

## SELURUH ANALISA KODE PROGRAM PEMBENTUKAN MODEL DATA MINING

### Analisis Kode Program Pembentukan Model

Kode yang saya buat ini melanjutkan alur kerja machine learning dari tahap pra-pemrosesan ke tahap pemodelan. Tujuannya adalah untuk melatih model klasifikasi *deep learning* untuk membedakan antara gambar otak dengan tumor dan otak yang sehat. Arsitektur yang dipilih adalah Vision Transformer (ViT), sebuah model canggih yang menggunakan mekanisme *attention* yang awalnya dikembangkan untuk pemrosesan bahasa.

#### Blok 1: Impor Pustaka dan Koneksi Google Drive

- **Tujuan:** Blok ini berfungsi sebagai fondasi dengan menyiapkan lingkungan kerja. Semua pustaka yang diperlukan untuk manipulasi data, pemrosesan gambar, pembangunan model, dan evaluasi diimporkan.
- **Analisis Rinci:**
  - **Koneksi Drive:** from google.colab import drive dan drive.mount sangat penting untuk bekerja di Google Colab, memungkinkan akses langsung ke file dataset yang tersimpan di Google Drive.
  - **Instalasi Pustaka:** !pip install split-folders timm torch-summary menginstal pustaka kunci:
    - **split-folders:** Alat bantu praktis untuk membagi dataset gambar ke dalam folder train, val, dan test.
    - **timm (PyTorch Image Models):** Pustaka yang sangat populer untuk mengakses ratusan arsitektur model *pre-trained*, termasuk ViT.
    - **torch-summary:** Untuk menampilkan ringkasan arsitektur model yang mudah dibaca.
  - **Pustaka Inti:** Impor torch, torchvision, pandas, numpy, dan matplotlib adalah standar untuk proyek *deep learning* dengan PyTorch. Mereka menangani tensor, pembuatan model, pemrosesan data, operasi numerik, dan visualisasi.

#### Blok 2: Membaca dan Menampilkan Metadata

- **Tujuan:** Memuat dan memeriksa file metadata.csv untuk mendapatkan pemahaman awal tentang data.
- **Analisis Rinci:**
  - **pd.read\_csv:** digunakan untuk memuat data tabular ke dalam DataFrame pandas, yang ideal untuk analisis data.
  - **labels\_df.head():** adalah langkah EDA (Exploratory Data Analysis) sederhana untuk melihat beberapa baris pertama dan memahami struktur kolom.
  - **os.listdir:** digunakan untuk memverifikasi isi direktori dataset, memastikan bahwa path yang digunakan benar.

### Blok 3: Pembagian Dataset dan Definisi Transformasi

- **Tujuan:** Mempersiapkan data gambar sebelum dimasukkan ke model. Ini adalah salah satu langkah paling krusial dalam pemodelan gambar.
- **Analisis Rinci:**
  - **Pembagian Dataset:** Baris `splitfolders.ratio(...)` (yang dikomentari) adalah langkah penting untuk membagi data secara proporsional menjadi set pelatihan (80%) dan validasi (20%). Ini memastikan model dievaluasi pada data yang belum pernah dilihatnya selama pelatihan.
  - **Data Augmentasi (train\_transform):** Ini adalah teknik untuk memperbanyak data pelatihan secara artifisial.
    - **RandomHorizontalFlip, RandomVerticalFlip, RandomRotation, ColorJitter:** Operasi ini membuat variasi baru dari gambar yang ada, membantu model menjadi lebih robust (tangguh) dan tidak terlalu overfitting terhadap data pelatihan.
    - **Resize(224, 224):** Menyeragamkan ukuran semua gambar agar sesuai dengan input yang diharapkan oleh model ViT.
    - **ToTensor():** Mengonversi gambar dari format PIL/Numpy ke format Tensor PyTorch.
    - **Normalize(...):** Menyesuaikan rentang nilai piksel gambar menggunakan *mean* dan *standard deviation* dari dataset ImageNet. Ini adalah praktik standar saat menggunakan model *pre-trained*.
  - **Transformasi Validasi (val\_transform):** Set validasi hanya melalui transformasi esensial (resize, to tensor, normalize) tanpa augmentasi acak. Ini penting untuk mendapatkan evaluasi yang konsisten dan tidak bias terhadap kinerja model.

### Blok 4: Membuat Dataset dan Menampilkan Transformasi

- **Tujuan:** Memuat data gambar dari folder menggunakan transformasi yang telah didefinisikan.
- **Analisis Rinci:**
  - **torchvision.datasets.ImageFolder:** Kelas ini sangat berguna karena secara otomatis memuat gambar dan menetapkan label berdasarkan nama folder tempat gambar tersebut berada.
  - **Kode ini membuat dua objek Dataset terpisah:** `train_set` dengan augmentasi dan `val_set` tanpa augmentasi. Ini adalah praktik yang benar.

## Blok 5 & 6: Visualisasi Efek Augmentasi Data

- **Tujuan:** Melakukan sanity check dengan memvisualisasikan hasil dari setiap langkah augmentasi.
- **Analisis Rinci:**
  - Ini adalah langkah EDA yang sangat baik. Dengan menampilkan gambar "Original vs Transformed", Anda dapat memastikan bahwa setiap transformasi bekerja seperti yang diharapkan.
  - Looping melalui `transforms_list` dan menerapkannya satu per satu pada gambar acak memberikan pemahaman visual yang jelas tentang apa yang akan "dilihat" oleh model selama pelatihan.
  - Logika untuk denormalisasi (`if name == 'Normalize': ...`) sangat penting. Gambar yang dinormalisasi tidak dapat ditampilkan dengan benar tanpa mengembalikannya ke rentang [0, 1].

## Blok 7 - 11: Analisis Data Eksploratif (EDA) Lanjutan

- **Tujuan:** Mendapatkan wawasan yang lebih dalam tentang karakteristik dataset sebelum melatih model.
- **Analisis Rinci:**
  - **Blok 7 (Jumlah Data per Kelas):** Menggunakan Counter untuk memeriksa apakah dataset seimbang (balanced). Ketidakseimbangan kelas dapat memengaruhi kinerja model, dan jika parah, mungkin memerlukan teknik penanganan khusus (misalnya, *weighted loss* atau *oversampling*).
  - **Blok 8 (Contoh Gambar per Kelas):** Memberikan konfirmasi visual bahwa data telah dilabeli dengan benar.
  - **Blok 9 (Analisis Dimensi):** Menunjukkan bahwa gambar dalam dataset memiliki ukuran yang bervariasi. Ini menegaskan pentingnya langkah `transforms.Resize` di Blok 3 untuk menyeragamkan input.
  - **Blok 10 (Distribusi Piksel):** Histogram nilai piksel memberikan gambaran tentang kontras dan kecerahan umum dalam dataset.
  - **Blok 11 (Analisis Channel Warna):** Memastikan bahwa sebagian besar gambar memiliki 3 channel (RGB), yang merupakan format standar untuk model *pre-trained* seperti ViT.

## Blok 12: Pengaturan Hyperparameter dan Persiapan Data untuk Model ViT

- **Tujuan:** Menetapkan semua parameter konfigurasi untuk proses pelatihan dan membuat DataLoader.
- **Analisis Rinci:**
  - **Hyperparameters:** Ini adalah "tombol-tombol" yang dapat disetel untuk mengontrol proses belajar model. Nilai-nilai seperti learning\_rate, batch\_size, dan num\_epochs sangat memengaruhi hasil akhir. Parameter spesifik ViT (patch\_size, embed\_dim, dll.) juga didefinisikan di sini.
  - **DataLoader:** Objek DataLoader membungkus Dataset. Fungsinya adalah untuk menyajikan data ke model dalam batch (kelompok kecil), yang lebih efisien secara komputasi.
    - **shuffle=True** untuk train\_loader sangat penting untuk mengurangi korelasi antar-batch dan membantu model belajar lebih baik.
    - **shuffle=False** untuk val\_loader memastikan urutan evaluasi selalu sama.

## Blok 13: Membangun dan Melatih Model ViT

- **Tujuan:** Ini adalah blok inti dari keseluruhan kode, di mana model didefinisikan, dilatih, dan divalidasi.
- **Analisis Rinci:**
  - **Pembuatan Model:**
    - **device :** Mengatur agar pelatihan berjalan di GPU (cuda) jika tersedia, yang secara dramatis mempercepat proses dibandingkan CPU.
    - **create\_model('vit\_base\_patch16\_224', pretrained=True, ...):** Ini adalah perintah kunci.
      - **'vit\_base\_patch16\_224':** Memilih arsitektur Vision Transformer (Base).
      - **pretrained=True:** Menggunakan teknik Transfer Learning. Model ini sudah dilatih pada dataset raksasa (ImageNet), sehingga ia sudah memiliki pengetahuan dasar tentang fitur-fitur visual. Kita hanya perlu "menyesuaikannya" (fine-tuning) untuk tugas spesifik klasifikasi tumor otak.
      - **num\_classes=num\_classes:** Mengganti lapisan klasifikasi akhir dari model asli (1000 kelas ImageNet) menjadi 2 kelas (Tumor, Sehat).
  - **Komponen Pelatihan:**
    - **criterion = nn.CrossEntropyLoss():** Fungsi loss yang umum dan efektif untuk masalah klasifikasi multi-kelas.

- **optimizer = optim.AdamW(...)**: Algoritma optimisasi modern yang bertugas memperbarui bobot model untuk meminimalkan loss.
- **scheduler = ReduceLROnPlateau(...)**: Mekanisme cerdas yang secara otomatis mengurangi *learning rate* jika kinerja model di set validasi tidak membaik setelah beberapa *epoch*. Ini membantu model "menyetel" dirinya lebih halus di tahap akhir pelatihan.

- **Training Loop:**

- **vit\_model.train()**: Mengaktifkan mode training (misalnya, mengaktifkan lapisan dropout).
- **optimizer.zero\_grad()**: Mengosongkan gradien dari iterasi sebelumnya.
- **loss.backward()**: Menghitung gradien (turunan) dari loss terhadap setiap parameter model (propagasi mundur).
- **optimizer.step()**: Memperbarui parameter model berdasarkan gradien yang dihitung.
- **vit\_model.eval()**: Mengalihkan model ke mode evaluasi (misalnya, menonaktifkan dropout) untuk hasil yang konsisten pada data validasi.
- **with torch.no\_grad()**: Menonaktifkan perhitungan gradien selama validasi untuk menghemat memori dan komputasi.

#### Blok 14: Visualisasi Loss Pelatihan

- **Tujuan:** Menganalisis kinerja pelatihan model secara visual dengan memplot kurva *loss* (kerugian) dari waktu ke waktu. Ini adalah alat diagnostik utama untuk memahami bagaimana proses belajar berlangsung.
- **Analisis Rinci:**
  - Kode ini menggunakan matplotlib untuk membuat grafik garis dari variabel train\_losses dan val\_losses yang telah dikumpulkan pada setiap epoch di Blok 13.
  - **Training Loss (Loss Pelatihan):** Menunjukkan seberapa baik model mencocokkan data pelatihan. Idealnya, kurva ini akan terus menurun.
  - **Validation Loss (Loss Validasi):** Menunjukkan seberapa baik model dapat menggeneralisasi pengetahuannya pada data baru yang belum pernah dilihat sebelumnya.
- **Interpretasi Kunci:**
  - **Baik:** Kedua kurva (training dan validasi) sama-sama menurun secara konsisten. Ini menandakan model belajar dengan baik.
  - **Overfitting:** Kurva *training loss* terus menurun, tetapi kurva *validation loss* mulai mendatar atau bahkan naik. Ini adalah tanda bahaya  yang berarti model terlalu "menghafal" data pelatihan dan tidak akan berkinerja baik pada data baru.

## Blok 15: Evaluasi Model dengan Confusion Matrix

- **Tujuan:** Mengevaluasi performa klasifikasi model secara kuantitatif pada data validasi menggunakan metrik standar dan visualisasi yang jelas.
- **Analisis Rinci:**
  - **vit\_model.eval():** Mengubah model ke mode evaluasi. Ini penting karena menonaktifkan lapisan seperti *dropout*, memastikan hasil evaluasi konsisten.
  - **Pengumpulan Prediksi:** Kode ini melakukan iterasi melalui *val\_loader*, mendapatkan prediksi dari model (*torch.max(outputs, 1)*), dan menyimpannya bersama dengan label asli (*y\_true* dan *y\_pred*).
  - **Confusion Matrix:**
    - Ini adalah tabel yang memvisualisasikan performa model dengan membandingkan label asli dengan prediksi. Sangat berguna untuk melihat jenis kesalahan apa yang dibuat model.
    - *sns.heatmap* dari pustaka Seaborn digunakan untuk membuat visualisasi yang berwarna dan mudah dibaca. Angka pada diagonal utama menunjukkan prediksi yang benar.
  - **Classification Report:**
    - **Precision:** Dari semua yang diprediksi sebagai "Tumor", berapa persen yang benar-benar "Tumor"? (Akurasi prediksi positif).
    - **Recall (Sensitivity):** Dari semua kasus "Tumor" yang sebenarnya, berapa persen yang berhasil dideteksi oleh model? (Tingkat deteksi).
    - **F1-Score:** Rata-rata harmonik dari *precision* dan *recall*, memberikan skor tunggal yang menyeimbangkan keduanya.

## Blok 16: Evaluasi Lengkap dengan ROC Curve

- **Tujuan:** Melakukan evaluasi yang lebih mendalam dengan menganalisis kemampuan diskriminatif model (kemampuannya memisahkan antara kelas positif dan negatif) di semua ambang batas klasifikasi.
- **Analisis Rinci:**
  - **Probabilitas:** Langkah kunci di sini adalah penggunaan *torch.softmax(outputs, dim=1)* untuk mengubah *logits* mentah dari model menjadi probabilitas (nilai antara 0 dan 1).
  - **Kurva ROC (Receiver Operating Characteristic):**
    - Ini adalah grafik yang memplot *True Positive Rate* (Recall) melawan *False Positive Rate* di berbagai ambang batas.
    - Kurva yang menempel di pojok kiri atas menunjukkan model yang sangat baik. Garis diagonal putus-putus mewakili model tebakan acak.

- **AUC (Area Under the Curve):**
  - Ini adalah skor tunggal (antara 0 dan 1) yang merangkum seluruh kurva ROC. Skor AUC yang mendekati 1.0 menandakan model memiliki kemampuan pemisahan kelas yang luar biasa. Skor 0.5 berarti tidak lebih baik dari tebakan acak.
  - Metrik ini sangat kuat untuk mengevaluasi model klasifikasi biner karena tidak bergantung pada satu ambang batas klasifikasi tertentu.

## Blok 17: Menyimpan Model

- **Tujuan:** Menyimpan bobot dan arsitektur model yang telah dilatih ke dalam file. Ini memungkinkan model untuk digunakan kembali di masa depan (untuk inferensi atau pelatihan lebih lanjut) tanpa harus mengulang seluruh proses dari awal.
- **Analisis Rinci:**
  - **Kode ini menunjukkan dua metode umum untuk menyimpan model di PyTorch:**
    - 1. `torch.save(vit_model.state_dict(), ...)` (.pth):**
      - Ini adalah metode yang direkomendasikan. `state_dict` adalah kamus (dictionary) yang hanya berisi parameter yang dapat dipelajari dari model (bobot dan bias).
      - Keuntungan: Fleksibel, ringan, dan tidak mudah rusak jika kode arsitektur model diubah. Anda perlu memiliki kelas model yang sama untuk memuatnya kembali.
    - 2. `pickle.dump(vit_model, ...)` (.pkl):**
      - Metode ini menyimpan seluruh objek model (arsitektur Python dan bobot) ke dalam satu file.
      - Keuntungan: Lebih sederhana untuk dimuat kembali.
      - Kerugian: Bisa menjadi rapuh (*brittle*). Jika Anda mengubah nama file atau struktur direktori dari kode definisi model asli, file .pkl mungkin gagal dimuat.

## Dibawah Ini Merupakan Full Kode Serta Penjelasannya Di Setiap Baris

### Blok 1: Impor Pustaka dan Koneksi Google Drive

```
# Mengimpor kelas 'drive' dari pustaka google.colab untuk menghubungkan Google Colab dengan Google Drive
from google.colab import drive
# Menghubungkan (mount) Google Drive ke direktori '/content/drive' di lingkungan Colab
drive.mount('/content/drive')
# Menginstal pustaka 'split-folders' untuk membagi dataset, 'timm' untuk model pretrained, dan 'torch-summary' untuk
ringkasannya
!pip install split-folders timm torch-summary
# Mengimpor pustaka 'os' untuk berinteraksi dengan sistem operasi, seperti mengakses file dan direktori
import os
# Mengimpor pustaka 'copy' untuk membuat salinan objek
import copy
# Mengimpor pustaka 'random' untuk menghasilkan angka acak
import random
# Mengimpor pustaka 'warnings' untuk mengelola pesan peringatan
import warnings
# Mengimpor pustaka 'numpy' (dengan alias 'np') untuk operasi numerik, terutama array multidimensi
import numpy as np
# Mengimpor pustaka 'pandas' (dengan alias 'pd') untuk manipulasi dan analisis data, khususnya DataFrame
import pandas as pd
# Mengimpor 'matplotlib.pyplot' (alias 'plt') untuk membuat visualisasi dan plot
import matplotlib.pyplot as plt
# Mengimpor 'seaborn' (alias 'sns') untuk visualisasi data statistik yang lebih menarik; mengatur gaya default ke 'darkgrid'
import seaborn as sns; sns.set(style='darkgrid')
# Mengimpor 'cv2' (OpenCV) untuk pemrosesan gambar
import cv2
# Mengimpor pustaka 'splitfolders' untuk membagi dataset gambar ke dalam folder train, val, dan test
import splitfolders
# Mengimpor kembali 'matplotlib.pyplot' (sudah diimpor sebelumnya, ini redundan)
import matplotlib.pyplot as plt
# Mengimpor kembali 'seaborn' (sudah diimpor sebelumnya, ini redundan)
import seaborn as sns
# Mengimpor kelas 'Image' dari pustaka 'Pillow' (PIL) untuk bekerja dengan gambar
from PIL import Image
# Mengimpor kelas 'Counter' dari pustaka 'collections' untuk menghitung frekuensi item dalam list
from collections import Counter
# Mengimpor kelas 'Path' dari pustaka 'pathlib' untuk bekerja dengan path file secara object-oriented
from pathlib import Path
# Mengimpor pustaka 'itertools' untuk bekerja dengan iterator
import itertools
# Mengimpor kembali pustaka 'pathlib' (sudah diimpor sebelumnya, ini redundan)
import pathlib
# Mengimpor pustaka utama 'torch' untuk deep learning
import torch
# Mengimpor 'nn' dari 'torch' yang berisi modul untuk membangun jaringan saraf (layer, fungsi aktivasi, dll.)
import torch.nn as nn
# Mengimpor 'F' dari 'torch.nn' yang berisi fungsi-fungsi seperti fungsi aktivasi dan loss
```

```

import torch.nn.functional as F
# Mengimpor 'optim' dari 'torch' yang berisi algoritma optimisasi seperti Adam dan SGD
from torch import optim
# Mengimpor 'DataLoader', 'Dataset', dan 'random_split' untuk memuat dan memproses dataset
from torch.utils.data import DataLoader, Dataset, random_split
# Mengimpor 'ReduceLROnPlateau' untuk mengurangi learning rate secara dinamis selama training
from torch.optim.lr_scheduler import ReduceLROnPlateau

# Mengimpor 'torchvision' yang menyediakan dataset populer, arsitektur model, dan transformasi gambar untuk PyTorch
import torchvision
# Mengimpor 'transforms' dan 'utils' dari 'torchvision'
from torchvision import transforms, utils
# Mengimpor 'ImageFolder' untuk memuat dataset dari folder yang terstruktur berdasarkan kelas
from torchvision.datasets import ImageFolder

# Mengimpor fungsi 'create_model' dari pustaka 'timm' (PyTorch Image Models)
from timm import create_model
# Mengimpor fungsi 'summary' dari 'torchsummary' untuk menampilkan ringkasan arsitektur model
from torchsummary import summary

# Mengimpor metrik evaluasi dari 'scikit-learn' seperti confusion matrix dan classification report
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, roc_curve
# Mengimpor 'tqdm' dan 'trange' dari 'tqdm.notebook' untuk menampilkan progress bar yang interaktif di notebook
from tqdm.notebook import tqdm, trange
# Mengimpor kembali 'roc_auc_score' dan 'roc_curve' dari 'sklearn.metrics' (redundan)
from sklearn.metrics import roc_auc_score, roc_curve

```

## Blok 2: Membaca dan Menampilkan Metadata

```

# Mendefinisikan path ke file metadata.csv di Google Drive
file_path = '/content/drive/MyDrive/TUGAS AI LANJUT/TUGASKU/metadata.csv'

# Membaca file CSV menggunakan pandas dan menyimpannya sebagai DataFrame
labels_df = pd.read_csv(file_path)
# Menampilkan lima baris pertama dari DataFrame dalam format markdown untuk visualisasi yang rapi
print(labels_df.head().to_markdown())
# Menampilkan daftar file dan folder di dalam direktori dataset tumor otak
os.listdir('/content/drive/MyDrive/TUGAS AI LANJUT/TUGASKU/Brain Tumor Data Set/Brain Tumor Data Set')
# Mendefinisikan path direktori utama data
data_dir = '/content/drive/MyDrive/TUGAS AI LANJUT/TUGASKU/Brain Tumor Data Set/Brain Tumor Data Set'
# Mengonversi path string menjadi objek Path dari pustaka pathlib untuk kemudahan manipulasi
data_dir = pathlib.Path(data_dir)

```

### Blok 3: Pembagian Dataset dan Definisi Transformasi

```
# Baris ini dikomentari, fungsinya untuk membagi data di 'data_dir' menjadi folder train (80%) dan val (20%) di 'output'  
#splitfolders.ratio(data_dir, output='/content/drive/MyDrive/TUGAS AI LANJUT/TUGASKU/brain_split', seed=20, ratio=(0.8,  
0.2))  
  
# Mendefinisikan path direktori data yang sudah dibagi menjadi folder train dan val  
data_dir = '/content/drive/MyDrive/TUGAS AI LANJUT/TUGASKU/brain_split'  
# Mengonversi path string menjadi objek Path  
data_dir = pathlib.Path(data_dir)  
# Membuat pipeline transformasi untuk data training (augmentasi data)  
train_transform = transforms.Compose([  
    transforms.Resize((224, 224)),          # Mengubah ukuran semua gambar menjadi 224x224 piksel  
    transforms.RandomHorizontalFlip(p=0.5),   # Membalik gambar secara horizontal dengan probabilitas 50%  
    transforms.RandomVerticalFlip(p=0.5),     # Membalik gambar secara vertikal dengan probabilitas 50%  
    transforms.RandomRotation(30),           # Merotasi gambar secara acak hingga 30 derajat  
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)), # Melakukan crop acak dan resize kembali ke 224x224  
    transforms.ColorJitter(brightness=0.3),    # Mengubah kecerahan gambar secara acak  
    transforms.ToTensor(),                  # Mengonversi gambar dari format PIL/numpy ke format tensor PyTorch  
    transforms.Normalize(mean=[0.485, 0.456, 0.406], # Menormalisasi tensor gambar menggunakan mean dari ImageNet  
                         std=[0.229, 0.224, 0.225]) # Menormalisasi tensor gambar menggunakan standar deviasi dari ImageNet  
])  
  
# Membuat pipeline transformasi untuk data validasi (tanpa augmentasi acak)  
val_transform = transforms.Compose([  
    transforms.Resize((224, 224)),          # Mengubah ukuran gambar menjadi 224x224 piksel  
    transforms.ToTensor(),                  # Mengonversi gambar ke format tensor PyTorch  
    transforms.Normalize(mean=[0.485, 0.456, 0.406], # Menormalisasi tensor dengan mean dari ImageNet  
                         std=[0.229, 0.224, 0.225]) # Menormalisasi tensor dengan std dari ImageNet  
])
```

### Blok 4: Membuat Dataset dan Menampilkan Transformasi

```
# Membuat dataset training dari folder "train" dan menerapkan transformasi train_transform  
train_set = torchvision.datasets.ImageFolder(data_dir.joinpath("train"), transform=train_transform)  
  
# Membuat dataset validasi dari folder "val" dan menerapkan transformasi val_transform  
val_set = torchvision.datasets.ImageFolder(data_dir.joinpath("val"), transform=val_transform)  
  
# Menampilkan transformasi yang diterapkan pada dataset training  
print("Train transforms:", train_set.transform)  
  
# Menampilkan transformasi yang diterapkan pada dataset validasi  
print("Val transforms:", val_set.transform)
```

## Blok 5: Definisi Transformasi Individual untuk Visualisasi

```
# Transformasi untuk hanya mengubah ukuran gambar dan mengubahnya menjadi tensor
resize_transform = transforms.Compose([
    transforms.Resize((224, 224)), # Ubah ukuran gambar menjadi 224x224 piksel
    transforms.ToTensor()         # Ubah ke tensor PyTorch
])

# Transformasi untuk selalu membalik gambar secara horizontal
horizontal_flip_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=1.0), # Flip horizontal dengan probabilitas 100%
    transforms.ToTensor()                 # Ubah ke tensor
])

# Transformasi untuk selalu membalik gambar secara vertikal
vertical_flip_transform = transforms.Compose([
    transforms.RandomVerticalFlip(p=1.0), # Flip vertikal dengan probabilitas 100%
    transforms.ToTensor()               # Ubah ke tensor
])

# Transformasi untuk merotasi gambar secara acak
rotation_transform = transforms.Compose([
    transforms.RandomRotation(30), # Rotasi gambar secara acak dalam rentang -30 sampai 30 derajat
    transforms.ToTensor()         # Ubah ke tensor
])

# Transformasi untuk melakukan zoom acak
zoom_transform = transforms.Compose([
    transforms.RandomResizedCrop(256, scale=(0.8, 1.0)), # Crop acak lalu resize ke 256x256 (seharusnya 224x224 agar konsisten)
    transforms.ToTensor()                 # Ubah ke tensor
])

# Transformasi untuk mengubah kecerahan secara acak
brightness_transform = transforms.Compose([
    transforms.ColorJitter(brightness=0.3), # Ubah brightness secara acak
    transforms.ToTensor()                 # Ubah ke tensor
])

# Transformasi untuk normalisasi gambar
normalize_transform = transforms.Compose([
    transforms.ToTensor(), # Ubah ke tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], # Normalisasi channel RGB
                        std=[0.229, 0.224, 0.225]) # Menggunakan nilai standar dari ImageNet
])

# Transformasi untuk hanya mengubah gambar menjadi tensor
totensor_transform = transforms.Compose([
    transforms.ToTensor() # Ubah ke tensor tanpa augmentasi lain
])

# Membuat list berisi tuple (nama_transformasi, objek_transformasi) untuk iterasi
```

```

transforms_list = [
    ('Resize', resize_transform),
    ('Horizontal Flip', horizontal_flip_transform),
    ('Vertical Flip', vertical_flip_transform),
    ('Rotation', rotation_transform),
    ('Zoom', zoom_transform),
    ('Brightness', brightness_transform),
    ('ToTensor', totensor_transform),
    ('Normalize', normalize_transform)
]

```

## Blok 6: Visualisasi Efek Augmentasi Data

```

# Memuat dataset dari folder "train" tanpa menerapkan transformasi apapun untuk mendapatkan gambar asli
original_dataset = torchvision.datasets.ImageFolder(data_dir.joinpath("train"))

# Membuat figure dan set sumbu subplot: jumlah baris sesuai jumlah transformasi, dengan 2 kolom (asli vs transformasi)
fig, axes = plt.subplots(len(transforms_list), 2, figsize=(12, 3 * len(transforms_list)))

# Menambahkan judul utama untuk keseluruhan plot
fig.suptitle('Original vs Transformed', fontsize=16, y=0.99)

# Melakukan iterasi melalui setiap transformasi dalam 'transforms_list'
for idx, (name, fn) in enumerate(transforms_list):
    # Mengambil satu gambar acak dari dataset asli
    img, _ = original_dataset[torch.randint(len(original_dataset), (1,)).item()]

    # Jika gambar sudah dalam format tensor, ubah kembali ke PIL Image untuk konsistensi
    img_pil = transforms.ToPILImage()(img) if isinstance(img, torch.Tensor) else img

    # Menerapkan fungsi transformasi (fn) pada gambar PIL
    img_t = fn(img_pil)

    # Mengubah tensor hasil transformasi menjadi array numpy dan mengubah urutan dimensi (C, H, W) -> (H, W, C)
    img_np = img_t.numpy().transpose(1, 2, 0)

    # Jika transformasi adalah normalisasi, lakukan denormalisasi agar gambar bisa ditampilkan dengan benar
    if name == 'Normalize':
        img_np = (img_np * [0.229, 0.224, 0.225] + [0.485, 0.456, 0.406]).clip(0, 1)
    else:
        # Untuk transformasi lain, pastikan nilai piksel tetap dalam rentang [0, 1]
        img_np = img_np.clip(0, 1)

    # Mengonversi gambar hasil transformasi (yang mungkin berwarna) menjadi grayscale
    gray = np.dot(img_np[:, :, :3], [0.2989, 0.5870, 0.1140])

    # Mengatur subplot untuk gambar original di kolom kiri
    ax_o = axes[idx, 0]
    # Menampilkan gambar original
    ax_o.imshow(np.array(img_pil))
    # Memberi judul pada subplot

```

```

ax_o.set_title(f"Original ({name})")
# Menghilangkan sumbu (x, y)
ax_o.axis('off')

# Mengatur subplot untuk gambar hasil transformasi di kolom kanan
ax_t = axes[idx, 1]
# Menampilkan gambar yang sudah ditransformasi dalam format grayscale
ax_t.imshow(gray, cmap='gray')
# Memberi judul pada subplot
ax_t.set_title(f"Transformed ({name})")
# Menghilangkan sumbu (x, y)
ax_t.axis('off')

# Menyesuaikan layout plot agar tidak ada judul atau gambar yang tumpang tindih
plt.tight_layout(rect=[0, 0, 1, 0.98])

# Menampilkan figure yang telah dibuat
plt.show()

```

## Blok 7: Analisis Eksplorasi Data (EDA)

```

# Mendefinisikan path ke direktori data mentah (sebelum di-split)
data_dir = '/content/drive/MyDrive/TUGAS AI LANJUT/TUGASKU/Brain Tumor Data Set/Brain Tumor Data Set'
# Mengonversi path string menjadi objek Path
data_dir = pathlib.Path(data_dir)

# Baris ini dikomentari (redundan), fungsinya membagi dataset
#splitfolders.ratio(data_dir, output='/content/drive/MyDrive/TUGAS AI LANJUT/TUGASKU/brain_split', seed=20, ratio=(0.8, 0.2))

# Mendefinisikan ulang path direktori yang sudah di-split (redundan)
#data_dir = '/content/drive/MyDrive/TUGAS AI LANJUT/TUGASKU/brain_split'
# Mengonversi kembali path menjadi objek Path (redundan)
#data_dir = pathlib.Path(data_dir)

# Membuat dataset menggunakan ImageFolder dari direktori data mentah
dataset = ImageFolder(data_dir)
# Mendefinisikan kamus (dictionary) untuk memetakan indeks kelas (0, 1) ke nama label yang mudah dibaca
CLA_label = {0: 'Brain Tumor', 1: 'Healthy'}

# 1. Menghitung jumlah data per kelas
# Membuat list yang berisi semua label dari dataset
labels = [label for _, label in dataset.samples]
# Menghitung frekuensi setiap label menggunakan Counter
class_counts = Counter(labels)
# Mencetak judul
print("Jumlah data per kelas:")
# Melakukan iterasi melalui hasil perhitungan dan mencetak jumlah untuk setiap kelas
for cls_idx, cnt in class_counts.items():
    print(f" {CLA_label[cls_idx]}: {cnt}")

```

## Blok 8: Visualisasi Contoh Gambar per Kelas

```
# 2. Menampilkan contoh gambar dari setiap kelas
# Menentukan jumlah gambar yang ingin ditampilkan untuk setiap kelas
n_per_class = 3

# Mengambil semua indeks kelas yang unik dari dataset dan mengurutkannya
classes = sorted({lab for _, lab in dataset.samples})

# Membuat grid subplot. Jumlah baris = jumlah kelas, jumlah kolom = n_per_class
fig, axes = plt.subplots(len(classes), n_per_class, figsize=(4*n_per_class, 4*len(classes)))

# Melakukan iterasi untuk setiap kelas (baris)
for row, cls_idx in enumerate(classes):
    # Mengumpulkan semua indeks gambar yang termasuk dalam kelas saat ini
    idxs = [i for i, (_, lab) in enumerate(dataset.samples) if lab == cls_idx]

    # Memilih secara acak beberapa indeks dari daftar yang dikumpulkan
    selected = random.sample(idxs, k=min(n_per_class, len(idxs)))

    # Melakukan iterasi untuk setiap gambar yang terpilih (kolom)
    for col, idx in enumerate(selected):
        # Mengambil path gambar dari dataset berdasarkan indeksnya
        img_path, _ = dataset.samples[idx]

        # Membuka gambar menggunakan PIL dan mengonversinya ke format RGB
        img = Image.open(img_path).convert('RGB')

        # Menentukan di subplot mana gambar akan ditampilkan
        ax = axes[row, col] if len(classes) > 1 else axes[col]

        # Menampilkan gambar pada subplot yang telah ditentukan
        ax.imshow(img)

        # Menetapkan judul untuk setiap gambar, berisi nama kelas dan nomor urutnya
        ax.set_title(f"{{CLA_label[cls_idx]}} {{col+1}}")

        # Menghilangkan sumbu x dan y dari subplot
        ax.axis('off')

    # Menambahkan judul utama untuk seluruh visualisasi
plt.suptitle("Contoh 3 Gambar per Kelas", y=1.02)

# Menyesuaikan layout agar tidak ada elemen yang tumpang tindih
plt.tight_layout()

# Menampilkan hasil plot
plt.show()
```

## Blok 9: Analisis Dimensi Gambar

```
# 3. Menganalisis ukuran dimensi gambar
# Mengambil 6 sampel gambar secara acak dari dataset
sample_paths = random.sample(dataset.samples, k=6)

# Mencetak judul untuk output teks
print("\nUkuran (dimensi) untuk 6 gambar acak:")
# Melakukan iterasi melalui 6 sampel yang telah dipilih
for img_path, _ in sample_paths:
    # Membuka gambar menggunakan PIL
    with Image.open(img_path) as img:
        # Mengambil lebar (width) dan tinggi (height) gambar
        w, h = img.size

        # Menampilkan nama file beserta dimensinya (lebar x tinggi)
        print(f" {Path(img_path).name}: {w} x {h}")

# (Opsiional) Visualisasi keenam gambar tersebut
# Membuat grid subplot berukuran 2 baris x 3 kolom
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

# Melakukan iterasi melalui setiap sumbu subplot dan path gambar secara bersamaan
for ax, (img_path, _) in zip(axes.flatten(), sample_paths):
    # Membuka gambar dan mengonversinya ke format RGB
    img = Image.open(img_path).convert("RGB")
    # Mengambil dimensi lebar dan tinggi gambar
    w, h = img.size
    # Menampilkan gambar pada subplot
    ax.imshow(img)
    # Menetapkan judul subplot dengan format "lebarxtinggi"
    ax.set_title(f"{w}x{h}")
    # Menghilangkan sumbu x dan y
    ax.axis("off")

# Menambahkan judul utama untuk keseluruhan visualisasi
plt.suptitle("Contoh 6 Gambar Acak dengan Ukuran (WxH)", y=1.02)

# Menyesuaikan layout agar rapi
plt.tight_layout()

# Menampilkan plot
plt.show()
```

## Blok 10: Analisis Distribusi Nilai Piksel

```
# 4. Menganalisis distribusi nilai piksel (histogram)
# Menginisialisasi sebuah list kosong untuk menyimpan nilai piksel dari semua gambar sampel
all_pixels = []

# Mengambil 6 sampel gambar acak dari dataset
sample_paths = random.sample(dataset.samples, k=min(6, len(dataset.samples)))

# Melakukan iterasi untuk setiap path gambar dalam sampel
for path, _ in sample_paths:
    # Membaca gambar menggunakan OpenCV dalam mode grayscale (hitam-putih)
    img = cv2.imread(str(path), cv2.IMREAD_GRAYSCALE)

    # Jika gambar tidak berhasil dibaca (misalnya file rusak), lanjutkan ke iterasi berikutnya
    if img is None:
        continue

    # Mengubah array gambar 2D menjadi 1D (flatten) dan menambahkannya ke list 'all_pixels'
    all_pixels.append(img.flatten())

# Menggabungkan semua array piksel 1D menjadi satu array numpy besar
all_pixels = np.concatenate(all_pixels)

# Membuat figure baru untuk plot histogram
plt.figure(figsize=(8,4))

# Membuat plot histogram dari semua nilai piksel. 'bins=256' karena nilai piksel grayscale berkisar 0-255
plt.hist(all_pixels, bins=256, alpha=0.7)

# Menambahkan judul pada plot
plt.title("Histogram Nilai Pixel")
# Menambahkan label untuk sumbu x
plt.xlabel("Nilai Pixel")
# Menambahkan label untuk sumbu y
plt.ylabel("Frekuensi")

# Menyesuaikan layout agar rapi
plt.tight_layout()

# Menampilkan plot histogram
plt.show()
```

## Blok 11: Analisis Jumlah Channel Warna

```
# 6. Menganalisis dimensi warna (jumlah channel)
# Menentukan jumlah sampel yang akan dianalisis, maksimal 100 atau sesuai jumlah data jika kurang dari 100
k = min(100, len(dataset.samples))
# Mengambil sampel gambar secara acak
sample_paths = random.sample(dataset.samples, k=k)

# Menginisialisasi Counter untuk menghitung jumlah gambar berdasarkan jumlah channel-nya
channel_counts = Counter()

# Melakukan iterasi melalui setiap path gambar dalam sampel
for img_path, _ in sample_paths:
    # Membaca gambar dengan flag IMREAD_UNCHANGED untuk mempertahankan channel aslinya (misal, alpha channel)
    img = cv2.imread(str(img_path), cv2.IMREAD_UNCHANGED)

    # Jika gambar gagal dibaca, lewati
    if img is None:
        continue

    # Memeriksa dimensi gambar: jika 3 dimensi, jumlah channel ada di shape[2]. Jika tidak, gambar adalah grayscale (1 channel).
    chans = img.shape[2] if img.ndim == 3 else 1

    # Menambahkan hitungan untuk jumlah channel yang ditemukan
    channel_counts[chans] += 1

# Mencetak hasil distribusi jumlah channel
print(f"\nDistribusi jumlah channel pada sample {k} gambar:")
# Melakukan iterasi melalui hasil perhitungan dan menampilkannya
for chans, cnt in channel_counts.items():
    print(f" {chans} channel: {cnt} gambar")

# Menampilkan contoh detail untuk 6 gambar acak
print("\nContoh 6 gambar beserta jumlah channel-nya:")
# Mengambil 6 sampel acak dari sampel sebelumnya
for img_path, _ in random.sample(sample_paths, k=min(6, len(sample_paths))):
    # Membaca gambar
    img = cv2.imread(str(img_path), cv2.IMREAD_UNCHANGED)

    # Menentukan jumlah channel (logika yang sama seperti sebelumnya)
    chans = img.shape[2] if img.ndim == 3 else 1

    # Mencetak nama file dan jumlah channel-nya
    print(f" {Path(img_path).name}: {chans} channel")
```

## Blok 12: Pengaturan Hyperparameter dan Persiapan Data untuk Model ViT

```
# Mendefinisikan ulang path direktori data mentah (redundan)
data_dir = '/content/drive/MyDrive/TUGAS AI LANJUT/TUGASKU/Brain Tumor Data Set/Brain Tumor Data Set'
# Mengonversi path menjadi objek Path (redundan)
data_dir = pathlib.Path(data_dir)

# Baris ini dikomentari (redundan)
#splitfolders.ratio(data_dir, output='/content/drive/MyDrive/TUGAS AI LANJUT/TUGASKU/brain_split', seed=20, ratio=(0.8,
#0.2))

# Mendefinisikan path ke data yang sudah di-split
data_dir = '/content/drive/MyDrive/TUGAS AI LANJUT/TUGASKU/brain_split'
# Mengonversi path menjadi objek Path
data_dir = pathlib.Path(data_dir)

# Mendefinisikan Hyperparameters untuk training model
learning_rate = 1e-4      # Laju pembelajaran (seberapa besar update bobot pada setiap iterasi)
weight_decay = 0.01        # Parameter untuk regularisasi L2, mencegah overfitting
batch_size = 32            # Jumlah sampel data yang diproses dalam satu iterasi
num_epochs = 60            # Jumlah berapa kali keseluruhan dataset dilewatkan melalui model
num_classes = 2             # Jumlah kelas output (Tumor Otak, Sehat)
img_size = 224              # Ukuran gambar input (tinggi dan lebar)
patch_size = 16              # Ukuran patch untuk model Vision Transformer (ViT)
embed_dim = 768            # Dimensi vektor embedding untuk setiap patch
num_heads = 8                # Jumlah kepala dalam Multi-Head Attention di ViT
depth = 12                  # Jumlah lapisan Transformer Encoder di ViT
mlp_dim = 3072              # Dimensi hidden layer di dalam MLP (Feed-Forward Network) ViT

# === Mendefinisikan Transformasi Data untuk Model ViT ===
# Transformasi untuk data training ViT
train_transform_vit = transforms.Compose([
    transforms.Resize((img_size, img_size)),    # Mengubah ukuran gambar
    transforms.RandomHorizontalFlip(p=0.5),    # Augmentasi: flip horizontal
    transforms.RandomVerticalFlip(p=0.5),     # Augmentasi: flip vertikal
    transforms.RandomRotation(30),           # Augmentasi: rotasi
    transforms.ColorJitter(brightness=0.3),   # Augmentasi: kecerahan
    transforms.ToTensor(),                 # Konversi ke Tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], # Normalisasi
                        std=[0.229, 0.224, 0.225])
])

# Transformasi untuk data validasi ViT (tanpa augmentasi acak)
val_transform_vit = transforms.Compose([
    transforms.Resize((img_size, img_size)),    # Mengubah ukuran gambar
    transforms.ToTensor(),                   # Konversi ke Tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], # Normalisasi
                        std=[0.229, 0.224, 0.225])
])

# Memuat dataset training dan validasi dengan transformasi yang sesuai untuk ViT
train_set_vit = ImageFolder(data_dir.joinpath("train"), transform=train_transform_vit)
```

```

val_set_vit = ImageFolder(data_dir.joinpath("val"), transform=val_transform_vit)

# Membuat DataLoader untuk memuat data secara batch, mengacak data training
train_loader = DataLoader(train_set_vit, batch_size=batch_size, shuffle=True, num_workers=2, pin_memory=True)
# Membuat DataLoader untuk data validasi (tidak perlu diacak)
val_loader = DataLoader(val_set_vit, batch_size=batch_size, shuffle=False, num_workers=2, pin_memory=True)

# Mencetak ukuran dataset training dan validasi
print(f"Train size: {len(train_set_vit)} | Validation size: {len(val_set_vit)}")

```

### Blok 13: Membangun dan Melatih Model ViT

```

# Menentukan device (perangkat) untuk training: 'cuda' jika GPU tersedia, jika tidak 'cpu'
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Membuat model Vision Transformer (ViT) menggunakan pustaka timm
vit_model = create_model(
    'vit_base_patch16_224',      # Nama arsitektur: ViT-Base dengan ukuran patch 16x16 dan input 224x224
    pretrained=True,            # Menggunakan bobot yang sudah dilatih sebelumnya pada dataset ImageNet
    num_classes=num_classes,    # Menyesuaikan layer output dengan jumlah kelas kita (2)
    drop_rate=0.1,              # Tingkat dropout untuk regularisasi
    drop_path_rate=0.1,         # Tingkat dropout pada koneksi antar blok transformer
    attn_drop_rate=0.1          # Tingkat dropout pada mekanisme attention
)

# Memindahkan model ke perangkat yang telah ditentukan (GPU atau CPU)
vit_model.to(device)
# === Menampilkan Ringkasan Model ===
# Menampilkan arsitektur model, jumlah parameter, dan ukuran output setiap layer
summary(vit_model, input_size=(3, img_size, img_size))

# === Mendefinisikan Fungsi Loss, Optimizer, dan Scheduler ===
# Mendefinisikan fungsi loss Cross-Entropy, cocok untuk masalah klasifikasi multi-kelas
criterion = nn.CrossEntropyLoss()
# Mendefinisikan optimizer AdamW, varian dari Adam yang lebih baik dalam menangani weight decay
optimizer = optim.AdamW(vit_model.parameters(), lr=learning_rate, weight_decay=weight_decay)
# Mendefinisikan scheduler untuk mengurangi learning rate secara otomatis jika loss validasi tidak membaik
scheduler = ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=3, verbose=True)

# === Loop Pelatihan (Training Loop) ===
# Menginisialisasi list untuk menyimpan nilai loss training dan validasi setiap epoch
train_losses, val_losses = [], []
# Melakukan iterasi sebanyak jumlah epoch yang telah ditentukan
for epoch in range(num_epochs):
    # Mengatur model ke mode training (mengaktifkan dropout, dll.)
    vit_model.train()
    # Menginisialisasi variabel untuk mengakumulasi loss dalam satu epoch
    running_loss = 0.0
    # Melakukan iterasi melalui setiap batch data di train_loader dengan progress bar
    for images, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs} - Training"):
        # Memindahkan gambar dan label ke device (GPU/CPU)

```

```

images, labels = images.to(device), labels.to(device)

# Mengosongkan gradien dari iterasi sebelumnya
optimizer.zero_grad()
# Melakukan forward pass: melewakan gambar melalui model untuk mendapatkan output (logits)
outputs = vit_model(images)
# Menghitung loss antara prediksi model dan label sebenarnya
loss = criterion(outputs, labels)
# Melakukan backward pass: menghitung gradien loss terhadap parameter model
loss.backward()
# Memperbarui bobot model menggunakan optimizer
optimizer.step()

# Menambahkan loss dari batch ini ke total running_loss
running_loss += loss.item()

# Menghitung rata-rata loss training untuk epoch ini
avg_train_loss = running_loss / len(train_loader)
# Menyimpan rata-rata loss training
train_losses.append(avg_train_loss)

# === Fase Validasi ===
# Mengatur model ke mode evaluasi (menonaktifkan dropout, dll.)
vit_model.eval()
# Menginisialisasi variabel untuk mengakumulasi loss validasi
val_loss = 0.0
# Mematikan perhitungan gradien untuk fase validasi untuk menghemat memori dan komputasi
with torch.no_grad():
    # Melakukan iterasi melalui setiap batch data di val_loader
    for images, labels in tqdm(val_loader, desc=f"Epoch {epoch+1}/{num_epochs} - Validation"):
        # Memindahkan gambar dan label ke device
        images, labels = images.to(device), labels.to(device)
        # Mendapatkan prediksi dari model
        outputs = vit_model(images)
        # Menghitung loss validasi
        loss = criterion(outputs, labels)
        # Menambahkan loss dari batch ini ke total loss validasi
        val_loss += loss.item()

# Menghitung rata-rata loss validasi untuk epoch ini
avg_val_loss = val_loss / len(val_loader)
# Menyimpan rata-rata loss validasi
val_losses.append(avg_val_loss)

# Memperbarui scheduler dengan loss validasi. Jika tidak ada peningkatan, LR akan dikurangi.
scheduler.step(avg_val_loss)

# Mencetak ringkasan loss training dan validasi untuk epoch saat ini
print(f"Epoch [{epoch+1}/{num_epochs}] - Train Loss: {avg_train_loss:.4f} - Val Loss: {avg_val_loss:.4f}")

# Mencetak pesan bahwa training telah selesai

```

```
print("Training selesai!")
```

#### Blok 14: Visualisasi Loss Pelatihan

```
# Membuat figure baru untuk plot loss
plt.figure(figsize=(8,6))
# Membuat plot untuk loss training dari waktu ke waktu (per epoch)
plt.plot(train_losses, label="Training Loss", marker='o')
# Membuat plot untuk loss validasi dari waktu ke waktu (per epoch)
plt.plot(val_losses, label="Validation Loss", marker='o')
# Menambahkan label pada sumbu x
plt.xlabel("Epoch")
# Menambahkan label pada sumbu y
plt.ylabel("Loss")
# Menambahkan judul pada plot
plt.title("Train vs Validation Loss per Epoch")
# Menampilkan legenda untuk membedakan garis plot
plt.legend()
# Menambahkan grid untuk kemudahan pembacaan
plt.grid(True)
# Menyesuaikan layout agar rapi
plt.tight_layout()
# Menampilkan plot
plt.show()
```

#### Blok 15: Evaluasi Model dengan Confusion Matrix

```
# Mengatur model ke mode evaluasi
vit_model.eval()

# Menginisialisasi list kosong untuk menyimpan label asli (ground truth)
y_true = []
# Menginisialisasi list kosong untuk menyimpan prediksi model
y_pred = []

# Mematikan perhitungan gradien selama evaluasi
with torch.no_grad():
    # Melakukan iterasi melalui data validasi dengan progress bar
    for images, labels in tqdm(val_loader, desc="Evaluating"):
        # Memindahkan gambar ke device
        images = images.to(device)
        # Mendapatkan output (logits) dari model
        outputs = vit_model(images)
        # Mendapatkan prediksi kelas dengan mengambil indeks dari nilai output tertinggi
        _, preds = torch.max(outputs, 1)
        # Menambahkan label asli (yang ada di CPU) ke list y_true
        y_true.extend(labels.cpu().numpy())
        # Menambahkan prediksi model (yang ada di CPU) ke list y_pred
        y_pred.extend(preds.cpu().numpy())

# Membuat confusion matrix menggunakan label asli dan prediksi
```

```

cm = confusion_matrix(y_true, y_pred)
# Membuat laporan klasifikasi yang berisi precision, recall, f1-score
cr = classification_report(y_true, y_pred, target_names=list(train_set_vit.classes))

# Membuat plot untuk confusion matrix
plt.figure(figsize=(6,6))
# Menggunakan heatmap dari Seaborn untuk memvisualisasikan confusion matrix
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=train_set_vit.classes, yticklabels=train_set_vit.classes)
# Menambahkan label pada sumbu x
plt.xlabel("Predicted Label")
# Menambahkan label pada sumbu y
plt.ylabel("True Label")
# Menambahkan judul pada plot
plt.title("Confusion Matrix")
# Menampilkan plot
plt.show()

# Menampilkan laporan klasifikasi
print("Classification Report:\n", cr)

```

#### Blok 16: Evaluasi Lengkap dengan ROC Curve

```

# Mengatur model ke mode evaluasi
vit_model.eval()

# Menginisialisasi list untuk menyimpan label asli, prediksi, dan probabilitas prediksi
y_true = []
y_pred = []
y_probs = []

# Mematikan perhitungan gradien
with torch.no_grad():
    # Melakukan iterasi melalui data validasi
    for images, labels in tqdm(val_loader, desc="Evaluating"):
        # Memindahkan gambar dan label ke device
        images = images.to(device)
        labels = labels.to(device)

        # Mendapatkan output (logits) dari model
        outputs = vit_model(images)
        # Menerapkan fungsi softmax untuk mengubah logits menjadi probabilitas
        probs = torch.softmax(outputs, dim=1)
        # Mendapatkan kelas prediksi dari probabilitas tertinggi
        _, preds = torch.max(probs, 1)

        # Menyimpan label asli
        y_true.extend(labels.cpu().numpy())
        # Menyimpan prediksi kelas
        y_pred.extend(preds.cpu().numpy())
        # Menyimpan probabilitas untuk kelas positif (label 1) untuk kurva ROC
        y_probs.extend(probs[:, 1].cpu().numpy())

```

```

# === Menghitung dan Menampilkan ROC AUC Score dan Curve ===
# Menghitung skor Area Under the Curve (AUC) dari kurva ROC
roc_auc = roc_auc_score(y_true, y_probs)
# Mencetak skor ROC AUC
print(f"ROC AUC Score: {roc_auc:.4f}")

# Menghitung False Positive Rate (FPR) dan True Positive Rate (TPR) untuk plot kurva ROC
fpr, tpr, _ = roc_curve(y_true, y_probs)
# Membuat figure baru
plt.figure()
# Membuat plot kurva ROC
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.4f})")
# Menambahkan garis diagonal sebagai referensi (model acak)
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
# Menambahkan label sumbu x
plt.xlabel("False Positive Rate")
# Menambahkan label sumbu y
plt.ylabel("True Positive Rate")
# Menambahkan judul plot
plt.title("Receiver Operating Characteristic (ROC) Curve")
# Menampilkan legenda
plt.legend(loc="lower right")
# Menambahkan grid
plt.grid()
# Menampilkan plot
plt.show()

# === Confusion Matrix dan Classification Report (redundan, sudah ada di blok sebelumnya) ===
# Membuat confusion matrix
cm = confusion_matrix(y_true, y_pred)
# Membuat classification report
cr = classification_report(y_true, y_pred, target_names=list(train_set_vit.classes))

# Membuat plot Confusion Matrix
plt.figure(figsize=(6,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=train_set_vit.classes,
            yticklabels=train_set_vit.classes)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# Menampilkan Classification Report
print("Classification Report:\n", cr)

```

## Blok 17: Menyimpan Model

```
# Mengimpor pustaka os (sudah diimpor di awal)
import os
# Mendefinisikan direktori tujuan untuk menyimpan model
save_dir = '/content/drive/MyDrive/AI LANJUT/TUGASKU/saved_model'
# Membuat direktori jika belum ada, exist_ok=True mencegah error jika folder sudah ada
os.makedirs(save_dir, exist_ok=True)

# Menggabungkan path direktori dan nama file untuk path penyimpanan lengkap
pth_path = os.path.join(save_dir, 'vit_model_brain.pth')
# Menyimpan hanya state_dict (bobot dan parameter yang telah dilatih) dari model
torch.save(vit_model.state_dict(), pth_path)
# Mencetak pesan konfirmasi lokasi penyimpanan model
print(f"Model disimpan dalam format .pth di: {pth_path}")

# Mengimpor pustaka pickle untuk serialisasi objek Python
import pickle
# Menggabungkan path direktori dan nama file untuk path penyimpanan .pkl
pkl_path = os.path.join(save_dir, 'vit_model_brain.pkl')
# Membuka file dalam mode 'write binary' (wb)
with open(pkl_path, 'wb') as f:
    # Menyimpan seluruh objek model (arsitektur + bobot) ke dalam file menggunakan pickle
    pickle.dump(vit_model, f)
# Mencetak pesan konfirmasi lokasi penyimpanan model
print(f"Model disimpan dalam format .pkl di: {pkl_path}")
```