

Highlight Model Experiment

24 Januari 2025

No	Nama Model	Akurasi	Precision	Recall	F1 Score
1	YoloV5 + CSPDarknet53		0.288	0.332	0.26
2	Simple CNN	Train: 73.38% Test:	Train: 0.7386 Test: 0.7622	Train: 0.7338 Test: 0.7595	Train: 0.7291 Test: 0.7564
3	CNN + ResNet18	Train: 95.03% Test: 75.95%	Train: 0.9504 Test: 0.7798	Train: 0.9503 Test: 0.7798	Train: 0.9503 Test: 0.7750
4	Fine-Tunning CNN + ResNet18	Train: 91.97% Test: 77.98%	Train: 0.9208 Test: 0.7798	Train: 0.9197 Test: 0.7798	Train: 0.9197 Test: 0.7750
5	CNN + ResNet50	Train: 93.25% Test: 80.23%	Train: 0.9331 Test: 0.7995	Train: 0.9325 Test: 0.8023	Train: 0.9325 Test: 0.7995
6	CNN + EfficientNet-B0	Train: 95.43% Test: 80.34%	Train: 0.9553 Test: 0.8036	Train: 0.9543 Test: 0.8034	Train: 0.9543 Test: 0.7992
7	CNN + EfficientNet-B3	Train: 98.31% Test: 79.85%	Train: 0.9832 Test: 0.7978	Train: 0.9831 Test: 0.7985	Train: 0.9831 Test: 0.7965

Kesimpulan Highlight Model Experiment (24 Januari 2025)

1. **EfficientNet-B0** adalah model terbaik secara keseluruhan dengan **80.34% Akurasi Pengujian** dan metrik stabil. Model ini memiliki keseimbangan yang baik antara performa pelatihan dan pengujian.
2. **ResNet50** adalah alternatif yang kuat dengan **80.23% Akurasi Pengujian**.
3. **EfficientNet-B3** menunjukkan performa pelatihan luar biasa tetapi memiliki sedikit tanda overfitting, dengan akurasi pengujian sedikit lebih rendah (**79.85%**).

Eksperimen ini dilakukan untuk membandingkan performa berbagai arsitektur model dalam klasifikasi, dengan mempertimbangkan Akurasi, Precision, Recall, dan F1 Score pada dataset pelatihan dan pengujian.

1. Model dengan Performa Terendah

- **YoloV5 + CSPDarknet53:**
 - Akurasi dan metrik lainnya menunjukkan performa sangat rendah (Akurasi: 28.8%).
 - Model ini kurang cocok untuk tugas klasifikasi dataset ini.

2. Model dengan Performa Sedang

- **Simple CNN:**
 - Performa pelatihan cukup stabil (73.38% Akurasi Train) namun pengujian meningkat (76.22% Akurasi Test).
 - Metrik konsisten namun masih kalah dengan model berbasis transfer learning seperti ResNet atau EfficientNet.
- **CNN + ResNet18:**
 - Performa meningkat signifikan dibanding Simple CNN.
 - Akurasi Pengujian mencapai 75.95%, dan metrik lainnya seperti Precision dan Recall stabil (~77.98%).
 - Dengan fine-tuning, akurasi pengujian meningkat menjadi 77.98%.

3. Model dengan Performa Terbaik

- **CNN + ResNet50:**
 - Model menunjukkan performa pengujian yang lebih baik dengan 80.23% Akurasi dan metrik lainnya seperti Precision (79.95%) dan Recall (80.23%).
 - Sangat cocok untuk dataset ini dibandingkan dengan ResNet18.
- **CNN + EfficientNet-B0:**
 - Model ini memberikan hasil terbaik di antara semua arsitektur dengan 80.34% Akurasi Pengujian, Precision (80.36%), dan Recall (80.34%).
 - Performa stabil baik pada pelatihan (95.43%) maupun pengujian.

- **CNN + EfficientNet-B3:**
 - Meskipun memiliki Akurasi Pelatihan Tertinggi (98.31%), akurasi pengujian sedikit lebih rendah (79.85%) dibandingkan EfficientNet-B0.
 - Model ini menunjukkan tanda-tanda overfitting karena performa pelatihan jauh lebih tinggi dibandingkan pengujian.

Streamlit APP

25 Januari 2025

Instalasi Proyek

Aplikasi ini membutuhkan **Python 3.11**. Ikuti langkah-langkah berikut untuk menjalankan proyek:

1. Clone Repository

Clone repository ke komputer lokal Anda:

```
git clone <repository-url>
cd <repository-folder>
```

2. Buat Virtual Environment

Buat dan aktifkan virtual environment untuk proyek:

```
python3.11 -m venv .venv
source .venv/bin/activate # Untuk macOS/Linux
.venv\Scripts\activate # Untuk Windows
```

3. Install Dependensi

Install semua dependensi yang dibutuhkan menggunakan `requirements.txt`:

```
pip install -r requirements.txt
```

4. Jalankan Aplikasi

Jalankan aplikasi **Streamlit** menggunakan perintah berikut:

```
streamlit run app.py
```

Aplikasi akan terbuka di browser pada alamat default `http://localhost:8501`.

Struktur File

Berikut adalah struktur file utama proyek:

```
project-folder/
    |
    └── .venv/          # Virtual environment
    └── models/         # Model terlatih
        └── Best Model EfficientNet B0.pth
    └── notebooks/      # Notebook Jupyter untuk eksperimen
    └── app.py          # File utama aplikasi Streamlit
    └── README.md       # Dokumentasi proyek
    └── requirements.txt # Daftar dependensi Python
    └── .gitignore       # File untuk mengabaikan file/direktori tertentu
```

Catatan Tambahan

1. **Model Path:** Pastikan file model terlatih berada di folder `models/` dengan nama `Best Model EfficientNet B0.pth`.
2. **Python Version:** Gunakan Python versi 3.11 untuk kompatibilitas penuh.
3. **Dependensi:** Jika ada kesalahan instalasi, pastikan versi pustaka yang digunakan sesuai dengan `requirements.txt`.

Implementasi Utama

Memuat Model

File model terletak di dalam folder `models/`, dan berikut adalah bagian kode yang memuat model:

```
# Load the model
@st.cache_resource
def load_model():
    model = efficientnet_b0(pretrained=False)

    # Modify the classifier to match the training configuration
    num_features = model.classifier[1].in_features
    model.classifier = nn.Sequential(
        nn.Linear(num_features, 256), # Intermediate layer
        nn.ReLU(),
        nn.Dropout(0.5), # Regularization
        nn.Linear(256, 4), # Output layer for 4 classes
    )

    state_dict = torch.load(
        "models/Best Model EfficientNet B0.pth", map_location=torch.device("cpu")
    )
    model.load_state_dict(state_dict)
    model.eval()
    return model
```

Prediksi Gambar

Fungsi untuk memproses gambar yang diunggah pengguna dan menghasilkan prediksi:

```
def predict_image(model, image):
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.5], [0.5])
    ])
    image_tensor = transform(image).unsqueeze(0)
    with torch.no_grad():
        output = model(image_tensor)
        _, predicted = torch.max(output, 1)
        probabilities = torch.nn.functional.softmax(output, dim=1)
    return predicted.item(), probabilities.squeeze().tolist()
```

Antarmuka Pengguna

Antarmuka pengguna berbasis Streamlit untuk memuat gambar, menjalankan prediksi, dan menampilkan hasil:

```
st.title("Deteksi Kerusakan Permukaan Kain")
uploaded_file = st.file_uploader("Pilih file gambar", type=["jpg", "png", "jpeg"])

if uploaded_file is not None:
    image = Image.open(uploaded_file).convert("RGB")
    st.image(image, caption="Gambar yang Diunggah", use_container_width=True)

    st.write("Mendeteksi kelas kerusakan...")
    predicted_class, probabilities = predict_image(model, image)
    st.write(f"Kelas yang Diprediksi: {defect_classes[predicted_class]}")

    st.write("Probabilitas untuk Setiap Kelas:")
    for i, prob in enumerate(probabilities):
        st.write(f"{defect_classes[i]}: {prob * 100:.2f}%")
```

Penjelasan Google Collab

24 Januari 2025

How to Run?

Run mulai dari atas sampai bawah

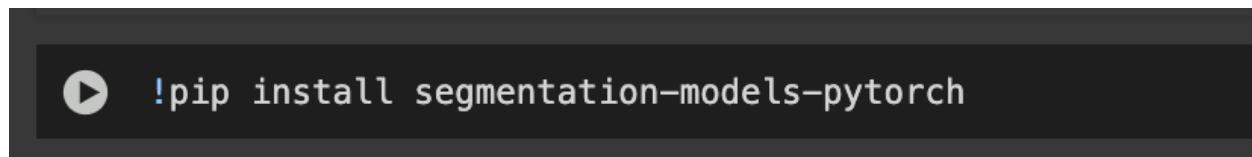
Link Google Collab

[co CNN-Defect Kain](#)

1. Deskripsi Project

Preparation

Import Library & Package



```
▶ !pip install segmentation-models-pytorch
```

```
[1]: from google.colab import drive
import os
import pandas as pd
import matplotlib.pyplot as plt
import zipfile
from collections import Counter

import albumentations as A
from albumentations.pytorch import ToTensorV2
import cv2
import numpy as np
import shutil

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, models, transforms
from torch.utils.data import DataLoader, Dataset
import torch.optim as optim
import time
import copy
from torch.cuda.amp import GradScaler, autocast
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import random
from PIL import Image
import segmentation_models_pytorch as smp
from torch.optim.lr_scheduler import ReduceLROnPlateau
from torchvision.models import efficientnet_b3
```

Load Dataset

```
# Path ke file ZIP dan folder tujuan
zip_path = '/content/drive/MyDrive/Freelance/16_01_2025/merge.zip' # Ganti dengan lokasi file ZIP Anda
extract_path = 'dataset/' # Folder tujuan setelah unzip

# Membuka dan mengekstrak file ZIP
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print(f"Dataset berhasil diekstrak ke: {extract_path}")

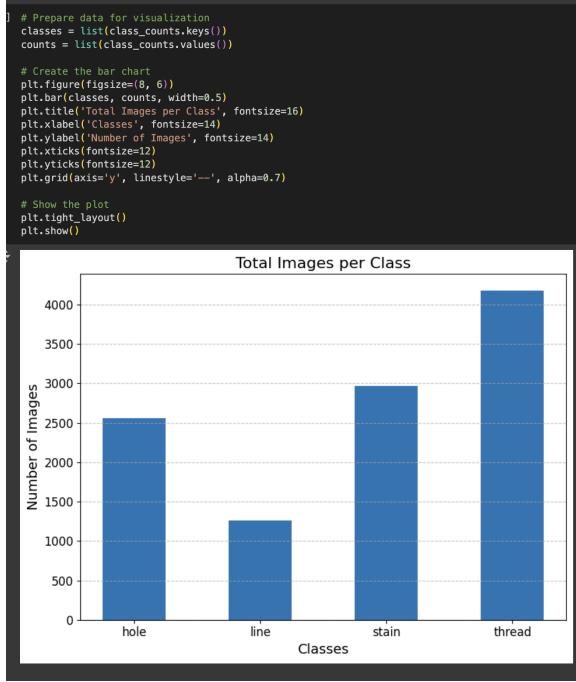
# Melihat isi folder setelah ekstraksi
extracted_files = os.listdir(extract_path)
print("Isi folder dataset:")
print(extracted_files)

Dataset berhasil diekstrak ke: dataset/
Isi folder dataset:
['merge']
```

2. Use YoLO

EDA

Gambar	Penjelasan
<pre>1 DATASET_PATH = "/content/dataset/merge" # Define class names class_names = ['hole', 'line', 'stain', 'thread'] # Path to the 'labels' folder labels_path = f'{DATASET_PATH}/labels'</pre>	<p>Berikut adalah penjelasan singkat tentang kode yang diberikan:</p> <ol style="list-style-type: none">1. DATASET_PATH = "/content/dataset/merge" Baris ini mendefinisikan lokasi dataset utama dalam folder <code>/content/dataset/merge</code>.2. class_names = ['hole', 'line', 'stain', 'thread'] Variabel <code>class_names</code> berisi daftar nama kelas yang ada di dataset, yaitu: <code>hole</code>, <code>line</code>, <code>stain</code>, dan <code>thread</code>. Nama-nama ini biasanya merepresentasikan kategori atau jenis cacat pada kain.3. labels_path = f'{DATASET_PATH}/labels' Baris ini menentukan path untuk folder <code>labels</code>, yaitu sub-folder dalam direktori dataset utama yang berisi file label. Folder ini akan berada di <code>/content/dataset/merge/labels</code>.
<pre># Initialize a dictionary to count images for each class class_counts = {class_name: 0 for class_name in class_names} # Iterate through label files for label_file in os.listdir(labels_path): if label_file.endswith('.txt'): # Ensure only .txt files are processed file_path = os.path.join(labels_path, label_file) with open(file_path, 'r') as file: lines = file.readlines() # Track unique classes present in the current image image_classes = set() for line in lines: class_id = int(line.split()[0]) # The first value in each line is the class ID image_classes.add(class_id) # Update the counts for each class present in this image for class_id in image_classes: class_counts[class_names[class_id]] += 1 # Print the total images per class for class_name, count in class_counts.items(): print(f"Class '{class_name}': {count} images") Class 'hole': 2557 images Class 'line': 1261 images Class 'stain': 2972 images Class 'thread': 4180 images</pre>	<p>Kode ini menghitung jumlah gambar yang mengandung setiap kelas dari file label di folder <code>labels_path</code>:</p> <ol style="list-style-type: none">1. class_counts: Kamus untuk menyimpan jumlah gambar per kelas, diinisialisasi dengan nilai 0 untuk setiap kelas.2. Iterasi file: Memproses setiap file label <code>.txt</code> di folder.3. Ekstrak kelas: Membaca file, mengidentifikasi kelas unik dalam setiap gambar.4. Perbarui hitungan: Menambahkan jumlah gambar yang mengandung kelas tersebut.5. Cetak hasil: Menampilkan jumlah gambar untuk setiap kelas.



Kode ini membuat visualisasi jumlah gambar per kelas dalam bentuk diagram batang:

1. Persiapan data:

- `classes` adalah daftar nama kelas.
- `counts` adalah jumlah gambar untuk setiap kelas (diambil dari `class_counts`).

2. Diagram batang:

- `plt.bar()` digunakan untuk membuat diagram batang dengan nama kelas sebagai sumbu X dan jumlah gambar sebagai sumbu Y.

3. Penyesuaian tampilan:

- Ukuran figur diatur dengan `figsize`.
- Judul, label sumbu, ukuran teks, dan grid horizontal ditambahkan untuk memperjelas visualisasi.

4. Tampilkan plot:

- `plt.tight_layout()` memastikan tata letak tidak tumpang tindih.
- `plt.show()` menampilkan plot di layar.

Data Preprocessing

Gambar	Penjelasan
<pre>1 # Path to your dataset images_path = os.path.join(DATASET_PATH, 'images') labels_path = os.path.join(DATASET_PATH, 'labels') 1 # Output paths for train, val, and test output_path = 'data_train' train_ratio = 0.7 val_ratio = 0.2 test_ratio = 0.1</pre>	<p>Kode ini menentukan jalur dataset dan rasio pembagian data:</p> <ol style="list-style-type: none">1. <code>images_path</code> dan <code>labels_path</code>: Jalur ke folder gambar dan label dalam dataset.2. <code>output_path</code>: Folder output untuk menyimpan data terpisah (train, val, test).3. <code>train_ratio</code>, <code>val_ratio</code>, dan <code>test_ratio</code>: Rasio pembagian data menjadi set pelatihan (70%), validasi (20%), dan pengujian (10%).
<pre>1 # Create train, val, and test folders for split in ['train', 'val', 'test']: os.makedirs(os.path.join(output_path, split, 'images'), exist_ok=True) os.makedirs(os.path.join(output_path, split, 'labels'), exist_ok=True) # Get all image filenames image_files = [f for f in os.listdir(images_path) if f.endswith('.jpg') or f.endswith('.png')] random.shuffle(image_files) # Calculate number of files for each subset train_count = int(len(image_files) * train_ratio) val_count = int(len(image_files) * val_ratio) # Split data train_files = image_files[:train_count] val_files = image_files[train_count:train_count + val_count] test_files = image_files[train_count + val_count:] # Function to move files def move_files(file_list, split): for file_name in file_list: # Move image shutil.copy2(os.path.join(images_path, file_name), os.path.join(output_path, split, 'images', file_name)) # Move corresponding label label_file = file_name.replace('.jpg', '.txt').replace('.png', '.txt') if os.path.exists(os.path.join(labels_path, label_file)): shutil.copy2(os.path.join(labels_path, label_file), os.path.join(output_path, split, 'labels', label_file)) # Move files to respective folders move_files(train_files, 'train') move_files(val_files, 'val') move_files(test_files, 'test') print("Dataset has been split into train, val, and test!") Dataset has been split into train, val, and test!</pre>	<p>Kode ini membagi dataset menjadi folder <code>train</code>, <code>val</code>, dan <code>test</code> dengan langkah berikut:</p> <ol style="list-style-type: none">1. Buat folder output: Folder untuk gambar dan label dibuat di dalam folder <code>train</code>, <code>val</code>, dan <code>test</code>.2. Dapatkan file gambar: Semua file gambar (<code>.jpg</code> atau <code>.png</code>) dari folder <code>images_path</code> dikumpulkan dan diacak.3. Hitung jumlah file per subset: Data dibagi berdasarkan rasio <code>train_ratio</code>, <code>val_ratio</code>, dan sisanya untuk <code>test</code>.4. Pisahkan data: File gambar dipisahkan ke subset pelatihan, validasi, dan pengujian.5. Pindahkan file: Fungsi <code>move_files</code> menyalin file gambar dan label yang sesuai ke folder subset masing-masing.6. Output akhir: Dataset berhasil dipisahkan ke dalam folder <code>train</code>, <code>val</code>, dan <code>test</code>.

```

train_labels_path = '/content/data_train/train/labels'
train_images_path = '/content/data_train/train/images'

# Class distribution (update based on your data)
class_counts = {'hole': 2557, 'line': 1261, 'stain': 2972, 'thread': 4180}
target_count = max(class_counts.values()) # Balance to the largest class

# Oversample minority classes
for class_name, count in class_counts.items():
    if count < target_count:
        # Find all label files for this class
        label_files = [f for f in os.listdir(train_labels_path) if class_name in f]
        additional_needed = target_count - count

        for _ in range(additional_needed):
            label_file = random.choice(label_files)
            image_file = label_file.replace('.txt', '.jpg')

            # Copy image and label
            shutil.copy2(os.path.join(train_labels_path, label_file),
                        os.path.join(train_labels_path, f'copy_{label_file}'))
            shutil.copy2(os.path.join(train_images_path, image_file),
                        os.path.join(train_images_path, f'copy_{image_file}'))

print("Oversampling completed!")
Oversampling completed!

```

Kode ini melakukan **oversampling** untuk menyeimbangkan jumlah data antar kelas di dataset pelatihan:

- Jalur dataset pelatihan:** Menentukan jalur ke folder **label** dan **gambar** di subset pelatihan.
- Distribusi kelas:** `class_counts` menyimpan jumlah gambar per kelas, dan `target_count` diatur ke jumlah tertinggi (kelas mayoritas).
- Oversampling kelas minoritas:** Untuk setiap kelas dengan jumlah data lebih kecil dari `target_count`:
 - Label file kelas tersebut ditemukan.
 - File dipilih secara acak untuk diduplikasi hingga jumlahnya sama dengan `target_count`.
- Salin file:** Label (.txt) dan gambar (.jpg) yang sesuai disalin dengan nama baru (`copy_` prefix).
- Hasil akhir:** Dataset pelatihan kini memiliki jumlah data yang seimbang antar kelas, dan proses oversampling selesai.

```

class_counts = [2557, 1261, 2972, 4180]
total_images = sum(class_counts)
class_weights = [total_images / count for count in class_counts]
normalized_weights = np.array(class_weights) / sum(class_weights)
print("Normalized Class Weights:", normalized_weights)

Normalized Class Weights: [0.22223004 0.45062825 0.19119859 0.13594311]

```

Kode ini menghitung bobot kelas yang dinormalisasi berdasarkan distribusi data:

- class_counts:** Daftar jumlah gambar per kelas.
- total_images:** Total jumlah gambar di semua kelas.
- class_weights:** Bobot awal untuk setiap kelas, dihitung dengan rumus $\text{total_images}/\text{count}$ / $\text{total_images}/\text{count}$. Kelas dengan data lebih sedikit mendapatkan bobot lebih besar.
- normalized_weights:** Bobot dinormalisasi sehingga jumlah total bobot sama dengan 1.
- Cetak hasil:** Menampilkan bobot kelas yang dinormalisasi.

YoloV5 + CSPDarknet54

Gambar	Penjelasan
--------	------------

```

!git clone https://github.com/ultralytics/yolov5.git
%cd yolov5

Cloning into 'yolov5'...
remote: Enumerating objects: 17265, done.
remote: Counting objects: 100% (105/105), done.
remote: Compressing objects: 100% (91/91), done.
remote: Total 17265 (delta 57), reused 14 (delta 14), pack-reused 17160 (from 4)
Receiving objects: 100% (17265/17265), 16.01 MiB | 16.78 MiB/s, done.
Resolving deltas: 100% (11799/11799), done.
/content/yolov5

!touch models/cspdarknet53_yolov5s.yaml

```

1. **!git clone**

https://github.com/ultralytics/yolov5.git:

Mengunduh repository YOLOv5 dari GitHub ke direktori lokal.

2. **%cd yolov5:**

Berpindah ke folder **yolov5** yang baru saja diunduh.

3. **!touch models/cspdarknet53_yolov5s.yaml:**

Membuat file kosong bernama **cspdarknet53_yolov5s.yaml** di dalam folder **models**. File ini biasanya digunakan untuk mendefinisikan arsitektur model kustom YOLOv5.

```

!python train.py \
    --img 416 \
    --batch 64 \
    --epochs 10 \
    --data "../data_train/data.yaml" \
    --cfg models/cspdarknet53_yolov5s.yaml \
    --weights yolov5s.pt \
    --hyp data/hyps/hyp.yaml \
    --device 0

```

Berikut penjelasan singkat kode:

1. **!python train.py:**

Memulai proses pelatihan menggunakan skrip **train.py** di YOLOv5.

2. **--img 416:**

Ukuran gambar input diatur menjadi 416x416 piksel.

3. **--batch 64:**

Batch size diatur menjadi 64 gambar per iterasi pelatihan.

4. **--epochs 10:**

Pelatihan akan berlangsung selama 10 epoch.

5. **--data "../data_train/data.yaml":**

Menggunakan file konfigurasi dataset **data.yaml** yang berisi informasi seperti jalur data, kelas, dan jumlah kelas.

6. **--cfg models/cspdarknet53_yolov5s.yaml:**

File konfigurasi model arsitektur kustom, dalam hal ini menggunakan **cspdarknet53_yolov5s.yaml**.

7. **--weights yolov5s.pt:**

Model pre-trained YOLOv5s digunakan sebagai bobot awal untuk pelatihan.

8. **--hyp data/hyps/hyp.yaml:**

Menggunakan file hyperparameter **hyp.yaml** untuk mengatur parameter pelatihan (misalnya, learning rate, augmentation).

9. **--device 0:**

Melatih model di GPU dengan ID 0.

Result:

cspdarknet53_YOLOv5s summary: 199 layers, 5805489 parameters, 0 gradients

Class	Images	Instances	P	R
mAP50	mAP50-95:	100%	15/15 [00:32<00:00,	2.18s/it]

		all	1819	3455	0.288	0.332	0.26
0.0937		hole	1819	611	0.253	0.167	
0.154	0.0529	line	1819	291	0.221	0.67	0.458
0.158		stain	1819	1098	0.304	0.163	
0.16	0.0583	thread	1819	1455	0.375	0.327	
0.27	0.106						

3. CNN Architecture

Data Preparation

Gambar	Penjelasan

```

DATASET_PATH = "/content/dataset/merge"
# Define class names
class_names = ['hole', 'line', 'stain', 'thread']

# Path to the 'labels' folder
labels_path = f'{DATASET_PATH}/labels'

# Initialize a dictionary to count images for each class
class_counts = {class_name: 0 for class_name in class_names}

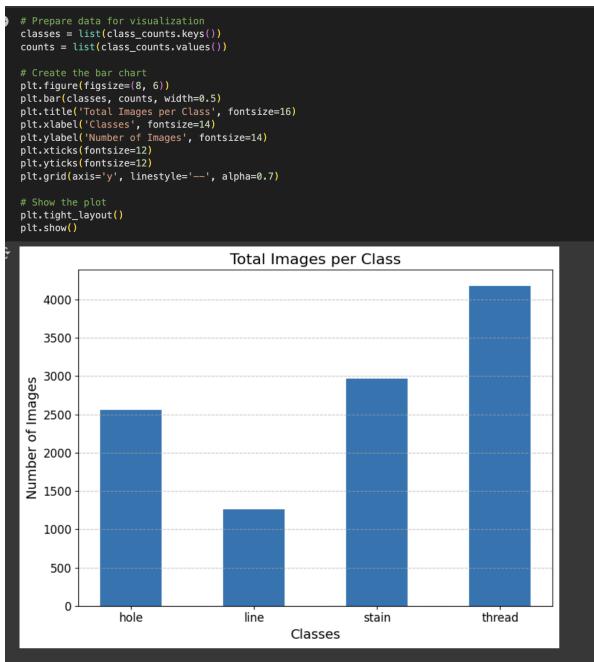
# Iterate through label files
for label_file in os.listdir(labels_path):
    if label_file.endswith('.txt'): # Ensure only .txt files are processed
        file_path = os.path.join(labels_path, label_file)
        with open(file_path, 'r') as file:
            lines = file.readlines()
            # Track unique classes present in the current image
            image_classes = set()
            for line in lines:
                class_id = int(line.split()[0]) # The first value in each line is the class ID
                image_classes.add(class_id)
            # Update the counts for each class present in this image
            for class_id in image_classes:
                class_counts[class_names[class_id]] += 1

# Print the total images per class
for class_name, count in class_counts.items():
    print(f"Class '{class_name}': {count} images")

```

Kode ini menghitung jumlah gambar yang berisi setiap kelas dari dataset berdasarkan file label:

1. **Dataset Path:** Lokasi dataset diatur ke `"/content/dataset/merge"`. Folder `labels` berisi file label dengan anotasi kelas untuk setiap gambar.
2. **Class Names:** Nama-nama kelas didefinisikan dalam `class_names` (hole, line, stain, thread).
3. **Class Counts:** Kamus `class_counts` diinisialisasi untuk melacak jumlah gambar per kelas, dimulai dari nol.
4. **Iterasi File Label:**
 - Hanya file `.txt` dalam folder `labels` yang diproses.
 - Setiap file dibaca, dan ID kelas (`class_id`, angka pertama di setiap baris) ditambahkan ke set `image_classes` untuk menghindari penghitungan ganda dalam satu gambar.
5. **Update Counts:** Untuk setiap `class_id` yang ditemukan dalam file, jumlah gambar untuk kelas terkait ditingkatkan.
6. **Cetak Hasil:** Jumlah total gambar per kelas ditampilkan:
 - Hole: 2557 gambar
 - Line: 1261 gambar
 - Stain: 2972 gambar
 - Thread: 4180 gambar



Kode ini membuat visualisasi distribusi jumlah gambar untuk setiap kelas menggunakan diagram batang:

1. Persiapan Data:

- **classes**: Daftar nama kelas diambil dari kunci `class_counts`.
- **counts**: Jumlah gambar untuk setiap kelas diambil dari nilai `class_counts`.

2. Pembuatan Diagram Batang:

- `plt.bar()`: Membuat diagram batang dengan kelas pada sumbu X dan jumlah gambar pada sumbu Y.
- Lebar batang diatur ke 0.5.

3. Penyesuaian Tampilan:

- Judul ('Total Images per Class') dan label sumbu (Classes, Number of Images) ditambahkan.
- Ukuran font untuk judul, label, dan angka sumbu disesuaikan.
- Grid horizontal ditambahkan untuk mempermudah pembacaan, menggunakan garis putus-putus.

4. Tampilkan Plot:

- `plt.tight_layout()`: Memastikan elemen plot tidak tumpang tindih.
- `plt.show()`: Menampilkan diagram batang.

```

] images_path = os.path.join(DATASET_PATH, "images")
labels_path = os.path.join(DATASET_PATH, "labels")
output_path = "dataset_cnn" # New dataset for CNN

# Create train and test folders
os.makedirs(os.path.join(output_path, "train"), exist_ok=True)
os.makedirs(os.path.join(output_path, "test"), exist_ok=True)

```

Kode ini mempersiapkan struktur folder untuk dataset yang akan digunakan oleh model CNN:

1. **images_path** dan **labels_path**:

Menentukan jalur ke folder **images** dan **labels** dari dataset utama.

2. **output_path**:

Menentukan jalur folder output baru (**dataset_cnn**) untuk menyimpan dataset yang akan digunakan oleh model CNN.

3. **Membuat Folder train dan test**:

- Di dalam folder **dataset_cnn**, dibuat subfolder **train** dan **test**.
- **exist_ok=True** memastikan tidak ada error jika folder sudah ada.

```

# Define train-test split ratio
train_ratio = 0.8

# Get all image files
image_files = [f for f in os.listdir(images_path) if f.endswith(".jpg")]

# Process images and labels
classwise_images = {} # To store images grouped by class
for img_file in image_files:
    label_file = img_file.replace(".jpg", ".txt")
    label_path = os.path.join(labels_path, label_file)

    # Read the label file only if it's not empty
    if os.path.exists(label_path) and os.path.getsize(label_path) > 0: # Check for file size
        with open(label_path, "r") as f:
            lines = f.readlines()
            # Take the first class ID (handle cases with multiple objects)
            if lines: # Proceed if lines is not empty
                class_id = lines[0].split()[0].strip()
            else:
                print("Warning: Label file '{label_file}' is empty. Skipping this image.")
                continue # Skip to the next image

        # Group by class
        if class_id not in classwise_images:
            classwise_images[class_id] = []
        classwise_images[class_id].append(img_file)

# Create class-wise folders and move files
for class_id, images in classwise_images.items():
    # Create class folder
    train_class_path = os.path.join(output_path, "train", f"class_{class_id}")
    test_class_path = os.path.join(output_path, "test", f"class_{class_id}")
    os.makedirs(train_class_path, exist_ok=True)
    os.makedirs(test_class_path, exist_ok=True)

    # Split into train and test
    split_index = int(len(images) * train_ratio)
    train_images = images[:split_index]
    test_images = images[split_index:]

    # Move images
    for img in train_images:
        shutil.copy2(os.path.join(images_path, img), os.path.join(train_class_path, img))
    for img in test_images:
        shutil.copy2(os.path.join(images_path, img), os.path.join(test_class_path, img))

print("Dataset conversion completed!")

Dataset conversion completed!

```

Kode ini membagi dataset menjadi data **pelatihan** dan **pengujian**, diorganisasi berdasarkan kelas, dengan langkah berikut:

1. Definisi Rasio Train-Test:

- **train_ratio = 0.8**: 80% gambar digunakan untuk pelatihan, sisanya untuk pengujian.

2. Ambil Semua File Gambar:

- Mengambil semua file gambar dengan ekstensi **.jpg** dari folder dataset.

3. Pengelompokan Berdasarkan Kelas:

- Membaca file label yang sesuai untuk setiap gambar.
- File label diproses untuk mendapatkan **ID kelas** dari objek pertama di dalamnya.
- Gambar dikelompokkan berdasarkan kelas ke dalam dictionary **classwise_images**.

4. Pembuatan Folder Berdasarkan Kelas:

- Membuat subfolder untuk setiap kelas di dalam folder **train** dan **test**.

5. Pemisahan Train-Test:

- Gambar untuk setiap kelas dibagi menjadi data pelatihan dan pengujian menggunakan rasio **train_ratio**.

6. Pindahkan Gambar:

- Gambar dari data pelatihan dan pengujian disalin ke folder masing-masing kelas di **train** dan **test**.

EDA

Gambar	Penjelasan
--------	------------

```

} # Path to the dataset
dataset_path = "dataset_cnn" # Replace with your dataset folder
train_path = os.path.join(dataset_path, "train")
test_path = os.path.join(dataset_path, "test")

# Helper function: Count images in each class
def count_images_per_class(base_path):
    class_counts = {}
    for class_folder in os.listdir(base_path):
        class_folder_path = os.path.join(base_path, class_folder)
        if os.path.isdir(class_folder_path):
            num_images = len([f for f in os.listdir(class_folder_path) if f.endswith('.jpg', '.png')])
            class_counts[class_folder] = num_images
    return class_counts

```

Kode ini bertujuan untuk menghitung jumlah gambar di setiap kelas dalam dataset yang telah terstruktur:

1. Definisi Jalur Dataset:

- `dataset_path`: Jalur ke folder dataset yang berisi subfolder `train` dan `test`.
- `train_path` dan `test_path`: Jalur ke folder pelatihan dan pengujian.

2. Fungsi `count_images_per_class`:

- Input: `base_path` (jalur folder dataset, misalnya `train_path` atau `test_path`).
- Proses:
 - Iterasi melalui setiap subfolder di dalam `base_path` (subfolder diharapkan merupakan kelas).
 - Menghitung jumlah file gambar (`.jpg` atau `.png`) di setiap subfolder kelas.
 - Menyimpan hasilnya dalam dictionary `class_counts`, dengan nama kelas sebagai kunci dan jumlah gambar sebagai nilai.
- Output: Dictionary berisi jumlah gambar per kelas.

```

# 1. Analyze Class Distribution
train_counts = count_images_per_class(train_path)
test_counts = count_images_per_class(test_path)

print("Train Class Distribution:", train_counts)
print("Test Class Distribution:", test_counts)

# Plot Class Distribution
def plot_class_distribution(counts, title):
    classes = list(counts.keys())
    values = list(counts.values())
    plt.figure(figsize=(10, 5))
    plt.bar(classes, values)
    plt.title(title)
    plt.xlabel("Classes")
    plt.ylabel("Number of Images")
    plt.show()

plot_class_distribution(train_counts, "Train Class Distribution")
plot_class_distribution(test_counts, "Test Class Distribution")

```

Kode ini menganalisis distribusi kelas dalam dataset pelatihan dan pengujian, serta membuat plot untuk visualisasi:

1. Hitung Distribusi Kelas:

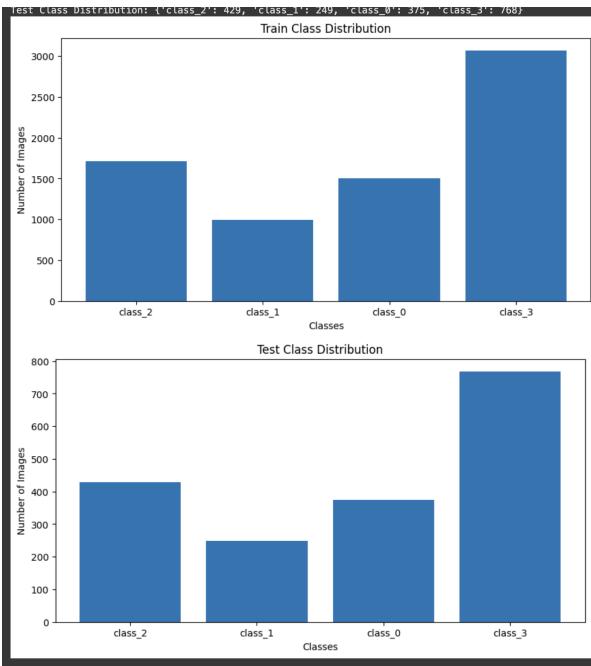
- `train_counts`: Distribusi jumlah gambar per kelas dalam folder pelatihan dihitung dengan `count_images_per_class(train_path)`.
- `test_counts`: Distribusi jumlah gambar per kelas dalam folder pengujian dihitung dengan `count_images_per_class(test_path)`.

2. Cetak Distribusi:

- Distribusi kelas untuk pelatihan dan pengujian dicetak untuk dianalisis.

3. Fungsi `plot_class_distribution`:

- Input:
 - `counts`: Dictionary distribusi kelas (misalnya `train_counts` atau `test_counts`).
 - `title`: Judul grafik (misalnya "Train



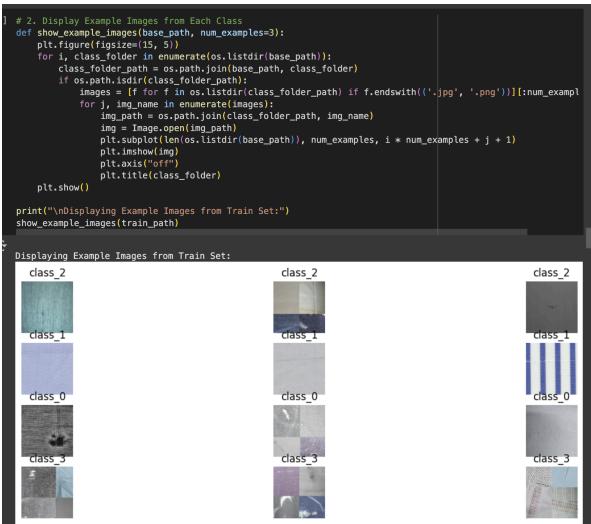
Class Distribution").

- Proses:

- Membuat diagram batang (`plt.bar`) berdasarkan kelas (sumbu X) dan jumlah gambar (sumbu Y).
- Menambahkan judul, label sumbu, dan menampilkan grafik.

4. Visualisasi Distribusi:

- Grafik distribusi kelas dibuat untuk data pelatihan dan pengujian menggunakan fungsi `plot_class_distribution`.



Kode ini menampilkan contoh gambar dari setiap kelas dalam dataset pelatihan dengan langkah berikut:

1. Fungsi `show_example_images`:

- Input:

- `base_path`: Jalur ke folder dataset (misalnya, `train_path`).
- `num_examples`: Jumlah gambar contoh yang akan ditampilkan untuk setiap kelas (default: 3).

- Proses:

- Iterasi melalui setiap folder kelas dalam `base_path`.
- Untuk setiap kelas:
 - Mengambil hingga `num_examples` gambar pertama (ekstensi `.jpg` atau `.png`).
 - Membuka gambar menggunakan `Image.open`.
 - Menampilkan gambar dalam sub-plot dengan judul nama kelas.

- Output: Gambar-gambar dari setiap kelas ditampilkan dalam satu plot.

2. Menampilkan Contoh Gambar dari Train Set:

- Memanggil

`show_example_images(train_path)` untuk menampilkan contoh gambar dari folder pelatihan.

3. Visualisasi:

- Setiap baris dalam plot mewakili satu kelas, dengan kolom menunjukkan contoh gambar dari kelas tersebut.
- Gambar tidak memiliki sumbu dan diberi judul nama kelas.

```
# 3. Analyze Image Properties
def analyze_image_properties(base_path):
    dimensions = []
    for class_folder in os.listdir(base_path):
        class_folder_path = os.path.join(base_path, class_folder)
        if os.path.isdir(class_folder_path):
            images = [f for f in os.listdir(class_folder_path) if f.endswith('.jpg', '.png')]
            for img_name in images:
                img_path = os.path.join(class_folder_path, img_name)
                img = Image.open(img_path)
                dimensions.append(img.size) # (width, height)

    # Count unique dimensions
    dim_counter = Counter(dimensions)
    print("\nUnique Image Dimensions (Width x Height):")
    for dim, count in dim_counter.items():
        print(f"({dim}): {count} Images")

    # Plot aspect ratio distribution
    aspect_ratios = [w / h for w, h in dimensions]
    plt.figure(figsize=(10, 5))
    plt.hist(aspect_ratios, bins=20, edgecolor="black")
    plt.title("Aspect Ratio Distribution")
    plt.xlabel("Aspect Ratio (Width / Height)")
    plt.ylabel("Frequency")
    plt.show()

print("\nAnalyzing Image Properties from Train Set:")
analyze_image_properties(train_path)

Analyzing Image Properties from Train Set:
Unique Image Dimensions (Width x Height):
(640, 640): 7276 Images

```

Kode ini menganalisis properti gambar, seperti dimensi dan rasio aspek, dalam dataset pelatihan dengan langkah berikut:

1. Fungsi `analyze_image_properties`:

- **Input:**
 - `base_path`: Jalur ke folder dataset (misalnya, `train_path`).
- **Proses:**
 - **Dimensi gambar:**
 - Iterasi melalui setiap folder kelas.
 - Membaca semua gambar dalam folder kelas menggunakan `Image.open` dan mencatat dimensi `(width, height)` untuk setiap gambar.
 - **Hitung dimensi unik:**
 - Menggunakan `Counter` untuk menghitung jumlah gambar dengan setiap kombinasi dimensi unik.
 - Menampilkan dimensi gambar unik beserta jumlahnya.
 - **Distribusi rasio aspek:**
 - Menghitung rasio aspek untuk setiap gambar sebagai `width / height`.
 - Membuat histogram untuk distribusi rasio aspek dengan 20 bin.

2. Cetak Dimensi Unik:

- Dimensi unik dari gambar dan jumlahnya ditampilkan di konsol.

3. Visualisasi Rasio Aspek:

- Histogram menunjukkan distribusi rasio

- aspek gambar di dataset pelatihan.
- 4. Pemanggilan Fungsi:**
- `analyze_image_properties(train_path)` digunakan untuk menganalisis properti gambar dalam dataset pelatihan.

Data Preprocessing

Handle Imbalancing

```
[1] # Paths
train_path = "nn/train" Add text cell

# Define augmentations
augment = transforms.Compose([
    transforms.RandomRotation(15),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1)
])

# Oversample and augment minority classes
target_count = max([len(os.listdir(os.path.join(train_path, cls))) for cls in os.listdir(train_path)])

for cls in os.listdir(train_path):
    class_path = os.path.join(train_path, cls)
    images = os.listdir(class_path)
    while len(images) < target_count:
        img_name = random.choice(images)
        img_path = os.path.join(class_path, img_name)
        img = Image.open(img_path)
        img_aug = augment(img)
        img_aug.save(os.path.join(class_path, f"aug_{len(images)}.jpg"))
        images.append(f"aug_{len(images)}.jpg")

print("Oversampling and augmentation completed!")

```

Oversampling and augmentation completed!

Kode ini melakukan **oversampling** dan **augmentasi data** pada kelas minoritas dalam dataset pelatihan:

1. **Jalur Dataset Pelatihan:**
 - `train_path` menunjukkan folder dataset pelatihan yang berisi subfolder berdasarkan kelas.
2. **Definisi Augmentasi:**
 - Menggunakan `torchvision.transforms.Compose` untuk membuat transformasi gambar, termasuk:
 - **RandomRotation(15)**: Memutar gambar secara acak hingga 15 derajat.
 - **RandomHorizontalFlip()**: Membalik gambar secara horizontal secara acak.
 - **ColorJitter**: Mengubah kecerahan, kontras, saturasi, dan rona gambar secara

acak.

3. Oversampling Kelas Minoritas:

- `target_count`: Menentukan jumlah gambar di kelas dengan data terbanyak (kelas mayoritas).
- Untuk setiap kelas, gambar di kelas tersebut diduplikasi dan diberi augmentasi hingga mencapai `target_count`.

4. Proses Augmentasi:

- Memilih gambar acak dari kelas.
- Membuka gambar menggunakan `PIL.Image.open`.
- Menerapkan transformasi augmentasi menggunakan `augment`.
- Menyimpan gambar yang telah diubah ke folder kelas dengan nama baru (`aug_<index>.jpg`).

5. Output:

- Dataset pelatihan kini memiliki jumlah gambar yang seimbang antar kelas dengan tambahan gambar hasil augmentasi.

Resizing & Normalize Images

```
# Define preprocessing transformations
transform = transforms.Compose([
    transforms.Resize((224, 224)), # Resize to 224x224
    transforms.ToTensor(), # Convert image to tensor
    transforms.Normalize([0.5], [0.5]) # Normalize to range [-1, 1]
])

# Apply transformations to train and test sets
train_dataset = datasets.ImageFolder("dataset_cnn/train", transform=transform)
test_dataset = datasets.ImageFolder("dataset_cnn/test", transform=transform)

# Create DataLoaders
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32, shuffle=False)

print("Data preprocessing completed!")

Data preprocessing completed!
```

Kode ini mempersiapkan dataset untuk pelatihan model dengan langkah berikut:

1. Transformasi Preprocessing:

- `Resize((224, 224))`: Mengubah ukuran gambar menjadi 224x224 piksel (umum untuk arsitektur CNN seperti ResNet).
- `ToTensor()`: Mengubah gambar menjadi tensor PyTorch.
- `Normalize([0.5], [0.5])`: Menormalkan nilai piksel gambar ke rentang [-1,1][-1, 1][-1,1] dengan rata-rata 0,5 dan standar deviasi 0,5.

2. Dataset ImageFolder:

- **train_dataset**: Membaca dataset pelatihan dari folder `dataset_cnn/train` dan menerapkan transformasi preprocessing.
- **test_dataset**: Membaca dataset pengujian dari folder `dataset_cnn/test` dengan transformasi yang sama.

3. DataLoader:

- **train_loader**: Membuat DataLoader untuk dataset pelatihan dengan:
 - `batch_size=32`: Memproses 32 gambar per batch.
 - `shuffle=True`: Mengacak data di setiap epoch untuk meningkatkan pelatihan.
- **test_loader**: Membuat DataLoader untuk dataset pengujian tanpa pengacakan (`shuffle=False`).

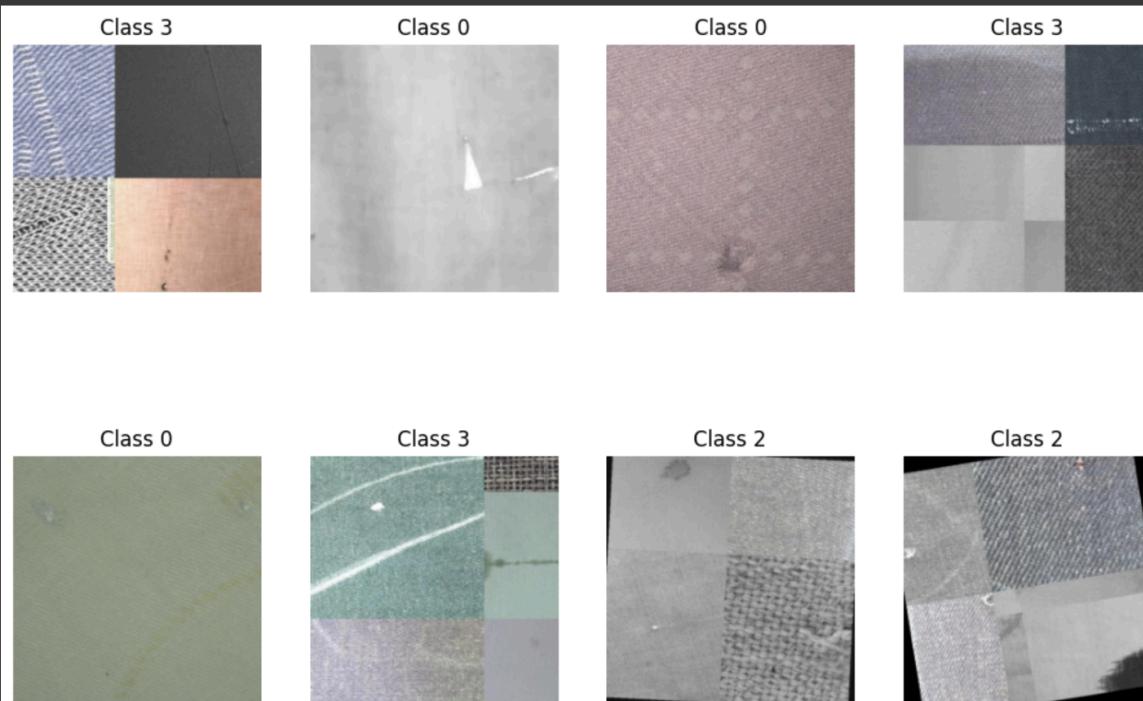
4. Output:

- Dataset telah diproses, diubah menjadi tensor, dan disiapkan untuk digunakan dalam pelatihan dan pengujian model.

```
# Visualize some preprocessed images
data_iter = iter(train_loader)
images, labels = next(data_iter)

# Denormalize and display images
def denormalize(tensor):
    return tensor * 0.5 + 0.5 # Reverse normalization to range [0, 1]

plt.figure(figsize=(12, 8))
for i in range(8):
    plt.subplot(2, 4, i+1)
    img = denormalize(images[i]).permute(1, 2, 0) # Convert to HWC format
    plt.imshow(img.numpy())
    plt.title(f"Class {labels[i].item()}")
    plt.axis("off")
plt.show()
```



Kode ini menampilkan beberapa gambar yang telah diproses dari dataset pelatihan dengan langkah berikut:

1. **Ambil Batch dari DataLoader:**

- `data_iter = iter(train_loader)`: Membuat iterator untuk `train_loader`.
- `images, labels = next(data_iter)`: Mengambil satu batch gambar dan label dari DataLoader.

2. **Fungsi Denormalisasi:**

- `denormalize(tensor)`: Mengembalikan tensor gambar dari rentang $[-1,1][-1, 1][-1,1]$ ke $[0,1][0, 1][0,1]$, membalik transformasi normalisasi.

3. **Visualisasi Gambar:**

- `plt.figure(figsize=(12, 8))`: Membuat plot dengan ukuran 12x8 inci.
- Iterasi pada 8 gambar pertama di batch:
 - Denormalisasi gambar menggunakan `denormalize(images[i])`.
 - Ubah format tensor dari **CHW** (**C**hannels-**H**eight-**W**idth) menjadi **HWC** (**H**eight-**W**idth-**C**hannels) menggunakan `permute(1, 2, 0)`.
 - Tampilkan gambar menggunakan `plt.imshow()`.
 - Tambahkan judul berupa kelas gambar (`labels[i].item()`) dan hapus sumbu dengan `plt.axis("off")`.

4. **Tampilkan Gambar:**

- `plt.show()`: Menampilkan grid dengan 8 gambar dari batch yang diambil.

```

# Class Distribution
train_counts = count_images_per_class(train_path)

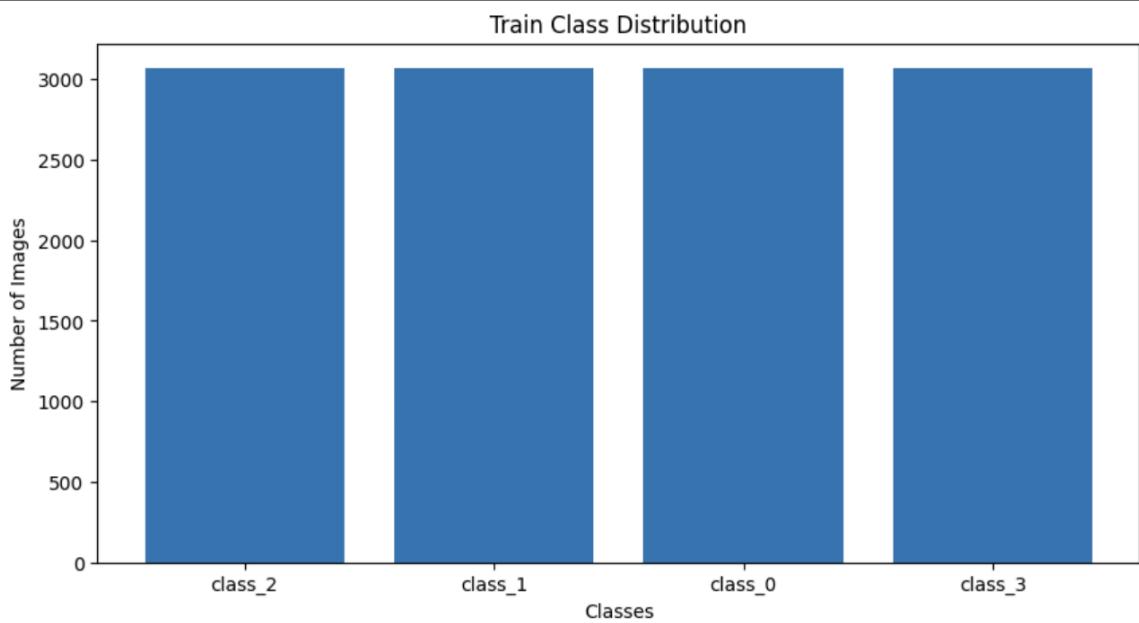
print("Train Class Distribution:", train_counts)

# Plot Class Distribution
def plot_class_distribution(counts, title):
    classes = list(counts.keys())
    values = list(counts.values())
    plt.figure(figsize=(10, 5))
    plt.bar(classes, values)
    plt.title(title)
    plt.xlabel("Classes")
    plt.ylabel("Number of Images")
    plt.show()

plot_class_distribution(train_counts, "Train Class Distribution")

```

Train Class Distribution: {'class_2': 3068, 'class_1': 3068, 'class_0': 3068, 'class_3': 3068}



Kode ini menganalisis dan memvisualisasikan distribusi jumlah gambar di setiap kelas pada dataset pelatihan:

1. Hitung Distribusi Kelas:

- `train_counts = count_images_per_class(train_path)`: Menggunakan fungsi `count_images_per_class` untuk menghitung jumlah gambar dalam setiap kelas di folder pelatihan.

2. Cetak Distribusi Kelas:

- `print("Train Class Distribution:", train_counts)`: Menampilkan distribusi kelas dalam format dictionary, di mana kunci adalah nama kelas, dan nilai adalah jumlah gambar.

3. Fungsi `plot_class_distribution`:

- Input:

- **counts**: Dictionary distribusi kelas.
 - **title**: Judul untuk grafik.
- Proses:
 - Membuat diagram batang menggunakan kelas sebagai sumbu X dan jumlah gambar sebagai sumbu Y.
 - Menambahkan judul dan label untuk sumbu.
4. **Plot Distribusi Kelas**:
 - Memanggil `plot_class_distribution` dengan data distribusi kelas (`train_counts`) dan judul "**Train Class Distribution**".

Modeling

Simple CNN

Gambar	Penjelasan
--------	------------

```

class DefectClassifierCNN(nn.Module):
    def __init__(self, num_classes=4):
        super(DefectClassifierCNN, self).__init__()
        # Convolutional layers
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1) # Input: (3, 224, 224), 0
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1) # Output: (64, 112, 112)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1) # Output: (128, 56, 56)

        # Pooling layer
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2) # Downsamples feature maps by half

        # Fully connected layers (input size will be calculated dynamically)
        self.fc1 = None # Placeholder, initialized later
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, num_classes)

        # Dropout
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))

        # Dynamically initialize the first FC layer
        if self.fc1 is None:
            self.fc1 = nn.Linear(x.view(x.size(0), -1).size(1), 256).to(x.device)

        x = torch.flatten(x, 1) # Flatten feature maps into a 1D vector
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x) # Output logits for each class

        return x

```

Berikut penjelasan kode arsitektur CNN untuk klasifikasi cacat pada gambar:

1. Struktur Model DefectClassifierCNN

- **Input:** Gambar dengan 3 channel (RGB) dan ukuran 224×224 $\times 3$.
- **Output:** Logit dengan jumlah kelas yang ditentukan oleh `num_classes` (default: 4).

2. Layer-Layer

a. Convolutional Layers:

- **conv1:**
 - Mengubah gambar dari 3 channel menjadi 32 feature maps.
 - Output: $32 \times 224 \times 224$ $\times 32$.
- **conv2:**
 - Mengubah dari 32 menjadi 64 feature maps.
 - Output: $64 \times 112 \times 112$ $\times 64$.
- **conv3:**
 - Mengubah dari 64 menjadi 128 feature maps.
 - Output: $128 \times 56 \times 56$ $\times 128$.

b. Pooling Layer:

- **self.pool:**
 - Max pooling dengan kernel 2×2 $\times 2$, mengurangi ukuran spatial feature maps menjadi setengah (downsampling).

c. Fully Connected Layers:

- **fc1:**
 - Diinisialisasi secara dinamis berdasarkan ukuran output dari convolutional layers.
- **fc2:**
 - Mengurangi dimensi dari 256 ke 128.
- **fc3:**
 - Menghasilkan logits dengan jumlah output sama dengan `num_classes`.

d. Dropout:

- Dropout dengan probabilitas 0.5 untuk mencegah overfitting selama pelatihan.

3. Forward Pass

	<ol style="list-style-type: none"> 1. Input gambar melewati conv1, conv2, dan conv3, diikuti dengan aktivasi ReLU dan pooling. 2. Feature maps diratakan menggunakan torch.flatten. 3. Jika fc1 belum diinisialisasi, ukuran inputnya dihitung secara dinamis dari tensor. 4. Data melewati fully connected layers (fc1, fc2, fc3) dengan aktivasi ReLU dan dropout di antara layer. 5. Output: Logits dengan ukuran <code>num_classesnum__classesnum_classes</code>.
<pre> # Hyperparameters batch_size = 32 learning_rate = 0.001 epochs = 20 num_classes = 4 device = torch.device("cuda" if torch.cuda.is_available() else "cpu") # Data preprocessing transform = transforms.Compose([transforms.Resize((224, 224)), transforms.ToTensor(), transforms.Normalize([0.5], [0.5]) # Normalize to [-1, 1]]) # Load datasets train_dataset = datasets.ImageFolder("dataset_cnn/train", transform=transform) test_dataset = datasets.ImageFolder("dataset_cnn/test", transform=transform) train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True) test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False) # Initialize model model = DefectClassifierCNN(num_classes=num_classes).to(device) </pre>	<p>1. Hyperparameter:</p> <ul style="list-style-type: none"> • batch_size: Ukuran batch untuk DataLoader (32 gambar per iterasi). • learning_rate: Kecepatan pembelajaran model (0.001). • epochs: Jumlah iterasi penuh melintasi dataset (20 epoch). • num_classes: Jumlah kelas (4). • device: Perangkat pelatihan, menggunakan GPU jika tersedia, atau CPU. <p>2. Preprocessing Data:</p> <ul style="list-style-type: none"> • Transformasi Gambar: <ul style="list-style-type: none"> ◦ Resize((224, 224)): Mengubah ukuran gambar menjadi 224x224 piksel. ◦ ToTensor(): Mengonversi gambar ke tensor PyTorch. ◦ Normalize([0.5], [0.5]): Menormalkan piksel gambar ke rentang [-1, 1]. <p>3. Dataset dan DataLoader:</p> <ul style="list-style-type: none"> • Dataset: <ul style="list-style-type: none"> ◦ train_dataset: Membaca data pelatihan dari folder <code>dataset_cnn/train</code> dan menerapkan transformasi. ◦ test_dataset: Membaca data pengujian dari folder <code>dataset_cnn/test</code> dengan transformasi yang sama. • DataLoader: <ul style="list-style-type: none"> ◦ train_loader: DataLoader untuk pelatihan dengan <code>shuffle=True</code> untuk mengacak data. ◦ test_loader: DataLoader untuk pengujian tanpa pengacakan (<code>shuffle=False</code>). <p>4. Inisialisasi Model:</p> <ul style="list-style-type: none"> • model =

	<p>DefectClassifierCNN(num_classes=num_classes):</p> <ul style="list-style-type: none"> ○ Membuat instance dari model CNN yang telah didefinisikan sebelumnya, dengan jumlah kelas sesuai dataset. ○ .to(device): Memindahkan model ke perangkat (GPU atau CPU) untuk pelatihan.
<pre># Loss function and optimizer criterion = nn.CrossEntropyLoss() optimizer = optim.Adam(model.parameters(), lr=learning_rate) for epoch in range(epochs): model.train() train_loss = 0 all_labels = [] all_predictions = [] for images, labels in train_loader: images, labels = images.to(device), labels.to(device) # Forward pass outputs = model(images) loss = criterion(outputs, labels) # Backward pass optimizer.zero_grad() loss.backward() optimizer.step() # Track loss train_loss += loss.item() # Track predictions and labels _, predicted = torch.max(outputs, 1) all_predictions.extend(predicted.cpu().numpy()) all_labels.extend(labels.cpu().numpy()) # Calculate metrics train_accuracy = (np.array(all_predictions) == np.array(all_labels)).mean() * 100 train_precision = precision_score(all_labels, all_predictions, average="weighted") train_recall = recall_score(all_labels, all_predictions, average="weighted") train_f1 = f1_score(all_labels, all_predictions, average="weighted") print(f"Epoch [{epoch+1}/{epochs}]") print(f"Loss: {train_loss/len(train_loader):.4f} Accuracy: {(train_accuracy:.2f)%}") print(f"Precision: {(train_precision:.4f)} Recall: {(train_recall:.4f)} F1 Score: {(train_f1:.4f)}") print(f"Precision: 0.4379 Recall: 0.4564 F1 Score: 0.4258") Precision: 0.4379 Recall: 0.4564 F1 Score: 0.4258</pre>	<p>Kode ini melatih model CNN menggunakan CrossEntropyLoss dan Adam optimizer, serta menghitung metrik evaluasi untuk setiap epoch. Berikut langkah-langkahnya:</p> <h3>1. Inisialisasi Komponen</h3> <ul style="list-style-type: none"> ● Loss Function: <ul style="list-style-type: none"> ○ criterion = nn.CrossEntropyLoss(): Menghitung loss untuk klasifikasi multi-kelas. ● Optimizer: <ul style="list-style-type: none"> ○ optimizer = optim.Adam(model.parameters(), lr=learning_rate): Optimizer Adam dengan learning rate yang ditentukan. <h3>2. Training Loop</h3> <p>Dilakukan untuk setiap epoch dalam jumlah yang telah ditentukan (epochs):</p> <h4>a. Persiapan Model dan Variabel:</h4> <ul style="list-style-type: none"> ● model.train(): Mengatur model ke mode pelatihan. ● train_loss, all_labels, dan all_predictions: Diinisialisasi untuk melacak loss dan prediksi selama epoch. <h4>b. Iterasi Batch pada DataLoader:</h4> <p>Untuk setiap batch dalam train_loader:</p> <ol style="list-style-type: none"> 1. Input dan Label ke Perangkat: <ul style="list-style-type: none"> ○ images.to(device): Memindahkan gambar ke GPU/CPU. ○ labels.to(device): Memindahkan label ke GPU/CPU. 2. Forward Pass: <ul style="list-style-type: none"> ○ outputs = model(images): Model memproses gambar dan menghasilkan prediksi. 3. Hitung Loss: <ul style="list-style-type: none"> ○ loss = criterion(outputs, labels): Menghitung loss antara prediksi dan label. 4. Backward Pass dan Optimasi: <ul style="list-style-type: none"> ○ optimizer.zero_grad(): Membersihkan gradien sebelumnya.

- `loss.backward()`: Menghitung gradien loss terhadap parameter model.
- `optimizer.step()`: Memperbarui parameter model berdasarkan gradien.

5. Lacak Hasil:

- Loss per batch ditambahkan ke `train_loss`.
- Prediksi batch ditentukan menggunakan `torch.max(outputs, 1)` dan disimpan bersama label.

3. Evaluasi Metrik

Setelah semua batch selesai:

- **Akurasi:** Menghitung persentase prediksi yang benar.
- **Precision, Recall, F1 Score:**
 - Menggunakan fungsi dari scikit-learn (`precision_score`, `recall_score`, `f1_score`) dengan rata-rata tertimbang (`average="weighted"`).

4. Logging

Hasil epoch dicetak:

- Loss rata-rata per batch.
- Akurasi, Precision, Recall, dan F1 Score untuk seluruh epoch.

```
Epoch [20/20]
Loss: 0.6684 | Accuracy: 73.38%
Precision: 0.7386 | Recall: 0.7338 | F1 Score: 0.7291
```

Pre-Trained Model ResNet18

Gambar	Penjelasan
--------	------------

```

] # Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Data preprocessing
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5]) # Normalize to [-1, 1]
])

# Load datasets
train_dataset = datasets.ImageFolder("dataset_cnn/train", transform=transform)
test_dataset = datasets.ImageFolder("dataset_cnn/test", transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Load pretrained ResNet18
model = models.resnet18(pretrained=True)

# Modify the last fully connected layer
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 4) # 4 classes
model = model.to(device)

```

Kode ini mempersiapkan data untuk digunakan dalam pelatihan model dengan langkah-langkah berikut:

1. Perangkat untuk Komputasi

- **device = torch.device("cuda" if torch.cuda.is_available() else "cpu"):**
 - Memilih perangkat untuk pelatihan.
 - Jika GPU tersedia, model akan dilatih menggunakan GPU, jika tidak maka menggunakan CPU.

2. Preprocessing Data

- **Transformasi Gambar:**
 - **Resize((224, 224)):** Mengubah ukuran gambar menjadi 224x224 piksel.
 - **ToTensor()**: Mengubah gambar menjadi tensor.
 - **Normalize([0.5], [0.5]):** Menormalkan nilai piksel gambar ke rentang [-1,1][-1, 1][-1,1], yang membantu mempercepat pelatihan dan stabilitas model.

3. Muat Dataset

- **datasets.ImageFolder:**
 - Membaca dataset gambar dari folder yang diorganisasi berdasarkan subfolder kelas:
 - **train_dataset:** Dataset pelatihan diambil dari folder `dataset_cnn/train` dengan transformasi yang telah didefinisikan.
 - **test_dataset:** Dataset pengujian diambil dari folder `dataset_cnn/test` dengan transformasi yang sama.

4. DataLoader

- **train_loader:**
 - DataLoader untuk dataset pelatihan:
 - **batch_size=32:** Memproses 32 gambar per batch.
 - **shuffle=True:** Mengacak urutan gambar untuk meningkatkan generalisasi model.
- **test_loader:**
 - DataLoader untuk dataset pengujian:

- **shuffle=False**: Tidak mengacak urutan gambar untuk menjaga konsistensi evaluasi.

```
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 20

for epoch in range(epochs):
    model.train()
    train_loss = 0
    all_labels = []
    all_predictions = []

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Track loss
        train_loss += loss.item()

        # Track predictions and labels
        _, predicted = torch.max(outputs, 1)
        all_predictions.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

    # Calculate metrics
    train_accuracy = np.array(all_predictions) == np.array(all_labels).mean() * 100
    train_precision = precision_score(all_labels, all_predictions, average="weighted")
    train_recall = recall_score(all_labels, all_predictions, average="weighted")
    train_f1 = f1_score(all_labels, all_predictions, average="weighted")

    print(f"Epoch [{epoch+1}/{epochs}]")
    print(f"Loss: {train_loss/len(train_loader):.4f} | Accuracy: {train_accuracy:.2f}%")
    print(f"Precision: {train_precision:.4f} | Recall: {train_recall:.4f} | F1 Score: {train_f1:.4f}")

    if (epoch+1) % 5 == 0:
        print("Training loop completed for epoch", epoch+1)
```

Kode ini melatih model CNN menggunakan **CrossEntropyLoss** dan **Adam optimizer**, dengan logging metrik evaluasi (Accuracy, Precision, Recall, dan F1 Score) di setiap epoch. Berikut penjelasannya:

1. Komponen Utama

- **Loss Function:**
 - **criterion = nn.CrossEntropyLoss()**: Digunakan untuk tugas klasifikasi multi-kelas.
- **Optimizer:**
 - **optimizer = optim.Adam(model.parameters(), lr=0.001)**: Mengoptimalkan parameter model dengan algoritma Adam dan learning rate 0.001.
 - **epochs = 20**: Model dilatih selama 20 epoch.

2. Training Loop

Dilakukan untuk setiap epoch:

a. Persiapan:

- **model.train()**: Mengatur model ke mode pelatihan.
- **Variabel untuk Tracking**:
 - **train_loss**: Menyimpan total loss untuk setiap epoch.
 - **all_labels** dan **all_predictions**: Menyimpan label asli dan prediksi untuk menghitung metrik evaluasi.

b. Iterasi Batch:

Untuk setiap batch dalam **train_loader**:

1. **Memindahkan Data ke Perangkat**:
 - Gambar dan label dipindahkan ke GPU/CPU menggunakan **.to(device)**.

2. **Forward Pass:**
 - **outputs = model(images):** Model memproses gambar dan menghasilkan output (logits).
3. **Hitung Loss:**
 - **loss = criterion(outputs, labels):** Menghitung selisih antara prediksi dan label.
4. **Backward Pass dan Optimasi:**
 - **optimizer.zero_grad():** Membersihkan gradien sebelumnya.
 - **loss.backward():** Menghitung gradien loss terhadap parameter model.
 - **optimizer.step():** Memperbarui parameter model menggunakan gradien.
5. **Tracking Loss dan Prediksi:**
 - Loss batch ditambahkan ke **train_loss**.
 - Prediksi dihitung dengan **torch.max(outputs, 1)** (kelas dengan probabilitas tertinggi).
 - Prediksi dan label asli disimpan untuk evaluasi metrik.

3. Evaluasi Metrik

Setelah semua batch selesai:

- **Akurasi:**
 - Menghitung persentase prediksi yang benar.
- **Precision, Recall, F1 Score:**
 - Menggunakan fungsi scikit-learn:
 - **precision_score:** Akurasi prediksi kelas tertentu.
 - **recall_score:** Kemampuan model mendeteksi kelas tertentu.
 - **f1_score:** Harmoni antara precision dan recall.
 - **average="weighted"** memastikan metrik memperhitungkan jumlah sampel di setiap kelas.

4. Logging Hasil

Hasil pelatihan per epoch dicetak:

- **Loss rata-rata** untuk seluruh batch.
 - **Accuracy, Precision, Recall, dan F1 Score** untuk evaluasi performa.
-

```

# Testing loop
model.eval()
test_loss = 0
all_labels = []
all_predictions = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Track loss
        test_loss += loss.item()

        # Track predictions and labels
        _, predicted = torch.max(outputs, 1)
        all_predictions.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Calculate metrics
test_accuracy = (np.array(all_predictions) == np.array(all_labels)).mean() * 100
test_precision = precision_score(all_labels, all_predictions, average="weighted")
test_recall = recall_score(all_labels, all_predictions, average="weighted")
test_f1 = f1_score(all_labels, all_predictions, average="weighted")

print(f"Test Loss: {test_loss/len(test_loader):.4f} | Test Accuracy: {test_accuracy:.2f}%")
print(f"Precision: {test_precision:.4f} | Recall: {test_recall:.4f} | F1 Score: {test_f1:.4f}")

Test Loss: 0.8849 | Test Accuracy: 75.95%
Precision: 0.7622 | Recall: 0.7595 | F1 Score: 0.7564

```

Kode ini melakukan evaluasi model CNN pada dataset pengujian dengan langkah berikut:

1. Model Evaluasi

- **model.eval()**: Mengatur model ke mode evaluasi. Pada mode ini:
 - Layer seperti dropout dinonaktifkan.
 - Gradient tidak dihitung untuk menghemat memori dan waktu.
-

2. Testing Loop

- **Disable Gradient Calculation**:
 - **torch.no_grad()**: Mematikan perhitungan gradien untuk mempercepat inferensi dan menghemat memori.
 - **Iterasi Batch pada test_loader**:
 - Data gambar dan label dipindahkan ke perangkat menggunakan **.to(device)**.
 - **Forward Pass**:
 - **outputs = model(images)**: Model menghasilkan output prediksi (logits).
 - **loss = criterion(outputs, labels)**: Menghitung loss menggunakan **CrossEntropyLoss**.
 - **Tracking Loss dan Prediksi**:
 - Loss batch ditambahkan ke **test_loss**.
 - Prediksi dihitung menggunakan **torch.max(outputs, 1)**.
 - Label asli dan prediksi disimpan untuk evaluasi metrik.
-

3. Perhitungan Metrik

	<p>Setelah semua batch selesai:</p> <ul style="list-style-type: none"> • Akurasi: <ul style="list-style-type: none"> ○ Persentase prediksi yang benar dibandingkan dengan label asli. • Precision, Recall, F1 Score: <ul style="list-style-type: none"> ○ Menggunakan fungsi dari scikit-learn dengan rata-rata tertimbang (<code>average="weighted"</code>) untuk memperhitungkan jumlah sampel di setiap kelas. <hr/> <h4>4. Logging Hasil</h4> <p>Hasil evaluasi dicetak:</p> <ul style="list-style-type: none"> • Test Loss: Rata-rata loss dari seluruh batch. • Test Accuracy: Persentase prediksi benar. • Precision: Akurasi prediksi per kelas, dengan rata-rata tertimbang. • Recall: Kemampuan model mendeteksi sampel setiap kelas. • F1 Score: Harmoni antara precision dan recall.
--	--

Fine-Tuning Pre-Trained Model ResNet18

Gambar	Penjelasan
<pre># Data Augmentation transform = transforms.Compose([transforms.Resize((224, 224)), transforms.RandomHorizontalFlip(p=0.5), transforms.RandomRotation(15), transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1), transforms.ToTensor(), transforms.Normalize([0.5], [0.5])]) # Load datasets train_dataset = datasets.ImageFolder("dataset_cnn/train", transform=transform) test_dataset = datasets.ImageFolder("dataset_cnn/test", transform=transforms.Compose([transforms.Resize((224, 224)), transforms.ToTensor(), transforms.Normalize([0.5], [0.5])])) train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True) test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False) # Optimizer with Weight Decay optimizer = optim.Adam(model.parameters(), lr=0.0001, weight_decay=1e-4) # Learning Rate Scheduler scheduler = ReduceLROnPlateau(optimizer, mode='min', patience=3, factor=0.1)</pre>	<p>Kode ini meningkatkan pelatihan model dengan augmentasi data, penggunaan weight decay, dan scheduler untuk pengaturan learning rate:</p> <hr/> <h4>1. Augmentasi Data</h4> <ul style="list-style-type: none"> • Augmentasi diterapkan pada dataset pelatihan untuk meningkatkan keragaman data: <ul style="list-style-type: none"> ○ Resize(224, 224): Mengubah ukuran gambar menjadi 224x224 piksel.

- **RandomHorizontalFlip(p=0.5)**: Membalik gambar secara horizontal dengan probabilitas 50%.
- **RandomRotation(15)**: Memutar gambar secara acak hingga 15 derajat.
- **ColorJitter**: Mengubah kecerahan, kontras, saturasi, dan hue secara acak.
- **ToTensor()**: Mengubah gambar menjadi tensor.
- **Normalize([0.5], [0.5])**: Menormalkan piksel gambar ke rentang $[-1,1]$ $[-1, 1]$ $[-1,1]$.
- Dataset pengujian hanya diterapkan **transformasi dasar** (resize, tensor, dan normalisasi), tanpa augmentasi.

2. Dataset dan DataLoader

- **train_dataset**: Dataset pelatihan dengan augmentasi.
- **test_dataset**: Dataset pengujian tanpa augmentasi.
- **DataLoader**:
 - **train_loader**: Dataset pelatihan dengan batch size 32 dan pengacakan (`shuffle=True`).
 - **test_loader**: Dataset pengujian tanpa pengacakan (`shuffle=False`).

3. Optimizer dengan Weight Decay

- **optim.Adam**: Mengoptimalkan parameter model dengan:
 - **lr=0.0001**: Learning rate rendah untuk pelatihan yang lebih stabil.
 - **weight_decay=1e-4**: Regularisasi untuk mencegah overfitting dengan menambahkan penalti terhadap nilai besar parameter model.

4. Scheduler Learning Rate

- **ReduceLROnPlateau:**

- Secara otomatis menurunkan learning rate jika loss validasi (atau metrik lain yang ditentukan) tidak membaik.
- **mode='min'**: Scheduler memonitor penurunan nilai loss.
- **patience=3**: Learning rate akan dikurangi jika tidak ada perbaikan setelah 3 epoch berturut-turut.
- **factor=0.1**: Mengurangi learning rate menjadi 10% dari nilai saat ini.

```
# Training Loop with Metrics and Early Stopping
best_loss = float('inf')
patience = 5
counter = 0

for epoch in range(epochs):
    model.train()
    train_loss = 0

    all_predictions = []
    all_labels = []

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    # Track predictions and labels for metrics
    _, predicted = torch.max(outputs, 1)
    all_predictions.extend(predicted.cpu().numpy())
    all_labels.extend(labels.cpu().numpy())

    # Calculate metrics
    train_accuracy = np.array(all_predictions) == np.array(all_labels).mean() * 100
    train_precision = precision_score(all_labels, all_predictions, average="weighted")
    train_recall = recall_score(all_labels, all_predictions, average="weighted")
    train_f1 = f1_score(all_labels, all_predictions, average="weighted")

    print(f"Epoch [{epoch+1}/{epochs}]")
    print(f"Loss: {train_loss/len(train_loader):.4f} | Accuracy: {train_accuracy:.2f}%")
    print(f"Precision: {train_precision:.4f} | Recall: {train_recall:.4f} | F1 Score: {train_f1:.4f}")

    # Early stopping
    scheduler.step(train_loss)
    if train_loss < best_loss:
        best_loss = train_loss
        counter = 0
        torch.save(model.state_dict(), "best_model.pth")
    else:
        counter += 1
        if counter >= patience:
            print("Early stopping triggered.")
            break
```

Kode ini melatih model CNN dengan logging metrik, pengaturan learning rate menggunakan scheduler, dan mekanisme **early stopping** untuk menghentikan pelatihan ketika performa model tidak membaik.

1. Inisialisasi

- **best_loss = float('inf')**: Nilai awal untuk menyimpan loss terbaik.
- **patience = 5**: Jumlah epoch tanpa perbaikan sebelum **early stopping** diaktifkan.
- **counter = 0**: Menghitung jumlah epoch berturut-turut tanpa perbaikan pada loss terbaik.

2. Training Loop

Untuk setiap epoch:

a. Mode Pelatihan:

- **model.train()**: Mengatur model ke mode pelatihan (dropout diaktifkan).

b. Forward dan Backward Pass:

- **Forward Pass:**
 - `outputs = model(images)`: Model menghasilkan prediksi.
 - `loss = criterion(outputs, labels)`: Menghitung loss menggunakan `CrossEntropyLoss`.
- **Backward Pass:**
 - `optimizer.zero_grad()`: Membersihkan gradien sebelumnya.
 - `loss.backward()`: Menghitung gradien loss terhadap parameter model.
 - `optimizer.step()`: Memperbarui parameter model menggunakan gradien.

c. Tracking Loss dan Metrik:

- **Loss:**
 - Total loss batch ditambahkan ke `train_loss`.
- **Prediksi dan Metrik:**
 - Prediksi dihitung dengan `torch.max(outputs, 1)`.
 - Menggunakan Scikit-learn untuk menghitung `Accuracy`, `Precision`, `Recall`, dan `F1 Score` dengan rata-rata tertimbang (`average="weighted"`).

3. Scheduler dan Early Stopping

a. Scheduler:

- `scheduler.step(train_loss)`: Menyesuaikan learning rate berdasarkan loss pelatihan.

b. Early Stopping:

- Jika `train_loss` lebih kecil dari `best_loss`:
 - Simpan loss sebagai `best_loss`.
 - Reset `counter = 0`.
 - Simpan model terbaik ke file `best_model.pth`.

- Jika **train_loss** tidak membaik:
 - Increment **counter**.
 - Jika **counter >= patience**, hentikan pelatihan dengan **early stopping**.
-

4. Output

- Mencetak:
 - Loss rata-rata untuk epoch tersebut.
 - Akurasi, Precision, Recall, dan F1 Score.
- Early stopping dihentikan jika model tidak membaik selama 5 epoch berturut-turut (sesuai nilai **patience**).

```
# Evaluate the model on the test set
model.eval()
test_loss = 0
all_predictions = []
all_labels = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        test_loss += loss.item()

        # Track predictions and labels for metrics
        _, predicted = torch.max(outputs, 1)
        all_predictions.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Calculate metrics
test_accuracy = (np.array(all_predictions) == np.array(all_labels)).mean() * 100
test_precision = precision_score(all_labels, all_predictions, average="weighted")
test_recall = recall_score(all_labels, all_predictions, average="weighted")
test_f1 = f1_score(all_labels, all_predictions, average="weighted")

print("Test Loss: {:.4f}" .format(test_loss/len(test_loader)))
print("Test Accuracy: {:.2f}%".format(test_accuracy))
print("Test Precision: {:.4f} | Test Recall: {:.4f} | Test F1 Score: {:.4f}")

Test Loss: 0.7019
Test Accuracy: 77.98%
Test Precision: 0.7798 | Test Recall: 0.7798 | Test F1 Score: 0.7750
```

Kode ini mengevaluasi model CNN pada dataset pengujian dengan menghitung **loss** dan metrik evaluasi (Accuracy, Precision, Recall, dan F1 Score). Berikut penjelasannya:

1. Model Evaluasi

- **model.eval()**: Mengatur model ke mode evaluasi (menonaktifkan dropout dan batch normalization).
 - **torch.no_grad()**: Mematikan perhitungan gradien untuk menghemat memori dan mempercepat inferensi.
-

2. Testing Loop

- Iterasi pada setiap batch dalam **test_loader**:
 - **Forward Pass**:
 - **outputs = model(images)**: Model menghasilkan prediksi logits.
 - **loss = criterion(outputs,**

- **labels**): Menghitung loss untuk batch saat ini.
- **Track Loss:**
 - Loss batch ditambahkan ke `test_loss`.
- **Prediksi:**
 - Prediksi dihitung dengan `torch.max(outputs, 1)`, memilih kelas dengan probabilitas tertinggi.
 - Label asli dan prediksi disimpan untuk menghitung metrik.

3. Hitung Metrik Evaluasi

- **Accuracy:**
 - Persentase prediksi yang benar dibandingkan dengan label asli.
 - `test_accuracy` dihitung dengan membandingkan semua prediksi dengan semua label.
- **Precision, Recall, F1 Score:**
 - Menggunakan fungsi dari Scikit-learn:
 - **Precision:** Akurasi prediksi untuk setiap kelas.
 - **Recall:** Kemampuan model mendeteksi setiap kelas.
 - **F1 Score:** Harmoni antara Precision dan Recall.
 - `average="weighted"` memastikan metrik mempertimbangkan jumlah sampel di setiap kelas.

4. Output

- **Loss Rata-rata:** `test_loss/len(test_loader)`, rata-rata dari semua batch.
- **Akurasi:** Persentase prediksi benar.
- **Precision, Recall, F1 Score:** Menyediakan evaluasi yang lebih

	mendetail tentang performa model.
<pre> # Load the trained model model = models.resnet18(pretrained=False) # Replace with ResNet50 if used num_features = model.fc.in_features model.fc = nn.Linear(num_features, 4) # Adjust the final layer for 4 classes model.load_state_dict(torch.load("best_model.pth")) # Load the saved model model = model.to(device) model.eval() # Set the model to evaluation mode # Evaluate the model on the test set test_loss = 0 all_predictions = [] all_labels = [] with torch.no_grad(): for images, labels in test_loader: images, labels = images.to(device), labels.to(device) # Forward pass outputs = model(images) loss = criterion(outputs, labels) test_loss += loss.item() # Store predictions and labels predicted = torch.argmax(outputs, 1) all_predictions.extend(predicted.cpu().numpy()) all_labels.extend(labels.cpu().numpy()) # Calculate metrics print(f"Test Loss: {(test_loss/len(test_loader)),4f}") print(f"Test Accuracy: {(test_accuracy:.2f)%}") print(f"Test Precision: {(test_precision:.4f)} Test Recall: {(test_recall:.4f)} Test F1 Score: {(test_f1:.4f)}") # Generate confusion matrix confusion_matrix(all_labels, all_predictions) disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=test_dataset.classes) disp.plot(cmap="Blues", xticks_rotation='vertical') plt.title("Confusion Matrix") plt.show() </pre>	<p>Kode ini memuat model yang telah dilatih, mengevaluasi performanya pada dataset pengujian, dan menampilkan metrik evaluasi serta confusion matrix.</p> <hr/> <h2>1. Memuat Model yang Telah Dilatih</h2> <ul style="list-style-type: none"> ● Model Arsitektur: <ul style="list-style-type: none"> ○ model = models.resnet18(pretrained=False): Membuat model ResNet-18 tanpa bobot pre-trained. ○ model.fc: Lapisan fully connected terakhir diubah untuk 4 kelas dengan nn.Linear(num_features, 4). ● Memuat Bobot: <ul style="list-style-type: none"> ○ model.load_state_dict(torch.load("best_model.pth")): Memuat bobot model terbaik yang telah disimpan. ○ model = model.to(device): Memindahkan model ke perangkat (GPU atau CPU). ● Mode Evaluasi: <ul style="list-style-type: none"> ○ model.eval(): Mengatur model ke mode evaluasi. <hr/> <h2>2. Evaluasi Dataset Pengujian</h2> <h3>a. Testing Loop:</h3> <ul style="list-style-type: none"> ● Tanpa Gradien: <ul style="list-style-type: none"> ○ torch.no_grad(): Mematikan perhitungan gradien untuk menghemat memori dan mempercepat inferensi. ● Iterasi pada Batch: <ul style="list-style-type: none"> ○ Data gambar dan label dipindahkan ke perangkat menggunakan .to(device).

- **Forward Pass:**
 - **outputs = model(images):** Model menghasilkan prediksi logits.
 - **loss = criterion(outputs, labels):** Menghitung loss untuk batch saat ini.
- **Simpan Hasil:**
 - Loss batch ditambahkan ke **test_loss**.
 - Prediksi dihitung dengan **torch.max(outputs, 1)** dan disimpan bersama label.

b. Hitung Metrik Evaluasi:

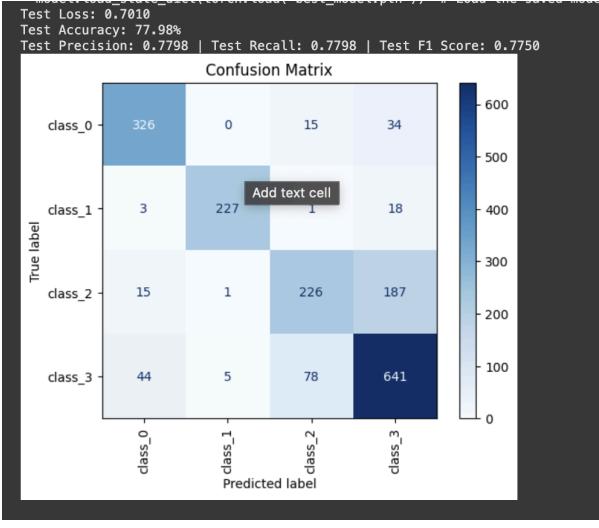
- **Accuracy:**
 - Persentase prediksi yang benar dibandingkan dengan label asli.
 - **Precision, Recall, F1 Score:**
 - Menggunakan fungsi Scikit-learn dengan rata-rata tertimbang (**average="weighted"**) untuk memperhitungkan distribusi kelas.
-

3. Confusion Matrix

- **confusion_matrix:**
 - Matriks yang menunjukkan jumlah prediksi benar dan salah untuk setiap kelas.
 - **ConfusionMatrixDisplay:**
 - Memvisualisasikan confusion matrix dengan nama kelas dari dataset.
-

4. Output

- **Metrik Evaluasi:**
 - Test Loss, Accuracy, Precision, Recall, dan F1 Score ditampilkan

	<ul style="list-style-type: none"> di konsol. <ul style="list-style-type: none"> Confusion Matrix: <ul style="list-style-type: none"> Visualisasi matriks kesalahan untuk melihat distribusi prediksi per kelas.
 <pre> Test Loss: 0.7010 Test Accuracy: 77.98% Test Precision: 0.7798 Test Recall: 0.7798 Test F1 Score: 0.7750 Confusion Matrix True label class_0 - 326 0 15 34 class_1 - 3 227 1 18 class_2 - 15 1 226 187 class_3 - 44 5 78 641 Predicted label class_0 class_1 class_2 class_3 </pre>	<h3>Kesimpulan</h3> <ul style="list-style-type: none"> Model menunjukkan performa yang baik dengan 78% akurasi dan metrik tambahan (precision, recall, dan F1 score) mendukung hasil ini. Namun, ada ruang untuk peningkatan: <ul style="list-style-type: none"> Mengoptimalkan hyperparameter (learning rate, batch size, weight decay, dll.). Menambah data atau melakukan augmentasi data tambahan. Menggunakan model yang lebih kompleks seperti ResNet50 atau EfficientNet untuk meningkatkan akurasi. Meninjau hasil confusion matrix untuk mengetahui kelas mana yang paling sering salah diprediksi.

Pre-Trained Model ResNet50

Gambar	Penjelasan

```

# Data augmentation for training set
train_transform = transforms.Compose([
    transforms.Resize((224, 224)), # Resize images to 224x224
    transforms.RandomHorizontalFlip(p=0.5), # Randomly flip images horizontally
    transforms.RandomRotation(15), # Rotate images up to 15 degrees
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1), # Brighten or darken images
    transforms.ToTensor(), # Convert to tensor
    transforms.Normalize([0.5], [0.5]) # Normalize to [-1, 1]
])

# Data transformation for test set
test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])

# Load datasets
train_dataset = datasets.ImageFolder("dataset_cnn/train", transform=train_transform)
test_dataset = datasets.ImageFolder("dataset_cnn/test", transform=test_transform)

# Create data loaders
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

```

Kode ini mempersiapkan dataset untuk pelatihan dan pengujian dengan menggunakan **data augmentation** pada dataset pelatihan, serta menerapkan **transformasi standar** pada dataset pengujian.

1. Data Augmentation untuk Training Set

- **Transformasi Augmentasi:**

- **Resize((224, 224)):** Mengubah ukuran gambar menjadi 224x224 piksel.
 - **RandomHorizontalFlip(p=0.5):** Membalik gambar secara horizontal dengan probabilitas 50%.
 - **RandomRotation(15):** Memutar gambar secara acak hingga 15 derajat.
 - **ColorJitter:** Mengubah kecerahan, kontras, saturasi, dan hue secara acak untuk meningkatkan keragaman visual.
 - **ToTensor():** Mengubah gambar menjadi tensor PyTorch.
 - **Normalize([0.5], [0.5]):** Menormalkan nilai piksel gambar ke rentang [-1,1][-1, 1][-1,1].
-

2. Transformasi untuk Test Set

- Transformasi dasar tanpa augmentasi:

- **Resize((224, 224)):** Mengubah ukuran gambar.
 - **ToTensor():** Mengonversi gambar ke tensor.
 - **Normalize([0.5], [0.5]):** Menormalkan piksel untuk menjaga konsistensi dengan dataset pelatihan.
-

3. Memuat Dataset

- **datasets.ImageFolder:**

- **train_dataset:** Memuat dataset pelatihan dari folder **dataset_cnn/train** dengan augmentasi.
- **test_dataset:** Memuat dataset pengujian

dari folder `dataset_cnn/test` dengan transformasi standar.

4. DataLoader

- **train_loader:**
 - Batch size: 32.
 - **shuffle=True:** Mengacak urutan data untuk meningkatkan generalisasi.
- **test_loader:**
 - Batch size: 32.
 - **shuffle=False:** Tidak mengacak urutan untuk menjaga konsistensi evaluasi.

```
] # Load pretrained ResNet50
model = models.resnet50(pretrained=True)

# Modify the final fully connected layer for 4 classes
num_features = model.fc.in_features
model.fc = nn.Sequential(
    nn.Linear(num_features, 256), # Add an intermediate layer
    nn.ReLU(),
    nn.Dropout(0.5),           # Dropout for regularization
    nn.Linear(256, 4)          # Output layer for 4 classes
)
model = model.to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001, weight_decay=1e-4)
```

Kode ini memuat model **ResNet50** dengan bobot pre-trained dan menyesuaikan arsitektur akhir untuk klasifikasi 4 kelas. Berikut penjelasannya:

1. Memuat ResNet50 Pre-trained

- **model = models.resnet50(pretrained=True):**
 - Menggunakan ResNet50 yang telah dilatih sebelumnya pada dataset ImageNet.
 - Pre-trained weights membantu model memulai pelatihan dengan fitur yang sudah terlatih, mempercepat konvergensi dan meningkatkan akurasi, terutama pada dataset kecil.

2. Modifikasi Fully Connected Layer

- **model.fc:**
 - Layer fully connected terakhir dimodifikasi untuk klasifikasi 4 kelas:
 - **nn.Linear(num_features, 256):**
 - Mengubah jumlah fitur dari output ResNet50 menjadi 256 untuk intermediate layer.

- **nn.ReLU()**:
 - Fungsi aktivasi ReLU untuk non-linearitas.
 - **nn.Dropout(0.5)**:
 - Dropout dengan probabilitas 50% untuk mencegah overfitting.
 - **nn.Linear(256, 4)**:
 - Layer output untuk 4 kelas.
-

3. Memindahkan Model ke Perangkat

- **model = model.to(device)**:
 - Memindahkan model ke perangkat yang tersedia (GPU atau CPU) untuk pelatihan.
-

4. Fungsi Loss

- **criterion = nn.CrossEntropyLoss()**:
 - Fungsi loss untuk klasifikasi multi-kelas. Membandingkan logits output model dengan label ground truth.
-

5. Optimizer

- **optimizer = optim.Adam(model.parameters(), lr=0.0001, weight_decay=1e-4)**:
 - Menggunakan optimizer Adam:
 - **lr=0.0001**: Learning rate rendah untuk pelatihan yang stabil.
 - **weight_decay=1e-4**: Regularisasi L2 untuk mengurangi overfitting dengan menambahkan penalti pada parameter besar.

```

# Early stopping parameters
best_loss = float('inf')
patience = 5
counter = 0

# Training loop
epochs = 20
for epoch in range(epochs):
    model.train()
    train_loss = 0

    all_predictions = []
    all_labels = []

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    # Track predictions and labels for metrics
    _, predicted = torch.max(outputs, 1)
    all_predictions.extend(predicted.cpu().numpy())
    all_labels.extend(labels.cpu().numpy())

    # Calculate metrics
    train_accuracy = (np.array(all_predictions) == np.array(all_labels)).mean() * 100
    train_precision = precision_score(all_labels, all_predictions, average="weighted")
    train_recall = recall_score(all_labels, all_predictions, average="weighted")
    train_f1 = f1_score(all_labels, all_predictions, average="weighted")

    print(f"Epoch [{epoch+1}/{epochs}]")
    print(f"Loss: {train_loss/len(train_loader):.4f} | Accuracy: {train_accuracy:.2f}%")
    print(f"Precision: {train_precision:.4f} | Recall: {train_recall:.4f} | F1 Score: {train_f1:.4f}")

    # Early stopping
    scheduler.step(train_loss)
    if train_loss < best_loss:
        best_loss = train_loss
        counter = 0
        torch.save(model.state_dict(), "best_model.pth")
    else:
        counter += 1
        if counter >= patience:
            print("Early stopping triggered.")
            break

# Early stopping triggered | Recall: 0.9325 | F1 Score: 0.9325
Epoch [20/20]
Loss: 0.1756 | Accuracy: 93.25%
Precision: 0.9331 | Recall: 0.9325 | F1 Score: 0.9325

```

Learning Rate Scheduler:

- **scheduler = ReduceLROnPlateau:**

Menurunkan **learning rate** secara adaptif jika **loss** tidak membaik setelah 3 epoch (patience). Learning rate dikurangi sebesar **10%** (factor = 0.1).

Early Stopping:

- **best_loss:** Menyimpan nilai loss terbaik.
- **patience = 5:** Jika loss tidak membaik selama 5 epoch berturut-turut, pelatihan dihentikan.
- **counter:** Menghitung jumlah epoch tanpa perbaikan pada loss.

Training Loop:

- **Pelatihan:**

- Data gambar dimasukkan ke dalam model untuk menghasilkan **output logits**.
- **Loss** dihitung dengan membandingkan output model dengan label sebenarnya.
- Optimizer memperbarui parameter model menggunakan **backpropagation**.

- **Metrik Evaluasi:**

- Menghitung **Accuracy**, **Precision**, **Recall**, dan **F1 Score** menggunakan prediksi dan label.

Early Stopping & Scheduler:

- Jika **train_loss** lebih rendah dari **best_loss**, model disimpan sebagai model terbaik (**best_model.pth**) dan counter direset.
- Jika tidak, **counter** meningkat, dan jika mencapai **patience**, pelatihan dihentikan.

Model berhasil mencapai performa yang baik dengan **93.25% akurasi** di dataset pelatihan.

Early stopping tidak terpicu karena **train_loss** terus membaik hingga epoch terakhir (20).

Scheduler membantu menyesuaikan **learning rate**, yang dapat meningkatkan stabilitas pelatihan.

```

# Load the best model
model.load_state_dict(torch.load("best_model.pth"))
model.eval()

# Test the model
test_loss = 0
all_predictions = []
all_labels = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

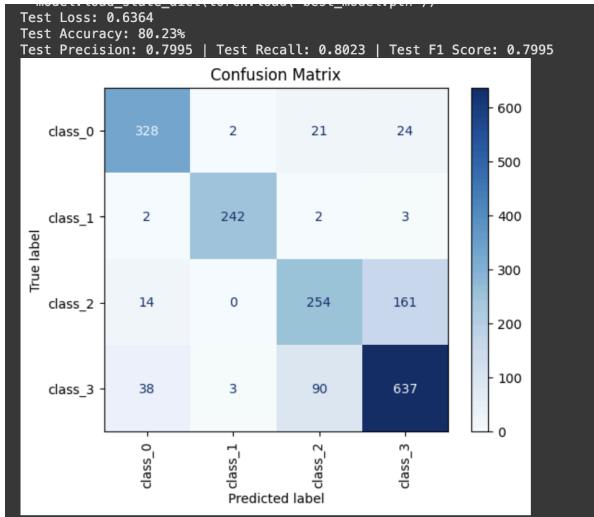
        # Track predictions and labels for metrics
        _, predicted = torch.max(outputs, 1)
        all_predictions.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Calculate metrics
test_accuracy = (np.array(all_predictions) == np.array(all_labels)).mean() * 100
test_precision = precision_score(all_labels, all_predictions, average="weighted")
test_recall = recall_score(all_labels, all_predictions, average="weighted")
test_f1 = f1_score(all_labels, all_predictions, average="weighted")

# Print metrics
print(f"Test Loss: {test_loss/len(test_loader):.4f}")
print(f"Test Accuracy: {test_accuracy:.2f}%")
print(f"Test Precision: {test_precision:.4f} | Test Recall: {test_recall:.4f} | Test F1 Score: {test_f1:.4f}")

# Confusion matrix
cm = confusion_matrix(all_labels, all_predictions)
disp = ConfusionMatrixDisplay(cm, display_labels=train_dataset.classes)
disp.plot(cmap="Blues", xticks_rotation="vertical")
plt.title("Confusion Matrix")
plt.show()

```



Memuat Model Terbaik:

- **model.load_state_dict(torch.load("best_model.pth")):** Memuat bobot model terbaik yang disimpan selama pelatihan.
- **model.eval():** Mengatur model ke mode evaluasi, di mana layer seperti dropout dinonaktifkan.

Pengujian Model:

- **Tanpa Gradien:**
 - **torch.no_grad():** Mematikan perhitungan gradien untuk menghemat memori dan mempercepat inferensi.
- **Iterasi pada test_loader:**
 - Gambar dan label dipindahkan ke perangkat menggunakan **.to(device)**.
 - **Forward Pass:**
 - Model menghasilkan prediksi logits.
 - **Loss** dihitung dengan **CrossEntropyLoss**.
 - **Tracking Prediksi:**
 - Prediksi dihitung dengan **torch.max(outputs, 1)**, menyimpan prediksi dan label untuk menghitung metrik.

Menghitung Metrik:

- **Accuracy:**
 - Persentase prediksi yang benar dibandingkan dengan label asli.
- **Precision, Recall, F1 Score:**
 - Menggunakan fungsi Scikit-learn dengan rata-rata tertimbang untuk memperhitungkan distribusi kelas.

Confusion Matrix:

- **confusion_matrix:** Matriks yang menunjukkan distribusi prediksi benar dan salah untuk setiap kelas.
- **ConfusionMatrixDisplay:** Menampilkan confusion matrix secara visual dengan pewarnaan (Blues).

Output:

- Model menunjukkan performa pengujian yang cukup baik dengan **80.23% akurasi**, namun ada sedikit penurunan performa dibandingkan pelatihan (**93.25% akurasi**).
- Metrik **Precision, Recall, dan F1 Score** mendekati nilai akurasi, menunjukkan bahwa model cukup konsisten dalam menangani ketidakseimbangan kelas.
- **Confusion Matrix** memberikan informasi rinci tentang kelas mana yang sering salah diprediksi, sehingga dapat digunakan untuk analisis lebih lanjut dan perbaikan.

Pre-Trained Model EfficientNet-B0

Gambar	Penjelasan
<pre># Data augmentation for training set train_transform = transforms.Compose([transforms.Resize(224, 224), transforms.RandomHorizontalFlip(p=0.5), transforms.RandomRotation(15), transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1), transforms.ToTensor(), transforms.Normalize([0.5], [0.5])]) # Data transformation for test set test_transform = transforms.Compose([transforms.Resize(224, 224), transforms.ToTensor(), transforms.Normalize([0.5], [0.5])]) # Load datasets train_dataset = datasets.ImageFolder("dataset_cnn/train", transform=train_transform) test_dataset = datasets.ImageFolder("dataset_cnn/test", transform=test_transform) # Data loaders batch_size = 32 train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True) test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)</pre>	<p>Penjelasan Singkat Kode</p> <p>Kode ini bertujuan untuk mempersiapkan dataset pelatihan dan pengujian dengan transformasi gambar dan augmentasi data. Berikut penjelasan setiap bagian:</p> <hr/> <p>1. Data Augmentation untuk Training Set</p> <ul style="list-style-type: none"> • Transformasi: <ul style="list-style-type: none"> ◦ Resize(224, 224): Mengubah ukuran gambar menjadi 224x224 piksel. ◦ RandomHorizontalFlip(p=0.5): Membalik gambar secara horizontal dengan probabilitas 50%. ◦ RandomRotation(15): Memutar gambar secara acak hingga 15 derajat. ◦ ColorJitter: Mengubah kecerahan, kontras, saturasi, dan hue secara acak untuk menambah

- keragaman visual.
 - **ToTensor()**: Mengubah gambar menjadi tensor.
 - **Normalize([0.5], [0.5])**: Menormalkan nilai piksel gambar ke rentang [-1,1][-1, 1] [-1,1].
-

2. Transformasi untuk Test Set

- **Transformasi Standar:**

- **Resize((224, 224))**: Mengubah ukuran gambar untuk menjaga konsistensi.
 - **ToTensor()**: Mengonversi gambar ke tensor.
 - **Normalize([0.5], [0.5])**: Menormalkan nilai piksel tanpa augmentasi tambahan (agar pengujian tetap obyektif).
-

3. Memuat Dataset

- **datasets.ImageFolder**:

- **train_dataset**:
 - Memuat dataset pelatihan dari folder `dataset_cnn/train` dengan augmentasi transformasi.
 - **test_dataset**:
 - Memuat dataset pengujian dari folder `dataset_cnn/test` dengan transformasi standar.
-

4. DataLoader

- **train_loader**:

- Batch size: 32.
- **shuffle=True**: Mengacak urutan data untuk meningkatkan generalisasi selama pelatihan.

- **test_loader**:

- Batch size: 32.
- **shuffle=False**: Tidak mengacak

	<p>urutan data untuk evaluasi yang konsisten.</p>
<pre> # Load pretrained EfficientNet-B0 model = efficientnet_b0(pretrained=True) # Modify the classifier num_features = model.classifier[1].in_features model.classifier = nn.Sequential(nn.Linear(num_features, 256), # Add an intermediate layer nn.ReLU(), nn.Dropout(0.5), # Regularization nn.Linear(256, 4) # Output layer for 4 classes) model = model.to(device) # Define loss function and optimizer criterion = nn.CrossEntropyLoss() optimizer = optim.Adam(model.parameters(), lr=0.0001, weight_decay=1e-4) </pre>	<p>Kode ini memuat model EfficientNet-B0 pre-trained, memodifikasi layer klasifikasi untuk 4 kelas, dan menyiapkan fungsi loss serta optimizer.</p> <p>Memuat EfficientNet-B0 Pre-trained</p> <ul style="list-style-type: none"> ● efficientnet_b0(pretrained=True): <ul style="list-style-type: none"> ○ Menggunakan EfficientNet-B0 yang telah dilatih sebelumnya pada dataset ImageNet. ○ Memanfaatkan pre-trained weights untuk mempercepat pelatihan dan meningkatkan akurasi, terutama pada dataset yang lebih kecil. <hr/> <p>2. Modifikasi Layer Klasifikasi</p> <ul style="list-style-type: none"> ● Layer Klasifikasi Asli: <ul style="list-style-type: none"> ○ EfficientNet-B0 memiliki layer akhir default untuk 1000 kelas (ImageNet). ● Modifikasi: <ul style="list-style-type: none"> ○ num_features = model.classifier[1].in_features: <ul style="list-style-type: none"> ■ Mendapatkan jumlah input dari layer terakhir EfficientNet. ○ model.classifier: <ul style="list-style-type: none"> ■ Layer classifier diubah menjadi: <ul style="list-style-type: none"> ■ nn.Linear(num_features, 256): Intermediate layer dengan 256 unit. ■ nn.ReLU(): Aktivasi non-linear ReLU. ■ nn.Dropout(0.5): Dropout dengan probabilitas 50% untuk

- regularisasi.
- **nn.Linear(256, 4)**: Layer akhir untuk 4 kelas.
-

3. Memindahkan Model ke Perangkat

- **model = model.to(device)**:
 - Memindahkan model ke perangkat (GPU atau CPU) untuk pelatihan.
-

4. Fungsi Loss

- **criterion = nn.CrossEntropyLoss()**:
 - Fungsi loss untuk klasifikasi multi-kelas, membandingkan logits output dengan label ground truth.
-

5. Optimizer

- **optimizer = optim.Adam**:
 - Mengoptimalkan parameter model dengan:
 - **lr=0.0001**: Learning rate rendah untuk pelatihan stabil.
 - **weight_decay=1e-4**: Regularisasi L2 untuk mencegah overfitting dengan menambahkan penalti pada parameter besar.

```

from torch.optim.lr_scheduler import CosineAnnealingLR

# Learning rate scheduler
scheduler = CosineAnnealingLR(optimizer, T_max=10)

# Early stopping parameters
best_loss = float('inf')
patience = 5
counter = 0

# Training loop
epochs = 20
for epoch in range(epochs):
    model.train()
    train_loss = 0

    all_predictions = []
    all_labels = []

    # Forward pass
    outputs = model(images)
    loss = criterion(outputs, labels)

    # Backward pass
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    train_loss += loss.item()

    # Track predictions and labels for metrics
    _, predicted = torch.max(outputs, 1)
    all_predictions.extend(predicted.cpu().numpy())
    all_labels.extend(labels.cpu().numpy())

    # Calculate metrics
    train_accuracy = (np.array(all_predictions) == np.array(all_labels)).mean() * 100
    train_precision = precision_score(all_labels, all_predictions, average="weighted")
    train_recall = recall_score(all_labels, all_predictions, average="weighted")
    train_f1 = f1_score(all_labels, all_predictions, average="weighted")

    print(f"Epoch [{epoch+1}/{epochs}]")
    print(f"Loss: {train_loss/len(train_loader):.4f} | Accuracy: {train_accuracy:.2f}%")
    print(f"Precision: {train_precision:.4f} | Recall: {train_recall:.4f} | F1 Score: {train_f1:.4f}")

    # Early stopping
    scheduler.step(train_loss)
    if train_loss < best_loss:
        best_loss = train_loss
        counter = 0
        torch.save(model.state_dict(), "best_model_efficientnet_b0.pth")
    else:
        counter += 1
        if counter >= patience:
            print("Early stopping triggered.")
            break

```

Kode ini melatih model **EfficientNet-B0** dengan menggunakan **CosineAnnealingLR** sebagai scheduler untuk pengaturan learning rate, serta menerapkan mekanisme **early stopping**.

1. Learning Rate Scheduler

- **scheduler = CosineAnnealingLR(optimizer, T_max=10):**
 - Scheduler ini menyesuaikan learning rate secara **kosinusial** dari nilai awal hingga mendekati nol selama **T_max=10** epoch.
 - Membantu model mencapai konvergensi yang lebih stabil di akhir pelatihan.

2. Early Stopping

- **best_loss = float('inf')**: Menyimpan nilai loss terbaik selama pelatihan.
- **patience = 5**: Jika loss tidak membaik selama 5 epoch berturut-turut, pelatihan dihentikan.
- **counter = 0**: Menghitung jumlah epoch tanpa perbaikan.

3. Training Loop

a. Mode Pelatihan:

- **model.train()**: Mengaktifkan mode pelatihan dengan mengaktifkan dropout dan layer yang memerlukan gradien.

b. Forward dan Backward Pass:

- **Forward Pass:**
 - **outputs = model(images)**: Model menghasilkan predksi logits.
 - **loss = criterion(outputs, labels)**: Menghitung loss antara predksi

- dan label ground truth.
- **Backward Pass:**
 - Gradien dihitung dengan `loss.backward()`.
 - Optimizer memperbarui parameter model dengan `optimizer.step()`.
- c. **Tracking Loss dan Prediksi:**
 - Loss batch ditambahkan ke `train_loss`.
 - Prediksi dihitung dengan `torch.max(outputs, 1)` dan disimpan untuk evaluasi metrik.
- d. **Evaluasi Metrik:**
 - **Accuracy, Precision, Recall, dan F1 Score** dihitung menggunakan Scikit-learn berdasarkan semua prediksi dan label untuk setiap epoch.

4. Scheduler dan Early Stopping

- **Scheduler:**
 - `scheduler.step(train_loss)` menyesuaikan learning rate berdasarkan pola cosine annealing.
- **Early Stopping:**
 - Jika loss membaik, model disimpan sebagai `best_model_efficientnet_b0.pth`.
 - Jika loss tidak membaik selama **5 epoch**, pelatihan dihentikan lebih awal.

Model Performance:

- Model memiliki performa tinggi dengan akurasi 95.43% dan metrik lainnya mendekati nilai yang sama.

```

# Load the best model
model.load_state_dict(torch.load("best_model_efficientnet_b0.pth"))
model.eval()

# Test the model
test_loss = 0
all_predictions = []
all_labels = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

        # Track predictions and labels for metrics
        _, predicted = torch.max(outputs, 1)
        all_predictions.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Calculate metrics
test_accuracy = (np.array(all_predictions) == np.array(all_labels)).mean() * 100
test_precision = precision_score(all_labels, all_predictions, average="weighted")
test_recall = recall_score(all_labels, all_predictions, average="weighted")
test_f1 = f1_score(all_labels, all_predictions, average="weighted")

# Print metrics
print(f"Test Loss: {test_loss/len(test_loader):.4f}")
print(f"Test Accuracy: {test_accuracy:.2f}%")
print(f"Test Precision: {test_precision:.4f} | Test Recall: {test_recall:.4f} | Test F1 Score: {test_f1:.4f}")

# Confusion matrix
cm = confusion_matrix(all_labels, all_predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=train_dataset.classes)
disp.figure_( figsize=(8, 5), xticks_rotation="vertical" )
plt.title("Confusion Matrix")
plt.show()

```

Kode ini mengevaluasi **model terbaik** yang telah disimpan pada dataset pengujian untuk menghitung **loss** dan metrik evaluasi (Accuracy, Precision, Recall, F1 Score), serta menampilkan **confusion matrix**.

1. Memuat Model Terbaik

- **model.load_state_dict(torch.load("best_model_efficientnet_b0.pth")):**
 - Memuat bobot model terbaik yang disimpan selama pelatihan.
- **model.eval():**
 - Mengatur model ke mode evaluasi (dropout dinonaktifkan).

2. Evaluasi Dataset Pengujian

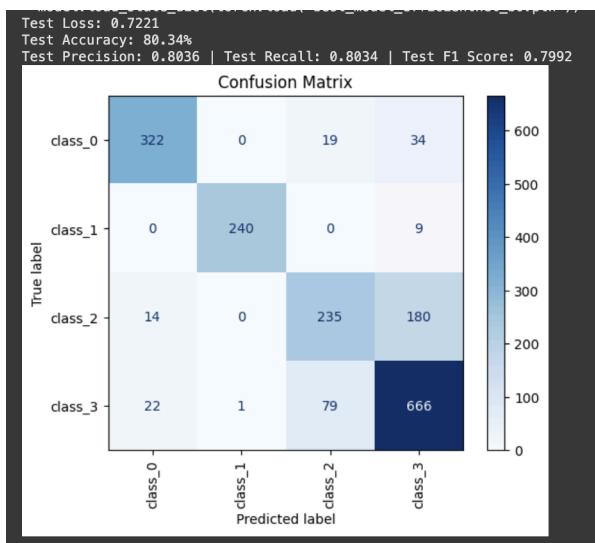
- **Tanpa Gradien:**
 - **torch.no_grad()**: Mematikan perhitungan gradien untuk menghemat memori dan mempercepat inferensi.
- **Iterasi pada Batch:**
 - Data gambar dan label dipindahkan ke perangkat (GPU/CPU).
 - **Forward Pass:**
 - **outputs = model(images)**: Model menghasilkan prediksi logits.
 - **loss = criterion(outputs, labels)**: Menghitung loss batch saat ini.
 - **Tracking:**
 - Prediksi dihitung dengan **torch.max(outputs, 1)**.
 - Prediksi dan label disimpan untuk evaluasi metrik.

3. Menghitung Metrik

- **Accuracy:**
 - Persentase prediksi benar dibandingkan total data uji.
- **Precision:**
 - Ketepatan prediksi model untuk kelas yang benar.
- **Recall:**
 - Kemampuan model mendeteksi sampel yang benar dari setiap kelas.
- **F1 Score:**
 - Kombinasi harmonis antara Precision dan Recall.
- **Confusion Matrix:**
 - Matriks yang menunjukkan distribusi prediksi benar dan salah untuk setiap kelas.

4. Visualisasi Confusion Matrix

- **ConfusionMatrixDisplay:**
 - Menampilkan **confusion matrix** dengan pewarnaan untuk mempermudah interpretasi hasil.



Kesimpulan

- **Performa Model:**
 - Model menunjukkan performa pengujian yang baik dengan akurasi **80.34%**, mendekati hasil pelatihan.
- **Confusion Matrix:**
 - Memberikan gambaran rinci tentang distribusi prediksi model untuk setiap kelas.

Pre-Trained Model EfficientNet-B3

Gambar	Penjelasan
<pre># Load pretrained EfficientNet-B3 model = efficientnet_b3(pretrained=True) # Modify the classifier num_features = model.classifier[1].in_features model.classifier = nn.Sequential(nn.Linear(num_features, 256), # Add an intermediate layer nn.ReLU(), nn.Dropout(0.5), nn.Linear(256, 4) # 4 classes) model = model.to(device) # Loss function with label smoothing criterion = nn.CrossEntropyLoss(label_smoothing=0.1) # Optimizer and scheduler optimizer = optim.AdamW(model.parameters(), lr=0.001, weight_decay=1e-4) scheduler = optim.lr_scheduler.OneCycleR(optimizer, max_lr=0.0005, steps_per_epoch=len(train_loader), epochs=20)</pre>	<p>Kode ini memuat EfficientNet-B3 pre-trained, memodifikasi layer klasifikasi untuk 4 kelas, dan menyiapkan loss function dengan label smoothing, optimizer, serta learning rate scheduler.</p> <hr/> <h3>1. Memuat EfficientNet-B3 Pre-trained</h3> <ul style="list-style-type: none">• efficientnet_b3(pretrained=True):<ul style="list-style-type: none">○ Menggunakan EfficientNet-B3 dengan bobot pre-trained dari ImageNet.○ Pre-trained weights mempercepat pelatihan dan meningkatkan performa, terutama pada dataset kecil. <hr/> <h3>2. Modifikasi Layer Klasifikasi</h3> <ul style="list-style-type: none">• model.classifier:<ul style="list-style-type: none">○ Layer akhir dimodifikasi untuk 4 kelas:<ul style="list-style-type: none">■ nn.Linear(num_features, 256): Intermediate layer dengan 256 unit.■ nn.ReLU(): Aktivasi ReLU untuk non-linearitas.■ nn.Dropout(0.5): Dropout untuk mencegah overfitting.■ nn.Linear(256, 4): Layer output untuk 4 kelas. <hr/> <h3>3. Loss Function dengan Label Smoothing</h3> <ul style="list-style-type: none">• criterion = nn.CrossEntropyLoss(label_smoothing=0.1):<ul style="list-style-type: none">○ Label Smoothing:<ul style="list-style-type: none">■ Mengurangi keyakinan model yang berlebihan pada satu kelas dengan mendistribusikan probabilitas kecil ke kelas lain.■ Membantu meningkatkan

generalisasi dan mencegah overfitting.

4. Optimizer

- **optimizer = optim.AdamW:**
 - Menggunakan AdamW, yang merupakan varian Adam dengan regularisasi weight decay yang lebih efektif:
 - **lr=0.001:** Learning rate awal.
 - **weight_decay=1e-4:** Regularisasi L2 untuk mencegah overfitting.
-

5. Learning Rate Scheduler

- **scheduler = optim.lr_scheduler.OneCycleLR:**
 - Scheduler ini meningkatkan dan menurunkan learning rate secara kosinusial dalam satu siklus:
 - **max_lr=0.0005:** Learning rate maksimum selama pelatihan.
 - **steps_per_epoch=len(train_loader):** Menyesuaikan siklus dengan jumlah batch per epoch.
 - **epochs=20:** Durasi siklus sesuai dengan jumlah epoch.

```

best_loss = float('inf')
patience = 5
counter = 0
epochs = 20

for epoch in range(epochs):
    model.train()
    train_loss = 0

    all_predictions = []
    all_labels = []

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    # Update scheduler
    scheduler.step()

    # Track predictions and labels for metrics
    _, predicted = torch.max(outputs, 1)
    all_predictions.extend(predicted.cpu().numpy())
    all_labels.extend(labels.cpu().numpy())

    # Calculate metrics
    train_accuracy = (np.array(all_predictions) == np.array(all_labels)).mean() * 100
    train_precision = precision_score(all_labels, all_predictions, average="weighted")
    train_recall = recall_score(all_labels, all_predictions, average="weighted")
    train_f1 = f1_score(all_labels, all_predictions, average="weighted")

    print(f"Epoch [{epoch+1}/{epochs}]")
    print(f"Loss: {train_loss/len(train_loader):.4f} | Accuracy: {train_accuracy:.2f}%")
    print(f"Precision: {train_precision:.4f} | Recall: {train_recall:.4f} | F1 Score: {train_f1:.4f}")

    # Save the best model
    if train_loss < best_loss:
        best_loss = train_loss
        counter = 0
        torch.save(model.state_dict(), "best_model_efficientnet_b3.pth")
    else:
        counter += 1
        if counter >= patience:
            print("Early stopping triggered.")
            break

```

Kode ini melatih model **EfficientNet-B3** menggunakan **OneCycleLR Scheduler** untuk pengaturan learning rate, melacak metrik performa, dan menerapkan mekanisme **early stopping** untuk menghentikan pelatihan jika model tidak membaik.

1. Inisialisasi

- **best_loss = float('inf')**: Menyimpan nilai loss terbaik selama pelatihan.
- **patience = 5**: Jika loss tidak membaik selama 5 epoch berturut-turut, pelatihan dihentikan.
- **counter = 0**: Menghitung jumlah epoch tanpa perbaikan pada loss terbaik.
- **epochs = 20**: Jumlah epoch maksimum.

2. Training Loop

Dilakukan untuk setiap epoch:

a. Mode Pelatihan:

- **model.train()**: Mengaktifkan mode pelatihan dengan dropout dan batch normalization diaktifkan.

b. Forward dan Backward Pass:

- **Forward Pass:**
 - **outputs = model(images)**: Model menghasilkan prediksi logits.
 - **loss = criterion(outputs, labels)**: Menghitung loss menggunakan **CrossEntropyLoss** dengan label smoothing.
- **Backward Pass:**
 - **optimizer.zero_grad()**: Membersihkan gradien sebelumnya.
 - **loss.backward()**: Menghitung gradien loss terhadap parameter model.
 - **optimizer.step()**: Memperbarui parameter model.

c. Scheduler:

- **scheduler.step()**: Menyesuaikan learning rate secara dinamis sesuai dengan siklus **OneCycleLR**.

d. Tracking dan Metrik Evaluasi:

- **Loss**:
 - Total loss batch ditambahkan ke **train_loss**.
 - **Metrik**:
 - **Accuracy, Precision, Recall, dan F1 Score** dihitung menggunakan Scikit-learn berdasarkan semua prediksi dan label.
-

3. Early Stopping

- **Save Best Model**:
 - Jika **train_loss** membaik (lebih kecil dari **best_loss**), model disimpan sebagai **best_model_efficientnet_b3.pth** dan counter direset.
 - **Hentikan Pelatihan**:
 - Jika **train_loss** tidak membaik selama 5 epoch berturut-turut (**counter >= patience**), pelatihan dihentikan lebih awal untuk mencegah overfitting.
-

Output

- Untuk setiap epoch, dicetak:
 - **Loss rata-rata** pada dataset pelatihan.
 - **Accuracy**: Persentase prediksi yang benar.
 - **Precision, Recall, dan F1 Score**: Metrik tambahan untuk mengevaluasi performa klasifikasi.

```

# Load the best model
model.load_state_dict(torch.load("/content/best_model_efficientnet_b3.pth"))
model.eval()

# Test the model
test_loss = 0
all_predictions = []
all_labels = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

        # Track predictions and labels for metrics
        _, predicted = torch.max(outputs, 1)
        all_predictions.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Calculate metrics
test_accuracy = (np.array(all_predictions) == np.array(all_labels)).mean() * 100
test_precision = precision_score(all_labels, all_predictions, average="weighted")
test_recall = recall_score(all_labels, all_predictions, average="weighted")
test_f1 = f1_score(all_labels, all_predictions, average="weighted")

# Print metrics
print(f"Test Loss: {test_loss/len(test_loader):.4f}")
print(f"Test Accuracy: {test_accuracy:.2f}%")
print(f"Test Precision: {test_precision:.4f} | Test Recall: {test_recall:.4f} | Test F1 Score: {test_f1:.4f}")

# Confusion matrix
cm = confusion_matrix(all_labels, all_predictions)
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticks(rotation="vertical"))
plt.title("Confusion Matrix")
plt.show()

```

Kode ini memuat **model terbaik** yang telah dilatih, mengevaluasinya pada dataset pengujian, dan menghitung metrik evaluasi (Loss, Accuracy, Precision, Recall, F1 Score) serta menampilkan **confusion matrix**.

1. Memuat Model Terbaik

- **model.load_state_dict(torch.load("best_model_efficientnet_b3.pth")):**
 - Memuat bobot model terbaik yang disimpan selama pelatihan.
- **model.eval():**
 - Mengatur model ke mode evaluasi, di mana dropout dan batch normalization dinonaktifkan.

2. Evaluasi Dataset Pengujian

- **Tanpa Gradien:**
 - **torch.no_grad()**: Mematikan perhitungan gradien untuk menghemat memori dan mempercepat inferensi.
- **Iterasi pada Test Loader:**
 - Data gambar dan label dipindahkan ke perangkat (GPU/CPU).
 - **Forward Pass:**
 - **outputs = model(images)**: Model menghasilkan prediksi logits.
 - **loss = criterion(outputs, labels)**: Menghitung loss untuk batch tersebut.
 - **Tracking:**
 - Prediksi dihitung dengan **torch.max(outputs, 1)**.
 - Prediksi dan label disimpan untuk evaluasi metrik.

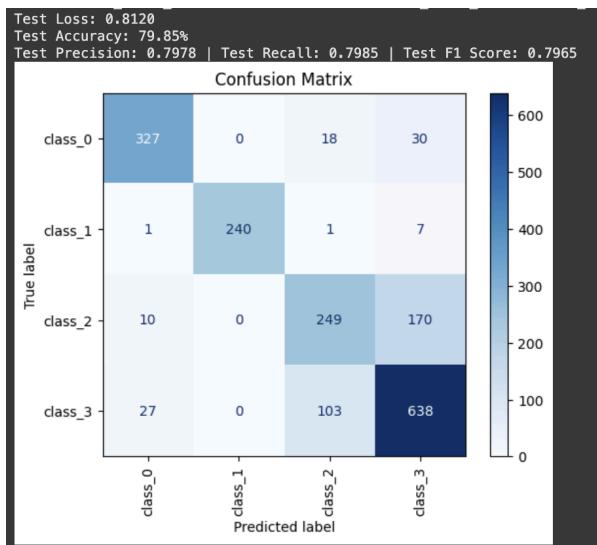
3. Menghitung Metrik

- **Accuracy:**
 - Persentase prediksi yang benar dibandingkan dengan total data pengujian.
- **Precision:**

- Ketepatan prediksi model untuk kelas yang benar.
- **Recall:**
 - Kemampuan model mendeteksi sampel yang benar di setiap kelas.
- **F1 Score:**
 - Harmoni antara Precision dan Recall.
- **Confusion Matrix:**
 - Matriks kesalahan yang menunjukkan distribusi prediksi model untuk setiap kelas.

4. Visualisasi Confusion Matrix

- **ConfusionMatrixDisplay:**
 - Menampilkan **confusion matrix** secara visual dengan pewarnaan (Blues) untuk mempermudah interpretasi hasil.



Performa Model:

- Model menunjukkan performa yang baik dengan akurasi pengujian **79.85%**.
- Metrik Precision, Recall, dan F1 Score menunjukkan konsistensi performa pada semua kelas.