

LAPORAN UJIAN AKHIR SEMESTER
MATA KULIAH *MACHINE LEARNING*
TK-45-GAB-G04

Disusun untuk memenuhi nilai UAS mata kuliah *Machine Learning*
di Program Studi S1 Teknik Komputer



Disusun oleh:

Anita Firda Nuralifah
1103213117

FAKULTAS TEKNIK ELEKTRO
UNIVERSITAS TELKOM
BANDUNG
2024

Computer Vision Hugging Face Course

(Chapter 1-4)

Chapter 1

Computer vision adalah bidang yang fokus pada pengembangan teknologi agar mesin dapat "melihat" dan memahami data visual. Ini melibatkan pengembangan metode untuk memperoleh, memproses, dan menganalisis gambar atau video untuk menghasilkan representasi dan interpretasi yang bermakna tentang dunia. Teknologi ini memungkinkan mesin untuk menganalisis informasi visual guna memahami lingkungan sekitar.

Di India, dengan jumlah sepeda motor terbanyak di dunia, banyak pengendara yang lupa memakai helm, yang bisa berisiko cedera serius. Untuk mengatasi ini, pemerintah India bersama lembaga lain mengembangkan sistem computer vision yang secara otomatis mendeteksi pengendara tanpa helm dan nomor plat kendaraan mereka, kemudian memberikan denda untuk mencegah pelanggaran. Selain itu, computer vision juga diterapkan di berbagai bidang seperti kesehatan, ritel, dan industri lainnya.

Mobil otonom sangat bergantung pada computer vision untuk memahami lingkungan sekitarnya. Mereka menggunakan kamera dan sensor untuk mengenali objek, pejalan kaki, rambu lalu lintas, marka jalan, dan kendaraan lain. Algoritma computer vision membantu kendaraan ini untuk membuat keputusan secara real-time, seperti mengendalikan setir, akselerasi, atau pengereman. Perusahaan seperti Tesla, Waymo, dan Uber aktif mengembangkan teknologi ini untuk membuat transportasi lebih aman dan efisien.

Ekstraksi fitur adalah proses penting dalam computer vision yang memungkinkan mesin memproses data visual, seperti pengenalan wajah, pelacakan objek, dan deteksi anomali. Aplikasinya meliputi sistem keamanan (misalnya, pengenalan wajah di bandara), keamanan otomotif (seperti pelacakan objek untuk kendaraan otonom), serta deteksi anomali untuk keamanan publik dan kontrol kualitas industri. Teknologi ini terus berkembang dan memberikan solusi yang lebih canggih di berbagai sektor.

Chapter 2

1. Transfer Learning and Fine Tuning using Resnet18 Architecture

```
!pip install timm torch torchvision
```

```

Requirement already satisfied: timm in /usr/local/lib/python3.10/dist-packages (1.0.12)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.5.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.20.1+cu121)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from timm) (6.0.2)
Requirement already satisfied: huggingface_hub in /usr/local/lib/python3.10/dist-packages (from timm) (0.27.0)
Requirement already satisfied: safetensors in /usr/local/lib/python3.10/dist-packages (from timm) (0.4.5)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.16.1)
Requirement already satisfied: typing_extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.4.2)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2024.10.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (11.0.0)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from huggingface_hub->timm) (24.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface_hub->timm) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface_hub->timm) (4.67.1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2->torch) (3.0.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface_hub->timm) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface_hub->timm) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface_hub->timm) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface_hub->timm) (2024.12.14)

```

Perintah `!pip install timm torch torchvision` menginstal tiga pustaka Python penting untuk deep learning dan visi komputer. `torch` adalah inti dari PyTorch yang menyediakan komputasi tensor dan jaringan saraf, `torchvision` menyediakan dataset, model pra-terlatih, dan alat transformasi gambar, sementara `tim` menawarkan model visi terbaru dan utilitas tambahan. Bersama-sama, ketiganya memudahkan pengembangan, pelatihan, dan penerapan model visi komputer yang canggih.

Library yang digunakan

```

import torch #untuk membangun dan melatih model deep learning.
import torch.nn as nn #untuk membangun neural network.
import torch.optim as optim #untuk melatih model deep learning.
from torch.utils.data import DataLoader #untuk membuat iterable dari dataset.
from torchvision import datasets, transforms #Modul ini berisi dataset populer, arsitektur model, dan transformasi gambar umum untuk computer vision.
import timm #Merupakan library yang menyediakan koleksi model dan fungsi yang telah dilatih sebelumnya (pretrained) untuk computer vision.

```

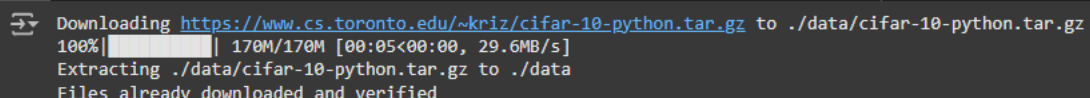
- `import torch` : untuk membangun dan melatih model deep learning.
- `import torch.nn as nn` : untuk membangun neural network.
- `import torch.optim as optim` #untuk melatih model deep learning.
- `from torch.utils.data import DataLoader` #untuk membuat iterable dari dataset.
- `from torchvision import datasets, transforms` #Modul ini berisi dataset populer, arsitektur model, dan transformasi gambar umum untuk computer vision.
- `import timm` #Merupakan library yang menyediakan koleksi model dan fungsi yang telah dilatih sebelumnya (pretrained) untuk computer vision.

Data preparation and augmentation

```
# Define data transformations
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

Kode di atas mendefinisikan serangkaian transformasi data untuk gambar sebelum digunakan dalam pelatihan model. `RandomHorizontalFlip()` dan `RandomVerticalFlip()` secara acak membalik gambar secara horizontal dan vertikal, meningkatkan variasi data sehingga model dapat mengenali objek dari berbagai orientasi. `ToTensor()` mengubah gambar menjadi tensor PyTorch dan menyesuaikan nilai piksel ke rentang $[0, 1]$. Terakhir, `Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))` menormalkan tensor dengan mengurangi mean dan membagi dengan standar deviasi untuk setiap channel warna (RGB), membantu stabilisasi proses pelatihan jaringan saraf. Transformasi ini secara keseluruhan meningkatkan kualitas dan keberagaman data serta memastikan format yang sesuai untuk pelatihan model deep learning.

```
# Load CIFAR-10 dataset
train_dataset = datasets.CIFAR10(root='./data', train=True,
    download=True, transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False,
    download=True, transform=transform)
```



```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170M/170M [00:05<00:00, 29.6MB/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Kode di atas digunakan untuk memuat dataset CIFAR-10 yang terdiri dari gambar-gambar berwarna dengan 10 kelas berbeda. `train_dataset` memuat data pelatihan dari CIFAR-10 dengan menyimpan file dataset di direktori `./data`, mengunduhnya jika belum tersedia, dan menerapkan transformasi yang telah didefinisikan sebelumnya. Sedangkan `test_dataset` memuat data pengujian dari CIFAR-10 dengan parameter `train=False`, yang berarti dataset ini digunakan untuk validasi atau pengujian model setelah pelatihan selesai. Dengan demikian, kedua dataset ini siap digunakan untuk melatih dan mengevaluasi performa model visi komputer.

```
# Define data loaders
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True,
    num_workers=4)
```

```
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False,
num_workers=4)
```

Kode di atas mendefinisikan **DataLoader** untuk memuat dataset pelatihan dan pengujian. `train_loader` memuat data pelatihan dalam batch berukuran 64 dengan urutan acak (`shuffle=True`) untuk meningkatkan variasi selama pelatihan, sementara `test_loader` memuat data pengujian tanpa mengacak urutan (`shuffle=False`) untuk evaluasi yang konsisten. Parameter `num_workers=4` memungkinkan pemuatan data dilakukan secara paralel, mempercepat proses. `DataLoader` ini memastikan iterasi dataset berjalan efisien dan optimal untuk pelatihan serta pengujian model.

Fine-tuning the model

```
# Define the model
model = timm.create_model('resnet18', pretrained=True)
model.fc = nn.Linear(model.fc.in_features, 10)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret "HF_TOKEN" does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
model.safetensors: 100% 46.8M/46.8M [00:00<00:00, 161MB/s]
```

Kode di atas mendefinisikan model ResNet18 menggunakan pustaka `timm` dengan bobot pra-pelatihan (`pretrained=True`) dari dataset ImageNet. Model ini dimodifikasi untuk menyelesaikan tugas klasifikasi dengan 10 kelas pada dataset CIFAR-10. Modifikasi dilakukan dengan mengganti layer fully connected terakhir (`model.fc`) dengan layer baru menggunakan `nn.Linear`, yang memiliki jumlah neuron sesuai dengan jumlah kelas (10). Hal ini memungkinkan model memanfaatkan fitur yang telah dipelajari dari ImageNet sambil disesuaikan untuk dataset yang lebih kecil seperti CIFAR-10, sehingga mempercepat pelatihan dan meningkatkan akurasi melalui transfer learning.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

Kode di atas mendefinisikan `criterion` sebagai fungsi *loss* `CrossEntropyLoss` untuk mengukur kesalahan prediksi pada tugas klasifikasi, dan `optimizer` menggunakan **SGD** dengan *learning rate* 0.001 dan *momentum* 0.9 untuk memperbarui bobot model secara efisien selama pelatihan.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

Kode di atas menentukan perangkat yang akan digunakan untuk pelatihan model. Variabel `device` memeriksa ketersediaan GPU dengan `torch.cuda.is_available()`. Jika GPU tersedia, model akan menggunakan GPU (`cuda`) untuk mempercepat komputasi; jika tidak, model akan dijalankan di CPU. Perintah `model.to(device)` memindahkan model ke perangkat yang telah

ditentukan, memastikan semua operasi komputasi dilakukan pada perangkat tersebut untuk efisiensi pelatihan dan inferensi.

```
num_epochs = 30
for epoch in range(num_epochs):
    model.train()
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    # Validation
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = correct / total
    print(f'Epoch {epoch+1}/{num_epochs}, Loss: {loss:.4f}, Accuracy: {accuracy:.4f}')
```

Kode di atas menjalankan *training loop* untuk melatih dan mengevaluasi model selama 30 epoch. Pada setiap epoch, model dilatih menggunakan data dari `train_loader`. Prosesnya melibatkan pemindahan data ke perangkat (`device`), menghitung *loss* dengan `criterion`, melakukan *backpropagation* dengan `loss.backward()`, dan memperbarui bobot model menggunakan `optimizer.step()`. Setelah pelatihan setiap epoch, model dievaluasi dalam mode `eval()` menggunakan data dari `test_loader`, di mana akurasi dihitung berdasarkan jumlah prediksi yang benar dibandingkan dengan total data. Akhirnya, *loss* pelatihan dan akurasi validasi dicetak untuk memantau kinerja model di setiap epoch.

```
# Save the trained model
torch.save(model.state_dict(), 'resnet18_cifar10.pth')
```

Kode di atas menyimpan bobot model yang telah dilatih ke file `'resnet18_cifar10.pth'` menggunakan `torch.save`, memungkinkan model digunakan kembali tanpa melatih ulang.

```
from PIL import Image
```

```
import torchvision.transforms as transforms

# Load the trained model
model = timm.create_model('resnet18')
model.fc = nn.Linear(model.fc.in_features, 10) # Change the last
fully connected layer for CIFAR-10
model.load_state_dict(torch.load('resnet18_cifar10.pth'))
model.eval()
```

Kode di atas digunakan untuk memuat model **ResNet18** yang telah dilatih sebelumnya pada dataset CIFAR-10. Model didefinisikan dengan mengganti *fully connected layer* terakhir agar sesuai dengan 10 kelas CIFAR-10, lalu bobotnya dimuat dari file 'resnet18_cifar10.pth' menggunakan `model.load_state_dict()`. Perintah `model.eval()` mengatur model ke mode evaluasi, memastikan bahwa layer seperti dropout atau batch normalization bekerja dengan benar selama inferensi. Dengan demikian, model siap digunakan untuk membuat prediksi pada data baru.

```
image_transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

Kode di atas mendefinisikan transformasi untuk preprocessing gambar, termasuk mengubah ukuran ke 32x32 piksel, mengonversi ke tensor PyTorch, dan menormalkan nilai piksel ke rentang [-1, 1]. Transformasi ini memastikan gambar sesuai dengan format dan skala input model.

```
from PIL import Image
import requests
from io import BytesIO
url = "https://madbarn.com/wp-content/uploads/2023/12/List-of-Gaited-
Horse-Breeds.jpg"
response = requests.get(url)
image = Image.open(BytesIO(response.content))
```

Kode tersebut digunakan untuk mengunduh dan membuka gambar dari URL menggunakan Pillow. `requests.get()` mengambil konten gambar, yang kemudian diubah menjadi objek byte-stream dan dibuka dengan `Image.open()`. Namun, saat di running terjadi error `UnidentifiedImageError` dalam kasus ini, masalahnya tampaknya karena objek `response` memiliki kode status 403, yang berarti "Forbidden" atau "Dilarang". Ini menunjukkan bahwa server yang menghosting gambar menolak akses ke file gambar. Oleh karena itu, `PIL.Image.open()` tidak dapat membaca data gambar dari objek `BytesIO` karena kontennya bukanlah gambar melainkan pesan error dari server.

```
# image_path = '/content/images.jpeg'
# image = Image.open(image_path)
input_image = image_transform(image).unsqueeze(0) # Add batch
dimension
```

Kode ini memproses gambar dengan transformasi seperti mengubah ukuran, normalisasi, dan menambahkan dimensi batch menggunakan `unsqueeze(0)`, sehingga formatnya sesuai untuk inferensi model. Namun, codingan tersebut gagal di running karena terjadi error di codingan sebelumnya.

```
# Make prediction
import matplotlib.pyplot as plt
class_names = [
    'airplane', 'automobile', 'bird', 'cat', 'deer',
    'dog', 'frog', 'horse', 'ship', 'truck'
]
plt.imshow(image)
with torch.no_grad():
    model_output = model(input_image)
    _, predicted_class = torch.max(model_output, 1)

predicted_class_index = predicted_class.item()
predicted_class_name = class_names[predicted_class_index]

print(f'Predicted class: {predicted_class_index} -
{predicted_class_name}')
```

Kode ini memprediksi kelas gambar menggunakan model yang telah dilatih. Gambar ditampilkan dengan `plt.imshow(image)`, model menghasilkan prediksi, dan kelas dengan probabilitas tertinggi diambil menggunakan `torch.max()`. Nama kelas diperoleh dari daftar `class_names`, lalu hasil prediksi dicetak. Namun, codingan tersebut gagal di running karena terjadi error di codingan sebelumnya.

2. Transfer Learning and Fine-tuning

```
# General imports
import os # Untuk berinteraksi dengan sistem operasi, misalnya path,
direktori, dsb.
import time # Untuk mengukur waktu eksekusi (profiling) atau membuat
jeda (sleep).
import copy # Untuk menduplikasi objek (shallow / deep copy).
import numpy as np # Library untuk komputasi numerik, menyediakan
array multidimensi dan berbagai fungsi matematika.
import collections # Menghadirkan struktur data tambahan seperti
Counter, defaultdict, OrderedDict, dsb.

# Plotting imports
```



```

from PIL import Image # Untuk memuat dan memanipulasi berbagai format
gambar.
import matplotlib.pyplot as plt # Library plotting 2D dasar; cocok
untuk menampilkan grafik secara statis.
import plotly.graph_objects as go # Modul graph objects untuk
membuat plot interaktif dengan kontrol lebih detail.
from plotly.subplots import make_subplots # Memungkinkan membuat
layout subplot yang kompleks secara interaktif.
import plotly.express as px # Library "express" untuk menghasilkan
plot interaktif sederhana dan cepat.

# Torch imports
import torch # Kerangka kerja utama PyTorch untuk komputasi numerik
berbasis tensor.
import torch.nn as nn # Submodul PyTorch yang berisi komponen-komponen
neural network (layer, loss, dsb).
import torch.backends.cudnn as cudnn # Opsi backend CUDA (CUDNN) untuk
performa tinggi pada GPU.
import torch.optim as optim # Submodul untuk optimizer (SGD, Adam,
dsb).
from torch.optim.lr_scheduler import CosineAnnealingLR # Scheduler
untuk mengatur laju pembelajaran (learning rate) secara bertahap
menggunakan kurva cosinus.

# Torchvision imports
import torchvision # Paket yang menyediakan dataset, model, dan
transformasi umum untuk visi komputer.
from torchvision import datasets, models # Berisi dataset vision
populer (seperti CIFAR, ImageNet), serta model pra-terlatih (ResNet,
VGG, dsb).
from torchvision.transforms import v2 # Modul baru untuk transformasi
data (augmentasi, normalisasi, dsb) pada gambar versi 2.

plt.ion()

# selecting the available device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)

```

Kode ini merupakan persiapan untuk pengembangan model *machine learning*, mencakup pengelolaan data, visualisasi, dan pemrosesan komputasi. Library seperti `os` dan `time` membantu pengelolaan sistem serta pengukuran waktu eksekusi, sedangkan `numpy` digunakan untuk operasi numerik seperti manipulasi matriks atau statistik. Untuk visualisasi, `matplotlib` digunakan untuk grafik statis, dan `plotly` memungkinkan grafik interaktif. Modul `PyTorch` dan `Torchvision` dapat dimanfaatkan untuk fitur pendukung seperti transformasi data dan pengelolaan pipeline data, meskipun sebagian besar cocok untuk *deep learning*. Bagian

perangkat (device) menentukan apakah GPU (cuda) atau CPU yang akan digunakan, dengan output cuda menunjukkan bahwa GPU tersedia dan akan digunakan, yang bermanfaat untuk mempercepat komputasi berbasis tensor, terutama pada tugas *machine learning* yang berat seperti pengolahan data besar atau simulasi kompleks.

Data preparation and augmentation

```
# Classes in CIFAR-10 dataset (airplane, automobile, bird, cat, deer,
dog, frog, horse, ship, truck)
NUM_CLASSES = 10
# Image size for VGG19 (224x224)
IMAGE_SIZE = 224

mean, std = [0.4914, 0.4822, 0.4465], [0.247, 0.243, 0.261]

# Data transformations for training set
train_transforms = v2.Compose([
    v2.Resize((IMAGE_SIZE, IMAGE_SIZE)),
    v2.AutoAugment(policy=v2.AutoAugmentPolicy.CIFAR10),
    v2.ToTensor(),
    v2.Normalize(mean, std)
])

# Data transformations for validation set
validation_transforms = v2.Compose([
    v2.Resize((IMAGE_SIZE, IMAGE_SIZE)),
    v2.ToTensor(),
    v2.Normalize(mean, std)
])
```

Kode ini mendefinisikan transformasi data untuk dataset CIFAR-10, yang terdiri dari 10 kelas (seperti pesawat, mobil, burung, dll.), dengan mengubah gambar menjadi ukuran 224x224 untuk digunakan dengan model seperti VGG19. Transformasi untuk data pelatihan mencakup resizing, augmentasi otomatis menggunakan kebijakan *AutoAugment* untuk meningkatkan variasi data, konversi ke tensor, dan normalisasi menggunakan rata-rata (mean) dan standar deviasi (std) dari CIFAR-10. Transformasi untuk data validasi serupa, tetapi tanpa augmentasi, karena validasi bertujuan mengevaluasi model pada data tanpa perubahan tambahan. Transformasi ini memastikan data konsisten dan sesuai dengan input model.

```
# Dataset for training objectives
train_set = torchvision.datasets.CIFAR10(
    root='./data',
    train=True,
    download=True,
    transform=train_transforms
```

```
)

# Dataset for validation objectives
validation_set = torchvision.datasets.CIFAR10(
    root='./data',
    train=False,
    download=True,
    transform=validation_transforms
)

Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100% | 170M/170M [00:02<00:00, 59.9MB/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Kode ini memuat dataset CIFAR-10 untuk tujuan pelatihan dan validasi. **train_set** memuat data pelatihan dengan transformasi yang telah didefinisikan sebelumnya (**train_transforms**) untuk meningkatkan variasi data dan menyesuaikan input model. **validation_set** memuat data validasi dengan transformasi minimal (**validation_transforms**) untuk memastikan evaluasi dilakukan pada data yang tidak dimodifikasi secara berlebihan. Kedua dataset diunduh otomatis ke direktori `./data` jika belum tersedia. Dataset ini digunakan untuk membangun pipeline data yang konsisten dalam proses pelatihan dan validasi model.

```
# Number of images in each dataset
print('Train dataset size:', len(train_set))
print('Validation dataset size:', len(validation_set))

Train dataset size: 50000
Validation dataset size: 10000
```

Kode ini menghitung dan mencetak jumlah gambar dalam dataset pelatihan (**train_set**) dan dataset validasi (**validation_set**) menggunakan fungsi **len()**. Informasi ini penting untuk memahami ukuran dataset yang digunakan, membantu dalam perencanaan proses pelatihan seperti menentukan ukuran batch, jumlah iterasi, dan mengevaluasi keseimbangan data. Output ini memberikan gambaran tentang jumlah data yang tersedia untuk melatih dan menguji model.

```
def plot_class_distribution(dataset, dataset_name):
    """
    Print and plot the class distribution of a dataset.

    Args:
        dataset (torch.utils.data.Dataset): The dataset to analyze.
        dataset_name (str): The name of the dataset.

    Returns:
        None
```

```

"""

# Extract labels from the dataset
labels = [y for _, y in dataset]

# Count the number of images per category
counter = collections.Counter(labels)

# Print class image counter
print(f"Class Image Counter for {dataset_name} Data")
print(counter, "\n")

# Optionally, plot the class distribution
plt.bar(counter.keys(), counter.values())
plt.xlabel("Class")
plt.ylabel("Number of Images")
plt.title(f"Class Distribution for {dataset_name} Data")
plt.show()

```

Fungsi ini digunakan untuk menganalisis dan memvisualisasikan distribusi kelas dalam sebuah dataset. Fungsi mengambil dataset dan nama dataset sebagai argumen, kemudian mengekstrak label dari dataset untuk menghitung jumlah gambar di setiap kelas menggunakan **collections.Counter**. Distribusi jumlah gambar per kelas ditampilkan dalam bentuk teks dan divisualisasikan sebagai grafik batang. Fungsi ini membantu memeriksa keseimbangan data antar kelas, yang penting untuk memastikan model dilatih secara adil dan menghindari bias terhadap kelas tertentu.

```

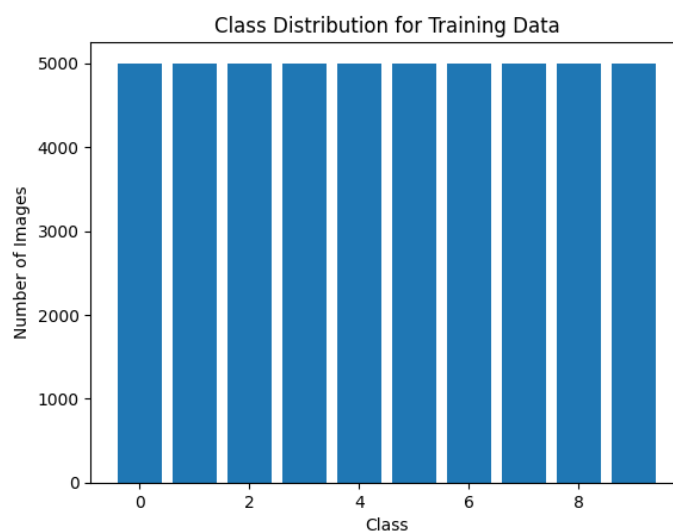
# Train dataset and test dataset (all classes) class distribution
plot_class_distribution(train_set, "Training")
plot_class_distribution(validation_set, "Validation")

```

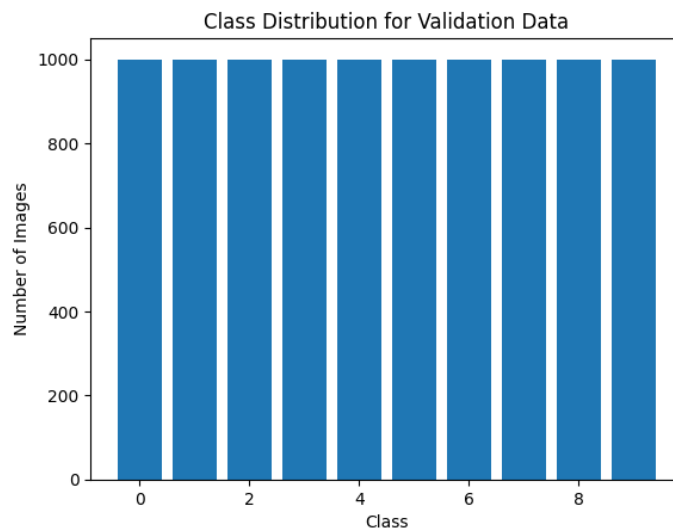
```

🔍 Class Image Counter for Training Data
Counter({6: 5000, 9: 5000, 4: 5000, 1: 5000, 2: 5000, 7: 5000, 8: 5000, 3: 5000, 5: 5000, 0: 5000})

```



```
Class Image Counter for Validation Data
Counter({3: 1000, 8: 1000, 0: 1000, 6: 1000, 1: 1000, 9: 1000, 5: 1000, 7: 1000, 4: 1000, 2: 1000})
```



Kode ini digunakan untuk memvisualisasikan distribusi kelas dalam dataset pelatihan dan validasi CIFAR-10. Fungsi **plot_class_distribution** menghitung jumlah gambar dalam setiap kelas menggunakan `collections.Counter`, kemudian menampilkan hasilnya dalam bentuk teks dan grafik batang. Distribusi yang seimbang, seperti yang terlihat pada output, memastikan bahwa setiap kelas memiliki jumlah gambar yang sama, yaitu 5.000 gambar per kelas untuk data pelatihan dan 1.000 gambar per kelas untuk data validasi. Hal ini penting untuk mencegah bias model terhadap kelas tertentu selama pelatihan.

Output:

1. **Training Data:** Semua 10 kelas memiliki jumlah yang sama, yaitu 5.000 gambar per kelas. Distribusi ini mendukung pelatihan model yang adil.
2. **Validation Data:** Sama seperti data pelatihan, semua kelas memiliki distribusi yang merata dengan 1.000 gambar per kelas. Hal ini memastikan evaluasi model dilakukan secara seimbang.

Grafik batang memperjelas bahwa dataset CIFAR-10 memiliki distribusi data yang seimbang untuk setiap kelas pada kedua subset (training dan validation).

```
# Define the batch size for processing samples per forward/backward
pass
batch_size = 128
```

Batch size sebesar 128 menentukan jumlah sampel yang diproses sekaligus dalam satu langkah pelatihan, memanfaatkan efisiensi komputasi dan paralelisme, serta memengaruhi kecepatan, stabilitas, dan penggunaan memori selama pelatihan model.

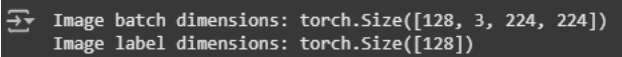
```
num_gpus = torch.cuda.device_count()

# DataLoader for training set
train_loader = torch.utils.data.DataLoader(
    train_set,
    batch_size=batch_size,
    shuffle=True,
    num_workers=2*num_gpus,
    pin_memory=True
)

# DataLoader for validation set
validation_loader = torch.utils.data.DataLoader(
    validation_set,
    batch_size=batch_size,
    shuffle=False,
    num_workers=2*num_gpus,
    pin_memory=True
)
```

`DataLoader` digunakan untuk memuat data pelatihan dan validasi secara efisien. **train_loader** mengacak data untuk pelatihan, sedangkan **validation_loader** memuat data tanpa pengacakan untuk evaluasi. Pengaturan **num_workers** memanfaatkan paralelisme GPU, dan **pin_memory=True** mempercepat transfer data ke GPU, mengoptimalkan proses pelatihan dan validasi.

```
# Checking the dataset
for images, labels in train_loader:
    print('Image batch dimensions:', images.shape)
    print('Image label dimensions:', labels.shape)
    break
```



```
Image batch dimensions: torch.Size([128, 3, 224, 224])
Image label dimensions: torch.Size([128])
```

Kode ini memeriksa dimensi batch data yang dimuat oleh **train_loader** untuk memastikan data telah diolah sesuai spesifikasi model. Iterasi pertama dari `DataLoader` mencetak dimensi batch gambar (**images.shape**) dan label (**labels.shape**).

Output:

- **Image batch dimensions: torch.Size([128, 3, 224, 224]):** Menunjukkan batch terdiri dari 128 gambar, masing-masing dengan 3 saluran warna (RGB) dan ukuran 224x224 piksel.
- **Image label dimensions: torch.Size([128]):** Menunjukkan batch memiliki 128 label, masing-masing terkait dengan satu gambar.

Ini memastikan data batch sudah sesuai format input model untuk pelatihan.

Visualize a few images

```
def imshow(tensor, title=None):
    """
    Display a batch of images in a grid.

    Args:
        tensor (torch.Tensor): The input tensor containing the images.
        title (str, optional): The title of the plot. Defaults to
None.
    """
    image = torchvision.utils.make_grid(tensor).numpy().transpose((1,
2, 0))

    # Denormalize the image
    mean = np.array([0.4914, 0.4822, 0.4465])
    std = np.array([0.247, 0.243, 0.261])
    image = std * image + mean
    image = np.clip(image, 0, 1)

    # Increase the plot size
    plt.figure(figsize=(10, 10))

    # Plot the image
    plt.imshow(image)
    if title:
        plt.title(title)
    plt.pause(0.001) # pause a bit so that plots are updated
```

Fungsi **imshow** digunakan untuk menampilkan gambar dalam bentuk grid dari batch tensor data. Tensor gambar diubah menjadi format grid menggunakan **torchvision.utils.make_grid**, kemudian di-*denormalize* menggunakan nilai rata-rata (**mean**) dan standar deviasi (**std**) dataset CIFAR-10 agar warna asli gambar dapat terlihat. Gambar ditampilkan menggunakan **matplotlib** dengan ukuran yang diperbesar. Fungsi ini mempermudah visualisasi data gambar yang sedang dilatih atau diuji, sehingga membantu memeriksa kualitas preprocessing atau augmentasi data.

```
# Image classes in CIFAR-10 dataset
classes = (
    "plane",
    "car",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck",
)

# Get a batch of training data
for inputs, labels in train_loader:
    # Display the first 8 images from the batch
    imshow(inputs[:5], title=[classes[x] for x in labels[:5]])

    break # Break the loop after it finishes
```



Kode ini digunakan untuk menampilkan contoh gambar beserta labelnya dari batch data pelatihan CIFAR-10. **classes** mendefinisikan nama kelas dalam dataset, dan batch gambar pertama diambil dari **train_loader**. Fungsi **imshow** digunakan untuk menampilkan 5 gambar pertama dari batch dengan judul yang mencantumkan nama kelasnya sesuai label. Kode ini membantu memverifikasi bahwa gambar dan label telah dimuat serta diproses dengan benar sebelum pelatihan dimulai.

Model training

```
# Set the learning rate for the optimizer
learning_rate = 1e-4
```

Kode ini menetapkan **learning rate** 10^{-4} , mengontrol ukuran langkah pembaruan bobot model untuk memastikan pelatihan stabil dan efisien.

```
def get_lr(opt):
    """
```



```
    Helper function to get the current learning rate from the
optimizer.
```

```
    Args:
```

```
        opt (Optimizer): The optimizer object.
```

```
    Returns:
```

```
        float: The current learning rate.
```

```
    """
```

```
    for param_group in opt.param_groups:
        return param_group["lr"]
```

Fungsi **get_lr** digunakan untuk mendapatkan nilai **learning rate** saat ini dari optimizer. Fungsi ini mengakses **param_groups** dalam optimizer untuk mengambil nilai **lr**, memudahkan pemantauan atau logging perubahan learning rate selama pelatihan model.

```
def metrics_batch(output, target):
```

```
    """
```

```
    Helper function to count the number of correct predictions in a
batch.
```

```
    Args:
```

```
        output (torch.Tensor): Model predictions.
```

```
        target (torch.Tensor): Target labels.
```

```
    Returns:
```

```
        int: Number of correct predictions.
```

```
    """
```

```
    # Get the predicted class for each example in the batch
```

```
    pred = output.argmax(dim=1, keepdim=True)
```

```
    # Compare predicted class with target class and count correct
predictions
```

```
    corrects = pred.eq(target.view_as(pred)).sum().item()
```

```
    return corrects
```

```
def loss_batch(loss_func, model_output, target, optimizer=None):
```

```
    """
```

```
    Helper function to compute the loss value per batch of data.
```

```
    Args:
```

```
        loss_func: Loss function.
```

```
        model_output (torch.Tensor): Model predictions.
```

```
        target (torch.Tensor): Target labels.
```

```
        optimizer: Optimizer for backpropagation (optional).
```

```
    Returns:
```

```
        tuple: Loss value, number of correct predictions.
```

```
    """
```

```

    # Calculate the loss using the specified loss function
    loss_value = loss_func(model_output, target)

    # Get performance metric (number of correct predictions)
    metric_batch = metrics_batch(model_output, target)

    if optimizer is not None:
        # Backpropagation and optimization step
        optimizer.zero_grad()
        #loss_value.requires_grad = True
        loss_value.backward()
        optimizer.step()

    return loss_value.item(), metric_batch

def loss_epoch(model, loss_func, data_loader, check_id=False,
optimizer=None):
    """
    Compute the average loss value and performance metric over an
    epoch.

    Args:
        model: The neural network model.
        loss_func: Loss function.
        data_loader: DataLoader for the dataset.
        check_id (bool): Flag to check only the first batch.
        optimizer: Optimizer for backpropagation (optional).

    Returns:
        tuple: Average loss value, average performance metric.
    """
    running_loss = 0.0
    running_metric = 0.0
    total_samples = len(data_loader.dataset)

    # Internal loop over batches in the DataLoader
    for inputs, labels in data_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        model_output = model(inputs)
        # Calculate loss and performance metric for the batch
        loss_batch_value, metric_batch_value = loss_batch(
            loss_func, model_output, labels, optimizer
        )
        running_loss += loss_batch_value

        if metric_batch_value is not None:
            running_metric += metric_batch_value

```

```

        if check_id:
            break # Stop if only checking the first batch

    # Compute the average loss and performance metric over the entire
epoch
    average_loss = running_loss / float(total_samples)
    average_metric = (
        running_metric / float(total_samples) if total_samples > 0
else None
    )

    return average_loss, average_metric

```

Fungsi-fungsi ini menghitung metrik evaluasi dan loss untuk batch atau epoch. **metrics_batch** menghitung prediksi yang benar, **loss_batch** menghitung loss dan memperbarui bobot jika ada optimizer, dan **loss_epoch** menghitung rata-rata loss dan metrik untuk seluruh epoch, mempermudah pelatihan dan evaluasi model.

Training loop

```

def train_val(model, params, verbose=False):
    """
    Trains and validates a model using the given parameters.

    Args:
        model (torch.nn.Module): The model to be trained and
validated.
        params (dict): A dictionary containing the parameters for
training and validation.
            - epochs (int): The number of epochs to train the model.
            - loss_func (torch.nn.Module): The loss function to be
used.
            - optimiser (torch.optim.Optimizer): The optimizer to be
used for training.
            - train_dl (torch.utils.data.DataLoader): The data loader
for the training dataset.
            - val_dl (torch.utils.data.DataLoader): The data loader
for the validation dataset.
            - check_id (int): The ID for checking the loss and metric.
            - lr_scheduler (torch.optim.lr_scheduler._LRScheduler):
The learning rate scheduler.
            - path (str): The path to save the best model weights.

        verbose (bool, optional): Whether to print additional
information during training. Defaults to False.

    Returns:

```

```

        tuple: A tuple containing the trained model, loss history, and
metric history.
        - model (torch.nn.Module): The trained model.
        - loss_history (dict): A dictionary containing the loss
values for each epoch.
        - metric_history (dict): A dictionary containing the
metric values for each epoch.
    """

    # extract model parameters
    epochs = params["epochs"]
    loss_func = params["loss_func"]
    opt = params["optimizer"]
    train_dl = params["train_dl"]
    val_dl = params["val_dl"]
    check_id = params["check_id"]
    lr_scheduler = params["lr_scheduler"]
    path = params["path"]

    loss_history = {"train": [], "val": []} # history of loss values
in each epoch
    metric_history = {"train": [], "val": []} # histroy of metric
values in each epoch
    best_model_wts = copy.deepcopy(model.state_dict()) # copy weights
for best model
    best_loss = float("inf") # initialize best loss to a large value

    # main loop
    for epoch in range(epochs):
        current_lr = get_lr(opt) # get current learning rate
        if verbose:
            print(f"Epoch {epoch}/{epochs-1}, current
lr={current_lr}")

        # train model on training dataset
        model.train()
        train_loss, train_metric = loss_epoch(model, loss_func,
train_dl, check_id, opt)

        # collect loss and metric for training dataset
        loss_history["train"].append(train_loss)
        metric_history["train"].append(train_metric)

        # evaluate model on validation dataset
        model.eval()
        with torch.no_grad():
            val_loss, val_metric = loss_epoch(model, loss_func,
val_dl, check_id)

```

```

# store best model
if val_loss < best_loss:
    best_loss = val_loss
    best_model_wts = copy.deepcopy(model.state_dict())

    # store weights into a local file
    torch.save(model.state_dict(), path)
    if verbose:
        print("Copied best model weights!")

# collect loss and metric for validation dataset
loss_history["val"].append(val_loss)
metric_history["val"].append(val_metric)

# learning rate schedule
lr_scheduler.step()

if verbose:
    print(
        f"train loss: {train_loss:.6f}, train accuracy: {100*train_metric:.2f}, validation loss: {val_loss:.6f}, validation accuracy: {100*val_metric:.2f}"
    )
    print("")

# load best model weights
model.load_state_dict(best_model_wts)

return model, loss_history, metric_history

```

Fungsi **train_val** melatih dan memvalidasi model dengan parameter yang ditentukan. Fungsi ini menjalankan proses pelatihan selama beberapa epoch, mencatat loss dan metrik kinerja untuk data pelatihan dan validasi di setiap epoch, serta menyimpan bobot model terbaik berdasarkan loss validasi terendah. Fungsi juga mengatur *learning rate* menggunakan scheduler dan memungkinkan logging tambahan jika **verbose** diaktifkan. Hasil akhirnya adalah model terlatih beserta riwayat loss dan metrik kinerjanya.

Display the model's metrics

```

def plot_out(loss_hist, metric_hist, epochs=None):
    """
    Plot training and validation loss, and training and validation
    metrics over epochs.

    Args:

```

```

        loss_hist (dict): Dictionary containing 'train' and 'val' keys
for loss history.
        metric_hist (dict): Dictionary containing 'train' and 'val'
keys for metric history.
        epochs (int): Number of epochs.
    """

    # Create subplot with two columns
    fig = make_subplots(
        rows=1, cols=2, subplot_titles=["Model Loss", "Model
Accuracy"]
    )

    # Plot Loss History
    for phase, color in zip(["train", "val"], ["#F1C40F", "#232323"]):
        fig.add_trace(
            go.Scatter(
                x=list(range(1, epochs + 1)),
                y=loss_hist[phase],
                name=phase,
                mode="lines",
                line_color=color,
            ),
            row=1,
            col=1,
        )

    # Plot Metric History
    for phase, color in zip(["train", "val"], ["#F1C40F", "#232323"]):
        fig.add_trace(
            go.Scatter(
                x=list(range(1, epochs + 1)),
                y=metric_hist[phase],
                name=phase,
                mode="lines",
                line_color=color,
            ),
            row=1,
            col=2,
        )

    # Update layout and display the plot
    fig.update_layout(
        template="plotly_white",
        showlegend=False,
        title="Loss & Accuracy History",
        height=400,
    )

```

```
fig.update_layout(yaxis2=dict(range=[0.4, 1]))
fig.show()
```

Fungsi `plot_out` memvisualisasikan riwayat *loss* dan metrik kinerja (misalnya, akurasi) selama pelatihan dan validasi model. Dengan menggunakan *Plotly*, fungsi ini membuat grafik interaktif yang menampilkan perubahan *loss* dan metrik untuk data pelatihan dan validasi di setiap epoch. Grafik ini membantu memantau proses pelatihan, seperti konvergensi model, overfitting, atau underfitting, sehingga memudahkan analisis dan penyesuaian parameter pelatihan.

Create the base model from the pre-trained convnets

```
# Create the base model from the pre-trained model VGG19
pre_vgg19 = models.vgg19(weights='IMAGENET1K_V1')
pre_vgg19
```

Kode ini memuat model **VGG19** pra-terlatih dengan bobot dari dataset **ImageNet (IMAGENET1K_V1)**. Model pra-terlatih ini digunakan sebagai dasar (*base model*) untuk tugas pembelajaran transfer, memungkinkan pengguna memanfaatkan fitur-fitur yang sudah dipelajari VGG19, sehingga mempercepat pelatihan dan meningkatkan akurasi pada dataset baru tanpa harus melatih model dari awal.

Feature extraction

```
# change the number of classes
pre_vgg19.classifier[6].out_features = NUM_CLASSES

# freeze convolution weights
for param in pre_vgg19.features.parameters():
    param.requires_grad = False
```

Kode ini menyesuaikan **VGG19** untuk dataset baru dengan mengubah jumlah output menjadi **NUM_CLASSES** dan membekukan bobot lapisan konvolusi, sehingga hanya melatih bagian *classifier* untuk mempercepat pelatihan dan mencegah overfitting.

Train and evaluate

```
# Send earlier defined model to device
device = torch.device("cuda:0")
pre_vgg19 = pre_vgg19.to(device)

# Loss function, optimizer, LR scheduler
loss_func = nn.CrossEntropyLoss(reduction="sum")
optimizer = optim.Adam(pre_vgg19.parameters(), lr=learning_rate)
lr_scheduler = CosineAnnealingLR(optimizer, T_max=5, eta_min=1e-6)
```

```
# Set Training Parameters
training_params = {
    "epochs": 10,
    "optimizer": optimizer,
    "loss_func": loss_func,
    "train_dl": train_loader,
    "val_dl": validation_loader,
    "check_id": False,
    "lr_scheduler": lr_scheduler,
    "path": "pre_vgg19.pt",
}

# Train and validate the model
pre_vgg19, loss_history, metric_history = train_val(
    pre_vgg19, training_params, verbose=False
)
```

Kode ini melatih model **VGG19** di GPU menggunakan **CrossEntropyLoss**, optimizer **Adam**, dan scheduler **CosineAnnealingLR**. Parameter pelatihan, seperti jumlah epoch, DataLoader, dan path penyimpanan model terbaik, diatur dalam dictionary. Fungsi **train_val** melatih model, memvalidasi performa, dan mengembalikan model terlatih beserta riwayat loss dan metriknya.

Learning curves

```
# Plot History
plot_out(loss_history, metric_history,
epochs=training_params["epochs"])
```



Kode ini memplot riwayat *loss* dan metrik kinerja selama pelatihan dan validasi model menggunakan fungsi **plot_out**. Grafik tersebut membantu memvisualisasikan perkembangan *loss* dan akurasi model selama beberapa epoch, memudahkan analisis performa model serta identifikasi potensi overfitting atau underfitting. Output tersebut adalah grafik riwayat *loss* dan akurasi untuk data pelatihan dan validasi selama 10 epoch. Grafik *loss* menunjukkan bahwa *train loss* (garis kuning) menurun secara signifikan, menandakan model semakin baik meminimalkan error pada data pelatihan, sementara *val loss* (garis hitam) stabil pada nilai

rendah, menunjukkan model tidak overfitting. Pada grafik akurasi, *train accuracy* meningkat mendekati stabilitas, dan *val accuracy* tetap konsisten tinggi, mencerminkan kemampuan generalisasi yang baik. Secara keseluruhan, model menunjukkan pelatihan yang efektif dan mampu bekerja baik pada data baru.

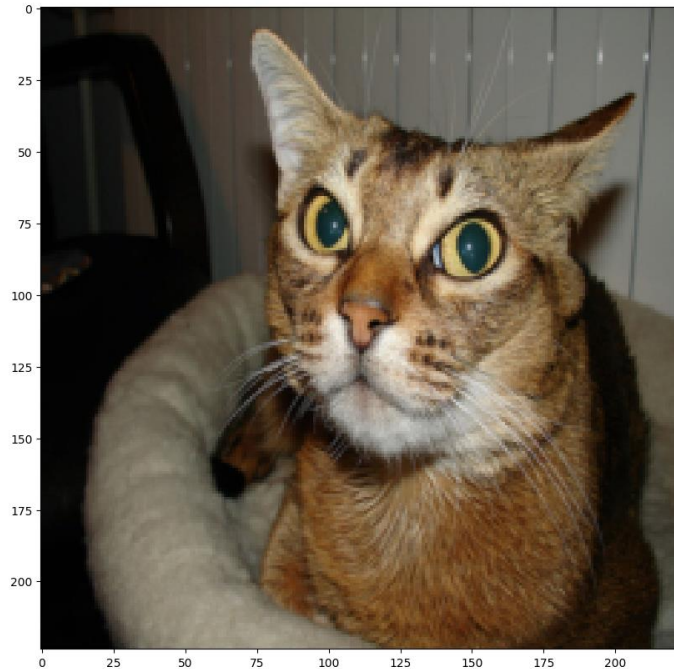
Inference on custom images

```
def visualize_model_predictions(model, img_path):  
    """  
    Visualizes model predictions on a single image.  
  
    Args:  
        model (torch.nn.Module): The trained model.  
        img_path (str): Path to the input image.  
  
    Returns:  
        None  
    """  
    # Set the model to evaluation mode  
    was_training = model.training  
    model.eval()  
  
    # Load and preprocess the input image  
    img = Image.open(img_path)  
    img = validation_transforms(img)  
    img = img.unsqueeze(0)  
    img = img.to(device)  
  
    # Perform forward pass to obtain predictions  
    with torch.no_grad():  
        outputs = model(img)  
        _, preds = torch.max(outputs, 1)  
  
        # Display the input image and predicted class  
        ax = plt.subplot(2, 2, 1)  
        ax.axis("off")  
        ax.set_title(f"Predicted: {classes[preds[0]]}")  
        imshow(img.cpu().data[0])  
  
    # Set the model back to its original training mode  
    model.train(mode=was_training)
```

Fungsi **visualize_model_predictions** menampilkan gambar input beserta label prediksi model. Gambar diproses, dievaluasi oleh model, dan ditampilkan dengan label prediksi, memudahkan interpretasi hasil model pada satu gambar.

```
# Visualize model predictions
```

```
visualize_model_predictions(pre_vgg19,  
img_path="/content/00000001_020.jpg")  
  
plt.ioff()  
plt.show()
```



Kode ini menggunakan fungsi **visualize_model_predictions** untuk memuat gambar dari path **/content/00000001_020.jpg**, memprosesnya, dan menampilkan gambar tersebut dengan label kelas prediksi dari model **VGG19**. Hal ini memudahkan pengguna untuk memverifikasi dan menganalisis kemampuan model dalam mengklasifikasikan gambar tertentu.

Chapter 3

1. DETR in transformers

```
%pip install timm
```

Perintah `%pip install timm` digunakan untuk menginstal library *timm* (*PyTorch Image Models*), yang menyediakan koleksi model visi komputer pra-terlatih, termasuk berbagai arsitektur *state-of-the-art*. Library ini mempermudah penggunaan, pelatihan ulang, atau fine-tuning model untuk tugas pengenalan gambar atau visi komputer lainnya.

```
# this example is mainly from https://huggingface.co/facebook/detr-  
resnet-50  
  
from transformers import DetrImageProcessor, DetrForObjectDetection  
import torch
```

```
from PIL import Image, ImageDraw
import requests
```

Kode ini mempersiapkan modul untuk memproses gambar dan menggunakan model **DETR** dari **Hugging Face** untuk deteksi objek. **DetrImageProcessor** memproses gambar, **DetrForObjectDetection** memuat model DETR, sementara **PIL** dan **requests** digunakan untuk memuat, menggambar, atau mengunduh gambar. Kombinasi ini memungkinkan penerapan deteksi objek secara langsung pada gambar input.

```
url = "http://images.cocodataset.org/val2017/000000039769.jpg"
image = Image.open(requests.get(url, stream=True).raw)
```

Kode ini mengambil gambar dari URL menggunakan **requests** dan memuatnya ke dalam format **PIL Image** menggunakan **Image.open**. Gambar tersebut kemudian siap untuk diproses lebih lanjut, seperti diterapkan pada model deteksi objek. Hal ini memungkinkan pemrosesan langsung gambar dari internet tanpa perlu menyimpannya secara lokal.

Model

```
model = DetrForObjectDetection.from_pretrained("facebook/detr-resnet-50")
```

Kode ini memuat model **DETR (Detection Transformer)** pra-terlatih dengan arsitektur **ResNet-50** dari **Hugging Face** menggunakan fungsi **from_pretrained**. Model ini siap digunakan untuk tugas deteksi objek tanpa perlu melatihnya dari awal, memanfaatkan bobot yang telah dilatih pada dataset besar seperti COCO.

Processor

```
processor = DetrImageProcessor.from_pretrained("facebook/detr-resnet-50")
inputs = processor(images=image, return_tensors="pt")
```

Kode ini memuat **DetrImageProcessor** pra-terlatih dari **Hugging Face** untuk memproses gambar menjadi format tensor PyTorch yang sesuai dengan model **DETR**. Gambar diubah melalui normalisasi, perubahan ukuran, dan transformasi lain agar kompatibel dengan input model, kemudian dikembalikan dalam format tensor dengan **return_tensors="pt"**. Hal ini mempermudah pipeline deteksi objek.

Outputs

```
outputs = model(**inputs)
```

Kode ini menghasilkan prediksi model DETR, termasuk kelas objek (**logits**) dan koordinat *bounding box* (**pred_boxes**) dari input yang telah diproses.

```
target_sizes = torch.tensor([image.size[:-1]])
```

```

results = processor.post_process_object_detection(outputs,
target_sizes=target_sizes, threshold=0.9)[0]

for score, label, box in zip(results["scores"], results["labels"],
results["boxes"]):
    box = [round(i, 2) for i in box.tolist()]
    print(
        f"Detected {model.config.id2label[label.item()]} with
confidence "
        f"{round(score.item(), 3)} at location {box}"
    )

```

```

➦ Detected remote with confidence 0.998 at location [40.16, 70.81, 175.55, 117.98]
Detected remote with confidence 0.996 at location [333.24, 72.55, 368.33, 187.66]
Detected couch with confidence 0.995 at location [-0.02, 1.15, 639.73, 473.76]
Detected cat with confidence 0.999 at location [13.24, 52.05, 314.02, 470.93]
Detected cat with confidence 0.999 at location [345.4, 23.85, 640.37, 368.72]

```

Kode ini memproses output model DETR untuk mendeteksi objek dalam gambar. **post_process_object_detection** memetakan prediksi model ke ukuran asli gambar dengan ambang kepercayaan 0.9. Loop **for** menampilkan setiap objek yang terdeteksi dengan mencetak nama kelas, tingkat kepercayaan, dan koordinat *bounding box*-nya. Kode ini membantu menginterpretasikan hasil deteksi objek secara langsung.

```

img_draw = ImageDraw.Draw(image)
for score, label, box in zip(results["scores"], results["labels"],
results["boxes"]):
    img_draw.rectangle((box[0], box[1]), (box[2], box[3]),
outline='Red')
    img_draw.text((box[0], box[1]),
model.config.id2label[label.item()], align="left")

```

Kode ini menggambar *bounding box* berwarna merah di sekitar objek yang terdeteksi dalam gambar dan menambahkan label nama kelas objek di sudut kiri atas *bounding box*. Hal ini memvisualisasikan hasil deteksi objek dari model DETR secara langsung pada gambar, mempermudah interpretasi dan analisis hasil deteksi.

```

display(image)

```



Kode **display(image)** digunakan untuk menampilkan gambar dengan *bounding box* dan label yang telah digambar sebelumnya. Ini memungkinkan visualisasi hasil deteksi objek secara langsung, sehingga pengguna dapat memeriksa kualitas deteksi model pada gambar input.

2. Fine-tuning Vision Transformers for Object Detection

Installation

```
!pip install -U -q datasets transformers[torch] evaluate timm  
alumentations accelerate
```

Kode ini menginstal atau memperbarui pustaka penting seperti datasets, transformers[torch], evaluate, timm, alumentations, dan accelerate untuk pemrosesan dataset, pelatihan model, dan evaluasi metrik, dengan opsi untuk memperbarui dan menonaktifkan output yang tidak perlu.

Datasets

```
from datasets import load_dataset  
  
dataset = load_dataset("anindya64/hardhat")  
dataset
```

Kode ini digunakan untuk memuat dataset "hardhat" dari Hugging Face menggunakan fungsi load_dataset. Dataset ini disimpan di repositori pengguna "anindya64" di Hugging Face. Setelah pemuatan, dataset akan tersedia dalam bentuk objek yang dapat diakses dan digunakan untuk analisis atau pelatihan model.

```
dataset['train'][0]
```

```
{'image': <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x375>,
 'image_id': 1,
 'width': 500,
 'height': 375,
 'objects': {'id': [1, 1],
 'area': [3068.0, 690.0],
 'bbox': [[178.0, 84.0, 52.0, 59.0], [111.0, 144.0, 23.0, 30.0]],
 'category': ['helmet', 'helmet']]}
```

Kode ini mengakses data pertama dari subset 'train' pada dataset, yang berisi informasi atau gambar untuk digunakan dalam analisis atau pelatihan model.

```
# extract out the train and test set
```

```
train_dataset = dataset['train']
test_dataset = dataset['test']
```

Kode ini mengekstrak subset 'train' dan 'test' dari dataset yang telah dimuat, menyimpannya masing-masing dalam variabel train_dataset dan test_dataset. Kedua variabel ini kemudian dapat digunakan secara terpisah untuk pelatihan dan evaluasi model machine learning.

```
sample = train_dataset[0]
sample
```

```
{'image': <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x375>,
 'image_id': 1,
 'width': 500,
 'height': 375,
 'objects': {'id': [1, 1],
 'area': [3068.0, 690.0],
 'bbox': [[178.0, 84.0, 52.0, 59.0], [111.0, 144.0, 23.0, 30.0]],
 'category': ['helmet', 'helmet']]}
```

Kode ini mengambil sampel pertama dari train_dataset dan menyimpannya dalam variabel sample. Output yang diberikan menunjukkan informasi tentang gambar dan objek yang terdeteksi di dalamnya. Gambar tersebut berukuran 500x375 piksel dengan dua objek yang masing-masing adalah helmet. Setiap objek memiliki informasi tambahan seperti id, area, dan bbox yang mendeskripsikan posisi dan ukuran objek dalam gambar. bbox berisi koordinat bounding box untuk setiap objek.

```
def draw_image_from_idx(dataset, idx):
    sample = dataset[idx]
    image = sample["image"]
    annotations = sample['objects']
    draw = ImageDraw.Draw(image)
    width, height = sample['width'], sample['height']

    for i in range(len(annotations["id"])):
        box = annotations["bbox"][i]
        class_idx = annotations["id"][i]
        x, y, w, h = tuple(box)
        if max(box) > 1.0:
```

```

        x1, y1 = int(x), int(y)
        x2, y2 = int(x + w), int(y + h)
    else:
        x1 = int(x * width)
        y1 = int(y * height)
        x2 = int((x + w) * width)
        y2 = int((y + h) * height)
    draw.rectangle((x1, y1, x2, y2), outline="red", width=1)
    draw.text((x1, y1), annotations["category"][i], fill="white")
return image

```

```
draw_image_from_idx(dataset=train_dataset, idx=10)
```



Fungsi **draw_image_from_idx** menggambar bounding box dan label objek pada gambar berdasarkan indeks **idx**. Fungsi ini mengonversi koordinat relatif ke piksel, lalu menggambar kotak pembatas dan label untuk setiap objek yang terdeteksi. Outputnya adalah gambar dengan objek yang diberi kotak dan label.

```

# Now let's make a simple function on plotting multiple images

import matplotlib.pyplot as plt

def plot_images(dataset, indices):
    """
    Plot images and their annotations.
    """
    num_rows = len(indices) // 3
    num_cols = 3
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))

    for i, idx in enumerate(indices):
        row = i // num_cols
        col = i % num_cols

        # Draw image

```



```

image = draw_image_from_idx(dataset, idx)

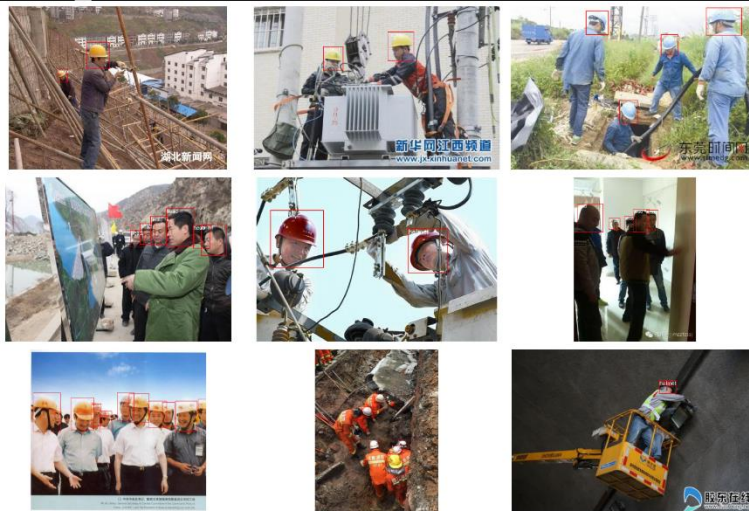
# Display image on the corresponding subplot
axes[row, col].imshow(image)
axes[row, col].axis('off')

plt.tight_layout()
plt.show()

```

Fungsi **plot_images** menampilkan beberapa gambar dari dataset dalam grid, dengan bounding box dan label objek yang digambar menggunakan **draw_image_from_idx**. Outputnya adalah grid gambar dengan anotasi objek.

```
plot_images(train_dataset, range(9))
```



Kode ini memanggil fungsi **plot_images** untuk menampilkan 9 gambar pertama dari **train_dataset** dalam format grid 3x3. Setiap gambar akan digambar dengan bounding box dan label objek yang terdeteksi. Outputnya adalah grid 3x3 yang berisi gambar dengan objek yang diberi kotak pembatas dan labelnya.

AutoImageProcessor

```

from transformers import AutoImageProcessor

checkpoint = "facebook/detr-resnet-50-dc5"
image_processor = AutoImageProcessor.from_pretrained(checkpoint)

```

Kode ini memuat **AutoImageProcessor** dari pustaka **transformers** untuk memproses gambar dengan model **DETR (Detection Transformer)** yang telah dilatih sebelumnya menggunakan checkpoint **facebook/detr-resnet-50-dc5**. **AutoImageProcessor** digunakan untuk mempersiapkan gambar sebelum diberikan ke model DETR untuk deteksi objek.

Preprocessing the dataset


```
# now creating random augmentations using albumentations

import albumentations
import numpy as np
import torch

transform = albumentations.Compose(
    [
        albumentations.Resize(480, 480),
        albumentations.HorizontalFlip(p=1.0),
        albumentations.RandomBrightnessContrast(p=1.0),
    ],
    bbox_params=albumentations.BboxParams(format="coco",
label_fields=["category"]),
)
```

Kode ini menggunakan **albumentations** untuk menerapkan augmentasi pada gambar, seperti **resize**, **horizontal flip**, dan **penyesuaian kecerahan/kontras**. Augmentasi juga diterapkan pada bounding box objek dengan format **COCO**.

```
def formatted_anns(image_id, category, area, bbox):
    annotations = []
    for i in range(0, len(category)):
        new_ann = {
            "image_id": image_id,
            "category_id": category[i],
            "isCrowd": 0,
            "area": area[i],
            "bbox": list(bbox[i]),
        }
        annotations.append(new_ann)

    return annotations
```

Fungsi **formatted_anns** memformat anotasi objek ke dalam format COCO, mencakup ID gambar, kategori, area, dan bounding box, lalu mengembalikannya sebagai daftar anotasi.

```
sample
{
  'image': <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x375>,
  'image_id': 1,
  'width': 500,
  'height': 375,
  'objects': {
    'id': [1, 1],
    'area': [3068.0, 690.0],
    'bbox': [[178.0, 84.0, 52.0, 59.0], [111.0, 144.0, 23.0, 30.0]],
    'category': ['helmet', 'helmet']}
}
```

Kodingan ini mengambil sample dari dataset yang berisi gambar dan anotasi terkait. Outputnya menunjukkan bahwa image adalah gambar dengan ukuran 500x375 piksel, image_id adalah 1, dan objects berisi informasi tentang dua objek dengan kategori "helmet",

area, dan bbox (bounding box) yang menunjukkan posisi objek dalam gambar. Anotasi ini digunakan untuk mendeteksi objek pada gambar tersebut.

```
train_dataset
Dataset({
  features: ['image', 'image_id', 'width', 'height', 'objects'],
  num_rows: 5297
})
```

Kodingan ini menunjukkan **train_dataset**, yang merupakan dataset dengan 5.297 baris data. Outputnya menggambarkan struktur dataset yang memiliki lima fitur utama: **image** (gambar), **image_id** (ID gambar), **width** dan **height** (dimensi gambar), serta **objects** (informasi objek dalam gambar, seperti kategori dan posisi). Dataset ini digunakan untuk pelatihan model deteksi objek.

```
# transforming a batch

def transform_aug_ann(examples):
    image_ids = examples["image_id"]
    images, bboxes, area, categories = [], [], [], []
    for image, objects in zip(examples["image"], examples["objects"]):
        image = np.array(image.convert("RGB"))[:, :, ::-1]
        out = transform(image=image, bboxes=objects["bbox"],
category=objects["id"])

        area.append(objects["area"])
        images.append(out["image"])
        bboxes.append(out["bboxes"])
        categories.append(out["category"])

    targets = [
        {"image_id": id_, "annotations": formatted_anns(id_, cat_,
ar_, box_)}
        for id_, cat_, ar_, box_ in zip(image_ids, categories, area,
bboxes)
    ]

    return image_processor(images=images, annotations=targets,
return_tensors="pt")
```

Fungsi **transform_aug_ann** mengubah gambar dan anotasi dalam batch dengan transformasi seperti resize, flip, dan perubahan kontras. Gambar yang telah dimodifikasi dan anotasi terkait (seperti bbox, kategori, dan area) kemudian diproses menjadi tensor menggunakan **image_processor**, siap digunakan untuk pelatihan model.

```
# Apply transformations for both train and test dataset
```

```
train_dataset_transformed =
train_dataset.with_transform(transform_aug_ann)
test_dataset_transformed =
test_dataset.with_transform(transform_aug_ann)
```

Kode ini menggunakan **with_transform** untuk menerapkan transformasi **transform_aug_ann** pada dataset pelatihan dan pengujian, mempersiapkan data untuk proses pelatihan dan evaluasi model.

```
train_dataset_transformed[0]
{
  'pixel_values': tensor([[[[ 0.1083, 0.1083, 0.1083, ..., -0.0801, -0.2513, -0.3712],
    [ 0.1083, 0.1083, 0.1083, ..., -0.0801, -0.2513, -0.3712],
    [ 0.1083, 0.1083, 0.0912, ..., -0.0972, -0.2684, -0.3883],
    ...,
    [-0.2684, -0.3541, -0.4739, ..., -0.5424, -0.6109, -0.6452],
    [-0.2684, -0.3541, -0.4911, ..., -0.5596, -0.6452, -0.6794],
    [-0.2684, -0.3541, -0.4911, ..., -0.5767, -0.6623, -0.7137]]],
    [[ 0.1702, 0.1702, 0.1702, ..., 0.3803, 0.2052, 0.0826],
    [ 0.1702, 0.1702, 0.1702, ..., 0.3627, 0.1877, 0.0476],
    [ 0.1702, 0.1702, 0.1527, ..., 0.3452, 0.1527, 0.0126],
    ...,
    [ 0.1877, 0.0826, -0.0749, ..., -0.2325, -0.2850, -0.3200],
    [ 0.1877, 0.0826, -0.0924, ..., -0.2500, -0.3200, -0.3550],
    [ 0.1877, 0.0826, -0.0924, ..., -0.2500, -0.3375, -0.3901]]],
    [[ 0.4788, 0.4788, 0.4788, ..., 1.0365, 0.8622, 0.7402],
    [ 0.4788, 0.4788, 0.4788, ..., 1.0191, 0.8448, 0.7054],
    [ 0.4788, 0.4788, 0.4614, ..., 1.0017, 0.8099, 0.6705],
    ...,
    [ 0.8274, 0.7228, 0.5834, ..., 0.4614, 0.3916, 0.3568],
    [ 0.8274, 0.7228, 0.5659, ..., 0.4439, 0.3742, 0.3219],
    [ 0.8274, 0.7228, 0.5659, ..., 0.4265, 0.3568, 0.3045]]]),
  'pixel_mask': tensor([[[1, 1, 1, ..., 1, 1, 1],
    [1, 1, 1, ..., 1, 1, 1],
    [1, 1, 1, ..., 1, 1, 1],
    ...,
    [1, 1, 1, ..., 1, 1, 1],
    [1, 1, 1, ..., 1, 1, 1],
    [1, 1, 1, ..., 1, 1, 1]]]),
  'labels': {'size': tensor([800, 800]), 'image_id': tensor([1]), 'class_labels': tensor([1, 1]), 'boxes': tensor([[[0.5920, 0.3027, 0.1040, 0.1573],
    [0.7550, 0.4240, 0.0460, 0.0800]]]), 'area': tensor([8522.2217, 1916.6666]), 'iscrowd': tensor([0, 0]), 'orig_size': tensor([480, 480])}}
```

Pada kode ini, **train_dataset_transformed[0]** mengembalikan contoh pertama dari dataset yang sudah diproses. Outputnya berisi dua bagian utama: **pixel_values** (tensor berisi nilai piksel gambar yang telah dinormalisasi) dan **labels** (informasi terkait objek yang ada dalam gambar). **labels** menyimpan informasi seperti ukuran gambar, ID gambar, label kelas, kotak pembatas (bounding boxes), luas objek, serta apakah objek tersebut merupakan objek yang terhitung sebagai "crowd" atau bukan. **pixel_mask** menunjukkan area gambar yang relevan, yaitu 1 untuk area yang terdeteksi dan 0 untuk area yang tidak terdeteksi. Data ini siap digunakan untuk pelatihan model deteksi objek.

```
def collate_fn(batch):
    pixel_values = [item["pixel_values"] for item in batch]
    encoding = image_processor.pad(pixel_values, return_tensors="pt")
    labels = [item["labels"] for item in batch]
    batch = {}
    batch["pixel_values"] = encoding["pixel_values"]
    batch["pixel_mask"] = encoding["pixel_mask"]
    batch["labels"] = labels
    return batch
```

Fungsi **collate_fn** menggabungkan data dalam batch, memproses **pixel_values** menggunakan **image_processor.pad** agar memiliki ukuran yang konsisten, dan mengumpulkan **labels**. Hasilnya adalah sebuah dictionary yang berisi **pixel_values**, **pixel_mask**, dan **labels** yang siap digunakan oleh model.

```
from transformers import AutoModelForObjectDetection

id2label = {0:'head', 1:'helmet', 2:'person'}
label2id = {v: k for k, v in id2label.items()}

model = AutoModelForObjectDetection.from_pretrained(
    checkpoint,
    id2label=id2label,
    label2id=label2id,
    ignore_mismatched_sizes=True,
)
```

Kode ini memuat model deteksi objek dari checkpoint yang ditentukan, dengan pengaturan **id2label** dan **label2id** untuk mengonversi ID ke label dan sebaliknya. Pengaturan **ignore_mismatched_sizes=True** memungkinkan pemuatan model meskipun ada perbedaan ukuran. Model siap digunakan untuk deteksi objek dengan label yang sesuai.

```
# Login to Hugging Face, to upload your model on the fly while training.

from huggingface_hub import notebook_login

notebook_login
```

Kode ini menggunakan **notebook_login** dari **huggingface_hub** untuk login ke akun Hugging Face, memungkinkan pengunggahan model selama pelatihan.

```
# Define the training Arguments

from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="detr-resnet-50-hardhat-finetuned",
    per_device_train_batch_size=8,
    num_train_epochs=3,
    max_steps=1000,
    fp16=True,
    save_steps=10,
    logging_steps=30,
    learning_rate=1e-5,
    weight_decay=1e-4,
    save_total_limit=2,
```

```

        remove_unused_columns=False,
        push_to_hub=True,
    )

```

Kode ini mendefinisikan argumen pelatihan untuk model, seperti ukuran batch, jumlah epoch, pengaturan FP16, langkah penyimpanan dan logging,

```

from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=collate_fn,
    train_dataset=train_dataset_transformed,
    eval_dataset=test_dataset_transformed,
    tokenizer=image_processor,
)

trainer.train()

```

Kode ini melatih model menggunakan Trainer dari transformers dengan mengonfigurasi argumen pelatihan, data, dan tokenizer. Namun, terjadi error 401 Unauthorized saat mencoba mengunggah model ke Hugging Face, yang menunjukkan bahwa pengguna tidak terautentikasi atau kredensial yang digunakan salah. Kemungkinan penyebabnya adalah login Hugging Face belum berhasil atau token akses tidak valid.

```

# delete this model, since it is already been uploaded to hub

del model
torch.cuda.synchronize()

```

Kode ini menghapus model dari memori dan memastikan sinkronisasi operasi GPU dengan torch.cuda.synchronize().

```

import requests

# download a sample image

url = "https://huggingface.co/datasets/hf-vision/course-
assets/blob/main/test-helmet-object-detection.jpg"
image = Image.open(requests.get(url, stream=True).raw)

image

```

Kode ini berusaha untuk mengunduh dan membuka gambar dari URL menggunakan requests dan PIL.Image.open(). Namun, terjadi error UnidentifiedImageError yang disebabkan oleh ketidakmampuan PIL untuk mengenali atau memproses file gambar dari URL tersebut. Hal

ini kemungkinan disebabkan oleh URL yang tidak langsung mengarah pada file gambar atau format gambar yang tidak didukung.

```
from transformers import pipeline

# make the object detection pipeline

obj_detector = pipeline("object-detection", model="anindya64/detr-resnet-50-dc5-hardhat-finetuned")
results = obj_detector(train_dataset[0]['image'])

results
```

Kode ini membuat pipeline deteksi objek menggunakan model "anindya64/detr-resnet-50-dc5-hardhat-finetuned". Pipeline ini digunakan untuk mendeteksi objek dalam gambar pertama dari train_dataset dan menyimpan hasil deteksinya dalam variabel results, yang berisi label dan posisi bounding box objek yang terdeteksi.

```
def plot_results(image, results, threshold = 0.7):
    image = Image.fromarray(np.uint8(image))
    draw = ImageDraw.Draw(image)
    for result in results:
        score = result['score']
        label = result['label']
        box = list(result['box'].values())
        if score > threshold:
            x, y, x2, y2 = tuple(box)
            draw.rectangle((x, y, x2, y2), outline="red", width=1)
            draw.text((x, y), label, fill="white")
            draw.text((x+0.5, y-0.5), text=str(score), fill='green' if
score > 0.7 else 'red')
    return image
```

Fungsi plot_results menggambar bounding box dan label objek pada gambar, berdasarkan hasil deteksi dengan skor lebih dari 0.7. Skor ditampilkan dengan warna hijau jika tinggi, dan merah jika rendah. Fungsi ini mengembalikan gambar yang sudah dianotasi.

```
plot_results(image, results)
```

Code ini mencoba mengakses variabel image, namun terjadi error, Error NameError: name 'image' is not defined terjadi karena variabel image belum didefinisikan sebelum dipanggil dalam fungsi plot_results.

Clubbing it altogether into a function

```
def predict(image, pipeline, threshold=0.7):
    results = pipeline(image)
    return plot_results(image, results, threshold)
```

Fungsi predict menjalankan deteksi objek pada gambar menggunakan pipeline yang diberikan, kemudian memproses dan menggambar hasil deteksi (kotak dan label) pada gambar sesuai dengan skor yang lebih tinggi dari threshold.

```
img = test_dataset[0]['image']  
predict(img, obj_detector)
```



Kode tersebut mengambil gambar pertama dari dataset pengujian (test_dataset[0]['image']), lalu menjalankan fungsi predict untuk mendeteksi objek pada gambar tersebut menggunakan model deteksi objek (obj_detector). Hasil deteksi objek digambarkan pada gambar, menampilkan kotak dan label sesuai dengan skor yang melewati ambang batas (threshold).

```
from tqdm.auto import tqdm  
  
def plot_images(dataset, indices):  
    """  
    Plot images and their annotations.  
    """  
    num_rows = len(indices) // 3  
    num_cols = 3  
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))  
  
    for i, idx in tqdm(enumerate(indices), total=len(indices)):  
        row = i // num_cols  
        col = i % num_cols  
  
        # Draw image  
        image = predict(dataset[idx]['image'], obj_detector)  
  
        # Display image on the corresponding subplot  
        axes[row, col].imshow(image)  
        axes[row, col].axis('off')  
  
    plt.tight_layout()  
    plt.show()
```

```
plot_images(test_dataset, range(6))
```



Fungsi `plot_images` menampilkan beberapa gambar dari dataset dengan hasil deteksi objek. Gambar dipilih berdasarkan indeks, kemudian objek pada gambar dideteksi menggunakan fungsi `predict`. Hasilnya ditampilkan dalam grid subplot menggunakan `tqdm` untuk menunjukkan progress.

```
from transformers import AutoModelForObjectDetection
```

```
id2label = {0:'head', 1:'helmet', 2:'person'}
label2id = {v: k for k, v in id2label.items()}
```

```

Some weights of the model checkpoint at facebook/detr-resnet-50-dc were not used when initializing DetrForObjectDetection: ['model.backbone.conv_encoder.model.layer1.0.downsample.1.num_batches_tracked', 'model.backbone.conv_encoder.
- This is expected if you are initializing DetrForObjectDetection from the checkpoint of a model trained on another task or with another architecture (e.g. Initializing a BertForSequenceClassification model from a BertForPreTraining
is not expected if you are initializing DetrForObjectDetection from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification mode
Some weights of DetrForObjectDetection were not initialized from the model checkpoint at facebook/detr-resnet-50-dc and are newly initialized because the shapes did not match:
- class_labels.classifier.bias: found shape torch.Size([92]) in the checkpoint and torch.Size([4]) in the model instantiated
- class_labels.classifier.weight: found shape torch.Size([92, 256]) in the checkpoint and torch.Size([4, 256]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

```

Pada kode di atas, model deteksi objek `AutoModelForObjectDetection` diinisialisasi dengan checkpoint dari `facebook/detr-resnet-50-dc5`. Id untuk label (seperti "head", "helmet", "person") didefinisikan menggunakan `id2label` dan `label2id`. Kemudian, model diatur agar hanya parameter terkait prediksi bounding box dan klasifikasi kelas yang dapat diperbarui (parameter lainnya dibekukan). Namun, output menunjukkan bahwa beberapa bobot dari checkpoint yang digunakan tidak sesuai dengan model yang sedang diinisialisasi, misalnya

terkait dengan layer `num_batches_tracked`, yang tidak digunakan pada model deteksi objek ini. Selain itu, beberapa bobot pada klasifikasi label tidak sesuai dengan ukuran yang diharapkan, sehingga model memerlukan pelatihan lebih lanjut untuk menyesuaikan bobot tersebut sebelum bisa digunakan untuk prediksi.

3. Knowledge Distillation with Vision Transformers – Example

```
!pip install datasets
!pip install evaluate
```

- `!pip install datasets`: Menginstal library datasets untuk memuat dataset dari Hugging Face Hub.
- `!pip install evaluate`: Menginstal library evaluate untuk mengukur kinerja model.

```
from datasets import load_dataset, DatasetDict, Dataset
from transformers import AutoImageProcessor,
AutoModelForImageClassification, ViTConfig, ViTForImageClassification
from transformers import ViTImageProcessor, ViTConfig,
ViTForImageClassification
from transformers import TrainingArguments, Trainer
import evaluate

import torch
import torch.nn as nn
import torch.nn.functional as F

import numpy as np

import matplotlib.pyplot as plt
```

- `datasets`: Untuk memuat dataset yang digunakan dalam pelatihan dan evaluasi model.
- `transformers`: Berbagai kelas untuk memproses gambar dan memuat model klasifikasi gambar seperti `AutoImageProcessor`, `AutoModelForImageClassification`, dan model berbasis Vision Transformer (ViT) seperti `ViTConfig`, `ViTForImageClassification`.
- `TrainingArguments`, `Trainer`: Untuk menyusun argumen pelatihan dan melatih model.
- `evaluate`: Untuk evaluasi model menggunakan metrik yang sesuai.
- `torch`: Untuk mendefinisikan dan melatih model berbasis PyTorch, dengan pengimporan modul `nn` dan `F` untuk lapisan neural network dan fungsi aktivasi.
- `numpy`: Untuk perhitungan numerik.
- `matplotlib.pyplot`: Untuk visualisasi data, khususnya dalam menampilkan gambar.

```
dataset = load_dataset("pcuenq/oxford-pets")
```

Kode ini memuat dataset **Oxford Pets** menggunakan fungsi `load_dataset` dari pustaka `datasets`, yang berisi gambar hewan peliharaan dengan label kategori, siap digunakan untuk pelatihan model klasifikasi gambar.

```
id2label = {int_label: str_label for int_label, str_label in
enumerate(sorted(list(set(dataset['train']['label']))))}
label2id = {str_label: int_label for int_label, str_label in
enumerate(sorted(list(set(dataset['train']['label']))))}
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
teacher_model_name = "asusevski/vit-base-patch16-224-oxford-pets"
teacher_model =
AutoModelForImageClassification.from_pretrained(teacher_model_name).to
(device)
teacher_model.eval()
processor = ViTImageProcessor.from_pretrained("google/vit-base-
patch16-224")
accuracy = evaluate.load("accuracy")
```

Kode ini memetakan label numerik ke string, memuat model **ViT** yang sudah dilatih untuk dataset Oxford Pets, dan mengonfigurasi perangkat (GPU/CPU). Model diatur dalam mode evaluasi dan akurasi dipersiapkan untuk evaluasi.

```
dataset["train"][0]['image']
```

Kode ini mengambil gambar pertama dari dataset pelatihan (train) pada indeks 0. Gambar ini disimpan dalam field 'image' yang bisa digunakan untuk analisis lebih lanjut, seperti pemrosesan citra atau prediksi model.

```
inputs = processor(dataset["train"][0]["image"], return_tensors="pt")
# Here, using default processor on the image
inputs = {key: value.to(device) for key, value in inputs.items()} #
Converting to a dictionary
model_logits = teacher_model(**inputs).logits
model_prediction = torch.argmax(model_logits, dim=1).item()
print(id2label[model_prediction])
```

Pada kode tersebut, gambar pertama dari dataset diambil dan diproses menggunakan processor untuk menyiapkan input bagi model. Namun, terjadi error karena objek yang diberikan ke processor bukan dalam format yang diterima, yaitu berupa dict. Sebagai hasilnya, fungsi `make_list_of_images` mengeluarkan error yang mengatakan bahwa input gambar seharusnya bertipe `PIL.Image.Image`, `numpy.ndarray`, atau tipe lain yang valid, bukan dict.

```
temperature_no_change = 1
low_temperature = 2
high_temperature = 8
```

```

teacher_distribution = F.softmax(model_logits / temperature_no_change,
dim=-1).cpu().detach().numpy().reshape(-1)
teacher_distribution_low_temp = F.softmax(model_logits /
low_temperature, dim=-1).cpu().detach().numpy().reshape(-1)
teacher_distribution_high_temp = F.softmax(model_logits /
high_temperature, dim=-1).cpu().detach().numpy().reshape(-1)

fig, axs = plt.subplots(1, 2, figsize=(10, 4))
axs[0].bar(id2label.keys(), teacher_distribution, width = 0.4)
axs[0].set_title("Teacher probability over all classes")
axs[0].set_ylabel('Teacher Probability')
axs[0].set_xlabel("Class ID")

axs[1].bar(np.fromiter(id2label.keys(), dtype=float) - 0.4,
teacher_distribution_low_temp, width = 0.4, label='Temperature=2')
axs[1].bar(np.fromiter(id2label.keys(), dtype=float) + 0.4,
teacher_distribution_high_temp, width = 0.4, label='Temperature=8')
axs[1].set_title("Teacher probability over all classes, Temperature =
2 and 8")
axs[1].set_ylabel('Teacher Probability')
axs[1].set_xlabel("Class ID")
axs[1].legend()
plt.tight_layout()
plt.show()

```

Kode ini menghitung distribusi probabilitas kelas menggunakan softmax pada logit model untuk tiga suhu berbeda (1, 2, dan 8), lalu menampilkan grafik distribusinya. Error `NameError` terjadi karena variabel `model_logits` di codingan sebelumnya juga terjadi error.

```

def transforms(example):
    # Convert to RGB - there are some example in the Oxford Pets
    dataset that are RGBA, and even at least one gif
    example['image'] = example['image'].convert('RGB')

    # Feed image into ViT image processor
    inputs = processor(example['image'], return_tensors='pt')

    # Add to example
    example['pixel_values'] = inputs['pixel_values'].squeeze()
    example['label'] = label2id[example['label']]
    return example

```

Fungsi transforms mengubah gambar ke format RGB, memprosesnya dengan ViTImageProcessor, dan mengonversi label ke ID numerik. Hasilnya adalah gambar yang siap diproses bersama label yang telah dipetakan.

```

transformed_dataset = dataset.map(transforms)

```

```
transformed_dataset.set_format("pt", columns=["pixel_values"],
output_all_columns=True)
```

Pada kode tersebut, fungsi transforms mencoba mengonversi gambar yang ada dalam dataset ke format RGB dengan menggunakan metode `.convert('RGB')`, namun terjadi error `AttributeError: 'dict' object has no attribute 'convert'`. Hal ini terjadi karena dataset yang diterima oleh fungsi transforms berbentuk dictionary, dan objek gambar seharusnya berupa `PIL.Image.Image`.

```
def collate_fn(batch):
    return {
        'pixel_values': torch.cat([x['pixel_values'].unsqueeze(dim=0)
for x in batch], dim=0),
        'labels': torch.tensor([x['label'] for x in batch])
    }
```

Fungsi `collate_fn` menggabungkan gambar dan label dalam batch menjadi tensor besar. Gambar diproses dengan menambahkan dimensi baru dan digabungkan menggunakan `torch.cat`, sedangkan label dikumpulkan menjadi tensor. Fungsi ini mengembalikan dictionary dengan kunci `'pixel_values'` dan `'labels'`.

```
train_test_dataset =
transformed_dataset['train'].train_test_split(test_size=0.2)
train_val_dataset =
train_test_dataset['train'].train_test_split(test_size=(0.1/0.8))
train_test_valid_dataset = DatasetDict({
    'train': train_val_dataset['train'],
    'valid': train_val_dataset['test'],
    'test': train_test_dataset['test']
}))
```

Pada kode tersebut, tujuan utamanya adalah membagi dataset menjadi tiga subset: train, valid, dan test. Pertama, dataset pelatihan diambil dan dibagi menjadi data pelatihan dan pengujian dengan proporsi 80:20. Selanjutnya, dataset pelatihan dibagi lagi menjadi data pelatihan dan validasi dengan proporsi 90:10. Namun, error `NameError: name 'transformed_dataset' is not defined` muncul karena variabel `transformed_dataset` belum didefinisikan atau tidak ada di dalam konteks kode tersebut, sehingga menyebabkan kegagalan dalam menjalankan operasi pembagian dataset.

```
config = ViTForImageClassification.from_pretrained("WinKawaks/vit-
tiny-patch16-224").config
config.id2label = id2label
config.label2id = label2id
config.num_labels=len(id2label)
base_model = ViTForImageClassification(config).to(device)
```

Kode ini memuat konfigurasi model ViT untuk klasifikasi gambar, memodifikasinya dengan id2label, label2id, dan jumlah label, lalu menginisialisasi model ViT dengan konfigurasi tersebut dan memindahkannya ke perangkat (GPU atau CPU).

```
from huggingface_hub import notebook_login
```

```
notebook_login()
```

notebook_login: Untuk autentikasi dan mengakses dataset atau model yang memerlukan izin.

```
training_args = TrainingArguments(  
    output_dir="oxford-pets-vit-from-scratch",  
    per_device_train_batch_size=48,  
    per_device_eval_batch_size=48,  
    evaluation_strategy="epoch",  
    save_strategy="epoch",  
    logging_steps=100,  
    num_train_epochs=10,  
    learning_rate=3e-4,  
    push_to_hub=True,  
    load_best_model_at_end=True,  
)
```

Kode ini mengonfigurasi argumen pelatihan untuk model, seperti ukuran batch, jumlah epoch, strategi evaluasi dan penyimpanan, laju pembelajaran, serta pengaturan untuk memuat model terbaik dan mengunggah ke Hugging Face Hub.

```
accuracy = evaluate.load("accuracy")  
  
def compute_metrics(eval_pred):  
    predictions, labels = eval_pred  
    acc = accuracy.compute(references=labels,  
        predictions=np.argmax(predictions, axis=1))  
    return {"accuracy": acc["accuracy"]}
```

Kode ini mendefinisikan fungsi compute_metrics untuk menghitung akurasi dengan membandingkan prediksi model dan label asli, lalu mengembalikan nilai akurasi.

```
trainer = Trainer(  
    model=base_model,  
    args=training_args,  
    data_collator=collate_fn,  
    compute_metrics=compute_metrics,  
    train_dataset=train_test_valid_dataset["train"],  
    eval_dataset=train_test_valid_dataset["test"],  
    tokenizer=processor  
)
```

Kode ini mendefinisikan sebuah objek Trainer untuk melatih model menggunakan dataset pelatihan dan evaluasi. Namun, error NameError terjadi karena objek train_test_valid_dataset

belum didefinisikan sebelumnya, atau terjadi kesalahan penamaan. Variabel `train_test_valid_dataset` yang seharusnya berisi pembagian dataset pelatihan, validasi, dan pengujian error di codingan sebelumnya, sehingga menyebabkan error saat mencoba mengaksesnya.

```
trainer.train()
trainer.evaluate(train_test_valid_dataset["test"])
```

Kode ini berusaha menjalankan metode `train()` dan `evaluate()` pada objek `trainer`. Namun, error `NameError` terjadi karena objek `trainer` tidak terdefinisi sebelumnya dan juga error di codingan sebelumnya.

```
student_model = ViTForImageClassification(config).to(device)
```

Kode ini membuat model `student_model` dengan arsitektur Vision Transformer (ViT) untuk klasifikasi gambar, menggunakan konfigurasi yang telah ditentukan sebelumnya dan memindahkannya ke perangkat yang tersedia (CPU/GPU).

```
class ImageDistilTrainer(Trainer):
    def __init__(self, teacher_model=None, student_model=None,
temperature=None, lambda_param=None, *args, **kwargs):
        super().__init__(model=student_model, *args, **kwargs)
        self.teacher = teacher_model
        self.student = student_model
        self.loss_function = nn.KLDivLoss(reduction="batchmean")
        device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
        self.teacher.to(device)
        self.teacher.eval()
        self.temperature = temperature
        self.lambda_param = lambda_param

    def compute_loss(self, inputs, return_outputs=False):
        student_output = self.student(**inputs)

        with torch.no_grad():
            teacher_output = self.teacher(**inputs)

        # Compute soft targets for teacher and student
        soft_teacher = F.softmax(teacher_output.logits /
self.temperature, dim=-1)
        soft_student = F.log_softmax(student_output.logits /
self.temperature, dim=-1)

        # Compute the loss
        distillation_loss = self.loss_function(soft_student,
soft_teacher) * (self.temperature ** 2)
```

```

        # Compute the true label loss
        student_target_loss = student_output.loss

        # Calculate final loss
        loss = (1. - self.lambda_param) * student_target_loss +
self.lambda_param * distillation_loss
        return (loss, student_output) if return_outputs else loss

```

Kelas ImageDistilTrainer adalah turunan dari Trainer yang digunakan untuk distilasi model, memindahkan pengetahuan dari model guru ke model siswa. Kelas ini menghitung kehilangan distilasi menggunakan KL Divergence antara output softmax model guru dan siswa, serta kehilangan label target untuk model siswa. Gabungan keduanya menghasilkan kehilangan akhir yang dikontrol oleh parameter lambda. Model guru dipindahkan ke perangkat yang tersedia (CPU/GPU) dan diatur ke mode evaluasi.

```

training_args = TrainingArguments(
    output_dir="oxford-pets-vit-with-kd",
    per_device_train_batch_size=48,
    per_device_eval_batch_size=48,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_steps=100,
    num_train_epochs=10,
    learning_rate=3e-4,
    push_to_hub=True,
    load_best_model_at_end=True,
)

```

training_args mengatur konfigurasi pelatihan model, seperti ukuran batch, strategi evaluasi, jumlah epoch, laju pembelajaran, dan pengaturan untuk menyimpan dan mengunggah model.

```

teacher_model.eval()
trainer = ImageDistilTrainer(
    student_model=student_model,
    teacher_model=teacher_model,
    args=training_args,
    data_collator=collate_fn,
    compute_metrics=compute_metrics,
    train_dataset=train_test_valid_dataset["train"],
    eval_dataset=train_test_valid_dataset["test"],
    tokenizer=processor,
    temperature=5,
    lambda_param=0.9
)

```

Pada kode tersebut, objek `train_test_valid_dataset` yang digunakan untuk mengakses dataset pelatihan dan pengujian error di codingan sebelumnya, sehingga menghasilkan error `NameError`.

```
trainer.train()
trainer.evaluate(train_test_valid_dataset["test"])
```

Kode ini berusaha menjalankan metode `train()` dan `evaluate()` pada objek `trainer`. Namun, error `NameError` terjadi karena objek `trainer` tidak terdefinisi sebelumnya dan juga error di codingan sebelumnya.

4. Image classification using LoRA with Vision Transformers

Import

```
%%capture
!pip install datasets
!pip install evaluate
!pip install accelerate -U
!pip install peft
```

- `!pip install datasets`: Menginstal library `datasets` untuk memuat dataset dari Hugging Face Hub.
- `!pip install evaluate`: Menginstal library `evaluate` untuk mengukur kinerja model.
- `!pip install accelerate -U`: Menginstal atau memperbarui library `accelerate` untuk mempercepat pelatihan model.
- `!pip install peft`: Menginstal `peft` untuk pembelajaran parameter efisien (parameter-efficient fine-tuning) pada model besar.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torchvision.transforms as T
from datasets import load_dataset
from transformers import AutoImageProcessor, ViTForImageClassification
from transformers import Trainer, TrainingArguments
import evaluate
```

- `numpy`, `pandas`: Untuk manipulasi data.
- `matplotlib.pyplot`: Untuk visualisasi data.
- `torch`, `torch.nn`: Untuk membangun dan melatih model deep learning.
- `torchvision.transforms as T`: Untuk transformasi dan augmentasi gambar.

- datasets: Untuk memuat dataset dari Hugging Face Hub.
- transformers: Untuk menggunakan model berbasis transformer seperti Vision Transformer (ViT).
- evaluate: Untuk mengevaluasi performa model.

Authentication

```
from huggingface_hub import notebook_login

notebook_login()
```

notebook_login: Untuk autentikasi dan mengakses dataset atau model yang memerlukan izin.

Loading the datasets

```
dataset = load_dataset('pcuenq/oxford-pets')
```

load_dataset('pcuenq/oxford-pets'): Memuat dataset **Oxford Pets** dari Hugging Face Hub, yang berisi gambar hewan peliharaan untuk tugas klasifikasi gambar.

```
dataset['train'][0]
```

`dataset['train'][0]` mengakses sampel pertama dari subset **train** dalam dataset Oxford Pets. Outputnya adalah dictionary berisi 'image' (gambar hewan peliharaan) dan 'label' (kelas hewan). Ini digunakan untuk memeriksa data atau memvisualisasikan gambar sebelum preprocessing dan pelatihan model.

```
classes = dataset['train'].unique('label')
print(len(classes), classes)
```

Kode ini digunakan untuk mendapatkan daftar unik dari semua label yang ada di subset **train** dari dataset. Fungsi `unique('label')` mengembalikan nilai-nilai unik dari kolom 'label', yang mewakili kelas-kelas dalam dataset. Dengan `len(classes)`, Anda dapat mengetahui jumlah total kelas, dan `classes` akan mencetak daftar label tersebut. Hal ini berguna untuk memahami struktur data, seperti jumlah dan jenis kategori yang akan diklasifikasikan oleh model.

```
def show_samples(ds, rows=2, cols=4):
    samples = ds.shuffle().select(np.arange(rows*cols))
    fig = plt.figure(figsize=(cols*4, rows*4))
    for i in range(rows*cols):
        img = samples[i]['image']
        label = samples[i]['label']
        fig.add_subplot(rows, cols, i+1)
        plt.imshow(img)
        plt.title(label)
```

```
plt.axis('off')
```

```
show_samples(dataset['train'], rows=3, cols=5)
```

Fungsi `show_samples` digunakan untuk menampilkan sampel gambar dari dataset dalam bentuk grid. Namun code tersebut error, error terjadi karena gambar (img) yang diambil dari dataset berformat **PIL.Image** atau tipe objek lain yang tidak kompatibel langsung dengan **Matplotlib**, yang membutuhkan data dalam format array numpy dengan tipe data seperti float32 atau uint8.

Preprocessing

```
dataset = dataset['train'].train_test_split(train_size=0.8)
dataset
```

```
DatasetDict({
  train: Dataset({
    features: ['path', 'label', 'dog', 'image'],
    num_rows: 5912
  })
  test: Dataset({
    features: ['path', 'label', 'dog', 'image'],
    num_rows: 1478
  })
})
```

Kode ini membagi dataset **train** menjadi 80% data pelatihan dan 20% data pengujian. Hal ini dilakukan untuk melatih model pada satu bagian dan menguji kinerjanya pada bagian lain.

```
model_name = "vit-base-patch16-224"
model_checkpoint = f"google/{model_name}"

processor = AutoImageProcessor.from_pretrained(model_checkpoint)
processor
```

Kode ini memuat preprocessor untuk model Vision Transformer (ViT) **vit-base-patch16-224**, yang akan mempersiapkan gambar dengan preprocessing yang diperlukan sebelum diberikan ke model.

```
label2id = {c:idx for idx,c in enumerate(classes)}
id2label = {idx:c for idx,c in enumerate(classes)}
```

Kode ini membuat dua dictionary: **label2id** untuk mengonversi label ke ID numerik, dan **id2label** untuk mengonversi ID kembali ke label.

```
def transforms(batch):
    batch['image'] = [x.convert('RGB') for x in batch['image']]
    inputs = processor([x for x in
batch['image']], return_tensors='pt')
    inputs['labels']=[label2id[y] for y in batch['label']]
    return inputs
```

Fungsi **transforms** mengonversi gambar ke format **RGB**, memprosesnya menjadi tensor, dan mengubah label menjadi ID numerik, agar siap digunakan oleh model.

```
dataset = dataset.with_transform(transforms)
```

Kode ini menerapkan fungsi **transforms** ke dataset, memproses gambar dan label agar siap digunakan oleh model.

```
def collate_fn(batch):
    return {
        'pixel_values': torch.stack([x['pixel_values'] for x in
batch]),
        'labels': torch.tensor([x['labels'] for x in batch])
    }
```

Fungsi **collate_fn** menggabungkan **pixel_values** dan **labels** dalam batch menjadi tensor agar siap digunakan oleh model.

Metric

```
accuracy = evaluate.load('accuracy')

def compute_metrics(eval_preds):
    logits, labels = eval_preds
    predictions = np.argmax(logits,axis=1)
    acc = accuracy.compute(predictions=predictions, references=labels)
    return acc
```

Kode ini memuat metrik **accuracy** menggunakan **evaluate.load('accuracy')** dan mendefinisikan fungsi **compute_metrics** untuk menghitung akurasi model. Fungsi ini menerima prediksi dan label aktual sebagai input, kemudian menghitung prediksi dengan mengambil argmax dari nilai logits. Setelah itu, fungsi ini menghitung akurasi dengan membandingkan prediksi dan label menggunakan metode **accuracy.compute** dan mengembalikan nilai akurasi.

```
def print_trainable_parameters(model):
    trainable_params = 0
    all_param = 0
    for _, param in model.named_parameters():
        all_param += param.numel()
        if param.requires_grad:
            trainable_params += param.numel()
    print(
        f"trainable params: {trainable_params} || all params: {all_param} || trainable%: {100 * trainable_params / all_param:.2f}"
    )
```

Fungsi **print_trainable_parameters** digunakan untuk menghitung dan menampilkan jumlah parameter yang dapat dilatih (trainable parameters) dan total parameter dalam model. Fungsi

ini menghitung total parameter dengan `param.numel()` dan memisahkan mana yang memerlukan gradien (parameter yang dapat dilatih). Hasilnya menunjukkan jumlah parameter yang dapat dilatih, total parameter, serta persentase parameter yang dapat dilatih dibandingkan dengan total parameter model.

```
from transformers import AutoModelForImageClassification,
TrainingArguments, Trainer

model = AutoModelForImageClassification.from_pretrained(
    model_checkpoint,
    label2id=label2id,
    id2label=id2label,
    ignore_mismatched_sizes=True,
)
print_trainable_parameters(model)
```

Kode ini memuat model **Vision Transformer (ViT)** untuk tugas klasifikasi gambar menggunakan **`AutoModelForImageClassification.from_pretrained`** dengan checkpoint yang telah ditentukan. Model ini juga disesuaikan dengan parameter **`label2id`** dan **`id2label`** untuk konversi label dan ID. **`ignore_mismatched_sizes=True`** digunakan untuk mengabaikan ketidaksesuaian ukuran antara model dan data jika ada. Setelah memuat model, fungsi **`print_trainable_parameters`** dipanggil untuk menampilkan jumlah parameter yang dapat dilatih dan total parameter dalam model.

```
from peft import LoraConfig, get_peft_model

config = LoraConfig(
    r=16,
    lora_alpha=16,
    target_modules=["query", "value"],
    lora_dropout=0.1,
    bias="none",
    modules_to_save=["classifier"],
)

lora_model = get_peft_model(model, config)
print_trainable_parameters(lora_model)

trainable params: 618277 || all params: 86445386 || trainable%: 0.72
```

Kode ini menerapkan teknik **Low-Rank Adaptation (LoRA)** pada model dengan menggunakan **`LoraConfig`** untuk mengonfigurasi parameter LoRA, seperti rank ($r=16$), alpha ($\text{lora_alpha}=16$), target modul yang disesuaikan (misalnya `query` dan `value`), serta dropout dan pengaturan bias. Dengan **`get_peft_model`**, model yang telah dimodifikasi untuk LoRA

ini diterapkan pada model yang sudah dimuat sebelumnya. Setelah itu, fungsi **print_trainable_parameters** digunakan untuk menampilkan jumlah parameter yang dapat dilatih dan total parameter dalam model LoRA. Output menunjukkan bahwa dari total 86.4 juta parameter, hanya 618.277 parameter yang dapat dilatih, yang berarti 0,72% dari total parameter model. Ini mengindikasikan bahwa hanya sebagian kecil parameter yang disesuaikan menggunakan teknik LoRA, menghemat sumber daya komputasi.

model

model adalah objek model yang digunakan untuk klasifikasi gambar, dimuat dari checkpoint pre-trained atau disesuaikan dengan teknik seperti **LoRA** untuk tugas klasifikasi. Model ini digunakan untuk melakukan prediksi setelah dilatih.

Training

```
batch_size = 128

args = TrainingArguments(
    f"{model_checkpoint}-finetuned-lora-oxford-pets",
    per_device_train_batch_size=batch_size,
    learning_rate=5e-3,
    num_train_epochs=5,
    per_device_eval_batch_size=batch_size,
    gradient_accumulation_steps=4,
    logging_steps=10,
    save_total_limit=2,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    metric_for_best_model="accuracy",
    report_to='tensorboard',
    fp16=True,
    push_to_hub=True,
    remove_unused_columns=False,
    load_best_model_at_end=True,
)
```

Kode ini mengonfigurasi pengaturan pelatihan model, termasuk batch size, learning rate, jumlah epoch, dan strategi evaluasi dan penyimpanan berdasarkan epoch. Model akan dilatih dengan **mixed precision (fp16)**, log disimpan di **TensorBoard**, dan model terbaik disimpan serta didorong ke Hugging Face Hub.

```
trainer = Trainer(
    model=model,
    args=args,
    data_collator=collate_fn,
    compute_metrics=compute_metrics,
```

```

train_dataset=dataset["train"],
eval_dataset=dataset["test"],
tokenizer=processor
)

```

Kode ini mengonfigurasi **Trainer** untuk melatih model dengan dataset dan pengaturan yang telah ditentukan. Code ini mengalami error, Error **403 Forbidden** terjadi karena token yang digunakan tidak memiliki izin untuk membuat model di namespace "**anitaa27**" di Hugging Face.

```

trainer.train()

```

Kode **trainer.train()** memulai proses pelatihan model dengan pengaturan yang telah ditentukan, termasuk data, epoch, dan metrik evaluasi. Code ini mengalami error dikarenakan codingan sebelumnya terjadi error.

Sharing your model and inference

```

repo_name = f"alanahmet/{model_name}-finetuned-lora-oxfordPets"
lora_model.push_to_hub(repo_name)

```

Kode ini mengunggah model yang telah dilatih ke Hugging Face Hub. Error **403 Forbidden** terjadi karena token yang digunakan tidak memiliki izin yang cukup untuk mengakses atau mengunggah ke repositori yang dimaksud.

```

from peft import PeftConfig, PeftModel

config = PeftConfig.from_pretrained(repo_name)
model = AutoModelForImageClassification.from_pretrained(
    config.base_model_name_or_path,
    label2id=label2id,
    id2label=id2label,
    ignore_mismatched_sizes=True, # provide this in case you're
    planning to fine-tune an already fine-tuned checkpoint
)
# Load the Lora model
inference_model = PeftModel.from_pretrained(model, repo_name)

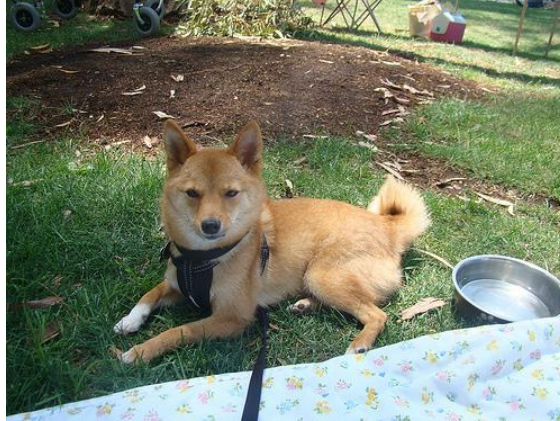
```

Kode ini mengonfigurasi dan memuat model yang telah dilatih menggunakan teknik **Low-Rank Adaptation (LoRA)**. Pertama, **PeftConfig.from_pretrained(repo_name)** memuat konfigurasi LoRA dari repositori yang sudah ada. Kemudian, model dasar (**base model**) dimuat menggunakan **AutoModelForImageClassification** dengan parameter **label2id** dan **id2label** untuk konversi label. **ignore_mismatched_sizes=True** memastikan ukuran model sesuai meskipun model sebelumnya sudah dilatih. Terakhir, **PeftModel.from_pretrained**

digunakan untuk memuat model LoRA yang telah disesuaikan dan siap digunakan untuk inferensi.

```
from PIL import Image
import requests

url = "https://huggingface.co/datasets/alanahmet/LoRA-pets-
dataset/resolve/main/shiba_inu_136.jpg"
image = Image.open(requests.get(url, stream=True).raw)
image
```



Kode ini digunakan untuk mengunduh gambar dari URL yang diberikan menggunakan **requests.get**, kemudian membuka gambar tersebut dengan **PIL.Image.open**. Gambar yang diunduh dan dibuka akan disimpan dalam variabel **image** dan dapat ditampilkan atau diproses lebih lanjut. URL yang digunakan mengarah pada gambar **shiba inu** yang ada di repositori Hugging Face.

```
image_processor = AutoImageProcessor.from_pretrained(model_checkpoint)
```

Kode ini memuat **AutoImageProcessor** dari Hugging Face untuk memproses gambar sesuai dengan model yang ditentukan melalui **model_checkpoint**. **AutoImageProcessor** secara otomatis memilih kelas pemrosesan gambar yang sesuai dengan model. Output menunjukkan bahwa kelas pemrosesan gambar cepat (**ViTImageProcessorFast**) tersedia untuk model ini, namun kelas yang lebih lambat digunakan secara default. Jika ingin menggunakan pemrosesan gambar yang lebih cepat, pengguna dapat menambahkan argumen **use_fast=True**.

```
encoding = image_processor(image.convert("RGB"), return_tensors="pt")
print(encoding.pixel_values.shape)
```


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

Kode ini memproses gambar menjadi tensor dengan ukuran yang sesuai untuk model **ViT**. Output **torch.Size([1, 3, 224, 224])** menunjukkan bahwa gambar diubah menjadi tensor dengan dimensi batch size 1, 3 saluran (RGB), dan ukuran 224x224 piksel.

```
import torch

# forward pass
with torch.no_grad():
    outputs = inference_model(**encoding)
    logits = outputs.logits

predicted_class_idx = logits.argmax(-1).item()
print("Predicted class:",
inference_model.config.id2label[predicted_class_idx])
```

 Predicted class: shiba inu

Kode ini melakukan prediksi kelas gambar dengan model LoRA, menggunakan **argmax** untuk menentukan kelas dengan nilai tertinggi. Output **"Predicted class: shiba inu"** menunjukkan bahwa model memprediksi gambar sebagai **shiba inu**.

5. Swin Transformer

```
%pip install datasets
```

!pip install datasets: Menginstal library datasets untuk memuat dataset dari Hugging Face Hub.

```
from datasets import load_dataset
from transformers import AutoImageProcessor,
SwinForImageClassification, SwinV2ForImageClassification
import torch
```

- `load_dataset` digunakan untuk memuat dataset dari berbagai sumber.
- `AutoImageProcessor` digunakan untuk memproses gambar, misalnya mengubah ukuran, normalisasi, atau konversi ke format yang sesuai untuk input model.
- `SwinForImageClassification` dan `SwinV2ForImageClassification` adalah model Swin Transformer yang digunakan untuk klasifikasi gambar.
- `torch` adalah pustaka PyTorch yang digunakan untuk manipulasi tensor dan komputasi model.

```
## Initialize a pre-trained Swin V1 model
model = SwinForImageClassification.from_pretrained("microsoft/swin-
tiny-patch4-window7-224")
image_processor = AutoImageProcessor.from_pretrained("microsoft/swin-
tiny-patch4-window7-224")
```



```
## You can uncomment and run this instead to use Swin V2
# model =
Swinv2ForImageClassification.from_pretrained("microsoft/swinv2-tiny-
patch4-window8-256")
# image_processor =
AutoImageProcessor.from_pretrained("microsoft/swinv2-tiny-patch4-
window8-256")
```

Kode ini memuat model Swin Transformer untuk klasifikasi gambar, dengan memanfaatkan **Swin V1** atau **Swin V2**. Model dan pemroses gambar dimuat dari repositori Hugging Face menggunakan SwinForImageClassification dan AutoImageProcessor. Jika diperlukan, model Swin V2 dapat digunakan dengan mengganti bagian yang dikomentari.

```
## Load images
dataset = load_dataset("huggingface/cats-image")

## Choose an image from the dataset you want to classify
image = dataset["test"]["image"][0]
image
```



Kode ini memuat dataset gambar dari Hugging Face dengan menggunakan load_dataset untuk dataset **cats-image**. Selanjutnya, kode memilih gambar pertama dari set data uji ("test") untuk digunakan dalam klasifikasi dan menyimpannya dalam variabel image. Gambar ini siap untuk diproses lebih lanjut menggunakan model klasifikasi.

```
## Process the selected image
inputs = image_processor(image, return_tensors="pt")

with torch.no_grad():
    logits = model(**inputs).logits

## Get the output and convert to a human-readable label
predicted_label_id = logits.argmax(-1).item()
predicted_label_text = model.config.id2label[predicted_label_id]
```

```
print(predicted_label_text)
```

```
tabby, tabby cat
```

Kode ini memproses gambar untuk mendapatkan input model, kemudian melakukan inferensi dengan model Swin tanpa menghitung gradien. Prediksi label diambil dari logits dan dikonversi ke teks yang terbaca. Output "tabby, tabby cat" menunjukkan gambar tersebut berhasil diklasifikasikan sebagai kucing tabby.

6. Fine-Tuning: Vision Transformers Multi-Label Image Classification

```
!pip install -Uq transformers datasets timm accelerate evaluate
```

Perintah tersebut menginstal pustaka penting untuk pembelajaran mesin. transformers digunakan untuk bekerja dengan model Transformer seperti BERT dan ViT, datasets membantu mengelola dan memproses dataset, timm menyediakan arsitektur model gambar pralatih, accelerate mengoptimalkan pelatihan pada perangkat keras seperti GPU/TPU, dan evaluate digunakan untuk menghitung metrik evaluasi seperti akurasi. Semua pustaka ini mendukung alur kerja yang efisien dalam pengembangan dan evaluasi model.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torchvision.transforms as T

from pathlib import Path
from PIL import Image

import datasets

from transformers.optimization import get_cosine_schedule_with_warmup

from timm import list_models, create_model

from accelerate import Accelerator, notebook_launcher

import evaluate
```

- **numpy**: Pustaka untuk komputasi numerik dan manipulasi array/matriks.
- **pandas**: Pustaka untuk analisis dan manipulasi data berbasis tabel (DataFrame).
- **matplotlib.pyplot**: Pustaka untuk visualisasi data seperti grafik atau plot.

- **torch**: Pustaka PyTorch untuk pengembangan model pembelajaran mesin dan deep learning.
- **torch.nn**: Modul PyTorch untuk membangun arsitektur model seperti lapisan neural network.
- **torchvision.transforms (disingkat T)**: Modul untuk melakukan transformasi data pada gambar.
- **Pathlib.Path**: Modul Python untuk menangani jalur file dan direktori.
- **PIL.Image**: Modul untuk memproses gambar seperti membuka, mengubah format, atau menyimpan gambar.
- **datasets**: Pustaka Hugging Face untuk memuat dan mengelola dataset.
- **transformers.optimization.get_cosine_schedule_with_warmup**: Fungsionalitas dari Hugging Face untuk membuat scheduler *learning rate* berbasis kurva kosinus dengan periode pemanasan.
- **timm.list_models** dan **timm.create_model**: Fungsionalitas dari timm untuk melihat daftar model prelatih yang tersedia dan membuat model tertentu.
- **accelerate.Accelerator** dan **notebook_launcher**: Modul untuk mempercepat pelatihan model menggunakan GPU/TPU dan mempermudah peluncuran proses pelatihan.
- **evaluate**: Pustaka Hugging Face untuk menghitung metrik evaluasi seperti akurasi, presisi, atau F1-score.

```
dataset =
datasets.load_dataset('fuliucansheng/pascal_voc', 'voc2007_main')
```

Kode tersebut memuat dataset **pascal_voc** dengan konfigurasi **voc2007_main** dari repositori Hugging Face pengguna **fuliucansheng**. Dataset ini sering digunakan untuk tugas klasifikasi dan deteksi objek.

```
dataset
```

```
DatasetDict({
  train: Dataset({
    features: ['id', 'image', 'height', 'width', 'classes', 'objects'],
    num_rows: 2501
  })
  validation: Dataset({
    features: ['id', 'image', 'height', 'width', 'classes', 'objects'],
    num_rows: 2510
  })
  test: Dataset({
    features: ['id', 'image', 'height', 'width', 'classes', 'objects'],
    num_rows: 4952
  })
})
```

Kode tersebut menampilkan dataset **DatasetDict** dengan tiga subset: **train** (2501 contoh), **validation** (2510 contoh), dan **test** (4952 contoh). Setiap subset memiliki fitur seperti **id**, **image**, dimensi gambar, label kelas, dan objek, yang siap digunakan untuk pelatihan, validasi, dan pengujian model.

```
class_names = [
    "Aeroplane", "Bicycle", "Bird", "Boat", "Bottle",
    "Bus", "Car", "Cat", "Chair", "Cow", "Diningtable",
    "Dog", "Horse", "Motorbike", "Person",
    "Potted plant", "Sheep", "Sofa", "Train", "Tv/monitor"
]
```

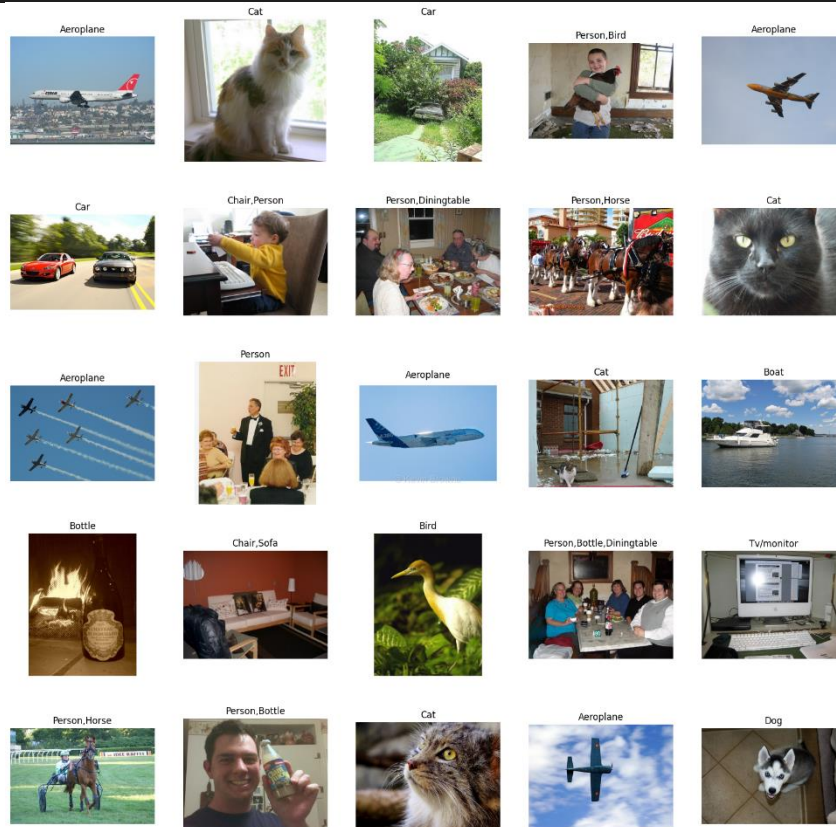
Kode tersebut mendefinisikan daftar **class_names** yang berisi 20 kategori objek dalam dataset PASCAL VOC, seperti "Aeroplane", "Bicycle", dan "Cat", untuk mengaitkan label numerik dengan nama kelas.

```
label2id = {c:idx for idx,c in enumerate(class_names)}
id2label = {idx:c for idx,c in enumerate(class_names)}
```

Kode tersebut membuat dua kamus: **label2id** yang memetakan nama kelas ke indeks, dan **id2label** yang memetakan indeks kembali ke nama kelas, untuk memudahkan konversi antara label string dan numerik.

```
def show_samples(ds, rows, cols):
    samples = ds.shuffle().select(np.arange(rows*cols)) # selecting
    random images
    fig = plt.figure(figsize=(cols*4, rows*4))
    # plotting
    for i in range(rows*cols):
        img = samples[i]['image']
        labels = samples[i]['classes']
        # getting string labels and combining them with a comma
        labels = ','.join([id2label[lb] for lb in labels])
        fig.add_subplot(rows, cols, i+1)
        plt.imshow(img)
        plt.title(labels)
        plt.axis('off')
```

```
show_samples(dataset['train'], rows=5, cols=5)
```



Fungsi **show_samples** menampilkan gambar acak dari dataset dalam grid yang ditentukan oleh **rows** dan **cols**. Gambar-gambar tersebut diberi judul label kelas yang dikonversi dari ID numerik, dan sumbu plot dimatikan. Fungsi ini memungkinkan visualisasi sampel gambar secara mudah.

Preprocessing our Dataset

```
img_size = (224,224)

train_tfms = T.Compose([
    T.Resize(img_size),
    T.RandomHorizontalFlip(),
    T.RandomRotation(30),
    T.CenterCrop(img_size),
    T.ToTensor(),
    T.Normalize(
        mean = (0.5,0.5,0.5),
        std = (0.5,0.5,0.5)
    )
])

valid_tfms = T.Compose([
```

```

        T.Resize(img_size),
        T.ToTensor(),
        T.Normalize(
            mean = (0.5,0.5,0.5),
            std = (0.5,0.5,0.5)
        )
    ])

```

Kode tersebut mendefinisikan transformasi gambar untuk pelatihan (**train_tfms**) dengan augmentasi acak (resize, flip, rotasi) dan normalisasi, serta untuk validasi (**valid_tfms**) dengan resize dan normalisasi tanpa augmentasi. Transformasi ini mempersiapkan gambar untuk pelatihan dan evaluasi model.

```

def train_transforms(batch):
    # Convert all images to RGB format
    if isinstance(batch['image'], list):
        # Batch processing
        batch['image'] = [x.convert('RGB') for x in batch['image']]
        inputs = [train_tfms(x) for x in batch['image']]
        batch['pixel_values'] = torch.stack(inputs) # Stack tensor
    else:
        # Single sample processing
        batch['image'] = batch['image'].convert('RGB')
        batch['pixel_values'] = train_tfms(batch['image'])

    # One-hot encode the multilabels
    all_labels = [torch.tensor(labels) for labels in batch['classes']]

    # Create one-hot encoding for each image's classes
    one_hot_labels = [nn.functional.one_hot(label,
num_classes=20).sum(dim=0) for label in all_labels]

    # Stack them into a batch
    batch['labels'] = torch.stack(one_hot_labels)

    return batch

def valid_transforms(batch):
    # Convert all images to RGB format
    if isinstance(batch['image'], list):
        # Batch processing
        batch['image'] = [x.convert('RGB') for x in batch['image']]
        inputs = [train_tfms(x) for x in batch['image']]
        batch['pixel_values'] = torch.stack(inputs) # Stack tensor
    else:
        # Single sample processing

```

```

        batch['image'] = batch['image'].convert('RGB')
        batch['pixel_values'] = train_tfms(batch['image'])

    # One-hot encode the multilabels
    all_labels = [torch.tensor(labels) for labels in batch['classes']]

    # Create one-hot encoding for each image's classes
    one_hot_labels = [nn.functional.one_hot(label,
num_classes=20).sum(dim=0) for label in all_labels]

    # Stack them into a batch
    batch['labels'] = torch.stack(one_hot_labels)

    return batch

```

Fungsi **train_transforms** dan **valid_transforms** mengonversi gambar menjadi format RGB, menerapkan transformasi, dan mengubah label menjadi one-hot encoding. Keduanya memastikan gambar dan label siap digunakan dalam pelatihan dan validasi model.

```

train_dataset = dataset['train'].with_transform(train_transforms)
valid_dataset = dataset['validation'].with_transform(valid_transforms)
test_dataset = dataset['test'].with_transform(valid_transforms)

len(train_dataset), len(valid_dataset), len(test_dataset)

```

→ (2501, 2510, 4952)

Kode tersebut menerapkan transformasi pada dataset pelatihan, validasi, dan pengujian menggunakan **with_transform**, kemudian menghitung dan menampilkan jumlah sampel dalam setiap subset dataset yang telah ditransformasi.

Data collection

```

def collate_fn(batch):
    return {
        'pixel_values': torch.stack([x['pixel_values'] for x in
batch]),
        'labels': torch.stack([x['labels'] for x in batch]).float()
    }

```

Fungsi **collate_fn** menggabungkan sampel gambar dan label dalam dataset menjadi satu batch, dengan mengonversi **pixel_values** dan **labels** menjadi tensor yang siap digunakan oleh model.

```

def param_count(model):
    params = [(p.numel(), p.requires_grad) for p in model.parameters()]
    trainable = sum([count for count, trainable in params if
trainable])

```

```

total = sum([count for count, _ in params])
frac = (trainable / total) * 100
return total, trainable, frac

```

Fungsi **param_count** menghitung jumlah total parameter, parameter yang dapat dilatih, dan persentase parameter yang dapat dilatih dalam model.

```

def train(model_name, batch_size=16, epochs=1, lr=2e-4):
    """
    contains all of our training loops.
    1. define Accelerator instance
    2. define dataloaders, model, optimizer, loss function, scheduler
    3. write training, validation and testing loop.
    """

    accelerator = Accelerator() # create instance

    # define dataloaders

    train_dl = torch.utils.data.DataLoader(
        train_dataset,
        batch_size = batch_size, # the batch_size will be per-device
        shuffle=True,
        num_workers=4,
        collate_fn=collate_fn
    )

    valid_dl = torch.utils.data.DataLoader(
        valid_dataset,
        batch_size = batch_size*2,
        shuffle=False,
        num_workers=4,
        collate_fn=collate_fn
    )

    test_dl = torch.utils.data.DataLoader(
        test_dataset,
        batch_size = batch_size*2,
        shuffle=False,
        num_workers=4,
        collate_fn=collate_fn
    )

    # timm model
    model = create_model(
        model_name,
        pretrained = True,
        num_classes = 20
    ).to(accelerator.device) # device placement: accelerator.device

```



```

total, trainable, frac = param_count(model)
accelerator.print(f"{total = :}, {trainable = :}, {frac:.2f}%")

# loss, optimizer, scheduler

loss_fn = nn.BCEWithLogitsLoss()

optimizer =
torch.optim.AdamW(model.parameters(), lr=lr, weight_decay=0.02)

scheduler = get_cosine_schedule_with_warmup(
    optimizer,
    num_warmup_steps = int(0.1 * len(train_dl)),
    num_training_steps=len(train_dl)
)

model, optimizer, scheduler, train_dl, valid_dl, test_dl =
accelerator.prepare(
    model, optimizer, scheduler, train_dl, valid_dl, test_dl
)

# loops for number of epochs
for epoch in range(1, epochs+1):

    model.train() # set model to train

    train_metric = evaluate.load('roc_auc', 'multilabel') # load
metric

    running_loss = 0.

    for batch in train_dl:

        logits = model(batch['pixel_values'])

        loss = loss_fn(logits, batch['labels'])
        accelerator.backward(loss) # backpropagation
        optimizer.step() # update weights
        scheduler.step() # update LR
        optimizer.zero_grad() # set grad values to zero

        running_loss += loss.item() # keep track of loss

    # prepare for metrics
    logits, labels = accelerator.gather_for_metrics(
        (logits, batch['labels'])

```

```

        )
        train_metric.add_batch(references=labels,
prediction_scores=logits)

    # loss and metric over 1 epoch
    train_loss = running_loss / len(train_dl)
    train_roc_auc =
train_metric.compute(average='micro')['roc_auc']

    accelerator.print(f"\n{epoch = }")
    accelerator.print(f"{train_loss = :.3f} | {train_roc_auc =
:.3f}")

    # validation loop

    model.eval() # set model for evaluation

    running_loss = 0.
    valid_metric = evaluate.load('roc_auc', 'multilabel')

    for batch in valid_dl:

        with torch.no_grad():
            logits = model(batch['pixel_values'])

            loss = loss_fn(logits, batch['labels'])
            running_loss += loss.item()

            logits, labels = accelerator.gather_for_metrics(
                (logits, batch['labels'])
            )
            valid_metric.add_batch(references=labels,
prediction_scores=logits)

    valid_loss = running_loss / len(valid_dl)
    valid_roc_auc =
valid_metric.compute(average='micro')['roc_auc']

    accelerator.print(f"{valid_loss = :.3f} | {valid_roc_auc =
:.3f}")

    # save model
    accelerator.save_model(model, f'./{model_name}-pascal')

    # testing loop after all epochs are over

```

```

test_metric = evaluate.load('roc_auc','multilabel')

for batch in test_dl:

    with torch.no_grad():
        logits = model(batch['pixel_values'])

        logits, labels = accelerator.gather_for_metrics(
            (logits, batch['labels']))
        )
        test_metric.add_batch(references=labels,
prediction_scores=logits)

test_roc_auc = test_metric.compute(average='micro')['roc_auc']

accelerator.print(f"\n\nTEST AUROC: {test_roc_auc:.3f}")

```

Fungsi **train** menjalankan seluruh proses pelatihan model, mulai dari mempersiapkan data hingga evaluasi. Pertama, fungsi ini membuat **Accelerator** untuk mendukung pelatihan terdistribusi. Kemudian, data dibagi menjadi batch menggunakan **DataLoader** untuk pelatihan, validasi, dan pengujian. Model didefinisikan menggunakan pustaka **timm** dengan parameter yang sesuai. Setelah itu, fungsi ini mengonfigurasi **loss function**, **optimizer**, dan **scheduler**. Proses pelatihan dilakukan dalam beberapa epoch dengan perhitungan **roc_auc** untuk metrik. Selama pelatihan, model diperbarui dan disimpan setiap epoch, dan setelah pelatihan, evaluasi dilakukan pada data uji. Fungsi ini mengakhiri dengan menampilkan **AUROC** untuk evaluasi akhir pada dataset uji.

```

model_name = 'swin_s3_base_224'
notebook_launcher(train, (model_name,8,5,5e-5), num_processes = 2)

```

Kode tersebut mencoba menjalankan pelatihan model secara paralel dengan **notebook_launcher**, tetapi muncul error **FileNotFoundError** karena sistem tidak dapat menemukan perintah **nvidia-smi**, yang diperlukan untuk mengakses GPU. Error ini terjadi karena tidak adanya akses ke GPU atau perintah tersebut di lingkungan saat ini.

```

# intialize the model

model = create_model(
    model_name,
    num_classes=20
)

```

Kode tersebut menginisialisasi model dengan menggunakan fungsi **create_model** dari pustaka **timm**. Model yang dibuat sesuai dengan nama **model_name** yang diberikan, dan

jumlah kelas output ditentukan sebesar 20, sesuai dengan jumlah kelas pada dataset. Model ini siap untuk digunakan dalam pelatihan atau evaluasi.

```
from safetensors.torch import load_model
```

Kode tersebut mengimpor **load_model** dari **safetensors.torch** untuk memuat model yang disimpan dalam format **safetensors** ke dalam **PyTorch**.

```
load_model(model, f'./{model_name}-pascal/model.safetensors')
```

Kode tersebut mencoba memuat model yang disimpan dalam format **safetensors** dengan menggunakan fungsi **load_model**. Model yang ingin dimuat diharapkan berada di direktori **./swin_s3_base_224-pascal/model.safetensors**. Namun, error **FileNotFoundError** muncul karena file **model.safetensors** tidak ditemukan di lokasi tersebut. Ini berarti file model yang diinginkan tidak ada di direktori yang ditentukan, yang menyebabkan kegagalan dalam memuat model.

```
def show_predictions(rows=2, cols=4):
    model.eval()
    samples = test_dataset.shuffle().select(np.arange(rows*cols))
    fig = plt.figure(figsize=(cols*4, rows*4))

    for i in range(rows*cols):

        img = samples[i]['image']
        inputs = samples[i]['pixel_values'].unsqueeze(0)
        labels = samples[i]['classes']
        labels = ', '.join([id2label[lb] for lb in labels])

        with torch.no_grad():
            logits = model(inputs)

        # apply sigmoid activation to convert logits to probabilities
        # getting labels with confidence threshold of 0.5
        predictions = logits.sigmoid() > 0.5

        # converting one-hot encoded predictions back to list of
labels
        predictions = predictions.float().numpy().flatten() # convert
boolean predictions to float
        pred_labels = np.where(predictions==1)[0] # find indices where
prediction is 1
        pred_labels = ', '.join([id2label[label] for label in
pred_labels]) # converting integer labels to string

        label = f"labels: {labels}\npredicted: {pred_labels}"
        fig.add_subplot(rows, cols, i+1)
```



```
import torch
import torch.nn as nn

from huggingface_hub import notebook_login

from datasets import load_dataset, DatasetDict

from transformers import AutoImageProcessor, ViTForImageClassification

from transformers import Trainer, TrainingArguments

import evaluate
```

- **io**: Digunakan untuk manipulasi input/output berbasis file, misalnya membaca file gambar dari memori.
- **PIL.Image**: Digunakan untuk memproses dan memanipulasi gambar, seperti membuka, mengubah format, atau memodifikasi gambar.
- **numpy**: Library untuk komputasi numerik, termasuk operasi pada array dan matriks.
- **pandas**: Library untuk manipulasi dan analisis data berbasis tabel (DataFrame).
- **matplotlib.pyplot**: Digunakan untuk visualisasi data seperti membuat grafik atau plot.
- **torch & torch.nn**: Pustaka PyTorch untuk penghitungan tensor dan membangun model pembelajaran mesin, termasuk definisi lapisan jaringan saraf.
- **huggingface_hub.notebook_login**: Untuk autentikasi ke Hugging Face Hub dalam lingkungan notebook, memungkinkan pengguna mengunggah atau mengunduh model/dataset.
- **datasets.load_dataset & DatasetDict**: Untuk memuat dataset dari pustaka Hugging Face Datasets dan mengelolanya dalam struktur yang terorganisasi (train, test, valid).
- **transformers.AutoImageProcessor**: Komponen otomatis dari Hugging Face untuk memproses gambar sesuai dengan model tertentu.
- **transformers.ViTForImageClassification**: Model Vision Transformer (ViT) untuk tugas klasifikasi gambar.
- **transformers.Trainer & TrainingArguments**: Alat dari Transformers untuk melatih model secara mudah dengan konfigurasi fleksibel.

- **evaluate:** Pustaka untuk menghitung metrik evaluasi pada tugas pembelajaran mesin, seperti akurasi atau F1-score.

```
# Login onto Hugging Face hub to load any private dataset/model.
# We need to login as we'll also upload our model to the hub
notebook_login()
```

notebook_login: Untuk autentikasi dan mengakses dataset atau model yang memerlukan izin.

```
dataset = load_dataset('pcuenq/oxford-pets')
dataset
```

Kode tersebut memuat *Oxford-IIIT Pet Dataset* menggunakan Hugging Face dan menyimpannya sebagai objek `DatasetDict` dengan pembagian data seperti *train* dan *test*. Menampilkan dataset akan menunjukkan informasi dataset, seperti

```
dataset['train'][0]
```

`dataset['train'][0]` mengakses sampel pertama dari subset **train** dalam dataset Oxford Pets. Outputnya adalah dictionary berisi 'image' (gambar hewan peliharaan) dan 'label' (kelas hewan). Ini digunakan untuk memeriksa data atau memvisualisasikan gambar sebelum preprocessing dan pelatihan model.

```
labels = dataset['train'].unique('label')
print(len(labels), labels)
```

Kode ini mengekstrak daftar unik label dari bagian *train* dalam dataset menggunakan fungsi `unique()` pada kolom 'label'. Hasilnya adalah daftar semua kelas hewan peliharaan dalam dataset, yang terdiri dari 37 label, termasuk berbagai ras kucing dan anjing seperti *Siamese*, *Birman*, dan *shiba inu*. Output juga mencetak jumlah total label (37) diikuti oleh daftar nama label tersebut. Hal ini berguna untuk memahami distribusi kelas dalam dataset.

```
def show_samples(ds, rows, cols):
    samples = ds.shuffle().select(np.arange(rows*cols)) # selecting
    random images
    fig = plt.figure(figsize=(cols*4, rows*4))
    # plotting
    for i in range(rows*cols):
        img_bytes = samples[i]['image']['bytes']
        img = Image.open(io.BytesIO(img_bytes))
        label = samples[i]['label']
        fig.add_subplot(rows, cols, i+1)
        plt.imshow(img)
        plt.title(label)
        plt.axis('off')

show_samples(dataset['train'], rows=3, cols=5)
```



Fungsi `show_samples` menampilkan sampel gambar acak dari dataset dalam grid dengan jumlah baris dan kolom yang ditentukan. Gambar diambil dari atribut **bytes**, diubah menjadi objek gambar dengan **Pillow**, dan ditampilkan bersama labelnya. Fungsi ini memproses dataset pelatihan dan menampilkan 15 gambar dalam grid 3x5.

Preprocessing our dataset

```
split_dataset = dataset['train'].train_test_split(test_size=0.2) # 80%
train, 20% evaluation
eval_dataset = split_dataset['test'].train_test_split(test_size=0.5) #
50% validation, 50% test

# recombining the splits using a DatasetDict

our_dataset = DatasetDict({
    'train': split_dataset['train'],
    'validation': eval_dataset['train'],
    'test': eval_dataset['test']
})

our_dataset
DatasetDict({
  train: Dataset({
    features: ['path', 'label', 'dog', 'image'],
    num_rows: 5912
  })
  validation: Dataset({
    features: ['path', 'label', 'dog', 'image'],
    num_rows: 739
  })
  test: Dataset({
    features: ['path', 'label', 'dog', 'image'],
    num_rows: 739
  })
})
```


Kode tersebut membagi dataset pelatihan menjadi tiga bagian: **train**, **validation**, dan **test**. Pertama, dataset pelatihan dibagi menjadi 80% data pelatihan dan 20% data evaluasi menggunakan **train_test_split**. Kemudian, data evaluasi dibagi lagi secara merata (50%-50%) menjadi data validasi dan data pengujian. Ketiga bagian tersebut digabung kembali dalam sebuah objek **DatasetDict** untuk mempermudah pengelolaan dataset selama proses pelatihan dan evaluasi.

Hasil output menunjukkan bahwa dataset baru memiliki tiga bagian:

- **train**: 5912 baris (80% dari dataset awal),
- **validation**: 739 baris,
- **test**: 739 baris. Masing-masing bagian mencakup fitur seperti **path**, **label**, **dog**, dan **image**.

```
label2id = {c:idx for idx,c in enumerate(labels)}  
id2label = {idx:c for idx,c in enumerate(labels)}
```

Kode tersebut membuat dua dictionary: **label2id** yang memetakan label ke ID numerik, dan **id2label** yang memetakan ID numerik kembali ke label. Ini mempermudah konversi antara label tekstual dan ID numerik selama pelatihan model.

Image Processor

```
processor = AutoImageProcessor.from_pretrained('google/vit-base-  
patch16-224')  
processor
```

Kode tersebut memuat model pre-trained **AutoImageProcessor** dari Hugging Face dengan nama model **google/vit-base-patch16-224**, yang merupakan processor untuk memproses gambar menggunakan arsitektur Vision Transformer (ViT). **AutoImageProcessor** digunakan untuk mengonversi gambar menjadi format yang kompatibel untuk input model ViT, seperti normalisasi pixel dan resizing.

```
def transforms(batch):  
    batch['image'] =  
    [Image.open(io.BytesIO(x['bytes'])).convert('RGB') for x in  
    batch['image']]  
    inputs = processor(batch['image'], return_tensors='pt')  
    inputs['labels']=[label2id[y] for y in batch['label']]  
    return inputs
```

Fungsi **transforms** mengonversi gambar dari format bytes menjadi objek gambar RGB, memprosesnya menjadi tensor menggunakan **processor**, dan mengubah label menjadi ID

menggunakan **label2id**. Hasilnya adalah dictionary dengan pixel values dan labels yang siap digunakan.

```
processed_dataset = our_dataset.with_transform(transforms)
```

Kode **processed_dataset = our_dataset.with_transform(transforms)** menerapkan fungsi **transforms** yang telah dibuat sebelumnya ke seluruh dataset **our_dataset**. Ini berarti setiap gambar dalam dataset akan diproses menjadi format yang sesuai (RGB) dan labelnya akan diubah menjadi ID yang relevan. Hasilnya adalah dataset yang telah diproses dan siap digunakan untuk pelatihan model.

Data Collation

```
def collate_fn(batch):
    return {
        'pixel_values': torch.stack([x['pixel_values'] for x in
batch]),
        'labels': torch.tensor([x['labels'] for x in batch])
    }
```

Fungsi **collate_fn** menggabungkan data dalam batch dengan mengonversi gambar ke dalam tensor menggunakan **torch.stack** dan label menggunakan **torch.tensor**. Ini mempermudah pengolahan data saat pelatihan model.

```
accuracy = evaluate.load('accuracy')
def compute_metrics(eval_preds):
    logits, labels = eval_preds
    predictions = np.argmax(logits,axis=1)
    score = accuracy.compute(predictions=predictions,
references=labels)
    return score
```

Fungsi **compute_metrics** menghitung akurasi dengan mengonversi logits menjadi prediksi kelas dan membandingkannya dengan label yang benar, menggunakan modul **evaluate** untuk mengukur akurasi.

Loading our Model

```
model = ViTForImageClassification.from_pretrained(
    'google/vit-base-patch16-224',
    num_labels = len(labels),
    id2label = id2label,
    label2id = label2id,
    ignore_mismatched_sizes = True
)
```

Kode ini memuat model **ViT** pre-trained dari Hugging Face, mengatur jumlah label, serta menambahkan kamus **id2label** dan **label2id** untuk konversi label. Parameter

`ignore_mismatched_sizes` diatur ke `True` untuk mengabaikan ketidaksesuaian ukuran saat memuat model.

```
model
```

Kode tersebut menginisialisasi model **ViTForImageClassification** dengan pre-trained weights dari `google/vit-base-patch16-224`, dan mengonfigurasi label serta pemetaan ID. Model ini siap digunakan untuk klasifikasi gambar.

```
for name, p in model.named_parameters():
    if not name.startswith('classifier'):
        p.requires_grad = False
```

Kode tersebut mengatur agar parameter model selain bagian "classifier" tidak dapat diubah selama proses pelatihan. Ini dilakukan dengan menyet `requires_grad = False` untuk parameter yang tidak memiliki nama yang dimulai dengan 'classifier', yang berarti hanya bagian classifier yang akan diperbarui selama pelatihan.

```
num_params = sum([p.numel() for p in model.parameters()])
trainable_params = sum([p.numel() for p in model.parameters() if
p.requires_grad])

print(f"{num_params = :}, {trainable_params = :}")
num_params = 85,827,109 | trainable_params = 28,453
```

Kode ini menghitung jumlah total parameter (`num_params`) dalam model dan jumlah parameter yang dapat dilatih (`trainable_params`). `num_params` mencakup semua parameter di model, sedangkan `trainable_params` hanya mencakup parameter yang memiliki `requires_grad = True`, yang berarti dapat diperbarui selama pelatihan. Output menunjukkan bahwa model memiliki 85,827,109 parameter secara total, dan hanya 28,453 parameter yang dapat dilatih.

```
training_args = TrainingArguments(
    output_dir="anitaa27",
    per_device_train_batch_size=16,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_steps=100,
    num_train_epochs=5,
    learning_rate=3e-4,
    save_total_limit=2,
    remove_unused_columns=False,
    push_to_hub=True,
    report_to='tensorboard',
    load_best_model_at_end=True,
)
```

Kode ini mengonfigurasi pengaturan pelatihan model, termasuk ukuran batch, strategi evaluasi dan penyimpanan, jumlah epoch, laju pembelajaran, dan pengaturan pelaporan ke TensorBoard. Model terbaik disimpan di akhir pelatihan dan dipublikasikan ke Hugging Face Hub.

```
trainer = Trainer(  
    model=model,  
    args=training_args,  
    data_collator=collate_fn,  
    compute_metrics=compute_metrics,  
    train_dataset=processed_dataset["train"],  
    eval_dataset=processed_dataset["validation"],  
    tokenizer=processor  
)
```

Kode ini mengonfigurasi **Trainer** untuk melatih model dengan dataset dan pengaturan yang telah ditentukan. Code ini mengalami error, Error **403 Forbidden** terjadi karena token yang digunakan tidak memiliki izin untuk membuat model di namespace "**anitaa27**" di Hugging Face.

```
trainer.train()
```

Kode **trainer.train()** memulai proses pelatihan model dengan pengaturan yang telah ditentukan, termasuk data, epoch, dan metrik evaluasi. Code ini mengalami error dikarenakan codingan sebelumnya terjadi error.

```
trainer.evaluate(processed_dataset['test'])
```

Kode ini mengevaluasi model menggunakan dataset pengujian yang telah diproses, dan mengembalikan hasil kinerja model. Namun, error **NameError** terjadi karena objek **trainer** tidak terdefinisi sebelumnya dan juga error di codingan sebelumnya.

```
def show_predictions(rows,cols):  
    samples =  
our_dataset['test'].shuffle().select(np.arange(rows*cols))  
    processed_samples = samples.with_transform(transforms)  
    predictions =  
trainer.predict(processed_samples).predictions.argmax(axis=1) #  
predicted labels from logits  
    fig = plt.figure(figsize=(cols*4,rows*4))  
    for i in range(rows*cols):  
        img_bytes = samples[i]['image']['bytes']  
        img = Image.open(io.BytesIO(img_bytes))  
        prediction = predictions[i]  
        label = f"label: {samples[i]['label']}\npredicted:  
{id2label[prediction]}"  
        fig.add_subplot(rows,cols,i+1)
```

```
plt.imshow(img)
plt.title(label)
plt.axis('off')
```

```
show_predictions(rows=5,cols=5)
```

Kode ini bertujuan untuk menampilkan beberapa gambar dari dataset uji beserta label sebenarnya dan prediksi yang dihasilkan oleh model. Fungsi `show_predictions` memilih sampel acak, mengolah gambar menggunakan transformasi, lalu memprediksi label menggunakan model dengan `trainer.predict`. Namun, error terjadi karena variabel `trainer` belum didefinisikan sebelumnya dan juga error di codingan sebelumnya.

```
kwargs = {
    "finetuned_from": model.config._name_or_path,
    "dataset": 'pcuenq/oxford-pets',
    "tasks": "image-classification",
    "tags": ['image-classification'],
}
```

Kode ini mendefinisikan dictionary `kwargs` yang berisi metadata tentang model, termasuk model dasar (`finetuned_from`), dataset yang digunakan (`dataset`), jenis tugas (`tasks`), dan label terkait (`tags`). Ini digunakan untuk menyimpan informasi model sebelum diunggah atau digunakan lebih lanjut.

```
trainer.save_model()
trainer.push_to_hub('🐶🐱', **kwargs)
```

Kode ini mencoba untuk menyimpan dan mengunggah model ke Hugging Face Hub menggunakan `trainer.save_model()` dan `trainer.push_to_hub()`. Namun, error `NameError` terjadi karena objek `trainer` belum didefinisikan atau tidak tersedia.

8. Vision Transformers : Semantic Segmentation Transfer Learning

```
!pip install datasets
!pip install evaluate
```

- `!pip install datasets`: Menginstal library `datasets` untuk memuat dataset dari Hugging Face Hub.
- `!pip install evaluate`: Menginstal library `evaluate` untuk mengukur kinerja model.

```
import json
import torch
import datasets
import requests
import evaluate
import numpy as np
import huggingface_hub
```

```

from PIL import Image
import albumentations as A
from tqdm.auto import tqdm
import matplotlib.pyplot as plt
from typing import Tuple, Any
from dataclasses import dataclass
from datasets import load_dataset
import matplotlib.patches as mpatches
from huggingface_hub import hf_hub_download
from torch.utils.data import Dataset, DataLoader
from transformers import (
    MaskFormerImageProcessor,
    AutoImageProcessor,
    MaskFormerForInstanceSegmentation,
)

torch.manual_seed(42)

```

Kode ini mengimpor berbagai pustaka untuk pemrosesan data dan pelatihan model. Beberapa pustaka yang digunakan antara lain torch untuk pengolahan tensor, datasets untuk memuat dataset, evaluate untuk evaluasi, dan transformers untuk model dan pemrosesan gambar. Pustaka lainnya seperti albumentations dan PIL digunakan untuk augmentasi gambar dan manipulasi citra. Selain itu, kode ini juga mengimpor modul untuk memanfaatkan Hugging Face Hub dan menyediakan berbagai utilitas seperti tqdm untuk progres bar dan matplotlib untuk visualisasi. Pustaka MaskFormerForInstanceSegmentation dari Hugging Face digunakan untuk instance segmentation pada gambar.

```

# it seems that we need to login to huggingface to have access to the
dataset segments/sidewalk-semantic
huggingface_hub.notebook_login()

```

Fungsi `huggingface_hub.notebook_login()` akan membuka dialog untuk memasukkan kredensial Hugging Face, yang memungkinkan akses ke dataset atau model yang bersifat pribadi atau terlarang jika tidak login.

Loading and explore the dataset

```

hf_dataset_id = "segments/sidewalk-semantic"
dataset = load_dataset(hf_dataset_id)

dataset = dataset.shuffle(seed=1)
dataset = dataset["train"].train_test_split(test_size=0.2)
train_ds, test_ds = dataset["train"], dataset["test"]

dataset = train_ds.train_test_split(test_size=0.05)
train_ds, val_ds = dataset["train"], dataset["test"]

```

Kode mencoba untuk memuat dataset **segments/sidewalk-semantic** dari Hugging Face menggunakan fungsi **load_dataset**. Kemudian, dataset tersebut diacak dan dibagi menjadi subset pelatihan (train), validasi (val), dan pengujian (test). Namun, muncul error **DatasetNotFoundError** karena dataset yang diminta adalah dataset **gated** (terbatas aksesnya) di Hugging Face. Artinya, pengguna harus meminta izin atau akses untuk menggunakan dataset ini.

```
filename = "id2label.json"
id2label = json.load(
    open(hf_hub_download(hf_dataset_id, filename,
repo_type="dataset"), "r")
)
id2label = {int(k): v for k, v in id2label.items()}
print(id2label)
```

Kode mencoba untuk mengunduh file **id2label.json** dari dataset **segments/sidewalk-semantic** yang ada di Hugging Face menggunakan fungsi **hf_hub_download**. File ini berisi pemetaan antara ID kelas dan label teks. Namun, muncul error **GatedRepoError** yang menunjukkan bahwa akses ke dataset ini dibatasi. Artinya, dataset tersebut adalah **gated dataset** (terbatas) dan pengguna tidak memiliki izin untuk mengaksesnya. Pengguna perlu mengunjungi halaman dataset tersebut di Hugging Face untuk meminta akses.

```
example = train_ds[0]
print(example)
segmentation_map = np.array(example["label"])
image_array = np.array(example["pixel_values"])
print(
    f"Shape : Image: {image_array.shape} - Segmentation map: {segmentation_map.shape}"
)

print(segmentation_map)
```

Error **NameError** terjadi karena variabel **train_ds** belum didefinisikan atau tidak ada dan juga dikarenakan error pada codingan sebelumnya.

```
def show_samples(dataset: datasets.Dataset, n: int = 5):
    """
    Displays 'n' samples from the dataset.
    ----
    Args:
        - dataset: The dataset which should contain 'pixel_values' and 'label' in its items.
        - n (int): Number of samples to display.
```

```

"""
if n > len(dataset):
    raise ValueError("n is larger than the dataset size")

fig, axs = plt.subplots(n, 2, figsize=(10, 5 * n))

for i in range(n):
    sample = dataset[i]
    image, label = np.array(sample["pixel_values"]),
sample["label"]

    axs[i, 0].imshow(image)
    axs[i, 0].set_title("Image")
    axs[i, 0].axis("off")

    axs[i, 1].imshow(image)
    axs[i, 1].imshow(label, cmap="nipy_spectral", alpha=0.5)
    axs[i, 1].set_title("Segmentation Map")
    axs[i, 1].axis("off")

plt.tight_layout()
plt.show()

```

Fungsi `show_samples` menampilkan gambar dan peta segmentasi dari dataset. Fungsi ini menerima argumen `dataset` dan `n` (jumlah sampel yang ditampilkan). Setiap sampel ditampilkan dalam dua kolom: gambar asli dan peta segmentasi dengan transparansi. Fungsi ini juga memeriksa apakah jumlah sampel yang diminta melebihi ukuran dataset.

```
show_samples(train_ds, n=4)
```

Kode tersebut mencoba memanggil fungsi `show_samples` dengan dataset `train_ds`, namun error `NameError: name 'train_ds' is not defined` muncul karena variabel `train_ds` belum didefinisikan sebelumnya.

Prepare the Dataset for training & Sanity checks

```

preprocessor = MaskFormerImageProcessor(
    ignore_index=0,
    do_reduce_labels=False,
    do_resize=False,
    do_rescale=False,
    do_normalize=False,
)
ade_mean = np.array([123.675, 116.280, 103.530]) / 255
ade_std = np.array([58.395, 57.120, 57.375]) / 255

train_transform = A.Compose(
    [

```



```

        A.RandomCrop(width=512, height=512),
        A.HorizontalFlip(p=0.5),
        A.Normalize(mean=ade_mean, std=ade_std),
    ]
)

test_transform = A.Compose(
    [
        A.Resize(width=512, height=512),
        A.Normalize(mean=ade_mean, std=ade_std),
    ]
)

@dataclass
class SegmentationDataInput:
    original_image: np.ndarray
    transformed_image: np.ndarray
    original_segmentation_map: np.ndarray
    transformed_segmentation_map: np.ndarray

class SemanticSegmentationDataset(Dataset):
    def __init__(self, dataset: datasets.Dataset, transform: Any) -> None:
        """
        Dataset for Semantic Segmentation.
        ---
        Args:
            - dataset: A dataset containing images and segmentation
maps.
            - transform: A transformation function to apply to the
images and segmentation maps.
        """
        self.dataset = dataset
        self.transform = transform

    def __len__(self) -> int:
        return len(self.dataset)

    def __getitem__(self, idx: int) -> Tuple[np.ndarray, np.ndarray]:
        sample = self.dataset[idx]
        original_image = np.array(sample["pixel_values"])
        original_segmentation_map = np.array(sample["label"])

        transformed = self.transform(
            image=original_image, mask=original_segmentation_map
        )
        transformed_image = transformed["image"].transpose(

```

```

        2, 0, 1
    ) # Transpose to channel-first format
    transformed_segmentation_map = transformed["mask"]

    return SegmentationDataInput(
        original_image=original_image,
        transformed_image=transformed_image,
        original_segmentation_map=original_segmentation_map,
        transformed_segmentation_map=transformed_segmentation_map,
    )

def collate_fn(batch: SegmentationDataInput) -> dict:
    original_images = [sample.original_image for sample in batch]
    transformed_images = [sample.transformed_image for sample in
batch]
    original_segmentation_maps = [sample.original_segmentation_map for
sample in batch]
    transformed_segmentation_maps = [
        sample.transformed_segmentation_map for sample in batch
    ]

    preprocessed_batch = preprocessor(
        transformed_images,
        segmentation_maps=transformed_segmentation_maps,
        return_tensors="pt",
    )

    preprocessed_batch["original_images"] = original_images
    preprocessed_batch["original_segmentation_maps"] =
original_segmentation_maps

    return preprocessed_batch

```

Di kode tersebut, terjadi error `ValidationError` karena input yang diberikan untuk parameter `mean` dan `std` berupa array NumPy (`np.ndarray`), padahal `A.Normalize()` membutuhkan input berupa list atau sequence. Pydantic yang digunakan di `albumentations` tidak bisa memvalidasi array tersebut.

Training

```

# pixel level module contains both the backbone and the pixel decoder
for param in model.model.pixel_level_module.parameters():
    param.requires_grad = False

# Confirm that the parameters are correctly frozen
for name, param in model.model.pixel_level_module.named_parameters():
    assert not param.requires_grad

```

Kode ini membekukan parameter di dalam **pixel_level_module** model dengan mengatur **requires_grad = False**, agar parameter tersebut tidak diperbarui selama pelatihan. Kemudian, pengecekan dilakukan untuk memastikan parameter tersebut benar-benar dibekukan.

```
def show_inference_samples(
    model: MaskFormerForInstanceSegmentation,
    dataloader: DataLoader,
    preprocessor: AutoImageProcessor,
    n: int = 5,
):
    """
    Displays 'n' samples from the dataloader with model inference
    results.
    ----
    Args:
        - model: The trained model.
        - dataloader: DataLoader containing the dataset for inference.
        - preprocessor: The preprocessor used for post-processing the
    outputs.
        - n (int): Number of samples to display.

    """
    model.to(device)
    model.eval()

    if n > len(dataloader.dataset):
        raise ValueError("n is larger than the dataset size")

    fig, axs = plt.subplots(n, 2, figsize=(10, 5 * n))

    with torch.no_grad():
        for i, batch in enumerate(dataloader):
            if i >= n:
                break

            pixel_values = batch["pixel_values"].to(device)
            outputs = model(pixel_values=pixel_values)

            original_images = batch["original_images"]
            target_sizes = [
                (image.shape[0], image.shape[1]) for image in
original_images
            ]
            predicted_segmentation_maps = (
                preprocessor.post_process_semantic_segmentation(
                    outputs, target_sizes=target_sizes
                )
            )
```

```

        ground_truth_segmentation_maps =
batch["original_segmentation_maps"]

        # Assuming original_images are numpy arrays and already in
the right format
        image = original_images[i]
        ground_truth_map = ground_truth_segmentation_maps[i]
        predicted_map = predicted_segmentation_maps[i]

        axs[i, 0].imshow(image)
        axs[i, 0].imshow(ground_truth_map, cmap="nipy_spectral",
alpha=0.5)
        axs[i, 0].set_title("Ground Truth")
        axs[i, 0].axis("off")

        axs[i, 1].imshow(image)
        axs[i, 1].imshow(
            predicted_map.cpu().numpy(), cmap="nipy_spectral",
alpha=0.5
        )
        axs[i, 1].set_title("Prediction")
        axs[i, 1].axis("off")

plt.tight_layout()
plt.show()

```

Fungsi `show_inference_samples` menampilkan hasil prediksi model untuk sejumlah sampel. Fungsi ini menjalankan inferensi pada setiap gambar dalam dataloader, memproses hasil prediksi, dan menampilkan gambar bersama dengan peta segmentasi ground truth dan prediksi model dalam bentuk subplot.

```
show_inference_samples(model, test_dataloader, preprocessor, n=2)
```

Fungsi `show_inference_samples` dipanggil untuk menampilkan hasil inferensi pada dua sampel dari data uji. Namun, error `NameError: name 'test_dataloader' is not defined` muncul karena variabel `test_dataloader` belum didefinisikan sebelumnya dalam kode.

Chapter 4

1. CLIP and CROP

```

import os
import torch
import warnings
import requests
import PIL
import numpy as np

```

```
import matplotlib.pyplot as plt
from PIL import Image
from transformers import CLIPProcessor, CLIPModel,
YolosImageProcessor, YolosForObjectDetection
```

Kode tersebut mengimpor beberapa pustaka yang digunakan untuk pemrosesan gambar dan model pembelajaran mesin. Pustaka seperti torch, numpy, dan PIL digunakan untuk manipulasi gambar dan tensor. matplotlib.pyplot digunakan untuk menampilkan gambar, sedangkan transformers mengimpor model dan pemroses dari Hugging Face, termasuk CLIPProcessor, CLIPModel untuk pemrosesan teks-gambar, dan YolosImageProcessor, YolosForObjectDetection untuk deteksi objek menggunakan model YOLO.

```
warnings.simplefilter("ignore")
os.environ["TOKENIZERS_PARALLELISM"] = "true"
```

Kode tersebut menonaktifkan peringatan dengan warnings.simplefilter("ignore") dan mengaktifkan pemrosesan paralel untuk tokenisasi dengan os.environ["TOKENIZERS_PARALLELISM"] = "true", yang mempercepat pemrosesan model dari Hugging Face.

```
DETECTION_MODEL_NAME = "hustvl/yolos-tiny"
MULTIMODAL_MODEL_NAME = "openai/clip-vit-base-patch16"
```

Kode mendefinisikan dua model: DETECTION_MODEL_NAME untuk deteksi objek ("hustvl/yolos-tiny") dan MULTIMODAL_MODEL_NAME untuk model multimodal CLIP ("openai/clip-vit-base-patch16").

```
det_image_processor =
YolosImageProcessor.from_pretrained(DETECTION_MODEL_NAME)
det_model =
YolosForObjectDetection.from_pretrained(DETECTION_MODEL_NAME)
```

Kode ini memuat model YOLOS untuk deteksi objek dan pemrosesan gambar. det_image_processor memproses gambar, sementara det_model adalah model deteksi objek YOLOS yang siap digunakan.

```
mm_model = CLIPModel.from_pretrained(MULTIMODAL_MODEL_NAME)
mm_processor = CLIPProcessor.from_pretrained(MULTIMODAL_MODEL_NAME)
```

Kode ini memuat model CLIP dan prosesoranya. mm_model adalah model CLIP yang digunakan untuk pemrosesan multimodal (teks dan gambar), sedangkan mm_processor digunakan untuk memproses gambar dan teks agar dapat diproses oleh model CLIP. Kedua objek ini diinisialisasi dengan model yang telah dilatih sebelumnya menggunakan nama model MULTIMODAL_MODEL_NAME.

```
url =  
"https://i.pinimg.com/736x/1b/51/42/1b5142c269f2e9a356202af3f5569a87.j  
pg"  
image = Image.open(requests.get(url, stream=True).raw)
```

Kode ini mengunduh gambar dari URL yang diberikan dan membuka gambar tersebut menggunakan pustaka PIL (Python Imaging Library). `requests.get(url, stream=True).raw` digunakan untuk mengambil gambar dari internet secara langsung, dan `Image.open()` digunakan untuk membuka gambar tersebut dalam format yang dapat diproses lebih lanjut dalam kode. Dengan cara ini, gambar dapat digunakan dalam pemrosesan atau analisis lebih lanjut.

```
def extract_list_images_detected(image, prob):  
    """  
    The function `extract_list_images_detected` takes an image and a  
    probability threshold as input,  
    performs object detection on the image using a pre-trained model,  
    and returns a list of cropped  
    images of the detected objects along with their corresponding  
    scores.  
  
    Args:  
    :param image: The `image` parameter is the input image from which  
    you want to extract the detected  
    images. It should be a PIL image object  
    :param prob: The parameter "prob" is the probability threshold for  
    filtering out the detected  
    objects. It is used to determine which objects are considered  
    significant enough to be included in  
    the final list of images. Objects with a probability score higher  
    than the threshold will be  
    included, while objects with a lower score will be filtered out  
  
    Returns:  
    images_list: returns a list of cropped out images.  
    scores: returns a list of confidence scores corresponding scores.  
    """  
  
    inputs = det_image_processor(images=image, return_tensors="pt")  
    outputs = det_model(**inputs)  
  
    logits = outputs.logits  
    bboxes = outputs.pred_boxes  
    probas = outputs.logits.softmax(-1)[0, :, :-1]  
  
    keep = probas.max(-1).values > prob
```

```

    outs = det_image_processor.post_process(outputs,
torch.tensor(image.size[::-1]).unsqueeze(0))
    bboxes_scaled = outs[0]['boxes'][keep].detach().numpy()
    labels = outs[0]['labels'][keep].detach().numpy()
    scores = outs[0]['scores'][keep].detach().numpy()

    images_list = []
    for i,j in enumerate(bboxes_scaled):
        xmin = int(j[0])
        ymin = int(j[1])
        xmax = int(j[2])
        ymax = int(j[3])

        im_arr = np.array(image)
        roi = im_arr[ymin:ymax, xmin:xmax]
        roi_im = Image.fromarray(roi)
        images_list.append(roi_im)

    return images_list, scores

```

Fungsi `extract_list_images_detected` mendeteksi objek dalam gambar dan mengembalikan gambar objek yang terdeteksi beserta skor kepercayaannya. Fungsi ini memproses gambar dengan model deteksi objek, menyaring objek yang memiliki probabilitas lebih tinggi dari ambang batas, dan mengembalikan gambar cropped bersama skor kepercayaan.

```

def extract_image_with_description(images_list, text, scores):
    """
    The function `extract_image_with_description` takes a list of
    images, a text description, and
    scores, and returns the image with the highest score.

    Args:
        images_list: The `images_list` parameter is a list of images. Each
        image can be represented
        as a file path or a URL
        text: The `text` parameter is a string that represents the
        description or caption of the
        image
        scores: The "scores" parameter is a list of scores corresponding
        to each image in the
        "images_list". These scores represent the confidence or relevance
        of each image to the given text

    Returns:
        the image with the highest score (PIL.Image) and its corresponding
        score.
    """

```

```

input_ = mm_processor(text = [text], images=images_list ,
return_tensors="pt", padding=True)
output = mm_model(**input_)
logits_per_image = output.logits_per_text
probs = logits_per_image.softmax(-1)
l_idx = np.argsort(probs[-1].detach().numpy())[::-1][0:1]

final_ims = []
for i,j in enumerate(images_list):
    param_dict = {}
    if i in l_idx:
        param_dict['image'] = images_list[i]
        param_dict['score'] = probs[-1].detach().numpy()[i]
        final_ims.append(param_dict)

fi = sorted(final_ims, key=lambda item: item.get("score"),
reverse=True)
return fi[0]['image'], fi[0]['score']

```

Fungsi `extract_image_with_description` memilih gambar yang paling relevan dengan teks yang diberikan, berdasarkan skor kepercayaan. Fungsi ini mengurutkan gambar berdasarkan kesesuaian dengan teks dan mengembalikan gambar dengan skor tertinggi.

```

list_rois, conf_scores = extract_list_images_detected(image,
prob=0.85)

```

Kode ini memanggil fungsi `extract_list_images_detected` untuk mendeteksi objek dalam gambar dengan ambang batas probabilitas 0.85. Fungsi ini mengembalikan dua hasil: `list_rois` (gambar objek yang terdeteksi) dan `conf_scores` (skor kepercayaan untuk setiap objek).

```

plt.imshow(image)
plt.axis("off")
plt.tight_layout()

```



Kode ini menampilkan gambar dengan menghilangkan sumbu dan memastikan tata letak gambar rapi menggunakan `plt.imshow(image)` dan `plt.axis("off")`, serta `plt.tight_layout()`.

```
TEXT_DESC = ["Man carrying a green bag", "man riding a bicycle",  
"yellow colored taxi car", "red colored bus"]  
images_list = []  
for txt in TEXT_DESC:  
    img_dict = {}  
    img, score = extract_image_with_description(list_rois, txt,  
conf_scores)  
    img_dict['image'] = img  
    img_dict['description'] = txt  
    img_dict['conf-score'] = score  
    images_list.append(img_dict)
```

Kode ini mengambil deskripsi gambar dari `TEXT_DESC` dan menggunakan fungsi `extract_image_with_description` untuk mencari gambar yang relevan dengan setiap deskripsi. Gambar, deskripsi, dan skor kepercayaan disimpan dalam list `images_list`.

```
fig = plt.figure(figsize=(10, 20))  
for i in range(len(images_list)):  
    plt.subplot(1, len(images_list), i + 1)  
    plt.imshow(images_list[i]['image'])  
    plt.title(images_list[i]['description'])  
    plt.xlabel("confidence-score:" + str(round(images_list[i]['conf-  
score'], 3)))  
    plt.xticks(ticks=[])  
    plt.yticks(ticks=[])  
plt.tight_layout()  
plt.show()
```



Kode ini menampilkan gambar-gambar yang relevan dengan deskripsi dan skor kepercayaan mereka dalam satu baris. Setiap gambar diberi judul deskripsi dan skor kepercayaan. Axis X dan Y disembunyikan. Setelah pengaturan layout, gambar ditampilkan dengan `plt.show()`.

2. CLIP transfer learning

```
!pip install evaluate
!pip install datasets
```

- !pip install evaluate: Menginstal library evaluate untuk mengukur kinerja model.
- !pip install datasets: Menginstal library datasets untuk memuat dataset dari Hugging Face Hub.

```
import os
import random
from functools import partial
from typing import Any
import evaluate
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
from datasets import Dataset, DatasetDict, load_dataset, load_metric
from peft import LoraConfig, get_peft_model
from PIL import Image
from torch.utils.data import DataLoader
from tqdm.notebook import tqdm
from transformers import (CLIPImageProcessor, CLIPModel, CLIPProcessor,
                          CLIPTokenizerFast, Trainer, TrainingArguments)
from transformers.tokenization_utils_base import BatchEncoding
from datasets.formatting.formatting import LazyBatch

-----
ImportError                                Traceback (most recent call last)
<ipython-input-14-700e004a5917> in <cell line: 10>()
      8 import torch
      9 import torch.nn as nn
--> 10 from datasets import Dataset, DatasetDict, load_dataset, load_metric
     11 from peft import LoraConfig, get_peft_model
     12 from PIL import Image

ImportError: cannot import name 'load_metric' from 'datasets' (/usr/local/lib/python3.10/dist-packages/datasets/__init__.py)

-----
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
-----
```

Sudah dilakukan installasi terlebih dahulu, namun code tetap error sehingga tidak bisa melanjutkan codingan.

Link Youtube :

<https://youtu.be/eLGIFDvrmHM>