

**PENGEMBANGAN MODEL DETEKSI OBJEK ALAT
PELINDUNG DIRI DALAM RUANG LINGKUP
PERUSAHAAN MANUFAKTUR**

Ditujukan untuk Memenuhi Ulangan Akhir Semester

Mata Kuliah Sistem Cerdas

(SIS61526)



DIBUAT OLEH:

| | |
|-------------------------|---------------|
| AZIZAH PAUZIAH | 2210631250006 |
| NASYWA SYIFA HANIFAH | 2210631250023 |
| ADITYA ALFAREZY DAMANIK | 2210631250037 |
| VERNANDA AYU PRASTIKA | 2210631250102 |

**PROGRAM STUDI SISTEM INFORMASI
FAKULTAS ILMU KOMPUTER
UNIVERSITAS SINGAPERBANGSA KARAWANG
KARAWANG**

2025

BAB I

PENDAHULUAN

1.1 Latar Belakang

Kemajuan teknologi informasi telah memasuki zaman di mana perkembangan teknologi berlangsung lebih cepat daripada yang pernah diantisipasi sebelumnya. Perangkat elektronik tidak hanya berperan sebagai alat untuk mengolah informasi, melainkan telah menjadi salah satu elemen kunci dalam persaingan untuk mencapai keunggulan yang lebih baik (Khadafi & Dwiyaksa, 2015). Salah satu proses di perusahaan yang bisa dimudahkan dengan kemajuan teknologi informasi adalah pengecekan penggunaan serta kepatuhan penggunaan alat pelindung diri dalam area kerja.

Perusahaan manufaktur yang memproduksi bahan jadi (*finished goods*) tentunya terlibat dalam serangkaian proses produksi yang melibatkan mesin dan peralatan berat yang dapat menimbulkan potensi bahaya bagi para pekerjanya.

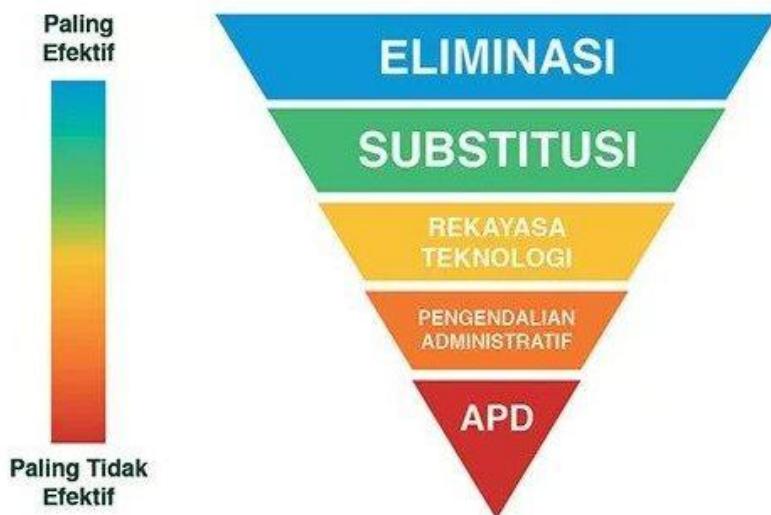


Gambar 1.1 Jumlah Kecelakaan Kerja Indonesia
(Sumber: dataindonesia.id, 2023)

Informasi yang diperoleh dari DataIndonesia.id mengungkapkan bahwa Badan Penyelenggara Jaminan Sosial (BPJS) Ketenagakerjaan mencatat peningkatan yang signifikan dalam jumlah kecelakaan kerja di Indonesia. Berdasarkan Gambar 1.1 yang diperoleh dari Januari hingga November 2022, terdapat 265.334 kasus kecelakaan kerja, menunjukkan kenaikan sebesar 13,26% dari total kasus sepanjang tahun 2021 yang mencapai 234.270 kasus. Tren ini menunjukkan bahwa jumlah kasus kecelakaan kerja di Indonesia terus meningkat secara konsisten 2017 hingga 2022, meskipun data tersebut baru mencakup periode 11 bulan, jumlahnya telah mencatat rekor tertinggi dalam rentang waktu tersebut.

Dalam perusahaan manufaktur, penggunaan alat pelindung diri sebenarnya dianggap sebagai opsi paling tidak efektif dalam pengendalian bahaya. Hierarki pengendalian bahaya merupakan sistem yang digunakan buat mengurangi risiko bahaya yang bisa muncul pada suatu organisasi. Ini didasarkan pada prinsip bahwa terdapat beberapa tingkatan kontrol yang dapat digunakan, mulai dari menghilangkan bahaya sampai mengurangi risiko melalui kontrol yang lebih baik.

Standar OHSAS 18001 yang sekarang dikenal sebagai 45001:2018, adalah standar internasional untuk sistem manajemen kesehatan dan keselamatan kerja (K3). Standar tersebut memiliki persyaratan untuk organisasi/perusahaan supaya membangun hierarki pengendalian K3. Hierarki pengendalian bahaya tersebut dijelaskan pada gambar di bawah ini:



Gambar 1.2 Hierarki Pengendalian Bahaya Risiko K3

APD merupakan tingkatan terakhir dalam hierarki yang dinilai paling tidak efektif. APD adalah peralatan yang digunakan untuk melindungi tubuh pekerja dari bahaya yang mungkin muncul selama proses kerja. Penempatan APD dalam tingkatan terakhir bukanlah tanpa alasan, hal tersebut karena APD hanyalah melindungi pekerja dari dampak bahaya bukan menghilangkan bahaya itu sendiri. Selain itu, APD sendiri memiliki beberapa kekurangan serta sangat tergantung pada *manpower* yang sedang beroperasi di lapangan. Efektivitas APD sangat bergantung pada perilaku kerja, jika *manpower* sendiri tidak patuh, tidak mengenakan, tidak sesuai syarat maka APD tidak dapat memberikan perlindungan yang maksimal. Permasalahan tersebut sebenarnya berasal dari ketidaknyamanan *manpower* dalam menggunakan APD dalam jangka waktu panjang.

Dalam perusahaan, agar kepatuhan *manpower* dalam mengikuti persyaratan dan peraturan APD terdapat sebuah divisi yang secara langsung mengawasi *manpower* tersebut. Divisi tersebut dinamakan EHS (Environment, Health & Safety). Namun, pengawasan tersebut tentunya akan sangat sulit karena luasnya area kerja dan *staff* EHS yang tentunya tidak selalu sebanding dengan banyaknya *manpower* yang diawasi. Sehingga terdapat kesulitan yang sangat merepotkan untuk mengawasi

manpower tersebut, proses ini sering kali tidak teliti sehingga memungkinkan banyaknya penyimpangan yang dilakukan *manpower* dalam penggunaan APD.

Berdasarkan permasalahan tersebut, penulis memiliki tujuan untuk mengembangkan suatu model machine learning yang dapat mendeteksi object berupa APD dan terintegrasi dengan aplikasi web. Sistem ini dirancang untuk digunakan dalam perusahaan manufaktur yang melibatkan aktivitas yang dapat berdampak pada keselamatan atau kesehatan kerja. Model ini dapat mendeteksi penggunaan APD seperti Sepatu bot, Pelindung telinga, Kaca, Sarung tangan, Helm, Masker, Orang, dan Rompi. Dengan adanya pengembangan sistem ini, diharapkan perusahaan manufaktur dapat menyelesaikan permasalahan yang sedang dihadapi mereka.

1.2 Rumusan Masalah

Berdasarkan penjelasan mengenai latar belakang yang telah disampaikan, permasalahan yang akan diteliti dapat dirumuskan sebagai berikut:

1. Bagaimana merencanakan dan mengembangkan model machine learning object detection menggunakan CNN untuk PT. XYZ?
2. Bagaimana hasil pengujian Safety Check Sheet System (SCSS) berbasis web yang telah dikembangkan tersebut?

1.3 Batasan Masalah

Agar penelitian ini lebih terarah dan fokus, maka dilakukan beberapa batasan sebagai berikut:

1. Supaya pembahasan tidak terlalu luas dan tetap fokus, dalam proyek ini penulis membatasi hal-hal yang akan dibahas sebagai berikut:
2. Sistem deteksi hanya dibuat untuk mengenali rompi keselamatan (vest), bukan alat pelindung diri (APD) lainnya seperti helm, sepatu, atau masker.
3. Data yang digunakan untuk melatih sistem diambil dari Roboflow, dengan dataset khusus yang hanya berisi gambar orang yang memakai atau tidak memakai vest.

4. Model yang digunakan dibuat dengan algoritma CNN (Convolutional Neural Network), meskipun format datanya memakai format YOLO.
5. Sistem ini hanya ditujukan untuk penggunaan dalam lingkungan perusahaan manufaktur, khususnya untuk memantau pemakaian vest oleh pekerja.
6. Sistem tidak menghilangkan bahaya secara langsung, tapi hanya membantu dalam mengawasi kepatuhan pekerja terhadap penggunaan APD.

BAB II

PEMBAHASAN

A. Informasi Dataset

Dalam pengembangan model, digunakan dataset yang relevan dengan alat pelindung diri, dataset tersebut didapatkan dari beberapa website, namun sebagai sumber utama data berasal dari Roboflow . Dataset dari Roboflow merupakan dataset yang berisi banyak image mengenai vest, sesuai dengan batasan dari proyek yang dimana hanya bisa mendeteksi apakah seseorang menggunakan vest atau tidak. Dataset tersebut dapat diakses melalui link: <https://universe.roboflow.com/mainel/vest-cye3g>. Dataset memiliki dua class, yakni no vest dan vest. Dengan informasi metric sebagai berikut:

| | |
|-----------|-------|
| mAP@50 | 98.3% |
| Precision | 99.7% |
| Recall | 96.5% |

Dataset sendiri memiliki total hingga 2637 images yang dimana, dari semua image tersebut dibagi menjadi beberapa bagian, pembagian tersebut sebagai berikut:

| | Jumlah Image | Persentase (%) |
|-----------|--------------|----------------|
| Train Set | 1841 | 70 |
| Valid Set | 531 | 20 |
| Test Set | 265 | 10 |

Dataset dari Roboflow dapat diunduh atau didapatkan melalui kode. Untuk mendapatkan melalui kode dapat digunakan kode seperti dibawah berikut:

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="meXIKI7Irtem8nZQ364G")
project = rf.workspace("mainel").project("vest-cye3g")
version = project.version(1)
dataset = version.download("yolov11")
```

Selain dari kode, dapat langsung diunduh dalam bentuk zip. Kedua cara pengunduhan tersebut membutuhkan pengguna harus melakukan login terlebih dahulu. Dalam mengunduh dataset dari Roboflow, pengguna dapat memilih format dari dataset yang diinginkan. Roboflow menyediakan banyak sekali format, seperti pada gambar di bawah ini:



Dalam pengembangan model machine learning ini, digunakan format YOLOv11, namun ketika proses training tetap menggunakan algoritma CNN tanpa menggunakan pre-trained model dari YOLO.

B. Format Data YOLO

Format data YOLO (You Only Look Once) adalah cara menyimpan anotasi bounding box untuk deteksi objek dalam bentuk teks yang sederhana dan efisien. Format ini digunakan selama proses pelatihan dan inferensi,

terutama saat menggunakan model YOLO versi apa pun (v1–v8, dan seterusnya).

Struktur dataset dari YOLO juga sudah ditetapkan dan distandarisasikan yang dimana struktur datasetnya adalah sebagai berikut:

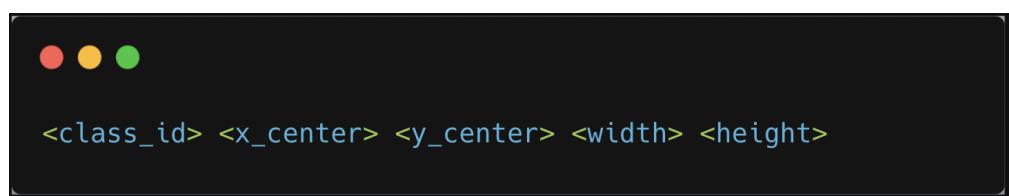
dataset/

```
  └── images/
      ├── train/
      ├── val/
      └── test/
  └── labels/
      ├── train/
      ├── val/
      └── test/
└── data.yaml
```

Dalam struktur dataset tersebut terdapat beberapa folder dan file yang masing-masing memiliki kegunaan, yakni:

1. images/ : Berisi gambar vest dalam format .png atau .jpg
2. labels/ : Berisi file anotasi dari gambar dalam format .txt
3. data.yaml: Konfigurasi dataset (class names, path, dll)

Dalam setiap gambar yang ada di dataset terdapat sebuah file .txt yang berisi anotasi letak serta informasi objek pada sebuah gambar. Anotasi sendiri memiliki format yakni,



Dalam format tersebut terdapat beberapa bagian, berikut adalah deskripsi dari tiap-tiap bagian tersebut:

1. class_id: Indeks kelas, berupa integer mulai dari 0
2. x_center: Koordinat X titik tengah bounding box (relatif)
3. y_center: Koordinat Y titik tengah bounding box (relatif)

4. width: Lebar bounding box (relatif)
5. height: Tinggi bounding box (relatif)

Selain itu, terdapat hal penting lain yakni file data.yaml yang dimana file ini berfungsi sebagai file konfigurasi dataset yang memberi tahu model tentang informasi penting yang dibutuhkan selama proses pelatihan dan validasi. File ini digunakan oleh framework seperti Ultralytics YOLO untuk memahami di mana gambar dan label disimpan, serta apa saja kelas-kelas objek yang akan dideteksi.

C. Struktur Project

Dalam pengembangan proyek, struktur file dan folder yang tertata dengan rapi merupakan salah satu cara untuk meningkatkan kualitas kode dan proyek. Dalam pengembangan model ini, file dikelompokkan ke dalam folder dengan nama yang relevan agar lebih mudah diatur dan dikembangkan. Berikut adalah struktur project dari pengembangan model:

```
├── README.md
├── build
│   ├── best_vest_model.h5
│   └── best_vest_model.onnx
├── images
│   └── vest_ppe
├── requirements.txt
└── runs
    └── detect
        ├── yolo_comparison_model
        │   ├── args.yaml
        │   └── weights
        └── yolo_comparison_model2
            ├── args.yaml
            ├── labels.jpg
            ├── labels_correlogram.jpg
            ├── train_batch0.jpg
            └── train_batch1.jpg
```

```
|   └── train_batch2.jpg
|   └── weights
|
└── src
    ├── crawl-image.ipynb
    ├── preparation.ipynb
    ├── train-with-cnn.ipynb
    └── train-with-yolo.ipynb
    └── yolo11n.pt
```

Berikut adalah penjelasan dari masing-masing file dan folder yang ada di dalam project:

1. README.md

File dokumentasi utama. Biasanya berisi penjelasan proyek, cara menjalankan kode, arsitektur model, dan instruksi pelatihan atau deployment.

2. build/

Folder ini menyimpan hasil pelatihan model CNN, dalam dua format:

2.1 best_vest_model.h5: Model dalam format Keras (TensorFlow).

2.2 best_vest_model.onnx: Model yang telah dikonversi ke ONNX untuk keperluan inferensi lintas platform (misalnya di Go, C++, atau edge devices).

3. images/vest_ppe

Berisi gambar dataset yang berkaitan dengan deteksi objek, kemungkinan besar untuk klasifikasi/deteksi vest (rompi keselamatan) atau alat pelindung diri (PPE - Personal Protective Equipment).

4. requirements.txt

File daftar dependensi Python. Isinya adalah paket-paket yang perlu diinstal (seperti torch, opencv-python, ultralytics, dsb.) agar proyek ini dapat dijalankan.

5. runs/detect/

Folder runs/detect adalah tempat di mana hasil dari proses pelatihan atau inferensi model YOLO disimpan secara otomatis oleh framework (dalam hal ini Ultralytics). Di dalam folder ini, terdapat subfolder yang dinamai

sesuai dengan eksperimen yang telah dijalankan, dalam hal ini adalah yolo_comparison_model dan yolo_comparison_model2. Masing-masing subfolder ini merepresentasikan hasil dari satu sesi pelatihan atau evaluasi model.

Setiap subfolder berisi berbagai berkas dan visualisasi yang berkaitan dengan eksperimen tersebut. File args.yaml menyimpan konfigurasi parameter pelatihan, seperti ukuran batch, jumlah epoch, ukuran gambar, atau jalur dataset. Ada juga gambar seperti train_batch0.jpg, train_batch1.jpg, dan seterusnya, yang menampilkan pratinjau data yang digunakan selama pelatihan—biasanya berisi gambar dengan anotasi bounding box, berguna untuk mengecek apakah data label sudah benar.

Beberapa hasil visualisasi lain seperti labels.jpg dan labels_correlogram.jpg memperlihatkan distribusi kelas dan hubungan antar label dalam dataset. Folder weights di dalam setiap eksperimen menyimpan file model YOLO hasil pelatihan, biasanya berupa file seperti best.pt atau last.pt, yang bisa digunakan kembali untuk inferensi atau deployment di masa depan.

6. src/

Berisi file kode dalam bentuk notebook (Jupyter) yang memiliki kegunaannya masing-masing:

- 6.1 crawl-image.ipynb: kemungkinan digunakan untuk mengambil dataset dari internet (web scraping/crawling).
- 6.2 preparation.ipynb: proses pra-pemrosesan data, seperti resize gambar, augmentasi, atau konversi anotasi.
- 6.3 train-with-cnn.ipynb: notebook untuk melatih model CNN kustom (non-YOLO).
- 6.4 train-with-yolo.ipynb: notebook untuk melatih model YOLO, mungkin dengan menggunakan ultralytics atau framework serupa.

7. yolo11n.pt

Model YOLO yang sudah dilatih, dalam format .pt (PyTorch). YOLO pada proyek ini adalah versi 11 dan n merujuk ke versi nano

(model ringan). Model ini digunakan untuk melakukan *transfer learning* sehingga dalam pengembangan hanya perlu disesuaikan datasetnya saja.

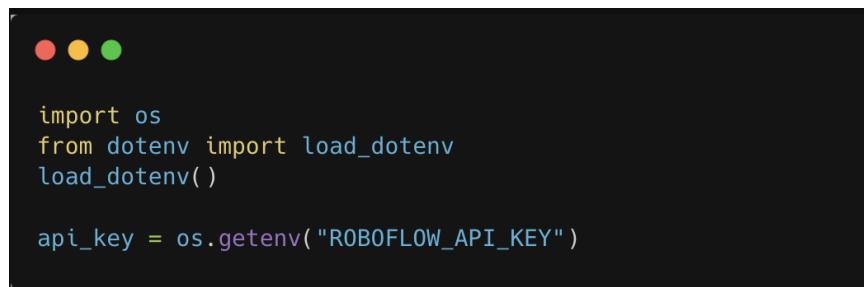
D. Training Model dengan CNN

Pengembangan model menggunakan algoritma CNN dilakukan dengan bahasa pemrograman Python dengan versi 3.11.13 dan dilakukan dalam Visual Studio Code. Seperti yang dijelaskan dalam struktur proyek, terdapat beberapa file yang digunakan untuk membentuk atau mengembangkan model. Oleh karena itu penjelasan akan dikelompokkan Berikut adalah penjelasan dari masing-masing file beserta kode program yang ada di dalamnya:

a. preparation.ipynb

Kode pada file ini bertujuan untuk mengunduh dataset dari Roboflow yang telah dipersiapkan untuk digunakan dalam pelatihan model deteksi objek menggunakan dengan format yang dipilih adalah YOLOv11.

1. Load library dan mendapatkan key



```
import os
from dotenv import load_dotenv
load_dotenv()

api_key = os.getenv("ROBOFLOW_API_KEY")
```

Untuk mengunduh dataset dari Roboflow, Kode ini dimulai dengan memuat environment variable dari file .env melalui library dotenv, yang memungkinkan penyimpanan aman dan terpisah dari informasi sensitif seperti API key. API key untuk Roboflow diambil dari *environment variable* dan disimpan dalam dengan nama api_key.

2. Persiapan Dataset

```
from roboflow import Roboflow
rf = Roboflow(api_key=api_key)
project = rf.workspace("mainel").project("vest-cye3g")
version = project.version(1)
dataset = version.download("yolov11", location="dataset")
```

Selanjutnya, kode mengimpor library Roboflow, yang merupakan cara Python untuk berinteraksi dengan layanan Roboflow. Objek Roboflow dibuat menggunakan API key tersebut untuk mengautentikasi pengguna.

Kemudian, skrip ini mengakses workspace bernama "mainel", dan dari sana mengambil proyek dengan ID "vest-cye3g". Setelah proyek ditemukan, kode mengakses versinya yang ke-1 dan mengunduh dataset tersebut dalam format yolov11. Dataset akan disimpan di folder lokal bernama dataset. Format YOLOv11 ini berarti data anotasi sudah disesuaikan agar kompatibel dengan pipeline pelatihan model YOLO versi terbaru.

b. train-with-cnn.ipynb

1. Import Library

```
# =====#
# BAGIAN 1: IMPORT LIBRARIES
# =====#

import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,
ModelCheckpoint
import warnings
import os
os.environ['PILLOW_VERSION'] = '1' # Force Keras to use Pillow
from PIL import Image
print("Pillow is working!")

# Kemudian import Keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
warnings.filterwarnings('ignore')

# Set random seed untuk reproducibility
np.random.seed(42)
tf.random.set_seed(42)

print("TensorFlow version:", tf.__version__)
print("GPU Available:", tf.config.list_physical_devices('GPU'))
```

Kode ini mempersiapkan environment Python untuk membangun dan melatih model klasifikasi citra menggunakan TensorFlow dan Keras, dengan bantuan beberapa library tambahan untuk analisis data, visualisasi, serta pra-pemrosesan gambar. Proses ini mencakup impor berbagai library, pengaturan konfigurasi, dan pengecekan awal terhadap perangkat keras yang tersedia, khususnya GPU.

Langkah pertama adalah mengimpor sejumlah library penting. Library seperti “os” dan “warnings” digunakan untuk manajemen sistem dan pengendalian peringatan. “cv2” (OpenCV) digunakan untuk pemrosesan citra atau dalam hal ini adalah image, sedangkan “numpy” dan “pandas” membantu dalam manipulasi data numerik dan tabular. “matplotlib.pyplot” dan “seaborn” digunakan untuk membuat visualisasi data, seperti grafik dan heatmap, yang penting dalam evaluasi model.

Selain itu, skrip mengimpor tool dari scikit-learn untuk membagi data menjadi set pelatihan dan pengujian, melakukan encoding label, serta menghitung metrik evaluasi seperti akurasi, confusion matrix, dan laporan klasifikasi.

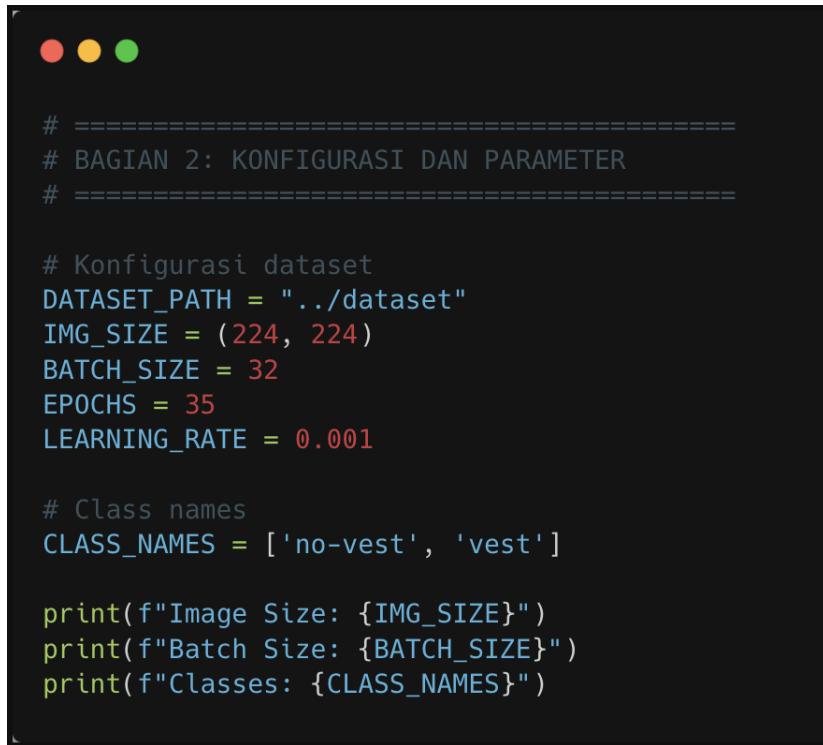
Kemudian, TensorFlow dan Keras diimpor, lengkap dengan modul-modul seperti “layers”, “ImageDataGenerator”, dan berbagai callback penting seperti “EarlyStopping”, “ReduceLROnPlateau”, dan “ModelCheckpoint”. Callback ini membantu mengontrol proses pelatihan, seperti menghentikan pelatihan saat performa stagnan atau menyimpan model terbaik secara otomatis.

Salah satu hal yang perlu adalah pengaturan environment variable “PILLOW_VERSION” ke “1”, yang merupakan cara untuk memaksa Keras menggunakan library “Pillow” sebagai backend untuk memproses gambar. Hal ini diikuti oleh impor langsung dari modul “PIL.Image” dan verifikasi bahwa Pillow

dapat digunakan dengan mencetak pesan “Pillow is working!”. Hal ini digunakan untuk memastikan Pillow berhasil di-import.

Untuk memastikan konsistensi hasil dan reproducibility, skrip menetapkan seed acak untuk NumPy dan TensorFlow ke nilai tetap, yaitu 42. Terakhir, versi TensorFlow yang sedang digunakan ditampilkan, dan sistem akan mencetak daftar perangkat GPU yang tersedia untuk menunjukkan apakah pelatihan dapat dilakukan menggunakan akselerasi perangkat keras.

2. Konfigurasi Dataset dan Parameter



```
# =====
# BAGIAN 2: KONFIGURASI DAN PARAMETER
# =====

# Konfigurasi dataset
DATASET_PATH = "../dataset"
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
EPOCHS = 35
LEARNING_RATE = 0.001

# Class names
CLASS_NAMES = ['no-vest', 'vest']

print(f"Image Size: {IMG_SIZE}")
print(f"Batch Size: {BATCH_SIZE}")
print(f"Classes: {CLASS_NAMES}")
```

Kode ini digunakan untuk menetapkan parameter dan konfigurasi utama yang akan digunakan dalam proses training model klasifikasi gambar. Pengaturan ini penting untuk memastikan bahwa seluruh pipeline machine learning berjalan konsisten dan efisien, serta sesuai dengan struktur data yang tersedia.

Pertama, ditentukan lokasi dataset dengan menetapkan DATASET_PATH ke direktori yang mengarah ke folder dataset. Path ini bersifat relatif terhadap lokasi kode.

Selanjutnya, ukuran gambar yang akan digunakan dalam proses training diatur ke (224, 224) piksel melalui variabel IMG_SIZE. Ini adalah ukuran standar yang banyak digunakan dalam arsitektur deep learning populer seperti MobileNet, ResNet, dan lainnya, karena memberikan keseimbangan antara resolusi informasi visual dan efisiensi komputasi.

Parameter BATCH_SIZE ditetapkan ke 32, yang berarti bahwa selama pelatihan, model akan memproses 32 gambar sekaligus dalam satu langkah pembaruan bobot. Penggunaan batch memungkinkan pelatihan lebih stabil dan efisien dibandingkan jika dilakukan satu per satu atau seluruh dataset sekaligus.

Jumlah epoch, yaitu jumlah iterasi penuh melalui seluruh dataset selama pelatihan, diatur ke 35. Ini adalah jumlah yang umum digunakan untuk memberi cukup waktu model belajar, sambil tetap memperhatikan risiko overfitting.

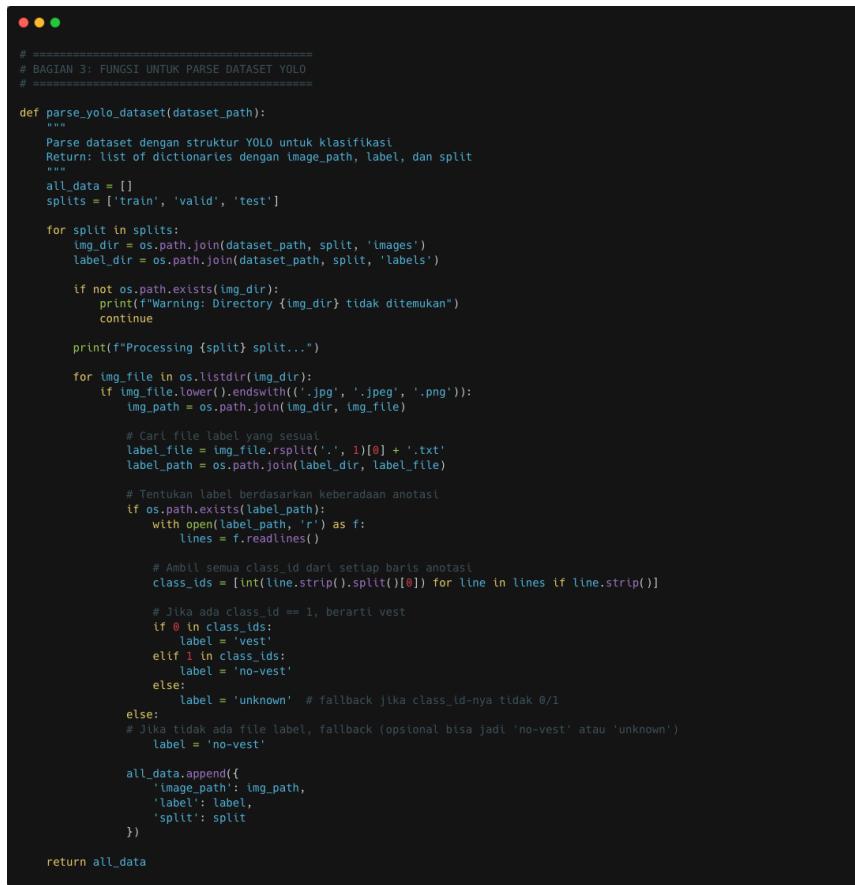
Tingkat pembelajaran (LEARNING_RATE) ditetapkan ke 0.001, yang merupakan nilai awal umum dalam banyak kasus. Nilai ini menentukan seberapa besar model akan menyesuaikan bobotnya pada setiap langkah berdasarkan kesalahan yang dihitung. Nilai terlalu besar dapat menyebabkan model tidak stabil, sementara nilai terlalu kecil dapat memperlambat proses pembelajaran.

Terakhir, CLASS NAMES didefinisikan sebagai daftar berisi dua kelas: 'no-vest' dan 'vest', yang menunjukkan bahwa model akan dikembangkan untuk membedakan antara gambar dengan dan tanpa rompi (vest). Ini adalah bagian dari task klasifikasi biner.

3. Parse YOLO Dataset

Pada bagian ketiga, kode selanjutnya akan mencoba untuk memarsing anotasi dari dataset yang bertipe yolov11. Sebelumnya dataset yang diunduh bertipe yolov11 sehingga memiliki format anotasi seperti yang telah dijelaskan sebelumnya.

Untuk membaca anotasi YOLO dibuatlah function parse_yolo_dataset() yang membaca dan mengekstraksi informasi dari dataset berformat YOLO. Tujuan utamanya adalah membentuk sebuah struktur data yang berisi informasi gambar dan label klasifikasinya, berdasarkan struktur folder standar YOLO yang terbagi ke dalam tiga subset: train, valid, dan test. Seperti yang telah dijelaskan sebelumnya. Berikut adalah kode dari function tersebut:



```
# =====#
# BAGIAN 3: FUNGSI UNTUK PARSE DATASET YOLO
# =====#

def parse_yolo_dataset(dataset_path):
    """
    Parse dataset dengan struktur YOLO untuk klasifikasi
    Return: list of dictionaries dengan 'image_path', 'label', dan 'split'
    """
    all_data = []
    splits = ['train', 'valid', 'test']

    for split in splits:
        img_dir = os.path.join(dataset_path, split, 'images')
        label_dir = os.path.join(dataset_path, split, 'labels')

        if not os.path.exists(img_dir):
            print("Warning: Directory {img_dir} tidak ditemukan")
            continue

        print(f"Processing {split} split...")

        for img_file in os.listdir(img_dir):
            if img_file.lower().endswith('.jpg', '.jpeg', '.png'):
                img_path = os.path.join(img_dir, img_file)

                # Cari file label yang sesuai
                label_file = img_file.rsplit('.', 1)[0] + '.txt'
                label_path = os.path.join(label_dir, label_file)

                # Tentukan label berdasarkan keberadaan anotasi
                if os.path.exists(label_path):
                    with open(label_path, 'r') as f:
                        lines = f.readlines()

                    # Ambil semua class_id dari setiap baris anotasi
                    class_ids = [int(line.strip().split()[0]) for line in lines if line.strip()]

                    # Jika ada class_id == 1, berarti vest
                    if 0 in class_ids:
                        label = 'vest'
                    elif 1 in class_ids:
                        label = 'no-vest'
                    else:
                        label = 'unknown' # fallback jika class_id-nya tidak 0/1

                    # Jika tidak ada file label, fallback (opsional bisa jadi 'no-vest' atau 'unknown')
                    label = 'no-vest'

                    all_data.append({
                        'image_path': img_path,
                        'label': label,
                        'split': split
                    })
    return all_data
```

Fungsi ini menerima satu argumen, yaitu dataset_path, yang menunjukkan direktori root tempat dataset YOLO disimpan. Di dalam direktori tersebut, terdapat subfolder bernama train, valid,

dan test, masing-masing berisi dua folder: images dan labels. Hal ini sesuai dengan struktur folder dari dataset.

Proses dimulai dengan mendefinisikan list kosong `all_data` yang akan menyimpan semua data hasil parsing. Lalu, fungsi melakukan iterasi terhadap ketiga split dataset (train, valid, dan test). Untuk setiap split, dibentuk path menuju folder gambar dan folder label. Jika folder gambar tidak ditemukan, akan dicetak peringatan dan proses untuk split tersebut dilewati.

Jika folder gambar ada, fungsi akan melanjutkan dengan memproses setiap file gambar di dalamnya. File gambar yang diakui adalah yang memiliki ekstensi .jpg, .jpeg, atau .png. Untuk setiap gambar, fungsi menentukan path lengkap ke file gambar dan juga mencari file label dengan nama yang sama namun berekstensi .txt.

Jika file label ditemukan, maka file tersebut dibuka dan dibaca seluruh barisnya. Setiap baris dalam file label merepresentasikan satu objek dalam gambar, dengan format `class_id x_center y_center width height` sesuai format YOLO. Fungsi kemudian mengekstrak semua `class_id` dari baris-baris tersebut.

Label klasifikasi kemudian ditentukan berdasarkan isi file label. Jika ditemukan class ID 0, maka gambar dianggap mengandung objek "vest". Jika hanya ada class ID 1, maka dianggap sebagai "no-vest". Bila tidak ditemukan keduanya, label akan diberi nilai "unknown" sebagai bentuk fallback. Jika file label tidak ada sama sekali, fungsi secara default menetapkan label sebagai "no-vest", hal ini dikarenakan bahwa jika tidak ada objek berarti tidak ada vest.

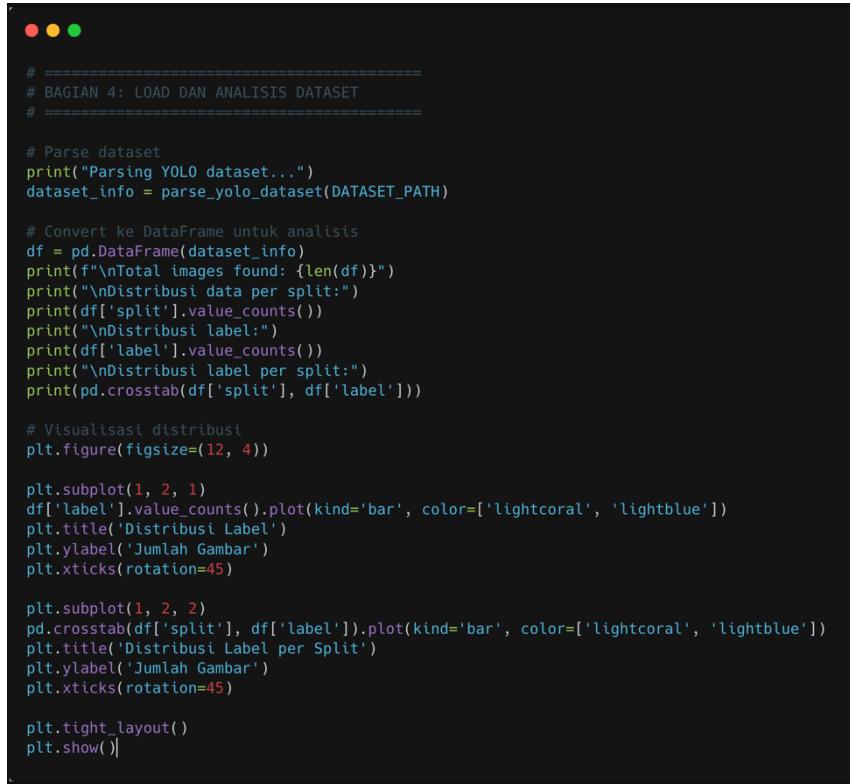
Setelah label ditentukan, informasi gambar tersebut disimpan dalam bentuk dictionary yang berisi tiga elemen: path gambar (`image_path`), label (`label`), dan informasi split (`split`). Dictionary ini kemudian ditambahkan ke list `all_data`.

Setelah seluruh gambar dari semua split diproses, fungsi mengembalikan all_data, yaitu list berisi informasi lengkap tentang semua gambar yang ditemukan dan label klasifikasinya.

4. Load dan Visualisasi Informasi Dataset

Pada bagian keempat, kode ini bertujuan untuk memuat, menyusun ulang, dan melakukan eksplorasi awal terhadap dataset YOLO yang telah diparse sebelumnya, hal ini dilakukan guna memahami struktur data sebelum digunakan dalam proses pelatihan model klasifikasi citra.

Berikut adalah kode program dari bagian ini yang diperlihatkan pada gambar di bawah:



```
# =====
# BAGIAN 4: LOAD DAN ANALISIS DATASET
# =====

# Parse dataset
print("Parsing YOLO dataset...")
dataset_info = parse_yolo_dataset(DATASET_PATH)

# Convert ke DataFrame untuk analisis
df = pd.DataFrame(dataset_info)
print(f"\nTotal images found: {len(df)}")
print("\nDistribusi data per split:")
print(df['split'].value_counts())
print("\nDistribusi label:")
print(df['label'].value_counts())
print("\nDistribusi label per split:")
print(pd.crosstab(df['split'], df['label']))

# Visualisasi distribusi
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
df['label'].value_counts().plot(kind='bar', color=['lightcoral', 'lightblue'])
plt.title('Distribusi Label')
plt.ylabel('Jumlah Gambar')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
pd.crosstab(df['split'], df['label']).plot(kind='bar', color=['lightcoral', 'lightblue'])
plt.title('Distribusi Label per Split')
plt.ylabel('Jumlah Gambar')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```

Pertama, fungsi `parse_yolo_dataset()` dipanggil untuk membaca isi dataset dari folder yang telah ditentukan sebelumnya (`DATASET_PATH`). Hasil dari proses parsing tersebut berupa list berisi dictionary untuk setiap gambar, yang mencakup path gambar, label klasifikasinya ('vest' atau 'no-vest'), serta split dataset-nya ('train', 'valid', atau 'test'). List ini disimpan dalam variabel `dataset_info`.

Data tersebut kemudian dikonversi menjadi sebuah objek DataFrame dari library pandas dan disimpan dalam variabel df. Dengan mengubahnya ke dalam bentuk tabular ini, proses analisis data menjadi lebih mudah dan terstruktur. Setelah itu, beberapa informasi ringkas ditampilkan ke konsol: jumlah total gambar yang ditemukan, jumlah gambar pada setiap split dataset, distribusi jumlah gambar berdasarkan label, serta distribusi gabungan antara label dan split dataset menggunakan pd.crosstab.

Untuk membantu pemahaman visual terhadap distribusi data, dua grafik batang (bar chart) dibuat menggunakan matplotlib. Grafik pertama menunjukkan total jumlah gambar untuk setiap label ('vest' dan 'no-vest') tanpa mempertimbangkan split dataset. Warna yang digunakan sengaja dibedakan agar mudah dibaca, yaitu lightcoral untuk satu kelas dan lightblue untuk yang lainnya.

Grafik kedua memperlihatkan distribusi label pada masing-masing split dataset (train, valid, dan test) secara berdampingan, menggunakan pd.crosstab() yang di-plot sebagai grafik batang. Ini berguna untuk melihat apakah data pada setiap subset memiliki proporsi kelas yang seimbang atau tidak, yang penting untuk memastikan bahwa model tidak bias selama pelatihan dan evaluasi.

Fungsi plt.tight_layout() digunakan untuk mengatur layout visual agar grafik tidak saling bertumpuk, dan plt.show() akan menampilkan semua visualisasi yang telah dibuat.

5. Processing Gambar

```
# =====
# BAGIAN 5: FUNGSI PREPROCESSING GAMBAR
# =====

def load_and_preprocess_image(img_path, img_size=IMG_SIZE):
    """
    Load dan preprocess gambar
    """
    try:
        # Baca gambar
        image = cv2.imread(img_path)
        if image is None:
            print(f"Warning: Tidak dapat membaca gambar {img_path}")
            return None

        # Convert BGR ke RGB
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Resize gambar
        image = cv2.resize(image, img_size)

        # Normalisasi pixel values ke [0, 1]
        image = image.astype(np.float32) / 255.0

        return image
    except Exception as e:
        print(f"Error processing {img_path}: {e}")
        return None

def create_dataset_from_dataframe(df, img_size=IMG_SIZE):
    """
    Create numpy arrays dari DataFrame
    """
    images = []
    labels = []

    print("Loading and preprocessing images...")
    for idx, row in df.iterrows():
        if idx % 500 == 0:
            print(f"Processed {idx}/{len(df)} images")

        img = load_and_preprocess_image(row['image_path'], img_size)
        if img is not None:
            images.append(img)
            labels.append(row['label'])

    print(f"Successfully loaded {len(images)} images")
    return np.array(images), np.array(labels)
```

Kode ini bertugas untuk menyiapkan data gambar dalam format yang dapat langsung digunakan oleh model deep learning, khususnya TensorFlow/Keras. Proses ini dilakukan dengan cara membaca gambar dari file, melakukan pra-pemrosesan standar, dan mengonversinya ke dalam array NumPy.

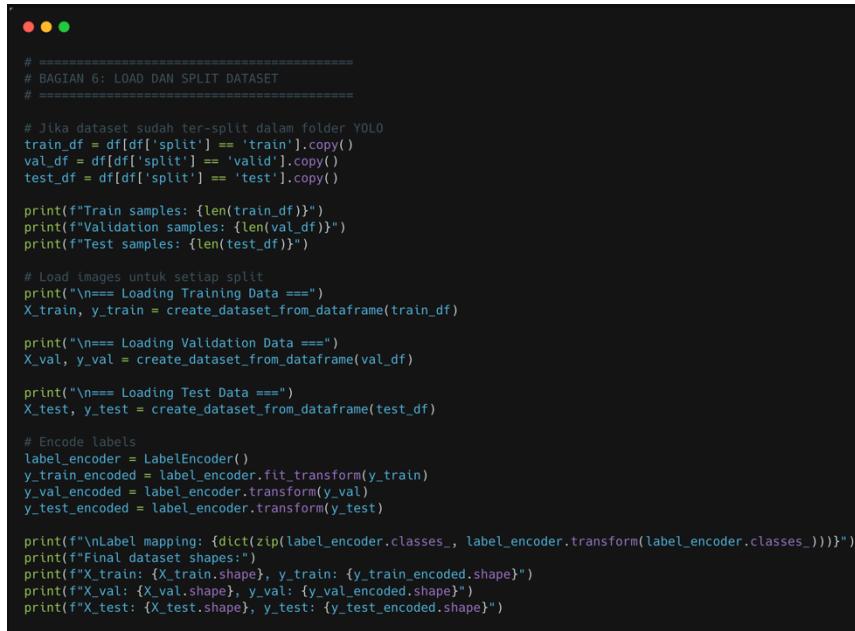
Fungsi pertama, `load_and_preprocess_image()`, menerima path ke file gambar dan ukuran target sebagai parameter. Fungsinya adalah membaca gambar menggunakan OpenCV (`cv2.imread`). Karena OpenCV membaca gambar dalam format BGR, maka gambardikonversi ke format RGB agar sesuai dengan konvensi umum dalam deep learning. Setelah itu, gambar diubah ukurannya ke dimensi tetap yang telah ditentukan (dalam hal ini, 224x224 piksel sesuai dengan `IMG_SIZE`). Terakhir, nilai piksel gambar dinormalisasi dari rentang 0–255 menjadi rentang 0–1 agar lebih stabil saat digunakan dalam neural network training.

Jika gambar gagal dibaca atau terjadi kesalahan saat pemrosesan, fungsi akan mencetak peringatan dan mengembalikan None.

Fungsi kedua, `create_dataset_from_dataframe()`, mengambil sebuah DataFrame (yang berisi daftar gambar beserta labelnya) dan mengiterasi baris demi baris. Untuk setiap baris, ia mengambil path gambar dan labelnya, lalu memanggil fungsi `load_and_preprocess_image()` untuk memuat dan mempersiapkan gambar. Hasil yang sukses dimasukkan ke dalam list `images`, dan label-nya dimasukkan ke dalam list `labels`. Fungsi ini juga mencetak progres pemrosesan setiap 500 gambar agar pengguna tahu sejauh mana proses telah berlangsung. Setelah seluruh data selesai diproses, list gambar dan label dikonversi ke array NumPy dan dikembalikan sebagai output.

6. Load dan Split Dataset

Kode dilanjutkan dengan me-load dan melakukan split terhadap dataset berdasarkan tipenya, apakah train, test, atau valid. Berikut adalah kode yang digunakan:



```
# =====#
# BAGIAN 6: LOAD DAN SPLIT DATASET
# =====#

# Jika dataset sudah ter-split dalam folder YOLO
train_df = df[df['split'] == 'train'].copy()
val_df = df[df['split'] == 'valid'].copy()
test_df = df[df['split'] == 'test'].copy()

print(f"Train samples: {len(train_df)}")
print(f"Validation samples: {len(val_df)}")
print(f"Test samples: {len(test_df)}")

# Load images untuk setiap split
print("\n==== Loading Training Data ===")
X_train, y_train = create_dataset_from_dataframe(train_df)

print("\n==== Loading Validation Data ===")
X_val, y_val = create_dataset_from_dataframe(val_df)

print("\n==== Loading Test Data ===")
X_test, y_test = create_dataset_from_dataframe(test_df)

# Encode labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_val_encoded = label_encoder.transform(y_val)
y_test_encoded = label_encoder.transform(y_test)

print(f"\nLabel mapping: {dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))}")
print("Final dataset shapes:")
print(f"X_train: {X_train.shape}, y_train: {y_train_encoded.shape}")
print(f"X_val: {X_val.shape}, y_val: {y_val_encoded.shape}")
print(f"X_test: {X_test.shape}, y_test: {y_test_encoded.shape}")
```

Kode ini bertugas untuk mempersiapkan dataset yang akan digunakan dalam pelatihan dan evaluasi model klasifikasi gambar. Proses dimulai dengan membagi DataFrame utama (df) menjadi

tiga subset berdasarkan kolom split, yaitu: train, valid, dan test. Masing-masing subset disalin menjadi DataFrame baru (train_df, val_df, test_df) untuk memastikan bahwa manipulasi selanjutnya tidak mempengaruhi data asli.

Setelah pembagian dilakukan, jumlah sampel dari masing-masing subset dicetak untuk memberikan gambaran kasar tentang proporsi data. Hal ini penting untuk memastikan bahwa tidak ada subset yang kekurangan data secara ekstrem, karena keseimbangan data sangat memengaruhi performa model.

Kemudian, gambar dari setiap subset dimuat dan diproses menggunakan fungsi `create_dataset_from_dataframe()` yang sebelumnya telah didefinisikan. Proses ini melibatkan pembacaan gambar dari file, konversi ke format RGB, resize, dan normalisasi nilai piksel. Hasilnya adalah array NumPy `X_train`, `X_val`, dan `X_test` yang berisi gambar, serta array label `y_train`, `y_val`, dan `y_test` dalam bentuk string seperti "vest" atau "no-vest".

Namun, sebelum label ini bisa digunakan dalam pelatihan model, mereka perlu dikonversi ke bentuk numerik. Untuk itu, digunakan `LabelEncoder` dari `sklearn`, yang secara otomatis memberi indeks numerik untuk setiap kelas unik. Misalnya, 'no-vest' bisa di-encode menjadi 0, dan 'vest' menjadi 1. Encoder ini dilatih pada label dari data training, lalu digunakan untuk mengubah label dari data validasi dan data uji agar konsisten.

Terakhir, kode mencetak pemetaan label (misalnya 'no-vest': 0, 'vest': 1) sebagai konfirmasi bahwa encoding berjalan sesuai harapan. Lalu, bentuk akhir dari setiap dataset (`X_train`, `y_train_encoded`, dst.) dicetak agar pengguna dapat memverifikasi dimensi array dan memastikan bahwa jumlah gambar dan labelnya cocok.

7. Visualisasi Sample Gambar

```
● ● ●

# =====
# BAGIAN 7: VISUALISASI SAMPLE GAMBAR
# =====

def plot_sample_images(X, y, label_encoder, num_samples=8):
    """
    Plot sample images dari dataset
    """
    plt.figure(figsize=(15, 8))

    for i in range(min(num_samples, len(X))):
        plt.subplot(2, 4, i+1)
        plt.imshow(X[i])
        label_name = label_encoder.inverse_transform([y[i]])[0]
        plt.title(f'Label: {label_name}')
        plt.axis('off')

    plt.tight_layout()
    plt.show()

print("Sample gambar dari training set:")
plot_sample_images(X_train, y_train_encoded, label_encoder)
```

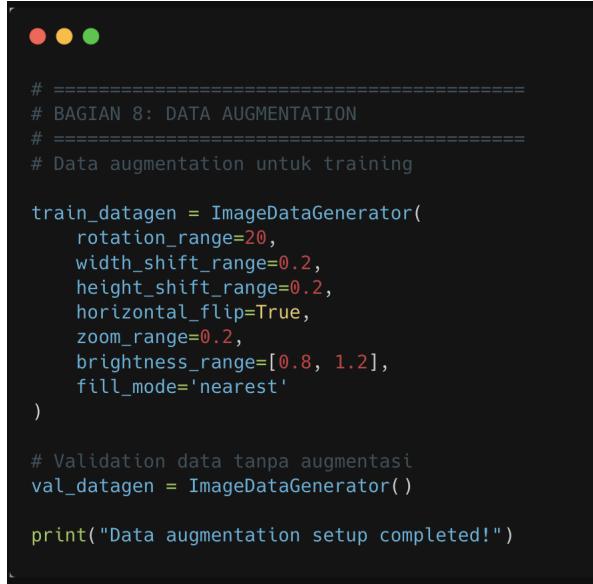
Bagian kode ini digunakan untuk menampilkan contoh-contoh gambar dari dataset pelatihan secara visual, guna memberikan pemahaman intuitif mengenai isi data yang akan digunakan untuk melatih model klasifikasi. Ini adalah langkah eksplorasi data yang sangat penting dalam machine learning, karena memungkinkan untuk melakukan verifikasi secara manual apakah gambar dan labelnya sesuai.

Fungsi `plot_sample_images()` menerima empat argumen: `X` (array gambar), `y` (array label dalam bentuk numerik), `label_encoder` (objek `LabelEncoder` yang digunakan untuk mengubah kembali label numerik menjadi string aslinya), dan `num_samples` yang menentukan berapa banyak gambar yang ingin ditampilkan (default-nya 8).

Di dalam fungsi, `matplotlib.pyplot` digunakan untuk membuat satu figure besar berukuran 15x8 inci. Fungsi melakukan iterasi sebanyak jumlah gambar yang ingin ditampilkan (maksimal sebanyak `num_samples`, tetapi tidak melebihi jumlah gambar yang tersedia). Pada setiap iterasi, fungsi mengambil gambar ke-`i` dari array `X`, menampilkannya di subplot, dan menggunakan `label_encoder.inverse_transform()` untuk mengonversi label numerik kembali ke bentuk aslinya seperti 'vest' atau 'no-vest'.

Setiap subplot diberi judul sesuai label gambar dan sumbu koordinatnya disembunyikan (`axis('off')`) agar tampilan lebih bersih. Setelah semua subplot diisi, `tight_layout()` digunakan untuk mengatur tata letak gambar agar tidak saling bertumpuk, dan `show()` memunculkan visualisasi.

8. Data Augmentasi



```
# =====
# BAGIAN 8: DATA AUGMENTATION
# =====
# Data augmentation untuk training

train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    zoom_range=0.2,
    brightness_range=[0.8, 1.2],
    fill_mode='nearest'
)

# Validation data tanpa augmentasi
val_datagen = ImageDataGenerator()

print("Data augmentation setup completed!")
```

Bagian kode ini menyiapkan proses augmentasi data, yaitu teknik yang digunakan untuk memperbesar keragaman dataset secara artifisial tanpa menambah data baru, dengan cara memodifikasi gambar-gambar yang ada. Tujuan utamanya adalah untuk membuat model lebih tahan terhadap variasi dan meningkatkan kemampuan generalisasi.

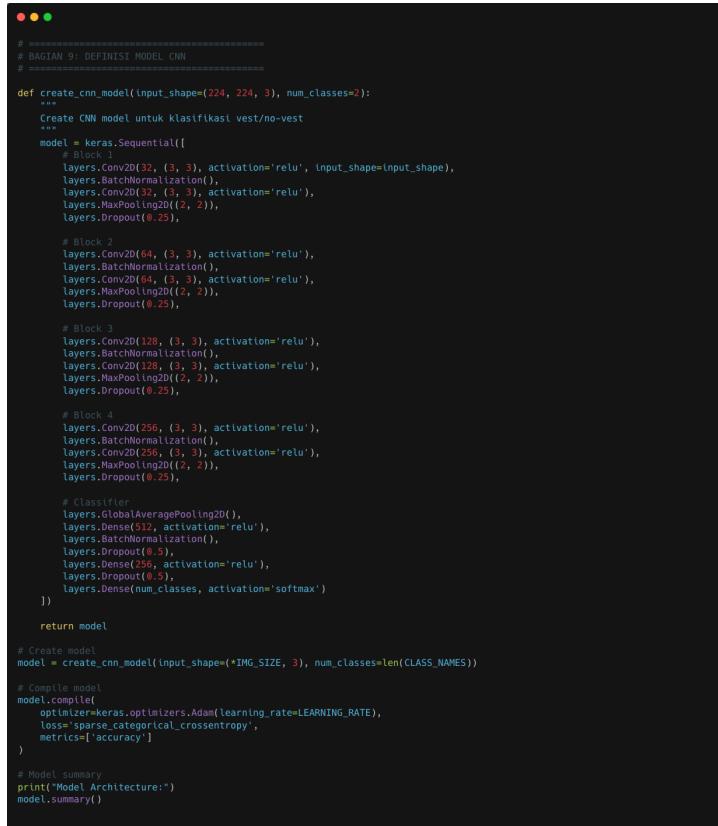
Pertama, objek `train_datagen` dibuat dari kelas `ImageDataGenerator` milik Keras, dengan sejumlah transformasi yang akan diterapkan secara acak pada gambar-gambar training selama pelatihan berlangsung:

- `rotation_range=20`: gambar akan diputar acak hingga ±20 derajat.
- `width_shift_range=0.2` dan `height_shift_range=0.2`: gambar dapat digeser secara horizontal dan vertikal hingga 20% dari ukuran gambar.

- horizontal_flip=True: gambar bisa dicerminkan secara horizontal.
- zoom_range=0.2: gambar dapat dizoom masuk atau keluar hingga 20%.
- brightness_range=[0.8, 1.2]: kecerahan gambar dapat diubah antara 80% hingga 120%.
- fill_mode='nearest': piksel kosong yang muncul karena transformasi (seperti rotasi atau geseran) akan diisi menggunakan nilai piksel terdekat.

Semua transformasi ini hanya diterapkan pada data pelatihan (train_datagen) untuk meningkatkan variasi dan mencegah overfitting. Sedangkan untuk data validasi (val_datagen), tidak diterapkan augmentasi apa pun. Ini karena data validasi digunakan untuk mengevaluasi performa model pada data yang lebih "murni", sehingga tidak boleh dimodifikasi dari bentuk aslinya.

9. Model Backbone CNN



```
# =====#
# BAGIAN 9: DEFINISI MODEL CNN
# =====#
def create_cnn_model(input_shape=(224, 224, 3), num_classes=2):
    """
    Create CNN model untuk klasifikasi vest/no-vest
    """
    model = keras.Sequential([
        # Block 1
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        layers.BatchNormalization(),
        layers.Conv2D(32, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Block 2
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Block 3
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Block 4
        layers.Conv2D(256, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.Conv2D(256, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Classifier
        layers.GlobalAveragePooling2D(),
        layers.Dense(512, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.5),
        layers.Dense(256, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])

    return model

# Create model
model = create_cnn_model(input_shape=(IMG_SIZE, 3), num_classes=len(CLASS_NAMES))

# Compile model
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=LEARNING_RATE),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Model summary
print("Model Architecture:")
model.summary()
```

Kode ini merupakan bagian penting dalam pengembangan model machine learning berbasis Convolutional Neural Network (CNN) untuk mengklasifikasikan gambar ke dalam dua kategori, yaitu "vest" (memakai rompi) dan "no-vest" (tidak memakai rompi). Fokus utama dari bagian ini adalah mendefinisikan arsitektur model, menginisialisasinya, lalu mengompilasinya agar siap untuk proses pelatihan.

Proses diawali dengan membuat fungsi `create_cnn_model()` yang bertanggung jawab untuk menyusun struktur jaringan CNN. Fungsi ini menggunakan pendekatan Sequential dari Keras, artinya lapisan-lapisan ditambahkan satu per satu secara berurutan. Model ini memiliki empat blok utama untuk ekstraksi fitur, dan satu blok akhir sebagai classifier. Setiap blok ekstraksi fitur terdiri dari dua lapis konvolusi, batch normalization untuk menjaga kestabilan distribusi aktivasi, pooling untuk mereduksi dimensi, dan dropout untuk mencegah overfitting. Seiring bertambahnya kedalaman jaringan, jumlah filter konvolusi meningkat, memungkinkan model belajar dari pola-pola visual yang semakin kompleks.

Setelah fitur diekstraksi dari gambar, data diteruskan ke lapisan classifier. Di sini, fitur spasial diringkas menggunakan global average pooling, kemudian dilanjutkan ke beberapa lapisan dense dengan ukuran besar (512 dan 256 neuron) yang juga dilengkapi batch normalization dan dropout untuk menjaga kestabilan dan menghindari overfitting. Lapisan terakhir adalah dense layer dengan aktivasi softmax yang menghasilkan dua output, masing-masing mewakili probabilitas untuk kelas "vest" dan "no-vest".

Setelah model dibuat, langkah selanjutnya adalah mengompilasinya. Optimizer yang digunakan adalah Adam, yang terkenal efisien dalam mengatur pembaruan bobot, dan fungsi loss yang digunakan adalah `sparse_categorical_crossentropy`, karena

label dikodekan sebagai angka (bukan vektor one-hot). Metrik yang digunakan untuk menilai kinerja model adalah akurasi.

Akhir dari bagian ini adalah pemanggilan `model.summary()`, yang mencetak ringkasan arsitektur jaringan—menampilkan nama-nama layer, output shape dari masing-masing layer, serta jumlah parameter yang dapat dilatih. Ini memberikan gambaran menyeluruh tentang kompleksitas dan ukuran model yang telah dibangun.

10. Callback dan Persiapan Training

Kode ini mempersiapkan proses pelatihan model CNN dengan cara mendefinisikan callbacks, yaitu mekanisme otomatis yang akan dijalankan selama pelatihan untuk mengatur atau menyimpan model berdasarkan kinerja saat validasi. Tujuan dari penggunaan callbacks adalah untuk meningkatkan efisiensi pelatihan dan memastikan model yang disimpan adalah model dengan performa terbaik pada data validasi. Berikut adalah kodennya:

```
# =====
# BAGIAN 10: CALLBACKS DAN TRAINING SETUP
# =====

# Callbacks
callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True,
        verbose=1
    ),
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=5,
        min_lr=1e-7,
        verbose=1
    ),
    ModelCheckpoint(
        '../build/best_vest_model.h5',
        monitor='val_accuracy',
        save_best_only=True,
        verbose=1
    )
]

print("Callbacks setup completed!")
print("Ready for training!")
```

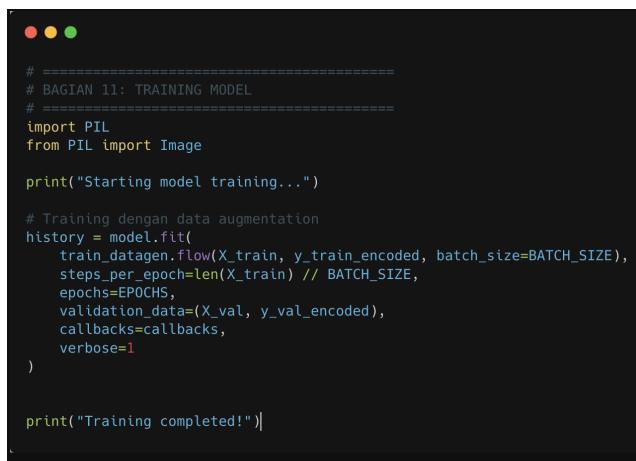
Dalam kode di atas terdapat tiga jenis callback yang digunakan yakni:

1. EarlyStopping digunakan untuk menghentikan pelatihan lebih awal jika model berhenti mengalami peningkatan. Dalam hal ini, proses pelatihan akan berhenti jika nilai val_loss (loss pada data validasi) tidak membaik selama 10 epoch berturut-turut. Opsi restore_best_weights=True memastikan bahwa bobot model yang dikembalikan adalah bobot terbaik sebelum performa mulai menurun, bukan bobot dari epoch terakhir.
2. ReduceLROnPlateau dipakai untuk menyesuaikan laju pembelajaran (learning rate) selama pelatihan. Jika val_loss tidak membaik dalam 5 epoch, maka learning rate akan dikurangi setengahnya (factor=0.5). Ini membantu model untuk beradaptasi lebih hati-hati saat mendekati konvergensi. Learning rate tidak akan diturunkan di bawah nilai minimum

(min_lr=1e-7), untuk mencegah perubahan yang terlalu kecil hingga tak berdampak.

3. ModelCheckpoint akan menyimpan bobot model ke file best_vest_model.h5 hanya jika model pada epoch tersebut memiliki akurasi validasi (val_accuracy) terbaik sejauh ini. Ini menjamin bahwa setelah pelatihan selesai, pengguna akan memiliki salinan dari model dengan performa validasi terbaik, bukan hanya model dari epoch terakhir.

11. Training Model



```
# =====#
# BAGIAN 11: TRAINING MODEL
# =====#
import PIL
from PIL import Image

print("Starting model training...")

# Training dengan data augmentation
history = model.fit(
    train_datagen.flow(X_train, y_train_encoded, batch_size=BATCH_SIZE),
    steps_per_epoch=len(X_train) // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=(X_val, y_val_encoded),
    callbacks=callbacks,
    verbose=1
)

print("Training completed!")
```

Bagian kode ini menandai dimulainya proses pelatihan model CNN yang telah didefinisikan sebelumnya. Inti dari bagian ini adalah pemanggilan fungsi `model.fit()`, yaitu fungsi utama dalam Keras untuk melakukan pelatihan. Proses pelatihan ini dilakukan dengan menggunakan data pelatihan (`X_train`, `y_train_encoded`) yang sebelumnya telah di-preprocess, dan dibungkus menggunakan `train_datagen.flow(...)`. Ini berarti model akan menerima batch data pelatihan secara bertahap yang telah diproses secara real-time dengan augmentasi gambar seperti rotasi, flipping, zoom, dan lainnya. Hal ini bertujuan untuk membuat model lebih general terhadap variasi visual dan mencegah overfitting.

Parameter `steps_per_epoch` ditentukan sebagai hasil bagi jumlah total gambar pelatihan dibagi dengan ukuran batch. Ini

menunjukkan berapa banyak batch yang akan diproses di setiap epoch.

Model dilatih selama sejumlah epoch yang sudah ditentukan sebelumnya yakni 35 (EPOCHS), dan pada setiap akhir epoch, kinerja model dievaluasi menggunakan data validasi (X_{val} , $y_{\text{val_encoded}}$) yang tidak melalui augmentasi.

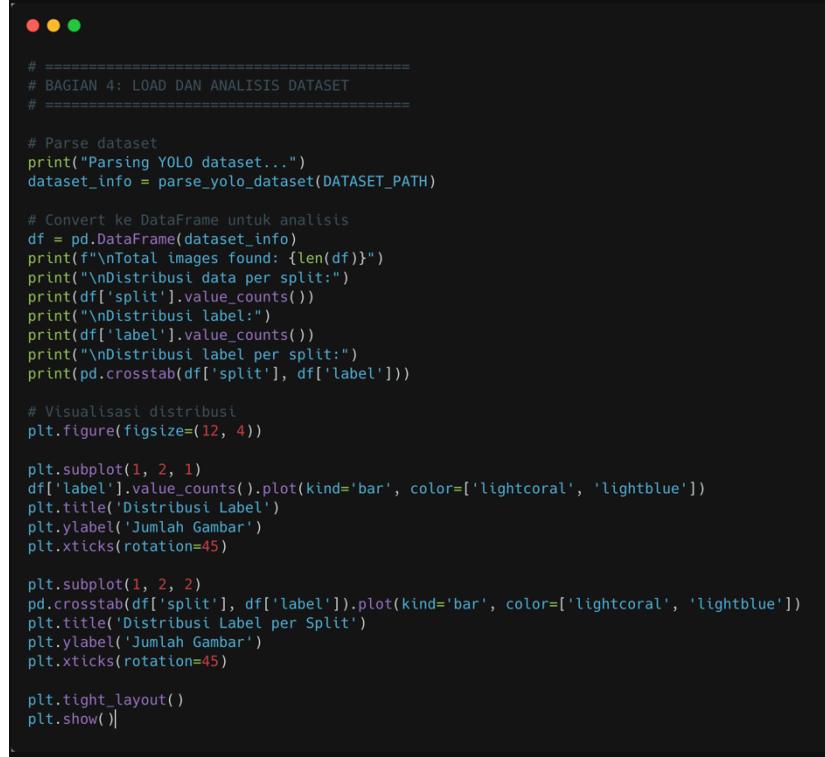
Selama proses ini, callback yang telah dikonfigurasi sebelumnya ikut dijalankan. Ini mencakup penghentian pelatihan otomatis jika model mulai overfitting (EarlyStopping), penyesuaian learning rate secara dinamis (ReduceLROnPlateau), serta penyimpanan model terbaik berdasarkan akurasi validasi (ModelCheckpoint).

Semua aktivitas pelatihan ditampilkan ke layar secara real-time (verbose=1), menampilkan metrik seperti loss dan akurasi untuk setiap epoch. Setelah seluruh proses selesai, sistem mencetak pesan bahwa pelatihan telah selesai, menandakan model siap untuk evaluasi atau penggunaan lebih lanjut. Bagian ini merupakan salah satu tahap terpenting dalam keseluruhan pipeline karena hasil dari proses ini adalah model yang telah belajar mengenali perbedaan visual antara gambar dengan rompi dan tanpa rompi.

12. Visualisasi Hasil Training

Kode ini digunakan untuk memvisualisasikan performa model selama proses pelatihan dengan cara memplot metrik akurasi dan loss dari setiap epoch untuk data pelatihan dan validasi. Tujuannya adalah memberikan gambaran visual tentang bagaimana model belajar seiring waktu, apakah proses pelatihan

berjalan stabil, mengalami overfitting, atau underfitting.



```
# =====#
# BAGIAN 4: LOAD DAN ANALISIS DATASET
# =====#
# Parse dataset
print("Parsing YOLO dataset...")
dataset_info = parse_yolo_dataset(DATASET_PATH)

# Convert ke DataFrame untuk analisis
df = pd.DataFrame(dataset_info)
print(f"\nTotal images found: {len(df)}")
print("\nDistribusi data per split:")
print(df['split'].value_counts())
print("\nDistribusi label:")
print(df['label'].value_counts())
print("\nDistribusi label per split:")
print(pd.crosstab(df['split'], df['label']))

# Visualisasi distribusi
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
df['label'].value_counts().plot(kind='bar', color=['lightcoral', 'lightblue'])
plt.title('Distribusi Label')
plt.ylabel('Jumlah Gambar')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
pd.crosstab(df['split'], df['label']).plot(kind='bar', color=['lightcoral', 'lightblue'])
plt.title('Distribusi Label per Split')
plt.ylabel('Jumlah Gambar')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```

Fungsi `plot_training_history()` menerima objek history yang dihasilkan oleh metode `model.fit()` pada tahap sebelumnya. Objek ini menyimpan log performa dari setiap epoch, seperti nilai akurasi (`accuracy`, `val_accuracy`) dan loss (`loss`, `val_loss`) pada data pelatihan dan validasi. Visualisasi dilakukan menggunakan dua subplot berdampingan:

1. Subplot pertama menampilkan grafik akurasi model

- Garis biru menggambarkan peningkatan akurasi pada data pelatihan dari epoch ke epoch.
- Garis oranye (atau warna lainnya tergantung setting) menunjukkan perubahan akurasi pada data validasi.

Grafik ini membantu melihat apakah model belajar dengan baik (akurasi naik), dan apakah validasi mengikuti tren pelatihan (tanda generalisasi yang baik).

2. Subplot kedua memperlihatkan nilai loss

- Menunjukkan seberapa baik model meminimalkan kesalahan prediksi.

- Jika loss pada data validasi mulai meningkat sementara loss pelatihan terus menurun, itu pertanda model mulai overfitting.

Setiap subplot diberi label, judul, dan legenda agar mudah dibaca. Fungsi plt.tight_layout() digunakan agar tampilan antar subplot tidak saling bertumpuk, dan plt.show() akan menampilkan plot ke layar.

13. Evaluasi Model

```
# =====
# BAGIAN 13: EVALUASI MODEL
# =====

# Load best model
model.load_weights('best_vest_model.h5')

# Prediksi pada test set
print("Evaluating on test set...")
test_loss, test_accuracy = model.evaluate(X_test, y_test_encoded, verbose=0)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")

# Prediksi detail
y_pred_proba = model.predict(X_test)
y_pred = np.argmax(y_pred_proba, axis=1)

# Classification report
print("\nClassification Report:")
print(classification_report(y_test_encoded, y_pred,
                           target_names=label_encoder.classes_))

# Confusion Matrix
cm = confusion_matrix(y_test_encoded, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

Kode ini digunakan untuk mengevaluasi performa akhir model CNN pada test set, setelah pelatihan selesai dan model terbaik telah disimpan. Proses evaluasi ini melibatkan beberapa langkah penting:

Pertama, model memuat kembali bobot terbaik yang telah disimpan ke file best_vest_model.h5 selama proses pelatihan. File ini berisi versi model yang memiliki akurasi validasi terbaik berdasarkan callback ModelCheckpoint. Ini memastikan evaluasi

dilakukan terhadap model yang paling optimal, bukan hanya model hasil akhir pelatihan.

Setelah itu, model digunakan untuk mengevaluasi performanya terhadap data uji (`X_test` dan `y_test_encoded`). Fungsi `model.evaluate()` menghitung dua metrik utama:

- Loss: Seberapa besar kesalahan prediksi model terhadap data uji.
- Accuracy: Persentase prediksi yang benar.

Setelah evaluasi global, model digunakan untuk melakukan prediksi probabilistik terhadap seluruh data uji. Output dari `model.predict()` adalah probabilitas kelas untuk setiap gambar. Probabilitas ini kemudian dikonversi ke label prediksi final dengan `np.argmax`, yaitu memilih kelas dengan probabilitas tertinggi untuk masing-masing gambar.

Selanjutnya, `classification_report()` dari `sklearn` digunakan untuk mencetak laporan evaluasi yang lebih detail. Laporan ini mencakup:

- Precision: Ketepatan prediksi per kelas.
- Recall: Kemampuan model mendekripsi kelas tertentu.
- F1-Score: Rata-rata harmonis dari precision dan recall.
- Support: Jumlah sampel aktual untuk setiap kelas.

Akhirnya, confusion matrix dibuat menggunakan `confusion_matrix()` untuk melihat secara visual berapa banyak prediksi model yang benar dan salah untuk masing-masing kelas. Hasil ini divisualisasikan dalam bentuk heatmap menggunakan `seaborn.heatmap`, dengan sumbu Y mewakili label sebenarnya dan sumbu X mewakili label prediksi.

14. Infering Machine Learning

Bagian terakhir dari skrip ini berfokus pada pengujian model CNN yang telah dilatih terhadap gambar baru dan memberikan visualisasi hasil prediksi, khususnya untuk keperluan verifikasi dan demonstrasi.

```

# =====#
# BAGIAN 14: FUNGSI PREDIKSI UNTUK GAMBAR BARU
# =====#
def predict_single_image(model, img_path, label_encoder, img_size=IMG_SIZE):
    """
    Prediksi untuk single image
    """
    # Load dan preprocess gambar
    img = load_and_preprocess_image(img_path, img_size)
    if img is None:
        return None, None

    # Expand dimensions untuk batch
    img_batch = np.expand_dims(img, axis=0)

    # Prediksi
    pred_proba = model.predict(img_batch, verbose=0)
    pred_class = np.argmax(pred_proba, axis=1)[0]
    confidence = np.max(pred_proba)

    # Convert ke label name
    pred_label = label_encoder.inverse_transform([pred_class])[0]

    return pred_label, confidence

# Contoh penggunaan
def test_prediction_on_samples():
    """
    Test prediksi pada beberapa sample dari test set
    """
    plt.figure(figsize=(15, 10))

    # Ambil 8 sample random dari test set
    indices = np.random.choice(len(X_test), 8, replace=False)

    for i, idx in enumerate(indices):
        plt.subplot(2, 4, i+1)

        # Show image
        plt.imshow(X_test[idx])

        # Prediksi
        img_batch = np.expand_dims(X_test[idx], axis=0)
        pred_proba = model.predict(img_batch, verbose=0)
        pred_class = np.argmax(pred_proba, axis=1)[0]
        confidence = np.max(pred_proba)

        # True dan predicted labels
        true_label = label_encoder.inverse_transform([y_test_encoded[idx]])[0]
        pred_label = label_encoder.inverse_transform([pred_class])[0]

        # Color coding
        color = 'green' if true_label == pred_label else 'red'

        plt.title(f'True: {true_label}\nPred: {pred_label}\nConf: {confidence:.3f}', color=color, fontsize=10)
        plt.axis('off')

    plt.tight_layout()
    plt.show()

print("Testing predictions on sample images:")
test_prediction_on_samples()

print("\n" + "="*50)
print("CNN VEST CLASSIFICATION COMPLETED!")
print("="*50)
print(f"Final Test Accuracy: {test_accuracy:.4f}")
print("Model saved as: best_vest_model.h5")

```

Fungsi `predict_single_image` merupakan sebuah pipeline prediksi untuk satu gambar saja. Ia memuat dan melakukan praproses gambar menggunakan fungsi `load_and_preprocess_image` yang sebelumnya didefinisikan, lalu menyesuaikan bentuk input menjadi batch (karena model memproses dalam bentuk batch). Gambar kemudian dikirim ke model untuk mendapatkan probabilitas prediksi untuk masing-masing kelas. Dari sini, prediksi akhir ditentukan berdasarkan probabilitas tertinggi (`argmax`), dan tingkat keyakinan model terhadap prediksi tersebut (`max` dari probabilitas). Label numerik hasil prediksi dikonversi kembali menjadi nama kelas menggunakan `label_encoder`.

Setelah fungsi prediksi untuk satu gambar selesai, digunakan fungsi `test_prediction_on_samples` untuk mendemonstrasikan kinerja model secara visual pada data uji (test set). Fungsi ini memilih delapan gambar acak dari test set, kemudian memvisualisasikan masing-masing gambar beserta label aslinya, label prediksi dari model, dan tingkat keyakinan model terhadap prediksinya. Perbedaan antara prediksi dan label sebenarnya ditandai dengan warna: hijau jika prediksinya benar, merah jika salah.

Visualisasi ini memberi gambaran intuitif apakah model benar-benar memahami pola visual yang membedakan antara pekerja yang memakai vest dan yang tidak. Dengan menampilkan keyakinan model terhadap prediksinya (confidence score), pengguna juga bisa menilai seberapa percaya diri model terhadap hasilnya.

D. Output

a. Bagian 4 – Load dan Analisis Dataset

```
Parsing YOLO dataset...
Processing train split...
Processing valid split...
Processing test split...

Total images found: 2503

Distribusi data per split:
split
train    1841
valid     531
test      131
Name: count, dtype: int64

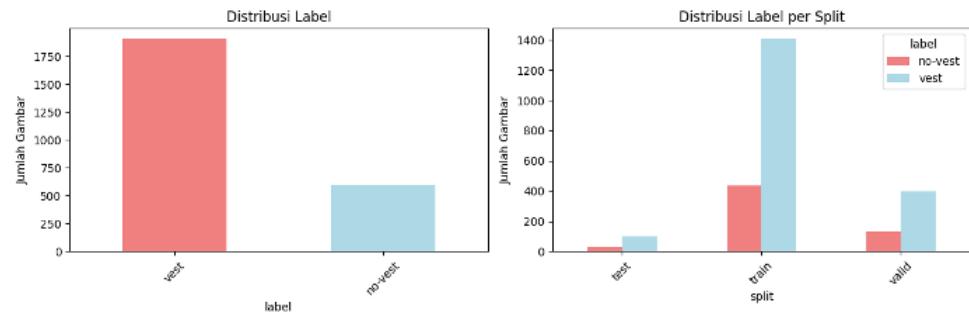
Distribusi label:
label
vest      1904
no-vest   599
Name: count, dtype: int64

Distribusi label per split:
label  no-vest  vest
split
test      29    102
train     436   1405
valid     134   397
```

Hasil parsing dataset YOLO ini menunjukkan bahwa terdapat total 2.503 gambar yang berhasil dibaca dan diproses, kemudian dibagi ke dalam tiga bagian: train, validasi, dan test. Sebagian besar data, yaitu 1.841 gambar, digunakan untuk pelatihan. Sementara itu, 531 gambar digunakan untuk validasi dan 131 gambar sisanya untuk pengujian. Ini adalah pembagian yang umum, di mana sebagian besar data difokuskan untuk melatih model, dan sisanya untuk mengevaluasi performa secara objektif.

Dalam hal anotasi objek, dataset ini mengandung dua jenis label: vest dan no-vest. Sebaran labelnya menunjukkan bahwa kelas vest mendominasi dengan jumlah total 1.904 anotasi, sementara no-vest hanya sebanyak 599. Ini mengindikasikan adanya ketidakseimbangan kelas, di mana objek dengan rompi keselamatan jauh lebih banyak daripada yang tidak mengenakannya. Ketimpangan ini perlu menjadi perhatian, karena dapat mempengaruhi cara model belajar dan menimbulkan bias terhadap kelas mayoritas.

Lebih lanjut, jika dilihat dari distribusi label dalam masing-masing bagian dataset, pola ini juga konsisten: pada data pelatihan, terdapat 1.405 objek vest dan 436 no-vest; di data validasi, 397 vest dan 134 no-vest; sedangkan pada data pengujian, 102 vest dan hanya 29 no-vest. Dengan kata lain, di setiap split, kelas vest tetap lebih dominan. Ini menunjukkan bahwa ketidakseimbangan kelas tidak hanya terjadi secara keseluruhan, tetapi juga merata dalam setiap subset data yang digunakan.



Selain dalam bentuk teks, dataset juga divisualisasikan dalam bentuk grafik agar lebih mudah untuk dibaca dan dipahami.

b. Bagian 6 – Load dan Split Dataset

```
Train samples: 1841
Validation samples: 531
Test samples: 131

==== Loading Training Data ====
Loading and preprocessing images...
Processed 0/1841 images
Processed 500/1841 images
Processed 1000/1841 images
Processed 1500/1841 images
Successfully loaded 1841 images

==== Loading Validation Data ====
Loading and preprocessing images...
Processed 2000/531 images
Successfully loaded 531 images

==== Loading Test Data ====
Loading and preprocessing images...
Processed 2500/131 images
Successfully loaded 131 images

Label mapping: {np.str_('no-vest'): np.int64(0), np.str_('vest'): np.int64(1)}
Final dataset shapes:
X_train: (1841, 224, 224, 3), y_train: (1841,)
X_val: (531, 224, 224, 3), y_val: (531,)
X_test: (131, 224, 224, 3), y_test: (131,)|
```

Bagian ini menunjukkan proses pemuatan dan preprocessing dataset citra untuk keperluan pelatihan model, di mana data gambar dari tiga subset, yakni train, validasi, dan test yang dimuat satu per satu ke dalam memori. Jumlah gambar yang berhasil dimuat sesuai dengan yang sebelumnya terdeteksi saat parsing, yaitu 1.841 untuk pelatihan, 531 untuk validasi, dan 131 untuk pengujian. Setiap gambar diproses terlebih dahulu, kemungkinan besar melalui resize ke dimensi tetap (dalam hal ini 224×224 piksel dengan 3 channel warna), sebelum dimasukkan ke dalam array numpy yang digunakan oleh model.

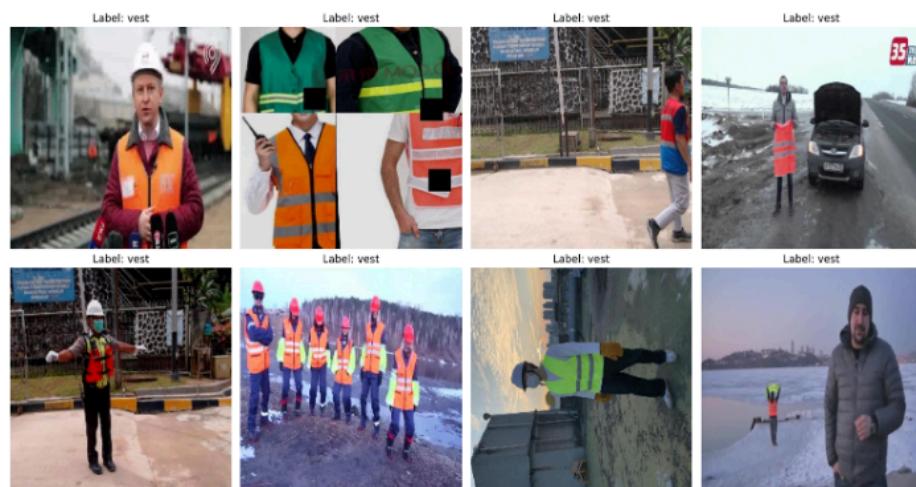
Selama proses pemuatan ini, ditampilkan juga informasi progres pemrosesan dalam blok-blok tertentu, seperti setiap 500 gambar, untuk memberi umpan balik visual bahwa sistem sedang bekerja dan tidak macet.

Di akhir proses ini, label dari kelas objek juga telah diubah ke dalam bentuk numerik yang siap digunakan oleh model pembelajaran mesin. Kelas no-vest diberikan label 0 dan vest

diberi label 1. Pemetaan ini penting untuk memastikan bahwa model mengenali masing-masing kelas sebagai nilai diskrit yang dapat dipelajari.

Hasil akhirnya adalah tiga pasang dataset dalam bentuk array: gambar (X) dan label (y) untuk masing-masing split. Dimensi dari array ini mengindikasikan bahwa seluruh gambar telah dinormalisasi ke ukuran 224 piksel per sisi dengan 3 kanal warna, dan label dikonversi menjadi vektor satu dimensi yang menyatakan kelas setiap gambar. Ini menandai bahwa proses pemuatan data berhasil dan dataset telah sepenuhnya siap untuk dilatih menggunakan arsitektur deep learning seperti CNN.

c. Bagian 7 – Visualisasi Sample Gambar



d. Bagian 8 – Model CNN



The screenshot shows a terminal window with a table of model architecture details. The table has three columns: Layer (type), Output Shape, and Param #.

| Layer (type) | Output Shape | Param # |
|---|----------------------|---------|
| conv2d (Conv2D) | (None, 222, 222, 32) | 896 |
| batch_normalization (BatchNormalization) | (None, 222, 222, 32) | 128 |
| conv2d_1 (Conv2D) | (None, 220, 220, 32) | 9,248 |
| max_pooling2d (MaxPooling2D) | (None, 110, 110, 32) | 0 |
| dropout (Dropout) | (None, 110, 110, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 108, 108, 64) | 18,496 |
| batch_normalization_1 (BatchNormalization) | (None, 108, 108, 64) | 256 |
| conv2d_3 (Conv2D) | (None, 106, 106, 64) | 36,928 |
| max_pooling2d_1 (MaxPooling2D) | (None, 53, 53, 64) | 0 |
| dropout_1 (Dropout) | (None, 53, 53, 64) | 0 |
| conv2d_4 (Conv2D) | (None, 51, 51, 128) | 73,856 |
| batch_normalization_2 (BatchNormalization) | (None, 51, 51, 128) | 512 |
| conv2d_5 (Conv2D) | (None, 49, 49, 128) | 147,584 |
| max_pooling2d_2 (MaxPooling2D) | (None, 24, 24, 128) | 0 |
| dropout_2 (Dropout) | (None, 24, 24, 128) | 0 |
| conv2d_6 (Conv2D) | (None, 22, 22, 256) | 295,168 |
| batch_normalization_3 (BatchNormalization) | (None, 22, 22, 256) | 1,024 |
| conv2d_7 (Conv2D) | (None, 20, 20, 256) | 590,080 |
| max_pooling2d_3 (MaxPooling2D) | (None, 10, 10, 256) | 0 |
| dropout_3 (Dropout) | (None, 10, 10, 256) | 0 |
| global_average_pooling2d (GlobalAveragePooling2D) | (None, 256) | 0 |
| dense (Dense) | (None, 512) | 131,584 |
| batch_normalization_4 (BatchNormalization) | (None, 512) | 2,048 |
| dropout_4 (Dropout) | (None, 512) | 0 |
| dense_1 (Dense) | (None, 256) | 131,328 |
| dropout_5 (Dropout) | (None, 256) | 0 |
| dense_2 (Dense) | (None, 2) | 514 |

Arsitektur model di atas terdiri dari beberapa blok konvolusi, normalisasi, pooling, dropout, dan diakhiri dengan lapisan dense untuk klasifikasi dua kelas (vest dan no-vest).

Model dimulai dengan serangkaian lapisan konvolusi (Conv2D) yang menangkap fitur spasial dari gambar input, dengan ukuran awal 224x224x3. Setiap blok konvolusi diikuti oleh BatchNormalization, yang membantu mempercepat dan menstabilkan proses pelatihan, serta disusul oleh MaxPooling2D yang mengurangi dimensi spasial (height dan width) sambil mempertahankan informasi penting.

Di antara beberapa blok juga disisipkan Dropout, sebuah teknik regularisasi yang membantu mencegah overfitting dengan "menonaktifkan" sebagian neuron secara acak selama pelatihan. Semakin dalam ke dalam jaringan, jumlah filter bertambah (dari 32, 64, 128, hingga 256), menunjukkan bahwa model menangkap fitur yang semakin kompleks di setiap tahap.

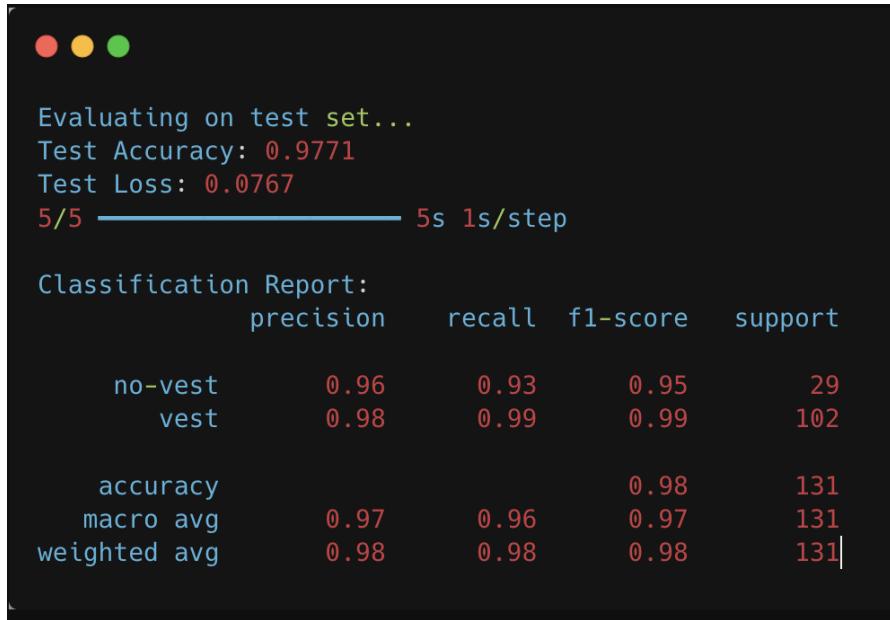
Setelah semua fitur diekstraksi, dilakukan Global Average Pooling, yang merangkum fitur spasial menjadi satu vektor fitur per channel, ini efisien dan membantu mencegah overfitting karena tidak terlalu banyak parameter seperti halnya Flatten.

Vektor hasil pooling kemudian dilewatkan ke beberapa lapisan dense (fully connected). Lapisan dense pertama memiliki 512 neuron, lalu diikuti batch normalization dan dropout, kemudian dilanjutkan ke dense dengan 256 neuron, dan akhirnya ke dense terakhir berisi 2 neuron, sesuai jumlah kelas target, yang menghasilkan probabilitas untuk masing-masing kelas.

Secara total, model ini memiliki lebih dari 1,4 juta parameter yang dapat dilatih. Ini cukup besar untuk sebuah tugas klasifikasi biner, dan menunjukkan bahwa model memiliki kapasitas tinggi untuk menangkap kompleksitas data, asalkan

didukung jumlah data pelatihan yang cukup dan augmentasi yang baik.

e. Evaluasi Model



```
Evaluating on test set...
Test Accuracy: 0.9771
Test Loss: 0.0767
5/5 ━━━━━━━━ 5s 1s/step

Classification Report:
precision    recall   f1-score   support
no-vest      0.96     0.93     0.95      29
vest         0.98     0.99     0.99     102
accuracy           0.98
macro avg       0.97     0.96     0.97     131
weighted avg    0.98     0.98     0.98     131
```

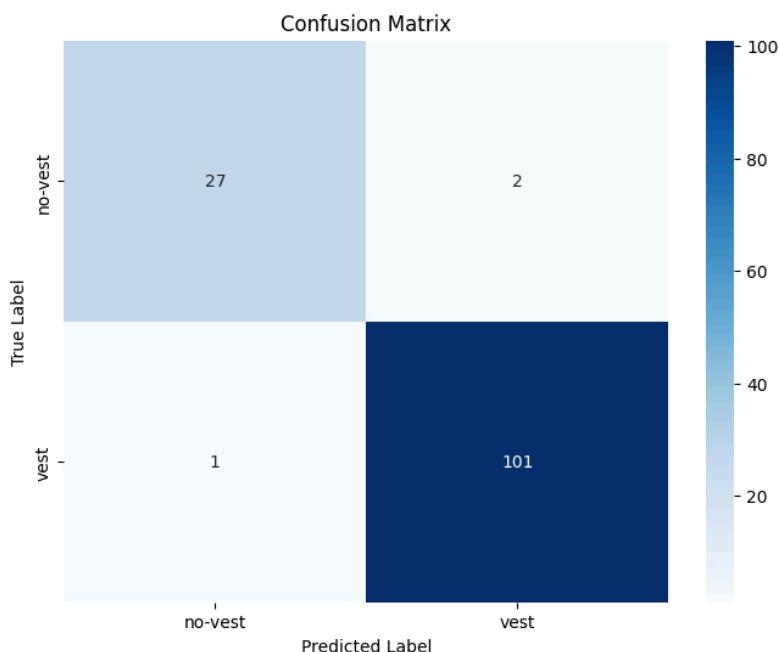
Model CNN yang dilatih menunjukkan performa yang sangat tinggi ketika diuji pada data yang belum pernah dilihat sebelumnya. Akurasi keseluruhan model pada test set mencapai sekitar 97.7%, yang berarti sebagian besar prediksi model sesuai dengan label sebenarnya. Angka ini mencerminkan bahwa model tidak hanya mampu menghafal pola dari data pelatihan, tetapi juga mampu menggeneralisasi dengan baik pada data baru.

Jika kita melihat lebih dalam pada hasil evaluasi melalui classification report, kita dapat memahami bagaimana model menangani masing-masing kelas. Untuk kelas vest, yang jumlah datanya lebih banyak dalam test set, model menunjukkan performa hampir sempurna. Precision-nya sangat tinggi, yaitu 98%, yang berarti ketika model memprediksi vest, prediksinya hampir selalu benar. Recall-nya bahkan lebih tinggi lagi, 99%, menunjukkan bahwa model berhasil menemukan hampir semua gambar vest dalam test set. F1-score yang mendekati 1 (0.99) memperkuat bahwa keseimbangan antara presisi dan recall dalam menangani

kelas ini sangat baik.

Sementara untuk kelas no-vest, meskipun jumlah sampelnya jauh lebih sedikit, performa model tetap kuat. Precision-nya adalah 96%, artinya model jarang salah mengira gambar lain sebagai no-vest. Namun recall-nya sedikit lebih rendah, yaitu 93%, menunjukkan ada beberapa gambar no-vest yang tidak dikenali dengan benar oleh model dan justru diklasifikasikan sebagai vest. Meskipun demikian, F1-score sebesar 0.95 tetap menunjukkan bahwa model menangani kelas ini dengan sangat layak.

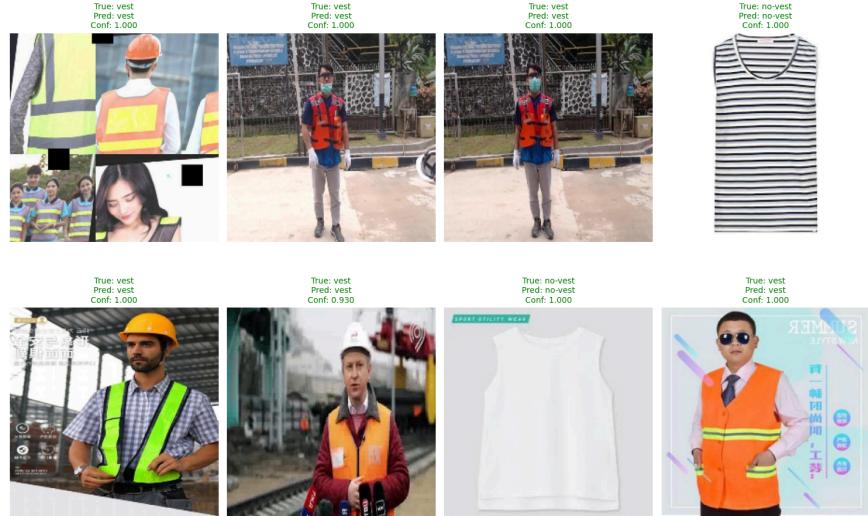
Rata-rata metrik yang dihitung secara makro (menganggap setiap kelas memiliki bobot yang sama) menunjukkan nilai sekitar 97% untuk presisi, recall, dan F1-score, sedangkan metrik berbobot (weighted average) yang memperhitungkan jumlah sampel per kelas juga mendekati 98%. Ini mengindikasikan bahwa model tidak hanya akurat, tetapi juga adil dan seimbang dalam menangani distribusi kelas yang tidak sama.



Evaluasi model juga ditampilkan dalam bentuk confusion matrix yang dimana menampilkan hasil dari True-Negative

dan False-Negative yang dapat digunakan untuk menilai seberapa sering model memprediksi yang benar dan salah.

f. Hasil Prediksi Gambar Baru



Gambar di atas merupakan hasil prediksi dari gambar yang belum dikenali. Dapat dilihat pada gambar di atas bahwa terdapat label dan hasil confidence yang dimana rata-rata nilai confidence nya cukup tinggi serta model dapat dengan baik mengenali objek pada sebuah gambar apakah memiliki vest atau tidak ada vest.

BAB III

KESIMPULAN

Berdasarkan hasil dari proyek ini, dapat disimpulkan bahwa sistem deteksi rompi keselamatan (vest) yang dibuat dengan algoritma CNN sudah berhasil berjalan dengan baik. Sistem ini mampu mengenali apakah seseorang memakai rompi atau tidak, dengan tingkat akurasi yang cukup tinggi, yaitu sekitar 97,7% saat diuji. Data latih yang digunakan berasal dari Roboflow, dan hanya fokus pada dua jenis gambar, yaitu yang memakai rompi dan yang tidak. Sistem ini juga sudah dibuat dalam bentuk aplikasi web, sehingga bisa digunakan secara langsung oleh perusahaan, khususnya untuk membantu bagian Kesehatan dan Keselamatan Kerja (EHS) dalam mengawasi para pekerja. Kehadiran sistem ini bisa sangat membantu, apalagi di area kerja yang luas atau jika jumlah petugas pengawas terbatas. Walau begitu, sistem ini masih terbatas hanya untuk rompi saja, jadi ke depannya akan lebih baik jika bisa mendeteksi alat pelindung lainnya juga seperti helm atau masker, agar pengawasan lebih lengkap dan akurat.