```
!pip install torchinfo
```

```
Collecting torchinfo
  Downloading torchinfo-1.8.0-py3-none-any.whl.metadata (21 kB)
Downloading torchinfo-1.8.0-py3-none-any.whl (23 kB)
Installing collected packages: torchinfo
Successfully installed torchinfo-1.8.0
```

```
# Data handling
import pandas as pd
import numpy as np

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
from PIL import Image

# Preprocessing
from sklearn.model_selection import train_test_split as tts

# Torch
import torch
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
from torchinfo import summary
from torchvision.models import vit_b_16, ViT_B_16_Weights

# Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report

# os
import os

# OrderedDict
from collections import OrderedDict

# tqdm
from tqdm.auto import tqdm

# Path
from pathlib import Path

# random
import random

# typing
from typing import Dict, List
```

```
# warnings
import warnings
warnings.filterwarnings("ignore")
```

```
import kagglehub

# Download versi terbaru dari dataset
path = kagglehub.dataset_download("trainingdatapro/skin-defects-acne-redness-and-bags-under-the-eyes")

print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/trainingdatapro/skin-defects-acne-redness-and-bags-under-the-eyes?dataset_version_number=1...
100%|██████████| 256M/256M [00:14<00:00, 18.4MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/trainingdatapro/skin-defects-acne-redness-and-bags-under-the-eyes/versions/1
```

```
from pathlib import Path

# Tetapkan path dataset
IMAGE_PATH = Path("/root/.cache/kagglehub/datasets/trainingdatapro/skin-defects-acne-redness-and-bags-under-the-eyes/versions/1")

# Membuat list semua gambar
IMAGE_PATH_LIST = list(IMAGE_PATH.glob("*/*/*/*.jpg"))

# Cetak total gambar
print(f'Total Images = {len(IMAGE_PATH_LIST)}')
```

```
Total Images = 87
```

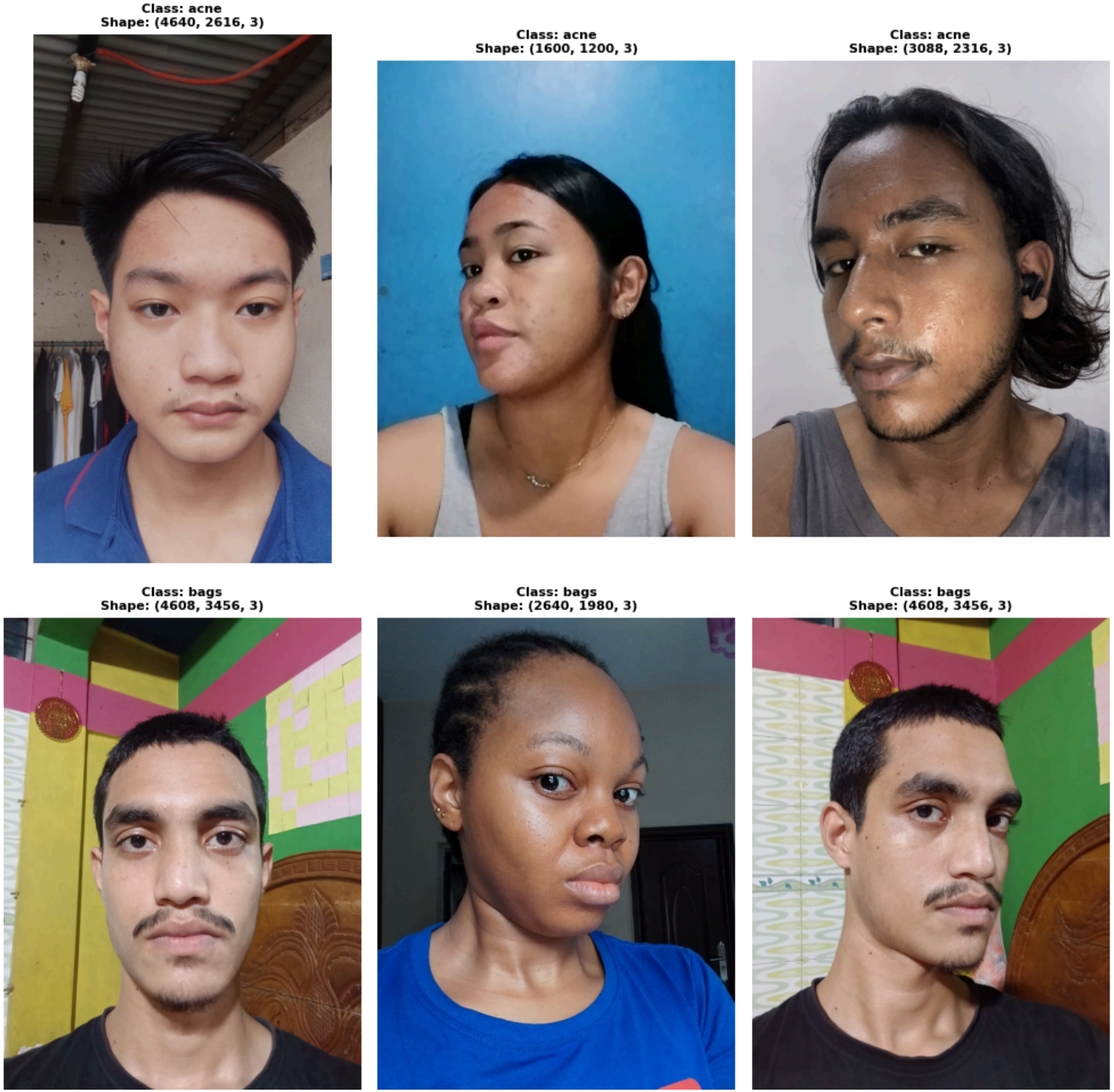
```
from pathlib import Path

# Tetapkan path dataset
IMAGE_PATH = Path("/root/.cache/kagglehub/datasets/trainingdatapro/skin-defects-acne-redness-and-bags-under-the-eyes/versions/1")

# Membuat list semua gambar (menggunakan pola glob rekursif)
IMAGE_PATH_LIST = list(IMAGE_PATH.glob("**/*.jpg"))

# Cetak total gambar
print(f'Total Images = {len(IMAGE_PATH_LIST)}')

# Daftar kelas (kelas berada di subfolder pada level kedua)
classes = sorted([p.name for p in IMAGE_PATH.glob("*/*") if p.is_dir()])

print("**" * 20)
print(f"Total Classes = {len(classes)}")
print("**" * 20)
```

```
# Cetak jumlah gambar per kelas
for c in classes:
    # Kelas mungkin berada lebih dalam, jadi kita sesuaikan path
    class_folders = list(IMAGE_PATH.glob(f"*/{c}"))  # Cari folder di dalam level kedua
    if not class_folders:
        print(f"Class folder for '{c}' not found.")
        continue  # Jika folder kelas tidak ditemukan, lanjut ke kelas berikutnya

    class_path = class_folders[0]  # Ambil folder pertama yang cocok

    # Debug: pastikan kita mencetak path yang benar
    print(f"Class path: {class_path}")

    # Mencari semua gambar di dalam subfolder kelas menggunakan pencarian rekursif (**) jika gambar ada lebih dalam
    total_images_class = list(class_path.glob("**/*.jpg"))

    # Debug: memastikan apakah gambar terbaca atau tidak
    if len(total_images_class) == 0:
        print(f"No images found for class: {c}")
    else:
        print(f"* {c}: {len(total_images_class)} images")
```

```
Total Images = 87
****************************************
Total Classes = 3
****************************************
Class path: /root/.cache/kagglehub/datasets/trainingdatapro/skin-defects-acne-redness-and-bags-under-the-eyes/versions/1/files/acne
* acne: 30 images
Class path: /root/.cache/kagglehub/datasets/trainingdatapro/skin-defects-acne-redness-and-bags-under-the-eyes/versions/1/files/bags
* bags: 27 images
Class path: /root/.cache/kagglehub/datasets/trainingdatapro/skin-defects-acne-redness-and-bags-under-the-eyes/versions/1/files/redness
* redness: 30 images
```

```
import random
import cv2
import matplotlib.pyplot as plt
from pathlib import Path

# Tetapkan jumlah gambar yang ingin dilihat untuk setiap kelas
NUM_IMAGES = 3

# Buat plot dengan subplots sesuai dengan jumlah kelas dan gambar per kelas
fig, ax = plt.subplots(nrows=len(classes), ncols=NUM_IMAGES, figsize=(10, 15))

# Inisialisasi untuk penomoran baris subplot
p = 0

for c in classes:
    # Mencari folder kelas
    class_folders = list(IMAGE_PATH.glob(f"*/{c}"))
```

```
    if not class_folders:
        print(f"Class folder for '{c}' not found.")
        continue  # Jika folder kelas tidak ditemukan, lanjutkan ke kelas berikutnya

    class_path = class_folders[0]  # Gunakan folder pertama yang cocok

    # Mencari semua gambar di folder kelas menggunakan pola glob rekursif
    total_images_class = list(class_path.glob("**/*.jpg"))

    # Jika jumlah gambar lebih sedikit dari NUM_IMAGES, gunakan semua gambar yang tersedia
    if len(total_images_class) < NUM_IMAGES:
        images_selected = total_images_class
    else:
        images_selected = random.sample(total_images_class, k=NUM_IMAGES)

    for i, img_path in enumerate(images_selected):
        # Membaca gambar menggunakan OpenCV
        img_bgr = cv2.imread(str(img_path))

        # Pastikan gambar terbaca dengan benar
        if img_bgr is None:
            print(f"Failed to load image: {img_path}")
            continue

        # Konversi gambar dari BGR ke RGB (karena OpenCV membaca gambar dalam format BGR)
        img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)

        # Menampilkan gambar di subplot
        ax[p, i].imshow(img_rgb)
        ax[p, i].axis("off")
        ax[p, i].set_title(f"Class: {c}\nShape: {img_rgb.shape}", fontsize=8, fontweight="bold", color="black")

    # Naikkan indeks untuk baris subplot berikutnya
    p += 1

# Mengatur tata letak subplot agar tidak tumpang tindih
fig.tight_layout()
plt.show()
```


Class: acne
Shape: (4640, 2616, 3)
Class: acne
Shape: (1600, 1200, 3)
Class: acne
Shape: (3088, 2316, 3)
Class: bags
Shape: (4608, 3456, 3)
Class: bags
Shape: (2640, 1980, 3)
Class: bags
Shape: (4608, 3456, 3)


Class: redness
Shape: (5184, 2916, 3)


Class: redness
Shape: (1300, 1080, 3)


Class: redness
Shape: (1284, 1080, 3)

```
images_path = [None] * len(IMAGE_PATH_LIST)
labels = [None] * len(IMAGE_PATH_LIST)

for i,image_path in enumerate(IMAGE_PATH_LIST):
    images_path[i] = image_path
    labels[i] = image_path.parent.parent.stem

df_path_and_label = pd.DataFrame({'path':images_path,
                                  'label':labels})
df_path_and_label.head()
```

	path	label
0	/root/.cache/kagglehub/datasets/trainingdatapr...	redness
1	/root/.cache/kagglehub/datasets/trainingdatapr...	redness
2	/root/.cache/kagglehub/datasets/trainingdatapr...	redness
3	/root/.cache/kagglehub/datasets/trainingdatapr...	redness
4	/root/.cache/kagglehub/datasets/trainingdatapr...	redness

```
SEED = 123

df_train, df_rest = tts(df_path_and_label,
                        test_size = 0.3,
                        random_state = SEED,
                        stratify = df_path_and_label["label"])

df_val, df_test = tts(df_rest,
                      test_size = 0.5,
                      random_state = SEED,
                      stratify = df_rest["label"])
```

```
# We have to define the mapping of the classes to convert the labels to numbers.
label_map = dict(zip(classes, range(0, len(classes))))
label_map
```

```
{'acne': 0, 'bags': 1, 'redness': 2}
```

```
from torchvision.models import vit_b_16, ViT_B_16_Weights
from torchvision import transforms

# Load pretrained ViT model weights
weights = ViT_B_16_Weights.DEFAULT

# Mengambil transformasi default dari model ViT
auto_transforms = weights.transforms()
```

```
# Cetak transformasi default yang digunakan oleh model ViT
print("Default Transformations from ViT_B_16_Weights:")
print(auto_transforms)

# Jika ingin membuat transformasi secara manual
manual_transforms = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalisasi manual
])

print("\nManual Transformations:")
print(manual_transforms)
```

```
Default Transformations from ViT_B_16_Weights:
ImageClassification(
    crop_size=[224]
    resize_size=[256]
    mean=[0.485, 0.456, 0.406]
    std=[0.229, 0.224, 0.225]
    interpolation=InterpolationMode.BILINEAR
)

Manual Transformations:
Compose(
    Resize(size=256, interpolation=bilinear, max_size=None, antialias=True)
    CenterCrop(size=(224, 224))
    ToTensor()
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
)
```

```
class CustomDataset(Dataset):
    def __init__(self, df:pd.DataFrame, transforms, label_map:dict):
        self.df = df
        self.transforms = transforms
        self.label_map = label_map

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        df_new = self.df.copy()
        df_new = df_new.reset_index(drop = True)
        df_new["label"] = df_new["label"].map(self.label_map)
        image_path = df_new.iloc[idx, 0]
        image = Image.open(image_path).convert("RGB")
        image = self.transforms(image)
        label = df_new.iloc[idx, 1]
```

```
        return image,label
```

```
train_dataset = CustomDataset(df_train, auto_transforms, label_map)
valid_dataset = CustomDataset(df_val, auto_transforms, label_map)
```

```
BATCH_SIZE = 1
NUM_WORKERS = os.cpu_count()

train_dataloader = DataLoader(dataset = train_dataset,
                              batch_size = BATCH_SIZE,
                              shuffle = True,
                              num_workers = NUM_WORKERS)
valid_dataloader = DataLoader(dataset = valid_dataset,
                              batch_size = BATCH_SIZE,
                              shuffle = True,
                              num_workers = NUM_WORKERS)
```

```
# Let's visualize the dimensions of a batch.
batch_images, batch_labels = next(iter(train_dataloader))
batch_images.shape, batch_labels.shape
(torch.Size([1, 3, 224, 224]), torch.Size([1]))
```

```
(torch.Size([1, 3, 224, 224]), torch.Size([1]))
```

```
# GPU
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

```
'cuda'
```

```
# We define the model to use with the pre-trained weights.
model = vit_b_16(weights = weights)
```

```
Downloading: "https://download.pytorch.org/models/vit_b_16-c867db91.pth" to /root/.cache/torch/hub/checkpoints/vit_b_16-c867db91.pth
100%|██████████| 330M/330M [00:01<00:00, 180MB/s]
```

```
# Let's visualize the architecture of the model.
summary(model = model,
        input_size = [1, 3, 224, 224],
        col_names = ["input_size", "output_size", "num_params", "trainable"],
        col_width = 15,
        row_settings = ["var_names"])
```

```
==========================================================================================================================================
Layer (type (var_name))                                  Input Shape     Output Shape    Param #         Trainable
==========================================================================================================================================
```

```
VisionTransformer (VisionTransformer)                   [1, 3, 224, 224] [1, 1000]       768            True
├─Conv2d (conv_proj)                                    [1, 3, 224, 224] [1, 768, 14, 14] 590,592        True
├─Encoder (encoder)                                     [1, 197, 768]   [1, 197, 768]   151,296        True
│    └─Dropout (dropout)                                [1, 197, 768]   [1, 197, 768]   --             --
│    └─Sequential (layers)                              [1, 197, 768]   [1, 197, 768]   --             True
│    │    └─EncoderBlock (encoder_layer_0)              [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    │    └─EncoderBlock (encoder_layer_1)              [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    │    └─EncoderBlock (encoder_layer_2)              [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    │    └─EncoderBlock (encoder_layer_3)              [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    │    └─EncoderBlock (encoder_layer_4)              [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    │    └─EncoderBlock (encoder_layer_5)              [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    │    └─EncoderBlock (encoder_layer_6)              [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    │    └─EncoderBlock (encoder_layer_7)              [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    │    └─EncoderBlock (encoder_layer_8)              [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    │    └─EncoderBlock (encoder_layer_9)              [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    │    └─EncoderBlock (encoder_layer_10)             [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    │    └─EncoderBlock (encoder_layer_11)             [1, 197, 768]   [1, 197, 768]   7,087,872      True
│    └─LayerNorm (ln)                                   [1, 197, 768]   [1, 197, 768]   1,536          True
├─Sequential (heads)                                    [1, 768]        [1, 1000]       --             True
│    └─Linear (head)                                    [1, 768]        [1, 1000]       769,000        True
==========================================================================================================================================
Total params: 86,567,656
Trainable params: 86,567,656
Non-trainable params: 0
Total mult-adds (M): 173.23
==========================================================================================================================================
Input size (MB): 0.60
Forward/backward pass size (MB): 104.09
Params size (MB): 232.27
Estimated Total Size (MB): 336.96
==========================================================================================================================================
```

```
for param in model.conv_proj.parameters():
    param.requires_grad = False
```

```
for param in model.encoder.parameters():
    param.requires_grad = False
```

```
# Let's see if the parameters were frozen.
summary(model = model,
        input_size = [1,3,224,224],
        col_names = ["input_size", "output_size", "num_params", "trainable"],
        col_width = 15,
        row_settings = ["var_names"])
```

```
==========================================================================================================================================
Layer (type (var_name))                                 Input Shape     Output Shape    Param #         Trainable
==========================================================================================================================================
VisionTransformer (VisionTransformer)                   [1, 3, 224, 224] [1, 1000]       768            Partial
├─Conv2d (conv_proj)                                    [1, 3, 224, 224] [1, 768, 14, 14] (590,592)      False
├─Encoder (encoder)                                     [1, 197, 768]   [1, 197, 768]   151,296        False
│    └─Dropout (dropout)                                [1, 197, 768]   [1, 197, 768]   --             --
```

```
│    └─Sequential (layers)                                [1, 197, 768]   [1, 197, 768]   --              False
│    │    └─EncoderBlock (encoder_layer_0)                [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_1)                [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_2)                [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_3)                [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_4)                [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_5)                [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_6)                [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_7)                [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_8)                [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_9)                [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_10)               [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_11)               [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    └─LayerNorm (ln)                                     [1, 197, 768]   [1, 197, 768]   (1,536)         False
├─Sequential (heads)                                      [1, 768]        [1, 1000]       --              True
│    └─Linear (head)                                      [1, 768]        [1, 1000]       769,000         True
==========================================================================================================================================
Total params: 86,567,656
Trainable params: 769,768
Non-trainable params: 85,797,888
Total mult-adds (M): 173.23
==========================================================================================================================================
Input size (MB): 0.60
Forward/backward pass size (MB): 104.09
Params size (MB): 232.27
Estimated Total Size (MB): 336.96
==========================================================================================================================================
```

```
output_shape = len(classes)

model.heads = nn.Sequential(OrderedDict([('head', nn.Linear(in_features = 768,
                                                            out_features = output_shape))]))
```

```
# One last time let's take a look if the last layer was modified.
summary(model = model,
        input_size = [1,3,224,224],
        col_names = ["input_size", "output_size", "num_params", "trainable"],
        col_width = 15,
        row_settings = ["var_names"])
```

```
==========================================================================================================================================
Layer (type (var_name))                                 Input Shape     Output Shape    Param #         Trainable
==========================================================================================================================================
VisionTransformer (VisionTransformer)                   [1, 3, 224, 224] [1, 3]          768             Partial
├─Conv2d (conv_proj)                                    [1, 3, 224, 224] [1, 768, 14, 14] (590,592)       False
├─Encoder (encoder)                                     [1, 197, 768]   [1, 197, 768]   151,296         False
│    └─Dropout (dropout)                                [1, 197, 768]   [1, 197, 768]   --              --
│    └─Sequential (layers)                              [1, 197, 768]   [1, 197, 768]   --              False
│    │    └─EncoderBlock (encoder_layer_0)              [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_1)              [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_2)              [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_3)              [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_4)              [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
```

```
│    │    └─EncoderBlock (encoder_layer_5)                  [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_6)                  [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_7)                  [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_8)                  [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_9)                  [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_10)                 [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    │    └─EncoderBlock (encoder_layer_11)                 [1, 197, 768]   [1, 197, 768]   (7,087,872)     False
│    └─LayerNorm (ln)                                       [1, 197, 768]   [1, 197, 768]   (1,536)         False
├─Sequential (heads)                                        [1, 768]        [1, 3]          --              True
│    └─Linear (head)                                        [1, 768]        [1, 3]          2,307           True
==========================================================================================================================================
Total params: 85,800,963
Trainable params: 3,075
Non-trainable params: 85,797,888
Total mult-adds (M): 172.47
==========================================================================================================================================
Input size (MB): 0.60
Forward/backward pass size (MB): 104.09
Params size (MB): 229.20
Estimated Total Size (MB): 333.89
==========================================================================================================================================
```

```
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr = 0.01)
```

```
def train_step(model:torch.nn.Module,
               dataloader:torch.utils.data.DataLoader,
               loss_fn:torch.nn.Module,
               optimizer:torch.optim.Optimizer):

    model.train()

    train_loss = 0.
    train_accuracy = 0.

    for batch,(X,y) in enumerate(dataloader):
        X,y = X.to(device), y.to(device)
        optimizer.zero_grad()
        y_pred_logit = model(X)
        loss = loss_fn(y_pred_logit, y)
        train_loss += loss.item()

        loss.backward()
        optimizer.step()

        y_pred_prob = torch.softmax(y_pred_logit, dim = 1)
        y_pred_class = torch.argmax(y_pred_prob, dim = 1)
        train_accuracy += accuracy_score(y.cpu().numpy(),
                                         y_pred_class.detach().cpu().numpy())

    train_loss = train_loss/len(dataloader)
```

```
    train_accuracy = train_accuracy/len(dataloader)

    return train_loss, train_accuracy
```

```
def save_checkpoint(filename, model, loss, epoch, optimizer, metric):
    state = {"filename":filename,
             "model":model.state_dict(),
             "loss":loss,
             "epoch":epoch,
             "optimizer":optimizer.state_dict(),
             "metric":metric}

    torch.save(state, filename)
```

```
def valid_step(model:torch.nn.Module,
               dataloader:torch.utils.data.DataLoader,
               loss_fn:torch.nn.Module):

    model.eval()

    valid_loss = 0.
    valid_accuracy = 0.

    with torch.inference_mode():
        for batch,(X,y) in enumerate(dataloader):
            X,y = X.to(device), y.to(device)
            y_pred_logit = model(X)
            loss = loss_fn(y_pred_logit, y)
            valid_loss += loss.item()

            y_pred_prob = torch.softmax(y_pred_logit, dim = 1)
            y_pred_class = torch.argmax(y_pred_prob, dim = 1)

            valid_accuracy += accuracy_score(y.cpu().numpy(), y_pred_class.detach().cpu().numpy())

    valid_loss = valid_loss/len(dataloader)
    valid_accuracy = valid_accuracy/len(dataloader)

    return valid_loss, valid_accuracy
```

```
def train(model:torch.nn.Module,
          train_dataloader:torch.utils.data.DataLoader,
          valid_dataloader:torch.utils.data.DataLoader,
          loss_fn:torch.nn.Module,
          optimizer:torch.optim.Optimizer,
          epochs:int = 10):

    results = {"train_loss":[],
               "train_accuracy":[],
```

```
               "valid_loss":[],
               "valid_accuracy":[]}

    best_valid_loss = float("inf")

    for epoch in tqdm(range(epochs)):
        train_loss, train_accuracy = train_step(model = model,
                                                dataloader = train_dataloader,
                                                loss_fn = loss_fn,
                                                optimizer = optimizer)

        valid_loss, valid_accuracy = valid_step(model = model,
                                                dataloader = valid_dataloader,
                                                loss_fn = loss_fn)

        if valid_loss < best_valid_loss:
            best_valid_loss = valid_loss
            file_name = "best_model.pth"
            save_checkpoint(file_name, model, best_valid_loss, epoch, optimizer, valid_accuracy)

        print(f"Epoch: {epoch + 1} | ",
              f"Train Loss: {train_loss:.4f} | ",
              f"Train Accuracy: {train_accuracy:.4f} | ",
              f"Valid Loss: {valid_loss:.4f} | ",
              f"Valid Accuracy: {valid_accuracy:.4f}")

        results["train_loss"].append(train_loss)
        results["train_accuracy"].append(train_accuracy)
        results["valid_loss"].append(valid_loss)
        results["valid_accuracy"].append(valid_accuracy)

    return results
```

```
# Training!!!
EPOCHS = 100

torch.cuda.manual_seed(SEED)
torch.manual_seed(SEED)

MODEL_RESULTS = train(model.to(device),
                      train_dataloader,
                      valid_dataloader,
                      loss_fn,
                      optimizer,
                      EPOCHS)
```

100% 100/100 [16:41<00:00, 10.06s/it]

```
Epoch: 1 |  Train Loss: 2.1804 |  Train Accuracy: 0.4333 |  Valid Loss: 1.6055 |  Valid Accuracy: 0.3846
Epoch: 2 |  Train Loss: 2.3547 |  Train Accuracy: 0.5333 |  Valid Loss: 0.7802 |  Valid Accuracy: 0.7692
Epoch: 3 |  Train Loss: 1.2370 |  Train Accuracy: 0.7000 |  Valid Loss: 0.8841 |  Valid Accuracy: 0.6154
Epoch: 4 |  Train Loss: 0.8212 |  Train Accuracy: 0.7000 |  Valid Loss: 3.2707 |  Valid Accuracy: 0.4615
Epoch: 5 |  Train Loss: 0.3131 |  Train Accuracy: 0.9000 |  Valid Loss: 1.1169 |  Valid Accuracy: 0.6923
Epoch: 6 |  Train Loss: 0.0977 |  Train Accuracy: 0.9500 |  Valid Loss: 0.9242 |  Valid Accuracy: 0.6923
Epoch: 7 |  Train Loss: 0.0741 |  Train Accuracy: 0.9667 |  Valid Loss: 0.2504 |  Valid Accuracy: 0.8462
Epoch: 8 |  Train Loss: 0.1026 |  Train Accuracy: 0.9500 |  Valid Loss: 0.3934 |  Valid Accuracy: 0.9231
Epoch: 9 |  Train Loss: 0.2444 |  Train Accuracy: 0.8833 |  Valid Loss: 2.3341 |  Valid Accuracy: 0.6923
Epoch: 10 |  Train Loss: 0.5818 |  Train Accuracy: 0.9333 |  Valid Loss: 1.1219 |  Valid Accuracy: 0.6923
Epoch: 11 |  Train Loss: 0.0020 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4576 |  Valid Accuracy: 0.9231
Epoch: 12 |  Train Loss: 0.0004 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4627 |  Valid Accuracy: 0.9231
Epoch: 13 |  Train Loss: 0.0004 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4605 |  Valid Accuracy: 0.9231
Epoch: 14 |  Train Loss: 0.0004 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4656 |  Valid Accuracy: 0.9231
Epoch: 15 |  Train Loss: 0.0004 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4640 |  Valid Accuracy: 0.9231
Epoch: 16 |  Train Loss: 0.0003 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4633 |  Valid Accuracy: 0.9231
Epoch: 17 |  Train Loss: 0.0003 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4617 |  Valid Accuracy: 0.9231
Epoch: 18 |  Train Loss: 0.0003 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4618 |  Valid Accuracy: 0.9231
Epoch: 19 |  Train Loss: 0.0003 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4613 |  Valid Accuracy: 0.9231
Epoch: 20 |  Train Loss: 0.0003 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4604 |  Valid Accuracy: 0.9231
Epoch: 21 |  Train Loss: 0.0003 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4576 |  Valid Accuracy: 0.9231
Epoch: 22 |  Train Loss: 0.0003 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4579 |  Valid Accuracy: 0.9231
Epoch: 23 |  Train Loss: 0.0003 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4590 |  Valid Accuracy: 0.9231
Epoch: 24 |  Train Loss: 0.0003 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4571 |  Valid Accuracy: 0.9231
Epoch: 25 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4568 |  Valid Accuracy: 0.9231
Epoch: 26 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4575 |  Valid Accuracy: 0.9231
Epoch: 27 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4588 |  Valid Accuracy: 0.9231
Epoch: 28 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4542 |  Valid Accuracy: 0.9231
Epoch: 29 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4539 |  Valid Accuracy: 0.9231
Epoch: 30 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4512 |  Valid Accuracy: 0.9231
Epoch: 31 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4549 |  Valid Accuracy: 0.9231
Epoch: 32 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4509 |  Valid Accuracy: 0.9231
Epoch: 33 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4553 |  Valid Accuracy: 0.9231
Epoch: 34 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4506 |  Valid Accuracy: 0.9231
Epoch: 35 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4490 |  Valid Accuracy: 0.9231
Epoch: 36 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4521 |  Valid Accuracy: 0.9231
Epoch: 37 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4494 |  Valid Accuracy: 0.9231
Epoch: 38 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4512 |  Valid Accuracy: 0.9231
Epoch: 39 |  Train Loss: 0.0002 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4494 |  Valid Accuracy: 0.9231
Epoch: 40 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4491 |  Valid Accuracy: 0.9231
Epoch: 41 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4476 |  Valid Accuracy: 0.9231
Epoch: 42 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4488 |  Valid Accuracy: 0.9231
Epoch: 43 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4518 |  Valid Accuracy: 0.9231
Epoch: 44 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4493 |  Valid Accuracy: 0.9231
Epoch: 45 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4478 |  Valid Accuracy: 0.9231
Epoch: 46 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4476 |  Valid Accuracy: 0.9231
Epoch: 47 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4522 |  Valid Accuracy: 0.9231
Epoch: 48 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4477 |  Valid Accuracy: 0.9231
Epoch: 49 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4504 |  Valid Accuracy: 0.9231
Epoch: 50 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4503 |  Valid Accuracy: 0.9231
Epoch: 51 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4570 |  Valid Accuracy: 0.9231
Epoch: 52 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4443 |  Valid Accuracy: 0.9231
Epoch: 53 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4515 |  Valid Accuracy: 0.9231
Epoch: 54 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4445 |  Valid Accuracy: 0.9231
```

```
Epoch: 55 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4450 |  Valid Accuracy: 0.9231
Epoch: 56 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4480 |  Valid Accuracy: 0.9231
Epoch: 57 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4473 |  Valid Accuracy: 0.9231
Epoch: 58 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4502 |  Valid Accuracy: 0.9231
Epoch: 59 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4451 |  Valid Accuracy: 0.9231
Epoch: 60 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4523 |  Valid Accuracy: 0.9231
Epoch: 61 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4476 |  Valid Accuracy: 0.9231
Epoch: 62 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4471 |  Valid Accuracy: 0.9231
Epoch: 63 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4469 |  Valid Accuracy: 0.9231
Epoch: 64 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4458 |  Valid Accuracy: 0.9231
Epoch: 65 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4523 |  Valid Accuracy: 0.9231
Epoch: 66 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4439 |  Valid Accuracy: 0.9231
Epoch: 67 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4502 |  Valid Accuracy: 0.9231
Epoch: 68 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4469 |  Valid Accuracy: 0.9231
Epoch: 69 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4486 |  Valid Accuracy: 0.9231
Epoch: 70 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4445 |  Valid Accuracy: 0.9231
Epoch: 71 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4481 |  Valid Accuracy: 0.9231
Epoch: 72 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4472 |  Valid Accuracy: 0.9231
Epoch: 73 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4490 |  Valid Accuracy: 0.9231
Epoch: 74 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4469 |  Valid Accuracy: 0.9231
Epoch: 75 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4499 |  Valid Accuracy: 0.9231
Epoch: 76 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4532 |  Valid Accuracy: 0.9231
Epoch: 77 |  Train Loss: 0.0001 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4482 |  Valid Accuracy: 0.9231
Epoch: 78 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4475 |  Valid Accuracy: 0.9231
Epoch: 79 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4480 |  Valid Accuracy: 0.9231
Epoch: 80 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4485 |  Valid Accuracy: 0.9231
Epoch: 81 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4507 |  Valid Accuracy: 0.9231
Epoch: 82 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4485 |  Valid Accuracy: 0.9231
Epoch: 83 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4501 |  Valid Accuracy: 0.9231
Epoch: 84 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4478 |  Valid Accuracy: 0.9231
Epoch: 85 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4457 |  Valid Accuracy: 0.9231
Epoch: 86 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4534 |  Valid Accuracy: 0.9231
Epoch: 87 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4509 |  Valid Accuracy: 0.9231
Epoch: 88 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4487 |  Valid Accuracy: 0.9231
Epoch: 89 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4524 |  Valid Accuracy: 0.9231
Epoch: 90 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4446 |  Valid Accuracy: 0.9231
Epoch: 91 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4474 |  Valid Accuracy: 0.9231
Epoch: 92 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4468 |  Valid Accuracy: 0.9231
Epoch: 93 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4492 |  Valid Accuracy: 0.9231
Epoch: 94 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4503 |  Valid Accuracy: 0.9231
Epoch: 95 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4455 |  Valid Accuracy: 0.9231
Epoch: 96 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4473 |  Valid Accuracy: 0.9231
Epoch: 97 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4499 |  Valid Accuracy: 0.9231
Epoch: 98 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4516 |  Valid Accuracy: 0.9231
Epoch: 99 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4546 |  Valid Accuracy: 0.9231
Epoch: 100 |  Train Loss: 0.0000 |  Train Accuracy: 1.0000 |  Valid Loss: 0.4478 |  Valid Accuracy: 0.9231
```

```
# Function to plot the loss and metric during each training epoch.
def loss_metric_curve_plot(model_results:Dict[str,List[float]]):

    train_loss = model_results["train_loss"]
    valid_loss = model_results["valid_loss"]

    train_accuracy = [float(value) for value in model_results["train_accuracy"]]
    valid_accuracy = [float(value) for value in model_results["valid_accuracy"]]

    fig,axes = plt.subplots(nrows = 1, ncols = 2, figsize = (10,4))
    axes = axes.flat

    axes[0].plot(train_loss, color = "red", label = "Train")
    axes[0].plot(valid_loss, color = "blue", label = "Valid")
    axes[0].set_title("CrossEntropyLoss", fontsize = 12, fontweight = "bold", color = "black")
    axes[0].set_xlabel("Epochs", fontsize = 10, fontweight = "bold", color = "black")
    axes[0].set_ylabel("Loss", fontsize = 10, fontweight = "bold", color = "black")
    axes[0].legend()

    axes[1].plot(train_accuracy, color = "red", label = "Train")
    axes[1].plot(valid_accuracy, color = "blue", label = "Valid")
    axes[1].set_title("Metric of performance: Accuracy", fontsize = 12, fontweight = "bold", color = "black")
    axes[1].set_xlabel("Epochs", fontsize = 10, fontweight = "bold", color = "black")
    axes[1].set_ylabel("Score", fontsize = 10, fontweight = "bold", color = "black")
    axes[1].legend()

    fig.tight_layout()
    fig.show()

    # Mengembalikan akurasi pelatihan dan validasi pada epoch terakhir
    final_train_accuracy = train_accuracy[-1] * 100
    final_valid_accuracy = valid_accuracy[-1] * 100
    print(f"Final Train Accuracy: {final_train_accuracy:.2f}%")
    print(f"Final Validation Accuracy: {final_valid_accuracy:.2f}%")

    return final_train_accuracy, final_valid_accuracy
```

```
loss_metric_curve_plot(MODEL_RESULTS)
```