In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries inst
# It is defined by the kaggle/python Docker image: https://github.com/kaggle
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# You can also write temporary files to /kaggle/temp/, but they won't be sav
```

```
/kaggle/input/single-electron/SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5
/kaggle/input/photon-data/SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5
```

1. Introduction

This report describes a deep learning approach to classify image-based data of photons and electrons. The objective is to distinguish between photon and electron events using a convolutional neural network (CNN), specifically leveraging the ResNet18 architecture in PyTorch.

2. Environment Setup

Install necessary libraries:

In [2]:
```
!pip install torch torchvision torchaudio
!pip install numpy matplotlib scikit-learn
!pip install h5py
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-pack
ages (2.5.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dis
t-packages (0.20.1+cu121)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.10/dist
-packages (2.5.1+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-p
ackages (from torch) (3.17.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/p
ython3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-p
ackages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-pac
kages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-pac
kages (from torch) (2024.12.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/d
ist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python
3.10/dist-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-pack
ages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/pyth
on3.10/dist-packages (from torchvision) (11.0.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.1
0/dist-packages (from jinja2->torch) (3.0.2)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.10/dist-pa
ckages (from numpy->torchvision) (1.3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python3.10/dist
-packages (from numpy->torchvision) (1.2.4)
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.10/dist-
packages (from numpy->torchvision) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.10/dist-packag
es (from numpy->torchvision) (2025.0.1)
Requirement already satisfied: tbb4py in /usr/local/lib/python3.10/dist-pac
kages (from numpy->torchvision) (2022.0.0)
Requirement already satisfied: mkl-service in /usr/local/lib/python3.10/dis
t-packages (from numpy->torchvision) (2.4.1)
Requirement already satisfied: intel-openmp>=2024 in /usr/local/lib/python
3.10/dist-packages (from mkl->numpy->torchvision) (2024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.10/dis
t-packages (from mkl->numpy->torchvision) (2022.0.0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.10/dis
t-packages (from tbb==2022.*->mkl->numpy->torchvision) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/lib/python
3.10/dist-packages (from mkl_umath->numpy->torchvision) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in /usr/local/l
ib/python3.10/dist-packages (from intel-openmp>=2024->mkl->numpy->torchvisi
on) (2024.2.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-pack
ages (1.26.4)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist
-packages (3.7.5)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/di
st-packages (1.2.2)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.10/dist-pa
ckages (from numpy) (1.3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python3.10/dist
-packages (from numpy) (1.2.4)
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.10/dist-
packages (from numpy) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.10/dist-packag
es (from numpy) (2025.0.1)
Requirement already satisfied: tbb4py in /usr/local/lib/python3.10/dist-pac
```

```
kages (from numpy) (2022.0.0)
Requirement already satisfied: mkl-service in /usr/local/lib/python3.10/dis
t-packages (from numpy) (2.4.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/di
st-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
10/dist-packages (from matplotlib) (4.55.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.
10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/d
ist-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/pytho
n3.10/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/di
st-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/d
ist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pytho
n3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-p
ackages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Requirement already satisfied: intel-openmp>=2024 in /usr/local/lib/python
3.10/dist-packages (from mkl->numpy) (2024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.10/dis
t-packages (from mkl->numpy) (2022.0.0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.10/dis
t-packages (from tbb==2022.*->mkl->numpy) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/lib/python
3.10/dist-packages (from mkl_umath->numpy) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in /usr/local/l
ib/python3.10/dist-packages (from intel-openmp>=2024->mkl->numpy) (2024.2.
0)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packa
ges (3.12.1)
Requirement already satisfied: numpy>=1.19.3 in /usr/local/lib/python3.10/d
ist-packages (from h5py) (1.26.4)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.10/dist-pa
ckages (from numpy>=1.19.3->h5py) (1.3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python3.10/dist
-packages (from numpy>=1.19.3->h5py) (1.2.4)
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.10/dist-
packages (from numpy>=1.19.3->h5py) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.10/dist-packag
es (from numpy>=1.19.3->h5py) (2025.0.1)
Requirement already satisfied: tbb4py in /usr/local/lib/python3.10/dist-pac
kages (from numpy>=1.19.3->h5py) (2022.0.0)
Requirement already satisfied: mkl-service in /usr/local/lib/python3.10/dis
t-packages (from numpy>=1.19.3->h5py) (2.4.1)
Requirement already satisfied: intel-openmp>=2024 in /usr/local/lib/python
3.10/dist-packages (from mkl->numpy>=1.19.3->h5py) (2024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.10/dis
t-packages (from mkl->numpy>=1.19.3->h5py) (2022.0.0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.10/dis
t-packages (from tbb==2022.*->mkl->numpy>=1.19.3->h5py) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/lib/python
3.10/dist-packages (from mkl_umath->numpy>=1.19.3->h5py) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in /usr/local/l
```

```
ib/python3.10/dist-packages (from intel-openmp>=2024->mkl->numpy>=1.19.3->h
5py) (2024.2.0)
```

These packages are required for deep learning (PyTorch), data handling (NumPy, HDF5), and visualization.

3. Dataset Loading and Preprocessing

Load photon and electron images from HDF5 files:

In [3]:
```
import h5py
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset, random_split
import matplotlib.pyplot as plt

# Load dataset
def load_data(photon_path, electron_path):
    # Load photon data from HDF5
    with h5py.File(photon_path, "r") as f:
        photons = f["X"][:]  # Replace "X" with the actual dataset name ins

    # Load electron data from HDF5
    with h5py.File(electron_path, "r") as f:
        electrons = f["X"][:]  # Replace "X" with the actual dataset name in

    # Label assignment: Photon (1), Electron (0)
    photon_labels = np.ones(len(photons))
    electron_labels = np.zeros(len(electrons))

    # Combine data
    X = np.concatenate((photons, electrons), axis=0)  # Shape: (num_samples,
    y = np.concatenate((photon_labels, electron_labels), axis=0)

    # Normalize data
    X = X.astype(np.float32) / 255.0  # Scale pixel values to range [0,1]

    return X, y

# Paths to HDF5 files
photon_path = "/kaggle/input/photon-data/SinglePhotonPt50_IMGCROPS_n249k_RH
electron_path = "/kaggle/input/single-electron/SingleElectronPt50_IMGCROPS_

# Load the dataset
X, y = load_data(photon_path, electron_path)

# Convert to PyTorch tensors
X_tensor = torch.tensor(X, dtype=torch.float32).permute(0, 3, 1, 2)  # Conve
y_tensor = torch.tensor(y, dtype=torch.long)

# Split into train (80%) and test (20%) sets
train_size = int(0.8 * len(X))
test_size = len(X) - train_size
train_dataset, test_dataset = random_split(TensorDataset(X_tensor, y_tensor

# Dataloader
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Print shapes for verification
```

```
print("Train dataset size:", len(train_dataset))
print("Test dataset size:", len(test_dataset))
```

```
Train dataset size: 398400
Test dataset size: 99600
```

4. Model Definition (ResNet15)

In [4]:

```
import torch.nn.functional as F
from torchvision.models import resnet18

class ResNet15(nn.Module):
    def __init__(self, num_classes=2):
        super(ResNet15, self).__init__()
        self.resnet = resnet18(pretrained=False)  # Menggunakan ResNet18 sel
        self.resnet.conv1 = nn.Conv2d(2, 64, kernel_size=3, stride=1, paddi
        self.resnet.fc = nn.Linear(512, num_classes)  # Ubah output layer

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

# Inisialisasi model
model = ResNet15().to('cuda' if torch.cuda.is_available() else 'cpu')
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: U
serWarning: The parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: U
serWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavio
r is equivalent to passing `weights=None`.
  warnings.warn(msg)
```

5. Training the Model

In [5]:

```
# Hyperparameters
epochs = 10
learning_rate = 0.001
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Training loop
for epoch in range(epochs):
    model.train()
    total_loss = 0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        _, predicted = outputs.max(1)
        correct += predicted.eq(labels).sum().item()
        total += labels.size(0)

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {total_loss/len(train_loader)
```

```
Epoch [1/10], Loss: 0.6143, Accuracy: 66.74%
Epoch [2/10], Loss: 0.5691, Accuracy: 71.34%
Epoch [3/10], Loss: 0.5602, Accuracy: 72.07%
Epoch [4/10], Loss: 0.5552, Accuracy: 72.35%
Epoch [5/10], Loss: 0.5519, Accuracy: 72.65%
Epoch [6/10], Loss: 0.5492, Accuracy: 72.77%
Epoch [7/10], Loss: 0.5469, Accuracy: 72.92%
Epoch [8/10], Loss: 0.5445, Accuracy: 73.08%
Epoch [9/10], Loss: 0.5430, Accuracy: 73.23%
Epoch [10/10], Loss: 0.5416, Accuracy: 73.33%
```

6. Model Evaluation

In [6]:
```
# Evaluation function
def evaluate(model, test_loader):
    model.eval()
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = outputs.max(1)
            correct += predicted.eq(labels).sum().item()
            total += labels.size(0)

    print(f"Test Accuracy: {100 * correct/total:.2f}%")

# Run evaluation
evaluate(model, test_loader)
```

```
Test Accuracy: 72.62%
```

Optimized Model 1

In [14]:
```
.from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split

def load_data(photon_path, electron_path):
    with h5py.File(photon_path, "r") as f:
        photons = f["X"][:]

    with h5py.File(electron_path, "r") as f:
        electrons = f["X"][:]

    photon_labels = np.ones(len(photons))
    electron_labels = np.zeros(len(electrons))

    X = np.concatenate((photons, electrons), axis=0)
    y = np.concatenate((photon_labels, electron_labels), axis=0)

    # Normalisasi dan ubah format ke channel-first (NCHW)
    X = X.astype(np.float32) / 255.0
    X = np.transpose(X, (0, 3, 1, 2))  # NHWC → NCHW
    return X, y

photon_path = "/kaggle/input/photon-data/SinglePhotonPt50_IMGCROPS_n249k_RH
electron_path = "/kaggle/input/single-electron/SingleElectronPt50_IMGCROPS_

X, y = load_data(photon_path, electron_path)

# Split dataset
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, rand

X_train_tensor = torch.tensor(X_train)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
X_val_tensor = torch.tensor(X_val)
y_val_tensor = torch.tensor(y_val, dtype=torch.long)

train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
val_dataset = TensorDataset(X_val_tensor, y_val_tensor)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

In [15]:
```
import torchvision.models as models
import torch.nn as nn

model = models.resnet18(pretrained=False)

# Ganti layer pertama agar support 2 channels
model.conv1 = nn.Conv2d(2, 64, kernel_size=7, stride=2, padding=3, bias=Fal

# Ganti output layer (2 kelas: photon & electron)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)  # Softmax untuk 2 kelas
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: U
serWarning: The parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: U
serWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavio
r is equivalent to passing `weights=None`.
  warnings.warn(msg)
```

In [19]:
```
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
import torch.nn.functional as F

device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = model.to(device)

learning_rate = 0.0005
weight_decay = 1e-2

criterion = nn.CrossEntropyLoss()
optimizer = optim.AdamW(model.parameters(), lr=learning_rate, weight_decay=
scheduler = StepLR(optimizer, step_size=3, gamma=0.5)
```

In [20]:
```
epochs = 20

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    scheduler.step()

    accuracy = 100 * correct / total
    print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss/len(train_loade

print("Training complete!")
```

```
Epoch [1/20], Loss: 0.5438, Accuracy: 73.14%
Epoch [2/20], Loss: 0.5428, Accuracy: 73.16%
Epoch [3/20], Loss: 0.5421, Accuracy: 73.24%
Epoch [4/20], Loss: 0.5330, Accuracy: 73.91%
Epoch [5/20], Loss: 0.5304, Accuracy: 74.02%
Epoch [6/20], Loss: 0.5283, Accuracy: 74.20%
Epoch [7/20], Loss: 0.5209, Accuracy: 74.65%
Epoch [8/20], Loss: 0.5183, Accuracy: 74.84%
Epoch [9/20], Loss: 0.5162, Accuracy: 74.95%
Epoch [10/20], Loss: 0.5105, Accuracy: 75.28%
Epoch [11/20], Loss: 0.5086, Accuracy: 75.46%
Epoch [12/20], Loss: 0.5068, Accuracy: 75.52%
Epoch [13/20], Loss: 0.5033, Accuracy: 75.74%
Epoch [14/20], Loss: 0.5018, Accuracy: 75.79%
Epoch [15/20], Loss: 0.5008, Accuracy: 75.90%
Epoch [16/20], Loss: 0.4982, Accuracy: 76.04%
Epoch [17/20], Loss: 0.4979, Accuracy: 76.07%
Epoch [18/20], Loss: 0.4971, Accuracy: 76.09%
Epoch [19/20], Loss: 0.4959, Accuracy: 76.14%
Epoch [20/20], Loss: 0.4956, Accuracy: 76.17%
Training complete!
```

In [21]:
```
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in val_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

val_acc = 100 * correct / total
print(f"Validation Accuracy: {val_acc:.2f}%")
```

```
Validation Accuracy: 71.51%
```

7. Conclusion and Suggestions

- Final training accuracy: ~76%
- Final test accuracy: ~71.5%

Suggestions for Improvement:

- Apply image augmentation using torchvision.transforms
- Add regularization (Dropout layers)
- Tune hyperparameters (learning rate, scheduler, optimizer)
- Add early stopping to prevent overfitting