data pelatihan. Model *ResNet50* dan *ResNet101* yang disediakan *object detection api* telah di latih menggunakan dataset *COCO*. Untuk menggunakan model *pre-trained*, model

terlebih dahulu harus diunduh dari *TensorFlow 2 Detection Model Zoo*. Arsitektur ResNet50 dan ResNet101 dapat dilihat pada Gambar 3.6 dan Gambar 3.7.



Gambar 3. 6 Arsitektur Faster R-CNN dengan ResNet-50.



Gambar 3. 7 Arsitektur Faster R-CNN dengan ResNet-101.

3.4.6 Edit File *Config* Model *Pre-trained*

Sebelum dapat memulai pipeline pelatihan menggunakan *object detection API*, beberapa hal terlebih dahulu harus di siapkan, di antaranya file *record training* dan *testing, file labelmap*, model yang akan di latih, dan file config. File *config* adalah file yang berisi pengaturan dan parameter yang mendefinisikan perilaku dan karakteristik model objek deteksi, beberapa parameter penting di antaranya model yang digunakan, *feature extractor*, optimizer, jumlah kelas, dan banyak lainnya.

File config ini berfungsi sebagai cetak biru atau *blueprint* untuk melatih dan menerapkan model. File *config* sendiri terdiri dari 5 bagian[82]:

1. Model: bagian ini digunakan untuk mendefinisikan jenis model yang nanti akan dilatih (misalnya, *meta-arsitektur*, ekstraktor fitur).
2. *train_config*: bagian ini memutuskan parameter apa saja yang harus digunakan untuk melatih parameter model.
3. *eval_config*: bagian ini menentukan rangkaian metrik apa yang akan digunakan dan dilaporkan untuk evaluasi
4. *train_input_config*: bagian ini mendefinisikan dataset apa yang akan dilatih oleh model
5. dan *eval_input_config*: bagian ini mendefinisikan dataset apa yang akan dievaluasi oleh model.

Jika semua persyaratan sudah lengkap maka proses edit pada file config atau *fine-tuning* dapat dimulai. File *config* dapat dilihat pada Lampiran 2-5. Di dalam file *config* ada beberapa parameter yang harus diubah sebelum proses pelatihan dapat dimulai, parameter-parameter tersebut di antaranya adalah: *fine_tune_checkpoint*, *fine_tune_checkpoint_type*, *batch_size, dan num_steps* pada *train_config*. *Label_map_path* dan *input_path* pada *train_input_reader*. *Label_map_path* dan *input_path* pada *eval_input_reader*. Selain parameter-parameter tersebut jumlah kelas juga harus diubah sesuai dengan jumlah kelas pada dataset, jumlah kelas dapat diubah pada bagian *model* di dalam bagian faster_*rcnn* pada parameter *num_classes.* Proses *resize* juga di atur pada file config pada bagian *model* didalam bagian faster_*rcnn* pada parameter *image_resizer*.

Pada penelitian ini model *ResNet50* dan *ResNet101* akan dilatih menggunakan *optimizer momentum* dan *adam*, maka pada file config parameter *optimizer* di dalam bagian *train_config* juga harus disesuaikan dengan *optimizer* yang ingin digunakan. Akan ada total empat model yang akan dilatih, keempat model tersebut ialah model *ResNet50* dengan *optimizer momentum* dan *adam*, dan *ResNet101* dengan *optimizer momentum* dan *adam*. Empat file *config* tersebut dapat dilihat pada Tabel 3.3:

Tabel 3. 3 Skema Model.

Model	Nama file *config*	Model	Variasi
M1	*faster_rcnn_resnet50_v1_640x640_coco17_tpu-8_momentum.config*	*ResNet50*	*Optimizer momentum*
M2	*faster_rcnn_resnet50_v1_640x640_coco17_tpu-8_adam.config*	*ResNet50*	*Optimizer adam*
M3	*faster_rcnn_resnet101_v1_640x640_coco17_tpu-8_momentum.config*	*ResNet101*	*Optimizer momentum*
M4	*faster_rcnn_resnet101_v1_640x640_coco17_tpu-8_adam.config*	*ResNet101*	*Optimizer adam*

Untuk masing-masing file *config* pada Tabel 3.3 parameter-parameter penting yang dapat dilihat pada Tabel 3.4 akan digantikan. Pertama untuk parameter *num_classes* value *default* akan diganti menjadi jumlah kelas model yang akan dilatih (jumlah kelas adalah satu). Kemudian pada parameter *fine_tune_checkpoint* lokasi dari check point model pre-trained akan dimasukkan. Untuk parameter *fine_tune_checkpoint_type value default* "*classification*" akan diganti menjadi "detection" karena tugas yang akan dilakukan oleh model adalah deteksi objek. Selanjutnya untuk parameter batch_size dan *num_steps* value diubah sesuai dengan kekuatan sistem tempat melatih.

Parameter *optimizer* disesuaikan dengan file *config*-nya di mana untuk file *config* yang menggunakan *optimizer momentum* akan menggunakan parameter "*momentum_optimizer*" dengan *learning_rate_base* ".04". Untuk file *config* yang menggunakan *optimizer adam* akan menggunakan parameter "*adam_optimizer*" dengan *learning_rate_base* "1e-3" atau "0.001". Setelah itu untuk parameter *label_map_path* pada bagian *train_input_reader* dan *eval_input_reader value default* akan dimasukkan lokasi file *labelmap*. Terakhir untuk parameter *input_path* pada bagian *train_input_reader* akan dimasukkan lokasi file *record training*, dan untuk *input_path* pada bagian *eval_input_reader* akan dimasukkan lokasi file *record testing*.

Tabel 3. 4 Parameter yang dibuah pada config file.

Config Part	Parameter	Perubahan
model -> faster_rcnn	num_classes	Jumlah kelas
train_config	*fine_tune_checkpoint*	Lokasi dari *checkpoint* 0
	fine_tune_checkpoint_type	Tipe tugas (Detection)
	batch_size	*Batch size training*
	num_steps	*Jumlah steps pelatihan*
	optimizer	*Momentum / Adam dan base learning rate*
train_input_reader	*label_map_path*	Lokasi *labelmap*
	input_path	Lokasi file *record training*
eval_input_reader	*label_map_path*	Lokasi *labelmap*
	input_path	Lokasi file *record testing*

3.4.7 *Training Pipeline* / Pelatihan Model

Setelah file *config* sudah dikonfigurasi pelatihan dapat dimulai. Proses *training* mulai dengan memanggil skrip "*model_main_tf2.py*", skrip dapat dilihat pada Lampiran 15. Skrip ini merupakan skrip *training* dari *object detection API.* Proses *training* dapat dimulai dengan menjalankan *command* berikut:

```
python model_main_tf2.py --pipeline_config_path=pipeline.config --
model_dir=training -alsologtostderr
```

Ada beberapa *flag/parameter* yang di terima oleh skrip "*model_main_tf2.py*" yang di mana *flag pipeline_config_path* menunjuk ke file *config*, *flag model_dir* menunjuk ke direktori di mana checkpoint dan event pelatihan akan disimpan.

Selama proses pelatihan ada beberapa jenis *loss* dari model yang ditampilkan kemajuan pembelajaran dari model. *Loss* tersebut adalah *region proposal network localization loss, region proposal network objectness loss*, *box classifier classification loss, box classifier localization loss,* dan *total loss*. R*egion proposal network localization loss* menunjukkan seberapa akurat proposal lokalisasi *bounding box* dari RPN. R*egion proposal network objectness loss* menunjukkan seberapa baik RPN menentukan proposal lokalisasi sebagai memiliki objek atau dengan hanya latar belakang saja(tidak mengandung objek). *Box classifier classification loss* menunjukkan seberapa baik model dalam melakukan klasifikasi

dari objek dalam region proposal. *Box classifier localization loss* menunjukkan seberapa akurat prediksi lokasi *bounding box* dari objek yang dideteksi di dalam proposal region dari RPN[83]. *Terakhir total loss* adalah hasil jumlah dari seluruh *loss*.

3.4.8 Simpan Model

Setelah proses pelatihan model selesai, model akan disimpan. Penyimpanan model dilakukan dengan memanggil skrip "*exporter_main_v2.py*", skrip dapat dilihat pada Lampiran 16. Skrip ini berguan untuk penyimpanan model dari *object detection API.* Penyimpanan model dapat dimulai dengan menjalan kan *command* berikut:

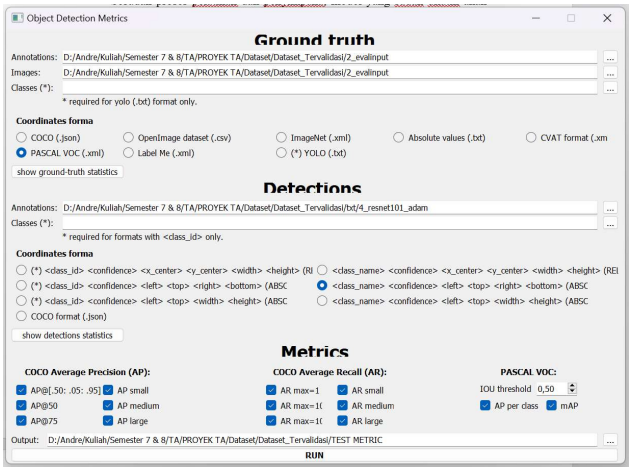
```
python exporter_main_v2.py --trained_checkpoint_dir=training --
pipeline_config_path=pipeline.config --output_directory
inference_graph
```

Ada beberapa *flag/parameter* yang diterima oleh skrip "*exporter_main_v2.py*" yang di mana *trained_checkpoint_dir* menunjuk pada file *checkpoint* yang akan disimpan, *pipeline_config_path* menunjuk pada lokasi file *config* yang akan disimpan, dan *output_directory* menunjuk pada lokasi folder di mana model akan disimpan. Output dari skrip ini adalah folder output yang berisikan folder *checkpoint*, file config, dan *SavedModel* dari model yang sudah dilatih.

3.5. Evaluasi Model

Sesudah proses pelatihan dan penyimpan, model yang sudah dilatih akan dievaluasi di mana *Average Precision* dan *precision recall curve* dari model akan dicari. Untuk melakukan evaluasi pada penelitian ini akan digunakan *toolkit* yang diciptakan oleh Rafael Padilla et al. [76]. Untuk menggunakan *toolkit* ini, *toolkit* terlebih dahulu harus diunduh dari *github* [84]. Sesudah *toolkit* diunduh kebutuhan/*requirement library* untuk menjalankan *toolkit* tersebut harus ada, untuk menginstall kebutuhan/*requirement library* menjalankan *command* "*python setup.py install*". Jika sudah program dapat dijalankan dengan menggunakan *command* "*python run.py*", ketika dijalankan program akan membuka *windows* baru seperti Gambar 3.8.

Program ini dapat mencari *metric* dari model objek deteksi dengan menyediakan *ground truth* dari dataset evaluasi dan hasil deteksi dari model. Hasil deteksi harus dalam format .txt dan masing-masing citra memiliki satu file txt. Sehingga total file .txt adalah sama dengan jumlah citra yang dievaluasi. File .txt tersebut berisikan deteksi yang terjadi pada citra tersebut, di mana tiap baris dalam file tersebut melambangkan satu deteksi. Format penulisan per baris dalam file .txt adalah kelas, skor_keyakinan, xmin, ymin, xmax, dan ymax.



Gambar 3. 8 Program toolkit evaluasi objek deteksi.

Luaran dari program ini adalah *metric* evaluasi objek deteksi dan *precision recall curve*. *Metric* evaluasi tersebut adalah sebagai berikut :

1. *Average Precision(AP)* pada *IoU* 0.50:0.95,
2. *Average Precision(AP)* pada *IoU* 0.50,
3. *Average Precision(AP)* pada *IoU* 0.75,
4. *AP Across Scale* pada objek *small*,
5. *AP Across Scale* pada objek *medium*,
6. *AP Across Scale* pada objek *large*,
7. *Average Recall(AR)* pada 1 deteksi per gambar,
8. *Average Recall(AR)* pada 10 deteksi per gambar,
9. *Average Recall(AR)* pada 100 deteksi per gambar,
10. *AR Across Scale* pada objek *small*,
11. *AR Across Scale* pada objek *medium*,
12. *AR Across Scale* pada objek *large*.

3.5.1 Ekstraksi Hasil Deteksi

Untuk mengekstrak hasil deteksi dari model digunakan skrip "*detect_from_image.py*", skrip ini dapat dilihat pada Lampiran 13. Skrip ini digunakan untuk menguji model pada dataset evaluasi. Skrip tersebut digunakan dengan *Command* berikut :

```
python .\detect_from_image.py -m
.\inference_graph_resnet101_momentum\saved_model -l
.\labelmap.pbtxt -i .\test_images\karies
```

Command diatas menerima beberapa *flag* atau parameter di antaranya "*m"* menunjuk ke direktori di mana file *inference graph* dari model yang akan di gunakan untuk evaluasi disimpan, *"l"* menunjuk ke direktori file *labelmap.pbtxt* berisi *labelmap* atau kelas yang ada, dan *"i"* menunjuk pada direktori berisikan citra yang akan dievaluasi.

Keluaran dari skrip tersebut adalah file csv berisikan seluruh data/hasil deteksi dari dataset evaluasi dari model yang digunakan. Tiap baris file csv ini terdiri dari nama_file_citra (nama file dari citra), kelas (kelas yang terdeteksi), skor_keyakinan(skor keyakinan dari kelas), xmin, ymin, xmax, dan ymax. Karena ada empat model yang dibuat maka skrip ini dijalankan empat kali untuk masing-masing model.

3.5.2 Konversi Hasil Deteksi Menjadi File txt

Untuk dapat menggunakan *toolkit* data hasil deteksi harus dalam format txt dan masing-masing citra memiliki satu file txt. Untuk mengubah file csv menjadi file txt digunakan skrip "*csv_to_txt.ipynb*", skrip ini dapat dilihat pada Lampiran 14. Skrip ini dijalankan menggunakan *jupyter notebook.* Skrip ini berfungsi untuk mengubah file csv hasil deteksi menjadi file txt untuk masing-masing citra hasil deteksi. Skrip ini menerima file csv kemudian tiap baris dari file csv tersebut akan dikelompokkan berdasarkan nama file menggunakan fungsi "*split*". Kemudian group ini akan diiterasi untuk mengekstrak hasil deteksi masing-masing citra dan akan menulis hasil tersebut pada file txt. Luaran dari skrip ini adalah file txt sebanyak citra yang dievaluasi/dideteksi yang berisikan data deteksi kelas, skor_keyakinan, xmin, ymin, xmax, dan ymax.