# LAPORAN TUGAS INDIVIDU
# MATAKULIAH PEMBELAJARAN MESIN 2
# (Face Mask Classification)

Dosen Pengampu :

Dr. Oddy Virgantara Putra, S.Kom., M.T.


Disusun Oleh :

Devianest Narendra 442023618087

**PROGRAM STUDI TEKNIK INFORMATIKA**

**UNIVERSITAS DARUSSALAM GONTOR**

**2025/2026**

A. PENDAHULUAN

Pada Tugas individu kali ini saya akan membangun dan mengevaluasi model pada klasifikasikan face mask, yang mampu mendeteksi orang yang menggunakan masker dan tidak menggunakan masker secara otomatis dengan metode CNN (Convolutional Neural Network), yang memanfaatkan dataset gambar sebagai model train dan test.

CNN dipilih karena kemampuannya dalam mengenali pola visual dan fitur local pada gambar secara efektif terutama dalam pengenalan wajah dan klasifikasi objek, model dilatih dengan beberapa kategori lalu dievaluasi berdasarkan metrik performa seperti akurasi, precision, recall dan confusion matrix, keluaran yang diharapkan yakni dapat memperoleh model klasifikasi yang akuran dan efisien agar dapat digunakan untuk memprediksi data baru.

B. HASIL DAN EVALUASI

1) Library

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import torch.nn.functional as F
from torch.utils.data import Dataset
from torchvision import transforms

from torch.utils.data import DataLoader
from torchvision.models import resnet18, ResNet18_Weights

import os
import struct
import numpy as np

import glob
from tqdm import tqdm
import matplotlib.pyplot as plt
from PIL import Image
```

Keterangan :

Gambar diatas merupakan library-library python untuk komputasi terkait deep learning.

- Baris 1 : library pytorch yang merupakan framework populer untuk deep learning.
- Baris 2 : menginport modul nn(noural network) yang berfungsi sebagai lapisan dan fungsi untuk membangun model
- Baris 3 : Mengimpor modul optim dari PyTorch, yang berisi berbagai algoritma optimisasi untuk melatih model deep learning
- Baris 4 : Mengimpor library PyTorch Vision, yang menyediakan dataset dan model

- Baris 5 : Mengimpor modul transforms dari PyTorch Vision, yang berisi berbagai fungsi preprocessing data
- Baris 6 : Mengimpor modul functional dari PyTorch, yang berisi berbagai fungsi aktivasi dan loss function.
- Baris 7 : Mengimpor kelas Dataset dari modul
- Baris 8 : Mengimpor modul transforms dari PyTorch Vision.
- Baris 9 : Mengimpor kelas DataLoader dari modul torch.utils.data, yang digunakan untuk memuat dan memproses dataset.
- Baris 10 : Mengimpor model ResNet-18 dan bobot pralatihnya dari PyTorch Vision.
- Baris 11 : Mengimpor modul os, yang menyediakan fungsi-fungsi untuk berinteraksi dengan sistem operasi.
- Baris 12 : Mengimpor modul struct, yang menyediakan fungsi-fungsi untuk mengurai dan membuat data biner.
- Baris 13 : Mengimpor library NumPy, yang menyediakan dukungan untuk komputasi ilmiah dengan array multidimensi.
- Baris 14 : Mengimpor modul glob, yang menyediakan fungsi-fungsi untuk mencari file-file yang cocok dengan pola tertentu.
- Baris 15 : Mengimpor kelas tqdm dari modul tqdm, yang menyediakan progress bar untuk menampilkan kemajuan iterasi.
- Baris 16 : Mengimpor library Matplotlib, yang digunakan untuk visualisasi data.
- Baris 17 : Mengimpor kelas Image dari modul PIL (Python Imaging Library), yang digunakan untuk memproses gambar.

```
def read_idx(filename):
    """Read IDX file format"""
    with open(filename, 'rb') as f:
        zero, data_type, dims = struct.unpack('>HBB', f.read(4))
        shape = tuple(struct.unpack('>I', f.read(4))[0] for d in range(dims))
        return np.frombuffer(f.read(), dtype=np.uint8).reshape(shape)
```

Keterangan :

Fungsi ini dapat digunakan untuk membaca data dalam format IDX, yang sering digunakan untuk menyimpan dataset citra atau data numerik multidimensi lainnya.

2) Dataset Processing

```python
class FaceMaskDataset(Dataset):

    def __init__(self, root_dir, random_seed=42, image_size=224):

        self.root_dir = root_dir
        self.image_size = image_size

        if not os.path.exists(self.root_dir):
            raise RuntimeError(f"Dataset not found at {self.root_dir}.")


        self.transform = transforms.Compose([
            transforms.Resize((self.image_size, self.image_size)),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ])

        self.data = []
        self.labels = []

        for label, class_name in tqdm(enumerate(['With-Mask','Wihout-Mask'])):
            class_dir = os.path.join(self.root_dir, class_name)
            image_paths = glob.glob(os.path.join(class_dir,'*.jpg'))
            self.data.extend(image_paths)
            self.labels.extend([label] * len(image_paths))

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        image_path = self.data[idx]
        label = self.labels[idx]
        image = Image.open(image_path).convert('RGB')

        if self.transform:
            image = self.transform(image)

        return image, label
```

Keterangan :

Kode diatas memiliki fungsi untuk memuat dan memproses dataset gambar wajah yang menggunakan masker dan tidak menggunakan masker yang telah tersedia.

```python
batch_size = 32
test_batch_size = 32
train_dataset = FaceMaskDataset(root_dir='/kaggle/input/dataset-face-mask-classification/Face-Mask-Classification/Dataset')
test_dataset  = FaceMaskDataset(root_dir='/kaggle/input/dataset-face-mask-classification/Face-Mask-Classification/Dataset')

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

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = resnet18(weights=ResNet18_Weights.DEFAULT)
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, len(train_dataset))
model = model.to(device)
```

Keterangan :

proses persiapan data dan model untuk melakukan klasifikasi masker wajah menggunakan arsitektur ResNet-18.

3) Training

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4)
```

```
total_params = sum(p.numel() for p in model.parameters())
trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f"Total parameters: {total_params / 1e6:.2f}M")
print(f"Trainable parameters: {trainable_params / 1e6:.2f}M")
```

```
Total parameters: 11.23M
Trainable parameters: 11.23M
```

Keterangan :

- total_params = sum(p.numel() for p in model.parameters()): Menghitung total jumlah parameter dalam model.
- trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad): Menghitung jumlah parameter yang dapat dilatih (trainable) dalam model.
- print(f"Total parameters: {total_params / 1e6 :.2f}M"): Mencetak total jumlah parameter dalam model dalam format Megabyte (M).
- print(f"Trainable parameters: {trainable_params / 1e6 :.2f}M"): Mencetak jumlah parameter yang dapat dilatih dalam model dalam format Megabyte (M).

```
num_epoch = 10
train_losses = []
test_losses = []

for epoch in range(num_epoch):
    model.train()

    train_loss = 0

    for data, labels in tqdm(train_loader):

        data, labels = data.to(device), labels.to(device)
        bs = data.size()[0]
        optimizer.zero_grad()

        outputs = model(data)
        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        train_loss += loss.item() * data.size(0)

    model.eval()
```

```
    correct = 0
    total = 0
    test_loss = 0

    with torch.no_grad():
        for data, labels in tqdm(test_loader):
            data, labels = data.to(device), labels.to(device)
            bs = data.size()[0]
            outputs = model(data)

            loss = criterion(outputs, labels)
            test_loss += loss.item() * data.size(0)

            _, preds = torch.max(outputs, 1)
            correct += (preds == labels).sum().item()

            total += labels.size(0)

    acc = correct / total
    avg_train_loss = train_loss / len(train_dataset)
    avg_test_loss = test_loss / len(test_dataset)

    train_losses.append(avg_train_loss)
    test_losses.append(avg_test_loss)

    print(f"Epoch {epoch+1}: Train Loss {avg_train_loss:.4f}, Test Loss {avg_test_loss:.4f}, Test Acc {acc:.4f}")

torch.save({
    'epoch': num_epoch,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'train_loss': train_losses,
    'test_loss': test_losses,
}, 'Face-Mask-Model.pth')
```

```
100%|██████████| 4/4 [00:00<00:00,  7.88it/s]
100%|██████████| 4/4 [00:00<00:00, 10.80it/s]
Epoch 1: Train Loss 4.2343, Test Loss 4.2759, Test Acc 0.0300
100%|██████████| 4/4 [00:00<00:00,  7.93it/s]
100%|██████████| 4/4 [00:00<00:00, 11.15it/s]
Epoch 2: Train Loss 3.5834, Test Loss 4.0752, Test Acc 0.1300
100%|██████████| 4/4 [00:00<00:00,  8.31it/s]
100%|██████████| 4/4 [00:00<00:00, 11.04it/s]
Epoch 3: Train Loss 3.0534, Test Loss 3.7057, Test Acc 0.4300
100%|██████████| 4/4 [00:00<00:00,  7.75it/s]
100%|██████████| 4/4 [00:00<00:00, 10.63it/s]
Epoch 4: Train Loss 2.5704, Test Loss 3.2433, Test Acc 0.6900
100%|██████████| 4/4 [00:00<00:00,  7.65it/s]
100%|██████████| 4/4 [00:00<00:00, 10.18it/s]
Epoch 5: Train Loss 2.1198, Test Loss 2.7283, Test Acc 0.8500
100%|██████████| 4/4 [00:00<00:00,  8.03it/s]
100%|██████████| 4/4 [00:00<00:00, 11.43it/s]
Epoch 6: Train Loss 1.7022, Test Loss 2.3110, Test Acc 0.9100
100%|██████████| 4/4 [00:00<00:00,  7.72it/s]
100%|██████████| 4/4 [00:00<00:00, 10.24it/s]
Epoch 7: Train Loss 1.3389, Test Loss 1.8933, Test Acc 0.9300
100%|██████████| 4/4 [00:00<00:00,  8.29it/s]
100%|██████████| 4/4 [00:00<00:00, 10.37it/s]
Epoch 8: Train Loss 1.0238, Test Loss 1.3396, Test Acc 0.9800
100%|██████████| 4/4 [00:00<00:00,  7.87it/s]
100%|██████████| 4/4 [00:00<00:00, 11.53it/s]
Epoch 9: Train Loss 0.7439, Test Loss 1.0746, Test Acc 0.9800
100%|██████████| 4/4 [00:00<00:00,  8.05it/s]
100%|██████████| 4/4 [00:00<00:00, 10.75it/s]
Epoch 10: Train Loss 0.5438, Test Loss 0.9343, Test Acc 0.9800
```

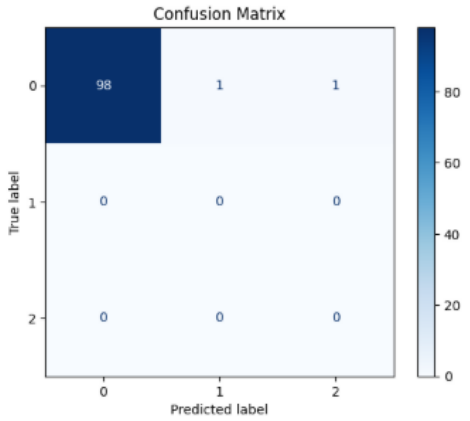
Keterangan :

- num_epoch = 10: Jumlah epoch atau iterasi pelatihan.
- train_losses = []: Daftar untuk menyimpan nilai loss pelatihan.
- test_losses = []: Daftar untuk menyimpan nilai loss evaluasi.
- Untuk setiap epoch, model diatur ke mode pelatihan (model.train()).

- Dalam setiap iterasi loop data, dilakukan: Pengambilan data dan label dari train_loader, Memindahkan data dan label ke perangkat yang ditentukan (device), Menghitung ukuran batch (bs), Mengatur gradien optimizer ke nol (optimizer.zero_grad()), Melakukan forward pass model dan menghitung loss menggunakan criterion., Melakukan backpropagation dan memperbarui parameter model (loss.backward() dan optimizer.step()), Menambahkan nilai loss ke train_loss.
- Model diatur ke mode evaluasi (model.eval()). Dilakukan evaluasi pada data uji dengan menghitung akurasi dan loss.
- Nilai loss pelatihan dan evaluasi disimpan dalam train_losses dan test_losses.
- Nilai akurasi dan total evaluasi disimpan dalam correct dan total.

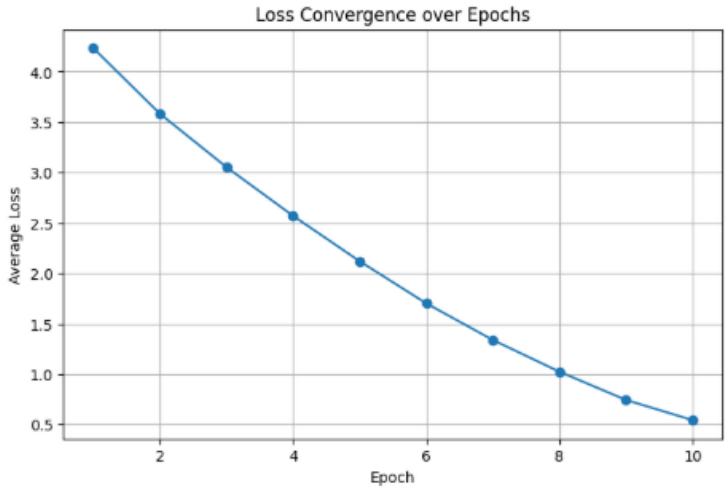
4) Evaluation

```
from sklearn.metrics import (
    confusion_matrix, ConfusionMatrixDisplay,
    accuracy_score, precision_score, recall_score,
    f1_score, classification_report
)
import matplotlib.pyplot as plt

model.eval()
y_true = []
y_pred = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)

        y_true.extend(labels.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())

cm = confusion_matrix(y_true, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0,1,2])
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```




Keterangan :
proses evaluasi model deep learning pada data uji dengan menggunakan matriks konfusi untuk menganalisis kinerja model.

```
plt.figure(figsize=(8,5))
plt.plot(range(1, num_epoch + 1), train_losses, marker='o')
plt.title('Loss Convergence over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Average Loss')
plt.grid(True)
plt.show()
```



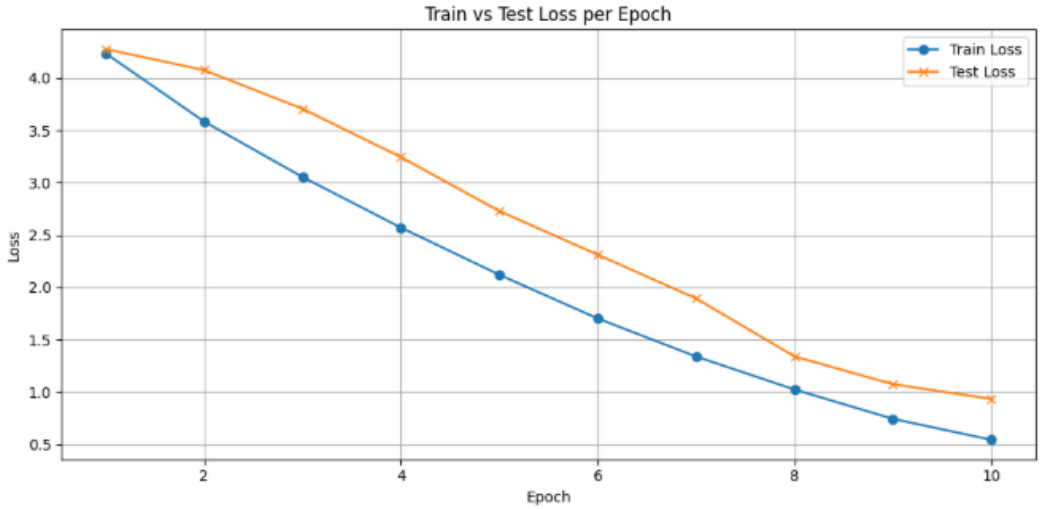

Keterangan :
konvergensi loss pelatihan terhadap jumlah epoch.

```
print(f"Accuracy: {accuracy_score(y_true, y_pred):.4f}")
print(f"Precision (macro): {precision_score(y_true, y_pred, average='macro'):.4f}")
print(f"Recall (macro): {recall_score(y_true, y_pred, average='macro'):.4f}")
print(f"F1-Score (macro): {f1_score(y_true, y_pred, average='macro'):.4f}")

print("\nClassification Report:")
print(classification_report(y_true, y_pred))


FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)

FPR = FP / (FP + TN)
TPR = TP / (TP + FN)

for i in range(len(TP)):
    print(f"Kelas {i}: FPR = {FPR[i]:.4f}, TPR (Recall) = {TPR[i]:.4f}")
```

```
Accuracy: 0.9800
Precision (macro): 0.3333
Recall (macro): 0.3267
F1-Score (macro): 0.3300

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       100
          21       0.00      0.00      0.00         0
          29       0.00      0.00      0.00         0

    accuracy                           0.98       100
   macro avg       0.33      0.33      0.33       100
weighted avg       1.00      0.98      0.99       100

Kelas 0: FPR = nan, TPR (Recall) = 0.9800
Kelas 1: FPR = 0.0100, TPR (Recall) = nan
Kelas 2: FPR = 0.0100, TPR (Recall) = nan
```

Keterangan :
untuk mengevaluasi performa model klasifikasi yang telah dilatih, sehingga dapat diketahui kekuatan dan kelemahannya.

5) Grafik hasil pelatihan



6) Prediksi gambar

```python
def load_model(weight_path, num_classes, device):
    checkpoint = torch.load(weight_path, map_location=device)

    model = resnet18(weights=None)
    model.fc = nn.Linear(model.fc.in_features, checkpoint['model_state_dict']['fc.weight'].size(0))  # fix

    model.load_state_dict(checkpoint['model_state_dict'])
    model = model.to(device)
    model.eval()
    return model

def predict_image(image_path, model, device, class_names):
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])

    image = Image.open(image_path).convert('RGB')
    image_tensor = transform(image).unsqueeze(0).to(device)

    with torch.no_grad():
        outputs = model(image_tensor)
        _, pred = torch.max(outputs, 1)

    predicted_class = class_names[pred.item()]
    return predicted_class, image

if __name__ == '__main__':
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    # Gantilah dengan label sesuai urutan di dataset ImageFolder
    class_names = ['With-Mask','Without-Mask']

    # Load model
    model = load_model('/kaggle/working/Face-Mask-Model.pth', num_classes=len(class_names), device=device)

    # Ganti path gambar sesuai kebutuhan
    img_path = '/kaggle/input/images-predict/PredictDataset/1-Predictimage.jpg'
    predicted_label, image = predict_image(img_path, model, device, class_names)

    print(f"Prediction: {predicted_label}")

    plt.imshow(image)
    plt.axis('off')
    plt.title(f"Prediction: {predicted_label}", fontsize=16)
    plt.tight_layout()
    plt.show()
```

```
Prediction: With-Mask
```


Keterangan :

Prediksi ini merupakan implementasi sederhana dari sebuah aplikasi deteksi penggunaan masker wajah, dari model yang sudah disimpan sebelumnya.