2. TINJAUAN PUSTAKA

Untuk mendukung pembuatan tugas akhir ini, dibutuhkan beberapa tinjauan pustaka dan teori penunjang sebagai bahan acuan dan referensi. Dengan demikian pembuatan tugas akhir ini menjadi terarah.

2.1 Pengenalan Seeed Studio reComputer J1010

Seeed Studio reComputer J1010 adalah *edge computer* yang dibuat dengan modul produksi Nvidia Jetson Nano 4GB. Nvidia Jetson Nano adalah komputer dalam ukuran kecil yang mampu menjalankan beberapa *neural networks* secara pararel untuk pengaplikasian seperti *image classification, object detection, segmentation,* dan *speech processing*.

Gambar 2.1 Seeed Studio reComputer J1010 dengan Modul Nvidia Jetson Nano

Modul Nvidia Jetson Nano bekerja dengan *system image* yang telah dipasangkan pada microSD. Spesifikasi yang dimiliki dari Nvidia Jetson Nano adalah antara lain GPU dengan 128 – core Maxwell, CPU dengan Quad-core ATM 157 @ 1.43 GHz, dan memori hingga 4 GB. Untuk fitur *input* dan *output* yang dimiliki adalah port HDMI, USB dengan 1 USB 3.0 dan 2 USB 2.0 Micro-B. Nvidia Jetson Nano juga dilengkapai dengan beberapa protokol komunikasi yaitu dengan GPIO (*General-Purpose Input/Output*), I^2C(*Inter-IC)*, I^2S (*Inter-IC Sound Bus)*, SPI (*Serial Peripheral Interface)*, dan UART (*Universal Asynchronous Receiver-Transmitter).*

Nvidia Jetson Nano mampu untuk menjalankan jaringan yang terkemuka dengan *framework Machine Learning* seperti Tensorflow, Pytorch, Darknet, Caffe, MXNet, dan lain sebagainya. Dengan CUDA (*Compute Unified Device Architecture*), Jetson Nano mampu untuk mempercepat performa *Neural Network* agar sistem dapat menjalankan jaringan yang terkemuka dengan baik (*Jetson Nano Developer KIT | Nvidia Jetson Nano, n.d.)*.

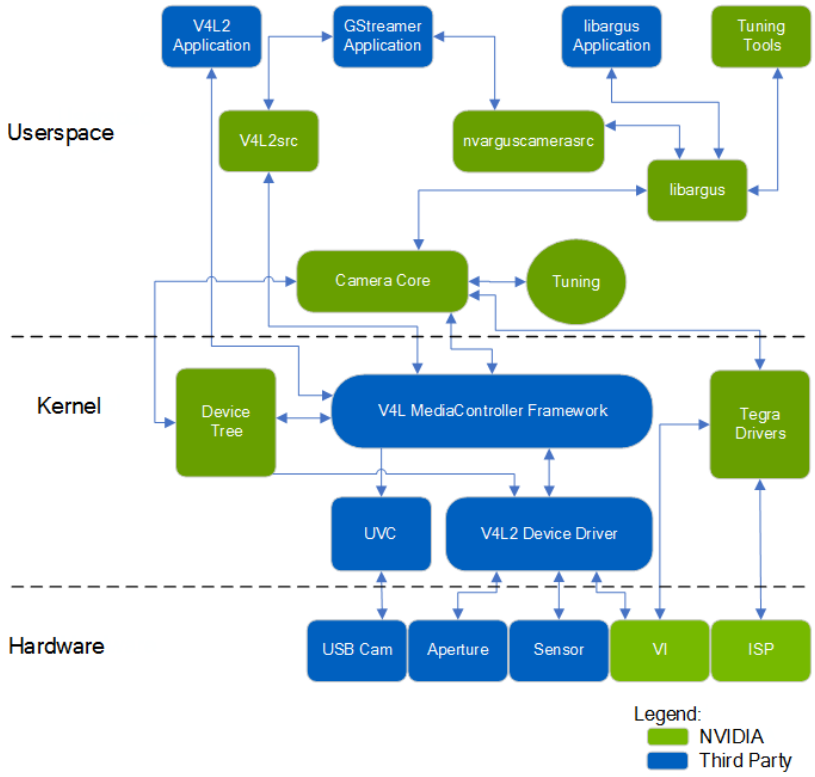
2.2 Pengenalan Kamera RGB yang digunakan

Proyek ini akan menggunakan Kamera RGB (*Red, Green,* Blue) dari NYK Nemesis A90 Everest. NYK Nemesis A90 Everest merupakan kamera *RGB* yang digunakan sebagai *streamer webcam* yang telah didukung dengan dengan *noise reducing mic*, HD *High Resolution* 1920x1080 dengan 30 FPS (*Frame per Second*), 360 derajat rotasi, *auto focus, auto exposure, white balance* (*NYK A90 Everest | Nemesis, n.d.*). Proyek ini menggunakan kamera tersebut karena memiliki jangkauan tangkap layar yang luas dan hasil tangkap yang cukup jelas. NYK Nemesis juga memiliki kamera versi *webcam* lainnya seperti *NYK A80 Nighthawk*, *NYK A95 Albatros*, dan lain sebagainya.

Gambar 2.2 NYK Nemesis A90 Everest

Kamera RGB lainnya yang dapat digunakan dalam mendeteksi kecacatan adalah Raspberry Pi *Camera Module* V2. Namun, dalam memanggil fungsi kamera sistem membutuhkan Nvarguscamera, sedangkan menggunakan *usb camera* membutuhkan v4l2src untuk memanggil fungsi kamera. Nvarguscamerasrc merupakan *plugin* dari kamera Nvidia yang menggunakan ISP (*interface* dari CSI (*Camera Serial Interface*)) dan mengendalikan menggunakan Argus API, sedangkan v4l2src merupakan standar Linux v4l2 yang dapat menggunakan *kernel* secara langsung. Kamera CSI membutuhkan Nvarguscamerasrc untuk

memanggil fungsi kamera. Akan tetapi harus melakukan *pre-processing* sebelum mengendalikannya. Namun, *usb camera* tidak menggunakan ISP dalam memanggil fungsi kamera, sehingga tidak melakukan *pre-processing* untuk menangkap gambar (Camera Architecture Stack | docs.nvidia, n.d.). Oleh sebab itu, kamera yang berjenis *CSI Camera* membutuhkan waktu lebih lama untuk melakukan proses tangkap gambar. Pada saat menjalankan *Neural Network*, sistem kerja GPU akan dipacu, sehingga seluruh kemampuan proses akan berpusat pada *Neural Network* sebagai prioritas kerja sistem. Akibat dari prioritas sistem pada *Neural Network* membuat *pre-processing* pada *CSI camera* menjadi terhambat. Dikarenakan kamera membutuhkan waktu proses yang relatif lama saat menangkap gambar, proyek ini akan menggunakan *USB Camera*, NYK Nemesis A90 Everest, sebagai kamera *RGB* guna menangkap gambar.



Gambar 2.3 *Camera Architecture Stack*

Sumber: *Nvidia Jetson Nano.* (n.d.). Retrieved October 8, 2022, from https://developer.nvidia.com/embedded/jetson-nano-developer-kit

2.3 Bahasa Pemrograman Python

Python adalah salah satu bahasa pemrograman yang tingkat tinggi yang telah ditafsirkan atau *Interpreted, object oriented*, dan dengan semantik dinamis. *Python* merupakan bahasa pemrograman yang telah populer dan banyak digunakan diberbagai bidang dikarenakan simpel dan memiliki sintaks yang mudah untuk dipelajari.

Python telah banyak digunakan diberbagai banyak bidang. *Python* cukup terkenal dikalangan pengembang karena berkembangnya kegunaan bahasa pemrograman diberbagai bidang. Para pengembang atau *developer*, menggunakan *python* dalam membuat aplikasi seperti pengembangan *web*, permainan, statistika, dan lain sebagainya (*Python.Org, n.d.*).

Python dilengkapi dengan *Python Package Index* yaitu suatu *third module yang* memudahkan pengembang dalam membangun sebuah sistem pengkodean. Pengembang dapat dengan mudah mengunduh segala kebutuhan dalam membangun sebuah sistem kode dengan *repository* yang telah disediakan oleh *python*. *Python* memiliki berbagai modul seperti membuat sebuah *virtual environment* yang dapat bekerja dalam sistem *python* itu sendiri, hingga modul projek yang dapat diunduh dan dipasang dalam sebuah sistem (*Pypi.Org, n.d.*).

Bahasa pemrograman *Python* sering dibandingkan dengan bahasa pemrograman lainnya seperti *Java*, *JavaScript*, *Perl*, dan lain sebagainya. Perbandingan dari bahasa pemrograman tersebut adalah sebagai berikut:

Tabel 2.1

Perbandingan Bahasa Pemrograman

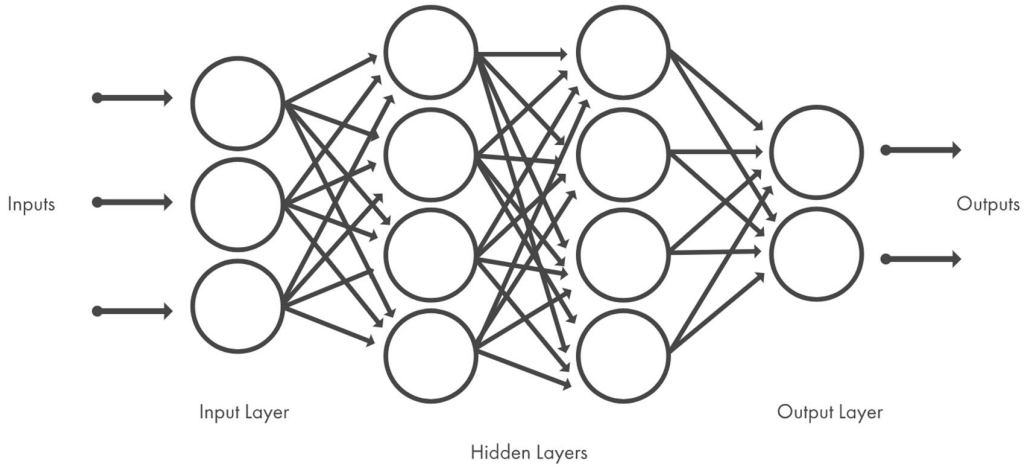
Bahasa Pemrograman	Python
Java	*Python* memiliki kecepatan yang lebih lambat daripada *Java* dalam segi jalan program. Namun, dibalik itu *Python* memiliki program yang secara menyeluruh lebih pendek daripada *Java*. Di sisi lain, *Python* memiliki program yang lebih dinamis dan lebih mudah untuk dipahami.
JavaScript	*Python* dapat menggunakan model *function* pemrograman yang lebih simple tanpa perlu mendefinisikan kelas. *Python* juga dapat digunakan dalam memprogram skala yang lebih besar dengan penggunaan ulang kode yang lebih lagi.
Perl	Kedua bahasa pemrograman memiliki fitur yang sama, tetapi memiliki fungsi utama yang berbeda. Di mana *Perl* difokuskan untuk *application-oriented,* sedangkan *Python* digunakan untuk struktur data.

Sumber: *Python* (n.d.). Python. Retrieved October 10, 2022, from https://www.python.org

2.4 Pengenalan Deep Learning

Deep learning merupakan sub bidang dari *Machine Learning* dan subset *multilayer Neural Network* yang terhubung bagaikan neuron saraf manusia. *Neural Network Deep Learning* memproses informasi secara dinamis berdasarkan data sumber. Bagaikan otak manusia, *Deep Learning* memproses data secara paralel untuk mempelajari, melakukan klasifikasi pola *outcome*, dan menyimpan Informasi data.

Deep Learning telah diaplikasikan ke berbagai bidang kegiatan untuk membantu dan memudahkan aktivitas manusia. Seperti *digital assistant* sebagai contoh Google Home Assistant, Siri, Alexa, dan lain sebagainya hingga *self–driving cars*. Pada bidang industri manufaktur *deep learning* telah membantu dalam mengoptimalkan proses *quality control* dari sebuah *production line*. *Deep Learning* juga dapat membantu mesin untuk berpikir bagaikan otak manusia, sehingga mampu melakukan aktivitas seperti *assembly* rangkaian mesin.



Gambar 2.4 Komponen *Neural Network*

Sumber: Deep Learning (n.d.). Deep learning. Retrieved June 21, 2023, from https://www.mathworks.com/discovery/deep-learning.html

Terdapat beberapa komponen yang menunjang jalannya *Deep Learning.* Komponen yang dimaksud adalah *Input Layer*, *Hidden Layer*, *Output Layer*. *Input layer* merupakan komponen *Deep Learning* yang berisikan data. *Hidden layer* adalah lapisan neural network yang bertujuan agar model data dapat lebih beradaptasi dengan input data yang diterima. *Deep Learning* dapat memiliki *hidden layer* yang berjumlah banyak sesuai dengan kebutuhan kompleksitas model. *Output layer* adalah hasil dari model yang memberikan sebuah kesimpulan dari model data yang diterima.

2.5 Pengenalan Algoritma YOLOv4-Tiny Sebagai Algoritma *Deep Learning*

YOLO merupakan akronim dari *You Only Look Once*. YOLO adalah salah satu algoritma *Deep Learning* yang memiliki *1 – stage Algorithm* yaitu model yang memiliki 1 fase tanpa memiliki proses deteksi *intermediate region* (Redmon. J & Farhadi. A, 2018). *1 – stage Algorithm* mendapatkan prediksi dari gambar dengan metode *region-free.* YOLO memproses keseluruhan dari image yang diinput dan melakukan prediksi berdasarkan *bounding box* dan kelas yang telah di-*assign*. Adapun struktur dari algoritma *Deep Learning* YOLO adalah sebagai berikut.

- Gambar Input

  Algoritma menerima gambar input dan melakukan penyamaan resolusi gambar sesuai dengan input *height* dan input *width* yang ada dalam konfigurasi model *Neural Network*.

- *Pre-processing*

  Gambar input yang diterima oleh algoritma akan dilakukan *pre-processing* seperti menyamakan nilai *pixel* dan rasio dari gambar agar sesuai dengan kebutuhan algoritma YOLO

- *Backbone Network*

  Dalam *backbone network* yang dimiliki oleh YOLO, terdapat CNN atau *convolutional neural network* yang memiliki fungsi untuk melakukan pengambilan dan pembelajaran fitur (*feature extraction*) dari sebuah input gambar.

- *Feature Extraction*

  Backbone network akan memproses gambar input melalui beberapa layer atau lapisan dari CNN. Melalui CNN sebagai *backbone network* YOLO, menjadikan model *neural network* untuk dapat melakukan *feature extraction* dalam beberapa ukuran hasil pembagian *grid*.

- *Grid Generation*

  YOLO membagi input image menjadi beberapa *grid* yang bertujuan untuk melakukan prediksi *bounding box* dan label kelas sesuai dengan lokasi dan besaran hasil anotasi. Besar nilai *grid* dapat tergantung dengan kebutuhan akurasi dan kecepatan dari model sendiri.

- *Prediction*

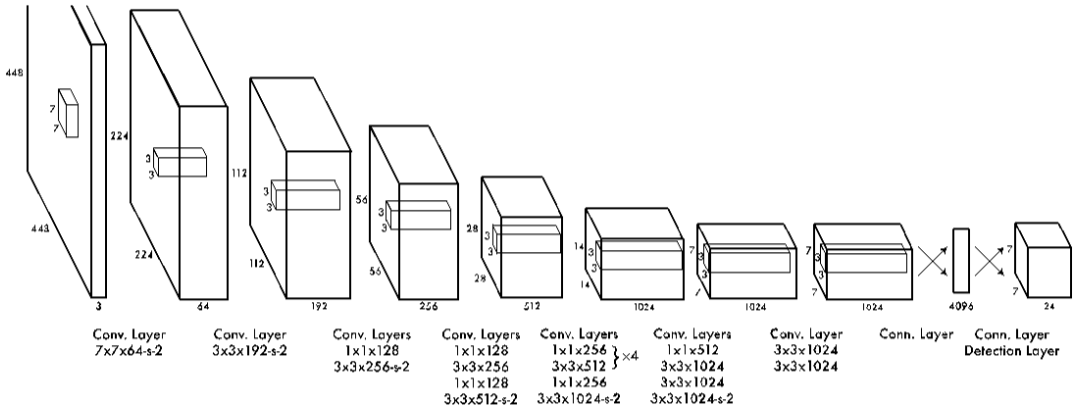
  Setiap kolom grid, YOLO akan memprediksi sesuai dengan *bounding box* dan kelas dari masing-masing fitur. Sesuai dengan hasil anotasi dengan format YOLO, terdapat nilai letak horizontal (X), nilai letak vertikal (Y), nilai *width*, dan nilai *height* yang mempresentasikan lokasi dan nilai besaran dari *bounding box*.

- *Non-Maximum Suppression*

  Non-maximum suppression merupakan metode yang berguna untuk menghilangkan pengulangan (*redundancy*) dan *overlapping* dari prediksi *bounding box*.

- *Output*

  Hasil dari proses algoritma YOLO merupakan prediksi *bounding box*, kelas, dan nilai *confidence* dari prediksi yang dilakukan.



Gambar 2.5 *YOLO Architecture*

Sumber: Ammar. A, et al. (2019). *Aerial Images Processing for Car Detection using Convolutional Neural Networks: Comparison between Faster R-CNN and YoloV3*. https:// https://arxiv.org/pdf/2004.11970.pdf

Keunikan dari algoritma YOLO adalah kemampuannya dalam mendeteksi secara *real time*. YOLO memproses keseluruhan image hanya 1 kali, sehingga membuat YOLO untuk dapat mendeteksi dalam kecepatan yang tinggi dan cocok untuk *inference* yang cepat. Perbedaannya dengan algoritma lainnya seperti RCNN (*Region Based Convolutional Neural Network*), YOLO melakukan prediksi *bounding box* dan probabilitas kelas secara menyeluruh. Berbeda dengan RCNN yang perlu menggunakan *Region Proposal* yang kompleks dan melakukan proses kembali, sehingga mengorban waktu yang lebih. Dengan ini YOLO memiliki keunikannya yaitu mampu mendeteksi dengan cepat karena simplisitas dan kemampuan mendeteksi dalam 1 kali.

YOLOv4-Tiny adalah model *real-time object detection* yang memiliki ukuran jaringan atau *network* yang lebih kecil dibandingkan dengan YOLOv4. YOLOv4-Tiny memiliki spesifikasi sebagai berikut:

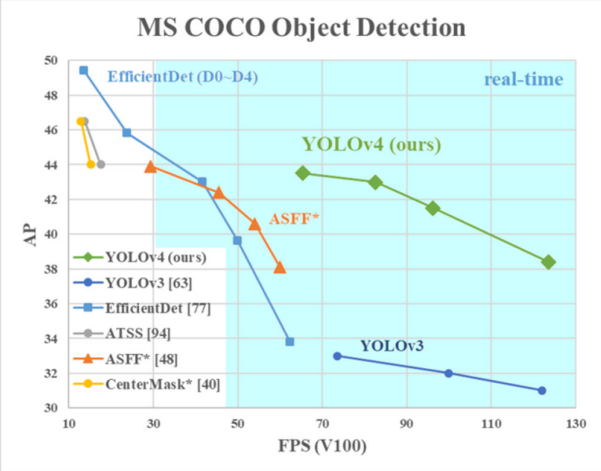
Tabel 2.2

Spesifikasi YOLO v4 Tiny

Metric	Value
Type	Detection
GFLOPs	6.9289
Mparams	6.0535
Source Framework	Keras

Sumber: *Yolov4 Tiny.* (2019). OpenVINO. Retrieved October 10, 2022, from https://docs.openvino.ai/2022.3/omz_models_model_yolo_v4_tiny_tf.html

YOLOv4-Tiny memiliki tipe arsitektur yang sama dengan YOLOv4. Selain YOLOv4 terdapat beberapa algoritma yang dipakai dalam mendeteksi objek. Terdapat YOLOv3 yaitu salah satu pendahulunya, EfficientDet, ATSS, ASFF, CenterMask dan lain sebagainya. Menurut hasil penelitian dari sebuah jurnal, perbandingan dari AP *(Average Precision)* dan perfoma FPS (*Frame Per Second*) dari berbagai algoritma adalah seperti pada gambar 2.5



Gambar 2.6 Perbandingan FPS dari Algoritma

Sumber: Bochkovskiy. A, et al. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection.* https://doi.org/10.48550/arXiv.2004.10934, p. 1.

YOLOv4-Tiny telah banyak digunakan diberbagai jenis aplikasi pendeteksi objek. Algoritma YOLOv4 digunakan dalam penugasan deteksi objek, Segmentasi, pelacakan objek, *self-driving cars,* dan lain sebagainya. Karena YOLOv4-Tiny hanya membutuhkan performa yang

lebih rendah daripada YOLOv4, YOLOv4-Tiny dapat dengan baik berjalan di *embedded system,* seperti Raspberry Pi hingga Nvidia Jetson Nano. (Bochkovskiy. A, et al., 2020).

Salah satu *Neural Network* yang menggunakan YOLOv4 tiny sebagai algoritma adalah *Neural Network Darknet*. *Darknet* adalah *Open-Source Neural Network Framework* yang tertulis dalam bahasa C dan bekerja dengan CUDA yang memiliki kemampuan yang cepat dan mudah dalam proses instalasi (Darknet | pjreddie.com, n.d.). CUDA merupakan *parallel computing platform and programming model* yang dikembangkan oleh NVIDIA untuk komputasi secara *general* dalam GPU (CUDA | *developer.nvidia.com, n.d.*). Sistem CUDA digunakan dalam proyek ini karena *Neural Network Darknet* membutuhkan CUDA untuk menjalankan *network* dengan *graphic processing unit* untuk mengoptimalkan performa *training* hingga *deploy* model data. *Darknet* dapat berjalan menggunakan CPU dalam proses *Neural Network*. Namun, dibalik kemampuan *Darknet* dalam berproses menggunakan CPU, *Darknet* dapat berjalan dengan lebih optimal jika berjalan menggunakan GPU. Oleh karena itu, pada proyek ini *Darknet* akan berjalan dengan GPU agar proses deteksi dapat berjalan dengan perfoma yang optimal.

2.6 Pengenalan *Training*, *Validation*, *Test*,dan *Deploy* dengan *Artificial Intelligence*

Dalam melakukan *training*, validasi, dan *deploy* dengan *Artificial Intelligence*, terdapat beberapa langkah-langkah untuk melakukannya. Sebelum melakukan *training*, terlebih dahulu pengembang harus menyiapkan *dataset* yang akan digunakan. Untuk itu, pengembang dapat menggunakan *dataset* yang telah ada atau *pre-trained dataset* seperti COCO *dataset* atau dapat juga membuat *dataset* sendiri dengan mengumpulkannya menggunakan gambar-gambar yang telah ditangkap. Dalam mengumpulkan data, dibutuhkan data yang cukup untuk memberikan hasil akurasi yang terbaik setelah melawati tahap *training*. Pada saat melakukan *training* dibutuhkan data sebanyak 150 – 500 per kelas (Shahinfar. S, et al., 2020). Dengan jumlah data 150 hingga 500 per kelas, maka model *training* dapat memberikan hasil presisi dan sentivitas yang tinggi. Oleh karena itu, jumlah data dapat mempengaruhi tingkat akurasi saat melakukan pendeteksi kecacatan.

Dalam mengolah *dataset,* data dipisahkan menjadi 2 bagian untuk *training set* dan *validation set*. Merujuk pada metode yang akan digunakan, maka tipe *Deep Learning* yang akan digunakan adalah *Supervised Learning,* sehingga pada *training dataset* setiap objek yang Ingin dipelajari oleh mesin harus diberikan label (Huawei AI Academy Training Materials*, 2020*). Untuk melakukan proses anotasi pada data yang diperoleh dibutuhkan *software* yang mampu menandai letak fitur yang ingin diambil pada setiap data. Salah satu *software* yang dipakai pada

proyek ini adalah *Label Studio*. *Label Studio* merupakan *open-source data labelling platform* yang digunakan sebagai alat anotasi (Open-Source Data Labelling Platform | *Label Studio, n.d.*). *Label Studio* dapat terintegrasi dengan *python virtual environment*, sehingga dalam proses instalasi *tool* tersebut hanya perlu menjalankan perintah "python pip" pada sistem komputasi yang diinginkan. Hasil anotasi dapat berupa beberapa tipe file. Akan tetapi, proyek ini akan menggunakan algoritma YOLO, sehingga format data adalah YOLO. Dengan menggunakan dataset hasil anotasi dengan tipe format YOLO, file konfigurasi akan terkumpul dalam gambar dan nilai *pixel bounding box* yang telah di-*assign*.

Setelah melakukan proses anotasi, dapat dilakukan proses *training.* Proses *training* dapat dilakukan sesuai dengan berapa kali pelatihan yang akan dilakukan dengan mengatur "Epoch" dari program. Setelah selesai melakukan *training*, model yang telah dilakukan akan melanjutkan ke fase *validation* untuk mencari angka *confidence threshold* yang baik untuk mendapatkan nilai presisi dan sensitivitas. Dengan menggunakan *weight* file hasil training, model data dapat melanjutkan fase *test* di mana *dataset* yang tidak dilakukan *labelling* akan digunakan sebagai *test data.* Setelah melakukan *training* hingga *test,* model dapat dijalankan dengan *weight* yang sesuai dengan arsitektur yang ditetapkan, merujuk pada batasan maka akan menggunakan *YOLOv4-Tiny Weight.* Kemudian, program akan menjalankan model dan akan terlihat seperti contoh dibawah ini.

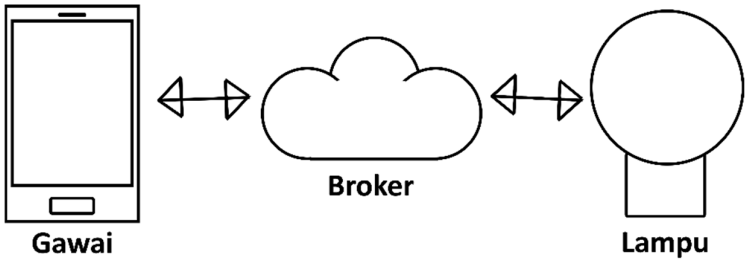
Gambar 2.7 Contoh Deteksi Objek dengan *YOLOv4-Tiny*

2.7 Pengenalan Metode Komunikasi MQTT

MQTT (*Message Queuing Telemetry Transport*) adalah *OASIS Standard Messaging Protocol* untuk IOT atau *Internet of Things*. MQTT merupakan protokol yang bekerja secara *machine to machine* selayaknya *Internet of Things* yang memiliki basis *open source*. MQTT didesain untuk mengirim dan menerima pesan untuk berkomunikasi secara jarak jauh ke perangkat-perangkat dengan ukuran *bandwidth* yang minim. MQTT dapat juga bekerja dalam kondisi jaringan yang memiliki *latency* tinggi atau jaringan yang tidak dapat diandalkan. MQTT telah digunakan diberbagai industri seperti otomotif, logistik, manufaktur, *smart home,* dan lain sebagainya. (*MQTT: The Standard for IoT Messaging|mqtt.org*).

MQTT menggunakan *broker* sebagai perantara pihak *publisher* dengan *subscriber.* Pada sistem MQTT terdapat beberapa jenis *broker* yang dapat digunakan. Seorang *programmer* dapat menggunakan *online broker* atau *offline broker* sesuai dengan kebutuhan sistem masing-masing. Dalam menjalankan *broker* seorang *programmer* pertama-tama harus menentukan *host broker* sebagai perantara antara *publisher* dengan *subscriber*. *Programmer* harus menentukan topik yang akan digunakan dalam sistem dan topik itu harus sama dengan yang digunakan oleh *publisher* dan *subscriber*. *Programmer* juga harus menetukan *message* yang akan dikirimkan dan diterima oleh *broker*.

MQTT dapat mendukung jalannya sistem IOT. Salah satu aplikasi sistem IOT yang dapat MQTT dukung adalah sistem rumah pintar. Sebagai contoh gawai dapat mengirimkan perintah nyala atau mati pada sebuah lampu melalui MQTT *broker* sesuai dengan *host* dan *topic* yang telah ditentukan. Dengan MQTT sistem IOT dapat lebih mudah digunakan dan dapat dikontrol dengan lebih leluasa.



Gambar 2.8 Sistem Protokol MQTT