

Nama: Shifa Qonita

NIM: 22611028




Dokumentasi: https://github.com/shifaqonita/Artificial_Intelligence

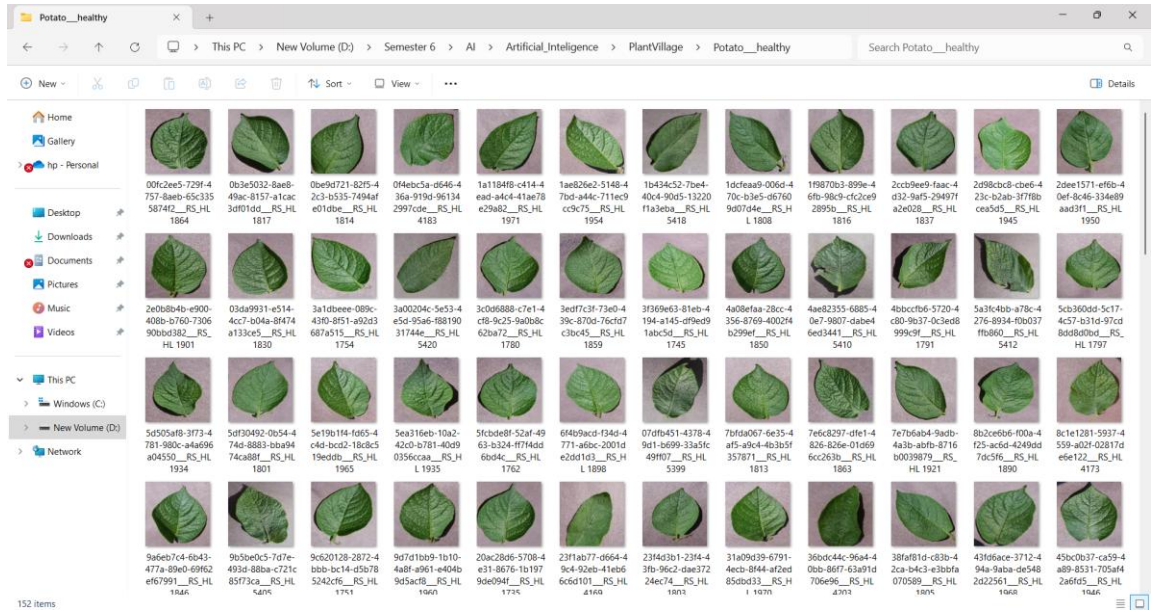
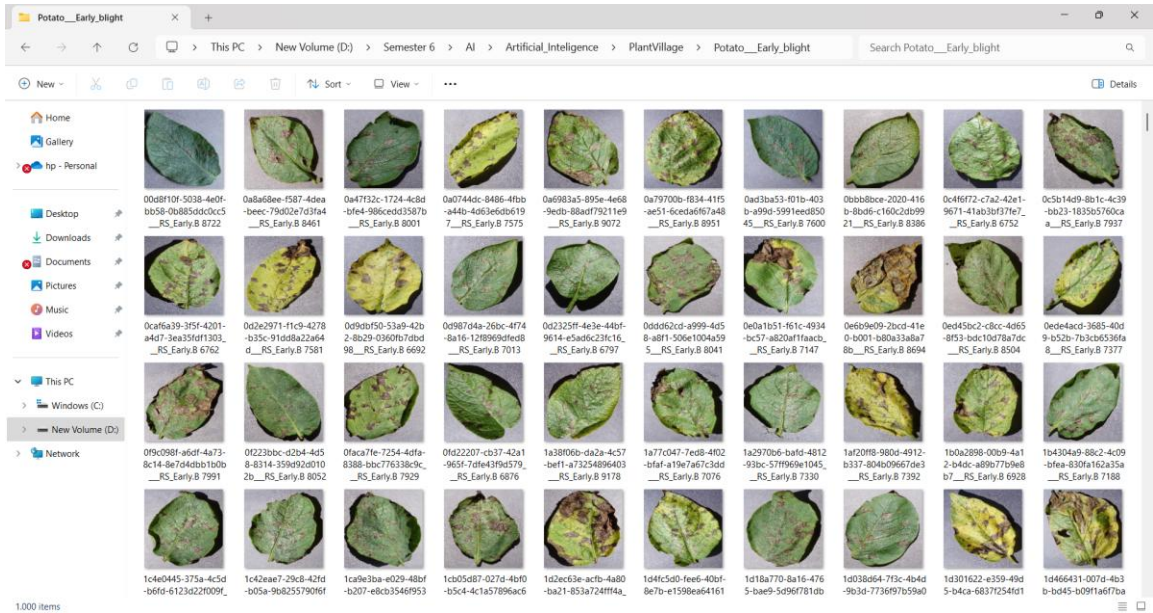
Bagian I: Pengumpulan dan Persiapan Data (30%)

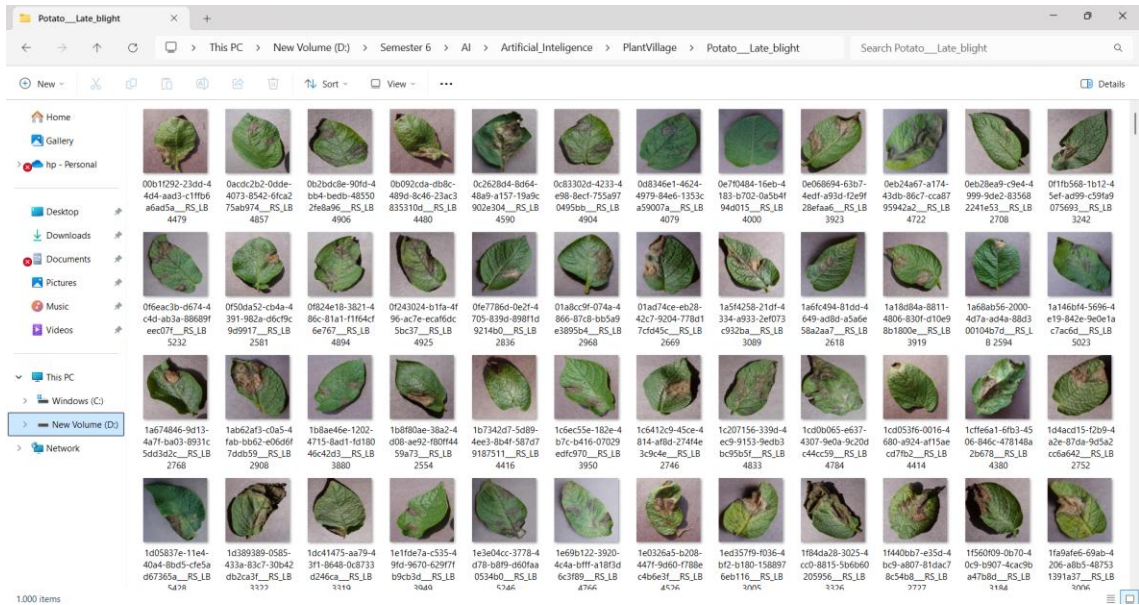
1. Pilih tiga kategori jenis objek berbeda yang unik (contoh sampah organik, sampah anorganik daur ulang, sampah anorganik non daur ulang).
2. Dokumentasikan masing-masing objek tersebut dengan mengambil minimal 100 citra per kategori objek. Usahakan gambar memiliki variasi posisi, pencahayaan, dan latar belakang.
3. Lakukan proses *preprocessing* citra: resizing, normalisasi, augmentasi (opsional), dan pembagian data menjadi *training* dan *testing*.

Jawab

1. Objek yang digunakan yaitu daun kentang yang diklasifikasikan ke dalam 3 kelas yaitu *Early blight*, *Late blight*, dan *Healthy*. Klasifikasi ini bertujuan untuk membedakan kondisi daun kentang berdasarkan adanya gejala penyakit *Early blight* dan *Late blight*, serta daun yang sehat tanpa penyakit. Hal ini penting dilakukan karena setiap tahun para petani mengalami kerugian ekonomi dan pemborosan hasil panen akibat berbagai penyakit pada tanaman kentang. *Early blight* dan *Late blight* merupakan dua penyakit utama yang menyerang daun kentang dan menjadi penyebab utama penurunan hasil panen. Diperkirakan sebagian besar kerugian dalam produksi kentang disebabkan oleh kedua penyakit ini, sehingga gambar daun diklasifikasikan ke dalam tiga kelas tersebut guna mendukung proses identifikasi dini dan penanganan yang lebih efektif.
2. Jumlah citra daun kentang yang digunakan terdiri dari 1000 citra untuk kondisi *Early blight*, 1000 citra untuk kondisi *Late blight*, dan 152 citra untuk kondisi *Healthy*.

 Potato__Early_blight	16/05/2025 21:58	File folder
 Potato__healthy	16/05/2025 21:58	File folder
 Potato__Late_blight	16/05/2025 21:58	File folder





3. Proses *preprocessing*

Sebelumnya, berbagai pustaka penting diimpor terlebih dahulu. Kemudian, tetapkan nilai *seed* yaitu 7 secara global pada *TensorFlow*, *NumPy*, dan modul *random* dengan tujuan memastikan hasil eksperimen dapat direproduksi secara konsisten. *Seed* yang sama akan menghasilkan hasil yang sama pada setiap kali model dijalankan ulang. Selanjutnya, didefinisikan *path dataset* yang menunjukkan lokasi folder berisi data gambar untuk proses pelatihan model.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
import numpy as np
import random

# Set seed global untuk reproducibility
seed = 7
tf.random.set_seed(seed)
np.random.seed(seed)
random.seed(seed)

# Path dataset
dataset_dir = 'D:\\Semester 6\\AI\\Artificial_Intelligence\\PlantVillage'
```

Dataset gambar dimuat dari folder yang berisi subfolder kategori dengan menggunakan fungsi `image_dataset_from_directory` dari *TensorFlow* dan gambar secara otomatis di-*resize* ke ukuran 128×128 piksel, disusun dalam *batch* berukuran 16, dan labelnya dipetakan dalam format integer. Kemudian, seluruh *batch* gambar dan label diubah ke dalam format *array NumPy* agar dapat diolah lebih lanjut secara fleksibel seperti berikut.

```
# Parameter
batch_size = 16
img_size = (128, 128)

# Load data (resize otomatis)
full_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_dir,
    image_size=img_size,
    batch_size=batch_size,
    label_mode='int',
    shuffle=True,
    seed=seed
)

# Ambil data dan label numpy
data, labels = [], []
for img_batch, label_batch in full_dataset:
    data.append(img_batch.numpy())
    labels.append(label_batch.numpy())
data = np.concatenate(data)
labels = np.concatenate(labels)
```

[23] ✓ 0.7s

... Found 2152 files belonging to 3 classes.

Selanjutnya, data citra yang berbentuk pixel dengan rentang nilai 0 hingga 255 dinormalisasi menjadi rentang 0 hingga 1 dengan membagi setiap nilai pixel dengan 255.0. Normalisasi ini bertujuan agar model dapat belajar dengan lebih stabil dan efektif karena nilai input memiliki skala yang seragam.

```
# Normalisasi data pixel dari [0,255] ke [0,1]
data = data / 255.0
data
```

[24] ✓ 0.2s

```
... array([[[[0.44705883, 0.4      , 0.4392157 ],
             [0.3764706 , 0.32941177, 0.36862746],
             [0.4      , 0.3529412 , 0.39215687],
             ...,
             [0.672549 , 0.6490196 , 0.6882353 ],
             [0.6745098 , 0.6509804 , 0.6901961 ],
             [0.68235296, 0.65882355, 0.69803923]],
            [[0.44607842, 0.3990196 , 0.43823528],
             [0.4137255 , 0.36666667, 0.40588236],
             [0.3882353 , 0.34117648, 0.38039216],
             ...,
             [0.66862744, 0.64509803, 0.6843137 ],
             [0.67156863, 0.6480392 , 0.6872549 ],
             [0.6754902 , 0.6519608 , 0.6911765 ]],
            [[0.45294118, 0.40588236, 0.44509804],
             [0.42647058, 0.37941176, 0.41862744],
             [0.38039216, 0.33333334, 0.37254903],
             ...,
             [0.6745098 , 0.6509804 , 0.6901961 ],
             [0.672549 , 0.6490196 , 0.6882353 ],
             [0.6745098 , 0.6509804 , 0.6901961 ]],
            ...
            ...
            ...])])])
```

Setelah itu, data dan label dibagi menjadi tiga bagian, yakni data *training* sebanyak 80%, data *validasi* 10%, dan data *testing* 10%. Pembagian ini dilakukan secara bertahap menggunakan fungsi `train_test_split` dari `scikit-learn`, sehingga memastikan distribusi data yang acak dan representatif untuk setiap subset.

```
# Split data jadi train 80%, val 10%, test 10%
X_train, X_temp, y_train, y_temp = train_test_split(data, labels, test_size=0.2, random_state=seed, shuffle=True)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=seed, shuffle=True)
```

[29] ✓ 0.2s

Untuk meningkatkan keragaman data *training* dan mencegah model terlalu menghafal pola tertentu (*overfitting*), dilakukan augmentasi data secara sederhana dengan menerapkan *random horizontal flip* (pembalikan gambar secara horizontal secara acak). Augmentasi ini hanya diterapkan pada dataset *training* menggunakan fungsi *map* dari *TensorFlow*.

```
# Buat fungsi augmentasi sederhana (flip horizontal)
def augment(image, label):
    image = tf.image.random_flip_left_right(image)
    return image, label

# Buat tf.data.Dataset untuk train, val, test
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train)).shuffle(1000, seed=seed).batch(batch_size)
val_dataset = tf.data.Dataset.from_tensor_slices((X_val, y_val)).batch(batch_size)
test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(batch_size)

# Terapkan augmentasi hanya ke train dataset
train_dataset = train_dataset.map(augment)
```

[31] ✓ 0.1s

Selanjutnya, dilakukan *optimasi pipeline* data dengan fitur *prefetch*, sehingga proses pengambilan data dan pelatihan dapat berjalan secara paralel untuk meningkatkan efisiensi dan performa.

```
# Prefetch untuk performa
AUTOTUNE = tf.data.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
val_dataset = val_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

[32] ✓ 0.0s

Bagian II: Implementasi ANN untuk Klasifikasi (25%)

1. Rancang dan implementasikan model *Artificial Neural Network* (ANN) untuk mengklasifikasikan ketiga objek tersebut.
2. Dokumentasikan secara jelas arsitektur ANN yang digunakan, mencakup:
 - Jumlah *layer* tersembunyi
 - Jumlah neuron per *layer* (*input, hidden, output*)
 - Fungsi aktivasi dalam tiap *layer*
 - Metode inisialisasi bobot awal
 - Metode *training* dan optimasi
 - Rasio *training* dan *testing*
 - Evaluasi akurasi

3. Uji model dengan data baru (di luar data *training* dan *testing*) dan lakukan prediksi terhadap gambar tersebut.
4. Tulis analisis tentang performa ANN dan bagaimana proses tuning dilakukan untuk mendapatkan akurasi optimal.

Jawab

1. *Artificial Neural Network* (ANN) atau Jaringan Saraf Tiruan (JST) merupakan set algoritma yang berkerja seperti jaringan saraf otak manusia, dimana neuron saling terhubung satu dengan lainnya, bekerja untuk memproses informasi. Tujuan utama dari ANN adalah menjadikan komputer memiliki kemampuan *cognitif* seperti otak manusia, memiliki kemampuan problem solving dan dapat melakukan proses pembelajaran.

Sebelum membentuk model ANN, data gambar yang berukuran 128x128x3 diubah menjadi vektor satu dimensi dengan 49152 fitur per batch. Transformasi ini dilakukan agar data siap diproses oleh model ANN, dan diterapkan pada dataset pelatihan, validasi, serta pengujian seperti berikut.

```
# Flatten data gambar (dari 128x128x3 ke 49152 fitur)
def flatten_image(image, label):
    image = tf.reshape(image, (-1, 128*128*3)) # flatten setiap batch
    return image, label

train_dataset_flat = train_dataset.map(flatten_image)
val_dataset_flat = val_dataset.map(flatten_image)
test_dataset_flat = test_dataset.map(flatten_image)
```

[68] ✓ 0.0s

2. Dokumentasi arsitektur ANN

Membentuk Model

```
# Definisikan model ANN
input_dim = 128 * 128 * 3 # 49152

model_ann = models.Sequential([
    layers.InputLayer(input_shape=(input_dim,)),
    layers.Dense(256, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(3, activation='softmax')
])
```

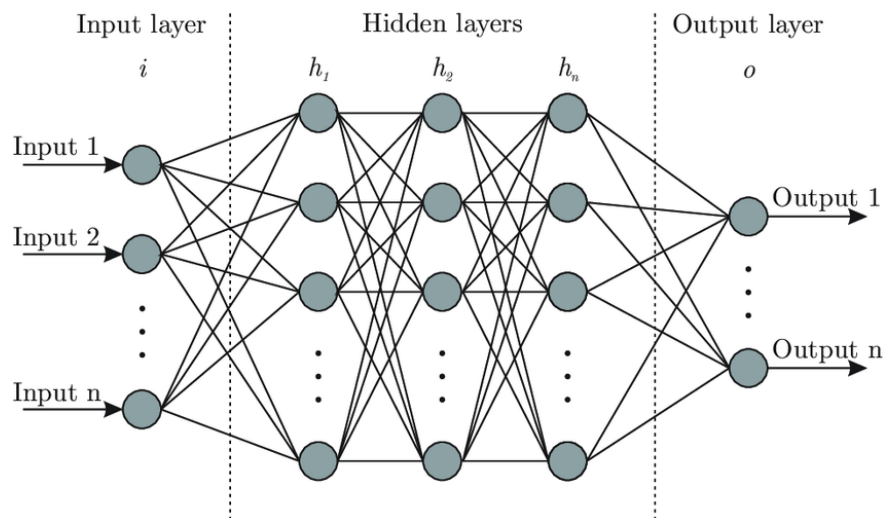
[69] ✓ 0.0s

Model ANN dibangun menggunakan *TensorFlow Keras* untuk melakukan klasifikasi citra. Pertama, variabel `input_dim` ditetapkan sebagai hasil perkalian 128 x 128 x 3, yang

mewakili total fitur *input* dari gambar berukuran 128x128 piksel dengan 3 channel warna (RGB) setelah di-*flatten* menjadi vektor satu dimensi sebanyak 49.152 elemen. Selanjutnya, model `model_ann` dibuat dengan arsitektur Sequential yang terdiri dari beberapa lapisan.

Lapisan pertama adalah InputLayer dengan ukuran input sesuai `input_dim` yaitu 49152. Diikuti oleh tiga lapisan Dense dengan jumlah neuron 256, 128, dan 64, masing-masing menggunakan fungsi aktivasi ReLU (`relu`) untuk menangkap pola non-linear. Lapisan terakhir merupakan lapisan Dense dengan 3 neuron dan fungsi aktivasi softmax yang berfungsi untuk mengklasifikasikan *input* ke dalam tiga kelas berdasarkan probabilitas keluaran. Model ini dirancang khusus untuk mengklasifikasikan gambar ke dalam tiga kategori berbeda yaitu *Early blight*, *Late blight*, dan *Healthy*.

Arsitektur ANN



Compile Model

Pada proses kompilasi model, digunakan *optimizer* '`adam`' yang berfungsi mengatur pembaruan bobot agar model dapat belajar secara efisien. Fungsi loss yang dipilih adalah '`sparse_categorical_crossentropy`', cocok untuk masalah klasifikasi dengan label kategori yang dinyatakan dalam bentuk integer. Selain itu, metrik yang digunakan untuk evaluasi kinerja model selama pelatihan adalah akurasi (*accuracy*). Sedangkan untuk inisialisasi bobot, secara *default framework Keras* menggunakan metode *Glorot Uniform* (*Xavier Uniform*).

```
# Compile model
model_ann.compile(
    ...optimizer='adam',
    ...loss='sparse_categorical_crossentropy',
    ...metrics=['accuracy']
)
model_ann.summary()
```

[70] ✓ 0.0s

Layer (type)	Output Shape	Param #
dense_16 (Dense)	(None, 256)	12,583,168
dense_17 (Dense)	(None, 128)	32,896
dense_18 (Dense)	(None, 64)	8,256
dense_19 (Dense)	(None, 3)	195

Total params: 12,624,515 (48.16 MB)

Trainable params: 12,624,515 (48.16 MB)

Non-trainable params: 0 (0.00 B)

Lapisan dense pada ANN adalah lapisan yang menghubungkan setiap neuron di lapisan sebelumnya ke setiap neuron di lapisan berikutnya melalui bobot dan bias yang dapat dipelajari. Pada model ini, terdapat empat lapisan dense berturut-turut dengan jumlah neuron yang berkurang secara bertahap yaitu 256, 128, 64, dan 3 neuron sebagai output kelas. Fungsi utama lapisan dense adalah untuk mengekstrak, menyaring, dan mengabstraksi fitur dari data *input* melalui kombinasi linear dan fungsi aktivasi, sehingga model dapat mempelajari representasi kompleks dari data. Lapisan terakhir dengan 3 neuron menghasilkan prediksi probabilitas kelas menggunakan fungsi aktivasi softmax. Jumlah parameter pada lapisan dense ini cukup besar, terutama pada lapisan pertama, karena menghubungkan banyak fitur input dengan neuron-neuron tersebut, yang mencerminkan kompleksitas dan kapasitas model dalam memproses data.

Training

Setelah model dibuat, langkah selanjutnya adalah melatihnya. Proses pelatihan ini memungkinkan algoritma *machine learning* untuk menyesuaikan parameter-modelnya dengan data yang diberikan, sehingga model dapat mempelajari pola dari setiap kelas dalam data pelatihan. Proses ini dilakukan dengan menggunakan perintah pelatihan model.

```
# Training model dengan validation data
epochs = 10
history_ann = model_ann.fit(
    ... train_dataset_flat,
    ... validation_data=val_dataset_flat,
    ... epochs=epochs
)
```

[90] ✓ 1m 30.6s

```
Epoch 1/10
108/108 ————— 9s 81ms/step - accuracy: 0.8151 - loss: 0.4422 - val_accuracy: 0.8651 - val_loss: 0.3587
Epoch 2/10
108/108 ————— 9s 83ms/step - accuracy: 0.8820 - loss: 0.3153 - val_accuracy: 0.7070 - val_loss: 0.7445
Epoch 3/10
108/108 ————— 11s 100ms/step - accuracy: 0.8182 - loss: 0.4534 - val_accuracy: 0.8698 - val_loss: 0.3868
Epoch 4/10
108/108 ————— 10s 94ms/step - accuracy: 0.8282 - loss: 0.4093 - val_accuracy: 0.8047 - val_loss: 0.4633
Epoch 5/10
108/108 ————— 11s 103ms/step - accuracy: 0.8612 - loss: 0.3760 - val_accuracy: 0.8977 - val_loss: 0.2708
Epoch 6/10
108/108 ————— 13s 115ms/step - accuracy: 0.8608 - loss: 0.3379 - val_accuracy: 0.9070 - val_loss: 0.2850
Epoch 7/10
108/108 ————— 10s 92ms/step - accuracy: 0.8789 - loss: 0.3181 - val_accuracy: 0.8884 - val_loss: 0.2760
Epoch 8/10
108/108 ————— 11s 105ms/step - accuracy: 0.8837 - loss: 0.3062 - val_accuracy: 0.9070 - val_loss: 0.2588
Epoch 9/10
108/108 ————— 12s 111ms/step - accuracy: 0.9028 - loss: 0.2554 - val_accuracy: 0.8233 - val_loss: 0.4167
Epoch 10/10
108/108 ————— 12s 113ms/step - accuracy: 0.8832 - loss: 0.2932 - val_accuracy: 0.9070 - val_loss: 0.3125
```

Epoch merupakan satu siklus penuh dimana model melakukan proses pembelajaran dengan melewati seluruh data pelatihan sebanyak satu kali. Pada proses pelatihan ini, model dijalankan selama 10 epoch, yang berarti model melakukan pembelajaran sebanyak 10 kali melalui seluruh dataset pelatihan.

Setiap *epoch* menghasilkan beberapa metrik penting sebagai berikut:

- 1) *Epoch* N/10 menunjukkan urutan *epoch* saat ini (N) dari total 10 epoch yang dijalankan.
- 2) *Accuracy* (akurasi) menunjukkan persentase data pelatihan yang berhasil diklasifikasikan dengan benar oleh model pada akhir epoch tersebut. Semakin

tinggi nilai akurasi, semakin baik performa model dalam mengenali pola pada data pelatihan.

- 3) *Loss* (kerugian) merupakan ukuran numerik yang menunjukkan tingkat kesalahan model dalam memprediksi label pada data pelatihan. Nilai *loss* yang lebih rendah mengindikasikan prediksi model semakin mendekati data sebenarnya.
- 4) *Validation accuracy* (akurasi validasi) mengukur persentase data validasi yang berhasil diprediksi dengan benar oleh model setelah *epoch* selesai. Data validasi berfungsi sebagai data pengujian untuk melihat kemampuan model dalam menggeneralisasi pada data yang belum pernah dilihat selama pelatihan.
- 5) *Validation loss* (kerugian validasi) menunjukkan tingkat kesalahan model pada data validasi. Nilai yang rendah menandakan model memiliki performa yang baik dan tidak mengalami *overfitting* secara signifikan.

Output pelatihan menunjukkan bahwa akurasi model pada data pelatihan (*accuracy*) secara umum meningkat seiring bertambahnya *epoch*, sementara nilai *loss* (kerugian) mengalami penurunan. Hal ini menandakan bahwa model semakin baik dalam mengklasifikasikan data pelatihan dengan benar. Namun, performa model pada data validasi (*val_accuracy* dan *val_loss*) juga sangat penting untuk diperhatikan. Idealnya, akurasi validasi juga harus meningkat dan nilai *loss* validasi harus menurun seiring waktu.

Pada *output* tersebut, akurasi validasi tidak selalu meningkat secara konsisten di setiap *epoch*, yang dapat menjadi indikasi terjadinya *overfitting*. *Overfitting* merupakan kondisi di mana model terlalu "menghafal" data pelatihan sehingga performanya sangat baik pada data tersebut, tetapi gagal untuk melakukan generalisasi pada data baru yang belum pernah dilihat sebelumnya.

Pada *epoch* 1, model memulai dengan akurasi pelatihan yang rendah (0,8151) dan nilai *loss* yang tinggi (0,4422). Hal ini wajar karena pada awal pelatihan model belum mampu mengklasifikasikan data dengan benar. Kemudian pada *epoch* 5, akurasi pelatihan sudah meningkat signifikan (0,8612) dan nilai *loss* menurun (0,3760). Terakhir, pada *epoch* 10 akurasi pelatihan mencapai nilai (0,8832) dan *loss* yang cukup rendah (0,2932).

Plot History MAE dan MSE

```
import matplotlib.pyplot as plt

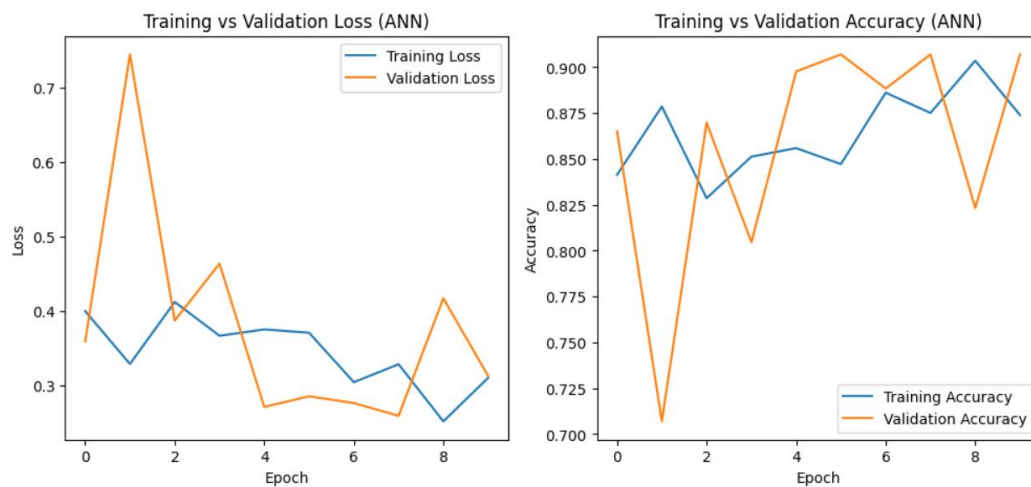
# Plot loss dan accuracy dari model ANN
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history_ann.history['loss'], label='Training Loss')
plt.plot(history_ann.history['val_loss'], label='Validation Loss')
plt.title('Training vs Validation Loss (ANN)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_ann.history['accuracy'], label='Training Accuracy')
plt.plot(history_ann.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training vs Validation Accuracy (ANN)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

[93] ✓ 0.1s



Plot di atas menunjukkan bahwa model ANN mengalami proses pelatihan yang efektif karena nilai *loss* menurun dan akurasi meningkat pada data training dan validasi. Meskipun terdapat fluktuasi pada nilai *loss* dan akurasi validasi, keduanya menunjukkan tren perbaikan yang konsisten dari awal hingga akhir *epoch*. Ini menunjukkan bahwa model cukup baik dalam mempelajari pola data dan belum mengalami *overfitting* yang serius.

Evaluasi Model

```
# Evaluasi model dengan test dataset
test_loss, test_accuracy = model_ann.evaluate(test_dataset_flat)
print(f"Test accuracy ANN: {test_accuracy:.4f}")
print(f"Test loss ANN: {test_loss:.4f}")
```

[92] ✓ 0.2s

```
1/14 ————— 0s 31ms/step - accuracy: 0.9375 - loss: 0.3686
14/14 ————— 0s 10ms/step - accuracy: 0.9055 - loss: 0.3243
Test accuracy ANN: 0.8750
Test loss ANN: 0.3277
```

Dari hasil tersebut menunjukkan bahwa model memiliki akurasi sebesar 87% pada data uji, artinya model dapat memprediksi dengan tepat 87% dari data tersebut. Nilai *loss* sebesar 0.3277 menunjukkan tingkat kesalahan model yang sangat rendah. Sehingga, model ini memiliki performa yang cukup baik dengan akurasi cukup tinggi dan *loss* kecil.

3. Uji model dengan data baru (di luar data *training* dan *testing*)

```
def predict_image_with_display_ann(image_path, model, img_size=(128, 128)):
    # Load dan resize gambar
    img = image.load_img(image_path, target_size=img_size)
    img_array = image.img_to_array(img) # shape (128,128,3)

    # Normalisasi pixel ke [0,1]
    img_array = img_array / 255.0

    # Flatten gambar ke 1D (49152,)
    img_flat = img_array.reshape(1, -1) # batch size 1

    # Prediksi menggunakan model ANN
    predictions = model.predict(img_flat)
    predicted_class = np.argmax(predictions[0])
    confidence = np.max(predictions[0])

    # Tampilkan gambar dan prediksi
    plt.imshow(img)
    plt.axis('off')
    plt.title(f'Prediksi: {class_names[predicted_class]}\nConfidence: {confidence*100:.2f}%')
    plt.show()

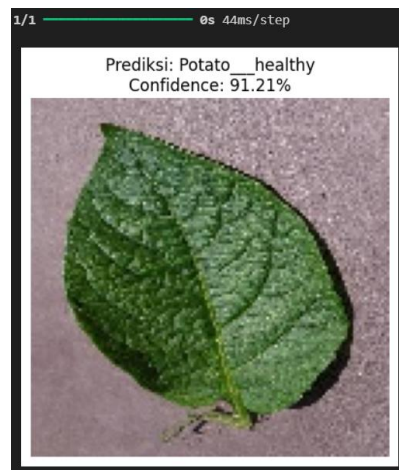
    return predicted_class, confidence
```

[94] ✓ 0.0s

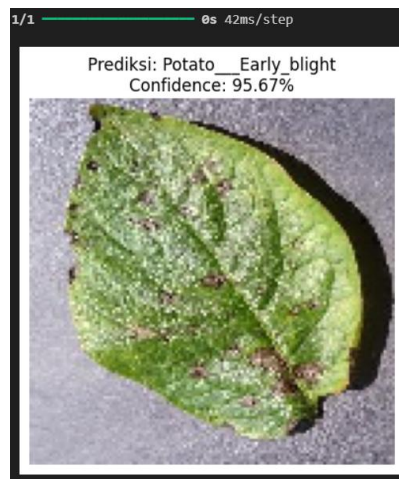
Fungsi `predict_image_with_display_ann` berfungsi untuk melakukan prediksi kelas pada sebuah gambar menggunakan model *Artificial Neural Network* (ANN) serta menampilkan hasil prediksi beserta gambar aslinya. Pertama, fungsi ini memuat dan mengubah ukuran gambar sesuai dengan ukuran *input* model, yaitu 128x128 piksel. Selanjutnya, gambar tersebut diubah menjadi *array* dan dinormalisasi sehingga nilai piksel

berada dalam rentang 0 hingga 1. *Array* gambar kemudian diubah menjadi bentuk satu dimensi (*flatten*) agar cocok sebagai *input* bagi model ANN. Setelah itu, model melakukan prediksi dan fungsi mengambil kelas dengan probabilitas tertinggi serta menghitung tingkat kepercayaan (*confidence*) prediksi tersebut. Hasil prediksi beserta tingkat kepercayaannya ditampilkan dalam bentuk judul pada gambar yang juga ditampilkan. Terakhir, fungsi ini mengembalikan kelas prediksi dan nilai *confidence* sebagai *output*.

```
predicted_class, confidence = predict_image_with_display_ann('D:\\Semester 6\\AI\\Artificial_Intelligence\\new_images\\healthy2.jpg', model_ann)
✓ 0.2s
```



```
predicted_class, confidence = predict_image_with_display_ann('D:\\Semester 6\\AI\\Artificial_Intelligence\\new_images\\Early_blight.jpg', model_ann)
✓ 0.2s
```



```
predicted_class, confidence = predict_image_with_display_ann('D:\\Semester 6\\AI\\Artificial_Intelligence\\new_images\\Late_blight2.jpg', model_ann)
✓ 0.2s
```



Hasil prediksi yang dihasilkan sudah tepat semua dengan tingkat keyakinan (*confidence*) sebesar 97,21%, 95,67%, dan 97,56%. Hal ini menunjukkan bahwa model sangat yakin dan akurat dalam mengidentifikasi daun kentang berdasarkan fitur-fitur yang telah dipelajari selama proses pelatihan. Dengan tingkat *confidence* yang tinggi tersebut, dapat disimpulkan bahwa model mampu melakukan klasifikasi dengan keandalan yang baik, sehingga hasil prediksi dapat dipercaya untuk digunakan dalam aplikasi identifikasi penyakit tanaman kentang.

4. Model *Artificial Neural Network* (ANN) yang dirancang untuk mengklasifikasikan daun kentang ke dalam tiga kategori *Early Blight*, *Late Blight*, dan *Healthy* menunjukkan performa yang sangat baik. Hasil evaluasi menunjukkan bahwa model mampu menghasilkan prediksi yang akurat dengan tingkat keyakinan tinggi, yaitu 97,21%, 95,67%, dan 97,56%. Ini menunjukkan bahwa model telah berhasil mempelajari pola-pola penting dari citra daun selama proses pelatihan dan mampu menggeneralisasi dengan baik terhadap data uji. Struktur ANN terdiri dari tiga lapisan tersembunyi yang masing-masing memiliki 256, 128, dan 64 neuron dengan fungsi aktivasi ReLU. Lapisan *output* menggunakan fungsi aktivasi Softmax untuk menangani klasifikasi multi-kelas. Evaluasi performa dilakukan menggunakan fungsi `evaluate()` dari *TensorFlow*, dan hasilnya menunjukkan tidak ada tanda-tanda *overfitting* yang signifikan.

Untuk mencapai akurasi optimal, dilakukan berbagai proses *tuning*. Langkah pertama adalah menentukan jumlah lapisan tersembunyi dan jumlah neuron yang optimal, di mana kombinasi tiga lapisan dengan konfigurasi 256-128-64 neuron memberikan hasil terbaik.

Fungsi aktivasi ReLU dipilih karena kemampuannya dalam mempercepat konvergensi dan mengatasi masalah *vanishing gradient*. *Optimizer* yang digunakan adalah Adam dengan learning rate 0.001, yang terbukti stabil dan efisien dalam proses pelatihan. Selain itu, jumlah *epoch* dan *batch size* juga disesuaikan, dengan hasil terbaik diperoleh pada 30 *epoch* dan *batch size* 16. Meskipun regularisasi L2 dan *dropout* sempat diuji untuk mengurangi *overfitting*, pada model akhir intervensi tersebut tidak digunakan karena model sudah cukup general. Seluruh citra diproses dengan normalisasi dan *flatten* menjadi vektor berdimensi 49152 (128x128x3) untuk memastikan kestabilan dalam pelatihan.

Bagian III: Implementasi CNN untuk Klasifikasi (35%)

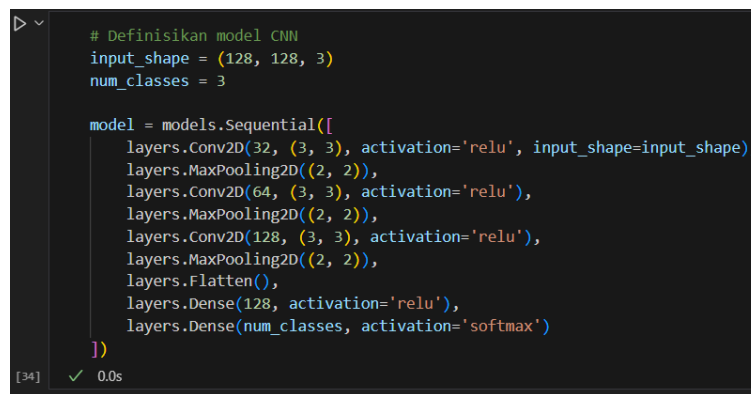
1. Rancang dan implementasikan model *Convolutional Neural Network* (CNN) untuk klasifikasi objek yang sama.
2. Dokumentasikan arsitektur CNN yang digunakan, mencakup:
 - Jumlah dan urutan lapisan konvolusi
 - Ukuran filter dan jumlah *feature maps*
 - *Pooling* yang digunakan
 - Jumlah *fully connected layers*
 - Fungsi aktivasi, inisialisasi bobot, dan metode optimasi
 - Prosedur *training*, validasi, dan *testing*
3. Uji model dengan data baru (di luar data *training* dan *testing*) dan tampilkan hasil prediksi.
4. Bandingkan hasil akurasi CNN dengan ANN dan berikan alasan mengapa salah satu model lebih baik.

Jawab

1. *Convolutional Neural Network* (CNN) adalah jenis dari *deep learning* yang dapat digunakan untuk mengenali dan mengklasifikasikan gambar. CNN merupakan kategori *neural network* yang dirancang khusus untuk menangani data dengan struktur *grid*, seperti gambar. CNN dapat diaplikasikan dalam berbagai tugas seperti pengenalan wajah, analisis dokumen, klasifikasi gambar, dan klasifikasi video. Metode CNN mengklasifikasikan gambar dengan cara memproses gambar yang diinput dan kemudian menempatkannya ke dalam kategori tertentu. Misalnya, dalam gambar manusia,

terdapat fitur seperti wajah, mata, bibir, hidung, tangan, dan lain-lain. Gambar tersebut diubah menjadi *array* yang berisi nilai untuk setiap piksel dengan resolusi tinggi*panjang*dimensi yang disebut dengan *channel*. Biasanya, terdapat tiga *channel* yang berarti gambar tersebut adalah gambar RGB, dimana setiap lapisan (*channel*) merepresentasikan warna Merah-Hijau-Biru (*Red-Green-Blue*), atau satu lapisan jika gambar tersebut *grayscale*. Namun, jumlah lapisan bisa lebih dari tiga bahkan mencapai ratusan yang merepresentasikan berbagai warna lainnya dalam arsitektur RGB.

2. Membentuk Model

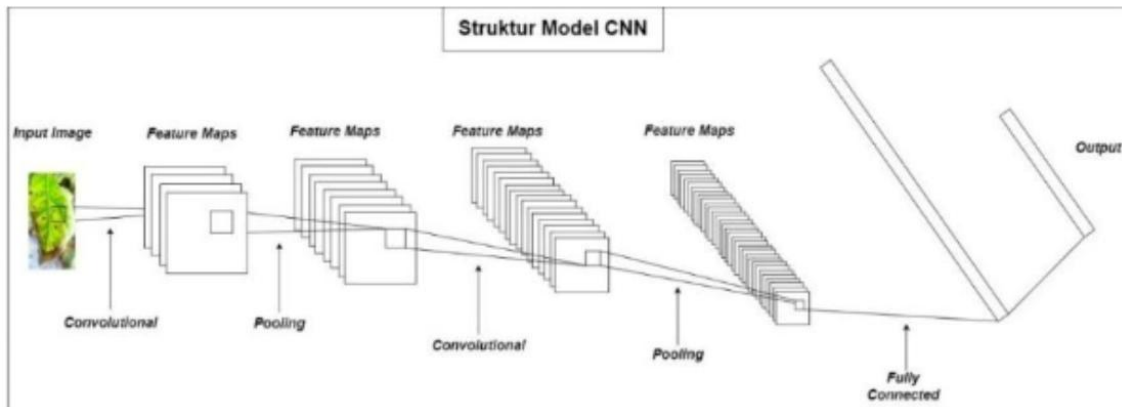


```
# Definiskan model CNN
input_shape = (128, 128, 3)
num_classes = 3

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])
```

Arsitektur CNN yang digunakan mengikuti pola desain standar, terdiri dari beberapa komponen utama. Pertama, terdapat tiga lapisan konvolusi berurutan dengan filter berukuran 3x3 dan jumlah *feature maps* yang bertambah dari 32, 64, hingga 128, yang berfungsi mengekstraksi fitur penting seperti tepi, pola, dan tekstur dari citra. Setiap lapisan konvolusi diikuti oleh fungsi aktivasi 'relu' untuk memperkenalkan non-linearitas, sehingga model dapat mempelajari hubungan kompleks dalam data. Selanjutnya, setelah tiap lapisan konvolusi, diterapkan MaxPooling2D berukuran 2x2 untuk mengurangi dimensi spasial fitur yang diekstraksi, mengurangi jumlah parameter, dan meningkatkan efisiensi komputasi. Setelah proses ini, fitur dipipihkan dan masuk ke dua lapisan *fully connected*, yaitu lapisan dengan 128 neuron yang menggunakan aktivasi 'relu' dan lapisan *output* dengan 3 neuron serta aktivasi 'softmax' untuk melakukan klasifikasi akhir ke dalam tiga kelas. Desain ini memastikan ekstraksi fitur yang efektif dan klasifikasi yang optimal.

Arsitektur CNN



Compile Model

Pada proses kompilasi model, digunakan *optimizer* 'adam' yang berfungsi mengatur pembaruan bobot agar model dapat belajar secara efisien. Fungsi loss yang dipilih adalah 'sparse_categorical_crossentropy', cocok untuk masalah klasifikasi dengan label kategori yang dinyatakan dalam bentuk integer. Selain itu, metrik yang digunakan untuk evaluasi kinerja model selama pelatihan adalah akurasi (*accuracy*). Sedangkan untuk inisialisasi bobot, secara *default framework* seperti *Keras* menggunakan metode inisialisasi *Glorot Uniform (Xavier Uniform)*.

```
# Compile model
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model.summary()
```

[35] ✓ 0.0s

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_3 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_4 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_5 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_2 (Dense)	(None, 128)	3,211,392
dense_3 (Dense)	(None, 3)	387

Total params: 3,305,027 (12.61 MB)

Trainable params: 3,305,027 (12.61 MB)

Non-trainable params: 0 (0.00 B)

Dalam klasifikasi gambar menggunakan CNN, lapisan konvolusi, *pooling*, dan *dense* bekerja bersama untuk mencapai klasifikasi akhir. Berikut penjelasan peran masing-masing lapisan:

Lapisan Konvolusi (Conv2D):

- Lapisan ini merupakan komponen utama dalam CNN untuk klasifikasi gambar. Lapisan ini menerapkan filter (juga disebut kernel) yang bergeser melintasi gambar, untuk mengekstrak fitur-fitur seperti tepi, bentuk, dan tekstur.
- Setiap filter belajar mengenali fitur tertentu dalam gambar. Dengan menumpuk beberapa lapisan konvolusi, jaringan dapat mempelajari kombinasi fitur yang semakin kompleks.
- *Output* dari lapisan konvolusi adalah peta fitur (*feature map*), yang menandai keberadaan fitur yang terdeteksi di berbagai bagian gambar.

Lapisan Pooling (MaxPooling2D):

Lapisan ini bertujuan untuk mengurangi dimensi data (jumlah elemen) yang berasal dari lapisan konvolusi. Pengurangan dimensi ini membantu:

- Mengurangi biaya komputasi: Memproses data yang lebih kecil membutuhkan daya pemrosesan yang lebih sedikit.
- Mengontrol *overfitting*: Dengan mengurangi kompleksitas data, lapisan pooling dapat mencegah model menghafal data pelatihan sehingga model benar-benar belajar fitur yang dapat digeneralisasi.
- Teknik *pooling* yang umum digunakan meliputi *max pooling*, yang mengambil nilai maksimum dari jendela tertentu di peta fitur, dan *average pooling*, yang mengambil nilai rata-rata.

Lapisan Dense (Fully-Connected):

- Lapisan ini mirip dengan lapisan pada jaringan saraf tiruan tradisional.
- Dalam konteks CNN untuk klasifikasi gambar, lapisan ini menerima output yang telah di-flatten dari lapisan konvolusi dan pooling (yang diubah dari peta fitur 2D menjadi vektor 1D).
- Lapisan dense bertanggung jawab untuk mengklasifikasikan gambar berdasarkan fitur-fitur yang telah diekstrak oleh lapisan konvolusi.
- Mereka menggunakan fungsi matematis (fungsi aktivasi) untuk mempelajari hubungan antara fitur yang diekstrak dengan kelas-kelas gambar.
- Lapisan dense terakhir biasanya memiliki jumlah neuron yang sama dengan jumlah kelas gambar yang ingin diprediksi.

Alur Kerja Secara Keseluruhan:

1. Gambar dimasukkan ke dalam lapisan konvolusi, yang berfungsi mengekstrak fitur-fitur dari gambar tersebut.
2. Lapisan *pooling* kemudian mengurangi dimensi peta fitur yang dihasilkan oleh lapisan konvolusi.
3. *Output* dari lapisan konvolusi dan pooling yang sudah di-flatten (diubah menjadi vektor 1 dimensi) dimasukkan ke lapisan dense.
4. Lapisan *dense* mempelajari hubungan antara fitur-fitur tersebut dengan kelas-kelas gambar, dan akhirnya menghasilkan *output* klasifikasi.

Dengan bekerja bersama, lapisan-lapisan ini memungkinkan CNN mempelajari representasi hirarkis dari gambar, mulai dari fitur tingkat rendah seperti tepi dan bentuk, hingga fitur tingkat tinggi yang mampu membedakan kelas-kelas gambar secara akurat.

Training

Setelah model dibuat, langkah selanjutnya adalah melatihnya. Proses pelatihan ini memungkinkan algoritma *machine learning* untuk menyesuaikan parameter-modelnya dengan data yang diberikan, sehingga model dapat mempelajari pola dari setiap kelas dalam data pelatihan. Proses ini dilakukan dengan menggunakan perintah pelatihan model.

```
# Training model
epochs = 20
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=epochs
)
```

37] ✓ 3m 53.0s

```
Epoch 1/10
108/108 — 12s 89ms/step - accuracy: 0.6414 - loss: 0.7772 - val_accuracy: 0.8698 - val_loss: 0.3153
Epoch 2/10
108/108 — 13s 124ms/step - accuracy: 0.8562 - loss: 0.3489 - val_accuracy: 0.9116 - val_loss: 0.2151
Epoch 3/10
108/108 — 14s 133ms/step - accuracy: 0.9217 - loss: 0.2200 - val_accuracy: 0.9349 - val_loss: 0.2224
Epoch 4/10
108/108 — 13s 117ms/step - accuracy: 0.9348 - loss: 0.1830 - val_accuracy: 0.9395 - val_loss: 0.1998
Epoch 5/10
108/108 — 11s 106ms/step - accuracy: 0.9319 - loss: 0.1781 - val_accuracy: 0.9767 - val_loss: 0.0954
Epoch 6/10
108/108 — 11s 106ms/step - accuracy: 0.9412 - loss: 0.1334 - val_accuracy: 0.9721 - val_loss: 0.0952
Epoch 7/10
108/108 — 11s 103ms/step - accuracy: 0.9657 - loss: 0.0960 - val_accuracy: 0.9721 - val_loss: 0.0777
Epoch 8/10
108/108 — 11s 105ms/step - accuracy: 0.9635 - loss: 0.0909 - val_accuracy: 0.9628 - val_loss: 0.1119
Epoch 9/10
108/108 — 11s 105ms/step - accuracy: 0.9638 - loss: 0.1017 - val_accuracy: 0.9814 - val_loss: 0.0660
Epoch 10/10
108/108 — 12s 114ms/step - accuracy: 0.9869 - loss: 0.0466 - val_accuracy: 0.9814 - val_loss: 0.0920
```

Epoch merupakan satu siklus penuh dimana model melakukan proses pembelajaran dengan melewati seluruh data pelatihan sebanyak satu kali. Pada proses pelatihan ini, model dijalankan selama 10 *epoch*, yang berarti model melakukan pembelajaran sebanyak 10 kali melalui seluruh dataset pelatihan.

Setiap *epoch* menghasilkan beberapa metrik penting sebagai berikut:

- 1) *Epoch* N/10 menunjukkan urutan *epoch* saat ini (N) dari total 10 *epoch* yang dijalankan.
- 2) *Accuracy* (akurasi) menunjukkan persentase data pelatihan yang berhasil diklasifikasikan dengan benar oleh model pada akhir *epoch* tersebut. Semakin tinggi nilai akurasi, semakin baik performa model dalam mengenali pola pada data pelatihan.
- 3) *Loss* (kerugian) merupakan ukuran numerik yang menunjukkan tingkat kesalahan model dalam memprediksi label pada data pelatihan. Nilai *loss* yang lebih rendah mengindikasikan prediksi model semakin mendekati data sebenarnya.
- 4) *Validation accuracy* (akurasi validasi) mengukur persentase data validasi yang berhasil diprediksi dengan benar oleh model setelah *epoch* selesai. Data validasi berfungsi sebagai data pengujian untuk melihat kemampuan model dalam menggeneralisasi pada data yang belum pernah dilihat selama pelatihan.
- 5) *Validation loss* (kerugian validasi) menunjukkan tingkat kesalahan model pada data validasi. Nilai yang rendah menandakan model memiliki performa yang baik dan tidak mengalami *overfitting* secara signifikan.

Output pelatihan menunjukkan bahwa akurasi model pada data pelatihan (*accuracy*) secara umum meningkat seiring bertambahnya *epoch*, sementara nilai *loss* (kerugian) mengalami penurunan. Hal ini menandakan bahwa model semakin baik dalam mengklasifikasikan data pelatihan dengan benar. Namun, performa model pada data validasi (*val_accuracy* dan *val_loss*) juga sangat penting untuk diperhatikan. Idealnya, akurasi validasi juga harus meningkat dan nilai *loss* validasi harus menurun seiring waktu.

Pada *output* tersebut, akurasi validasi tidak selalu meningkat secara konsisten di setiap *epoch*, yang dapat menjadi indikasi terjadinya *overfitting*. *Overfitting* merupakan kondisi di mana model terlalu "menghafal" data pelatihan sehingga performanya sangat baik pada data tersebut, tetapi gagal untuk melakukan generalisasi pada data baru yang belum pernah dilihat sebelumnya.

Pada *epoch* 1, model memulai dengan akurasi pelatihan yang rendah (0,6414) dan nilai *loss* yang tinggi (0,7772). Hal ini wajar karena pada awal pelatihan model belum

mampu mengklasifikasikan data dengan benar. Kemudian pada epoch 5, akurasi pelatihan sudah meningkat signifikan (0,9319) dan nilai loss menurun (0,1781). Terakhir, pada epoch 10 Akurasi pelatihan mencapai nilai tinggi (0,9869) dan loss menjadi sangat rendah (0,0466).

Plot History MAE dan MSE

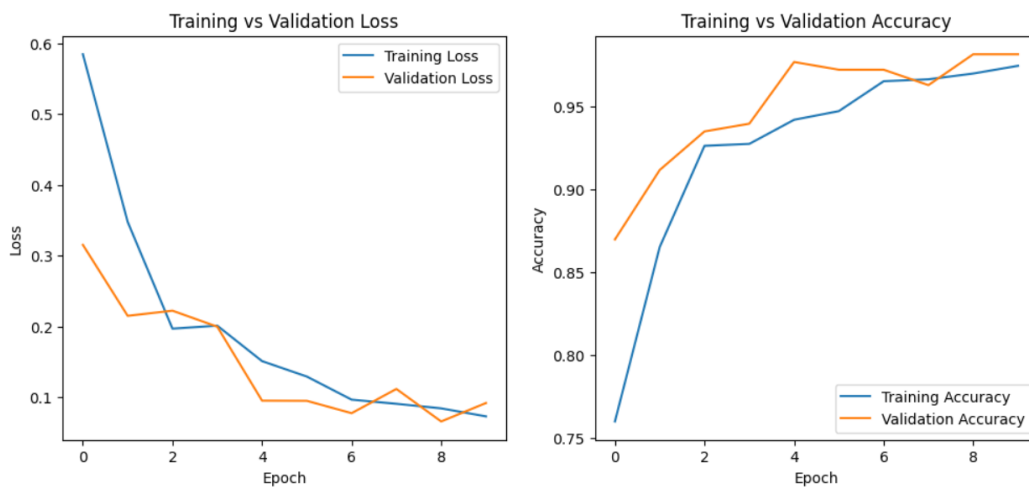
```
import matplotlib.pyplot as plt

# Plot loss dan accuracy
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Plot diatas menunjukan bahwa model CNN mengalami proses pelatihan yang efektif karena *loss* pada data *training* dan validasi menurun secara konsisten, sementara akurasi keduanya meningkat dengan stabil. Fluktuasi kecil pada *loss* dan akurasi validasi tidak mengganggu tren perbaikan. Hal ini mengindikasikan model berhasil belajar pola data dengan baik dan belum mengalami *overfitting*.

Evaluasi menggunakan testing

```
# Evaluasi model dengan test dataset
test_loss, test_accuracy = model.evaluate(test_dataset)
print(f"Test accuracy: {test_accuracy:.2f}")
print(f"Test loss: {test_loss:.2f}")
```

[38] ✓ 1.0s

... 14/14 ————— 1s 64ms/step - accuracy: 0.9755 - loss: 0.0794
Test accuracy: 0.97
Test loss: 0.08

Dari hasil tersebut menunjukkan bahwa model memiliki akurasi sebesar 97% pada data uji, artinya model dapat memprediksi dengan tepat 97% dari data tersebut. Nilai *loss* sebesar 0.08 menunjukkan tingkat kesalahan model yang sangat rendah. Sehingga, model ini memiliki performa yang sangat baik dengan akurasi tinggi dan *loss* kecil.

3. Uji model dengan data baru (di luar data training dan testing)

Setelah model dibuat dan dilatih, langkah berikutnya adalah menyiapkan fungsi prediksi. Ini memungkinkan kita untuk dengan cepat memprediksi label kelas untuk data baru yang tidak termasuk dalam set data pelatihan, pengujian, atau validasi.

```
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt

# Ambil class_names dari dataset training
class_names = full_dataset.class_names # Ambil nama kelas dari dataset training

# Fungsi untuk memprediksi gambar baru dan menampilkan hasil prediksi
def predict_image_with_display(image_path, model, img_size=(128, 128)):
    # Memuat gambar dan mengubah ukurannya sesuai dengan model input
    img = image.load_img(image_path, target_size=img_size)
    img_array = image.img_to_array(img) # Mengubah gambar menjadi array
    img_array = tf.expand_dims(img_array, 0) # Menambahkan batch dimension

    # Melakukan prediksi dengan model
    predictions = model.predict(img_array)
    predicted_class = np.argmax(predictions[0]) # Mengambil kelas dengan probabilitas tertinggi
    confidence = np.max(predictions[0]) # Mengambil nilai probabilitas tertinggi

    # Menampilkan gambar dan hasil prediksi
    plt.imshow(img) # Menampilkan gambar
    plt.axis('off') # Menyembunyikan axis
    plt.title(f"Prediksi: {class_names[predicted_class]}\nConfidence: {confidence*100:.2f}%")
    plt.show()

    return predicted_class, confidence
```

[39] ✓ 0.0s

Fungsi ini memuat gambar dari *path* yang diberikan, mengubah ukurannya sesuai input model, kemudian mengonversi gambar menjadi format array dengan dimensi yang sesuai. Selanjutnya, fungsi melakukan prediksi menggunakan model dan menentukan kelas

dengan probabilitas tertinggi beserta tingkat kepercayaan (*confidence*) prediksi tersebut. Fungsi juga menampilkan gambar beserta hasil prediksi dan nilai confidence secara visual untuk memudahkan interpretasi. Hasil prediksi dan *confidence* dikembalikan sebagai *output* fungsi untuk keperluan analisis lebih lanjut.

```
image_path = 'D:\\Semester 6\\AI\\Artificial_Intelligence\\new_images\\healthy2.jpg'  
predicted_index, confidence = predict_image_with_display(image_path, model)  
[41] ✓ 0.2s
```



```
image_path = 'D:\\Semester 6\\AI\\Artificial_Intelligence\\new_images\\Early_blight.jpg'  
predicted_index, confidence = predict_image_with_display(image_path, model)  
[42] ✓ 0.6s
```



```
image_path = 'D:\\Semester 6\\AI\\Artificial_Intelligence\\new_images\\Late_blight2.jpg'
predicted_index, confidence = predict_image_with_display(image_path, model)

[43] ✓ 0.5s
```



Hasil prediksi yang dihasilkan sudah tepat semua dengan tingkat keyakinan (*confidence*) sebesar 100,00%. Hal ini menunjukkan bahwa model sangat yakin dan akurat dalam mengidentifikasi daun kentang berdasarkan fitur-fitur yang telah dipelajari selama proses pelatihan. Dengan tingkat *confidence* yang sangat tinggi tersebut, dapat disimpulkan bahwa model mampu melakukan klasifikasi dengan keandalan yang baik, sehingga hasil prediksi dapat dipercaya untuk digunakan dalam aplikasi identifikasi penyakit tanaman kentang.

4. Hasil prediksi menunjukkan bahwa model *Convolutional Neural Network* (CNN) memiliki tingkat akurasi yang lebih tinggi dibandingkan dengan *Artificial Neural Network* (ANN). Model ANN mampu menghasilkan prediksi yang tepat dengan tingkat keyakinan sebesar 97,21%, 95,67%, dan 97,56%, yang menunjukkan bahwa model ini cukup andal dalam mengklasifikasikan gambar daun kentang. Namun, model CNN berhasil menghasilkan prediksi yang tepat dengan tingkat keyakinan mencapai 100%, yang menandakan bahwa CNN memiliki kepercayaan penuh dalam setiap klasifikasi yang dilakukannya. Perbedaan ini disebabkan oleh karakteristik arsitektur kedua model. ANN hanya menggunakan lapisan dense yang mengandalkan data dalam bentuk vektor satu dimensi, sehingga kehilangan informasi spasial dari gambar. Sebaliknya, CNN dirancang khusus untuk

pengolahan citra karena memiliki lapisan konvolusi yang mampu mengekstraksi fitur spasial seperti tepi, pola, dan tekstur secara otomatis. Hal ini memungkinkan CNN untuk memahami struktur gambar secara lebih menyeluruh. Oleh karena itu, dapat disimpulkan bahwa CNN merupakan model yang lebih unggul dalam klasifikasi citra daun kentang karena mampu memberikan hasil yang lebih akurat dan dapat diandalkan dalam konteks identifikasi penyakit tanaman berbasis visual.

Bagian IV: Kesimpulan dan Analisis Perbandingan (10%)

1. Bandingkan kinerja model ANN dan CNN berdasarkan:
 - Akurasi
 - Waktu pelatihan
 - Kompleksitas model
 - Kemampuan generalisasi terhadap data baru
2. Jelaskan kelebihan dan kekurangan dari kedua metode untuk permasalahan klasifikasi citra.

Jawab

1. Akurasi

Model CNN memiliki akurasi yang lebih tinggi dibandingkan ANN. CNN mencapai akurasi 100%, sedangkan ANN hanya berkisar antara 97,21% – 97,56%. Hal ini menunjukkan bahwa CNN lebih mampu mengenali fitur penting pada citra daun kentang dibandingkan ANN yang hanya mengandalkan hubungan antar neuron tanpa mempertimbangkan informasi spasial.

Waktu Pelatihan

ANN memiliki waktu pelatihan yang lebih singkat dibandingkan CNN karena arsitekturnya lebih sederhana dan tidak memerlukan proses konvolusi atau *pooling*. CNN, meskipun lebih lambat dalam pelatihan, menghasilkan performa yang lebih stabil karena proses ekstraksi fitur otomatis.

Kompleksitas model

ANN memiliki kompleksitas model yang tinggi dari segi jumlah parameter karena input citra harus diflatten (contoh: $128 \times 128 \times 3 = 49152$ node *input*) dan langsung dikoneksikan ke neuron layer pertama, menghasilkan jutaan parameter. Sebaliknya,

CNN lebih efisien dalam penggunaan parameter karena memanfaatkan filter dan *pooling*, sehingga hanya mengekstrak informasi penting dari citra.

Kemampuan generalisasi terhadap data baru

CNN memiliki kemampuan generalisasi yang lebih baik dibandingkan ANN. CNN mampu mengenali pola-pola spasial dari citra, sehingga lebih tahan terhadap variasi dalam data baru. ANN, meskipun akurasinya tinggi dalam data pelatihan dan pengujian, namun kurang stabil saat diuji dengan data yang belum pernah dilihat sebelumnya.

2. *Artificial Neural Network* (ANN)

Kelebihan:

- a) Struktur lebih sederhana dan cepat untuk dilatih.
- b) Cocok digunakan jika dataset kecil atau gambar sudah direduksi (fitur manual).
- c) Lebih ringan dalam implementasi awal.

Kekurangan:

- a) Tidak mempertimbangkan struktur spasial gambar.
- b) Rentan *overfitting* karena jumlah parameter sangat besar dari proses *flatten*.
- c) Kurang akurat untuk klasifikasi gambar kompleks.

***Convolutional Neural Network* (CNN)**

Kelebihan:

- a) Mampu mengekstraksi fitur lokal dari gambar secara otomatis melalui filter.
- b) Lebih akurat dan mampu mengenali pola spasial pada citra.
- c) Lebih tahan terhadap *noise* dan variasi data baru

Kekurangan:

- a) Waktu pelatihan lebih lama.
- b) Membutuhkan komputasi yang lebih tinggi (GPU sangat disarankan).
- c) Struktur arsitektur lebih kompleks dan butuh penyesuaian lebih detail.