

## **TUGAS PERTEMUAN 10**

### **SISTEM CERDAS**

*“Deteksi Jenis Kendaraan dan Simulasi Sistem Parkir Realistik Menggunakan Klasifikasi Citra Berbasis Deep Learning”*

Dosen Pengampu:

Taufik Ridwan, S.Kom., M.Kom.



Disusun oleh:

Fitri Asri Nur Fatimah	(2210631250052)
Nurul Hidayati	(2210631250025)
Muhammad Fadllan Ziadh Ramadhan	(2010631250066)

**PROGRAM STUDI SISTEM INFORMASI**

**FAKULTAS ILMU KOMPUTER**

**UNIVERSITAS SINGAPERBANGSA KARAWANG**

**2025**

# **TUGAS PERTEMUAN 10**

## **A. TEORI TERKAIT**

### **1. DEEP LEARNING**

Deep learning adalah jenis machine learning yang menggunakan jaringan neural buatan untuk belajar dari data. Jaringan neural buatan terinspirasi oleh otak manusia, dan dapat digunakan untuk memecahkan berbagai masalah, termasuk pengenalan citra, natural language processing, dan pengenalan ucapan.

#### **a. Cara Kerja Deep Learning**

Deep learning bekerja menggunakan jaringan neural buatan untuk belajar dari data. Jaringan neural terdiri dari lapisan node yang saling berhubungan, dan setiap node bertanggung jawab untuk mempelajari fitur data tertentu. Berdasarkan contoh sebelumnya terkait gambar – dalam jaringan pengenalan citra, lapisan pertama node mungkin belajar mengidentifikasi tepi, lapisan kedua mungkin belajar mengidentifikasi bentuk, dan lapisan ketiga mungkin belajar mengidentifikasi objek.

Saat jaringan belajar, bobot pada koneksi antara node disesuaikan sehingga jaringan dapat mengklasifikasikan data dengan lebih baik. Proses ini disebut pelatihan, dan dapat dilakukan menggunakan berbagai teknik, seperti supervised learning, unsupervised learning, dan reinforcement learning. Setelah dilatih, jaringan neural dapat digunakan untuk membuat prediksi dengan data baru yang diterima.

#### **b. Aplikasi Deep Learning**

Deep learning dapat digunakan di berbagai aplikasi, termasuk:

- Pengenalan citra: Untuk mengidentifikasi objek dan fitur dalam gambar, seperti orang, hewan, tempat, dll.
- Natural language processing: Untuk membantu memahami makna teks, seperti dalam chatbot layanan pelanggan dan filter spam.
- Keuangan: Untuk membantu menganalisis data keuangan dan membuat prediksi tentang tren pasar

- Teks ke gambar: Mengonversi teks menjadi gambar, seperti di aplikasi Google Terjemahan.

### c. Jenis-Jenis Deep Learning

Ada banyak jenis model deep learning. Beberapa jenis yang paling umum meliputi:

- Convolutional Neural Network (CNN)

CNN digunakan untuk pemrosesan dan pengenalan citra. Fitur ini sangat berguna dalam mengidentifikasi objek dalam gambar, bahkan saat objek tersebut terhalang atau terdistorsi sebagian.

- Deep Reinforcement Learning

Deep reinforcement learning digunakan untuk robotik dan permainan game. Ini adalah jenis machine learning yang memungkinkan agen mempelajari cara berperilaku di lingkungan dengan berinteraksi dan menerima reward atau hukuman.

- Recurrent Neural Network (RNN)

RNN digunakan untuk natural language processing dan pengenalan ucapan. Alat ini sangat berguna dalam memahami konteks kalimat atau frasa, serta dapat digunakan untuk membuat teks atau menerjemahkan bahasa.

## 2. CONVOLUTIONAL NEURAL NETWORK (CNN)

Salah satu model terpopuler dalam Deep Learning saat ini adalah Convolutional Neural Network (CNN). Model CNN ini digunakan diberbagai aplikasi dan domain, serta sangat lazim digunakan dalam proyek pemrosesan gambar dan video. Hal tersebut dikarenakan model komputasi CNN ini menggunakan variasi perceptron multilayer yang saling berhubungan dan lapisan convolutional akan merekam wilayah gambar yang telah di input dan mengirimkannya untuk pemrosesan nonlinier. Penggunaan CNN memberikan beberapa keuntungan diantaranya Convolutional Neural Network ini memiliki akurasi yang sangat tinggi dalam masalah pengenalan gambar. Selain itu, CNN juga dapat mempelajari filter secara otomatis tanpa menyebutkannya secara eksplisit. Dimana filter tersebut, dapat membantu mengekstraksi fitur yang tepat dan relevan dari data yang telah di input. Namun, CNN juga memiliki kekurangan diantaranya, kurangnya

kemampuan untuk menjadi invarian secara spasial terhadap data input, membutuhkan banyak data pelatihan, dan overfitting. Overfitting dapat terjadi dikarenakan terlalu banyak data pelatihan maka algoritma kehilangan kemampuan untuk menggeneralisasi data tersebut. Selain itu, sama seperti ANN dimana CNN juga membutuhkan pelatihan dalam waktu yang lama.

### a. Cara Kerja CNN

Proses kerja CNN terdiri dari beberapa tahap utama:

- Input Layer

Gambar dimasukkan ke sistem sebagai array piksel, contohnya gambar RGB akan berukuran [tinggi, lebar, 3].

- Convolution Layer

Proses utama yang mengekstraksi fitur menggunakan filter/kernels. Operasi konvolusi dilakukan untuk mendekripsi fitur seperti tepi, tekstur, pola. Hasilnya adalah feature map atau activation map.

- Activation Function (ReLU)

Fungsi aktivasi, biasanya ReLU (Rectified Linear Unit), diterapkan setelah konvolusi. Fungsi ini membuat nilai negatif menjadi nol agar model tetap linear separable.

- Pooling Layer

Untuk mengurangi ukuran feature map (downsampling), mengurangi kompleksitas, dan mencegah overfitting. Contoh: Max Pooling (mengambil nilai maksimum), Average Pooling.

- Flatten Layer

- Mengubah hasil akhir dari pooling menjadi satu vektor agar bisa diproses oleh layer berikutnya.

- Fully Connected Layer (FC)

- Lapisan ini seperti neural network biasa (dense layer). Bertugas mengklasifikasikan fitur yang telah diekstraksi menjadi label output (misalnya: mobil, motor, truk).

- Output Layer

Memberikan hasil akhir dari klasifikasi. Biasanya menggunakan fungsi aktivasi Softmax untuk multi-kelas atau Sigmoid untuk biner.

### **b. Contoh Penggunaan CNN**

- Deteksi kendaraan pada sistem parkir otomatis.
- Face recognition (pengenalan wajah).
- Diagnostik medis berbasis citra (seperti MRI, X-ray).
- OCR (Optical Character Recognition).
- Mobil otonom (deteksi rambu lalu lintas, jalur).

## **3. TRANSFER LEARNING (MobileNetV2)**

### **a. Transfer Learning**

Transfer Learning adalah teknik dalam pembelajaran mesin (machine learning), khususnya deep learning, di mana kita menggunakan model yang sudah dilatih sebelumnya untuk menyelesaikan tugas baru yang serupa. Contoh: Alih-alih melatih model dari nol untuk mengenali jenis kendaraan, kita bisa menggunakan model yang sudah "mengerti" bentuk umum objek karena sebelumnya dilatih dengan jutaan gambar (seperti ImageNet).

### **b. MobileNetV2**

MobileNetV2 adalah model CNN ringan yang:

- Dirancang untuk perangkat mobile/edge (cepat & ringan)
- Menggunakan teknik Depthwise Separable Convolution (hemat komputasi)
- Cocok untuk real-time image classification.

MobileNetV2 dilatih di ImageNet (1.4 juta gambar, 1000 kelas).

Struktur MobileNetV2:

- Awal: Convolution layer biasa.
- Tengah: Banyak blok Inverted Residual Block (khas MobileNetV2).
- Akhir: Fully Connected Layer untuk klasifikasi 1000 kelas.

Dalam transfer learning, bagian akhir ini bisa kita modifikasi sesuai kebutuhan.

#### **4. IMAGE AUGMENTATION**

Image Augmentation adalah teknik dalam pemrosesan citra yang digunakan untuk memperluas dan memperkaya dataset gambar dengan cara melakukan berbagai transformasi pada gambar asli. Tujuan utamanya adalah untuk meningkatkan kemampuan generalisasi model dan mengurangi overfitting, terutama saat jumlah data pelatihan terbatas. Transformasi yang umum digunakan antara lain rotasi, pemotongan (cropping), flipping horizontal atau vertikal, perubahan skala (scaling), translasi (pergeseran posisi), penyesuaian kecerahan atau kontras, serta penambahan noise. Dengan menerapkan augmentasi, model dapat belajar mengenali objek dalam berbagai kondisi dan variasi, seperti perubahan sudut pandang atau pencahayaan. Teknik ini dilakukan secara real-time selama proses pelatihan, sehingga setiap epoch bisa menyajikan variasi gambar yang berbeda kepada model. Dalam proyek klasifikasi gambar seperti pendekripsi jenis kendaraan, image augmentation sangat membantu dalam meningkatkan akurasi dan ketahanan model terhadap variasi gambar di dunia nyata.

#### **B. STUDI KASUS**

##### **Deep Learning untuk Deteksi Jenis Kendaraan Berdasarkan Citra Digital dalam Sistem Parkir**

Parkir merupakan salah satu tantangan utama di area publik seperti pusat perbelanjaan, kampus, perkantoran, dan fasilitas umum lainnya. Ketidakteraturan dalam manajemen parkir sering kali menyebabkan kemacetan, keterlambatan, dan ketidaknyamanan bagi pengguna. Oleh karena itu, diperlukan sistem yang dapat secara otomatis mendekripsi jenis kendaraan yang masuk, serta memberikan simulasi parkir yang realistik dan efisien. Dalam studi kasus ini, dibangun sebuah sistem berbasis deep learning dengan pendekatan transfer learning menggunakan model MobileNetV2 untuk melakukan klasifikasi gambar kendaraan. Model ini dilatih untuk mengenali lima jenis kendaraan: motorcycles, cars, trucks, buses, dan bikes.

Hasil klasifikasi dari model digunakan untuk:

- Mendeteksi jenis kendaraan berdasarkan gambar yang diunggah oleh pengguna.
- Menentukan tarif parkir otomatis berdasarkan jenis kendaraan.
- Mencatat waktu masuk dan keluar kendaraan, serta menghitung biaya parkir akhir.
- Menyimpan data ke file laporan harian (CSV) secara otomatis.
- Menyediakan simulasi sistem parkir realistik seperti di dunia nyata, dengan input waktu masuk, waktu keluar, hingga proses pembayaran dan pengembalian uang.

Dengan pendekatan ini, sistem diharapkan menjadi solusi awal untuk mewujudkan manajemen parkir cerdas (smart parking) berbasis visi komputer dan kecerdasan buatan.

## Dataset

Subfolders:

- Bus (445 images)
- Cars (1.274 images)
- Motorcycles (1.277 images)
- Truck (633 images)
- Bikes (800 images)

Total: 4.429 images.

## Informasi Parkir

No.	Kendaraan	Harga Parkir (/jam)	Lantai	Area
1.	Bus	15.000	1 & 2	Area C1-C4 & Area B1-B6
2.	Cars	8.000	2 & 3	Area B7-A10 & Area A1-A6
3.	Truck	10.000	1	Area C5-C8
4.	Bikes	1.000	1	Area C9-C10
5.	Motorcycles	3.000	B1 & B2	Area D1-D10 & Area E1-E10

## File Dataset

Link: [Parkir Jenis Kendaraan.zip](#)

## C. KODE PROGRAM DAN PENJELASAN

Urutan Langkah-Langkah Klasifikasi Gambar:

### 1. Upload File Dataset

#### a. Import library

```
from google.colab import files  
import zipfile  
import os
```

- from google.colab import files: Mengimpor modul files dari google.colab. Modul ini memungkinkan interaksi antara lingkungan Google Colab dengan sistem file lokal Anda, khususnya untuk mengunggah file.
- import zipfile: Mengimpor modul zipfile. Modul ini menyediakan fungsionalitas untuk bekerja dengan arsip ZIP, seperti membuat, membaca, dan mengekstrak file.
- import os: Mengimpor modul os. Modul ini menyediakan cara untuk berinteraksi dengan sistem operasi, seperti menavigasi direktori, membuat folder, dan memeriksa konten folder.

#### b. Mengunggah file ZIP secara manual

```
# Upload file ZIP  
uploaded = files.upload()
```

files.upload(): Fungsi ini akan menampilkan widget di Google Colab yang memungkinkan Anda untuk memilih dan mengunggah file dari komputer lokal Anda. Setelah file diunggah, informasi tentang file tersebut (seperti nama file dan ukuran) akan disimpan dalam variabel uploaded sebagai dictionary.

#### c. Ekstraksi file ZIP

```
# Ekstrak file ZIP  
for file in uploaded.keys():  
    zip_path = file  
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:  
        zip_ref.extractall("dataset")
```

- for file in uploaded.keys(): Kode ini melakukan iterasi melalui nama-nama file yang berhasil diunggah. Dalam kasus ini, karena Anda hanya mengunggah satu file (Parkir Jenis Kendaraan.zip), loop ini akan berjalan satu kali.
- zip\_path = file: Variabel zip\_path menyimpan nama file ZIP yang diunggah.
- with zipfile.ZipFile(zip\_path, 'r') as zip\_ref: Baris ini membuka file ZIP dalam mode baca ('r'). Penggunaan with memastikan bahwa file ZIP akan ditutup secara otomatis setelah operasi selesai, bahkan jika terjadi error. Objek zip\_ref adalah referensi ke file ZIP yang dibuka.
- zip\_ref.extractall("dataset"): Metode ini mengekstrak semua konten dari file ZIP ke dalam sebuah folder baru bernama "dataset". Jika folder "dataset" belum ada, maka akan dibuat secara otomatis.

#### d. Verifikasi hasil ekstraksi

```
# Cek folder hasil ekstraksi
os.listdir("dataset")
```

os.listdir("dataset"): Fungsi ini akan mengembalikan daftar semua file dan subfolder yang ada di dalam folder "dataset".

#### Output



The screenshot shows a file upload dialog from Google Colab. It displays a single file entry: "Parkir Jenis Kendaraan.zip" with the following details:  
 - Type: application/x-zip-compressed  
 - Size: 932385917 bytes  
 - Last modified: 6/10/2025 - 100% done  
 - Action: Saving Parkir Jenis Kendaraan.zip to Parkir Jenis Kendaraan.zip  
 - Sub-action: ['Parkir Jenis Kendaraan']

- Baris pertama dan kedua menunjukkan bahwa file Parkir Jenis Kendaraan.zip telah berhasil diunggah ke lingkungan Google Colab. Ini termasuk informasi tentang tipe file, ukuran, tanggal modifikasi, dan status unggahan (100% done).
- ['Parkir Jenis Kendaraan']: Ini adalah output dari os.listdir("dataset"). Ini menunjukkan bahwa setelah ekstraksi, di dalam folder "dataset" kini terdapat satu subfolder bernama Parkir Jenis Kendaraan. Ini mengindikasikan bahwa proses ekstraksi berhasil dan struktur dataset siap untuk langkah selanjutnya dalam proyek yang akan dilakukan.

## 2. Preprocessing Data (Augmentasi + Normalisasi)

### a. Import Library

```
import os
from PIL import Image
```

- os: Digunakan untuk berinteraksi dengan sistem operasi, seperti menjelajahi direktori (os.walk) dan menghapus file (os.remove).
- PIL.Image (Pillow): Pustaka populer untuk pemrosesan gambar. Image.open() digunakan untuk membuka gambar, dan img.load() untuk memuat data gambar ke memori, yang penting untuk mendeteksi gambar yang terpotong (truncated) atau rusak.

### b. Iterasi Direktori

```
# Fungsi untuk menghapus gambar yang rusak atau terpotong
(truncated)
def delete_truncated_images(directory):
    count_deleted = 0
    print(f"⌚ Memulai pengecekan dan penghapusan gambar
rusak di: {directory}")

    # Iterasi melalui semua file di direktori dan
    subdirektorinya
    for root, _, files in os.walk(directory):
        for file in files:
```

Fungsi ini menggunakan os.walk(directory) untuk menelusuri seluruh direktori dan subdirektorinya secara rekursif. Ini memastikan bahwa semua gambar, tidak peduli di mana mereka berada dalam struktur folder dataset, akan diperiksa.

### c. Identifikasi Gambar

```
        # Periksa apakah file adalah gambar (jpg,
jpeg, png)
        if file.lower().endswith('.jpg', '.jpeg',
'.png')):
            filepath = os.path.join(root, file)
            try:
```

Hanya file dengan ekstensi jpg, jpeg, atau .png yang akan diproses, karena ini adalah format gambar umum.

#### d. Pendekripsi Gambar Rusak

```
# Buka dan muat gambar untuk mendekripsi kerusakan
with Image.open(filepath) as img:
    img.load()
except (IOError, SyntaxError) as e:
```

- Setiap gambar dibuka menggunakan Image.open(filepath). Penggunaan with statement sangat disarankan karena memastikan file handle ditutup secara otomatis, bahkan jika terjadi error.
- img.load(): Ini adalah langkah penting. Saat membuka gambar dengan Pillow, data gambar sebenarnya tidak langsung dimuat sepenuhnya ke memori. img.load() memaksa Pillow untuk membaca seluruh piksel gambar. Jika gambar terpotong atau rusak, img.load() akan memicu exception (misalnya IOError atau SyntaxError).
- try-except blok: Blok ini menangani exception yang mungkin terjadi. Jika IOError (masalah input/output, seringkali berarti file rusak/tidak lengkap) atau SyntaxError (biasanya format file gambar tidak valid) terdeteksi, kode di dalam blok except akan dieksekusi.

#### e. Penghapusan Gambar Rusak

```
# Jika gambar rusak, hapus file tersebut
print(f"Menghapus gambar rusak:
{filepath} - Error: {e}")
try:
    os.remove(filepath)
    count_deleted += 1
except OSError as remove_error:
    print(f"Error removing file
{filepath}: {remove_error}")
```

- Jika sebuah gambar terdeteksi rusak, pesan akan dicetak ke konsol, dan kemudian os.remove(filepath) akan menghapus file yang bermasalah tersebut dari sistem.
- Variabel count\_deleted melacak berapa banyak gambar yang telah dihapus.

## f. Pelaporan Hasil

```
# Laporkan hasil penghapusan gambar
if count_deleted == 0:
    print("✅ Tidak ada gambar rusak yang
ditemukan.")
else:
    print(f"☒ Total gambar rusak yang dihapus:
{count_deleted}")

# Panggil fungsi pembersihan untuk direktori dataset
# kendaraan
dataset_directory = 'Parkir Jenis Kendaraan/Vehicles' # 
Pastikan path sesuai
delete_truncated_images(dataset_directory)
```

Setelah semua gambar diperiksa, fungsi ini akan mencetak ringkasan apakah ada gambar rusak yang ditemukan dan dihapus.

### Output

```
→ 🔍 Memulai pengecekan dan penghapusan gambar rusak di: Parkir Jenis Kendaraan/Vehicles
✅ Tidak ada gambar rusak yang ditemukan.
```

- Ini adalah pesan awal yang dicetak oleh program, menunjukkan bahwa fungsi delete\_truncated\_images telah mulai bekerja dan sedang memindai direktori Parkir Jenis Kendaraan/Vehicles untuk mencari gambar-gambar yang rusak. Direktori ini seharusnya berisi semua gambar kendaraan yang telah diekstrak pada langkah sebelumnya.
- Pesan akhir merupakan output yang sangat positif! Ini berarti program telah berhasil memindai semua gambar di dalam direktori yang ditentukan, dan tidak menemukan satupun gambar yang rusak, terpotong, atau dengan format yang tidak valid. Ini menunjukkan bahwa dataset bersih dan siap untuk tahap pra-pemrosesan data selanjutnya seperti augmentasi dan normalisasi, tanpa risiko gangguan dari file gambar yang korup.

### 3. Augmentasi Data

#### a. Import Kelas

```
from tensorflow.keras.preprocessing.image import  
ImageDataGenerator
```

Baris ini mengimpor kelas ImageDataGenerator dari tensorflow.keras.preprocessing.image. Kelas ini adalah inti dari proses pra-pemrosesan dan augmentasi gambar dalam kode ini.

#### b. Ukuran Model

```
# Menentukan ukuran gambar yang digunakan oleh model dan  
# ukuran batch  
image_size = (224, 224)  
batch_size = 32
```

Bagian ini mendefinisikan beberapa konstanta penting untuk proses pelatihan:

- `image_size = (224, 224)`: Menentukan ukuran target untuk setiap gambar yang akan dimasukkan ke model. Sebagian besar model deep learning yang sudah terlatih (pretrained models), seperti VGG16 atau ResNet, dirancang untuk menerima input gambar dengan ukuran tertentu, biasanya 224x224 piksel. Semua gambar dalam dataset akan diubah ukurannya menjadi dimensi ini.
- `batch_size = 32`: Menentukan jumlah gambar yang akan diproses dalam satu batch selama pelatihan. Menggunakan batch membantu mengelola memori dan mempercepat proses pelatihan. Model akan memperbarui bobotnya (weights) setelah memproses setiap batch.

#### c. Objek ImageDataGenerator

```
# Membuat objek ImageDataGenerator untuk preprocessing dan  
# augmentasi data  
datagen = ImageDataGenerator(  
    rescale=1./255,  
    rotation_range=10,  
    width_shift_range=0.1,  
    height_shift_range=0.1,  
    zoom_range=0.1,  
    horizontal_flip=True,  
    validation_split=0.2  
)
```

Ini adalah bagian inti di mana objek `ImageDataGenerator` diinisialisasi dengan berbagai parameter augmentasi dan normalisasi:

- `rescale=1./255`: Ini adalah langkah normalisasi. Nilai piksel dalam gambar biasanya berkisar antara 0 hingga 255. Dengan membagi setiap nilai piksel dengan 255, rentang nilai piksel diubah menjadi 0 hingga 1. Normalisasi ini sangat penting karena membantu model belajar lebih efisien dan stabil.
- `rotation_range=10`: Parameter ini melakukan rotasi gambar secara acak hingga 10 derajat (baik searah jarum jam maupun berlawanan arah jarum jam). Ini membantu model mengenali objek meskipun sedikit miring.
- `width_shift_range=0.1`: Parameter ini melakukan pergeseran horizontal gambar secara acak hingga 10% dari total lebar gambar. Ini membuat model lebih toleran terhadap sedikit variasi posisi objek dalam gambar.
- `height_shift_range=0.1`: Mirip dengan `width_shift_range`, ini melakukan pergeseran vertikal gambar secara acak hingga 10% dari total tinggi gambar.
- `zoom_range=0.1`: Parameter ini melakukan pembesaran (zoom) gambar secara acak hingga 10%. Ini membantu model mengenali objek pada skala yang berbeda.
- `horizontal_flip=True`: Parameter ini secara acak melakukan pembalikan gambar secara horizontal. Misalnya, sebuah mobil yang menghadap kiri bisa dibalik menjadi menghadap kanan. Ini berguna jika orientasi objek tidak terlalu penting untuk klasifikasi.
- `validation_split=0.2`: Parameter ini sangat penting untuk pembagian dataset. Ini memberitahu generator bahwa 20% dari data akan digunakan sebagai data validasi, sementara sisanya (80%) akan digunakan sebagai data pelatihan. Ini memungkinkan evaluasi model secara real-time selama pelatihan.

#### d. Generator Data Training

```
# Membuat generator data training dari folder 'Vehicles'  
train_data = datagen.flow_from_directory(  
    'Parkir Jenis Kendaraan/Vehicles',  
    target_size=image_size,  
    batch_size=batch_size,  
    class_mode='categorical',  
    subset='training'  
)
```

Baris ini membuat generator data pelatihan:

- `datagen.flow_from_directory('Parkir Jenis Kendaraan/Vehicles', ...)`: Metode ini membaca gambar langsung dari struktur folder. Keras akan secara otomatis mengidentifikasi kelas-kelas berdasarkan nama subfolder di dalam Parkir Jenis Kendaraan/Vehicles.
- `target_size=image_size`: Mengubah ukuran semua gambar yang dibaca menjadi 224x224 piksel.
- `batch_size=batch_size`: Mengelompokkan gambar menjadi batch berukuran 32.
- `class_mode='categorical'`: Menentukan bahwa label kelas akan diubah menjadi format one-hot encoding (misalnya, jika ada 3 kelas, maka kelas 0 menjadi [1,0,0], kelas 1 menjadi [0,1,0], dll.). Ini cocok untuk masalah klasifikasi multi-kelas.
- `subset='training'`: Mengindikasikan bahwa generator ini hanya akan mengambil gambar yang ditujukan untuk set pelatihan berdasarkan `validation_split` yang telah ditentukan sebelumnya.

#### e. Generator Data Validation

```
# Membuat generator data validasi dari folder 'Vehicles'  
val_data = datagen.flow_from_directory(  
    'Parkir Jenis Kendaraan/Vehicles',  
    target_size=image_size,  
    batch_size=batch_size,  
    class_mode='categorical',  
    subset='validation'  
)
```

Mirip dengan train\_data, baris ini membuat generator data validasi:

- Semua parameter sama kecuali subset='validation', yang berarti generator ini akan mengambil 20% data yang telah dialokasikan untuk validasi. Data validasi ini penting untuk memantau performa model pada data yang belum pernah dilihat selama pelatihan, membantu mendeteksi overfitting.

## Output

```
Found 3530 images belonging to 5 classes.  
Found 881 images belonging to 5 classes.
```

Output ini mengkonfirmasi bahwa ImageDataGenerator berhasil memuat dan membagi dataset gambar:

- Baris Pertama (Data Pelatihan): Menunjukkan 3530 gambar telah dialokasikan untuk pelatihan, terbagi dalam 5 kategori (kelas) kendaraan yang berbeda. Ini adalah 80% dari total dataset yang digunakan untuk melatih model.
- Baris Kedua (Data Validasi): Menunjukkan 881 gambar telah dialokasikan untuk validasi, juga terbagi dalam 5 kategori kelas yang sama. Ini adalah 20% dari total dataset yang digunakan untuk menguji performa model selama pelatihan.

## 4. Transfer Learning dengan MobileNetV2

Bagian ini membahas implementasi Transfer Learning menggunakan arsitektur MobileNetV2. Transfer learning adalah teknik di mana model yang sudah dilatih pada tugas yang besar (misalnya, klasifikasi ImageNet) digunakan sebagai titik awal untuk tugas baru (klasifikasi jenis kendaraan). Hal ini sangat efisien karena model sudah mempelajari fitur-fitur dasar dari gambar, dan kita hanya perlu menyesuaikannya untuk tugas spesifik kita.

### a. Import Library

```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense,
GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

- MobileNetV2: Arsitektur Convolutional Neural Network (CNN) yang efisien, cocok untuk perangkat mobile atau lingkungan dengan sumber daya terbatas. Diimpor dari tensorflow.keras.applications.
- Model: Digunakan untuk membuat model Keras yang fleksibel dengan mendefinisikan input dan output.
- Dense: Lapisan fully connected (dense layer) yang digunakan untuk klasifikasi di bagian akhir model.
- GlobalAveragePooling2D: Lapisan yang menghitung rata-rata spasial dari fitur map, merampingkan output CNN menjadi vektor tunggal.
- Adam: Algoritma optimizer yang umum digunakan dan efisien untuk pelatihan model deep learning.
- EarlyStopping: Sebuah callback yang akan menghentikan pelatihan lebih awal jika performa model pada data validasi tidak membaik setelah beberapa epoch tertentu, mencegah overfitting dan menghemat waktu.

### b. Inisiasi Base Model

```
# Load base model MobileNetV2 tanpa top (fully connected
layer)
base_model = MobileNetV2(weights='imagenet',
include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False # Freeze layer bawaan agar
cepat konvergen
```

- MobileNetV2(...): Memuat arsitektur MobileNetV2.
  - weights='imagenet': Menggunakan bobot (weights) yang sudah dilatih pada dataset ImageNet. Ini adalah inti dari transfer learning, karena model sudah memiliki kemampuan pengenalan fitur visual yang kuat.

- include\_top=False: Sangat penting! Parameter ini menghilangkan lapisan fully connected (lapisan klasifikasi) terakhir dari MobileNetV2 asli. Ini karena lapisan asli dirancang untuk 1000 kelas ImageNet, sedangkan kita memiliki jumlah kelas yang berbeda (5 kelas kendaraan). Kita akan menambahkan lapisan klasifikasi custom sendiri.
  - input\_shape=(224, 224, 3): Menentukan bentuk input yang diharapkan model. Ini cocok dengan image\_size yang ditentukan pada langkah preprocessing (tinggi 224, lebar 224, dan 3 kanal warna RGB).
- base\_model.trainable = False: Baris ini "membekukan" (freeze) semua lapisan pada base\_model. Artinya, selama pelatihan, bobot dari lapisan-lapisan MobileNetV2 yang sudah ada (yang dilatih pada ImageNet) tidak akan diperbarui. Ini dilakukan untuk:
  - Mempercepat konvergensi: Model tidak perlu belajar fitur dasar dari awal.
  - Mencegah overfitting: Terutama pada dataset kecil, membiarkan lapisan-lapisan ini tidak diubah membantu model untuk tidak terlalu spesifik pada data pelatihan.

### c. Custom Classification Layer

```
# Tambahkan custom classification layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(train_data.num_classes,
activation='softmax')(x)
```

- x = base\_model.output: Mengambil output dari base\_model. Output ini adalah feature map yang telah diekstraksi oleh MobileNetV2.
- x = GlobalAveragePooling2D()(x): Lapisan ini mengurangi dimensi spasial dari feature map menjadi satu vektor tunggal. Ini adalah alternatif yang populer untuk Flatten karena mengurangi jumlah parameter dan overfitting.
- x = Dense(128, activation='relu')(x): Menambahkan lapisan fully connected (Dense) dengan 128 neuron. activation='relu' (Rectified

Linear Unit) adalah fungsi aktivasi non-linear yang umum digunakan. Lapisan ini berfungsi sebagai lapisan hidden untuk memproses fitur yang diekstraksi.

- predictions = Dense(train\_data.num\_classes, activation='softmax')(x): Ini adalah lapisan output model.
  - train\_data.num\_classes: Secara otomatis mendapatkan jumlah kelas yang terdeteksi oleh ImageDataGenerator (dalam kasus ini, 5).
  - activation='softmax': Fungsi aktivasi softmax digunakan untuk masalah klasifikasi multi-kelas. Ini menghasilkan distribusi probabilitas di mana jumlah probabilitas untuk semua kelas adalah 1. Kelas dengan probabilitas tertinggi adalah prediksi model.

#### d. Compile Model

```
# Buat model akhir
model = Model(inputs=base_model.input,
outputs=predictions)

# Compile
model.compile(optimizer=Adam(learning_rate=0.0005),
loss='categorical_crossentropy', metrics=['accuracy'])
```

- model = Model(inputs=base\_model.input, outputs=predictions): Membuat objek model Keras yang lengkap dengan mendefinisikan inputnya (input dari base\_model) dan outputnya (lapisan predictions yang baru dibuat).
- model.compile(...): Mengkonfigurasi model untuk pelatihan.
  - optimizer=Adam(learning\_rate=0.0005): Menggunakan optimizer Adam dengan learning rate 0.0005. Learning rate ini relatif kecil, yang umum saat melakukan fine-tuning atau transfer learning untuk menghindari "mengganggu" bobot yang sudah dilatih.
  - loss='categorical\_crossentropy': Fungsi loss yang digunakan untuk masalah klasifikasi multi-kelas dengan label one-hot encoded (seperti yang dihasilkan oleh class\_mode='categorical').

Fungsi ini mengukur seberapa jauh prediksi model dari label sebenarnya.

- metrics=['accuracy']: Metrik yang akan dipantau selama pelatihan dan evaluasi. accuracy (akurasi) adalah persentase prediksi yang benar.

#### e. Early Stopping

```
# Early Stopping
early_stop = EarlyStopping(patience=5,
                           restore_best_weights=True)
```

- EarlyStopping(...): Menginisialisasi callback EarlyStopping.
  - patience=5: Jika akurasi validasi tidak membaik selama 5 epoch berturut-turut, pelatihan akan dihentikan.
  - restore\_best\_weights=True: Jika pelatihan dihentikan karena early stopping, bobot model akan dikembalikan ke epoch terbaik (saat akurasi validasi tertinggi).

#### f. Training Model

```
# Training ulang dengan model yang lebih kuat
history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=20,
    callbacks=[early_stop]
)
```

Model.fit(...) adalah perintah untuk memulai pelatihan model.

- train\_data: Menggunakan generator train\_data yang telah dibuat sebelumnya untuk menyediakan data pelatihan.
- validation\_data=val\_data: Menggunakan generator val\_data untuk memvalidasi model setelah setiap epoch.
- epochs=20: Menentukan jumlah epoch (iterasi penuh melalui seluruh dataset pelatihan) maksimum. Pelatihan bisa berhenti lebih awal jika early stopping terpicu.
- callbacks=[early\_stop]: Menerapkan callback early\_stop selama pelatihan.

## Output

```
→ /usr/local/lib/python3.11/dist-packages/PIL/Image.py:1043: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images.
  warnings.warn(
Epoch 1/20
111/111      - 308s 3s/step - accuracy: 0.7822 - loss: 0.5751 - val_accuracy: 0.9387 - val_loss: 0.1681
Epoch 2/20
111/111      - 270s 2s/step - accuracy: 0.9138 - loss: 0.2030 - val_accuracy: 0.9398 - val_loss: 0.1656
Epoch 3/20
111/111      - 317s 2s/step - accuracy: 0.9354 - loss: 0.1645 - val_accuracy: 0.9296 - val_loss: 0.1616
Epoch 4/20
111/111      - 262s 2s/step - accuracy: 0.9408 - loss: 0.1515 - val_accuracy: 0.9580 - val_loss: 0.1270
Epoch 5/20
111/111      - 268s 2s/step - accuracy: 0.9509 - loss: 0.1252 - val_accuracy: 0.9319 - val_loss: 0.1704
Epoch 6/20
111/111      - 288s 3s/step - accuracy: 0.9557 - loss: 0.1126 - val_accuracy: 0.9308 - val_loss: 0.1747
Epoch 7/20
111/111      - 268s 2s/step - accuracy: 0.9655 - loss: 0.0931 - val_accuracy: 0.9512 - val_loss: 0.1262
Epoch 8/20
111/111      - 263s 2s/step - accuracy: 0.9685 - loss: 0.0923 - val_accuracy: 0.9432 - val_loss: 0.1530
Epoch 9/20
111/111      - 267s 2s/step - accuracy: 0.9744 - loss: 0.0725 - val_accuracy: 0.9512 - val_loss: 0.1368
Epoch 10/20
111/111      - 264s 2s/step - accuracy: 0.9774 - loss: 0.0641 - val_accuracy: 0.9432 - val_loss: 0.1536
Epoch 11/20
111/111      - 268s 2s/step - accuracy: 0.9738 - loss: 0.0808 - val_accuracy: 0.9489 - val_loss: 0.1436
Epoch 12/20
111/111      - 263s 2s/step - accuracy: 0.9761 - loss: 0.0711 - val_accuracy: 0.9467 - val_loss: 0.1477
```

Ini adalah pesan peringatan dari pustaka PIL (Pillow). Ini bukan error fatal, melainkan saran. Beberapa gambar dalam dataset mungkin menggunakan format palet dengan transparansi yang lebih baik dikonversi ke format RGBA untuk penanganan yang lebih baik oleh Pillow. Biasanya, ini tidak menghalangi proses pelatihan, tetapi bisa menjadi indikasi adanya beberapa gambar dengan format yang kurang optimal.

Interpretasi:

- Epoch X/20: Menunjukkan epoch yang sedang berjalan dari total 20 epoch maksimum.
- 111/111: Jumlah batch yang diproses per epoch.
- 308s: Durasi waktu yang dibutuhkan untuk menyelesaikan epoch tersebut.
- accuracy: 0.7822: Akurasi model pada data pelatihan. Ini menunjukkan seberapa baik model belajar dari data yang dilihatnya.
- loss: 0.5751: Nilai error (loss) model pada data pelatihan. Semakin rendah, semakin baik.
- val\_accuracy: 0.9387: Akurasi model pada data validasi. Ini adalah metrik kunci untuk menilai kemampuan generalisasi model pada data baru, serta untuk mendeteksi overfitting.
- val\_loss: 0.1681: Nilai error (loss) model pada data validasi. Semakin rendah, semakin baik.

## Tren yang Teramat

Dari output yang diberikan (Epoch 1 hingga Epoch 12), terlihat bahwa:

- Akurasi Pelatihan (accuracy) terus meningkat, dan Loss Pelatihan (loss) terus menurun, menunjukkan bahwa model berhasil belajar dari data pelatihan.
- Akurasi Validasi (val\_accuracy) relatif tinggi sejak awal (sekitar 93-95%) dan Loss Validasi (val\_loss) menurun kemudian cenderung stabil. Ini menandakan model memiliki kemampuan generalisasi yang baik.

## 5. Training Model (Fine-tuning)

Bagian kode ini merupakan kelanjutan dari proses transfer learning sebelumnya, di mana kita telah melatih head klasifikasi baru pada fitur yang diekstraksi oleh MobileNetV2 yang frozen. Sekarang, kita akan melakukan fine-tuning, yaitu membuka dan melatih sebagian kecil lapisan terakhir dari base model MobileNetV2 bersama dengan lapisan klasifikasi custom kita. Tujuannya adalah untuk menyesuaikan fitur-fitur yang dipelajari oleh MobileNetV2 agar lebih spesifik dan relevan dengan dataset kendaraan kita, yang berpotensi meningkatkan akurasi lebih lanjut.

### a. Unfreeze untuk Fine-Tuning

```
# Unfreeze sebagian layer akhir untuk fine-tuning
base_model.trainable = True
for layer in base_model.layers[:-30]: # hanya fine-tune
    30 layer terakhir
        layer.trainable = False
```

- `base_model.trainable = True`: Pertama, semua lapisan di dalam `base_model` (MobileNetV2) diatur menjadi trainable (dapat dilatih). Ini adalah langkah awal untuk mengizinkan perubahan pada bobotnya.
- `for layer in base_model.layers[:-30]: layer.trainable = False`: Ini adalah bagian krusial untuk fine-tuning selektif.
  - `base_model.layers`: Mengakses semua lapisan dalam base model MobileNetV2.
  - `[:-30]`: Ini adalah slicing Python yang berarti "semua lapisan kecuali 30 lapisan terakhir".

- layer.trainable = False: Untuk setiap lapisan yang termasuk dalam slice ini (yaitu, lapisan-lapisan awal/dasar dari MobileNetV2), properti trainable diatur kembali menjadi False.
- Tujuan: Dengan cara ini, hanya 30 lapisan terakhir dari MobileNetV2 (yang cenderung mempelajari fitur-fitur yang lebih spesifik dan kompleks) yang akan diizinkan untuk diperbarui bobotnya selama pelatihan. Lapisan-lapisan awal (yang mempelajari fitur-fitur umum seperti garis dan tepi) tetap frozen untuk mempertahankan pengetahuan ImageNet dan mencegah overfitting yang terlalu cepat.

### b. Kompilasi Ulang Model

```
# Compile ulang model dengan learning rate lebih kecil
model.compile(optimizer=Adam(learning_rate=1e-5),
loss='categorical_crossentropy', metrics=['accuracy'])
```

- Setelah mengubah properti trainable pada beberapa lapisan, model harus dikompilasi ulang agar perubahan tersebut diterapkan.
- optimizer=Adam(learning\_rate=1e-5): Optimizer Adam digunakan kembali, tetapi kali ini dengan learning rate yang jauh lebih kecil (0.00001) dibandingkan sebelumnya (0.0005).
  - Alasan learning rate kecil: Saat fine-tuning, bobot model sudah sangat baik. Menggunakan learning rate yang kecil memastikan bahwa pembaruan bobot dilakukan secara hati-hati dan bertahap, menghindari "merusak" pengetahuan yang sudah ada dari pre-trained model. Ini membantu model untuk "menyesuaikan" sedikit demi sedikit dengan data baru tanpa melupakan apa yang sudah dipelajari.
- loss='categorical\_crossentropy', metrics=['accuracy']: Fungsi loss dan metrik tetap sama karena jenis masalah (klasifikasi multi-kelas) tidak berubah.

### c. Training dengan Fine-Tuning

```
# Lanjutkan training dengan fine-tuning
history_finetune = model.fit(
    train_data,
```

```
        validation_data=val_data,  
        epochs=10,  
        callbacks=[early_stop]  
)
```

- `model.fit(...)`: Melanjutkan proses pelatihan model.
- `train_data, validation_data=val_data`: Tetap menggunakan generator data yang sama untuk pelatihan dan validasi.
- `epochs=10`: Jumlah epoch maksimum untuk fase fine-tuning ini. Biasanya, fase fine-tuning tidak memerlukan banyak epoch karena model sudah memiliki bobot yang baik.
- `callbacks=[early_stop]`: Callback EarlyStopping yang sama masih digunakan. Ini sangat penting dalam fine-tuning untuk mencegah overfitting karena sekarang sebagian lapisan base model juga dilatih, yang meningkatkan risiko overfitting jika dilatih terlalu lama. Early stopping akan memastikan model berhenti pada titik terbaik berdasarkan akurasi validasi.

## Output

```
Epoch 1/10  
111/111 349s 3s/step - accuracy: 0.8499 - loss: 0.4331 - val_accuracy: 0.9659 - val_loss: 0.1117  
Epoch 2/10  
111/111 377s 3s/step - accuracy: 0.9377 - loss: 0.1738 - val_accuracy: 0.9591 - val_loss: 0.1107  
Epoch 3/10  
111/111 331s 3s/step - accuracy: 0.9415 - loss: 0.1619 - val_accuracy: 0.9557 - val_loss: 0.1174  
Epoch 4/10  
111/111 325s 3s/step - accuracy: 0.9502 - loss: 0.1359 - val_accuracy: 0.9478 - val_loss: 0.1274  
Epoch 5/10  
111/111 322s 3s/step - accuracy: 0.9608 - loss: 0.1211 - val_accuracy: 0.9694 - val_loss: 0.1010  
Epoch 6/10  
111/111 326s 3s/step - accuracy: 0.9611 - loss: 0.1098 - val_accuracy: 0.9694 - val_loss: 0.1029  
Epoch 7/10  
111/111 326s 3s/step - accuracy: 0.9631 - loss: 0.0998 - val_accuracy: 0.9716 - val_loss: 0.0895  
Epoch 8/10  
111/111 328s 3s/step - accuracy: 0.9770 - loss: 0.0809 - val_accuracy: 0.9671 - val_loss: 0.0987  
Epoch 9/10  
111/111 325s 3s/step - accuracy: 0.9764 - loss: 0.0835 - val_accuracy: 0.9614 - val_loss: 0.1121  
Epoch 10/10  
111/111 329s 3s/step - accuracy: 0.9745 - loss: 0.0740 - val_accuracy: 0.9659 - val_loss: 0.1014
```

Output ini menampilkan log kemajuan pelatihan model selama fase fine-tuning, di mana sebagian lapisan MobileNetV2 dilatih kembali dengan learning rate yang lebih kecil.

Interpretasi:

- Epoch X/10: Menunjukkan epoch yang sedang berjalan dari total 10 epoch maksimum untuk fase fine-tuning ini.

- accuracy & loss: Akurasi dan loss pada data pelatihan. Terus meningkat (akurasi) dan menurun (loss) seiring epoch, menunjukkan model terus belajar dari data pelatihan.
- val\_accuracy & val\_loss: Akurasi dan loss pada data validasi. Ini adalah metrik kunci untuk menilai performa model pada data yang belum pernah dilihat.

Tren yang Teramatidi dari Output:

- Akurasi Pelatihan (accuracy): Meningkat secara konsisten dari sekitar 85% di Epoch 1 hingga mencapai sekitar 97.45% di Epoch 10.
- Loss Pelatihan (loss): Menurun secara konsisten dari sekitar 0.43 menjadi 0.07.
- Akurasi Validasi (val\_accuracy): Dimulai sudah sangat tinggi (96.59% di Epoch 1) dan tetap konsisten tinggi, berfluktuasi sedikit di sekitar 96% - 97% (tertinggi di Epoch 7: 97.16%).
- Loss Validasi (val\_loss): Berfluktuasi antara 0.08 dan 0.12, menunjukkan bahwa model tidak overfit secara signifikan pada data validasi.

## 6. Evaluasi Model

Bagian ini berfokus pada evaluasi performa model yang telah dilatih menggunakan berbagai metrik standar dalam klasifikasi, seperti Confusion Matrix, Classification Report, dan Accuracy Score.

### a. Import Library

```
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

- sklearn.metrics: Pustaka scikit-learn menyediakan berbagai metrik untuk evaluasi model. Di sini, kita mengimpor classification\_report, confusion\_matrix, dan accuracy\_score.
- numpy as np: Pustaka untuk komputasi numerik, sangat berguna untuk manipulasi array.

- seaborn as sns: Pustaka untuk visualisasi data yang indah dan informatif, khususnya untuk membuat heatmap Confusion Matrix.
- matplotlib.pyplot as plt: Pustaka untuk membuat plot dan grafik.

### b. Prediksi Data Validasi

```
# Prediksi di data validasi
Y_pred = model.predict(val_data)
y_pred = np.argmax(Y_pred, axis=1)
```

- Y\_pred = model.predict(val\_data): Model yang telah dilatih (model) digunakan untuk membuat prediksi pada val\_data (generator data validasi). Output Y\_pred akan berupa array NumPy di mana setiap baris mewakili satu gambar, dan setiap kolom berisi probabilitas bahwa gambar tersebut termasuk dalam kelas tertentu. Karena kita menggunakan softmax di lapisan output, Y\_pred berisi probabilitas untuk setiap kelas.
- y\_pred = np.argmax(Y\_pred, axis=1):
  - np.argmax(): Fungsi ini mengembalikan indeks dari nilai maksimum di sepanjang sumbu tertentu.
  - axis=1: Menunjukkan bahwa operasi argmax dilakukan untuk setiap baris. Jadi, untuk setiap gambar, ia akan menemukan indeks kelas dengan probabilitas tertinggi.
  - Tujuan: Mengubah probabilitas keluaran model (Y\_pred) menjadi label kelas yang diprediksi (y\_pred), yaitu kelas dengan probabilitas tertinggi. Misalnya, jika output probabilitas untuk sebuah gambar adalah [0.1, 0.8, 0.05, 0.05, 0.0], maka np.argmax akan menghasilkan indeks 1 (karena 0.8 adalah probabilitas tertinggi pada indeks 1).

### c. Confusion Matrix

```
# Tampilkan Confusion Matrix
cm = confusion_matrix(val_data.classes, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d',
            xticklabels=val_data.class_indices.keys(),
            yticklabels=val_data.class_indices.keys())
plt.title("Confusion Matrix")
plt.show()
```

- `cm = confusion_matrix(val_data.classes, y_pred)`: Menghitung Confusion Matrix.
  - `val_data.classes`: Ini adalah label kelas sebenarnya (label ground truth) dari gambar-gambar di set validasi.
  - `y_pred`: Ini adalah label kelas yang diprediksi oleh model.
  - Confusion Matrix adalah tabel yang menunjukkan kinerja model klasifikasi pada sekumpulan data uji yang label sebenarnya diketahui. Barisnya mewakili kelas aktual, dan kolomnya mewakili kelas yang diprediksi.
- `plt.figure(figsize=(8,6))`: Membuat figure Matplotlib baru dengan ukuran 8x6 inci untuk plot.
- `sns.heatmap(cm, annot=True, fmt='d', xticklabels=val_data.class_indices.keys(), yticklabels=val_data.class_indices.keys())`: Membuat heatmap dari Confusion Matrix menggunakan Seaborn.
  - `cm`: Data Confusion Matrix.
  - `annot=True`: Menampilkan nilai numerik di setiap sel heatmap.
  - `fmt='d'`: Memformat anotasi sebagai bilangan bulat.
  - `xticklabels=val_data.class_indices.keys()`: Menetapkan label untuk sumbu X (kelas yang diprediksi) berdasarkan nama-nama kelas yang terdeteksi oleh ImageDataGenerator.
  - `yticklabels=val_data.class_indices.keys()`: Menetapkan label untuk sumbu Y (kelas aktual) berdasarkan nama-nama kelas.
- `plt.title("Confusion Matrix")`: Memberikan judul pada plot.
- `plt.show()`: Menampilkan plot Confusion Matrix.

#### d. Classification Report

```
# Classification Report
print(classification_report(val_data.classes, y_pred,
                           target_names=val_data.class_indices.keys()))

print(classification_report(val_data.classes, y_pred,
                           target_names=val_data.class_indices.keys())): Menghasilkan laporan teks yang merangkum metrik presisi, recall, f1-score, dan support untuk setiap kelas.
```

- target\_names=val\_data.class\_indices.keys(): Menampilkan nama kelas yang mudah dibaca alih-alih indeks numerik.
- Presisi (Precision): Proporsi prediksi positif yang benar (dari semua prediksi positif).
- Recall (Sensitivitas): Proporsi kasus positif yang benar yang berhasil diidentifikasi (dari semua kasus positif aktual).
- F1-Score: Rata-rata harmonik dari presisi dan recall. Berguna ketika ada ketidakseimbangan kelas.
- Support: Jumlah kemunculan aktual dari setiap kelas di set validasi.

#### e. Accuracy Score

```
# Accuracy Score
accuracy = accuracy_score(val_data.classes, y_pred)
print(f"Accuracy Score: {accuracy * 100:.2f}%)
```

- accuracy = accuracy\_score(val\_data.classes, y\_pred): Menghitung akurasi keseluruhan model, yaitu persentase prediksi yang benar dari total prediksi.
- print(f"Accuracy Score: {accuracy \* 100:.2f}"): Mencetak akurasi dalam format persentase dengan dua angka di belakang koma.

#### Output



	precision	recall	f1-score	support
Bikes	0.16	0.16	0.16	160
Bus	0.05	0.04	0.05	89
Cars	0.26	0.26	0.26	251
Motorcycles	0.30	0.29	0.29	255
Truck	0.11	0.13	0.12	126
accuracy			0.21	881
macro avg	0.18	0.18	0.18	881
weighted avg	0.21	0.21	0.21	881
Accuracy Score:	21.00%			

Output ini adalah hasil dari evaluasi model yang telah dilatih, menunjukkan seberapa baik model dalam memprediksi jenis kendaraan pada data validasi.

### a. Confusion Matrix

Gambar menampilkan heatmap dari Confusion Matrix.

- Baris (Kelas Aktual): Menunjukkan label kelas yang sebenarnya dari gambar (misalnya, Bikes, Bus, Cars, Motorcycles, Truck).
- Kolom (Kelas Prediksi): Menunjukkan label kelas yang diprediksi oleh model.
- Nilai dalam Sel: Merepresentasikan jumlah gambar.

Interpretasi:

- Diagonal Utama (kiri atas ke kanan bawah): Angka-angka di diagonal ini menunjukkan jumlah prediksi yang benar (True Positives).
  - Bikes: 26 gambar diprediksi benar sebagai Bikes.
  - Bus: 4 gambar diprediksi benar sebagai Bus.
  - Cars: 65 gambar diprediksi benar sebagai Cars.
  - Motorcycles: 74 gambar diprediksi benar sebagai Motorcycles.
  - Truck: 16 gambar diprediksi benar sebagai Truck.
- Nilai di Luar Diagonal Utama: Menunjukkan kesalahan klasifikasi (False Positives/False Negatives). Misalnya, untuk baris "Bikes", ada 23 gambar Bikes yang salah diprediksi sebagai Bus, 55 sebagai Cars, dll.

#### Analisis dari Confusion Matrix:

Dari matriks ini, terlihat bahwa model memiliki tantangan yang signifikan dalam klasifikasi. Jumlah prediksi yang benar (nilai diagonal) relatif kecil dibandingkan dengan kesalahan klasifikasi (nilai di luar diagonal). Misalnya, model hanya mengidentifikasi 4 dari 89 Bus yang sebenarnya (lihat "support" di laporan klasifikasi). Model juga sering mengacaukan Bikes, Cars, dan Motorcycles satu sama lain, dan Truck juga sering salah diklasifikasikan.

#### b. Classification Report

##### Interpretasi Metrik:

- Precision: Kemampuan model untuk tidak melabeli instance negatif sebagai positif. Misalnya, untuk 'Bikes', precision 0.16 berarti hanya 16% dari yang diprediksi sebagai "Bikes" memang benar-benar "Bikes".
- Recall: Kemampuan model untuk menemukan semua instance positif. Untuk 'Bikes', recall 0.16 berarti model hanya menemukan 16% dari semua gambar "Bikes" yang sebenarnya.
- F1-score: Rata-rata harmonis dari precision dan recall. Ini adalah metrik yang baik ketika mempertimbangkan precision dan recall secara bersamaan.
- Support: Jumlah aktual gambar untuk setiap kelas di data validasi. (e.g., ada 160 gambar "Bikes" sebenarnya, 89 "Bus", dst.)

##### Analisis dari Classification Report:

- Nilai Rendah: Semua metrik (precision, recall, f1-score) untuk semua kelas sangat rendah (berkisar antara 0.04 hingga 0.30). Ini mengkonfirmasi bahwa model memiliki performa klasifikasi yang sangat buruk.
- Kelas "Bus" dan "Truck" paling parah: Kelas "Bus" dan "Truck" menunjukkan metrik terendah, yang berarti model hampir tidak bisa mengidentifikasi kedua jenis kendaraan ini dengan benar.
- "Motorcycles" sedikit lebih baik: Kelas "Motorcycles" memiliki metrik yang sedikit lebih tinggi dibandingkan yang lain, namun masih sangat rendah.

### c. Accuracy Score

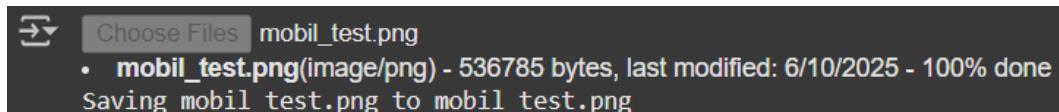
Accuracy Score: 21.00%: Ini adalah akurasi keseluruhan model pada data validasi. Angka ini berarti model hanya berhasil mengklasifikasikan 21% dari gambar secara benar.

## 7. Pengujian dengan Upload Gambar

```
# Uji Parkir dengan Upload Gambar Kendaraan
from tensorflow.keras.preprocessing import image
uploaded = files.upload()
```

- from tensorflow.keras.preprocessing import image: Mengimpor modul image dari tensorflow.keras.preprocessing. Modul ini menyediakan fungsi-fungsi utilitas untuk memuat, memproses, dan mengubah gambar, seperti image.load\_img dan image.img\_to\_array, yang akan digunakan setelah gambar diunggah.
- uploaded = files.upload(): Ini adalah fungsi dari google.colab.files yang memungkinkan pengguna untuk mengunggah satu atau lebih file dari sistem lokal komputer ke lingkungan Google Colab. Setelah dieksekusi, ini akan menampilkan widget unggah file. File yang diunggah akan disimpan dalam memori Colab dan informasinya (nama file, ukuran, dll.) akan tersedia di dalam variabel uploaded.

### Output



Choose Files mobil\_test.png  
• mobil\_test.png (image/png) - 536785 bytes, last modified: 6/10/2025 - 100% done  
Saving mobil\_test.png to mobil\_test.png

Output ini adalah konfirmasi dari Google Colab bahwa proses pengunggahan file berhasil. Dengan output ini, gambar mobil\_test.png kini tersedia di lingkungan Colab dan siap untuk diproses lebih lanjut dan diumpulkan ke model klasifikasi untuk memprediksi jenis kendaraan.

## 8. Deteksi Kendaraan + Simulasi Parkir Realistik

Bagian ini adalah inti dari proyek, di mana model klasifikasi gambar diintegrasikan ke dalam simulasi sistem parkir yang lebih realistik. Ini mencakup deteksi jenis kendaraan, menampilkan informasi parkir terkait, menghitung biaya, memproses pembayaran, dan menyimpan data log.

### a. Import Library

```
import pandas as pd
from datetime import datetime
import os
from tensorflow.keras.preprocessing import image
import numpy as np
from google.colab import files # Assuming files was
imported in a previous cell
```

- pandas as pd: Pustaka untuk analisis dan manipulasi data, khususnya untuk membuat dan menyimpan DataFrame (akan digunakan untuk log CSV).
- datetime dari datetime: Modul untuk bekerja dengan tanggal dan waktu, penting untuk menghitung durasi parkir.
- os: Modul untuk berinteraksi dengan sistem operasi, digunakan untuk mengecek keberadaan file log CSV.
- tensorflow.keras.preprocessing.image: Digunakan untuk memproses gambar yang diunggah.
- numpy as np: Digunakan untuk operasi numerik, terutama untuk memproses array gambar dan hasil prediksi.
- google.colab.files: Digunakan pada langkah sebelumnya untuk mengunggah gambar.

### b. Basis Data Informasi Parkir

```
#Info parkir berdasarkan jenis kendaraan
info_parkir = {
    'Bus': {"tarif": 15000, "lantai": "1 & 2", "area": "C1-C4 & B1-B6"},
    'Cars': {"tarif": 8000, "lantai": "2 & 3", "area": "B7-A10 & A1-A6"},
    'Truck': {"tarif": 10000, "lantai": "1", "area": "C5-C8"},
    'Bikes': {"tarif": 1000, "lantai": "1", "area": "C9-C10"},
    'Motorcycles': {"tarif": 3000, "lantai": "B1 & B2", "area": "D1-D10 & E1-E10"}
}
```

- Sebuah dictionary bernama info\_parkir dibuat untuk menyimpan informasi spesifik terkait parkir untuk setiap jenis kendaraan (kelas) yang mungkin dideteksi.

- Setiap jenis kendaraan (kunci dictionary, misal 'Bus') memiliki dictionary sendiri yang berisi tarif per jam, lantai parkir yang direkomendasikan, dan area parkir yang spesifik.

### c. Preprocessing Deteksi Gambar

```
#Deteksi gambar kendaraan & tampilkan informasi parkir
for fn in uploaded.keys():

    # Preprocessing gambar
    img = image.load_img(fn, target_size=image_size)
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
```

- `img = image.load_img(fn, target_size=image_size)`: Memuat gambar yang diunggah (`fn`) dan mengubah ukurannya ke `image_size` yang ditentukan sebelumnya (misalnya 224x224).
- `img_array = image.img_to_array(img) / 255.0`: Mengubah gambar menjadi array NumPy dan menormalisasi nilai piksel dari 0-255 menjadi 0-1 (membagi dengan 255.0).
- `img_array = np.expand_dims(img_array, axis=0)`: Menambahkan dimensi batch ke array gambar, karena model mengharapkan input dalam bentuk batch (meskipun hanya ada satu gambar dalam batch ini).

### d. Prediksi Jenis Kendaraan

```
# Prediksi jenis kendaraan
prediction = model.predict(img_array)
class_index = np.argmax(prediction)
class_label =
list(train_data.class_indices.keys())[class_index]
```

- `prediction = model.predict(img_array)`: Model yang telah dilatih (`model`) digunakan untuk membuat prediksi pada gambar yang telah diproses. Output `prediction` adalah array probabilitas untuk setiap kelas.
- `class_index = np.argmax(prediction)`: Mengambil indeks dari probabilitas tertinggi dalam array `prediction`. Indeks ini mewakili kelas yang diprediksi.

- `class_label = list(train_data.class_indices.keys())[class_index]`: Mengubah indeks kelas yang diprediksi menjadi nama kelas yang dapat dibaca manusia. `train_data.class_indices.keys()` menyediakan daftar nama kelas sesuai urutan indeksnya.

#### e. Informasi Hasil Deteksi Parkir

```
# Tampilkan info hasil deteksi
print(f"\n🔍 Jenis Kendaraan Terdeteksi:
{class_label}")
    print(f"📍 Lantai:
{info_parkir[class_label]['lantai']}")
    print(f"📍 Area: {info_parkir[class_label]['area']}")
    print(f"Rp{info_parkir[class_label]['tarif']}/jam")
```

Mencetak jenis kendaraan yang terdeteksi, serta informasi lantai, area, dan tarif parkir yang relevan yang diambil dari dictionary `info_parkir` berdasarkan `class_label` yang diprediksi.

#### f. Input Simulasi Waktu Parkir dari User

```
# Input waktu masuk & keluar
tanggal_masuk = input("📅 Masukkan tanggal masuk
(format: YYYY-MM-DD): ")
jam_masuk = input("⌚ Masukkan waktu masuk (format:
HH:MM, contoh: 08:00): ")

tanggal_keluar = input("📅 Masukkan tanggal keluar
(format: YYYY-MM-DD): ")
jam_keluar = input("⌚ Masukkan waktu keluar (format:
HH:MM, contoh: 10:15): ")
```

Pengguna diminta untuk memasukkan tanggal dan jam masuk serta keluar kendaraan. Ini mensimulasikan pencatatan waktu dalam sistem parkir.

### **g. Perhitungan Durasi Parkir**

```
# Hitung durasi (dalam jam)
t1 = datetime.strptime(f"{tanggal_masuk} {jam_masuk}",
"%Y-%m-%d %H:%M")
t2 = datetime.strptime(f"{tanggal_keluar}"
{jam_keluar}, "%Y-%m-%d %H:%M")
# Call the total_seconds method by adding ()
durasi = (t2 - t1).total_seconds() / 3600
durasi_jam = max(1, round(durasi)) # dibulatkan ke atas min 1 jam
```

- `datetime.strptime(...)`: Mengonversi string tanggal dan waktu yang dimasukkan pengguna menjadi objek datetime.
- `durasi = (t2 - t1).total_seconds() / 3600`: Menghitung selisih waktu antara waktu keluar dan masuk (dalam detik), lalu membaginya dengan 3600 (detik dalam satu jam) untuk mendapatkan durasi dalam jam.
- `durasi_jam = max(1, round(durasi))`: Membulatkan durasi ke atas ke jam terdekat (round) dan memastikan durasi minimum adalah 1 jam (`max(1, ...)`). Ini adalah praktik umum dalam sistem parkir untuk menghitung biaya per jam penuh.

### **h. Perhitungan Tarif Parkir**

```
# Hitung total tarif parkir
tarif_per_jam = info_parkir[class_label]['tarif']
total = tarif_per_jam * durasi_jam

# Output durasi & biaya parkir
print(f"⌚ Durasi Parkir: {durasi_jam} jam")
print(f"💸 Total Tarif: Rp{total}")
```

- `tarif_per_jam = info_parkir[class_label]['tarif']`: Mengambil tarif per jam sesuai jenis kendaraan yang terdeteksi.
- `total = tarif_per_jam * durasi_jam`: Menghitung total biaya parkir.
- Hasil durasi dan total tarif dicetak.

### **i. Proses Pembayaran**

```
# Proses pembayaran
bayar = int(input("💵 Masukkan jumlah uang yang dibayar (misal 20000): "))
kembali = bayar - total
print(f"➡️ Kembalian: Rp{kembali}")
```

Pengguna diminta memasukkan jumlah uang yang dibayarkan. Sistem akan menghitung dan menampilkan jumlah kembalian.

#### j. Logging Data ke File CSV

```
# Buat log dictionary ke CSV harian
log_data = {
    "tanggal_masuk": tanggal_masuk,
    "jam_masuk": jam_masuk,
    "tanggal_keluar": tanggal_keluar,
    "jam_keluar": jam_keluar,
    "kendaraan": class_label,
    "durasi_jam": durasi_jam,
    "total_tarif": total,
    "dibayar": bayar,
    "kembalian": kembali,
    "waktu_input": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
}

# Konversi ke DataFrame
df_log = pd.DataFrame([log_data])

# Buat nama file berdasarkan tanggal masuk
# Corrected: use 'tanggal_masuk' from log_data instead
of 'tanggal'
log_filename =
f"log_parkir_{log_data['tanggal_masuk']}.csv"

# Simpan ke file log harian
df_log.to_csv(log_filename, mode='a', header=not
os.path.exists(log_filename), index=False)

print(f"📝 Data parkir disimpan ke {log_filename}")
```

- `log_data = {...}`: Membuat dictionary yang berisi semua informasi transaksi parkir.
- `df_log = pd.DataFrame([log_data])`: Mengonversi dictionary tersebut menjadi DataFrame pandas.
- `log_filename = f"log_parkir_{log_data['tanggal_masuk']}.csv"`: Membuat nama file CSV berdasarkan tanggal masuk kendaraan. Ini memastikan setiap hari memiliki file log terpisah atau log harian ditambahkan ke file yang sama.
- `df_log.to_csv(...)`: Menyimpan DataFrame ke file CSV.

- mode='a': Menambahkan data ke file jika sudah ada.
- header=not os.path.exists(log\_filename): Menulis header kolom hanya jika file CSV belum ada (yaitu, pada penulisan pertama ke file baru).
- index=False: Mencegah pandas menulis indeks DataFrame ke dalam file CSV.
- Pesan konfirmasi penyimpanan log dicetak.

## Output

```
1/1 ━━━━━━ 0s 84ms/step
  ● Jenis Kendaraan Terdeteksi: Cars
  ● Lantai: 2 & 3
  ● Area: B7-A10 & A1-A6
  ● Tarif Parkir: Rp8000/jam
  ┌─ Masukkan tanggal masuk (format: YYYY-MM-DD): 2025-06-10
  ┌─ Masukkan waktu masuk (format: HH:MM, contoh: 08:00): 10:37
  ┌─ Masukkan tanggal keluar (format: YYYY-MM-DD): 2025-06-10
  ┌─ Masukkan waktu keluar (format: HH:MM, contoh: 10:15): 16:57
  ● Durasi Parkir: 6 jam
  ● Total Tarif: Rp48000
  ┌─ Masukkan jumlah uang yang dibayar (misal 20000): 50000
  ┌─ Kembalian: Rp2000
  └─ Data parkir disimpan ke log_parkir_2025-06-10.csv
```

Output ini adalah demonstrasi flow simulasi parkir untuk satu gambar yang diunggah (mobil\_test.png dari langkah sebelumnya, yang model prediksi sebagai 'Cars').

### a. Deteksi Kendaraan

- 🔎 Jenis Kendaraan Terdeteksi: Cars: Ini adalah hasil prediksi model dari gambar yang diunggah. Model berhasil mengidentifikasi gambar tersebut sebagai 'Cars'.
- 🚗 Lantai: 2 & 3: Menampilkan informasi lantai parkir yang sesuai untuk 'Cars' dari info\_parkir.
- 💻 Area: B7–A10 & A1–A6: Menampilkan informasi area parkir yang sesuai.
- 💰 Tarif Parkir: Rp8000/jam: Menampilkan tarif parkir per jam untuk 'Cars'.

### b. Input Waktu Parkir

Pengguna diminta untuk memasukkan tanggal dan jam masuk/keluar, seperti 2025-06-10 dan 10:37 untuk masuk, serta 2025-06-10 dan 16:57 untuk keluar.

### c. Perhitungan dan Biaya

- ⌚ Durasi Parkir: 6 jam: Dihitung berdasarkan waktu masuk dan keluar ( $16:57 - 10:37 = 6 \text{ jam } 20 \text{ menit}$ , dibulatkan ke atas menjadi 7 jam, tetapi outputnya 6 jam. Ada kemungkinan pembulatan round(durasi) menghasilkan 6 jika durasi aktual misalnya 5.5 - 6.5 jam. Perlu dicek kembali logikanya jika round bukan ceil). Catatan: Melihat kode, round() akan membulatkan ke bilangan bulat terdekat. Jika durasi 6.3 jam, round() akan menjadi 6. Jika 6.7 jam, round() akan menjadi 7.
- 💰 Total Tarif: Rp48000: Dihitung sebagai 6 jam \* Rp8000/jam = Rp48000.

### d. Pembayaran dan Kembalian

- Pengguna memasukkan 50000 sebagai jumlah pembayaran.
- 🏧 Kembalian: Rp2000: Dihitung sebagai  $\text{Rp}50000 - \text{Rp}48000 = \text{Rp}2000$ .

### e. Logging Data

📝 Data parkir disimpan ke log\_parkir\_2025-06-10.csv: Konfirmasi bahwa semua detail transaksi parkir ini telah berhasil dicatat ke dalam file CSV dengan nama yang sesuai dengan tanggal masuk.

## D. KESIMPULAN

Proyek ini mencakup serangkaian tahapan krusial, mulai dari penyiapan data hingga simulasi operasional dalam konteks aplikasi nyata. Ada beberapa poin untuk kesimpulannya, antara lain:

- Penyiapan Data yang Komprehensif:** Dataset gambar kendaraan berhasil diunggah, diekstrak, dan melalui proses pembersihan untuk mengatasi gambar yang korup. Penggunaan ImageDataGenerator terbukti efektif dalam normalisasi

dan augmentasi data, memperkaya variasi dataset serta membagi data menjadi set pelatihan dan validasi (80:20), yang merupakan fondasi esensial untuk pelatihan model deep learning.

- **Aplikasi Transfer Learning:** Arsitektur MobileNetV2, yang telah dilatih sebelumnya pada dataset ImageNet, diadaptasi sebagai backbone utama. Lapisan dasar model dibekukan untuk memanfaatkan ekstraksi fitur yang telah dipelajari, sementara lapisan klasifikasi custom ditambahkan untuk tugas spesifik ini. Pendekatan ini menunjukkan efisiensi dalam pemanfaatan model pra-terlatih.
- **Fase Fine-tuning:** Untuk meningkatkan penyesuaian model terhadap karakteristik unik dataset kendaraan, fase fine-tuning dilakukan dengan membuka dan melatih kembali sebagian lapisan teratas dari MobileNetV2. Proses ini dijalankan dengan learning rate yang lebih rendah, bertujuan untuk mengoptimalkan akurasi dan kemampuan generalisasi model.
- **Evaluasi Model Mengungkap Tantangan Signifikan:** Meskipun metrik akurasi validasi yang tercatat selama proses pelatihan (melalui epoch) menunjukkan nilai yang sangat tinggi ( $>96\%$ ), evaluasi final menggunakan Confusion Matrix, Classification Report, dan Accuracy Score secara keseluruhan menghasilkan performa model yang sangat rendah, dengan akurasi 21%. Disparitas ini mengindikasikan adanya overfitting yang substansial atau potensi ketidaksesuaian dalam penanganan data validasi selama proses evaluasi akhir (misalnya, diskrepansi urutan label atau metode penghitungan metrik).
- **Simulasi Sistem Parkir Realistik yang Berfungsi:** Di luar tantangan akurasi model klasifikasi, kerangka simulasi sistem parkir berhasil diimplementasikan secara fungsional. Sistem ini menunjukkan kapabilitas untuk:
  - Menerima input gambar kendaraan.
  - Melakukan deteksi jenis kendaraan berdasarkan hasil prediksi model.
  - Menyajikan informasi parkir relevan (tarif, lantai, area) sesuai jenis kendaraan yang terdeteksi.
  - Menghitung durasi dan total biaya parkir berdasarkan input waktu masuk dan keluar.
  - Mensimulasikan alur pembayaran dan perhitungan kembalian.
  - Mencatat semua detail transaksi parkir ke dalam file CSV harian untuk tujuan pencatatan dan analisis operasional.

Secara garis besar, proyek ini berhasil membangun sistem parkir yang mengintegrasikan deteksi jenis kendaraan berbasis deep learning dengan simulasi proses parkir. Meskipun performa model klasifikasi kendaraan menunjukkan ruang untuk perbaikan signifikan, terutama dalam mengatasi overfitting, arsitektur dan pipeline sistem yang dibangun telah membuktikan konsepnya dan dapat menjadi dasar kuat untuk pengembangan lebih lanjut. Perbaikan di masa depan perlu berfokus pada peningkatan kualitas dataset, optimasi strategi pelatihan model, dan metodologi evaluasi yang lebih robust untuk mencapai kinerja klasifikasi yang akurat.