Nama : Andhika Aria Pratama N

NIM : 1103202121

07 PyTorch Experiment Tracking

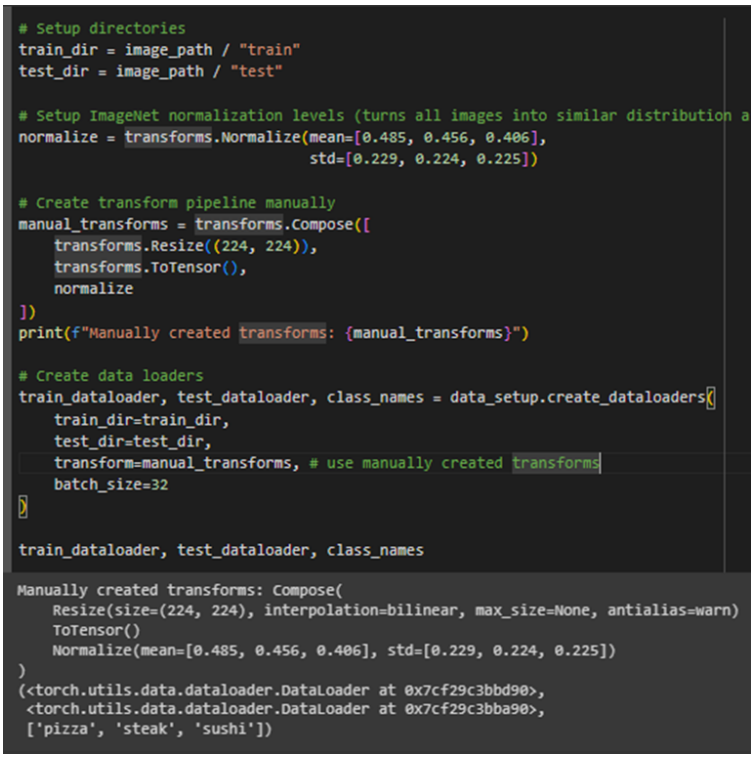
- Menyiapkan pengaturan

Menyiapkan beberapa library yang akan digunakan untuk pengolahan data nantinya.

- Mengunduh data

Mengunduh data berupa data gambar yang berisikan beberapa jenis makanan untuk dataset agar bisa diolah menjadi data. Serta data tersebut dilakukan ektrak data.

- Membuat dataset dan dataloaders



```
# Setup directories
train_dir = image_path / "train"
test_dir = image_path / "test"

# Setup ImageNet normalization levels (turns all images into similar distribution a
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

# Create transform pipeline manually
manual_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    normalize
])
print(f"Manually created transforms: {manual_transforms}")

# Create data loaders
train_dataloader, test_dataloader, class_names = data_setup.create_dataloaders(
    train_dir=train_dir,
    test_dir=test_dir,
    transform=manual_transforms, # use manually created transforms
    batch_size=32
)

train_dataloader, test_dataloader, class_names
```

```
Manually created transforms: Compose(
    Resize(size=(224, 224), interpolation=bilinear, max_size=None, antialias=warn)
    ToTensor()
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
)
(<torch.utils.data.dataloader.DataLoader at 0x7cf29c3bbd90>,
 <torch.utils.data.dataloader.DataLoader at 0x7cf29c3bba90>,
 ['pizza', 'steak', 'sushi'])
```

Kode di atas menyiapkan jalur pelatihan dan pengujian untuk dataset dengan transformasi manual yang mencakup penyesuaian ukuran gambar, konversi ke tensor, dan normalisasi menggunakan parameter rata-rata dan deviasi standar dari ImageNet. Hasilnya adalah transformasi manual yang disertakan dalam pipeline. Selanjutnya, data loader untuk pelatihan dan pengujian dibuat dengan menggunakan transformasi manual tersebut, dan daftar nama kelas juga diperoleh.

- Buat DataLoaders menggunakan transformasi yang dibuat secara otomatis

```
# Setup dirs
train_dir = image_path / "train"
test_dir = image_path / "test"

# Setup pretrained weights (plenty of these available in torchvision.models)
weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT

# Get transforms from weights (these are the transforms that were used to obtain
automatic_transforms = weights.transforms()
print(f"Automatically created transforms: {automatic_transforms}")

# Create data loaders
train_dataloader, test_dataloader, class_names = data_setup.create_dataloaders(
    train_dir=train_dir,
    test_dir=test_dir,
    transform=automatic_transforms, # use automatic created transforms
    batch_size=32
)

train_dataloader, test_dataloader, class_names
```

```
Automatically created transforms: ImageClassification(
    crop_size=[224]
    resize_size=[256]
    mean=[0.485, 0.456, 0.406]
    std=[0.229, 0.224, 0.225]
    interpolation=InterpolationMode.BICUBIC
)
(<torch.utils.data.dataloader.DataLoader at 0x7cf29c3bb610>,
 <torch.utils.data.dataloader.DataLoader at 0x7cf29c3bb250>,
 ['pizza', 'steak', 'sushi'])
```

Kode di atas menyiapkan jalur pelatihan dan pengujian untuk dataset dengan transformasi otomatis yang berasal dari berat pra-pelatihan model EfficientNet_B0. Transformasi ini mencakup penyesuaian ukuran gambar, cropping, normalisasi, dan parameter transformasi lainnya yang telah digunakan saat menghasilkan berat tersebut. Selanjutnya, data loader untuk pelatihan dan pengujian dibuat menggunakan transformasi otomatis tersebut, dan daftar nama kelas juga diperoleh.

- Mendapatkan model terlatih, membekukan lapisan dasar, dan mengubah kepala pengklasifikasi

```
# Note: This is how a pretrained model would be created in torchvision > 
# model = torchvision.models.efficientnet_b0(pretrained=True).to(device)

# Download the pretrained weights for EfficientNet_B0
weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT # NEW in tor

# Setup the model with the pretrained weights and send it to the target d
model = torchvision.models.efficientnet_b0(weights=weights).to(device)

# View the output of the model
# model
```

```
# Freeze all base layers by setting requires_grad attribute to False
for param in model.features.parameters():
    param.requires_grad = False

# Since we're creating a new layer with random weights (torch.nn.Linear),
# let's set the seeds
set_seeds()

# Update the classifier head to suit our problem
model.classifier = torch.nn.Sequential(
    nn.Dropout(p=0.2, inplace=True),
    nn.Linear(in_features=1280,
              out_features=len(class_names),
              bias=True).to(device))
```

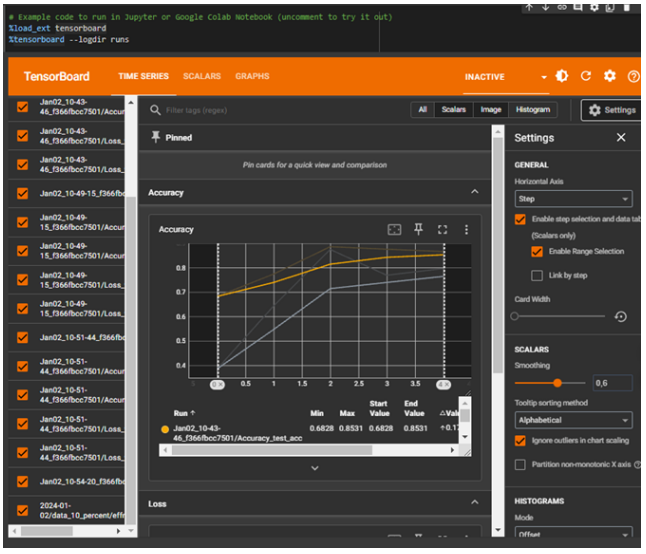
Kode di atas merupakan tahap untuk mengunduh berat pra-pelatihan EfficientNet_B0 dari torchvision, lalu membuat model baru dengan menggunakan berat tersebut. Langkah selanjutnya adalah membekukan semua lapisan dasar (features) model dengan mengatur atribut requires_grad menjadi False. Kemudian, sebuah lapisan pengklasifikasi baru dengan dropout dan linear layer ditambahkan

untuk memecahkan masalah klasifikasi sesuai dengan jumlah kelas yang ada dalam dataset. Sebagai langkah terakhir, biji acak diatur untuk menghasilkan hasil yang dapat direproduksi.

- Latih model dan lacak hasilnya

Pada tahap ini, sebuah fungsi `train` dibuat untuk melatih dan menguji model PyTorch. Model dilatih dan diuji selama lima epoch menggunakan fungsi `train_step` dan `test_step`. Selama pelatihan, metrik seperti loss dan akurasi dicetak untuk setiap epoch. Epoch yang dihasilkan memiliki nilai akhir yaitu train_loss sebesar 64% hasil tersebut setiap epochnya selalu menurun, train accuracy memiliki nilai hasil akhir sebesar 79% nilai tersebut selalu meningkat, begitu juga dengan test_loss memiliki nilai sebesar 60% sama seperti train_loss selalu menurun, begitu juga dengan test accuracy mendapatkan nilai 86% setiap epochnya selalu meningkat. Selain itu, hasil pelatihan dan pengujian disimpan dalam sebuah dictionary yang kemudian dihasilkan sebagai output. Terdapat pula penggunaan modul `SummaryWriter` dari TensorBoard untuk melacak dan menyimpan metrik pelatihan seperti loss dan akurasi. Proses ini dapat membantu analisis performa model secara interaktif. Seluruh pelatihan dan metrik disertakan dalam output, sehingga memudahkan evaluasi dan pemahaman performa model.

- Melihat hasil model pada tensorboard



Pada tahap ini mengaktifkan ekstensi TensorBoard pada lingkungan notebook, seperti Jupyter atau Google Colab, dengan menggunakan magic command `%load_ext tensorboard`. Kemudian, dengan menggunakan `%tensorboard --logdir runs`, TensorBoard diluncurkan dan diarahkan untuk mengamati direktori log yang disebut "runs". Fungsi dari ini adalah memberikan antarmuka grafis untuk menganalisis dan memvisualisasikan metrik eksperimen, seperti loss dan akurasi, yang dicatat selama pelatihan model dengan TensorBoard. Dengan TensorBoard, pengguna dapat melihat dinamika pelatihan, memeriksa arsitektur model, serta memahami perilaku dan kinerja model secara lebih mendalam.

- Buat fungsi pembantu untuk membangun instance SummaryWriter()

Pada tahap ini mengenalkan fungsi `create_writer` yang digunakan untuk membuat instance `SummaryWriter` dari TensorBoard, dengan menentukan log directory yang disusun berdasarkan

eksperimen, model, dan tambahan informasi seperti jumlah epochs. Fungsi ini membantu mengorganisir dan menyimpan log eksperimen secara terstruktur.

Selanjutnya, fungsi `train` diperbarui dengan menambahkan parameter `writer` yang merupakan instance `SummaryWriter`. Fungsi tersebut kemudian menggunakan `writer` untuk menyimpan dan melog metrik pelatihan, seperti loss dan akurasi, pada setiap iterasi epoch. Hal ini memberikan kemampuan untuk memvisualisasikan dan melacak performa model secara dinamis menggunakan TensorBoard.

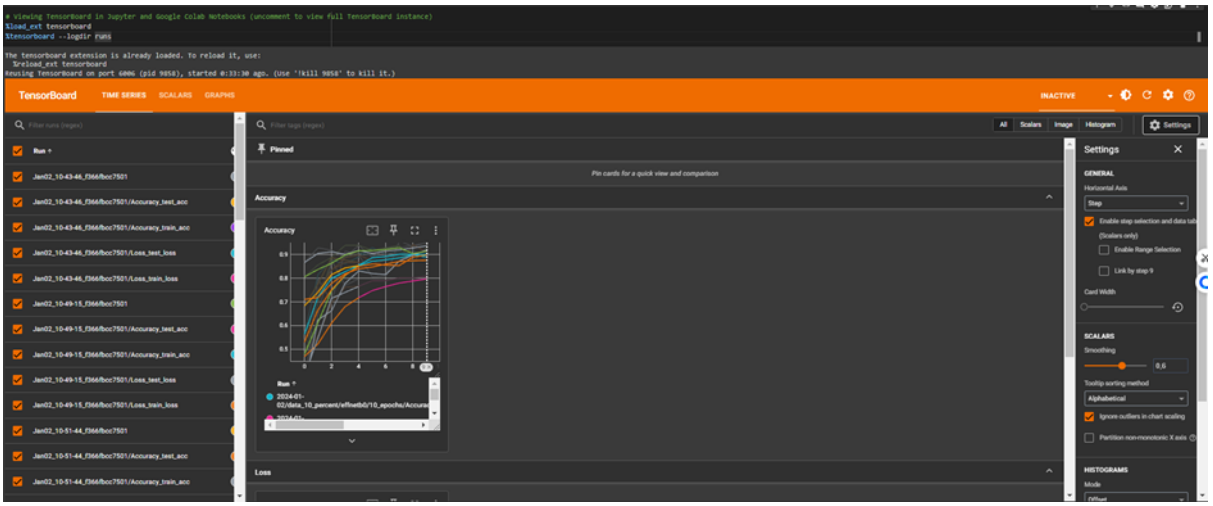
Penggunaan kedua fungsi ini terlihat dalam contoh penggunaan, di mana `create_writer` digunakan untuk membuat instance `SummaryWriter` untuk eksperimen tertentu, dan writer tersebut kemudian diteruskan sebagai parameter ke fungsi `train`. Dengan demikian, hasil pelatihan dapat dipantau dan divisualisasikan secara real-time melalui TensorBoard.

- Menyiapkan serangkaian percobaan pemodelan

Dalam tahap ini, dilakukan eksperimen pelatihan model deep learning menggunakan dua arsitektur EfficientNet, yaitu EfficientNetB0 dan EfficientNetB2, pada dua dataset yang berbeda, yaitu dataset dengan 10% dan 20% dari jumlah data asli. Eksperimen dilakukan dengan jumlah epoch yang berbeda (5 dan 10) untuk setiap kombinasi model dan dataset. Setiap eksperimen melibatkan pembuatan model baru, pelatihan model dengan dataset yang sesuai, pengujian pada dataset uji yang sama, pencatatan hasil pelatihan menggunakan TensorBoard, dan penyimpanan model terbaik.

Hasil eksperimen menunjukkan informasi tentang performa setiap model pada setiap epoch, termasuk kerugian (loss) dan akurasi pada data pelatihan dan pengujian. Model terbaik dari setiap eksperimen disimpan dalam format berkas pth. Selain itu, terdapat penggunaan TensorBoard untuk mencatat dan memvisualisasikan metrik pelatihan seperti loss dan akurasi pada setiap epoch. Total waktu pelatihan dan eksperimen untuk semua kombinasi model dan dataset juga dicatat.

- Melihat hasil pada tensorboard



Menggunakan tensorboard untuk visualisasi hasil eksperimen.

- Muat model terbaik dan buat prediksi dengannya

Pada tahap ini, dilakukan pengujian model terbaik (EfficientNetB2) yang telah dilatih pada dataset 20% dengan 10 epochs. Pertama, model terbaik diinisialisasi dan kemudian dibaca state_dict()-nya dari berkas yang telah disimpan. Ukuran berkas model tersebut dihitung dan dicetak dalam megabyte. Selanjutnya, tiga gambar acak dari dataset pengujian 20% dipilih, dan untuk setiap gambar, dilakukan prediksi menggunakan model terbaik dan hasilnya ditampilkan bersama dengan label prediksi. Terakhir, sebuah gambar khusus ("04-pizza-dad.jpeg") diunduh jika belum ada, dan prediksi dilakukan menggunakan model pada gambar tersebut, dengan hasilnya juga ditampilkan bersama dengan label prediksi.