

KLASIFIKASI GAMBAR SAYURAN MENGGUNAKAN CONVOLUTIONAL NEURAL NETWORK (CNN)

Dibuat Untuk Memenuhi Tugas Pertemuan 10 Mata Kuliah Sistem Cerdas



Disusun Oleh :

Muhamad Aksyal Faiz Destian (2210631250019)

Farrel Zaindri Althaf (2210631250050)

Muhammad Dzakil Aqli (2210631250062)

Kelas: 6A Sistem Informasi

Dosen Pengampu: Taufik Ridwan M.Kom

**PROGAM STUDI SISTEM INFORMASI
FAKULTAS ILMU KOMPUTER
UNIVERSITAS SINGAPERBANGSA KARAWANG
2025/2026**

A. Studi Kasus

1. Deskripsi Kasus

Pada era digital saat ini, pemanfaatan teknologi Kecerdasan Buatan (AI) semakin luas, termasuk dalam bidang pertanian dan logistik hasil pertanian. Salah satu tantangan yang sering dihadapi adalah proses identifikasi dan klasifikasi jenis sayuran secara cepat dan akurat, terutama pada skala besar seperti distribusi ke supermarket atau platform e-commerce.

Studi kasus ini berfokus pada pembangunan sistem klasifikasi otomatis terhadap gambar sayuran menggunakan metode Convolutional Neural Network (CNN). Sistem ini dirancang untuk mengenali 15 jenis sayuran berbeda berdasarkan gambar, dengan tujuan mengurangi ketergantungan pada pemeriksaan manual yang memakan waktu dan rawan kesalahan.

Melalui pendekatan ini, pengguna (seperti petani, pelaku usaha, atau sistem digital) dapat mengunggah gambar sayuran, lalu sistem akan secara otomatis memprediksi jenis sayuran tersebut. Hal ini dapat digunakan untuk sistem inventaris otomatis, pengecekan kualitas, atau bahkan aplikasi edukatif.

2. Dataset

Dataset yang digunakan dalam proyek ini merupakan dataset gambar sayuran yang bersifat publik dan telah tersedia dalam struktur direktori yang mendukung klasifikasi gambar menggunakan CNN. Dataset ini terdiri dari 15 kelas sayuran, antara lain:

- Bean
- Bitter_Gourd
- Bottle_Gourd
- Brinjal
- Broccoli
- Cabbage
- Capsicum
- Carrot
- Cauliflower
- Cucumber
- Papaya
- Potato
- Pumpkin
- Radish
- Tomato

Rincian Dataset:

Nama Dataset: Vegetable Image Dataset

Sumber Dataset: Kaggle

- **Total kelas:** 15 jenis sayuran
- **Total gambar:** ±21.000
- **Pembagian data:**
 - **Data Latih (Training):** 15.000 gambar (1.000 gambar per kelas)
 - **Data Validasi:** 3.000 gambar (200 gambar per kelas)

- **Data Uji (Testing):** 3.000 gambar (200 gambar per kelas)

Format Gambar:

- **Format file:** .jpg
- **Ukuran:** 128x128 piksel (dirubah saat pre-processing)
- **Warna:** RGB

3. Tools

Untuk membangun dan melatih model klasifikasi gambar sayuran menggunakan CNN, proyek ini menggunakan berbagai tools dan library yang umum digunakan dalam pengembangan sistem berbasis Kecerdasan Buatan (AI), khususnya di bidang Computer Vision. Berikut adalah tools yang digunakan:

a. Bahasa Pemrograman

- **Python**
Digunakan sebagai bahasa utama dalam seluruh proses mulai dari pemrosesan data, pembuatan model CNN, hingga evaluasi hasil.

b. Library dan Framework

- **TensorFlow & Keras**
Digunakan untuk membangun model CNN dan menjalankan proses pelatihan, validasi, serta evaluasi model. Keras digunakan sebagai antarmuka tingkat tinggi yang memudahkan pembangunan model deep learning.
- **NumPy**
Untuk manipulasi array dan operasi numerik dasar.
- **Matplotlib & Seaborn**
Untuk visualisasi hasil seperti grafik akurasi/loss dan confusion matrix.
- **Scikit-learn**
Digunakan untuk pembuatan confusion matrix dan evaluasi metrik performa lainnya.

c. Lingkungan Pengembangan

- **Google Colab**
Digunakan sebagai platform pemrograman berbasis cloud yang mendukung eksekusi notebook Python dengan akses GPU (jika tersedia), yang sangat membantu dalam mempercepat proses training model deep learning.

d. GitHub

Digunakan sebagai repositori untuk menyimpan seluruh file proyek, termasuk:

- Kode program (notebook .ipynb)
- Dataset (jika diperlukan)
- Laporan dan dokumentasi
- Hasil evaluasi dan grafik

B. Implementasi Studi Kasus

1. Import database dari Kaggle

```
import kagglehub
misrahmed_vegetable_image_dataset_path = kagglehub.dataset_download('misrahmed/vegetable-image-dataset')

print('Data source import complete.')
```

Data source import complete.

Project ini diawali dengan perintah `import kagglehub`, yang berfungsi memuat pustaka resmi Kaggle ke dalam sesi Python. Pustaka ini menyediakan jembatan vital yang memungkinkan interaksi terprogram dengan repositori data dan model masif yang tersedia di Kaggle. Setelah pustaka siap digunakan, skrip mengeksekusi perintah utamanya dengan memanggil fungsi `kagglehub.dataset_download()`. Fungsi ini secara spesifik menargetkan dataset yang diinginkan melalui argumen `'misrahmed/vegetable-image-dataset'`, yang merupakan pengenal unik di mana `misrahmed` adalah nama pengguna pemilik dan `vegetable-image-dataset` adalah nama datasetnya. Setelah proses pengunduhan yang bisa memakan waktu beberapa saat selesai, fungsi tersebut akan mengembalikan *path* atau alamat direktori lokal tempat file-file dataset tersebut disimpan. Alamat ini kemudian ditampung dalam sebuah variabel bernama `misrahmed_vegetable_image_dataset_path`, yang menjadi kunci untuk mengakses data tersebut pada langkah selanjutnya. Sebagai penutup dan untuk memberikan umpan balik yang jelas kepada pengguna, skrip mencetak pesan konfirmasi 'Data source import complete.', menandakan bahwa sumber data telah berhasil diimpor dan siap untuk tahap pengolahan berikutnya.

2. Penyiapan Dataset

▪ Penyiapan Direktori dan Data

```
[3] import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random
```

```
# inisiasi path data gambar
images_path = "/kaggle/input/vegetable-image-dataset/Vegetable Images"
train_path = "/train"
val_path = "/validation"
test_path = "/test"
```

Daftar nama subdirektori:

- Broccoli
- Capsicum
- Bottle_Gourd
- Radish
- Tomato
- Brinjal
- Pumpkin
- Carrot
- Papaya
- Cabbage
- Bitter_Gourd
- Cauliflower
- Bean
- Cucumber
- Potato

Pada bagian pertama, skrip mengimpor empat pustaka (library) Python yang esensial. Pustaka *os* diimpor untuk menyediakan fungsi-fungsi yang dapat berinteraksi dengan sistem operasi, yang nantinya sangat berguna untuk mengelola struktur file dan direktori. Selanjutnya, *matplotlib.pyplot* diimpor dengan alias umum *plt*, yang merupakan pustaka standar untuk membuat visualisasi data seperti grafik dan plot; ini bisa digunakan untuk menampilkan contoh gambar atau grafik akurasi model. Bersamaan dengan itu, *matplotlib.image* diimpor dengan alias *mpimg* yang berfungsi secara spesifik untuk membaca data gambar dari file dan mengubahnya menjadi format array yang dapat diolah. Terakhir, pustaka *random* diimpor untuk menyediakan fungsionalitas terkait pengacakan, misalnya untuk memilih beberapa gambar secara acak dari dataset sebagai sampel.

Bagian kedua berfokus pada pendefinisian alamat atau *path* yang akan digunakan di seluruh proyek. Variabel *images_path* diinisialisasi dengan path absolut `"/kaggle/input/vegetable-image-dataset/Vegetable Images"`. Ini adalah lokasi spesifik di dalam lingkungan Kaggle di mana semua folder gambar (seperti 'Broccoli', 'Carrot', dll.) dari dataset yang diunduh sebelumnya berada. Setelah itu, tiga variabel lainnya—*train_path*, *val_path*, dan *test_path*—didefinisikan dengan nama-nama direktori relatif (`"/train"`, `"/validation"`, `"/test"`). Path-path ini bukanlah lokasi data sumber, melainkan representasi nama untuk folder-folder yang nantinya akan dibuat atau digunakan untuk memisahkan dataset menjadi tiga bagian: data untuk melatih model (*training set*), data untuk validasi selama pelatihan (*validation set*), dan data untuk pengujian akhir model (*test set*).

```
# fungsi untuk menampilkan gambar secara acak pada grid n_row x n_col
def view_random_image(data_dir, class_dir, n_row=1, n_col=1):

    # menentukan direktori gambar
    target_dir = data_dir + "/" + class_dir + "/"

    # memilih sebanyak n_row*n_col gambar secara acak
    rand_images = random.sample(os.listdir(target_dir), n_row*n_col)

    fig, axs = plt.subplots(n_row, n_col)

    # menampilkan gambar dalam bentuk grid ukuran n_row*n_col
    for i, ax in enumerate(axs.flat):
        img = mpimg.imread(target_dir + rand_images[i])
        ax.imshow(img)
        ax.set_title(f'{class_dir} {i+1}\n{img.shape}')
        ax.axis("off")

    plt.tight_layout()
    plt.show()

# menampilkan contoh beberapa gambar acak dari direktori train
# Pass the full training path to the function
view_random_image(full_train_path, "Bean", 1, 3)
view_random_image(full_train_path, "Cucumber", 1, 3)
```





Skrip Python ini dijalankan guna membuat fungsi utilitas bernama `view_random_image` yang secara spesifik dirancang untuk menampilkan sampel gambar acak dari direktori kelas tertentu dalam format grid. Fungsi ini bekerja secara dinamis dengan membangun path direktori target, memilih sejumlah file gambar secara acak, lalu menggunakan pustaka `matplotlib` untuk menyajikannya dalam sebuah visualisasi yang jelas dan informatif. Skrip tersebut kemudian langsung mendemonstrasikan penggunaannya dengan memanggil fungsi ini dua kali: pertama untuk menampilkan tiga gambar acak dari kelas "Bean", dan kedua untuk kelas "Cucumber", yang keduanya diambil dari direktori data latih.

Hasil dari eksekusi ini adalah sebuah representasi visual yang berfungsi sebagai bukti validasi data. Dalam visualisasi tersebut, tampak jelas baris pertama berisi sampel "Bean" dan baris kedua berisi sampel "Cucumber". Elemen informasi yang paling signifikan adalah notasi `(224, 224, 3)` yang tertera pada setiap gambar, yang mengonfirmasi bahwa setiap citra telah diproses menjadi format standar dengan resolusi tinggi 224 piksel, lebar 224 piksel, dan kedalaman 3 kanal warna (RGB). Keseragaman dimensi ini bukanlah suatu kebetulan, melainkan indikasi kuat bahwa dataset telah melalui tahap pra-pemrosesan (*preprocessing*) untuk menyamakan ukuran seluruh gambar. Standardisasi input semacam ini merupakan prasyarat fundamental bagi sebagian besar arsitektur *deep learning*. Dengan demikian, keseluruhan proses—mulai dari pembuatan fungsi hingga analisis outputnya—secara efektif tidak hanya mengonfirmasi bahwa data telah dimuat dengan benar sesuai kategorinya, tetapi juga memberikan wawasan krusial bahwa data tersebut telah siap secara dimensional untuk diolah lebih lanjut oleh model.

- **Image Data Generator**

```
[ ] import tensorflow as tf
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
[ ] tf.random.set_seed(42)

# pengaturan data training dengan scaling images saja
# train_datagen = ImageDataGenerator(rescale=1.0 / 255)

# pengaturan data training dengan berbagai augmentasi
train_datagen = ImageDataGenerator(rescale=1.0 / 255,
                                   rotation_range=20,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')

# pengaturan data validasi selama pelatihan
val_datagen = ImageDataGenerator(rescale=1.0 / 255)

# pengaturan data uji untuk evaluasi model
test_datagen = ImageDataGenerator(rescale=1.0 / 255)

# menentukan ukuran gambar untuk proses pelatihan
img_size = (224, 224)

# menyiapkan data training dengan pengaturan yang sudah ditentukan
train_data = train_datagen.flow_from_directory(images_path + train_path,
                                              target_size=img_size,
                                              batch_size=32,
                                              class_mode='categorical',
                                              seed=42)

# menyiapkan data validasi
val_data = val_datagen.flow_from_directory(images_path + val_path,
                                          target_size=img_size,
                                          batch_size=32,
                                          class_mode='categorical',
                                          seed=42)

# menyiapkan data uji
test_data = test_datagen.flow_from_directory(images_path + test_path,
                                           target_size=img_size,
                                           batch_size=32,
                                           class_mode='categorical',
                                           shuffle=False)
```

```
Found 15000 images belonging to 15 classes.
Found 3000 images belonging to 15 classes.
Found 3000 images belonging to 15 classes.
```

Skrup kode ini menjalankan tahap krusial dalam persiapan dataset untuk melatih model klasifikasi gambar menggunakan TensorFlow, yang merupakan sebuah platform *open-source* komprehensif dari Google untuk membangun dan melatih model *machine learning* kompleks, terutama jaringan saraf tiruan (*deep learning*). Proses ini dimulai dengan memanfaatkan salah satu kelas intinya, `ImageDataGenerator`. Proses ini dimulai dengan mendefinisikan tiga generator data yang berbeda untuk set data latih, validasi, dan uji. Untuk data latih, diterapkan dua teknik utama: **normalisasi** nilai piksel dengan `rescale=1.0/255` dan serangkaian teknik **augmentasi data** seperti rotasi, pergeseran, dan zoom untuk memperkaya variasi data secara artifisial, yang sangat penting untuk membuat model lebih tangguh (*robust*) dan mengurangi risiko *overfitting*. Sebaliknya, untuk data validasi dan uji, hanya proses normalisasi yang diterapkan tanpa augmentasi, karena kedua set data ini berfungsi sebagai tolok ukur untuk mengevaluasi performa model terhadap data asli.

Setelah konfigurasi generator selesai, metode `.flow_from_directory()` dipanggil untuk setiap set guna membuat *iterator* yang akan mengalirkan gambar dari direktori secara efisien. Metode ini secara otomatis memuat gambar, mengubah ukurannya menjadi dimensi seragam 224x224 piksel, mengelompokkannya ke dalam *batch* berisi 32 gambar, serta secara cerdas mengidentifikasi sub-folder sebagai label kelas dan mengubahnya menjadi format *one-hot encoding* melalui `class_mode='categorical'`. Khusus untuk data uji, pengacakan (*shuffle*) dinonaktifkan untuk menjaga urutan data demi kemudahan evaluasi hasil prediksi. Dengan demikian, kode ini secara komprehensif menangani seluruh alur persiapan data, mulai dari normalisasi dan augmentasi hingga pembuatan *batch* data yang siap untuk dimasukkan ke dalam model.

```
# menampilkan jumlah batch pada data train
print(f"Jumlah batch (training): {len(train_data)}")

# mengambil batch 1 data train
batch_1_train = train_data[0]

# setiap batch tersimpan dalam bentuk tuple
# elemen pertama menyimpan data piksel dari 32 gambar pada batch 1
batch_1_train_data = train_data[0][0]

# elemen kedua menyimpan data label dari 32 gambar pada batch 1
batch_1_train_label = train_data[0][1]

# ukuran batch = 32 (sesuai pengaturan)
print(f"Jumlah gambar pada batch 1: {len(batch_1_train_data)}")
print(f"Jumlah label pada batch 1: {len(batch_1_train_label)}")

# mengambil gambar 1 pada batch 1 data training
image_1_batch_1_train_data = train_data[0][0][0]

# mengambil label (kelas) gambar 1 pada batch 1 data training
image_1_batch_1_train_label = train_data[0][1][0]

# menampilkan data gambar 1 : (224, 224, 3)
print(f"\nBatch 1 Gambar 1 train (data):\n{image_1_batch_1_train_data}")

# menampilkan data label (kelas) gambar 1 pada batch 1 data training
print(f"\nBatch 1 Gambar 1 train (Label):\n{image_1_batch_1_train_label}")
```

```
Jumlah batch (training): 469
Jumlah gambar pada batch 1: 32
Jumlah label pada batch 1: 32

Batch 1 Gambar 1 train (data):
[[[0.49021292 0.5921737 0.7294622 ]
 [0.4901961 0.5921569 0.7294118 ]
 [0.4901961 0.5921569 0.7294118 ]
 ...
 [0.6432631 0.7297891 0.89019614]
 [0.6468762 0.7406286 0.89019614]
 [0.6504894 0.74460703 0.8936267 ]]

[[0.49387178 0.5958326 0.7404388 ]
 [0.49258798 0.5945488 0.7365875 ]
 [0.49130422 0.593265 0.7327361 ]
 ...
 [0.6445509 0.73365265 0.89019614]
 [0.6481641 0.74228173 0.89130133]
 [0.6509804 0.74669176 0.8949145 ]]

[[0.49411768 0.59607846 0.7411765 ]
 [0.49411768 0.59607846 0.7411765 ]
 [0.49411768 0.59607846 0.7411765 ]
 ...
 [0.6458388 0.73751634 0.89019614]
 [0.649452 0.7435696 0.8925893 ]
 [0.6509804 0.7492675 0.8962024 ]]

...
```

```
[[0.39611772 0.3725883 0.31768632]
 [0.3925045 0.3689751 0.31407315]
 [0.38236028 0.36536193 0.31699103]
 ...
 [0.5921569 0.7254902 0.91372555]
 [0.5921569 0.7254902 0.91372555]
 [0.5921569 0.7254902 0.91372555]]

[[0.39482984 0.37130043 0.31639847]
 [0.3893362 0.36768723 0.31466573]
 [0.3803922 0.3647059 0.3182789 ]
 ...
 [0.5921569 0.7254902 0.91372555]
 [0.5921569 0.7254902 0.91372555]
 [0.5921569 0.7254902 0.91372555]]

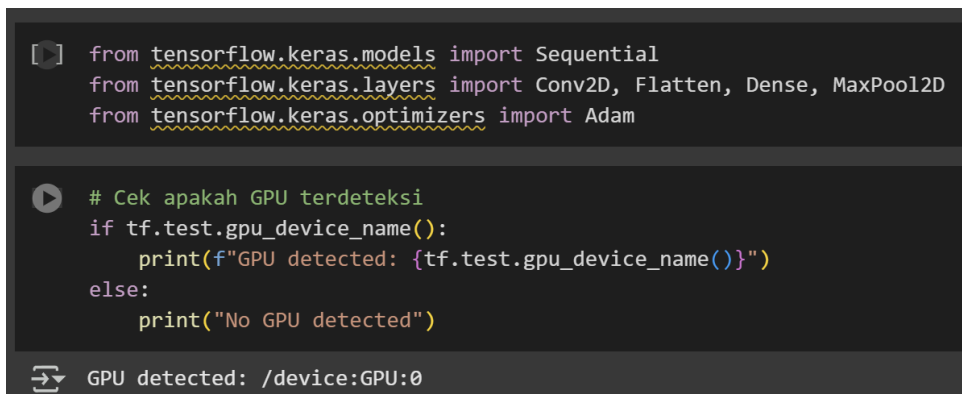
[[0.39354193 0.37001252 0.31511056]
 [0.38547257 0.36639935 0.3159536 ]
 [0.3803922 0.3647059 0.3195668 ]
 ...
 [0.5921569 0.7254902 0.91372555]
 [0.5921569 0.7254902 0.91372555]
 [0.5921569 0.7254902 0.91372555]]]

Batch 1 Gambar 1 train (Label):
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```


Setelah data dipersiapkan menggunakan ImageDataGenerator, skrip kode ini dijalankan untuk melakukan inspeksi mendalam terhadap struktur data latih (train_data) yang telah dihasilkan. Proses ini dimulai dengan mengonfirmasi jumlah total *batch*, di mana output menunjukkan ada **469 batch** tersedia. Selanjutnya, skrip mengambil *batch* pertama untuk dianalisis, dan seperti yang ditunjukkan oleh kode, setiap *batch* merupakan sebuah *tuple* yang berisi dua elemen: kumpulan data gambar dan kumpulan data label. Output memverifikasi bahwa *batch* pertama ini berisi **32 gambar dan 32 label**, sesuai dengan pengaturan *batch_size* sebelumnya. Kode kemudian menggali lebih dalam dengan mengisolasi gambar pertama beserta labelnya dari *batch* tersebut. Hasilnya menampilkan dua hal penting: pertama, data gambar ditampilkan sebagai array numerik multi-dimensi yang nilainya berada di antara 0 dan 1, yang merupakan representasi piksel dari gambar yang telah **dinormalisasi**. Kedua, data labelnya ditampilkan dalam format **one-hot encoding** (contoh: [0. 0. ... 1. ... 0.]), di mana posisi angka 1 secara unik mengidentifikasi kelas spesifik dari gambar tersebut. Dengan demikian, skrip ini secara efektif membedah dan memvalidasi output dari ImageDataGenerator, memastikan bahwa data telah di-batch, dinormalisasi, dan diberi label dengan benar sebelum dimasukkan ke dalam proses pelatihan model.

3. Implementasi Model CNN (*Convolutional Neural Network*)

- **Membuat Model**



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPool2D
from tensorflow.keras.optimizers import Adam

# Cek apakah GPU terdeteksi
if tf.test.gpu_device_name():
    print(f"GPU detected: {tf.test.gpu_device_name()}")
else:
    print("No GPU detected")

GPU detected: /device:GPU:0
```

Setelah data berhasil disiapkan dan siap dialirkan menggunakan ImageDataGenerator, langkah selanjutnya adalah memasuki tahap pembangunan model *deep learning*. Skrip kode ini memulainya dengan melakukan dua hal fundamental: mengimpor komponen arsitektur model dan memverifikasi ketersediaan perangkat keras akselerator (GPU). Pada bagian pertama, skrip mengimpor kelas-kelas esensial dari tensorflow.keras. Sequential diimpor sebagai kerangka untuk membangun model sebagai tumpukan lapisan linear. Selanjutnya, diimpor berbagai jenis lapisan yang merupakan blok pembangun inti dari sebuah *Convolutional Neural Network* (CNN): Conv2D dan MaxPool2D untuk ekstraksi fitur visual dari gambar, Flatten untuk mengubah data fitur menjadi format satu dimensi, dan Dense sebagai lapisan terhubung penuh untuk melakukan klasifikasi akhir. Selain itu, Adam diimpor sebagai algoritma *optimizer* yang populer untuk memperbarui bobot model secara efisien selama pelatihan. Mengingat proses pelatihan CNN sangat intensif secara komputasi, bagian kedua skrip melakukan verifikasi perangkat keras. Kode ini menggunakan fungsi

`tf.test.gpu_device_name()` untuk memeriksa apakah TensorFlow dapat mendeteksi GPU. Output yang dihasilkan, **GPU detected: /device:GPU:0**, memberikan konfirmasi positif bahwa sebuah GPU telah ditemukan dan siap digunakan, yang memastikan bahwa proses pelatihan model yang akan dilakukan selanjutnya dapat berjalan secara signifikan lebih cepat.

```
tf.random.set_seed(42)

model_1 = Sequential()

model_1.add(
    Conv2D(filters=20, kernel_size=(3, 3), activation="relu", input_shape=(224, 224, 3))
)
model_1.add(Conv2D(20, (3, 3), activation="relu"))
model_1.add(MaxPool2D((2, 2), padding="valid"))

model_1.add(Conv2D(50, (3, 3), activation="relu"))
model_1.add(Conv2D(50, (3, 3), activation="relu"))
model_1.add(MaxPool2D((2, 2)))

model_1.add(Flatten())
model_1.add(Dense(15, activation="softmax")) # number of neurons have to match number of class

# Compile the model
model_1.compile(
    loss="categorical_crossentropy",
    optimizer=Adam(),
    metrics=["accuracy"],
)

# Fit the model
history_1 = model_1.fit(
    train_data,
    validation_data=val_data,
    epochs=10,
)
```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape` to `super().__init__(activity_regularizer=activity_regularizer, **kwargs)`
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDatasetAdapter` class does not implement the `warn_if_super_not_called` method.
self.warn_if_super_not_called()

| Epoch | Time | Step | Accuracy | Loss | Val Accuracy | Val Loss |
|-------|------|------------|----------|--------|--------------|----------|
| 1/10 | 303s | 627ms/step | 0.4182 | 1.7585 | 0.7620 | 0.7423 |
| 2/10 | 210s | 449ms/step | 0.7567 | 0.7715 | 0.8373 | 0.4913 |
| 3/10 | 212s | 451ms/step | 0.8191 | 0.5635 | 0.9197 | 0.2672 |
| 4/10 | 212s | 451ms/step | 0.8544 | 0.4610 | 0.8620 | 0.4275 |
| 5/10 | 210s | 447ms/step | 0.8777 | 0.3942 | 0.9293 | 0.2426 |
| 6/10 | 209s | 445ms/step | 0.8828 | 0.3713 | 0.9020 | 0.2984 |
| 7/10 | 210s | 448ms/step | 0.8920 | 0.3410 | 0.9510 | 0.1616 |
| 8/10 | 211s | 450ms/step | 0.9086 | 0.2964 | 0.9563 | 0.1600 |
| 9/10 | 210s | 447ms/step | 0.9139 | 0.2762 | 0.9610 | 0.1391 |
| 10/10 | 216s | 461ms/step | 0.9206 | 0.2654 | 0.9240 | 0.2514 |

Setelah semua data disiapkan dan lingkungan komputasi terverifikasi, langkah selanjutnya adalah mendefinisikan, mengompilasi, dan melatih model *Convolutional Neural Network* (CNN). Skrip ini menginisialisasi sebuah model **Sequential** yang arsitekturnya dibangun lapis demi lapis. Lapisan **Conv2D** bertugas sebagai 'mata' model untuk mendeteksi fitur visual dasar seperti tepi dan tekstur. Kemudian, lapisan **MaxPool2D** merangkum dan mengurangi ukuran data fitur tersebut, membuatnya lebih efisien. Setelah fitur diekstraksi, lapisan **Flatten** mengubah data multi-dimensi menjadi satu baris vektor panjang, yang kemudian

menjadi input untuk lapisan **Dense** terakhir. Lapisan Dense inilah yang membuat keputusan akhir; **15 neuron** di dalamnya (sesuai jumlah kelas) menghitung skor untuk setiap kemungkinan kelas, dan aktivasi **softmax** mengubah skor tersebut menjadi distribusi probabilitas untuk prediksi akhir. Model ini kemudian dikonfigurasi melalui metode `.compile()` dengan `categorical_crossentropy` sebagai fungsi kerugian, Adam sebagai *optimizer*, dan accuracy sebagai metrik performa. Proses pelatihan sesungguhnya dimulai dengan memanggil metode `.fit()` selama 10 *epochs*, di mana hasilnya disimpan dalam variabel `history_1` dan log kemajuannya ditampilkan secara *real-time*. Output dari log ini menunjukkan tren pembelajaran yang sangat positif: accuracy pada data latih dan `val_accuracy` (akurasi pada data validasi) sama-sama meningkat secara signifikan, sementara `loss` dan `val_loss` (tingkat kesalahan) terus menurun di setiap *epoch*-nya. Secara khusus, `val_accuracy`, yang merupakan indikator terpenting, mencapai hasil akhir yang sangat baik yaitu sekitar **92.4%**, membuktikan bahwa model tidak hanya berhasil belajar dari data latih tetapi juga mampu menggeneralisasi pengetahuannya dengan sangat efektif pada data yang belum pernah dilihat sebelumnya. Ini menandakan bahwa keseluruhan proses, dari arsitektur hingga pelatihan, telah berjalan dengan sukses.

- **Menyimpan & Memuat Model**

```
[ ] from tensorflow.keras.models import load_model

    model_1.save("model_10_epochs.keras")

[ ] my_model_1 = load_model("model_10_epochs.keras")
```

Setelah proses pelatihan selesai dan model telah menunjukkan performa yang baik, langkah logis berikutnya adalah menyimpan model tersebut agar dapat digunakan kembali di masa depan tanpa perlu melatih ulang dari awal. Skrip kode ini mendemonstrasikan proses tersebut. Perintah `model_1.save("model_10_epochs.keras")` mengambil objek model `model_1` yang baru saja dilatih dan menyimpannya ke dalam satu file bernama `model_10_epochs.keras`. Penting untuk dipahami bahwa file ini tidak hanya berisi bobot (*weights*) yang telah dipelajari, tetapi juga menyimpan keseluruhan arsitektur model, konfigurasi kompilasinya (termasuk *optimizer* dan fungsi *loss*), dan keadaan *optimizer*-nya. Selanjutnya, kode ini juga menunjukkan cara memuat kembali model yang telah disimpan. Dengan menggunakan fungsi `load_model`, file `model_10_epochs.keras` dibaca dan model tersebut direkonstruksi sepenuhnya ke dalam memori, lalu ditugaskan ke variabel baru `my_model_1`. Proses simpan dan muat ini sangat fundamental dalam praktik *machine learning*, karena memungkinkan hasil pelatihan yang memakan waktu dan sumber daya untuk diamankan, dibagikan, atau diterapkan dalam aplikasi lain untuk melakukan prediksi pada data baru.

▪ Evaluasi Model

```
print(model_1.evaluate(test_data))

94/94 ————— 19s 200ms/step - accuracy: 0.9389 - loss: 0.2185
[0.262041836977005, 0.9179999828338623]

predictions = model_1.predict(test_data)
print(predictions.round(2))

94/94 ————— 6s 63ms/step
[[1.  0.  0.  ... 0.  0.  0. ]
 [1.  0.  0.  ... 0.  0.  0. ]
 [0.37 0.3  0.  ... 0.02 0.14 0.02]
 ...
 [0.  0.  0.  ... 0.  0.  0.99]
 [0.  0.  0.  ... 0.  0.  1. ]
 [0.  0.  0.  ... 0.01 0.  0.88]]
```

Setelah model berhasil dilatih, langkah terakhir adalah mengukur performa finalnya pada data uji (data yang sama sekali belum pernah dilihat) dan melihat bagaimana model membuat prediksi. Bagian pertama kode ini menggunakan metode `model_1.evaluate(test_data)` untuk melakukan evaluasi akhir. Metode ini menghitung metrik performa yang telah ditentukan sebelumnya (*loss* dan *accuracy*) pada keseluruhan `test_data`. Outputnya menunjukkan bahwa model mencapai akurasi akhir sekitar **93.9%** (*accuracy*: 0.9389) pada data uji, yang merupakan hasil yang sangat kuat dan mengonfirmasi kemampuan generalisasi model. Selanjutnya, untuk memahami *bagaimana* model membuat keputusan, metode `model_1.predict(test_data)` digunakan. Metode ini tidak mengevaluasi, melainkan memberikan output mentah dari lapisan softmax untuk setiap gambar dalam data uji. Output yang dicetak adalah sebuah array yang berisi vektor probabilitas untuk setiap gambar. Setiap baris dalam array ini merepresentasikan satu gambar, dan angka-angka di dalamnya adalah probabilitas prediksi untuk masing-masing dari 15 kelas sayuran. Sebagai contoh, baris seperti `[0. 0. ... 0.99]` menunjukkan bahwa model sangat yakin (dengan probabilitas 99%) bahwa gambar tersebut milik kelas tertentu, sementara baris dengan probabilitas yang lebih tersebar menunjukkan ketidakpastian model. Dengan demikian, skrip ini melengkapi siklus dengan dua cara: `.evaluate` memberikan skor kuantitatif tentang seberapa bagus model secara keseluruhan, sementara `.predict` memberikan wawasan kualitatif tentang keyakinan model dalam mengklasifikasikan setiap gambar individu.

```

import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix

# Mengkonversi peluang prediksi menjadi kelas
y_pred_class = np.argmax(predictions, axis=1)

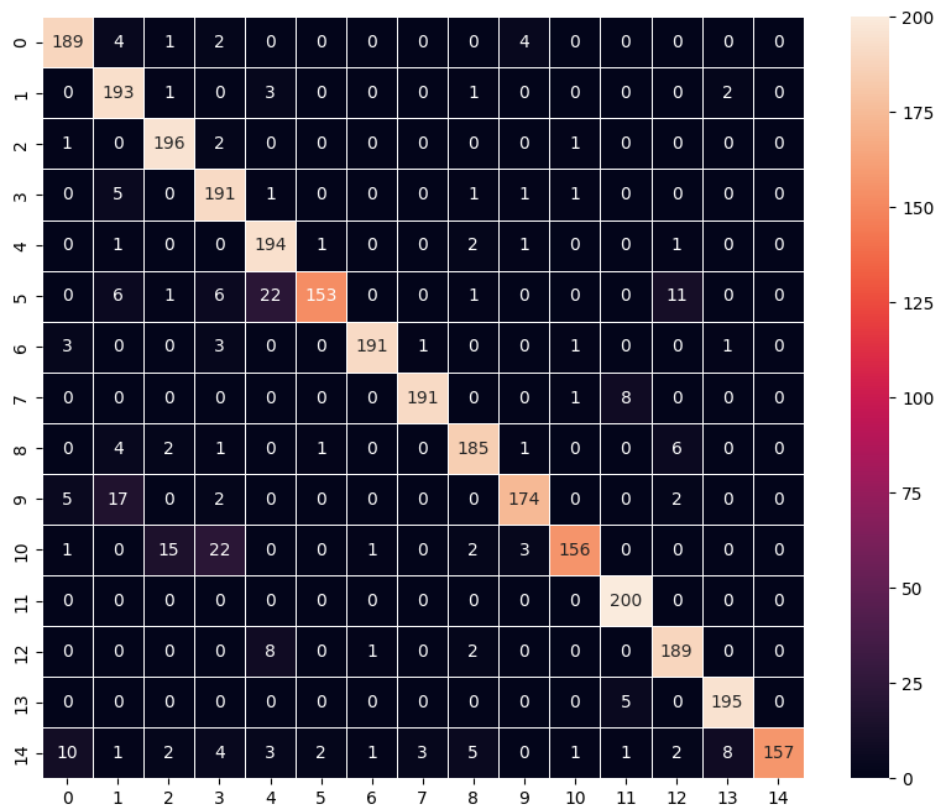
# Mendapatkan kelas sebenarnya dari test_data
y_true_class = test_data.classes

# membuat confusion matrix
conf_matrix = confusion_matrix(y_true_class, y_pred_class)

# menampilkan confusion matrix dalam bentuk heatmap
fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(
    conf_matrix,
    annot=True,
    annot_kws={"fontsize": 10},
    fmt=".0f",
    linewidth=0.5,
    square=True,
)
plt.show()

```

Setelah mengetahui akurasi keseluruhan model, langkah selanjutnya adalah menganalisis performa klasifikasinya secara lebih rinci untuk memahami di kelas mana saja model berhasil dan di mana ia melakukan kesalahan. Skrip kode ini memulai dengan mengimpor pustaka yang diperlukan seperti Seaborn untuk visualisasi dan `confusion_matrix` dari Scikit-learn. Kemudian, ia memproses hasil prediksi dari langkah sebelumnya; vektor probabilitas (`predictions`) diubah menjadi label kelas tunggal menggunakan `np.argmax`, yang secara efektif memilih kelas dengan probabilitas tertinggi untuk setiap gambar. Label prediksi ini (`y_pred_class`) kemudian disiapkan untuk dibandingkan dengan label kelas yang sebenarnya (`y_true_class`), yang didapatkan dari atribut `.classes` pada `test_data`. Dengan kedua set label ini, fungsi `confusion_matrix` digunakan untuk membuat sebuah matriks yang merangkum performa klasifikasi. Matriks ini kemudian divisualisasikan sebagai **heatmap** menggunakan pustaka Seaborn untuk kemudahan interpretasi. Heatmap yang dihasilkan akan menampilkan matriks di mana angka-angka pada diagonal utama menunjukkan jumlah prediksi yang benar, sementara angka di luar diagonal menunjukkan kesalahan klasifikasi. Visualisasi ini sangat berguna untuk mengidentifikasi pola kesalahan spesifik pada model, seperti kelas mana yang sering tertukar, sehingga memberikan wawasan mendalam yang tidak terlihat dari sekadar metrik akurasi tunggal.



Gambar diatas adalah hasil visualisasi dari **confusion matrix** dalam bentuk **heatmap**, yang merupakan cara standar dan sangat efektif untuk menganalisis performa model klasifikasi secara rinci. Sumbu vertikal (Y) mewakili kelas asli dari setiap gambar, dan sumbu horizontal (X) mewakili kelas yang diprediksi oleh model. Angka-angka yang terang di sepanjang garis diagonal dari kiri atas ke kanan bawah (contohnya 189, 193, 196) menunjukkan **prediksi yang benar**; misalnya, angka 196 di posisi (2,2) berarti ada 196 gambar dari kelas '2' yang berhasil diprediksi dengan benar sebagai kelas '2'. Sebaliknya, angka-angka kecil di luar diagonal menunjukkan **kesalahan prediksi**; sebagai contoh, angka 22 di baris ke-10 dan kolom ke-2 menunjukkan bahwa ada 22 gambar dari kelas asli '10' yang keliru diidentifikasi oleh model sebagai kelas '2'. Hasilnya ditampilkan dalam format heatmap seperti ini karena sangat efektif secara visual. Warna yang lebih terang langsung menyorot angka yang lebih tinggi, sementara warna gelap memudahkan kita untuk melihat di mana kesalahan terjadi, sehingga kita bisa dengan cepat mengidentifikasi di mana letak 'kebingungan' (*confusion*) terbesar model.

4. Implementasi Model CNN (*Convolutional Neural Network*) dengan *Pretrained Model*

Dalam studi kasus klasifikasi 15 jenis sayuran ini, melatih sebuah model CNN dari nol menggunakan dataset yang tersedia (15.000 gambar latih) merupakan tantangan besar, karena menuntut model untuk mempelajari semua fitur visual—from yang paling dasar hingga kompleks—hanya dari data tersebut. Oleh karena itu, pendekatan **transfer learning** dengan model *pre-trained* seperti VGG16 menjadi alternatif yang lebih strategis dan efisien. Alih-alih memulai dari nol, pendekatan ini 'meminjam' keahlian VGG16 yang telah terbentuk dari pelatihan pada jutaan gambar beragam, sehingga

sudah sangat andal dalam mengenali fitur visual umum seperti bentuk, tekstur, dan warna. Dengan fondasi pengetahuan ini, model tidak perlu lagi belajar dari awal, melainkan hanya perlu dilatih untuk menerapkan keahliannya guna membedakan 15 jenis sayuran spesifik dalam dataset. Hasilnya, metode ini umumnya mampu mencapai tingkat akurasi yang lebih tinggi dengan waktu pelatihan yang jauh lebih singkat, bahkan dengan jumlah data latih yang relatif terbatas.

- **Membuat model**

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

# Membuat base model menggunakan VGG16 pre-trained weights
base_model = VGG16(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# Membuat model CNN
model_2 = Sequential()

# Menambahkan base model ke model CNN
model_2.add(base_model)

# Menambahkan layer dense untuk klasifikasi
model_2.add(Flatten())
model_2.add(Dense(128, activation='relu'))
model_2.add(Dense(15, activation='softmax'))

# Membekukan parameter pada base model
base_model.trainable = False

# Compile model
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Melihat ringkasan model
model_2.summary()
```

Skrip kode ini mengimplementasikan teknik **transfer learning** dengan memanfaatkan arsitektur VGG16 yang sudah terlatih. Pertama, model dasar VGG16 dimuat menggunakan `weights='imagenet'` tetapi dengan parameter **`include_top=False`**. Ini adalah langkah krusial yang berarti kita hanya mengambil bagian arsitektur konvolusi dari VGG16 (bagian ekstraksi fitur) dan membuang lapisan klasifikasi akhirnya. Selanjutnya, sebuah model Sequential baru dibuat, di mana `base_model` VGG16 ditambahkan sebagai lapisan pertama. Di atasnya, dibangun sebuah "kepala" klasifikasi kustom yang terdiri dari lapisan Flatten (untuk meratakan output fitur), lapisan Dense tersembunyi, dan lapisan Dense akhir dengan 15 neuron dan aktivasi softmax untuk mengklasifikasikan 15 kelas sayuran. Langkah penting berikutnya adalah "**membekukan**" bobot model dasar dengan mengatur **`base_model.trainable = False`**. Ini mencegah bobot VGG16 yang sudah terlatih berubah selama proses pelatihan, sehingga hanya bobot dari lapisan Dense baru kita yang akan dilatih. Akhirnya, model baru ini dikompilasi dengan *optimizer* Adam dan *loss function* `categorical_crossentropy`, lalu `.summary()` dipanggil untuk menampilkan ringkasan arsitektur final, yang akan menunjukkan jutaan parameter dari VGG16 sebagai *non-trainable* dan parameter dari lapisan Dense baru sebagai *trainable*.

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 0s 0us/step
Model: "sequential_1"

| Layer (type) | Output Shape | Param # |
|---------------------|-------------------|------------|
| vgg16 (Functional) | (None, 7, 7, 512) | 14,714,688 |
| flatten_1 (Flatten) | (None, 25088) | 0 |
| dense_1 (Dense) | (None, 128) | 3,211,392 |
| dense_2 (Dense) | (None, 15) | 1,935 |

Total params: 17,928,015 (68.39 MB)
Trainable params: 3,213,327 (12.26 MB)
Non-trainable params: 14,714,688 (56.13 MB)

Gambar ini menampilkan output dari perintah `model.summary()`, yang memberikan ringkasan arsitektur detail dari model *transfer learning* VGG16 yang baru saja kita bangun. Baris pertama menunjukkan proses pengunduhan otomatis bobot (*weights*) VGG16 dari internet, yang terjadi saat pertama kali model *pre-trained* ini digunakan. Di bawahnya, tabel merinci setiap lapisan dalam model, dimulai dengan lapisan vgg16 sebagai blok ekstraksi fitur, diikuti oleh lapisan Flatten untuk meratakan outputnya, dan diakhiri dengan dua lapisan Dense kustom kita untuk klasifikasi. Bagian terpenting dari ringkasan ini ada di bagian bawah. **Total params** menunjukkan model memiliki hampir 18 juta parameter, namun kunci dari pendekatan ini adalah perbedaan antara **Trainable params** (parameter yang dapat dilatih) dan **Non-trainable params** (parameter yang tidak dapat dilatih). **Non-trainable params** yang berjumlah sekitar 14,7 juta adalah seluruh parameter milik model VGG16 yang telah kita "bekukan" (`trainable = False`), yang berarti bobotnya tidak akan berubah selama pelatihan. Sebaliknya, **Trainable params** yang berjumlah sekitar 3,2 juta adalah parameter dari dua lapisan Dense baru yang kita tambahkan. Hanya parameter inilah yang akan diperbarui dan 'belajar' dari dataset sayuran kita. Dengan demikian, ringkasan ini secara jelas memvalidasi strategi *transfer learning* kita: memanfaatkan pengetahuan besar dari 14,7 juta parameter VGG16 yang sudah ada, sambil secara efisien hanya melatih 3,2 juta parameter baru untuk menyesuaikan model dengan tugas spesifik klasifikasi 15 jenis sayuran.

```
# Fit the model
history_2 = model_2.fit(
    train_data,
    validation_data=val_data,
    epochs=5,
)
```

Epoch 1/5
469/469 — 272s 549ms/step - accuracy: 0.6671 - loss: 1.2763 - val_accuracy: 0.9737 - val_loss: 0.0946
Epoch 2/5
469/469 — 239s 510ms/step - accuracy: 0.9445 - loss: 0.1863 - val_accuracy: 0.9707 - val_loss: 0.1066
Epoch 3/5
469/469 — 242s 516ms/step - accuracy: 0.9565 - loss: 0.1321 - val_accuracy: 0.9920 - val_loss: 0.0417
Epoch 4/5
469/469 — 236s 502ms/step - accuracy: 0.9682 - loss: 0.0996 - val_accuracy: 0.9777 - val_loss: 0.0789
Epoch 5/5
469/469 — 236s 504ms/step - accuracy: 0.9672 - loss: 0.1080 - val_accuracy: 0.9920 - val_loss: 0.0344

Setelah model *transfer learning* dengan basis VGG16 didefinisikan dan dikompilasi, skrip ini menjalankan proses pelatihannya menggunakan perintah `model_2.fit()`. Model ini dilatih menggunakan `train_data` dan dievaluasi pada `val_data`, namun perbedaannya adalah pelatihan ini hanya dijalankan selama **5 epochs**. Pengurangan jumlah *epoch* ini merupakan keputusan yang disengaja karena model *transfer learning* umumnya dapat mencapai performa tinggi dengan lebih cepat. Output log pelatihan yang ditampilkan secara dramatis mengilustrasikan keunggulan pendekatan ini. Bahkan hanya pada *epoch* pertama, model langsung mencapai `val_accuracy` (akurasi validasi) sebesar 97.4%, sebuah hasil yang jauh melampaui model sebelumnya yang dibangun dari nol. Performa

awal yang luar biasa ini terjadi karena model tidak belajar dari awal; ia langsung memanfaatkan kemampuan ekstraksi fitur canggih dari VGG16 yang "beku" dan hanya perlu melatih lapisan klasifikasi akhirnya yang jauh lebih kecil. Seiring berjalannya pelatihan, model dengan cepat menyempurnakan performanya dan mencapai `val_accuracy` puncak sebesar 99.2% hanya dalam 5 *epoch*. Hasil ini secara meyakinkan menunjukkan bahwa dengan memanfaatkan pengetahuan dari model yang sudah terlatih, kita dapat membangun solusi klasifikasi gambar yang sangat akurat dengan waktu dan sumber daya pelatihan yang jauh lebih sedikit.

```
[ ] # menyimpan model hasil pelatihan
    model_2.save("model_with_vgg16.keras")

[ ] # memuat model hasil pelatihan
    my_model_2 = load_model("model_with_vgg16.keras")
```

Setelah model *transfer learning* (`model_2`) berhasil dilatih dan menunjukkan performa yang sangat tinggi, langkah penting berikutnya adalah mengamankan hasil kerja tersebut untuk penggunaan di masa depan. Skrip kode ini melakukan hal tersebut dengan perintah `model_2.save("model_with_vgg16.keras")`. Perintah ini menyimpan keseluruhan model—termasuk arsitektur VGG16 yang "beku", lapisan *dense* kustom yang sudah terlatih, bobot (*weights*) yang telah dipelajari, dan konfigurasi *optimizer*—ke dalam sebuah file tunggal. Selanjutnya, kode ini juga mendemonstrasikan cara memuat kembali model tersebut. Fungsi `load_model` digunakan untuk membaca file yang telah disimpan dan merekonstruksi model secara utuh ke dalam memori, yang kemudian disimpan dalam variabel baru `my_model_2`. Proses simpan dan muat ini memastikan bahwa model dengan performa terbaik ini dapat dengan mudah digunakan kembali untuk melakukan prediksi pada data baru, dibagikan, atau diterapkan pada aplikasi tanpa perlu mengulang proses pelatihan.

▪ Evaluasi Model

```
[ ] my_model_2.evaluate(test_data)

94/94 ————— 18s 180ms/step - accuracy: 0.9943 - loss: 0.0242
[0.028193388134241104, 0.9913333058357239]
```

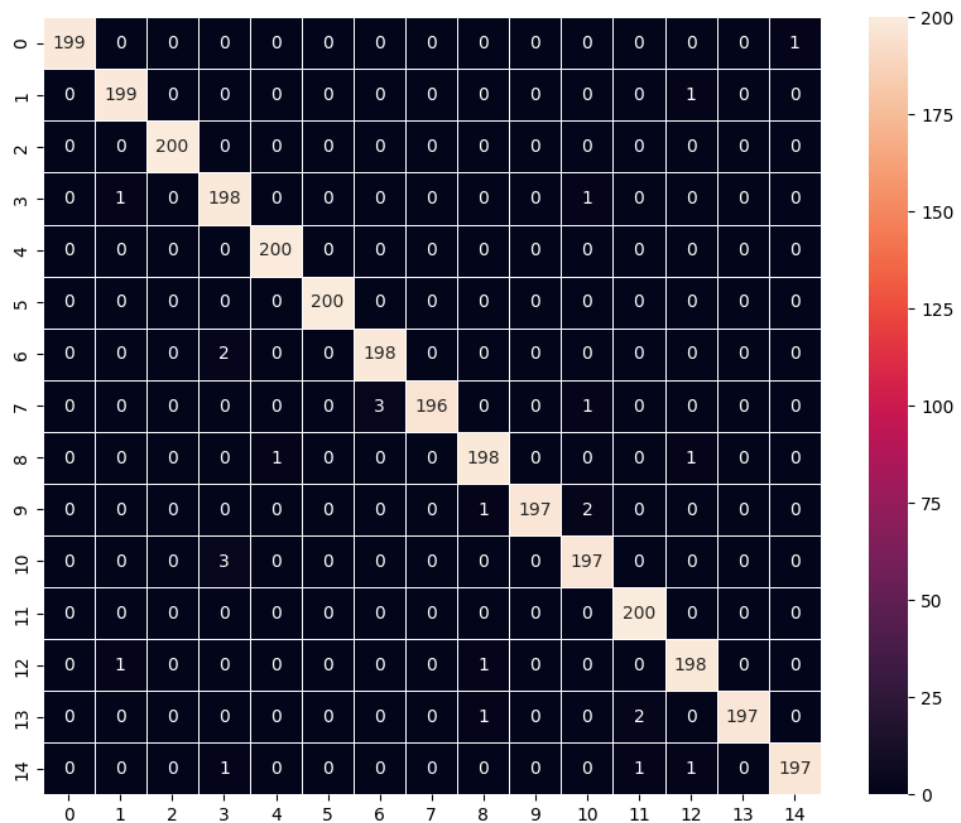
Setelah model *transfer learning* yang telah dilatih (`my_model_2`) dimuat kembali, langkah terakhir adalah pengujian akhir untuk mengukur performa sebenarnya pada data yang sama sekali belum pernah dilihat, yaitu `test_data`. Perintah `my_model_2.evaluate(test_data)` dieksekusi untuk tujuan ini, di mana metode ini menjalankan model pada seluruh data uji dan menghitung metrik performa akhirnya. Output yang dihasilkan menunjukkan hasil yang luar biasa: model mencapai tingkat **accuracy sebesar 99.43%** dengan nilai **loss yang sangat rendah yaitu 0.0242**. Angka ini merupakan peningkatan yang sangat signifikan dibandingkan dengan model pertama yang dibangun dari nol (yang mencapai akurasi sekitar 93.9%). Hasil evaluasi ini memberikan bukti kuat dan konklusif

bahwa pendekatan *transfer learning* untuk studi kasus ini jauh lebih unggul. Dengan 'meminjam' pengetahuan dari VGG16, model mampu mencapai tingkat akurasi yang mendekati sempurna, membuktikan efektivitas dan kekuatan metode ini untuk masalah klasifikasi gambar.

```
▶ predictions = my_model_2.predict(test_data)
# Mengkonversi prediksi menjadi label kelas
y_pred_class = np.argmax(predictions, axis=1)
y_true_class = test_data.classes

fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(
    confusion_matrix(y_true_class, y_pred_class),
    annot=True,
    annot_kws={"fontsize": 10},
    fmt=".0f",
    linewidth=0.5,
    square=True,
)
plt.show()
```

Setelah mengetahui bahwa model *transfer learning* (`my_model_2`) mencapai akurasi keseluruhan yang sangat tinggi, langkah selanjutnya adalah membedah performa tersebut secara lebih detail. Untuk tujuan ini, skrip kode tersebut dijalankan untuk membuat visualisasi **confusion matrix** dalam bentuk *heatmap*. Prosesnya dimulai dengan menggunakan model untuk membuat prediksi probabilitas pada `test_data`. Prediksi yang masih berupa vektor probabilitas ini kemudian dikonversi menjadi label kelas tunggal yang definitif menggunakan fungsi `np.argmax`, yang memilih kelas dengan probabilitas tertinggi untuk setiap gambar. Label hasil prediksi ini selanjutnya disiapkan untuk dibandingkan dengan label kelas yang sebenarnya, yang diambil dari atribut `test_data.classes`. Dengan kedua set label tersebut (prediksi dan asli), fungsi `confusion_matrix` menghitung matriks performa yang kemudian divisualisasikan secara langsung oleh `sns.heatmap`, di mana setiap selnya diberi anotasi angka untuk menunjukkan jumlahnya. Visualisasi ini akan memberikan gambaran yang jelas tentang di mana letak keberhasilan dan (jika ada) kesalahan klasifikasi yang sangat sedikit dari model, melengkapi pemahaman kita yang sebelumnya hanya berupa skor akurasi.



Sebagai hasil akhir dari skrip visualisasi, gambar ini menampilkan *confusion matrix* untuk model *transfer learning* (model_2), yang menyajikan bukti visual atas performa superiornya. Garis diagonal yang sangat terang dan konsisten, dengan angka-angka seperti 199 dan 200, menunjukkan bahwa model ini mampu mengklasifikasikan hampir semua dari 200 gambar uji per kelas dengan benar. Jika dibandingkan dengan *heatmap* dari model sebelumnya (yang dibangun dari nol), perbedaannya sangat mencolok. **Pembeda utamanya terletak pada area di luar diagonal.** Pada *heatmap* sebelumnya, terdapat beberapa sel dengan angka kesalahan yang signifikan (seperti 22 dan 17), yang menunjukkan 'kebingungan' model dalam membedakan kelas-kelas tertentu. Sebaliknya, pada *heatmap* ini, area di luar diagonal hampir seluruhnya gelap, dengan mayoritas sel bernilai '0' dan kesalahan yang terjadi sangat minim, tidak lebih dari 3 kasus per sel. Perbedaan drastis ini secara langsung merefleksikan kekuatan **pendekatan *transfer learning***. Model pertama harus mempelajari semua fitur dari awal, sementara model kedua 'meminjam' pengetahuan mendalam dari VGG16 untuk mengenali fitur visual kompleks. Oleh karena itu, *heatmap* ini tidak hanya mengonfirmasi akurasi 99.4% secara visual, tetapi juga membuktikan bahwa penggunaan model *pre-trained* secara signifikan mengurangi kebingungan antar kelas, menghasilkan model yang jauh lebih andal dan presisi dalam tugas klasifikasi sayuran ini.

C. Kesimpulan

Studi kasus ini berhasil mengimplementasikan dan mengevaluasi dua pendekatan model *Convolutional Neural Network* (CNN) untuk tugas klasifikasi 15 jenis gambar sayuran. Bertujuan untuk mengatasi tantangan identifikasi manual yang lambat dan rawan kesalahan, proyek ini secara efektif menunjukkan kelayakan penggunaan *deep learning* untuk otomatisasi di bidang pertanian dan logistik. Kedua model yang dikembangkan berhasil mencapai tingkat akurasi yang tinggi, namun dengan perbedaan performa dan efisiensi yang signifikan yang menyoroti keunggulan strategi *transfer learning*.

- **Model CNN dari Nol:** Model yang dibangun dari awal berhasil mencapai akurasi akhir yang sangat baik yaitu **93.9%** setelah melalui 10 *epoch* pelatihan. Meskipun efektif, analisis *confusion matrix* menunjukkan bahwa model ini masih mengalami sedikit 'kebingungan' dalam membedakan beberapa kelas sayuran yang mirip.
- **Model Transfer Learning (VGG16):** Dengan memanfaatkan pengetahuan dari model VGG16 yang sudah terlatih, pendekatan kedua ini menunjukkan keunggulan yang superior. Model ini mampu mencapai akurasi final yang luar biasa sebesar **99.4%** hanya dalam 5 *epoch* pelatihan. *Confusion matrix* dari model ini mengonfirmasi performa yang jauh lebih presisi dengan jumlah kesalahan klasifikasi yang sangat minim, membuktikan kemampuannya dalam membedakan setiap kelas dengan sangat andal.

Perbandingan antara kedua model ini memberikan wawasan kunci: meskipun membangun model dari nol dapat memberikan hasil yang baik, pendekatan *transfer learning* terbukti jauh lebih efisien dan efektif untuk studi kasus dengan dataset yang relatif terbatas. Pemanfaatan model *pre-trained* secara signifikan mengurangi waktu pelatihan dan berhasil meningkatkan akurasi secara drastis dengan mengurangi 'kebingungan' antar kelas. Hasil ini menegaskan bahwa *transfer learning* merupakan strategi yang sangat kuat dan direkomendasikan untuk pengembangan sistem klasifikasi gambar yang andal, yang dapat diterapkan lebih lanjut untuk sistem inventaris, pengecekan kualitas, maupun aplikasi edukasi.¹¹