Nama : Muhammad Daffa Saifullah

NIM : 1103204024

UAS Machine Learning

Technical Report

00_Pytorch Fundamental

---

1. **Apa itu PyTorch?**
    - PyTorch adalah perpustakaan pembelajaran mesin sumber terbuka yang dikembangkan oleh Facebook. Ini menyediakan grafik komputasi yang fleksibel dan dinamis, menjadikannya sangat cocok untuk penelitian pembelajaran mendalam dan jaringan saraf.
2. **Apa yang bisa digunakan PyTorch?**
    - PyTorch dapat digunakan untuk berbagai tugas dalam pembelajaran mesin, termasuk namun tidak terbatas pada:
        - Pelatihan dan penyebaran jaringan saraf
        - Tugas penglihatan komputer (analisis gambar dan video)
        - Pemrosesan bahasa alami
        - Pembelajaran penguatan
        - Pemindahan pembelajaran
        - Perhitungan ilmiah
3. **Siapa yang menggunakan PyTorch?**
    - PyTorch digunakan secara luas baik di dunia akademis maupun industri. Peneliti, insinyur, dan ilmuwan data menggunakan PyTorch untuk mengembangkan dan mengimplementasikan model pembelajaran mesin. Ini populer di lembaga penelitian dan di kalangan perusahaan yang terlibat dalam kecerdasan buatan dan pembelajaran mendalam.

4. Mengapa menggunakan PyTorch?

    - PyTorch dipilih karena grafik komputasi dinamisnya, kemudahan penggunaannya, dan dukungan komunitas yang kuat. Ini memungkinkan pengembangan model intuitif dan debugging, menjadikannya pilihan utama bagi banyak peneliti dan praktisi. Selain itu, PyTorch memiliki ekosistem yang aktif dan mendukung jaringan saraf dinamis, yang dapat diubah saat berjalan selama runtime.

5. Di mana Anda bisa mendapatkan bantuan?

    - Pengguna dapat mencari bantuan dari berbagai sumber, termasuk dokumentasi resmi PyTorch, forum seperti Stack Overflow, dan forum komunitas terkait pembelajaran mendalam. PyTorch memiliki komunitas yang aktif, dan banyak tutorial tersedia secara online.

6. Introduction to Tensors (Pengantar tentang Tensors):
   Tensor adalah struktur data matematika yang umum digunakan dalam komputasi numerik dan machine learning. Tensors dapat dianggap sebagai generalisasi dari skalar, vektor, dan matriks ke dimensi yang lebih tinggi.

7. Creating Tensors (Membuat Tensors):
   Dalam konteks pemrograman tensor, Anda dapat membuat tensor dengan menggunakan pustaka seperti TensorFlow atau PyTorch. Penggunaan umum adalah membuat tensor dengan nilai tertentu atau menginisialisasi tensor dengan nilai default.

8. Random Tensors (Tensors Acak):
   Tensors dapat diinisialisasi dengan nilai acak menggunakan fungsi-fungsi seperti tf.random.normal di TensorFlow atau torch.randn di PyTorch. Inisialisasi acak berguna dalam beberapa situasi, seperti inisialisasi bobot dalam model machine learning.

9. Zeros and Ones (Tensor Berisi Nol dan Satu):

Tensors dapat dibuat dengan mengisi semua elemennya dengan nol atau satu menggunakan fungsi seperti tf.zeros atau torch.ones. Inisialisasi dengan nilai konstan seperti nol atau satu sering digunakan pada awal pembuatan model.

10. Creating a Range and Tensors Like (Membuat Range dan Tensors Mirip):
    Fungsi-fungsi seperti tf.range atau torch.arange memungkinkan pembuatan tensor yang berisi urutan nilai dengan interval tertentu. Tensors mirip dapat dibuat dengan memanipulasi atau mengubah dimensi dari tensor yang sudah ada.

11. Tensor Datatypes (Tipe Data Tensors):
    Tensors dapat memiliki berbagai tipe data, seperti float32, float64, int32, int64, dll. Pemilihan tipe data tensor bergantung pada kebutuhan komputasi dan presisi yang diperlukan.

12. Manipulating Tensors (Operasi Tensor):
    Operasi tensor mencakup berbagai manipulasi yang dapat dilakukan pada tensor untuk memproses data. Ini adalah serangkaian fungsi dan operasi matematika yang memungkinkan Anda mengubah, menggabungkan, atau menghitung tensor. Beberapa operasi dasar termasuk penjumlahan, pengurangan, perkalian, dan fungsi-fungsi lainnya.

13. Basic Operations (Operasi Dasar):

- **Penjumlahan dan Pengurangan**: Tensors dapat ditambahkan atau dikurangkan elemennya secara langsung, asalkan dimensinya sesuai.

```
import tensorflow as tf

a = tf.constant([1, 2, 3])
b = tf.constant([4, 5, 6])

# Penjumlahan
c = tf.add(a, b)
```

Output: [5, 7, 9]

# Pengurangan
d = tf.subtract(a, b)
# Output: [-3, -3, -3]

- **Perkalian dan Pembagian**: Tensors dapat dikalikan atau dibagi dengan skalar atau tensor lainnya.
  import tensorflow as tf a = tf.constant([1, 2, 3])

  # Perkalian dengan skalar b = tf.multiply(a, 2)
  # Output: [2, 4, 6]

  # Pembagian dengan skalar c = tf.divide(a, 2)
  # Output: [0.5, 1.0, 1.5]
  Matrix Multiplication (Perkalian Matriks):
- Perkalian matriks adalah operasi yang sangat umum dalam komputasi tensor dan machine learning. Dalam TensorFlow atau PyTorch, Anda dapat menggunakan fungsi seperti **tf.matmul** (TensorFlow) atau **torch.mm** (PyTorch) untuk melakukan perkalian matriks.
  import tensorflow as tf

  # Matriks A berukuran (m x n)
  A = tf.constant([[1, 2], [3, 4]])

  # Matriks B berukuran (n x p)
  B = tf.constant([[5, 6], [7, 8]])

  # Perkalian matriks
  C = tf.matmul(A, B)

14. Finding the min, max, mean, sum, etc (aggregation):

Operasi agregasi seperti mencari nilai minimum, maksimum, rata-rata, atau jumlah dari elemen-elemen tensor sangat umum dalam analisis data dan machine learning. Dalam TensorFlow atau PyTorch, Anda dapat menggunakan fungsi seperti tf.reduce_min, tf.reduce_max, tf.reduce_mean, dan tf.reduce_sum (TensorFlow) atau fungsi yang setara di PyTorch untuk melakukan operasi-agregasi ini.

15. Positional min/max:

    Dalam beberapa kasus, selain menemukan nilai minimum atau maksimum dalam tensor, Anda mungkin juga perlu menemukan posisi (indeks) dari nilai tersebut. Di TensorFlow atau PyTorch, Anda dapat menggunakan fungsi seperti tf.argmin dan tf.argmax (TensorFlow) atau fungsi setara di PyTorch untuk mencapai ini.

16. Reshaping, Stacking, Squeezing, and Unsqueezing:

    - Reshaping (Mengubah Bentuk): Mengubah bentuk tensor dapat dilakukan menggunakan metode seperti tf.reshape di TensorFlow atau torch.view di PyTorch.

      ```
      import tensorflow as tf
      # Membuat tensor
      tensor = tf.constant([[1, 2, 3], [4, 5, 6]])

      # Mengubah bentuk tensor
      reshaped_tensor = tf.reshape(tensor, shape=(3, 2))
      ```

    - Stacking: Menggabungkan beberapa tensor menjadi satu tensor dengan dimensi tambahan. Misalnya, tf.stack di TensorFlow atau torch.stack di PyTorch.

      ```
      import tensorflow as tf
      # Membuat dua tensor
      tensor_a = tf.constant([1, 2, 3])
      ```

```
tensor_b = tf.constant([4, 5, 6])

# Menggabungkan tensor menjadi satu tensor
stacked_tensor = tf.stack([tensor_a, tensor_b])
```

- Squeezing dan Unsqueezing: Squeezing digunakan untuk menghapus dimensi yang memiliki ukuran 1, sedangkan unsqueezing menambah dimensi dengan ukuran 1.

```
import tensorflow as tf
# Membuat tensor dengan dimensi (3, 1)
tensor_a = tf.constant([[1], [2], [3]])

# Menghapus dimensi dengan ukuran 1
squeezed_tensor = tf.squeeze(tensor_a)

# Menambah dimensi dengan ukuran 1
unsqueezed_tensor = tf.expand_dims(squeezed_tensor, axis=0)
```

17. Running Tensors on GPUs (and Making Faster Computations):

- Getting a GPU:

  GPU (Graphics Processing Unit) adalah perangkat keras yang sangat efisien untuk melakukan operasi matematika paralel, dan karenanya, sering digunakan untuk melatih model deep learning yang memerlukan komputasi intensif.

  Jika Anda ingin menggunakan GPU untuk menjalankan model atau operasi tensor, Anda harus memiliki akses ke GPU. Ini bisa berupa GPU yang ada di komputer lokal Anda atau menggunakan layanan cloud yang menyediakan akses ke GPU, seperti Google Colab, AWS, atau Azure.

- Getting PyTorch to Run on the GPU:

PyTorch menyediakan dukungan bawaan untuk GPU. Pastikan Anda telah menginstal versi PyTorch yang mendukung GPU (biasanya versi yang menyertakan CUDA).
Jika PyTorch diinstal dengan dukungan CUDA, Anda dapat memeriksa apakah CUDA tersedia dengan menjalankan perintah berikut:

import torch print(torch.cuda.is_available())
Jika outputnya True, maka CUDA (dan GPU) tersedia.

- Putting Tensors (and Models) on the GPU:
  Untuk memindahkan tensor ke GPU, Anda dapat menggunakan metode to di PyTorch atau cuda di TensorFlow.

- Contoh memindahkan tensor ke GPU di PyTorch:
  import torch
  # Membuat tensor di CPU
  cpu_tensor = torch.tensor([1, 2, 3])

  # Memindahkan tensor ke GPU
  gpu_tensor = cpu_tensor.to('cuda')
- Selain itu, model PyTorch juga dapat dipindahkan ke GPU dengan cara yang sama.
  import torch import torch.nn as nn
  # Membuat model di CPU
  model = nn.Linear(3, 1)

  # Memindahkan model ke GPU
  model.to('cuda')
- Moving Tensors Back to the CPU:
  Setelah komputasi selesai di GPU, Anda mungkin perlu memindahkan hasilnya kembali ke CPU, terutama jika Anda ingin menggunakan hasilnya dalam operasi yang tidak didukung oleh GPU. Menggunakan

metode to di PyTorch atau cpu di TensorFlow untuk memindahkan tensor kembali ke CPU.

```
import torch
# Memindahkan tensor dari GPU ke CPU
gpu_tensor = gpu_tensor.to('cpu')
```

Demikian pula, model yang telah dipindahkan ke GPU juga dapat dipindahkan kembali ke CPU jika diperlukan.

01_Pytorch Workflow

---

1. **Data (Preparing and Loading)**:

- **Split Data into Training and Test Sets**:
    - Persiapan dan pembagian data adalah langkah awal dalam membangun model machine learning. Dataset umumnya dibagi menjadi dua bagian: training set untuk melatih model dan test set untuk mengevaluasi performa model.
    - PyTorch dapat bekerja dengan berbagai jenis dataset, termasuk struktur data kustom. Dalam prakteknya, dataset dibagi menggunakan modul **torch.utils.data.Dataset** dan **torch.utils.data.DataLoader** untuk kemudahan pelatihan.

```
from sklearn.model_selection import train_test_split

# Contoh: Pembagian data menjadi training dan test sets

all_data, all_labels = load_data()  # Fungsi ini mengembalikan data dan label

train_data, test_data, train_labels, test_labels = train_test_split(all_data, all_labels, test_size=0.2)
```

2. Build Model:

- PyTorch Model Building Essentials:
  Membangun model di PyTorch melibatkan pembuatan kelas yang mewarisi dari torch.nn.Module. Kelas ini biasanya memiliki dua metode utama: __init__ untuk mendefinisikan struktur model dan forward untuk mendefinisikan langkah-langkah komputasi yang dijalankan ketika model menerima input.

  ```
  import torch
  import torch.nn as nn

  class SimpleModel(nn.Module):
      def __init__(self, input_size, output_size):
          super(SimpleModel, self).__init__()
          self.linear = nn.Linear(input_size, output_size)

      def forward(self, x):
          return self.linear(x)

  # Membuat model
  input_size = 10
  output_size = 1
  model = SimpleModel(input_size, output_size)
  ```

- Checking the Contents of a PyTorch Model:
  Untuk memeriksa struktur dan parameter dari model PyTorch, Anda dapat mencetak model atau menggunakan metode print pada model. Ini membantu untuk memastikan bahwa model dibangun dengan benar dan sesuai dengan yang diinginkan.

- Making Predictions using torch.inference_mode():
  torch.inference_mode() tidak ada dalam PyTorch. Sebagai gantinya, kita menggunakan model.eval() untuk memastikan model dalam mode

evaluasi. Ini mengubah perilaku beberapa layer, seperti layer dropout, menjadi mode evaluasi, sehingga tidak ada dropout yang diaplikasikan saat membuat prediksi.

Jangan lupa untuk menggunakan torch.no_grad() ketika membuat prediksi untuk menghindari komputasi gradien yang tidak diperlukan

```
model.eval()  # Mengubah model ke mode evaluasi

with torch.no_grad():
    predictions = model(input_data)

model.train()  # Kembali ke mode pelatihan jika Anda melanjutkan pelatihan setelah membuat prediksi
```

3. Train Model:

- Creating a Loss Function and Optimizer in PyTorch:

  Loss Function:

  Loss function mengukur seberapa baik model kita melakukan prediksi terhadap target yang seharusnya. PyTorch menyediakan berbagai fungsi kerugian, tergantung pada jenis masalah yang dihadapi (misalnya, nn.CrossEntropyLoss untuk klasifikasi banyak kelas, atau nn.MSELoss untuk regresi).

  ```
  import torch.nn as nn

  criterion = nn.CrossEntropyLoss()
  # Contoh loss function untuk klasifikasi
  ```

- Optimizer:

  Optimizer bertanggung jawab untuk memperbarui parameter model agar loss function berkurang. torch.optim menyediakan berbagai optimizer seperti SGD, Adam, dll.

  ```
  import torch.optim as optim
  ```

```
optimizer = optim.SGD(model.parameters(), lr=0.01)  # Contoh optimizer SGD
```

4. Creating an Optimization Loop in PyTorch:

- PyTorch Training Loop:
  Loop pelatihan melibatkan iterasi melalui data pelatihan, melakukan prediksi dengan model, menghitung loss, melakukan backpropagation, dan memperbarui parameter model.

```
num_epochs = 10

for epoch in range(num_epochs):
    for inputs, labels in train_loader:
        optimizer.zero_grad()  # Mengatur gradien menjadi 0
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()  # Menghitung gradien
        optimizer.step()  # Memperbarui parameter
```

- PyTorch Testing Loop:
  Setelah melatih model, Anda ingin menguji performanya pada data yang tidak terlihat selama pelatihan. Loop pengujian melibatkan iterasi melalui data uji, melakukan prediksi, dan mengukur kinerja model

```
model.eval()  # Mode evaluasi

with torch.no_grad():
    total_correct = 0
    total_samples = 0

    for inputs, labels in test_loader:
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        total_correct += (predicted == labels).sum().item()
```

```
            total_samples += labels.size(0)

    accuracy = total_correct / total_samples
    print(f"Test Accuracy: {accuracy}")
```

Pastikan untuk mengubah model ke mode evaluasi dengan menggunakan model.eval() sebelum pengujian, dan kembali ke mode pelatihan dengan model.train() setelah pelatihan.

5. Saving and Loading a PyTorch Model:

- Saving a PyTorch Model's state_dict():
  state_dict() adalah metode di PyTorch yang memberikan representasi Python dari seluruh struktur dan parameter model. Ini sangat bermanfaat untuk menyimpan dan memuat model.

  ```
  # Simpan model ke file

  torch.save(model.state_dict(), 'model_weights.pth')
  ```

- Loading a Saved PyTorch Model's state_dict():

  ```
  # Membuat model dengan struktur yang sama

  loaded_model = SimpleModel(input_size, output_size)

  # Memuat state_dict ke model

  loaded_model.load_state_dict(torch.load('model_weights.pth'))
  ```

  Dengan menyimpan dan memuat state_dict(), kita dapat menyimpan dan memulihkan parameter model tanpa perlu menyimpan seluruh struktur model. Ini memudahkan ketika kita hanya ingin menyimpan parameter model untuk digunakan kembali nanti.

02_Pytorch Neural Network Classification

---

1. **Architecture of a Classification Neural Network**:

Arsitektur jaringan saraf untuk klasifikasi umumnya terdiri dari beberapa lapisan, termasuk:

- **Input Layer**: Menerima input yang sesuai dengan jumlah fitur dalam data.
- **Hidden Layers**: Lapisan-lapisan di antara input dan output layer. Mereka menggunakan fungsi aktivasi untuk mengekstraksi representasi fitur dari input.
- **Output Layer**: Memberikan output yang sesuai dengan jumlah kelas dalam tugas klasifikasi. Biasanya menggunakan fungsi aktivasi seperti softmax untuk output probabilitas kelas.

Sebagai contoh, dalam PyTorch, Anda dapat membuat model klasifikasi sederhana seperti berikut:

```
import torch.nn as nn

class SimpleClassifier(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleClassifier, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)
        self.softmax = nn.Softmax(dim=1)
    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.softmax(x)
        return x
```

2. Make Classification Data and Get it Ready:

2.1 Input and Output Shapes:

Dalam tugas klasifikasi, data input umumnya berupa matriks dengan setiap baris mewakili satu sampel dan setiap kolom mewakili fitur. Output biasanya berupa vektor probabilitas untuk setiap kelas atau label.

2.2 Turn Data into Tensors and Create Train and Test Splits:

Turn Data into Tensors:

Dalam PyTorch, data perlu diubah menjadi tensor agar dapat digunakan oleh model.

```
import torch
import numpy as np


# Contoh data
data = np.random.randn(100, 10)  # 100 sampel, 10 fitur


# Mengubah data menjadi tensor PyTorch
tensor_data = torch.tensor(data, dtype=torch.float32)
```

2.3 Create Train and Test Splits:

Pembagian data menjadi train set dan test set penting untuk evaluasi model.

```
from sklearn.model_selection import train_test_split
# Contoh label
labels = np.random.randint(0, 2, 100)  # 0 atau 1
# Membagi data menjadi train dan test sets
train_data, test_data, train_labels, test_labels = train_test_split(tensor_data, labels, test_size=0.2)
```

3. Building a Model:

3.1 Setup Loss Function and Optimizer:

- Setup Loss Function:
    - Fungsi kerugian digunakan untuk mengukur seberapa baik model memprediksi label yang benar. Pilihan fungsi kerugian bergantung pada jenis tugas yang dihadapi (misalnya, **nn.CrossEntropyLoss** untuk klasifikasi).

      import torch.nn as nn

      # Misalnya, untuk klasifikasi biner

      criterion = nn.BCELoss()

- Setup Optimizer:
    - Optimizer bertanggung jawab untuk memperbarui parameter model agar loss function berkurang. Beberapa optimizer yang umum digunakan termasuk SGD, Adam, dan RMSprop.

      import torch.optim as optim

      optimizer = optim.SGD(model.parameters(), lr=0.01)

3. Train Model:

3.2 Going from Raw Model Outputs to Predicted Labels:

- Raw Model Outputs (Logits):
    - Keluaran model sebelum melewati fungsi aktivasi. Dalam kasus klasifikasi, ini disebut logits.

      model = SimpleClassifier(input_size=10, hidden_size=5, output_size=2) raw_outputs = model(train_data)

- Prediction Probabilities:
    - Logits dapat diubah menjadi probabilitas kelas menggunakan fungsi aktivasi seperti softmax.

predictions_probs = nn.functional.softmax(raw_outputs, dim=1)

- **Prediction Labels**:
    - Untuk mendapatkan label prediksi, kita dapat menggunakan fungsi **torch.argmax** untuk memilih kelas dengan probabilitas tertinggi.

      predicted_labels = torch.argmax(predictions_probs, dim=1)

3.3 Building a Training and Testing Loop:

- **Training Loop**:
    - Loop pelatihan melibatkan iterasi melalui data pelatihan, menghitung loss, melakukan backpropagation, dan memperbarui parameter model.

      num_epochs = 10 for epoch in range(num_epochs): model.train()

      # Mode pelatihan optimizer.zero_grad()

      outputs = model(train_data) loss = criterion(outputs, train_labels) loss.backward() optimizer.step()

- **Testing Loop**:
    - Loop pengujian melibatkan iterasi melalui data uji dan mengukur performa model.

      model.eval()

      # Mode evaluasi with torch.no_grad(): test_outputs = model(test_data) test_loss = criterion(test_outputs, test_labels)

4. Improving a Model (From a Model Perspective):

4.1 Preparing Data to See if Our Model Can Model a Straight Line:

- Dalam beberapa kasus, kita mungkin perlu memodifikasi model untuk meningkatkan kinerjanya. Pertama-tama, kita perlu mempersiapkan data yang memungkinkan kita untuk menilai apakah model dapat memodelkan garis lurus atau fungsi yang sederhana.

  import torch import matplotlib.pyplot as plt

Persiapkan data garis lurus

X = torch.linspace(0, 1, 100).view(-1, 1) y = 2 * X + 1 + 0.1 * torch.randn(100, 1)

Garis lurus dengan sedikit noise

plt.scatter(X.numpy(), y.numpy()) plt.xlabel('X') plt.ylabel('y') plt.show()

4.2 Adjusting model_1 to Fit a Straight Line:

- Model yang mungkin tidak cocok untuk data yang ada dapat diperbarui untuk meningkatkan kinerjanya. Misalnya, jika model awalnya tidak dapat menangkap pola linear, kita dapat mengubah strukturnya.

  import torch.nn as nn class ImprovedModel(nn.Module): def __init__(self): super(ImprovedModel, self).__init__() self.linear = nn.Linear(1, 1) # Model linear sederhana def forward(self, x): return self.linear(x)

  Kemudian, kita dapat melatih model yang diperbarui pada data dan memeriksa apakah kinerjanya telah meningkat.

  improved_model = ImprovedModel()

  # Setup loss function and optimizer

  criterion = nn.MSELoss()

  optimizer = torch.optim.SGD(improved_model.parameters(), lr=0.01)

  # Training loop

  num_epochs = 100 for epoch in range(num_epochs):

  improved_model.train() optimizer.zero_grad() outputs = improved_model(X)

  loss = criterion(outputs, y)

  loss.backward() optimizer.step()

  # Plot hasil

  plt.scatter(X.numpy(), y.numpy(), label='Data')

```
plt.plot(X.numpy(), improved_model(X).detach().numpy(),
label='Improved Model', color='red')

plt.xlabel('X')

plt.ylabel('y')

 plt.legend()

 plt.show()
```

03. PyTorch Computer Vision

1. Di Mana Computer Vision Digunakan?

Computer vision adalah bidang studi yang memungkinkan mesin untuk menginterpretasi dan memahami informasi visual dari dunia. Ini memiliki aplikasi di berbagai domain, termasuk:

1. **Klasifikasi Gambar**: Mengidentifikasi objek atau scene dalam gambar.
2. **Deteksi Objek**: Menemukan dan mengklasifikasikan beberapa objek dalam sebuah gambar.
3. **Segmentasi Gambar**: Membagi gambar menjadi segmen dan memberikan label pada setiap segmen.
4. **Pengenalan Wajah**: Mengidentifikasi dan memverifikasi individu berdasarkan fitur wajah.
5. **Pengenalan Gerakan**: Mengenali gerakan yang dilakukan oleh manusia.

6. **Pengolahan Citra Medis**: Menganalisis citra medis untuk diagnosis dan perencanaan pengobatan.
7. **Kendaraan Otonom**: Memungkinkan kendaraan untuk mempersepsi dan memahami lingkungannya.
8. **Augmented Reality**: Meningkatkan tampilan dunia nyata dengan informasi yang dibuat oleh komputer.
9. **Kontrol Kualitas**: Memeriksa dan menilai kualitas produk dalam manufaktur.

2. Apa yang Akan Kami Bahas:

Pernyataan "Apa yang akan kami bahas" menunjukkan bahwa topik atau subtopik tertentu terkait dengan computer vision akan dijelaskan. Ini mungkin mencakup area seperti pra-pemrosesan gambar, arsitektur model, teknik pelatihan, dan aplikasi.

3. Perpustakaan Computer Vision di PyTorch:

Topik ini kemungkinan membahas tentang perpustakaan di PyTorch yang dirancang khusus untuk tugas-tugas computer vision. Beberapa perpustakaan computer vision yang terkemuka di PyTorch meliputi:

1. **TorchVision**: Perpustakaan computer vision yang disediakan oleh PyTorch, menawarkan dataset, model, dan utilitas untuk tugas-tugas computer vision umum.
2. **PyTorch Lightning**: Pembungkus PyTorch ringan yang menyederhanakan proses pelatihan untuk model deep learning, termasuk yang digunakan dalam computer vision.
3. **Fastai**: Perpustakaan tingkat tinggi yang dibangun di atas PyTorch, menyediakan abstraksi dan alat untuk menyederhanakan pengembangan model computer vision.
4. **Captum**: Perpustakaan interpretabilitas model untuk PyTorch, membantu memahami keputusan yang dibuat oleh model deep learning dalam computer vision.

4. Model 2: Building a Convolutional Neural Network (CNN):

Dalam topik ini, fokusnya adalah membangun Convolutional Neural Network (CNN), jenis arsitektur jaringan saraf yang efektif untuk tugas pengolahan gambar.

4.1 Stepping through nn.Conv2d():

- Langkah ini kemungkinan membahas operasi konvolusi dalam konteks modul **nn.Conv2d()** di PyTorch. Konvolusi adalah operasi penting dalam CNN untuk mengekstraksi fitur dari gambar.

```
import torch.nn as nn

# Contoh penggunaan nn.Conv2d()

conv_layer = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, stride=1, padding=1)
```

7.2 Stepping through nn.MaxPool2d():

- Max pooling adalah operasi untuk mereduksi dimensi spasial dari tensor. Modul **nn.MaxPool2d()** digunakan untuk melakukan operasi max pooling pada data.

```
# Contoh penggunaan nn.MaxPool2d()

pool_layer = nn.MaxPool2d(kernel_size=2, stride=2)
```

7.3 Setup a Loss Function and Optimizer for model_2:

- Untuk melatih model, Anda perlu menyiapkan fungsi kerugian (loss function) dan optimizer. Ini biasanya melibatkan pemilihan fungsi kerugian sesuai dengan tugas dan penggunaan optimizer seperti SGD atau Adam.

```
# Contoh pengaturan loss function dan optimizer untuk model_2 criterion_

model_2 = nn.CrossEntropyLoss()

optimizer_model_2 = optim.Adam(model_2.parameters(), lr=0.001)
```

7.4 Training and Testing model_2 using Our Training and Test Functions:

- Setelah persiapan, model_2 dilatih dan diuji menggunakan loop pelatihan dan pengujian yang mungkin telah dijelaskan sebelumnya.

  # Contoh pelatihan dan pengujian model_2 menggunakan fungsi pelatihan dan pengujian

  train_model(model_2, train_loader, criterion_model_2, optimizer_model_2, num_epochs=10)

  test_model(model_2, test_loader, criterion_model_2)

---

04. PyTorch Custom Datasets

---

1. What Should an Ideal Loss Curve Look Like?

Grafik kerugian ideal biasanya menurun seiring waktu. Pada awal pelatihan, kerugian dapat tinggi, tetapi seharusnya turun dengan berjalannya waktu seiring model belajar dari data pelatihan. Loss curve yang ideal menunjukkan konvergensi ke nilai minimum yang stabil.

1.1 How to Deal with Overfitting:

- Overfitting terjadi ketika model terlalu rumit dan dapat menangkap noise atau detail yang tidak relevan dari data pelatihan. Beberapa strategi untuk mengatasi overfitting melibatkan:
  - **Regularisasi**: Menambahkan istilah regularisasi ke fungsi kerugian untuk menghukum bobot yang terlalu besar.
  - **Dropout**: Menonaktifkan secara acak sebagian unit selama pelatihan untuk mencegah model bergantung terlalu kuat pada fitur tertentu.
  - **Data Augmentation**: Memperkaya dataset dengan membuat variasi kecil pada data pelatihan.

1.2 How to Deal with Underfitting:

- Underfitting terjadi ketika model terlalu sederhana dan tidak dapat menangkap kompleksitas dalam data. Beberapa cara untuk mengatasi underfitting melibatkan:
    - **Meningkatkan Kedalaman Model**: Menambahkan lapisan atau unit ke dalam model untuk meningkatkan kapasitasnya.
    - **Memperkaya Data**: Menambahkan lebih banyak data pelatihan atau menggunakan teknik augmentasi data.
    - **Mengurangi Regularisasi**: Memperlonggar aturan regularisasi untuk memungkinkan model mempelajari pola yang lebih rumit.

1.3 **The Balance Between Overfitting and Underfitting**:

- Menemukan keseimbangan antara overfitting dan underfitting seringkali melibatkan eksperimen. Beberapa cara untuk mencapai keseimbangan ini adalah:
    - **Validasi**: Menggunakan dataset validasi terpisah untuk mengevaluasi model selama pelatihan dan menghentikan pelatihan ketika kinerja di dataset validasi mulai menurun.
    - **Pemilihan Model yang Sesuai**: Pilih model yang memiliki kompleksitas sesuai dengan kompleksitas masalah yang dihadapi.
    - **Monitoring Metrik Kinerja**: Memantau metrik kinerja seperti akurasi atau kerugian pada set pelatihan dan validasi untuk menilai keseimbangan antara overfitting dan underfitting.

Memahami dan mengatasi overfitting serta underfitting adalah bagian penting dari pengembangan model yang dapat menggeneralisasi dengan baik pada data yang belum pernah dilihat sebelumnya.

2. **Apa yang Seharusnya Terjadi dalam Kurva Kerugian yang Ideal?**

- Kurva kerugian yang ideal umumnya menurun seiring waktu. Pada awal pelatihan, kerugian mungkin tinggi, tetapi seharusnya menurun seiring

dengan model mempelajari pola dari data pelatihan. Kurva kerugian yang ideal menunjukkan konvergensi ke nilai minimum yang stabil.

2.1 Cara Mengatasi Overfitting:

- Overfitting terjadi ketika model terlalu rumit dan dapat menangkap noise atau detail yang tidak relevan dari data pelatihan. Beberapa strategi untuk mengatasi overfitting melibatkan:
    - **Regularisasi**: Menambahkan istilah regularisasi ke fungsi kerugian untuk menghukum bobot yang terlalu besar.
    - **Dropout**: Menonaktifkan secara acak sebagian unit selama pelatihan untuk mencegah model bergantung terlalu kuat pada fitur tertentu.
    - **Data Augmentation**: Memperkaya dataset dengan membuat variasi kecil pada data pelatihan.

2.2 Cara Mengatasi Underfitting:

- Underfitting terjadi ketika model terlalu sederhana dan tidak dapat menangkap kompleksitas dalam data. Beberapa cara untuk mengatasi underfitting melibatkan:
    - **Meningkatkan Kedalaman Model**: Menambahkan lapisan atau unit ke dalam model untuk meningkatkan kapasitasnya.
    - **Memperkaya Data**: Menambahkan lebih banyak data pelatihan atau menggunakan teknik augmentasi data.
    - **Mengurangi Regularisasi**: Memperlonggar aturan regularisasi untuk memungkinkan model mempelajari pola yang lebih rumit.

2.3 Keseimbangan Antara Overfitting dan Underfitting:

- Menemukan keseimbangan antara overfitting dan underfitting seringkali melibatkan eksperimen. Beberapa cara untuk mencapai keseimbangan ini adalah:

- **Validasi**: Menggunakan dataset validasi terpisah untuk mengevaluasi model selama pelatihan dan menghentikan pelatihan ketika kinerja di dataset validasi mulai menurun.
- **Pemilihan Model yang Sesuai**: Pilih model yang memiliki kompleksitas sesuai dengan kompleksitas masalah yang dihadapi.
- **Monitoring Metrik Kinerja**: Memantau metrik kinerja seperti akurasi atau kerugian pada set pelatihan dan validasi untuk menilai keseimbangan antara overfitting dan underfitting.

3. Model 1: TinyVGG with Data Augmentation:

3.1 Create Transform with Data Augmentation:

- Transformasi dengan augmentasi data diperlukan untuk memperkaya dataset pelatihan dengan variasi dan membantu model untuk menggeneralisasi lebih baik. Contohnya dapat mencakup rotasi, flipping, dan pergeseran.

```
from torchvision import transforms

# Membuat transformasi dengan augmentasi data

data_transform = transforms.Compose([
transforms.RandomHorizontalFlip(),

transforms.RandomRotation(10),

transforms.RandomResizedCrop(64),

 transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) ])
```

3.2 Create Train and Test Datasets and DataLoaders:

- Selanjutnya, dataset dan DataLoader perlu dibuat untuk pelatihan dan pengujian dengan menggunakan transformasi yang telah dibuat.

```
from torchvision import datasets from torch.utils.data import DataLoader
```

```
# Membuat dataset pelatihan dan pengujian dengan DataLoader
train_dataset = datasets.CIFAR10(root='./data', train=True,
download=True, transform=data_transform)

test_dataset = datasets.CIFAR10(root='./data', train=False,
download=True, transform=data_transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

3.3 Construct and Train Model 1:

- Model TinyVGG perlu dibangun dan dilatih dengan menggunakan dataset yang telah diubah.

```
import torch.nn as nn import torch.optim as optim

# Membuat TinyVGG model

model_1 = TinyVGG()

# Setup fungsi kerugian dan optimizer

criterion_model_1 = nn.CrossEntropyLoss()

optimizer_model_1 = optim.SGD(model_1.parameters(),

 lr=0.001, momentum=0.9)

# Melatih model menggunakan data augmentasi

train_model(model_1, train_loader, criterion_model_1, optimizer_model_1,
num_epochs=10)
```

3.4 Plot the Loss Curves of Model 1:

- Setelah melatih model, kurva kerugian dapat diplot untuk mengevaluasi kinerja model pada set pelatihan.

```
import matplotlib.pyplot as plt

# Plot kurva kerugian

model_1 plot_loss_curve(model_1, train_loader, criterion_model_1)
```

Dalam hal ini, **plot_loss_curve** adalah fungsi yang dapat mencatat dan memplot kerugian selama pelatihan untuk evaluasi visual. Dengan melibatkan langkah-langkah ini, kita dapat membangun, melatih, dan mengevaluasi model TinyVGG dengan data augmentasi untuk meningkatkan generalisasi dan kinerjanya pada tugas pengenalan gambar CIFAR-10.

---

05. PyTorch Going Modular

---

1. Apa itu "Going Modular"?

- "Going modular" mengacu pada praktik struktur kode menjadi komponen modular atau modul. Sebuah modul adalah unit mandiri yang melakukan fungsi tertentu dan dapat digunakan kembali dalam bagian berbeda dari kode. Ini melibatkan memecah kode yang lebih besar menjadi bagian-bagian yang lebih kecil, dapat dikelola, dan independen, masing-masing melayani tujuan tertentu.

2.Mengapa Anda Ingin Menjadi Modular?

1. **Dapat Digunakan Kembali**: Kode modular mendukung dapat digunakannya kembali. Begitu Anda menulis dan menguji suatu modul, Anda dapat dengan mudah menggunakannya di bagian yang berbeda dari program atau bahkan dalam proyek lain.
2. **Kemudahan Perawatan**: Memecah kode menjadi komponen modular membuatnya lebih mudah untuk diperawat. Jika ada masalah atau kebutuhan untuk perbaikan, Anda dapat fokus pada modul tertentu tanpa memengaruhi seluruh kode.
3. **Skalabilitas**: Kode modular bersifat dapat diskalakan. Saat proyek Anda berkembang, Anda dapat menambahkan atau memodifikasi modul tanpa mengganggu seluruh sistem, memudahkan penyesuaian dengan persyaratan yang berubah.

4. **Kolaborasi**: Kode modular memfasilitasi kolaborasi di antara anggota tim. Anggota tim yang berbeda dapat bekerja pada modul yang berbeda secara bersamaan, dan integrasinya menjadi lebih mudah.
5. **Pengujian dan Debugging**: Pengujian dan debugging lebih mudah dikelola dalam kode modular. Anda dapat mengisolasi suatu modul dan menguji secara independen, memudahkan identifikasi dan perbaikan masalah.

3.Kelebihan dan Kekurangan Notebook vs Python Script:

Notebook:

Kelebihan:

1. **Interaktif**: Notebook memungkinkan pengalaman pengkodean yang interaktif dan eksploratif, yang bermanfaat untuk analisis data, visualisasi, dan pengembangan iteratif.
2. **Konten Kaya**: Notebook mendukung penambahan teks yang diformat, gambar, dan visualisasi bersamaan dengan kode, membuatnya cocok untuk membuat dokumen komprehensif dan dapat dibagikan.
3. **Visualisasi Mudah**: Visualisasi hasil intermediat dapat dilakukan dengan mudah dalam notebook, membantu pengguna memahami kode langkah demi langkah.

Kekurangan:

1. **Kendala Kontrol Versi**: Notebook dapat sulit untuk dikendalikan versinya, membuatnya sulit untuk melacak perubahan dari waktu ke waktu.
2. **Organisasi Kode Terbatas**: Meskipun Anda dapat struktur kode ke dalam sel, notebook tidak seefisien praktek rekayasa perangkat lunak tradisional dalam hal organisasi modular.

Python Script:

Kelebihan:

1. **Kendali Versi Lebih Baik**: Python script berfungsi baik dengan sistem kontrol versi, memungkinkan kolaborasi yang lebih mudah dan pelacakan perubahan dari waktu ke waktu.
2. **Modularitas**: Python script mendorong organisasi kode yang modular, memudahkan untuk mengelola proyek yang lebih besar dan mempromosikan penggunaan kembali kode.

Kekurangan:

1. **Kurang Interaktif**: Dibandingkan dengan notebook, menjalankan script dapat kurang interaktif, membuatnya lebih sulit untuk menjelajahi data dan bereksperimen secara interaktif.
2. **Tantangan Dokumentasi**: Meskipun docstring dan komentar dapat digunakan, mendokumentasikan kode dalam script mungkin kurang menarik secara visual dan mudah diakses dibandingkan dengan notebook.

---

06. PyTorch Transfer Learning

---

1. Apa itu Transfer Learning?

- Transfer Learning adalah pendekatan dalam pembelajaran mesin di mana model yang telah dilatih pada tugas tertentu digunakan sebagai titik awal untuk melatih model baru pada tugas terkait atau serupa. Ide utamanya adalah bahwa pengetahuan yang diperoleh oleh model dari satu tugas dapat di-transfer atau digunakan kembali untuk meningkatkan kinerja model pada tugas lain.

2.Mengapa Menggunakan Transfer Learning?

1. **Kurangnya Data**: Jika Anda memiliki dataset terbatas untuk tugas tertentu, transfer learning memungkinkan model untuk memanfaatkan pengetahuan yang telah diperoleh dari dataset yang lebih besar pada tugas serupa.

2. **Waktu dan Sumber Daya**: Melatih model dari awal pada dataset besar memerlukan waktu dan sumber daya komputasi yang besar. Transfer learning memungkinkan kita menggunakan model yang sudah ada dan melatihnya hanya pada bagian yang diperlukan untuk tugas kita.
3. **Kinerja yang Lebih Baik**: Model yang sudah dilatih pada tugas umum sering kali memiliki kemampuan ekstraksi fitur yang baik. Dengan transfer learning, kita dapat memanfaatkan kemampuan ini tanpa memerlukan latihan model dari awal.

3.**Dimana Menemukan Model yang Sudah Dilatih (Pretrained Models)**:

1. **TorchVision**: TorchVision, bagian dari ekosistem PyTorch, menyediakan sejumlah besar model yang sudah dilatih untuk tugas pengolahan gambar, seperti ResNet, VGG, dan lainnya.
2. **Hugging Face Model Hub**: Hugging Face menyediakan model-model yang sudah dilatih untuk tugas NLP (Natural Language Processing) dan tugas-tugas lainnya melalui Model Hub mereka.
3. **TensorFlow Hub**: Jika Anda menggunakan TensorFlow, TensorFlow Hub menyediakan koleksi model yang sudah dilatih untuk berbagai tugas.
4. **Kaggle Datasets**: Platform Kaggle menyediakan dataset dan kernel yang sering kali menggunakan model yang sudah dilatih untuk berbagai tugas. Model-model ini dapat diakses dan digunakan.

4.Create Datasets and DataLoaders

- Dalam konteks penggunaan torchvision.models atau model-model lainnya dalam PyTorch, langkah selanjutnya adalah mempersiapkan dataset dan DataLoader untuk pelatihan dan pengujian model. Ini melibatkan pembuatan transformasi data dan penggunaan kelas Dataset dan DataLoader.

4.1 Creating a Transform for torchvision.models (Manual Creation):

- Transformasi data digunakan untuk mempersiapkan data sebelum diberikan ke model. Jika kita menggunakan model dari torchvision.models, kita perlu menyesuaikan format masukan untuk sesuai dengan persyaratan model tersebut. Contoh transformasi manual:

```
from torchvision import transforms
# Transformasi manual untuk model dari torchvision.models
manual_transform = transforms.Compose([ transforms.Resize((224, 224)),  # Resize gambar ke ukuran yang diharapkan oleh model
transforms.ToTensor(),  # Konversi gambar menjadi tensor
transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalisasi nilai pixel
])
```

Transformasi ini secara manual menyesuaikan ukuran gambar, mengonversinya menjadi tensor, dan melakukan normalisasi. Hal ini sesuai dengan transformasi umum yang diperlukan oleh model-model torchvision.

4.2 Creating a Transform for torchvision.models (Auto Creation):

- Dalam beberapa kasus, kita dapat menggunakan fungsi pretrained=True untuk secara otomatis mendapatkan transformasi yang sesuai dengan model yang sudah dilatih:

```
import torchvision.transforms as T
# Transformasi otomatis untuk model torchvision.models dengan pretrained=True
auto_transform = T.Compose([
    T.Resize(256),
    T.CenterCrop(224),
    T.ToTensor(),
```

```
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Dengan mengatur pretrained=True, transformasi ini secara otomatis disesuaikan dengan persyaratan model dan dapat digunakan langsung dalam Dataset.

Setelah transformasi telah dibuat, langkah selanjutnya adalah membuat objek Dataset dan DataLoader menggunakan transformasi tersebut untuk memuat data pelatihan dan pengujian.

```
from torchvision import datasets
from torch.utils.data import DataLoader
# Membuat objek Dataset dengan transformasi manual
train_dataset_manual = datasets.CIFAR10(root='./data', train=True, download=True, transform=manual_transform)
test_dataset_manual = datasets.CIFAR10(root='./data', train=False, download=True, transform=manual_transform)
# Membuat DataLoader untuk dataset manual
train_loader_manual = DataLoader(train_dataset_manual, batch_size=64, shuffle=True)
test_loader_manual = DataLoader(test_dataset_manual, batch_size=64, shuffle=False)
# Membuat objek Dataset dengan transformasi otomatis
train_dataset_auto = datasets.CIFAR10(root='./data', train=True, download=True, transform=auto_transform)
test_dataset_auto = datasets.CIFAR10(root='./data', train=False, download=True, transform=auto_transform)
# Membuat DataLoader untuk dataset otomatis
train_loader_auto = DataLoader(train_dataset_auto, batch_size=64, shuffle=True)
```

test_loader_auto = DataLoader(test_dataset_auto, batch_size=64, shuffle=False)

Dengan membuat transformasi dan dataset seperti dijelaskan di atas, kita dapat menyiapkan data untuk pelatihan dan pengujian model dengan menggunakan model torchvision atau model lainnya dalam PyTorch.

5. Getting a Pretrained Model:

5.1 Which Pretrained Model Should You Use?

Pemilihan model yang sudah dilatih tergantung pada tugas yang akan dipecahkan. Misalnya, jika tugasnya adalah klasifikasi gambar, model-model populer seperti ResNet, VGG, atau MobileNet dapat digunakan. Jika tugasnya melibatkan pemrosesan bahasa alami, model-model seperti BERT atau GPT mungkin lebih sesuai.

5.2 Setting Up a Pretrained Model:

Menggunakan model yang sudah dilatih dari torchvision.models dalam PyTorch melibatkan mendownload model dan mengatur sesuai dengan kebutuhan kita. Contoh penggunaan model ResNet-18:

pythonCopy code

import torch import torchvision.models as models # Mendapatkan model ResNet-18 yang sudah dilatih pretrained_resnet18 = models.resnet18(pretrained=True)

5.3 Getting a Summary of Our Model with torchinfo.summary():

Untuk mendapatkan ringkasan model, kita dapat menggunakan **torchinfo** library. Instal library terlebih dahulu:

pip install torchinfo

Kemudian dapat digunakan sebagai berikut:

from torchinfo import summary

Mendapatkan ringkasan model

model_summary = summary(pretrained_resnet18, input_size=(batch_size, channels, height, width))

print(model_summary)

Ini memberikan informasi tentang arsitektur model, jumlah parameter, dan ukuran output setiap lapisan.

5.4 Freezing the Base Model and Changing the Output Layer to Suit Our Needs:

Setelah mendapatkan model yang sudah dilatih, seringkali kita ingin mempertahankan bobot yang sudah dipelajari pada lapisan-lapisan awal (basis model) dan hanya melatih beberapa lapisan akhir sesuai dengan tugas kita. Ini melibatkan pembekuan (freezing) lapisan-lapisan awal dan mengganti lapisan akhir sesuai kebutuhan.

```
# Membekukan lapisan-lapisan awal
for param in pretrained_resnet18.parameters():
    param.requires_grad = False
# Mengganti lapisan akhir (misalnya, untuk klasifikasi 10 kelas)
pretrained_resnet18.fc = torch.nn.Linear(pretrained_resnet18.fc.in_features, num_classes)
```

Dengan melakukan ini, kita dapat mengadaptasi model yang sudah dilatih untuk tugas spesifik kita dengan mempertahankan pengetahuan yang sudah ada pada lapisan-lapisan awal. Selanjutnya, kita dapat melatih model pada dataset kita sendiri.

---

07. PyTorch Experiment Tracking

---

1. Apa itu Experiment Tracking?

Experiment Tracking mengacu pada proses secara sistematis mencatat, memantau, dan mengelola berbagai aspek eksperimen machine learning. Ini melibatkan pencatatan parameter, konfigurasi, metrik, dan artefak yang dihasilkan selama proses eksperimen. Pelacakan eksperimen sangat penting untuk reproduktibilitas, kolaborasi, dan pemahaman kinerja berbagai konfigurasi model.

2.Mengapa Melacak Eksperimen?

- Reproduktibilitas: Melacak eksperimen memungkinkan para peneliti dan data scientist untuk mereproduksi pekerjaan mereka dengan dapat diandalkan. Dengan mencatat parameter dan konfigurasi, orang lain dapat membuat setup eksperimen yang sama dan memvalidasi hasil.
- Pembandingan: Ini memudahkan perbandingan antara berbagai konfigurasi model dan hyperparameter. Peneliti dapat menganalisis dampak perubahan secara real-time dan memilih model yang paling baik.
- Kolaborasi: Dalam lingkungan kolaboratif, pelacakan eksperimen memungkinkan anggota tim untuk berbagi dan memahami pekerjaan satu sama lain. Ini memberikan transparansi dan membantu menghindari upaya yang berlebihan.
- Pemecahan Masalah: Ketika eksperimen tidak menghasilkan hasil yang diharapkan, pelacakan memungkinkan identifikasi mudah terhadap konfigurasi dan parameter yang mungkin menyebabkan masalah. Ini membantu dalam proses debugging dan peningkatan kinerja model.
- Dokumentasi: Pelacakan eksperimen berfungsi sebagai dokumentasi untuk seluruh proses eksperimen. Ini mencakup perjalanan dari pengembangan model awal hingga konfigurasi akhir yang dioptimalkan.

3. Berbagai Cara Melacak Eksperimen Machine Learning:

- Logging Libraries: Perpustakaan logging seperti TensorBoard, MLflow, dan Neptune.ai menyediakan alat untuk mencatat dan memvisualisasikan metrik, parameter, dan artefak eksperimen. Ini memungkinkan pelacakan dan perbandingan secara real-time.
- Sistem Kontrol Versi: Memanfaatkan sistem kontrol versi seperti Git untuk melacak perubahan kode dan menyimpan berbagai versi kode dan checkpoint model. Ini memastikan reproduktibilitas kode dan memudahkan kolaborasi.

- Notebook dan Dokumentasi: Menjaga dokumentasi rinci dalam notebook Jupyter atau format lainnya. Ini dapat mencakup komentar, penjelasan, dan visualisasi dalam notebook untuk menjelaskan setup eksperimen dan hasil.
- Platform Pelacakan Eksperimen: Platform seperti DVC (Data Version Control), Comet.ml, dan WandB (Weights & Biases) menawarkan kemampuan pelacakan eksperimen yang komprehensif. Mereka memungkinkan pengguna untuk mencatat metrik, hyperparameter, versi kode, dan bahkan versi dataset.
- Logging Kustom: Mengimplementasikan logging kustom dalam kode untuk mencatat informasi penting selama eksperimen. Ini bisa melibatkan pencetakan ke konsol atau logging ke file.

4. Setting Up a Series of Modeling Experiments:

4.1 What Kind of Experiments Should You Run?

Dalam konteks machine learning, eksperimen pemodelan dapat melibatkan berbagai faktor, seperti:

- Variasi Hyperparameter: Eksperimen dengan berbagai nilai hyperparameter untuk menemukan konfigurasi yang optimal.
- Pemilihan Model: Eksperimen dengan berbagai arsitektur model untuk melihat mana yang paling sesuai dengan masalah yang dihadapi.
- Pemrosesan Data: Eksperimen dengan transformasi data, augmentasi, atau praproses yang berbeda untuk memahami pengaruhnya terhadap kinerja model.
- Dataset Variations: Eksperimen dengan dataset yang berbeda untuk melihat bagaimana variasi dataset memengaruhi hasil model.

4.2 What Experiments Are We Going to Run?

Misalnya, dalam konteks PyTorch, eksperimen pemodelan dapat mencakup:

- Eksperimen Hyperparameter: Mengubah tingkat learning, batch size, atau jumlah epoch untuk melihat bagaimana kinerja model berubah.

- Eksperimen Pemilihan Model: Mencoba beberapa arsitektur model, seperti ResNet, VGG, atau model-model kustom.
- Eksperimen Pemrosesan Data: Mencoba berbagai teknik augmentasi gambar atau praproses data.
- Eksperimen dengan Dataset Berbeda: Menggunakan dataset yang berbeda untuk melihat apakah model berperilaku secara konsisten di berbagai domain data.

4.3 Download Different Datasets:

```
# Menggunakan torchvision untuk mengunduh dataset CIFAR-10 dan MNIST sebagai contoh
from torchvision import datasets
# Unduh dataset CIFAR-10 dan MNIST
cifar10_dataset = datasets.CIFAR10(root='./data', train=True, download=True)
mnist_dataset = datasets.MNIST(root='./data', train=True, download=True)
```

4.4 Transform Datasets and Create DataLoaders:

```
from torchvision import transforms
from torch.utils.data import DataLoader
# Membuat transformasi untuk dataset CIFAR-10
cifar10_transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
# Membuat transformasi untuk dataset MNIST
mnist_transform = transforms.Compose([
```

```
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
# Membuat objek DataLoader untuk kedua dataset
cifar10_loader = DataLoader(dataset=cifar10_dataset, batch_size=64, shuffle=True)
mnist_loader = DataLoader(dataset=mnist_dataset, batch_size=64, shuffle=True)
```

4.5 Create Feature Extractor Models:

Membuat model ekstraktor fitur, yang mungkin merupakan model basis seperti ResNet atau model khusus untuk ekstraksi fitur tertentu.

```
import torch.nn as nn
import torchvision.models as models
# Membuat model ResNet sebagai contoh model ekstraktor fitur
class FeatureExtractorResNet(nn.Module):
    def __init__(self, pretrained=True):
        super(FeatureExtractorResNet, self).__init__()
        self.resnet = models.resnet18(pretrained=pretrained)
        self.feature_extractor = nn.Sequential(*list(self.resnet.children())[:-1])
    def forward(self, x):
        return self.feature_extractor(x)
# Membuat instance model ekstraktor fitur
feature_extractor = FeatureExtractorResNet()
```

4.6 Create Experiments and Set Up Training Code:

Setelah membuat model, transformasi dataset, dan DataLoader, kita dapat membuat fungsi untuk menjalankan eksperimen dan kode pelatihan.

```
import torch.optim as optim

def run_experiment(model, train_loader, test_loader, criterion, optimizer, num_epochs=5):
    for epoch in range(num_epochs):
        model.train()
        for inputs, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
        # Validasi model pada dataset pengujian
        model.eval()
        with torch.no_grad():
            total_correct = 0
            total_samples = 0
            for inputs, labels in test_loader:
                outputs = model(inputs)
                _, predicted = torch.max(outputs, 1)
                total_samples += labels.size(0)
                total_correct += (predicted == labels).sum().item()
```

```
        accuracy = total_correct / total_samples
        print(f'Epoch {epoch + 1}/{num_epochs}, Accuracy: {accuracy}')
# Contoh penggunaan fungsi eksperimen
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(feature_extractor.parameters(), lr=0.001, momentum=0.9)
run_experiment(feature_extractor, cifar10_loader, cifar10_loader, criterion, optimizer, num_epochs=5)
```

Dengan menyusun serangkaian eksperimen, kita dapat dengan sistematis menjalankan dan mengevaluasi model pada berbagai konfigurasi dan dataset. Ini membantu dalam memahami kinerja model dan memilih konfigurasi terbaik untuk tugas yang dihadapi.

---

08. PyTorch Paper Replicating

---

1. Membuat Dataset dan DataLoader:

1.1 Menyiapkan Transformasi untuk Gambar:

Transformasi digunakan untuk mempersiapkan data sebelum dimasukkan ke dalam model. Dalam konteks PyTorch, transformasi seringkali mencakup resizing, normalisasi, atau augmentasi gambar. Berikut adalah contoh membuat transformasi untuk gambar:

```
from torchvision import transforms
# Menyiapkan transformasi untuk gambar
image_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize gambar ke ukuran yang diharapkan oleh model
    transforms.ToTensor(),  # Konversi gambar menjadi tensor
```

```
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalisasi nilai pixel
])
```

Transformasi di atas melakukan resize gambar menjadi 224x224 piksel, mengonversi gambar menjadi tensor, dan melakukan normalisasi nilai pixel. Transformasi ini sesuai dengan transformasi umum yang diperlukan oleh model-model torchvision.

1.2 Mengubah Gambar menjadi DataLoader:

Setelah transformasi telah disiapkan, langkah selanjutnya adalah membuat objek Dataset dan DataLoader menggunakan transformasi tersebut. Dataset berfungsi sebagai antarmuka untuk mengakses data, dan DataLoader mengatur cara data dimuat ke dalam model dalam batch. Contoh:

```
from torchvision import datasets
from torch.utils.data import DataLoader
# Membuat objek Dataset dengan transformasi
dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=image_transform)
# Membuat DataLoader
dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
```

Dalam contoh di atas, dataset CIFAR-10 digunakan sebagai ilustrasi, tetapi Anda dapat mengganti dataset sesuai kebutuhan Anda.

1.3 Visualisasi Satu Gambar:

```
import matplotlib.pyplot as plt
import numpy as np
# Mengambil satu batch dari DataLoader
for images, labels in dataloader:
```

```
    single_image = images[0].permute(1, 2, 0)  # Mengubah urutan dimensi untuk visualisasi
    single_label = labels[0]
    break
# Denormalisasi nilai pixel
denormalized_image = 0.5 * single_image + 0.5
# Visualisasi gambar
plt.imshow(denormalized_image)
plt.title(f'Label: {single_label}')
plt.show()
```

Dalam contoh di atas, kita mengambil satu batch dari DataLoader dan menampilkan satu gambar beserta labelnya. Perlu diingat bahwa proses visualisasi mungkin perlu disesuaikan tergantung pada struktur data dan format label yang digunakan.

2. Mereplikasi Paper ViT: Sebuah Gambaran Umum:

2.1 Input dan Output, Lapisan, dan Blok:

- Input dan Output: Model Vision Transformer (ViT) mengambil gambar sebagai input, dan outputnya adalah representasi vektor kelas untuk klasifikasi. Ini berarti bahwa ViT diarsiteki untuk tugas klasifikasi gambar.
- Lapisan dan Blok: ViT menggunakan pendekatan transformer, yang terkenal dalam pemrosesan bahasa alami, untuk memproses gambar. Lapisan transformer dibangun dari beberapa blok. Setiap blok memiliki dua komponen utama: Attention Mechanism untuk memahami hubungan antara berbagai bagian gambar dan Feedforward Neural Network untuk memproses informasi tersebut.

2.2 Menjadi Spesifik: Apa yang Membentuk ViT?

- Pemisahan Gambar Menjadi Patches: Gambar dipecah menjadi potongan-potongan kecil yang disebut "patches". Pemisahan ini memungkinkan representasi dari setiap bagian gambar diinputkan ke dalam model.
- Posisi dan Embeddings: Setiap patch diberi sebuah embedding posisi, dan embedding ini bersamaan dengan embedding dari nilai pixel di setiap patch. Ini memastikan bahwa model dapat memahami informasi spasial dan mempertahankan urutan piksel.
- Multiple Layers of Transformer Blocks: Blok transformer yang terdiri dari attention mechanism dan feedforward network diiterasikan beberapa kali. Setiap blok memproses representasi dari blok sebelumnya.
- Pooling dan Klasifikasi: Akhirnya, representasi output dari blok terakhir diambil dan diinputkan ke layer klasifikasi untuk menghasilkan prediksi kelas.

Catatan Penting: Pengaturan ukuran patch, jumlah blok, dan dimensi embedding adalah beberapa hyperparameter kunci yang memengaruhi kinerja model ViT.

Langkah-langkah Replikasi:

- Pemisahan Gambar:
  Mempersiapkan gambar dan membaginya menjadi patches. Membuat embedding posisi dan menggabungkannya dengan embedding nilai pixel.
- Lapisan dan Blok Transformer
  Membuat blok transformer dengan attention mechanism dan feedforward network. Mengiterasikan blok tersebut untuk membentuk lapisan transformer.
- Pooling dan Klasifikasi:
  Mengambil representasi output dari lapisan terakhir. Menghubungkannya ke layer klasifikasi untuk mendapatkan prediksi kelas.
- Eksperimen dan Hyperparameter Tuning:
  Mengevaluasi model dengan dataset klasifikasi gambar. Menyesuaikan hyperparameter seperti ukuran patch, jumlah blok, dan dimensi embedding untuk optimalitas.

Dengan mereplikasi model ViT, kita dapat memahami lebih dalam arsitektur transformer yang diterapkan pada tugas visi komputer dan menguji kinerjanya pada dataset spesifik.