TECHNICAL REPORT MACHINE LEARNING
ZERO TO MASTERY LEARN PYTORCH
FOR DEEP LEARNING

Disusun Oleh:

Ratika Dwi Anggraini (NIM: 1103201250)

PROGRAM STUDI S1 TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO

UNIVERSITAS TELKOM

2023/2024

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dalam era perkembangan teknologi dan kecerdasan buatan, pengembangan model machine learning menjadi semakin penting. PyTorch, sebagai salah satu framework machine learning yang populer, memberikan kemudahan dan fleksibilitas dalam membangun, melatih, dan mengevaluasi model. Untuk mencapai keberhasilan dalam penggunaan PyTorch, pemahaman terhadap konsep-konsep dasar, seperti pembuatan tensors, manipulasi data, dan langkah-langkah workflow, sangat diperlukan. Latar belakang ini mendasari pentingnya pemahaman mendalam terhadap PyTorch untuk memanfaatkannya secara optimal dalam pengembangan model machine learning.

PyTorch adalah kerangka kerja pembelajaran mesin dan deep learning sumber terbuka. PyTorch dapat digunakan untuk melakukan manipulasi dan pemrosesan data serta menulis algoritma pembelajaran mesin menggunakan kode Python. Banyak perusahaan teknologi terbesar di dunia, seperti Meta (Facebook), Tesla, dan Microsoft, serta perusahaan penelitian kecerdasan buatan seperti OpenAI, menggunakan PyTorch untuk mendukung penelitian dan mengintegrasikan pembelajaran mesin ke dalam produk mereka.

1.2 Tujuan

Tujuan dalam technical report ini adalah sebagai berikut:

1. Untuk dapat mengetahui dan mengimplementasikan PyTorch Fundamentals.
2. Untuk dapat mengetahui dan mengimplementasikan PyTorch Workflow Fundamentals.
3. Untuk dapat mengetahui dan mengimplementasikan PyTorch Neural Network Classification.
4. Untuk dapat mengetahui dan mengimplementasikan PyTorch Computer Vision.
5. Untuk dapat mengetahui dan mengimplementasikan PyTorch Custom Datasets.

BAB II
PEMBAHASAN

2.1 PyTorch Fundamentals

1. Membuat Tensors

   Tensors dalam PyTorch adalah struktur data fundamental yang dapat merepresentasikan berbagai jenis data, memungkinkan pemrosesan data yang efisien dalam konteks machine learning. Pembuatan tensors dapat dilakukan untuk berbagai jenis data, mulai dari skalar, vektor, matriks, hingga tensor acak.

```
# Scalar
scalar = torch.tensor(7)
# Vector
vector = torch.tensor([1, 2, 3])
# Matrix
matrix = torch.tensor([[1, 2, 3], [4, 5, 6]])
# Tensor
tensor = torch.rand(size=(2, 3, 4))
```

2. Mendapatkan Informasi dari Tensors

   Pada PyTorch, informasi dari tensors dapat diakses untuk memahami karakteristik dasar dari struktur data tersebut. Informasi yang dapat diambil dari tensors adalah bentuk (shape) dan tipe data (dtype).

```
# Getting information from tensors
tensor.shape
tensor.dtype
```

3. Memanipulasi Tensors

   Pada PyTorch, tensors dapat dengan mudah dimanipulasi melalui berbagai operasi matematika seperti perkalian, penambahan, atau operasi matriks. Operasi ini memungkinkan untuk melakukan transformasi data dan perhitungan yang mendasar dalam konteks machine learning.

```
# Manipulating tensors
result = tensor * 2
result = torch.matmul(tensor, tensor.T)
```

4. Menangani Bentuk Tensors

   Bentuk tensor mencerminkan dimensi dan panjang setiap dimensi dari struktur data tersebut. Tantangan umum yang sering dihadapi adalah ketika terjadi ketidakcocokan bentuk antara dua tensor, yang dapat menyebabkan kesalahan saat menjalankan operasi

tertentu. Penanganan kesalahan bentuk tensor menjadi kritis untuk memastikan konsistensi dalam operasi-operasi yang dilakukan pada data.

```
# Dealing with tensor shapes
tensor_a = torch.rand(3, 4)
tensor_b = torch.rand(4, 3)
# This will raise a shape mismatch error
torch.matmul(tensor_a, tensor_b)
```

5. Indexing pada Tensors

Indexing pada tensors merupakan proses untuk mengakses nilai atau subset dari tensor. Proses ini mirip dengan indexing pada struktur data lain seperti Python list atau NumPy array. Dalam PyTorch, pengindeksan dapat dilakukan dengan berbagai cara untuk memanipulasi dan mengambil data dari tensor.

```
# Indexing on tensors
tensor[0]
tensor[:, 1]
```

6. Menggabungkan Tensors PyTorch dan NumPy

Salah satu keuntungan dari kemampuan untuk beralih antara PyTorch dan NumPy adalah fleksibilitas dan interoperabilitas dalam pengembangan model machine learning. Adanya penggabungan tersebut dapat memanfaatkan keunggulan masing-masing framework tanpa perlu melakukan konversi data yang kompleks, sehingga mempermudah proses pengembangan dan eksperimen dalam pengembangan model machine learning.

```
# Mixing PyTorch tensors and NumPy
numpy_array = np.array([1, 2, 3])
tensor_from_numpy = torch.from_numpy(numpy_array)
```

7. Reproduktibilitas

Reproducibility atau reproduktibilitas merupakan aspek penting dalam pengembangan model machine learning untuk memastikan hasil eksperimen yang konsisten dan dapat direproduksi.

```
# Reproducibility
torch.manual_seed(42)
```

8. Menjalankan Tensors pada GPU

Menjalankan tensors pada GPU adalah salah satu cara untuk meningkatkan kecepatan komputasi dalam pengembangan model machine learning. GPU (Graphics Processing Unit) memiliki kemampuan paralelisme yang tinggi, yang dapat dimanfaatkan untuk melakukan perhitungan secara simultan dan mempercepat proses pelatihan model.

```
# Running tensors on GPU
device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
tensor_on_gpu = tensor.to(device)
```

2.2 PyTorch Workflow Fundamentals

1. Pendefinisian Model Regresi Linear dengan PyTorch:
   - Pembuatan kelas `LinearRegressionModel` yang merupakan subclass dari `nn.Module`.
   - Pendefinisian fungsi forward untuk menghitung prediksi model.

```
import torch
from torch import nn

class LinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.weight = nn.Parameter(torch.randn(1,
requires_grad=True))
        self.bias = nn.Parameter(torch.randn(1,
requires_grad=True))

    def forward(self, X):
        return self.weight * X + self.bias

# Inisialisasi model dan tampilkan parameter awal
torch.manual_seed(42)
model_0 = LinearRegressionModel()
print("Initial model parameters:")
print("Weight:", model_0.weight.item())
print("Bias:", model_0.bias.item())
```

2. Membuat Data untuk Regresi Linear
   - Penggunaan PyTorch tensor untuk membuat data sintetis.
   - Pembagian data menjadi set pelatihan dan pengujian.

```
# Membuat data dan membaginya menjadi set pelatihan dan pengujian
X, y = generate_data()
X_train, y_train, X_test, y_test = split_data(X, y)

# Visualisasi data
plot_predictions(X_train, y_train, X_test, y_test)
```

3. Pelatihan Model Regresi Linear
   - Penggunaan loss function `nn.L1Loss()` dan optimizer `torch.optim.SGD()`.
   - Pembuatan loop pelatihan untuk menyesuaikan parameter model.
   - Pemindahan data dan model ke perangkat yang tepat (CPU atau GPU).
   - Evaluasi model menggunakan data pelatihan dan pengujian.

```
# Pelatihan model dengan loop pelatihan
```

```
train_linear_regression_model(model_0, X_train, y_train, X_test,
y_test, epochs=500, device=device)
```

4. Visualisasi dan Analisis Hasil Pelatihan
   - Penggunaan matplotlib untuk memvisualisasikan hasil prediksi model.
   - Analisis hasil pelatihan dan evaluasi model.

```
from pathlib import Path

# Menyimpan model
MODEL_PATH = Path("models")
MODEL_PATH.mkdir(parents=True, exist_ok=True)
MODEL_NAME = "01_pytorch_workflow_model_0.pth"
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME
torch.save(obj=model_0.state_dict(), f=MODEL_SAVE_PATH)

# Memuat model yang telah disimpan
loaded_model_0 = LinearRegressionModel()
loaded_model_0.load_state_dict(torch.load(f=MODEL_SAVE_PATH))
loaded_model_0.to(device)
```

5. Menyimpan dan Memuat Model
   - Penggunaan `torch.save()` untuk menyimpan state dictionary model.
   - Memuat model yang telah disimpan dan menempatkannya di perangkat target.

```
# Evaluasi model yang telah dimuat
loaded_model_0.eval()
with torch.inference_mode():
    loaded_model_preds = loaded_model_0(X_test)
assert torch.all(y_preds == loaded_model_preds), "Loaded model
predictions differ from the original model"
```

2.3 PyTorch Neural Network Classification

1. Klasifikasi Biner
   - Persiapan Data

     Sebelum membangun model, langkah awalnya adalah memuat dan mempersiapkan data. Dalam laporan ini, kami menggunakan PyTorch dan Pandas untuk membaca dataset dan membaginya menjadi dua set: pelatihan dan pengujian. Selanjutnya, melakukan normalisasi data agar dapat diolah lebih efektif oleh model.

```
# Load and preprocess binary classification data
X, y = load_binary_classification_data()
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Normalize data
scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- Membangun Model Tanpa Fungsi Non-Linier

  Model pertama yang dibangun adalah model klasifikasi biner sederhana tanpa menggunakan fungsi aktivasi non-linier. Kami menjelaskan penggunaan fungsi kerugian dan optimizer untuk pelatihan model ini, serta mengevaluasinya menggunakan metrik akurasi. Visualisasi batas keputusan juga diberikan untuk memberikan gambaran tentang kemampuan model.

```
# Build and train a binary classification model without non-
linear activation
model_1 = BinaryClassificationModel(input_features=2).to(device)
optimizer = torch.optim.SGD(model_1.parameters(), lr=0.01)
train_binary_classification_model(model_1, X_train, y_train,
optimizer, epochs=500)
```

- Meningkatkan Model dengan Fungsi Aktivasi Non-Linier

  Untuk menangani data non-linier, fungsi aktivasi ReLU (Rectified Linear Unit) ditambahkan pada model. Kami melakukan pelatihan ulang model dengan fungsi aktivasi ini dan mengevaluasi peningkatan kinerjanya. Hasil prediksi model pada data uji juga divisualisasikan untuk membandingkan perbedaan dengan model sebelumnya.

```
# Build and train a binary classification model with non-linear
activation (ReLU)
model_2 = BinaryClassificationModel(input_features=2,
hidden_units=8, non_linear=True).to(device)
optimizer = torch.optim.SGD(model_2.parameters(), lr=0.01)
train_binary_classification_model(model_2, X_train, y_train,
optimizer, epochs=500)
```

2. Reproduksi Fungsi Aktivasi Non-Linier

   Kami menunjukkan cara mereplikasi fungsi aktivasi ReLU secara manual dan menerapkannya pada data. Selain itu, kami memperkenalkan fungsi aktivasi sigmoid dan memberikan implementasinya menggunakan PyTorch.

```
# Replicate ReLU activation function manually
def relu(x):
    return torch.maximum(torch.tensor(0), x)

# Implement custom sigmoid activation function
def sigmoid(x):
    return 1 / (1 + torch.exp(-x))
```

3. Kombinasi Fungsi Aktivasi pada Model Multi-Kelas

   - Persiapan Data Multi-Kelas

Dataset multi-kelas dibuat dengan menggunakan `make_blobs` dari Scikit-Learn. Kami mengonversi data menjadi tensor PyTorch dan membaginya menjadi set pelatihan dan pengujian. Visualisasi data multi-kelas juga disajikan untuk memberikan gambaran tentang struktur data.

```
# Create multi-class data using make_blobs
X_blob, y_blob = make_blobs(n_samples=1000, n_features=2,
centers=4, cluster_std=1.5, random_state=42)

# Convert data to PyTorch tensors
X_blob = torch.from_numpy(X_blob).type(torch.float)
y_blob = torch.from_numpy(y_blob).type(torch.LongTensor)

# Split into train and test sets
X_blob_train, X_blob_test, y_blob_train, y_blob_test =
train_test_split(X_blob, y_blob, test_size=0.2, random_state=42)

# Visualize multi-class data
plot_multiclass_data(X_blob, y_blob)
```

- Pembuatan Model Multi-Kelas

Dalam bagian ini, model multi-kelas dikembangkan dengan menambahkan fungsi aktivasi dan hidden layers. Loss function dan optimizer yang sesuai dengan tugas klasifikasi multi-kelas juga diperkenalkan.

```
# Build multi-class classification model
model_4 = BlobModel(input_features=2, output_features=4,
hidden_units=8).to(device)
optimizer = torch.optim.SGD(model_4.parameters(), lr=0.1)
```

- Prediksi Probabilitas dan Label

Kami menjelaskan proses konversi logits menjadi probabilitas prediksi menggunakan fungsi softmax dan menentukan label prediksi menggunakan argmax pada hasil softmax. Visualisasi hasil prediksi model pada data uji juga diberikan.

```
# Make predictions with the multi-class model
model_4.eval()
with torch.inference_mode():
    y_logits = model_4(X_blob_test)

# Turn predicted logits in prediction probabilities
y_pred_probs = torch.softmax(y_logits, dim=1)

# Turn prediction probabilities into prediction labels
y_preds = y_pred_probs.argmax(dim=1)
```

- Pelatihan dan Evaluasi Model Multi-Kelas

Model multi-kelas dilatih selama beberapa epoch dengan menggunakan loss dan akurasi sebagai metrik evaluasi. Kami juga memvisualisasikan hasil prediksi model pada data uji untuk memberikan gambaran tentang kemampuan model pada tugas klasifikasi multi-kelas.

```
# Train and evaluate the multi-class model
train_and_evaluate_multiclass_model(model_4, X_blob_train,
y_blob_train, X_blob_test, y_blob_test, epochs=100)
```

4. Metrik Evaluasi Klasifikasi Tambahan

Kami memperkenalkan beberapa metrik evaluasi tambahan seperti presisi, recall, F1-score, confusion matrix, dan classification report. Pilihan implementasi menggunakan PyTorch atau Scikit-Learn juga disajikan.

```
# Additional classification evaluation metrics
torchmetrics_accuracy = Accuracy(task='multiclass',
num_classes=4).to(device)
torchmetrics_accuracy(y_preds, y_blob_test)
```

2.4 PyTorch Computer Vision

1. Persiapan dan Preprocessing Data

- Import Library

  Langkah pertama adalah mengimpor library yang diperlukan untuk pengolahan data dan pembuatan model.

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
from tqdm.auto import tqdm
from pathlib import Path
```

- Persiapan Dataset

  Pada tahap ini, kita menyiapkan dataset FashionMNIST yang akan digunakan untuk pelatihan dan pengujian model. Proses ini juga melibatkan definisi transformasi untuk mempersiapkan data.

```
# Definisi transformasi
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Unduh dataset FashionMNIST
```

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

# Definisi dataloader
train_dataloader = torch.utils.data.DataLoader(train_dataset,
batch_size=32, shuffle=True)
test_dataloader = torch.utils.data.DataLoader(test_dataset,
batch_size=32, shuffle=False)
```

- Eksplorasi Data

  Dalam bagian ini, kita menjalankan eksplorasi data untuk memahami struktur dan representasi visual dari dataset. Beberapa sampel gambar dari dataset FashionMNIST ditampilkan.

```
# Eksplorasi data dengan menampilkan beberapa sampel
class_names = train_dataset.classes
sample_images, sample_labels = next(iter(train_dataloader))

# Visualisasi beberapa sampel dari dataset
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(sample_images[i].squeeze(), cmap="gray")
    plt.title(class_names[sample_labels[i]])
    plt.axis(False)
plt.show()
```

2. Pembuatan Model

- Desain Model Baseline (`model_0`)

  Model baseline dibangun dengan dua layer linear tanpa aktivasi. Ini menjadi dasar perbandingan untuk model-model lainnya.

```
class FashionMNISTModelV0(nn.Module):
    def __init__(self, input_shape, hidden_units, output_shape):
        super(FashionMNISTModelV0, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(input_shape, hidden_units)
        self.fc2 = nn.Linear(hidden_units, output_shape)

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

- Pembuatan Model dengan ReLU (`model_1`)

Model ini mirip dengan model baseline, tetapi dengan penambahan fungsi aktivasi ReLU setelah layer pertama.

```
class FashionMNISTModelV1(nn.Module):
    def __init__(self, input_shape, hidden_units, output_shape):
        super(FashionMNISTModelV1, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(input_shape, hidden_units)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_units, output_shape)

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

- Pembuatan Model CNN (`model_2`)

  Model Convolutional Neural Network (CNN) dibuat untuk meningkatkan performa model dengan memanfaatkan konvolusi untuk mengidentifikasi pola dalam citra.

```
class FashionMNISTModelV2(nn.Module):
    def __init__(self, input_shape, output_shape):
        super(FashionMNISTModelV2, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1,
padding=1)
        self.relu1 = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1,
padding=1)
        self.relu2 = nn.ReLU()
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(64 * 7 * 7, output_shape)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.maxpool1(x)

        x = self.conv2(x)
        x = self.relu2(x)
        x = self.maxpool2(x)

        x = self.flatten(x)
        x = self.fc1(x)
        return x
```

3. Pelatihan Model

- Fungsi Pelatihan dan Evaluasi

  Dua fungsi utama, `train_step` dan `test_step`, digunakan untuk melatih dan mengevaluasi model. Ini mencakup perhitungan loss, optimasi, dan pengukuran akurasi.

```
def train_step(data_loader, model, loss_fn, optimizer,
accuracy_fn, device):
    model.train()
    total_loss = 0.0
    correct_preds = 0

    for inputs, labels in data_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = loss_fn(outputs, labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        correct_preds += accuracy_fn(outputs, labels)

    avg_loss = total_loss / len(data_loader)
    accuracy = correct_preds / len(data_loader.dataset)

    return avg_loss, accuracy


def test_step(data_loader, model, loss_fn, accuracy_fn, device):
    model.eval()
    total_loss = 0.0
    correct_preds = 0

    with torch.inference_mode():
        for inputs, labels in data_loader:
            inputs, labels = inputs.to(device),
labels.to(device)

            outputs = model(inputs)
            loss = loss_fn(outputs, labels)

            total_loss += loss.item()
            correct_preds += accuracy_fn(outputs, labels)
```

```
    avg_loss = total_loss / len(data_loader)
    accuracy = correct_preds / len(data_loader.dataset)

    return avg_loss, accuracy
```

- Pelatihan `model_0`

  Model baseline (`model_0`) dilatih dengan menggunakan fungsi-fungsi yang telah dibuat sebelumnya. Hasil pelatihan dievaluasi dengan metrik-metrik yang relevan.

```
# Setup loss and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(params=model_0.parameters(), lr=0.1)

# Training
epochs = 3
for epoch in tqdm(range(epochs)):
    train_loss, train_acc = train_step(train_dataloader,
model_0, loss_fn, optimizer, accuracy_fn, device)
    print(f"Epoch {epoch + 1}/{epochs} => Train Loss:
{train_loss:.4f}, Train Accuracy: {train_acc:.4f}")

# Evaluate `model_0`
model_0_results = eval_model(model_0, test_dataloader, loss_fn,
accuracy_fn)
```

- Pelatihan `model_1`

  Model dengan ReLU (`model_1`) dilatih dan dievaluasi dengan langkah-langkah yang serupa.

```
# Setup loss and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(params=model_1.parameters(), lr=0.1)

# Training
epochs = 3
for epoch in tqdm(range(epochs)):
    train_loss, train_acc = train_step(train_dataloader,
model_1, loss_fn, optimizer, accuracy_fn, device)
    print(f"Epoch {epoch + 1}/{epochs} => Train Loss:
{train_loss:.4f}, Train Accuracy: {train_acc:.4f}")

# Evaluate `model_1`
model_1_results = eval_model(model_1, test_dataloader, loss_fn,
accuracy_fn)
```

- Pelatihan `model_2`

  Model CNN (`model_2`) dilatih dan dievaluasi untuk membandingkan hasilnya dengan model lain.

```
# Setup loss and optimizer
```

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(params=model_2.parameters(), lr=0.1)

# Training
epochs = 3
for epoch in tqdm(range(epochs)):
    train_loss, train_acc = train_step(train_dataloader,
model_2, loss_fn, optimizer, accuracy_fn, device)
    print(f"Epoch {epoch + 1}/{epochs} => Train Loss:
{train_loss:.4f}, Train Accuracy: {train_acc:.4f}")

# Evaluate `model_2`
model_2_results = eval_model(model_2, test_dataloader, loss_fn,
accuracy_fn)
```

4. Perbandingan Model

   Hasil pelatihan dari ketiga model dibandingkan menggunakan grafik untuk memberikan pemahaman visual tentang performa masing-masing model.

```
import pandas as pd

# Combine model results into a DataFrame
compare_results = pd.DataFrame([model_0_results, model_1_results,
model_2_results])

# Add training times to results comparison
compare_results["training_time"] = [total_train_time_model_0,
total_train_time_model_1, total_train_time_model_2]

# Visualize model accuracy
compare_results.set_index("model_name")["model_acc"].plot(kind="ba
rh")
plt.xlabel("accuracy (%)")
plt.ylabel("model")
```

5. Prediksi dengan Model Terbaik (`model_2`)

   - Fungsi Prediksi

     Fungsi `make_predictions` dibuat untuk membuat prediksi probabilitas dengan model pada sampel data uji.

```
def make_predictions(model, data, device=device):
    pred_probs = []
    model.eval()
    with torch.inference_mode():
        for sample in data:
            sample = torch.unsqueeze(sample, dim=0).to(device)
            pred_logit = model(sample)
            pred_prob = torch.softmax(pred_logit.squeeze(),
dim=0)
```

```
            pred_probs.append(pred_prob.cpu())

    return torch.stack(pred_probs)


# Generate random test samples
random.seed(42)
test_samples = []
test_labels = []
for sample, label in random.sample(list(test_data), k=9):
    test_samples.append(sample)
    test_labels.append(label)

# Display the first test sample shape and label
print(f"Test sample image shape: {test_samples[0].shape}\nTest
sample label: {test_labels[0]} ({class_names[test_labels[0]]})")

# Make predictions on test samples with model_2
pred_probs = make_predictions(model=model_2, data=test_samples)
```

- Visualisasi Prediksi

  Visualisasi prediksi dilakukan dengan menampilkan beberapa sampel gambar beserta label prediksi dan label sebenarnya.

```
# Turn prediction probabilities into prediction labels
pred_classes = pred_probs.argmax(dim=1)

# Plot predictions
plt.figure(figsize=(9, 9))
nrows = 3
ncols = 3
for i, sample in enumerate(test_samples):
    plt.subplot(nrows, ncols, i + 1)
    plt.imshow(sample.squeeze(), cmap="gray")
    pred_label = class_names[pred_classes[i]]
    truth_label = class_names[test_labels[i]]
    title_text = f"Pred: {pred_label} | Truth: {truth_label}"
    if pred_label == truth_label:
        plt.title(title_text, fontsize=10, c="g")
    else:
        plt.title(title_text, fontsize=10, c="r")
    plt.axis(False)
```

6. Confusion Matrix

- Import Library

  Library yang diperlukan diimpor untuk membuat dan menampilkan confusion matrix.

```
from torchmetrics import ConfusionMatrix
from mlxtend.plotting import plot_confusion_matrix
```

- Prediksi dengan `model_2`

Prediksi dilakukan dengan `model_2` pada dataset uji.

```
# Make predictions with trained model_2
y_preds = []
model_2.eval()
with torch.inference_mode():
    for X, y in tqdm(test_dataloader, desc="Making
predictions"):
        X, y = X.to(device), y.to(device)
        y_logit = model_2(X)
        y_pred = torch.softmax(y_logit, dim=1).argmax(dim=1)
        y_preds.append(y_pred.cpu())
y_pred_tensor = torch.cat(y_preds)
```

- Pembuatan Confusion Matrix dan Visualisasi

  Confusion matrix dibuat dan divisualisasikan untuk memberikan pemahaman tentang sejauh mana model dapat mengklasifikasikan dengan benar.

```
# Setup confusion matrix instance
confmat = ConfusionMatrix(num_classes=len(class_names),
task='multiclass')
confmat_tensor = confmat(preds=y_pred_tensor,
target=test_data.targets)

# Plot confusion matrix
fig, ax = plot_confusion_matrix(
    conf_mat=confmat_tensor.numpy(),
    class_names=class_names,
    figsize=(10, 7)
)
```

7. Simpan dan Muat Model Terbaik (`model_2`)

- Simpan Model

  Model terbaik (`model_2`) disimpan dalam bentuk state_dict untuk dapat dimuat kembali di masa mendatang.

```
# Create models directory (if it doesn't exist)
MODEL_PATH = Path("models")
MODEL_PATH.mkdir(parents=True, exist_ok=True)

# Define model save path
MODEL_NAME = "03_pytorch_computer_vision_model_2.pth"
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME

# Save the model state dict
print(f"Saving model to: {MODEL_SAVE_PATH}")
torch.save(obj=model_2.state_dict(), f=MODEL_SAVE_PATH)
```

- Muat Model

  Model yang telah disimpan dimuat kembali dan diuji untuk memastikan integritasnya.

```
# Create a new instance of FashionMNISTModelV2
loaded_model_2 = FashionMNISTModelV2(input_shape=1,
output_shape=10)

# Load the saved state_dict()
loaded_model_2.load_state_dict(torch.load(f=MODEL_SAVE_PATH))

# Send model to GPU
loaded_model_2 = loaded_model_2.to(device)
```

- Evaluasi Model yang Dimuat

  Model yang dimuat dievaluasi untuk memastikan bahwa hasilnya mirip dengan model sebelum disimpan.

```
# Evaluate the loaded model
loaded_model_2_results = eval_model(
    model=loaded_model_2,
    data_loader=test_dataloader,
    loss_fn=loss_fn,
    accuracy_fn=accuracy_fn
)

# Check if results are close
torch.isclose(
    torch.tensor(model_2_results["model_loss"]),
    torch.tensor(loaded_model_2_results["model_loss"]),
    atol=1e-08,
    rtol=0.0001
)
```

2.5 PyTorch Custom Datasets

1. Instalasi dan Persiapan

   Pertama-tama, dilakukan pemasangan framework PyTorch dan pengimporan pustaka yang dibutuhkan. Selain itu, beberapa konfigurasi awal juga disesuaikan.

```
# Instalasi PyTorch
!pip install torch torchvision

# Pengimporan pustaka yang diperlukan
import torch
import torch.nn as nn
import torchvision
from torchvision import transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
from pathlib import Path
from typing import List
```

2. Persiapan Dataset dan DataLoade

Data yang digunakan merupakan dataset gambar makanan yang terdiri dari kategori pizza, steak, dan sushi. Dataset ini dipersiapkan dan disiapkan menjadi DataLoader untuk memudahkan proses pelatihan dan evaluasi model.

```
# Penentuan path dataset
data_path = Path("path/to/your/dataset")

# Pembuatan transformasi untuk augmentasi data
train_transform_trivial_augment = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.TrivialAugmentWide(num_magnitude_bins=31),
    transforms.ToTensor()
])

# Pembuatan transformasi untuk data testing tanpa augmentasi
test_transform = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.ToTensor()
])

# Pembuatan Dataset dan DataLoader
train_data_augmented =
torchvision.datasets.ImageFolder(root=data_path / "train",
transform=train_transform_trivial_augment)
test_data_simple = torchvision.datasets.ImageFolder(root=data_path
/ "test", transform=test_transform)

train_dataloader_augmented = DataLoader(train_data_augmented,
batch_size=32, shuffle=True)
test_dataloader_simple = DataLoader(test_data_simple,
batch_size=32, shuffle=False)
```

3. Pembangunan Model Pertama: TinyVGG

Selanjutnya, dilakukan pembangunan model TinyVGG yang akan digunakan sebagai dasar untuk klasifikasi gambar.

```
class TinyVGG(nn.Module):
    # Implementasi layer-layer model
    # ...

# Contoh penggunaan model
torch.manual_seed(42)
model_0 = TinyVGG(
    input_shape=3,
    hidden_units=10,
    output_shape=len(train_data_augmented.classes)
)
```

4. Pelatihan dan Evaluasi Model Pertama**

Langkah berikutnya adalah melatih model pertama dengan konfigurasi dan metrik evaluasi yang tepat.

```
# Penentuan jumlah epochs
NUM_EPOCHS = 5

# Pengaturan fungsi loss dan optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model_0.parameters(),
lr=0.001)

# Inisiasi waktu awal
from timeit import default_timer as timer
start_time = timer()
model_0_results = train(model=model_0,
                        train_dataloader=train_dataloader_augmente
d,
                        test_dataloader=test_dataloader_simple,
                        optimizer=optimizer,
                        loss_fn=loss_fn,
                        epochs=NUM_EPOCHS)
end_time = timer()
print(f"Total waktu pelatihan: {end_time - start_time:.3f} detik")

# Plot kurva loss
plot_loss_curves(model_0_results)
```

5. Penanganan Overfitting dan Underfitting

Ketika model mengalami overfitting atau underfitting, beberapa tindakan dapat diambil untuk menangani masalah tersebut.

```
# Pencegahan overfitting
# 1. Perolehan lebih banyak data
# 2. Penyederhanaan model
# 3. Pemanfaatan augmentasi data
# 4. Pemanfaatan transfer learning
# 5. Penggunaan dropout layers
# 6. Penerapan learning rate decay
# 7. Penerapan early stopping
```

6. Model Kedua: TinyVGG dengan Data Augmentation

Selanjutnya, dilakukan implementasi model kedua dengan menerapkan augmentasi data pada proses pelatihan.

```
# Pembuatan transformasi untuk augmentasi data
train_transform_trivial_augment = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.TrivialAugmentWide(num_magnitude_bins=31),
    transforms.ToTensor()
])
```

```
# Pembuatan transformasi untuk data testing tanpa augmentasi
test_transform = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.ToTensor()
])

# Pembuatan Dataset dan DataLoader
train_data_augmented

 = torchvision.datasets.ImageFolder(root=data_path / "train",
transform=train_transform_trivial_augment)
test_data_simple = torchvision.datasets.ImageFolder(root=data_path
/ "test", transform=test_transform)

train_dataloader_augmented = DataLoader(train_data_augmented,
batch_size=32, shuffle=True)
test_dataloader_simple = DataLoader(test_data_simple,
batch_size=32, shuffle=False)

# Pembuatan dan pelatihan model kedua
torch.manual_seed(42)
model_1 = TinyVGG(
    input_shape=3,
    hidden_units=10,
    output_shape=len(train_data_augmented.classes)
).to(device)

# Penentuan jumlah epochs
NUM_EPOCHS = 5

# Pengaturan fungsi loss dan optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model_1.parameters(),
lr=0.001)

# Inisiasi waktu awal
start_time = timer()
model_1_results = train(model=model_1,
                        train_dataloader=train_dataloader_augmente
d,
                        test_dataloader=test_dataloader_simple,
                        optimizer=optimizer,
                        loss_fn=loss_fn,
                        epochs=NUM_EPOCHS)
end_time = timer()
print(f"Total waktu pelatihan: {end_time - start_time:.3f} detik")
```

```
# Plot kurva loss
plot_loss_curves(model_1_results)
```

7. Perbandingan Hasil Model Pertama dan Kedua

   Meskipun performa kedua model belum memuaskan, maka dapat membandingkannya menggunakan plot kurva loss dan akurasi pada set pelatihan dan pengujian.

```
# Ubah hasil model menjadi DataFrame Pandas
import pandas as pd
model_0_df = pd.DataFrame(model_0_results)
model_1_df = pd.DataFrame(model_1_results)

# Plot hasil model
plt.figure(figsize=(15, 10))
epochs = range(len(model_0_df))

# Plot train loss
plt.subplot(2, 2, 1)
plt.plot(epochs, model_0_df["train_loss"], label="Model 0")
plt.plot(epochs, model_1_df["train_loss"], label="Model 1")
plt.title("Train Loss")
plt.xlabel("Epochs")
plt.legend()

# Plot test loss
plt.subplot(2, 2, 2)
plt.plot(epochs, model_0_df["test_loss"], label="Model 0")
plt.plot(epochs, model_1_df["test_loss"], label="Model 1")
plt.title("Test Loss")
plt.xlabel("Epochs")
plt.legend()

# Plot train accuracy
plt.subplot(2, 2, 3)
plt.plot(epochs, model_0_df["train_acc"], label="Model 0")
plt.plot(epochs, model_1_df["train_acc"], label="Model 1")
plt.title("Train Accuracy")
plt.xlabel("Epochs")
plt.legend()

# Plot test accuracy
plt.subplot(2, 2, 4)
plt.plot(epochs, model_0_df["test_acc"], label="Model 0")
plt.plot(epochs, model_1_df["test_acc"], label="Model 1")
plt.title("Test Accuracy")
plt.xlabel("Epochs")
plt.legend()
```

8. Prediksi pada Gambar Kustom

   Terakhir, membuat fungsi untuk membuat prediksi pada gambar kustom.

```python
# Fungsi prediksi dan plotting gambar
def pred_and_plot_image(model: torch.nn.Module,
                        image_path: str,
                        class_names: List[str] = None,
                        transform=None,
                        device: torch.device = device):
    # Implementasi fungsi
    # ...

# Penggunaan fungsi prediksi pada gambar kustom
pred_and_plot_image(model=model_1,
                    image_path=custom_image_path,
                    class_names=class_names,
                    transform=custom_image_transform,
                    device=device)
```

BAB III
KESIMPULAN

Secara garis besar, eksplorasi dasar-dasar PyTorch dalam membuat model machine learning memberikan pemahaman yang mendalam tentang langkah-langkah penting dalam pengembangan model. Mulai dari pembuatan tensors hingga manipulasi data, pemahaman dasar ini memberikan fondasi yang kuat untuk menguasai prinsip-prinsip inti dalam menggunakan PyTorch. Model regresi linear merupakan langkah awal yang menunjukkan bagaimana cara untuk membuat model dan melatihnya.

Klasifikasi biner dan multi-kelas menggambarkan kompleksitas tugas klasifikasi, dengan penekanan pada peningkatan kinerja model melalui penggunaan fungsi aktivasi non-linier. Selain itu, pentingnya pengendalian keacakan dan reproduktibilitas dapat memastikan eksperimen yang konsisten. Penggunaan GPU sebagai akselerator komputasi membuktikan bahwa PyTorch dapat dioptimalkan untuk menangani tugas komputasi yang lebih berat.

Eksplorasi PyTorch untuk tugas computer vision telah memberikan wawasan mendalam terhadap persiapan data, perancangan model, pelatihan, dan evaluasi model. Analisis perbandingan hasil pelatihan memberikan pemahaman yang signifikan mengenai keunggulan model CNN dalam tugas klasifikasi citra dibandingkan dengan model linear konvensional. Selanjutnya, penerapan augmentasi data pada model TinyVGG berhasil meningkatkan kemampuan generalisasi model. Keseluruhan, laporan ini memberikan panduan langkah demi langkah untuk membangun, melatih, dan mengevaluasi model PyTorch dalam domain computer vision, serta memperkenalkan konsep penggunaan augmentasi data sebagai strategi mitigasi overfitting.