TECHNICAL REPORT
MACHINE LEARNING OPTION 2
Deep Learning with PyTorch

Disusun oleh :

Nama	**: Thasya Mulia**
Nim	**: 1103201208**
Kelas	**: TK-44-04**
Dosen	**: Dr. Risman Adnan**

PROGRAM STUDI TEKNIK KOMPUTER
FAKULTAS TEKNIK ELEKTRO
UNIVERSITAS TELKOM
BANDUNG
2022/2023

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Machine Learning memainkan peran penting dalam berbagai bidang, termasuk pengenalan wajah, pengenalan suara, pengeklasifikasian teks, rekomendasi produk, prediksi pasar, pengenalan pola, pengolahan bahasa dan banyak lagi. Algoritma Machine Learning dirancang untuk mengidentifikasi pola dalam data dan menggunakan pola-pola ini untuk membuat prediksi atau mengambil keputusan yang relevan.

Machine Learning dan PyTorch adalah dua bidang yang saling terkait, di mana Machine Learning adalah konsep yang mendasari pengembangan sistem yang dapat belajar secara mandiri, sementara PyTorch adalah salah satu framework yang digunakan untuk mengimplementasikan dan melatih model Machine Learning secara efisien.

1.2 Tujuan

- Memahami PyTorch sebagai framework Machine Learning
- Menerapkan model Machine Learning dengan PyTorch
- Memenuhi kewajiban pelaksanaan ujian akhir mata kuliah Machine Learning

BAB 11
DASAR TEORI

Deep learning menggunakan neural networks buatan (model) yang merupakan sistem komputasi yang terdiri dari banyak lapisan unit yang saling terhubung. Dengan melewati data melalui unit-unit yang saling terhubung ini, neural networks dapat belajar untuk mendekati perhitungan yang diperlukan untuk mengubah masukan menjadi keluaran. Dalam PyTorch, neural networks dapat dibangun menggunakan paket `torch.nn`.

KONSEP-KONSEP DASAR PyTorch

1. Tensor: Struktur Data Utama

   Tensor adalah struktur data utama dalam PyTorch yang digunakan untuk merepresentasikan dan memanipulasi data dalam operasi Deep Learning. Tensor mirip dengan array multidimensional dan dapat berisi bilangan real, bilangan bulat, atau nilai boolean. PyTorch menyediakan berbagai fungsi untuk membuat, mengubah, dan mengakses elemen-elemen dalam tensor.

   Contoh:

```
import torch

# Membuat tensor
x = torch.tensor([1, 2, 3, 4, 5])

# Mengakses elemen tensor
print(x[0])  # Output: 1

# Mengubah bentuk tensor
y = x.view(5, 1)
print(y)
```

2. Autograd: Pemetaan Gradien

   Autograd adalah modul di PyTorch yang digunakan untuk menghitung dan melacak gradien dari operasi yang dilakukan pada tensor. Gradien sangat penting dalam Deep Learning karena digunakan dalam proses pelatihan model melalui algoritma backpropagation. PyTorch secara otomatis melacak setiap operasi yang

dilakukan pada tensor dan menyimpan informasi gradiennya, sehingga memudahkan perhitungan gradien dan pembaruan parameter model.

Contoh:

```
```
import torch

# Mengaktifkan fitur autograd
x = torch.tensor(2.0, requires_grad=True)

# Menggunakan tensor dalam perhitungan
y = 3*x + 4

# Menghitung gradien
y.backward()

Mendapatkan gradien tensor
print(x.grad) # Output: 3.0
```
```

3. Optimizer: Mengoptimalkan Model

Optimizer adalah algoritma yang digunakan untuk mengoptimalkan model Deep Learning dengan memperbarui nilai-nilai parameter berdasarkan gradien yang dihitung selama proses pelatihan. PyTorch menyediakan berbagai optimizer populer seperti Stochastic Gradient Descent (SGD), Adam, dan RMSprop. Optimizer membantu dalam mencapai konvergensi yang lebih cepat dan hasil yang lebih baik dalam pelatihan model.

Contoh:

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

# Membuat model dan optimizer
model = MyModel()

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

```
# Proses pelatihan
for epoch in range(num_epochs):
     optimizer.zero_grad()   # Menghapus gradien
sebelum setiap iterasi
    output = model(input)
    loss = compute_loss(output, target)
    loss.backward()
       optimizer.step()    #  Pembaruan  parameter
model
```

4. Dataset dan Dataloader: Memuat Data untuk Pelatihan

Dataset dan Dataloader adalah komponen penting dalam memuat dan mempersiapkan data untuk pelatihan model Deep Learning. Dataset menyimpan data pelatihan dan labelnya, sedangkan Dataloader digunakan untuk memuat data dalam batch-batch yang dapat diproses oleh model. PyTorch menyediakan kelas-kelas Dataset dan Dataloader yang memudahkan pengolahan data dalam pelatihan model.

Contoh:

```
```
import torch
from torch.utils.data import Dataset, DataLoader

# Definisikan kelas Dataset kustom
class MyDataset(Dataset):
    def __init__(self, data, labels):
        self.data = data
        self.labels = labels

    def __getitem

__(self, index):
        x = self.data[index]
        y = self.labels[index]
        return x, y
    def __len__(self):
        return len(self.data)
```
```

```
# Buat instance Dataset dan Dataloader
dataset = MyDataset(data, labels)
dataloader = DataLoader(dataset, batch_size=32,
shuffle=True)

# Proses pelatihan dengan Dataloader
for inputs, targets in dataloader:
    # Lakukan proses pelatihan
    ...
```

5. Arsitektur Model Deep Learning

Arsitektur model Deep Learning adalah struktur atau jaringan dari neuron-neuron yang digunakan untuk memproses data dan menghasilkan output yang diinginkan. Arsitektur model dapat mencakup berbagai jenis lapisan seperti lapisan konvolusi, lapisan rekurensi, dan lapisan terhubung penuh. PyTorch menyediakan kemudahan dalam mendefinisikan dan membangun arsitektur model Deep Learning dengan menggunakan kelas-kelas seperti `nn.Module` dan modul-modul terkaitnya.

Contoh:

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

# Definisikan kelas model kustom
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.fc1 = nn.Linear(10, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.fc1(x)
     x = torch.relu(x)
        x = self.fc2(x)
        return x
```
```

```
# Membuat instance model
model = MyModel()

# Menggunakan model dalam proses pelatihan atau inferensi
output = model(input)
```

BAB III
PEMBAHASAN

1. Persiapan Pembelajaran

   Sebelum memulai pembuatan model Deep Learning dengan PyTorch, perlu dilakukan beberapa langkah persiapan. Beberapa langkah yang dapat dilakukan antara lain:

   - Menginstalasi PyTorch: Pertama, pastikan PyTorch sudah terinstala. Kita dapat mengunjungi situs web resmi PyTorch untuk mendapatkan instruksi instalasi yang sesuai dengan sistem operasi dan kebutuhan.
   - Import Libraries: Di awal skrip atau notebook, impor library yang diperlukan, seperti `torch` dan `torch.nn`. Library ini akan digunakan untuk membuat dan melatih model Deep Learning.
   - Persiapan Data: Siapkan data yang akan digunakan dalam pelatihan model. Data tersebut harus dalam format yang sesuai dan dapat diakses melalui DataLoader untuk memuat batch-batch data.

2. Pembuatan Arsitektur Model

   Langkah selanjutnya adalah mendefinisikan arsitektur model Deep Learning yang akan digunakan. Arsitektur model terdiri dari berbagai lapisan (misalnya, lapisan konvolusi, lapisan rekurensi, lapisan terhubung penuh) yang akan membentuk aliran informasi dalam model. Dalam PyTorch, kita dapat menggunakan kelas `nn.Module` dan modul-modul terkait untuk mendefinisikan arsitektur model.

   Contoh:

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

# Definisikan kelas model kustom
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.fc1 = nn.Linear(10, 64)
```
```

```
    self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.fc1(x)
        x = torch.relu(x)
        x = self.fc2(x)
        return x

# Membuat instance model
model = MyModel()
```

3. Pelatihan Model

   Setelah mendefinisikan arsitektur model, langkah berikutnya adalah melatih model menggunakan data pelatihan. Dalam tahap ini, kita perlu menentukan fungsi loss, memilih optimizer, dan melakukan iterasi pelatihan untuk mengoptimalkan parameter model. Selama pelatihan, gradien akan dihitung menggunakan autograd dan model akan diperbarui menggunakan optimizer.

   Contoh:

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

# Membuat instance model dan optimizer
model = MyModel()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Proses pelatihan
for epoch in range(num_epochs):
    optimizer.zero_grad() # Menghapus gradien
sebelum setiap iterasi
    output = model(input)
    loss = compute_loss(output, target)
    loss.backward()
    optimizer.step() # Pembaruan parameter model
```
```

4. Evaluasi Model

   Setelah pelatihan selesai, langkah selanjutnya adalah mengevaluasi kinerja model yang telah dilatih. Evaluasi dapat dilakukan dengan menggunakan metrik evaluasi yang sesuai untuk tugas yang sedang dihadapi, seperti akurasi, presisi, atau recall. Evaluasi dapat dilakukan pada data validasi atau data uji yang terpisah dari data pelatihan.

   Contoh:

```
```
import torch

Evaluasi model
with torch.no_grad():
    model.eval() # Mengatur model dalam mode
evaluasi
    output = model(input)
    predicted_labels = torch.argmax(output, dim=1)
    accuracy = compute_accuracy(predicted_labels,
target)
```
```

5. Penyimpanan dan Pemulihan Model

   Setelah melatih model, seringkali penting untuk menyimpan model yang telah dilatih untuk digunakan di masa mendatang. Anda dapat menyimpan model ke dalam file menggunakan format yang sesuai, seperti format `pt` atau `pth`. Selain itu, Anda juga dapat memuat kembali model yang telah disimpan untuk penggunaan berikutnya.

   Contoh:

```
```
import torch

# Menyimpan model
torch.save(model.state_dict(), 'model.pth')

# Memuat kembali model
model = MyModel()
```
```

```
model.load_state_dict(torch.load('model.pth'))
```
```

**Percobaan Menggunakan Dataset CIFAR-10**

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import matplotlib.pyplot as plt

transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

test_dataset = datasets.CIFAR10(root='./data', train=False,
transform=transform, download=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
batch_size=128, shuffle=False)
# Mengambil beberapa gambar dari dataset pengujian
sample_images, sample_labels = next(iter(test_loader))

Menampilkan gambar-gambar tersebut
fig, axes = plt.subplots(2, 5, figsize=(12, 6))
axes = axes.ravel()
for i in range(10):
    axes[i].imshow(sample_images[i].permute(1, 2, 0))
    axes[i].set_title(f"Label: {sample_labels[i].item()}")
    axes[i].axis('off')
plt.tight_layout()
plt.show()

# Definisikan arsitektur model CNN
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1,
```
```

```
padding=1)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1,
padding=1)
        self.fc1 = nn.Linear(64 * 8 * 8, 512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        out = self.conv1(x)
        out = self.relu(out)
        out = self.maxpool(out)
        out = self.conv2(out)
        out = self.relu(out)
        out = self.maxpool(out)
        out = out.view(out.size(0), -1)
        out = self.fc1(out)
        out = self.relu(out)
        out = self.fc2(out)
        return out

# Persiapan data
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

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

train_loader =
torch.utils.data.DataLoader(dataset=train_dataset,
batch_size=128, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
batch_size=128, shuffle=False)

# Inisialisasi model, loss function, dan optimizer
model = CNN()
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Proses pelatihan
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    for i, (images, labels) in enumerate(train_loader):
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if (i + 1) % 100 == 0:
            print(f'Epoch [{epoch+1}/{num_epochs}], Step
[{i+1}/{len(train_loader)}], Loss: {loss.item()}')

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

    accuracy = 100 * correct / total
    print(f'Accuracy on test images: {accuracy}%')
```

Pada kode di atas, terdapat beberapa tahap yang dilakukan untuk menggunakan Convolutional Neural Network (CNN) dalam tugas klasifikasi gambar pada dataset CIFAR-10. Berikut penjelasan rinci untuk setiap tahap:

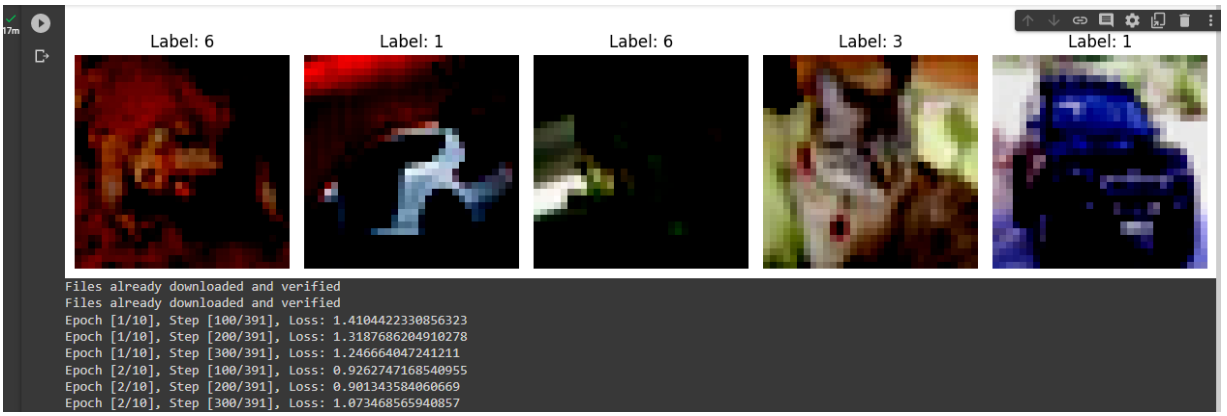
1. Persiapan Data:
    - Dilakukan transformasi pada data, termasuk penerapan flipping horizontal, konversi ke tensor, dan normalisasi.

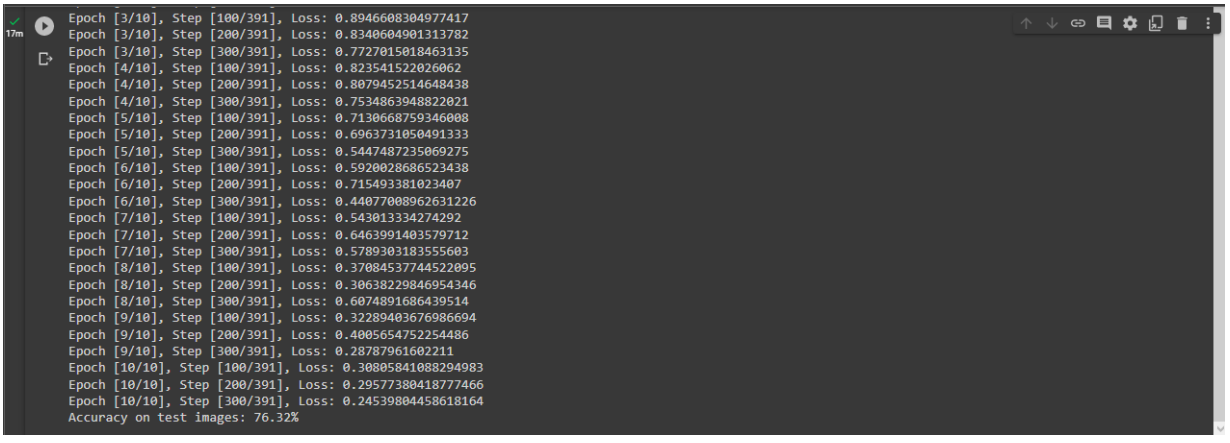
- Dataset pengujian dan pelatihan CIFAR-10 diunduh dan disimpan di direktori './data'.
- Dibuat DataLoader untuk pengujian dan pelatihan dengan menggunakan dataset yang telah diubah.

2. Menampilkan Gambar:
   - Dilakukan pengambilan beberapa gambar dari dataset pengujian menggunakan `next(iter(test_loader))`.
   - Gambar-gambar tersebut kemudian ditampilkan dalam bentuk matriks subplot menggunakan `matplotlib.pyplot`.
3. Arsitektur Model CNN:
   - Didefinisikan kelas `CNN` sebagai turunan dari `nn.Module`.
   - Model ini terdiri dari beberapa layer Convolution, ReLU, MaxPooling, dan Fully Connected (Linear).
   - Layer-layer ini diinisialisasi dalam metode `__init__` dan fungsi `forward` digunakan untuk meneruskan input melalui layer-layer tersebut.
4. Pelatihan Model:
   - Model CNN diinisialisasi.
   - Fungsi loss yang digunakan adalah CrossEntropyLoss.
   - Optimizer yang digunakan adalah Adam dengan learning rate 0.001.
   - Dilakukan pelatihan model dengan melakukan iterasi sejumlah `num_epochs`.
   - Pada setiap epoch, dilakukan iterasi pada data pelatihan menggunakan DataLoader.
   - Setiap iterasi, dihitung loss berdasarkan output model dan label, lalu dilakukan backpropagation dan optimasi menggunakan optimizer.
   - Setiap 100 langkah, dicetak loss yang terkini.
5. Evaluasi Model:
   - Model dinyatakan dalam mode evaluasi dengan menggunakan `model.eval()`.
   - Dilakukan evaluasi model pada dataset pengujian.

- Hitung akurasi dengan membandingkan label yang diprediksi oleh model dengan label asli.
- Akurasi dihitung sebagai persentase jumlah prediksi yang benar dari total data pengujian.
- Cetak akurasi pada dataset pengujian.

Gambar output pengujian :





BAB 1V
KESIMPULAN

Dalam diskusi di atas, kita membahas beberapa topik terkait dengan pemrosesan data dan model dalam PyTorch. Pertama, kita melihat bagaimana menggunakan transformasi data untuk mempersiapkan dataset, seperti mengubah ukuran gambar dan mengunduh dataset MNIST. Selanjutnya, kita melihat bagaimana membangun jaringan saraf tiruan (neural network) dengan menggunakan modul nn.Module dari PyTorch. Kita mengenal konsep forward pass, loss function, optimizer, dan proses pelatihan model menggunakan data pelatihan. Selain itu, kita juga menjelaskan penggunaan TensorBoard untuk memantau dan menganalisis pelatihan model. Selanjutnya, kita membahas cara menyimpan dan memuat model dalam PyTorch, baik dengan menyimpan seluruh model maupun hanya state_dict. Terakhir, kita melihat bagaimana mengatur pemrosesan model saat digunakan di GPU atau CPU. Diskusi ini memberikan pemahaman yang komprehensif tentang langkah-langkah penting dalam pengolahan data dan pengelolaan model menggunakan PyTorch.