Muhammad Nur Maajid_1103228145

Modul ini memperkenalkan Milestone Project 2, yang fokus pada replikasi sebuah makalah penelitian dalam bidang pembelajaran mesin menggunakan PyTorch. Proyek ini akan menciptakan Vision Transformer (ViT) dari awal dan mengaplikasikannya pada permasalahan FoodVision Mini.
Tujuan Paper Replicating: Replikasi makalah penelitian merupakan upaya untuk mengimplementasikan teknik-teknik terkini dalam pembelajaran mesin ke dalam kode sehingga dapat digunakan untuk permasalahan khusus.

Struktur Makalah Penelitian: Sebuah makalah penelitian pembelajaran mesin umumnya terdiri dari Abstract, Introduction, Method, Results, Conclusion, References, dan Appendix. Replikasi makalah melibatkan pengubahan gambar, diagram, rumus matematika, dan teks menjadi kode yang dapat digunakan, khususnya menggunakan PyTorch.

Alasan Replikasi Makalah: Makalah penelitian pembelajaran mesin mencakup hasil penelitian tim yang memerlukan berbulan-bulan hingga bertahun-tahun kerja. Replikasi makalah tersebut dapat memberikan wawasan yang berharga dan menjadi latihan yang bagus untuk meningkatkan keterampilan pembelajaran mesin.

George Hotz Quote: George Hotz, pendiri comma.ai, mendorong untuk meningkatkan keterampilan sebagai insinyur pembelajaran mesin dengan mendownload makalah, mengimplementasikannya, dan terus melakukannya untuk memperoleh keterampilan.

Tantangan Replikasi: Meskipun menantang, banyak alat dan perpustakaan pembelajaran mesin seperti HuggingFace, PyTorch Image Models (timm library), dan fast.ai lahir untuk membuat penelitian pembelajaran mesin lebih dapat diakses.

Sumber Referensi Makalah: Ada beberapa tempat untuk menemukan contoh kode untuk makalah penelitian pembelajaran mesin, termasuk arXiv, AK Twitter, Papers with Code, dan GitHub repository seperti vit-pytorch dari lucidrains.

Langkah 0: Persiapan
Memastikan modul yang diperlukan telah diunduh.

Mengimpor skrip Python seperti data_setup.py dan engine.py dari direktori going_modular yang diunduh dari repositori pytorch-deep-learning.

Memastikan instalasi torchinfo dan torchvision versi 0.13+, serta versi torch 1.12+.

Jika versi tidak sesuai, mengunduh dan menginstal versi yang diperlukan.

Melanjutkan dengan impor modul, mengatur kode yang agnostik perangkat, dan mendapatkan skrip helper_functions.py dari GitHub.

1. Mendapatkan Data
Mengunduh dataset gambar pizza, steak, dan sushi menggunakan fungsi download_data() dari helper_functions.py.

Menyiapkan jalur direktori untuk gambar pelatihan dan pengujian.

2. Membuat Dataset dan DataLoader
Membuat transformasi untuk mempersiapkan gambar, dengan mengatur ukuran gambar sesuai dengan yang disebutkan dalam ViT paper.

Membuat DataLoader untuk pelatihan dan pengujian menggunakan fungsi create_dataloaders() dari data_setup.py.

2.1 Persiapkan Transformasi Gambar
Buat ukuran gambar (IMG_SIZE) sesuai dengan yang tercantum pada Tabel 3 di paper Vision Transformer (ViT).

Buat pipeline transformasi manual yang melibatkan Resize dan ToTensor untuk persiapan data.

2.2 Ubah Gambar menjadi DataLoader
Buat DataLoader menggunakan fungsi create_dataloaders() dari data_setup.py.

Tetapkan BATCH_SIZE sesuai kebutuhan, dan gunakan transformasi manual yang telah dibuat sebelumnya.

Pin memory diaktifkan (pin_memory=True) untuk mempercepat komputasi.

2.3 Visualisasi Satu Gambar
Dengan menggunakan DataLoader, dapatkan batch gambar dan label dari pelatihan.

Pilih satu gambar dan label dari batch.

Visualisasikan gambar dan label menggunakan matplotlib.

3. Visualisasi Satu Gambar

1. Mendapatkan satu batch gambar dan label dari DataLoader pelatihan.
2. Memilih satu gambar dan label dari batch tersebut.
3. Menampilkan bentuk batch dan label.
4. Menampilkan gambar dan label menggunakan matplotlib.

3.1 Input dan Output, Layer, dan Blok

ViT adalah arsitektur jaringan saraf mendalam yang terdiri dari beberapa lapisan dan blok.

Lapisan adalah bagian dari arsitektur yang mengambil input, melakukan operasi, dan menghasilkan output.

Blok adalah kumpulan lapisan, yang juga mengambil input dan menghasilkan output.

3.2 Penguraian Arsitektur ViT

Arsitektur ViT terdiri dari beberapa tahapan: a. Patch + Position Embedding (Input): Mengubah gambar input menjadi urutan patch gambar dan menambahkan nomor posisi. b. Linear projection of flattened patches (Embedded Patches): Merubah patch gambar menjadi embedding menggunakan proyeksi linear. c. Norm: Normalisasi lapisan (LayerNorm) digunakan untuk mengurangi overfitting. d. Multi-Head Attention (MSA): Lapisan self-attention yang menerima input dari LayerNorm. e. MLP (Multilayer Perceptron): Blok MLP yang terdiri dari dua lapisan Linear dengan fungsi aktivasi GELU di antaranya. f. Transformer Encoder: Kumpulan lapisan yang terdiri dari MSA dan MLP, dengan koneksi skip antar-lapisan. g. MLP Head: Lapisan keluaran arsitektur, mengonversi fitur yang dipelajari menjadi kelas keluaran.

3.2.1 Eksplorasi Gambar 1

Gambar 1 dari paper ViT memberikan gambaran tentang model secara grafis.

Fokus pada lapisan, input, output, blok, dan koneksi antar-lapisan.

3.2.2 Eksplorasi Empat Persamaan

Empat persamaan di bagian 3.1 menyajikan matematika di balik empat komponen utama arsitektur ViT.

Terkait dengan koneksi dan output dari lapisan dan blok pada Gambar 1.

3.2.3 Eksplorasi Persamaan 1

Persamaan 1 membahas token kelas, embedding patch, dan embedding posisi input gambar.

3.2.4 Eksplorasi Persamaan 2
Persamaan 2 menjelaskan bahwa setiap lapisan terdiri dari blok Multi-Head Attention (MSA) dan blok LayerNorm.

3.2.5 Eksplorasi Persamaan 3
Persamaan 3 menyatakan bahwa setiap lapisan juga memiliki blok Multilayer Perceptron (MLP) dengan LayerNorm.

3.2.6 Eksplorasi Persamaan 4
Persamaan 4 menggambarkan bahwa output akhir (y) dari arsitektur adalah hasil LayerNorm dari output lapisan terakhir (L).

3.2.7 Eksplorasi Tabel 1
Tabel 1 menyajikan detail hyperparameter untuk model ViT berbagai ukuran (ViT-Base, ViT-Large, ViT-Huge).

Hyperparameter mencakup jumlah lapisan, dimensi embedding, ukuran MLP, jumlah kepala, dan jumlah parameter.

4. Equation 1: Split data into patches and creating the class, position, and patch embedding
Mengimplementasikan Equation 1 untuk membagi data menjadi potongan dan membuat embedding kelas, posisi, dan potongan. Proses ini dimulai dengan membuat embedding untuk potongan gambar (patch embedding) pada arsitektur Vision Transformer (ViT)
4.1 Menghitung Dimensi Input dan Output Patch Embedding Secara Manual:
Menciptakan nilai-nilai contoh untuk parameter seperti tinggi (height), lebar (width), jumlah saluran warna (color_channels), dan ukuran patch (patch_size).

Menghitung jumlah patch (number_of_patches) berdasarkan parameter-parameter tersebut.

4.2 Mengonversi Image Menjadi Patch:
Memvisualisasikan pengubahan satu gambar menjadi serangkaian patch.

Melihat bagaimana top row dari piksel-piksel yang di-patch dapat divisualisasikan.

4.3 Membuat Patch Embedding dengan torch.nn.Conv2d():
Menunjukkan cara menggunakan layer **torch.nn.Conv2d()** untuk membuat patch dari gambar.

Menciptakan convolutional layer dengan kernel_size dan stride yang sesuai dengan ukuran patch.

Menyaring gambar melalui convolutional layer untuk mendapatkan serangkaian feature map.

4.4 Meratakan Patch Embedding dengan torch.nn.Flatten():
Menggunakan **torch.nn.Flatten()** untuk meratakan feature map menjadi urutan satu dimensi.

Menyesuaikan dimensi tensor hasil untuk sesuai dengan bentuk keluaran yang diinginkan.

4.5 Mengubah Layer Penyematan Peta ViT menjadi Modul PyTorch
Bagian ini membahas pembuatan layer penyematan peta untuk ViT dalam bentuk modul PyTorch. Melibatkan subclassing dari **nn.Module** dan mencakup langkah-langkah berikut:

1. Membuat kelas **PatchEmbedding** yang merupakan subclass dari **nn.Module**.

2. Menginisialisasi kelas dengan parameter **in_channels=3**, **patch_size=16** (untuk ViT-Base), dan **embedding_dim=768**.

3. Membuat layer untuk mengubah gambar menjadi potongan menggunakan **nn.Conv2d**.

4. Membuat layer untuk meratakan peta fitur potongan menjadi satu dimensi menggunakan **nn.Flatten**.

5. Mendefinisikan metode **forward** untuk melewati input melalui layer yang dibuat pada langkah 3 dan 4.

6. Memastikan bentuk output sesuai dengan yang dibutuhkan oleh arsitektur ViT.

4.6 Membuat Penyemat Kelas Token
Pembahasan tentang pembuatan penyemat kelas token (class token) yang diperlukan dalam arsitektur ViT. Ini melibatkan pendekatan mirip dengan BERT, dengan menambahkan penyemat kelas pada awal urutan potongan gambar yang disematkan.

4.7 Membuat Penyemat Posisi
Membahas penciptaan penyemat posisi (position embedding) untuk menyimpan informasi posisi dalam urutan potongan gambar yang disematkan. Ini mencakup penggunaan penyemat posisi 1D yang dapat dipelajari.

4.8 Menggabungkan Semua Komponen: Dari Gambar ke Penyematan
Langkah-langkah akhir dalam menciptakan penyematan gambar dari input hingga output, sesuai dengan persamaan 1 dari bagian 3.1 dalam paper ViT. Proses ini melibatkan:

1. Menentukan ukuran potongan.

2. Mendapatkan gambar tunggal, mencetak bentuknya, dan menyimpan tinggi dan lebar.

3. Menambah dimensi batch pada gambar tunggal.

1. Membuat layer penyemat potongan menggunakan **PatchEmbedding**.

2. Melewati gambar melalui layer penyemat potongan.

3. Membuat penyemat kelas token.

4. Menyisipkan penyemat kelas token pada potongan gambar yang telah dibuat.

5. Membuat penyemat posisi.

6. Menambahkan penyemat posisi pada penyemat kelas token dan potongan gambar.

5. Membangun Transformer Encoder

Bagian ini fokus pada pembangunan Transformer Encoder, bagian kunci dalam arsitektur ViT. Transformer Encoder digunakan untuk mengolah penyematan peta gambar yang dihasilkan sebelumnya.

5.1 Membuat Layer Normalisasi dan Multi-Head Attention

Pembahasan dimulai dengan membuat layer normalisasi dan lapisan Multi-Head Attention.

1. Membuat kelas **MultiHeadSelfAttention** yang merupakan subclass dari **nn.Module**.

2. Menginisialisasi kelas dengan parameter **embedding_dim=768** dan **num_heads=12** (sesuai dengan ViT-Base).

3. Membuat lapisan Q, K, dan V untuk setiap kepala perhatian menggunakan **nn.Linear**.

4. Menggabungkan hasil Q, K, dan V untuk setiap kepala perhatian menggunakan fungsi **view** dan **transpose**.

5. Menerapkan perhatian self attention menggunakan matriks perhatian dan fungsi softmax.

6. Menggabungkan hasil self attention dari setiap kepala menggunakan lapisan linear.

5.2 Menerapkan Layer Normalisasi

Melibatkan implementasi layer normalisasi untuk mengatasi masalah munculnya nilai yang sangat besar atau sangat kecil selama pelatihan.

1. Membuat kelas **MLPHead** sebagai subclass dari **nn.Module**.

2. Menginisialisasi kelas dengan parameter **embedding_dim=768** dan **mlp_dim=3072** (sesuai dengan ViT-Base).

3. Membuat dua lapisan linear untuk ekspansi dan pemampatan.

4. Menambahkan fungsi aktivasi GELU dan dropout di antara kedua lapisan.

5.3 Menerapkan Multi-Head Attention

Menerapkan lapisan Multi-Head Attention untuk memproses penyematan peta gambar. Ini melibatkan pemisahan penyematan peta menjadi beberapa kepala untuk memahami hubungan antara potongan gambar.

1. Membuat kelas **TransformerEncoderBlock** sebagai subclass dari **nn.Module**.
2. Menginisialisasi kelas dengan parameter **embedding_dim=768**, **num_heads=12**, dan **mlp_dim=3072**.
3. Menerapkan lapisan normalisasi dan dropout pada input.
4. Meneruskan input melalui lapisan Multi-Head Self Attention.
5. Menambahkan koneksi residu dan normalisasi setelah lapisan pertama.
6. Meneruskan output melalui lapisan MLP Head.
7. Menambahkan koneksi residu dan normalisasi setelah lapisan kedua.
8. Menghasilkan output blok Transformer Encoder.

6. Membangun MLP Head

Bagian ini membahas pembuatan MLP Head, yang bertanggung jawab untuk mengubah keluaran dari Transformer Encoder menjadi representasi yang dapat digunakan untuk klasifikasi.

6.1 Menerapkan MLP Head

Implementasi MLP Head untuk mengubah output dari Transformer Encoder menjadi representasi yang sesuai untuk klasifikasi.

1. Membuat kelas **VisionTransformer** sebagai subclass dari **nn.Module**.
2. Menginisialisasi kelas dengan parameter arsitektur ViT, seperti **num_classes=1000** dan **num_layers=12** (untuk ViT-Base).
3. Membuat layer penyemat gambar, blok Transformer Encoder, dan lapisan klasifikasi akhir.
4. Meneruskan input gambar melalui layer penyemat, blok Transformer Encoder sebanyak **num_layers**, dan lapisan klasifikasi.

6.2 Menyusun Seluruh Arsitektur Model

Penggabungan semua komponen yang telah dibuat sebelumnya, termasuk penyematan gambar, Transformer Encoder, dan MLP Head, untuk membentuk arsitektur model ViT secara keseluruhan.

1. Menginisialisasi bobot model dengan inisialisasi He.

1. Menerapkan metode forward yang meneruskan input melalui layer penyemat, blok Transformer Encoder, dan lapisan klasifikasi.

7. Melatih dan Mengevaluasi Model

Bagian ini mencakup langkah-langkah untuk melatih dan mengevaluasi model Vision Transformer setelah keseluruhan arsitektur selesai dibangun.

7.1 Menyiapkan Data Pelatihan dan Pengujian

Langkah-langkah untuk menyiapkan data pelatihan dan pengujian yang akan digunakan untuk melatih dan mengevaluasi model.

1. Mendefinisikan transformasi data untuk augmentasi.

2. Menggunakan dataset ImageNet untuk pelatihan.

7.2 Melatih dan Mengevaluasi Model

Proses melatih model Vision Transformer menggunakan data pelatihan yang telah disiapkan sebelumnya, diikuti dengan evaluasi performa model pada data pengujian.

1. Menginisialisasi model, optimizer, dan fungsi kerugian.

2. Menerapkan loop pelatihan dengan perulangan epoch.

3. Melakukan pelatihan pada setiap batch gambar.

4. Menghitung loss dan melakukan backpropagation.

5. Evaluasi model pada set pengujian.

8. Menyatukan Semuanya untuk Membuat ViT

Setelah melalui banyak langkah, saatnya menyatukan semua komponen menjadi arsitektur Vision Transformer (ViT) yang lengkap. Proses ini melibatkan langkah-langkah berikut:

8.1 Persiapan ViT untuk Pelatihan

Menggabungkan semua blok yang telah dibuat ke dalam kelas **ViT** yang merupakan subclass dari **nn.Module**. Proses ini mencakup:

Inisialisasi kelas dengan hyperparameter dari Tabel 1 dan Tabel 3.

Membuat embedding token dan position embedding.

Menyusun blok Transformer Encoder dalam urutan yang tepat.

Membuat lapisan klasifikasi akhir (MLP Head).

Menerapkan metode forward untuk meneruskan input melalui seluruh arsitektur.

8.2 Inisialisasi dan Visualisasi Model ViT

Melakukan inisialisasi bobot model dengan inisialisasi He dan menyajikan visualisasi model ViT menggunakan **torchinfo.summary()** untuk mendapatkan gambaran input dan output dari setiap lapisan.

9. Persiapan Kode Pelatihan untuk Model ViT

9.1 Pembuatan Optimizer

Membuat optimizer untuk mengoptimalkan parameter-model ViT menggunakan Adam optimizer. Menetapkan nilai hyperparameter seperti laju belajar, nilai beta, dan weight decay sesuai dengan yang dijelaskan dalam bagian 9.1.

9.2 Pembuatan Fungsi Kerugian

Menggunakan fungsi kerugian **torch.nn.CrossEntropyLoss()** untuk tugas klasifikasi multi-kelas.

9.3 Pelatihan Model ViT

Menyusun dan melatih model ViT dengan menggunakan fungsi pelatihan **train()** dari skrip **engine.py**. Memasukkan optimizer, fungsi kerugian, dan dataloader pelatihan dan pengujian ke dalam fungsi pelatihan.

9.4 Yang Hilang dalam Pengaturan Pelatihan

Menyadari bahwa hasil pelatihan mungkin tidak optimal karena perbedaan skala dengan paper ViT asli. Berbagai faktor seperti jumlah data, jumlah epoch, dan batch size yang lebih kecil dapat mempengaruhi hasil pelatihan.

9.5 Plot Kurva Kerugian Model ViT

Menggunakan fungsi **plot_loss_curves** dari **helper_functions.py** untuk memvisualisasikan kurva kerugian model ViT selama pelatihan.

10. Menggunakan model ViT yang sudah di-pretrain dari torchvision.models pada dataset yang sama:

Pembahasan tentang manfaat menggunakan model yang sudah di-pretrain di bagian sebelumnya.

Pada dasarnya, penggunaan transfer learning (menggunakan model yang sudah di-pretrain) sangat bermanfaat, terutama ketika pelatihan model sendiri dari awal tidak memberikan hasil optimal.

10.1 Mengapa menggunakan model yang sudah di-pretrain:
Banyak penelitian machine learning modern menggunakan dataset besar dan sumber daya komputasi yang besar.

Original ViT yang sudah terlatih mungkin tidak dianggap sebagai setup pelatihan "super besar" di era machine learning modern.

Penggunaan model ViT-L/16 yang sudah di-pretrain pada dataset ImageNet-21k umumnya memberikan hasil baik dengan menggunakan sumber daya yang lebih sedikit.

10.2 Mendapatkan model ViT yang sudah di-pretrain dan membuat feature extractor:
Mendapatkan model ViT yang sudah di-pretrain dari torchvision.models.

Menyiapkan kode yang agnostik perangkat.

Membuat model ViT-Base dengan patch size 16 sebagai feature extractor untuk FoodVision Mini.

10.3 Menyiapkan data untuk model ViT yang sudah di-pretrain:
Mengunduh dan membuat DataLoaders untuk model ViT kita sendiri sebelumnya.

Mengunduh gambar pizza, steak, dan sushi untuk Food Vision Mini.

Transformasi gambar menjadi tensors dan DataLoaders.

10.4 Melatih model feature extractor ViT:
Menggunakan optimizer Adam dengan learning rate 1e-3 dan loss function CrossEntropyLoss.

Menggunakan engine.train() untuk melatih model.

10.5 Plot kurva loss model feature extractor ViT:
Plot kurva loss model feature extractor ViT.

10.6 Menyimpan model feature extractor ViT dan memeriksa ukuran file:
Menyimpan model menggunakan utils.save_model().

Memeriksa ukuran file model ViT feature extractor.

11. Melakukan prediksi pada gambar kustom:
Mengunduh gambar pizza dad dan menggunakan model ViT feature extractor untuk melakukan prediksi.