MODEL PEMBELAJARAN DAN LAPORAN AKHIR
PROJECT-BASED LEARNING
MATA KULIAH DEEP LEARNING
KELAS C

"IMPLEMENTASI ALGORITMA *CONVOLUTIONAL NEURAL NETWORK* (CNN) UNTUK KLASIFIKASI DETEKSI PEROKOK"

DISUSUN OLEH KELOMPOK "I":

1. ADELIA AZIZATUL HAQ (21083010009)
2. CHELSEA AYU A. (21083010028)
3. EDELIN FORTUNA (21083010087)
4. DIANA SINTHYA PUTRI (21083010090)
5. YUNITA NUR (21083010107)

DOSEN PENGAMPU:

ANDRI FAUZAN ADZIIMA, M.SI. (199502122024061001)

PROGRAM STUDI SAINS DATA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS PEMBANGUNAN NASIONAL "VETERAN"
JAWA TIMUR
2024

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 *Background*

Merokok merupakan salah satu penyebab utama berbagai penyakit kronis, termasuk kanker paru-paru, penyakit jantung, dan gangguan pernapasan. Menurut data dari Organisasi Kesehatan Dunia (WHO), lebih dari 8 juta orang meninggal setiap tahun akibat penyakit yang disebabkan oleh merokok. Jumlah perokok aktif di Indonesia saat masih mengalami peningkatan. Menurut Data Survei Kesehatan Indonesia (SKI) 2023 yang dilakukan oleh Kementerian Kesehatan (Kemenkes), jumlah perokok aktif diperkirakan mencapai 70 juta orang, dengan 7,4% di antaranya adalah perokok berusia 10-18 tahun. Kelompok anak-anak dan remaja adalah kelompok yang mengalami kenaikan jumlah perokok yang paling signifikan. Berdasarkan data Global Youth Tobacco Survey (GYTS) pada 2019, jumlah perokok di kalangan siswa usia 13-15 tahun meningkat dari 18,3% (2016) menjadi 19,2% (2019). Di sisi lain, data SKI 2023 menunjukkan bahwa kelompok usia 15-19 tahun adalah kelompok perokok terbanyak (56,5%), diikuti oleh usia 10-14 tahun (18,4%).

Peningkatan jumlah perokok ini berdampak pada meningkatnya perilaku merokok sembarangan, termasuk di kawasan yang seharusnya bebas dari rokok. Hal ini bertentangan dengan Undang-Undang Republik Indonesia Nomor 36 Tahun 2009 tentang Kesehatan, Pasal 115 Ayat 2, yang menyebutkan bahwa pemerintah daerah wajib menetapkan kawasan tanpa rokok (KTR) di wilayahnya. KTR adalah ruangan atau area yang dinyatakan dilarang untuk kegiatan merokok atau kegiatan memproduksi, menjual, mengiklankan, dan/atau mempromosikan produk tembakau. Untuk meningkatkan kinerja penerapan dan penegakan KTR ini, kepala daerah membentuk tim dan harus melakukan pengawasan melalui inspeksi mendadak (sidak) secara rutin setiap bulan dan melaporkannya secara berkala. Implementasi pengawasan terhadap kawasan tanpa rokok ini sering kali mengalami kendala. Pengawasan manual membutuhkan tenaga manusia yang harus terus menerus memantau kawasan tertentu, sementara sumber daya manusia yang tersedia terbatas. Selain itu, pemerintah harus mengalokasikan biaya lebih untuk menggaji petugas pengawas, yang menjadikan pendekatan ini kurang efisien, terutama jika diterapkan dalam skala besar.

Untuk mengatasi tantangan tersebut, diperlukan solusi berbasis teknologi seperti penerapan algoritma *Convolutional Neural Network* (CNN) yang dapat mengolah data gambar seperti penelitian yang dilakukan oleh Wijayanti pada tahun 2024 tentang deteksi kematangan buah stroberi. Oleh karena itu, algoritma CNN digunakan untuk mendeteksi perilaku merokok secara otomatis melalui analisis gambar atau video. Sistem ini memungkinkan pengawasan dilakukan secara *real-time* dengan cakupan yang lebih luas tanpa ketergantungan pada tenaga manusia dan bertujuan untuk membandingkan kinerja tiga variasi algoritma CNN yaitu AlexNet, MobileNetV2, dan ResNet50 dalam mendeteksi dan mengklasifikasi aktivitas merokok. Dengan menggunakan dataset gambar yang berisi dua kelas, yaitu merokok dan tidak merokok. Dengan demikian, teknologi ini tidak hanya membantu menegakkan aturan kawasan tanpa rokok secara lebih efektif, tetapi juga mendukung upaya pemerintah dalam mengurangi perilaku merokok sembarangan dan melindungi masyarakat dari paparan asap rokok.

1.2 *Issue*

Pengawasan kawasan tanpa rokok secara manual menghadapi tantangan, terutama dalam hal efisiensi waktu dan biaya. Proses pengawasan manual membutuhkan tenaga pengawas yang harus bekerja secara bergilir atau terus menerus untuk memantau perilaku masyarakat di berbagai lokasi. Hal ini tentu memakan banyak waktu, terutama jika area yang harus diawasi mencakup wilayah luas atau tempat yang sulit dijangkau. Selain itu, pengawasan manual juga membutuhkan alokasi anggaran yang besar untuk membayar gaji para petugas pengawas. Dalam jangka panjang, pendekatan ini tidak hanya membebani anggaran pemerintah, tetapi juga kurang efektif karena keterbatasan manusia dalam memantau area secara konsisten dan menyeluruh. Oleh karena itu, diperlukan solusi yang lebih efisien baik dari segi waktu maupun biaya, seperti penggunaan teknologi yang dapat melakukan pengawasan otomatis dengan cakupan yang lebih luas dan tanpa memerlukan tenaga manusia dalam jumlah besar.

1.3 *Objective*

Proyek ini bertujuan untuk mengembangkan algoritma *Convolutional Neural Network* (CNN) yang mampu mendeteksi perilaku merokok secara otomatis melalui analisis gambar atau video. Selain itu, penelitian ini juga bertujuan untuk membandingkan performa tiga variasi arsitektur CNN, yaitu AlexNet, MobileNetV2, dan ResNet50 dalam mendeteksi aktivitas

merokok. Perbandingan ini mencakup evaluasi akurasi dari masing-masing arsitektur, sehingga dapat menentukan model yang paling optimal untuk digunakan dalam skenario deteksi perokok di kawasan tanpa rokok. Hasil dari penelitian ini diharapkan dapat memberikan solusi berbasis teknologi yang lebih efisien dan efektif untuk mendukung implementasi kebijakan kawasan tanpa rokok.

1.4 *Benefit*

Penelitian ini memiliki beberapa manfaat penting. Pertama, penelitian ini memberikan kontribusi dalam pengembangan sistem deteksi perilaku berbasis kecerdasan buatan (AI), khususnya di bidang klasifikasi visual. Dengan menggunakan *Convolutional Neural Network* (CNN), penelitian ini membuka peluang untuk memperluas aplikasi teknologi AI dalam mendeteksi perilaku tertentu, termasuk pemantauan aktivitas di tempat umum. Kedua, sistem berbasis CNN mampu mempercepat proses identifikasi perokok secara signifikan dibandingkan dengan metode manual. Analisis otomatis yang dilakukan oleh sistem ini memungkinkan deteksi perilaku merokok dilakukan secara *real-time* tanpa memerlukan banyak tenaga manusia, sehingga lebih efisien dalam hal waktu dan sumber daya. Ketiga, teknologi ini mendukung implementasi kebijakan kawasan tanpa rokok dengan menyediakan alat yang canggih untuk memantau pelanggaran secara efisien. Sistem ini dapat membantu pemerintah dan institusi terkait dalam menegakkan aturan serta menciptakan lingkungan yang lebih sehat dan bebas dari asap rokok.

BAB II
METODOLOGI PENELITIAN

Pada PJBL ini, kami membandingkan empat metode untuk mencari metode terbaik yang cocok untuk mengklasifikasi deteksi perokok. Adapun metode yang kami gunakan, yaitu AlexNet, MobileNetV2, ResNet50, dan *Smoking method* yang nantinya akan dievaluasi kinerjanya dan memilih satu model dengan akurasi terbaik dalam mendeteksi kegiatan merokok. Metode terbaik tersebut akan dilakukan proses deployment berupa pembuatan *Graphical User Interface* (GUI). Adapun langkah-langkahnya adalah sebagai berikut.

2.1 Ekstrak File Zip Dataset

```
from zipfile import ZipFile

# Replace with the actual path to your zip file in Google Drive
zip_file_path = "/content/drive/My Drive/DATA-UAS.zip"
extract_path = "/content/drive/My Drive/extracted_files" # Replace with
your desired extraction path


try:
    with ZipFile(zip_file_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)
        print(f"Successfully extracted {zip_file_path} to
{extract_path}")
except FileNotFoundError:
    print(f"Error: Zip file not found at {zip_file_path}")
except Exception as e:
    print(f"An error occurred: {e}")
```

Output:

```
Successfully extracted /content/drive/My Drive/DATA-UAS.zip to
/content/drive/My Drive/extracted_files
```

2.2 Splitting Data

```
import os
import shutil
from sklearn.model_selection import train_test_split

def split_dataset(source_dir, dest_dir, split_ratios=(0.8, 0.1, 0.1)):

    assert sum(split_ratios) == 1.0, "Total rasio harus sama dengan
1!"
```

```
    # List kategori (label), yaitu nama subfolder dalam source_dir
    categories = os.listdir(source_dir)

    for category in categories:
        category_path = os.path.join(source_dir, category)

        # Periksa apakah ini folder kategori (bukan file biasa)
        if not os.path.isdir(category_path):
            continue

        # Ambil semua file dalam kategori ini
        files = os.listdir(category_path)
        files = [os.path.join(category_path, f) for f in files if
os.path.isfile(os.path.join(category_path, f))]

        # Split dataset untuk kategori ini
        train_files, temp_files = train_test_split(files, test_size=(1
- split_ratios[0]), random_state=42)
        val_files, test_files = train_test_split(temp_files,
test_size=(split_ratios[2] / (split_ratios[1] + split_ratios[2])),
random_state=42)

        # Copy file ke folder train, val, dan test sesuai kategori
        for split, split_files in zip(['train', 'val', 'test'],
[train_files, val_files, test_files]):
            split_dir = os.path.join(dest_dir, split, category)  #
Buat direktori sesuai kategori
            os.makedirs(split_dir, exist_ok=True)  # Buat direktori
jika belum ada

            # Salin file ke direktori tujuan
            for file_path in split_files:
                shutil.copy(file_path, split_dir)
                print(f"Copied {file_path} to {split_dir}")

    print("Dataset berhasil dipisahkan berdasarkan label!")

# Path direktori asal dan tujuan
source_directory = '/content/drive/MyDrive/UAS-SMOKING'
destination_directory = "/content/drive/MyDrive/DATA-UAS"

# Memisahkan dataset dengan rasio 80% train, 10% val, 10% test
split_dataset(source_directory, destination_directory,
split_ratios=(0.8, 0.1, 0.1))
```

Adapun detail penjelasan dari tahapan pembagian data menjadi data latih, data uji, dan data validasi adalah sebagai berikut.

```
def split_dataset(source_dir, dest_dir, split_ratios=(0.8, 0.1, 0.1)):
```

Memisahkan dataset menjadi data *train*, data *validation*, dan data *test* berdasarkan labelnya, yaitu *smoking* dan *not smoking*. `Source_dir` adalah direktori sumber dataset yang berisi 2 folder yang dijadikan sebagai label, *smoking* dan *not smoking*. `Dest_dir` adalah direktori tujuan hasil *splitting*. `Split_ratio` adalah rasio untuk *train*, val, dan *test*.

```
assert sum(split_ratios) == 1.0, "Total rasio harus sama dengan 1!"
```

Memastikan bahwa total nilai yang ada pada `splot_ratio` adalah 1.

```
categories = os.listdir(source_dir)
```

Menjadikan nama subfolder yang ada pada `source_dir` sebagai label.

```
for category in categories:
    category_path = os.path.join(source_dir, category)

    if not os.path.isdir(category_path):
        continue
```

Membuat *path* lengkap untuk folder label dan memastikan hanya bentuk direktori yang diproses.

```
files = os.listdir(category_path)
files = [os.path.join(category_path, f) for f in files if
os.path.isfile(os.path.join(category_path, f))]
```

Mengambil semua file yang ada di `category_path`.

```
train_files, temp_files = train_test_split(files, test_size=(1 -
split_ratios[0]), random_state=42)
val_files, test_files = train_test_split(temp_files,
test_size=(split_ratios[2] / (split_ratios[1] + split_ratios[2])),
random_state=42)
```

Diawali dengan membagi data menjadi data *train* dan *temp*. Kemudian *temp* dibagi lagi menjadi *test* dan val dengan perbandingan 1:1.

```
for split, split_files in zip(['train', 'val', 'test'], [train_files,
val_files, test_files]):
    split_dir = os.path.join(dest_dir, split, category)
    os.makedirs(split_dir, exist_ok=True)

    for file_path in split_files:
```

```
        shutil.copy(file_path, split_dir)
        print(f"Copied {file_path} to {split_dir}")
```

Membuat folder *train*, val, dan *test* dari masing-masing label dan menyalin file yang sesuai ke dalam folder tersebut.

```
# Path direktori asal dan tujuan
source_directory = '/content/drive/MyDrive/UAS-SMOKING'
destination_directory = "/content/drive/MyDrive/DATA-UAS"

# Memisahkan dataset dengan rasio 80% train, 10% val, 10% test
split_dataset(source_directory, destination_directory,
split_ratios=(0.8, 0.1, 0.1))
```

Membagi dataset dengan 80% data *training*, 10% data validasi, dan 10% data uji.

2.3 Memeriksa Lokasi Data dan Jumlah File Gambar pada Data

```
# Tentukan path folder
dataset = '/content/drive/MyDrive/DATA-UAS'

# Periksa isi folder
for root, dirs, files in os.walk(dataset):
    print(f"Path: {root}")
    print(f"Subdirektori: {dirs}")
    print(f"Jumlah file: {len(files)}\n")
```

Output :

```
Path: /content/drive/MyDrive/DATA-UAS
Subdirektori: ['CSV', 'test', 'val', 'train']
Jumlah file: 0

Path: /content/drive/MyDrive/DATA-UAS/CSV
Subdirektori: []
Jumlah file: 1

Path: /content/drive/MyDrive/DATA-UAS/test
Subdirektori: ['notsmoking', 'smoking']
Jumlah file: 0

Path: /content/drive/MyDrive/DATA-UAS/test/notsmoking
Subdirektori: []
Jumlah file: 352

Path: /content/drive/MyDrive/DATA-UAS/test/smoking
Subdirektori: []
Jumlah file: 339

Path: /content/drive/MyDrive/DATA-UAS/val
Subdirektori: ['notsmoking', 'smoking']
```

```
Jumlah file: 0

Path: /content/drive/MyDrive/DATA-UAS/val/notsmoking
Subdirektori: []
Jumlah file: 352

Path: /content/drive/MyDrive/DATA-UAS/val/smoking
Subdirektori: []
Jumlah file: 338

Path: /content/drive/MyDrive/DATA-UAS/train
Subdirektori: ['notsmoking', 'smoking']
Jumlah file: 0

Path: /content/drive/MyDrive/DATA-UAS/train/notsmoking
Subdirektori: []
Jumlah file: 2815

Path: /content/drive/MyDrive/DATA-UAS/train/smoking
Subdirektori: []
Jumlah file: 2706
```

Pada folder dataset terdapat dua folder lain, yaitu *smoking* dan *not smoking*. Di masing-masing subfolder tersebut, terdapat 3 folder lagi, yaitu *test, train,* dan val.

2.4 Transformasi Data

```
def save_dataloader_to_csv(dataloader, dataset_name, output_dir):
    os.makedirs(output_dir, exist_ok=True)
    csv_path = os.path.join(output_dir,
f"{dataset_name}_dataset.csv")

    # List untuk menyimpan data
    data = []

    # Iterasi DataLoader
    for batch_idx, (images, labels) in enumerate(dataloader):
        batch_size = images.size(0)
        for i in range(batch_size):
            # Ambil tensor gambar dan label
            image_tensor = images[i]
            label = labels[i].item()

            # Flatten tensor menjadi satu dimensi
            flattened_tensor = image_tensor.flatten().numpy()

            # Simpan ke list
            data.append({
                "image_name": f"{dataset_name}_img_{batch_idx *
batch_size + i}",  # Nama file sebagai placeholder
                "label": label,
```

```
                "tensor_values": ",".join(map(str, flattened_tensor))
            })

    # Buat DataFrame dari data
    df = pd.DataFrame(data)
    df.to_csv(csv_path, index=False)
    print(f"Data {dataset_name} berhasil disimpan ke {csv_path}")

basic_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Adapun penjelasan detail dari masing-masing baris adalah sebagai berikut.

```
def save_dataloader_to_csv(dataloader, dataset_name, output_dir):
```

Menyimpan dataloader ke dalam bentuk CSV. Dataloader merupakan data *pytorch* yang berisi *image* dan label. `Dataset_name` adalah nama splitting dataset (val, *train, test*). `Output_dir` merupakan direktori menyimpan file CSV.

```
os.makedirs(output_dir, exist_ok=True)
csv_path = os.path.join(output_dir, f"{dataset_name}_dataset.csv")
```

Membuat folder `output_dir` jika belum tersedia. Memberi nama file CSV sesuai dengan `dataset_name`.

```
data = []

for batch_idx, (images, labels) in enumerate(dataloader):
    batch_size = images.size(0)
    for i in range(batch_size):
        image_tensor = images[i]
        label = labels[i].item()

        flattened_tensor = image_tensor.flatten().numpy()

        data.append({
                "image_name": f"{dataset_name}_img_{batch_idx *
batch_size + i}",  # Nama file sebagai placeholder
                "label": label,
                "tensor_values": ",".join(map(str, flattened_tensor))
            })
```

Membuat list untuk menyimpan data dari dataloader sebelum masuk ke CSV. Melakukan iterasi gambar pada dataloader dan mengubah ke dalam bentuk tensor. Data yang sudah diubah ke dalam bentuk tensor akan dimasukkan pada file CSV.

```
df = pd.DataFrame(data)
df.to_csv(csv_path, index=False)
print(f"Data {dataset_name} berhasil disimpan ke {csv_path}")
```

Membuat dataframe dari data dan menyimpan dalam file CSV.

```
transforms.Resize((224, 224))
```

Menyesuaikan ulang ukuran gambar dengan melakukan pemotongan menjadi 224x224.

```
transforms.ToTensor()
```

Mengubah gambar menjadi format tensor

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

Melakukan normalisasi data gambar dengan nilai rata-rata dan standar deviasi dari masing-masing channel RGB (Red, Green, Blue).

2.5 Ekstraksi Label

```
source_base = "/content/drive/MyDrive/DATA-UAS"
train_dataset = datasets.ImageFolder(os.path.join(source_base,
"train"), transform=basic_transform)
val_dataset = datasets.ImageFolder(os.path.join(source_base, "val"),
transform=basic_transform)
test_dataset = datasets.ImageFolder(os.path.join(source_base, "test"),
transform=basic_transform)
```

Ekstraksi label bertujuan untuk memastikan bahwa terdapat data label yang akan dijadikan sebagai target/*output* hasil prediksi. Ekstraksi label diambil dari subfolder yang berada pada folder DATA-UAS. Masing-masing data yang ada pada subfolder tersebut akan dilakukan transformasi sesuai dengan tahapan transformasi pada `basic_transform`.

2.6 Membuat data loader

```
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

dataloaders = {
    'train': train_loader,
    'val': val_loader,
    'test': test_loader
}

# Direktori Output CSV
```

```
output_dir = "/content/drive/MyDrive/DATA-UAS/CSV"

# Simpan DataLoader ke CSV
save_dataloader_to_csv(train_loader, "train", output_dir)
save_dataloader_to_csv(val_loader, "val", output_dir)
save_dataloader_to_csv(test_loader, "test", output_dir)
```

Data loader bertujuan untuk mempermudah pengelolaan dataset selama pelatihan, validasi, dan pengujian model machine learning. Menyimpan dataloader ke CSV dari masing-masing dataloader (*train, test,* val).

2.7 Pemodelan dan Evaluasi dari AlexNet, MobileNetV2, dan ResNet50

a. Menentukan model yang ingin dilatih

```
models_list = [
    models.resnet50(weights=models.ResNet50_Weights.DEFAULT),  #
ResNet50
    models.alexnet(weights=models.AlexNet_Weights.DEFAULT),  #
AlexNet

models.mobilenet_v2(weights=models.MobileNet_V2_Weights.DEFAULT)
# MobileNetV2
]
```

Menentukan model yang akan digunakan ke dalam bentuk *list*.

b. Menyusun model untuk klasifikasi biner

```
for model in models_list:
    # Modifikasi final layer untuk klasifikasi biner
    if isinstance(model, models.ResNet):
        model.fc = nn.Linear(model.fc.in_features, 1)
    elif isinstance(model, models.AlexNet):
        model.classifier[6] =
nn.Linear(model.classifier[6].in_features, 1)
    elif isinstance(model, models.MobileNetV2):
        model.classifier[1] =
nn.Linear(model.classifier[1].in_features, 1)
```

Model akan mengklasifikasikan dan memberikan *input features* pada lapisan final. Nantinya model akan menghasilkan 1 nilai *output*, yaitu label pertama atau label kedua.

c. Mentransfer model pada GPU

```
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
model = model.to(device)
```

Mengecek apakah GPU tersedia di dalam sistem, jika tersedia maka model akan dijalankan pada GPU, tetapi jika tidak terdeteksi maka model akan tetap dijalankan di CPU dengan durasi yang lebih lambat.

d. Menerapkan optimizer pada model

```
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

Memperbarui parameter model selama proses pelatihan menggunakan *Stochastic Gradient Descent* yang cara kerjanya memperbarui parameter berdasarkan *gradient* terhadap *loss function.*

e. Memastikan proses perubahan pada model berhasil

```
print(f"Model {model.__class__.__name__} berhasil dimodifikasi.")
```

Output:

```
Downloading:
"https://download.pytorch.org/models/resnet50-11ad3fa6.pth" to
/root/.cache/torch/hub/checkpoints/resnet50-11ad3fa6.pth
100%|██████████| 97.8M/97.8M [00:00<00:00, 186MB/s]
Downloading:
"https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to
/root/.cache/torch/hub/checkpoints/alexnet-owt-7be5be79.pth
100%|██████████| 233M/233M [00:01<00:00, 161MB/s]
Downloading:
"https://download.pytorch.org/models/mobilenet_v2-7ebf99e0.pth"
to /root/.cache/torch/hub/checkpoints/mobilenet_v2-7ebf99e0.pth
100%|██████████| 13.6M/13.6M [00:00<00:00, 134MB/s]
Model ResNet berhasil dimodifikasi.
Model AlexNet berhasil dimodifikasi.
Model MobileNetV2 berhasil dimodifikasi.
```

f. Training model

```
# fungsi untuk train model
def train_model(model, dataloaders, criterion, optimizer,
num_epochs=10, device='cuda'):
    best_acc = 0.0  # Menyimpan akurasi terbaik
    best_model_wts = deepcopy(model.state_dict())  # Menyimpan
bobot terbaik model

    for epoch in range(num_epochs):
        print(f'Epoch {epoch + 1}/{num_epochs}')

        # Loop untuk training dan validasi
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0
```

```
            # Progress bar untuk batch
            loop = tqdm(dataloaders[phase],
desc=f'{phase.capitalize()} Phase', leave=False)

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

                # Pastikan label memiliki dimensi (batch_size, 1)
untuk BCEWithLogitsLoss
                labels = labels.view(-1, 1).float()

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)  # Output:
(batch_size, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # Konversi logits ke probabilitas untuk akurasi
                probs = torch.sigmoid(outputs)
                preds = (probs > 0.5).long()  # Prediksi biner: 0
atau 1

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds ==
labels.data)

                # Update progress bar
                loop.set_postfix(loss=loss.item())

            epoch_loss = running_loss /
len(dataloaders[phase].dataset)
            epoch_acc = running_corrects.double() /
len(dataloaders[phase].dataset)

            print(f'{phase.capitalize()} Loss: {epoch_loss:.4f}
Acc: {epoch_acc:.4f}')

            # Simpan loss dan akurasi untuk grafik
            if phase == 'train':
                train_losses.append(epoch_loss)
                train_accuracies.append(epoch_acc)
            else:
                val_losses.append(epoch_loss)
                val_accuracies.append(epoch_acc)

            # Simpan model terbaik berdasarkan akurasi validasi
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = deepcopy(model.state_dict())
```

```
    # Muat bobot terbaik setelah training selesai
    model.load_state_dict(best_model_wts)
    print(f'Best Validation Accuracy: {best_acc:.4f}')
    return model, train_losses, val_losses, train_accuracies,
val_accuracies
```

Adapun penjelasan detail dari masing-masing *syntax* tersebut adalah sebagai berikut.

```
best_acc = 0.0
best_model_wts = deepcopy(model.state_dict())
```

`Best_acc` digunakan untuk menyimpan akurasi terbaik sedangkan `best_model_wts` digunakan untuk menyimpan bobot terbaik model.

```
for epoch in range(num_epochs):
    print(f'Epoch {epoch + 1}/{num_epochs}')
```

Melakukan proses iterasi dengan maksimal proses adalah 30 iterasi.

```
for phase in ['train', 'val']:
    if phase == 'train':
        model.train()
    else:
        model.eval()
```

Menentukan fase dari model, jika model dalam fase train maka model akan dilatih. Tetapi jika model dalam fase val maka model akan dievaluasi kinerjanya.

```
loop = tqdm(dataloaders[phase], desc=f'{phase.capitalize()}
Phase', leave=False)

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

Menampilkan progress bar untuk masing-masing *batch* menggunakan tqdm. Input dan label juga dipindahkan pada device yang terdeteksi, CPU atau GPU.

```
labels = labels.view(-1, 1).float()
optimizer.zero_grad()
```

Mengubah dimensi label menjadi (`batch_size, 1`) dengan tipe data *float* sesuai dengan kebutuhan `BCEWithLogitsLoss`. Selain itu, gradien parameter juga dibersihkan sebelum komputasi agar gradien dari iterasi sebelumnya tidak tercampur dengan iterasi yang sedang berlangsung.

```
with torch.set_grad_enabled(phase == 'train'):
    outputs = model(inputs)
    loss = criterion(outputs, labels)

    if phase == 'train':
        loss.backward()
        optimizer.step()
```

Proses gradien ini hanya akan dihitung jika `phase == 'train'` yang mana berarti model sedang dalam fase pelatihan. Hanya dalam fase pelatihan menghitung gradien parameter berdasarkan *loss* dan memperbarui bobot model menggunakan gradien.

```
probs = torch.sigmoid(outputs)
preds = (probs > 0.5).long()

running_loss += loss.item() * inputs.size(0)
running_corrects += torch.sum(preds == labels.data)
```

Mengonversi logit menjadi probabilitas dengan prediksi biner 0 atau 1. Mengakumulasi *loss* total dari semua *batch* juga menghitung jumlah prediksi yang benar untuk akurasi.

```
epoch_loss = running_loss / len(dataloaders[phase].dataset)
epoch_acc = running_corrects.double() /
len(dataloaders[phase].dataset)

print(f'{phase.capitalize()} Loss: {epoch_loss:.4f} Acc:
{epoch_acc:.4f}')
```

Menghitung *loss* rata-rata per sampel dengan membagi *running_loss* dengan jumlah total sampel. Sedangkan menghitung akurasi dengan membagi jumlah prediksi yang benar dengan jumlah total sampel.

```
if phase == 'val' and epoch_acc > best_acc:
    best_acc = epoch_acc
    best_model_wts = deepcopy(model.state_dict())
```

Jika akurasi validasi pada *epoch* yang sedang berjalan lebih baik dibandingkan dengan sebelumnya, maka bobot model tersebut akan disimpan sebagai bobot terbaik.

```
model.load_state_dict(best_model_wts)
print(f'Best Validation Accuracy: {best_acc:.4f}')
return model
```

Menampilkan bobot terbaik setelah pelatihan selesai dengan mencetak nilai akurasi.

g. Evaluasi model

```
def evaluate_model(model, dataloader, criterion, device,
evaluate_metrics=True):
    model.eval()
    total_loss = 0.0
    all_preds, all_labels = [], []

    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)

            # Forward pass
            outputs = model(inputs).squeeze()
            loss = criterion(outputs, labels.float())
            total_loss += loss.item()

            # Konversi output ke prediksi biner
            preds = torch.sigmoid(outputs).detach().cpu().numpy()
> 0.5
            all_preds.extend(preds)
            all_labels.extend(labels.cpu().numpy())

    avg_loss = total_loss / len(dataloader)
    all_preds, all_labels = np.array(all_preds),
np.array(all_labels)
    accuracy = (all_preds == all_labels).mean() * 100

    if evaluate_metrics:
        # Confusion Matrix dan Classification Report
        cm = confusion_matrix(all_labels, all_preds)
        print("Confusion Matrix:")
        print(cm)
        print(classification_report(all_labels, all_preds,
target_names=["Smoking", "Not Smoking"]))

    return avg_loss, accuracy
```

Adapun penjelasan detail dari masing-masing *syntax* tersebut adalah sebagai berikut.

```
model.eval()
total_loss = 0.0
all_preds, all_labels = [], []
```

Mengatur model pada fase evaluasi. `Total_loss` dihitung dari semua batch dalam dataloader. `All_preds` digunakan untuk menyimpan prediksi model dan `all_labels` digunakan untuk menyimpan label aktual dari dataset.

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

Memindahkan data pada GPU.

```
outputs = model(inputs).squeeze()
loss = criterion(outputs, labels.float())
total_loss += loss.item()
```

Menghasilkan *output* berupa prediksi. Menghitung *loss* dari *criterion* yang sudah didefinisikan sebelumnya.

```
preds = torch.sigmoid(outputs).detach().cpu().numpy() > 0.5
```

Mengubah logit menjadi probabilitas dengan nilai probabilitas > 0,5 dianggap positif (1) dan nilai probabilitas < 0,5 dianggap negatif (0).

```
all_preds.extend(preds)
all_labels.extend(labels.cpu().numpy())
```

Menyimpan prediksi dan label.

```
avg_loss = total_loss / len(dataloader)
all_preds, all_labels = np.array(all_preds), np.array(all_labels)
accuracy = (all_preds == all_labels).mean() * 100
```

Menghitung rata-rata *loss* dan akurasi.

```
cm = confusion_matrix(all_labels, all_preds)
```

Menambahkan *confusion matrix* sebagai evaluasi metrik menggunakan data pengujian.

```
print(classification_report(all_labels, all_preds,
target_names=["Smoking", "Not Smoking"]))
```

Menampilkan *classification* report.

h. Menjalankan training dan evaluasi model

```
model_data = {}
for model in models_list:
    model = model.to(device)
    # optimizer menggunakan SGD
    optimizer = optim.SGD(model.parameters(), lr=0.001,
momentum=0.9)
    criterion = nn.BCEWithLogitsLoss()

    print(f"\nTraining model: {model.__class__.__name__}")

    # Train model
    model, train_losses, val_losses, train_accuracies,
val_accuracies = train_model(
        model, dataloaders, criterion, optimizer, num_epochs=10,
device=device
    )

    # Evaluate on test set
    print(f"Evaluating model: {model.__class__.__name__}")
    test_loss, test_acc = evaluate_model(model,
dataloaders['test'], criterion, device)

    # Simpan hasil
    model_data[model.__class__.__name__] = {
        'train_losses': train_losses,
        'val_losses': val_losses,
        'train_accuracies': train_accuracies,
        'val_accuracies': val_accuracies,
        'test_loss': test_loss,
        'test_accuracy': test_acc
    }
```

Output:

```
Training model: ResNet
Epoch 1/10
Train Phase:  45%|████      | 155/345 [01:50<01:16,  2.47it/s,
loss=0.475]/usr/local/lib/python3.10/dist-packages/PIL/Image.py:1
054: UserWarning: Palette images with Transparency expressed in
bytes should be converted to RGBA images
  warnings.warn(
Train Loss: 0.4835 Acc: 0.8015
Val Loss: 0.2846 Acc: 0.9144
Epoch 2/10
Train Loss: 0.2525 Acc: 0.9028
Val Loss: 0.1868 Acc: 0.9405
Epoch 3/10
Train Loss: 0.1630 Acc: 0.9395
Val Loss: 0.1865 Acc: 0.9347
Epoch 4/10
Train Loss: 0.0985 Acc: 0.9660
Val Loss: 0.1389 Acc: 0.9492
Epoch 5/10
```

```
Train Loss: 0.0599 Acc: 0.9820
Val Loss: 0.1385 Acc: 0.9390
Epoch 6/10
Train Loss: 0.0421 Acc: 0.9895
Val Loss: 0.1169 Acc: 0.9550
Epoch 7/10
Train Loss: 0.0264 Acc: 0.9936
Val Loss: 0.1333 Acc: 0.9507
Epoch 8/10
Train Loss: 0.0211 Acc: 0.9953
Val Loss: 0.1201 Acc: 0.9608
Epoch 9/10
Train Loss: 0.0201 Acc: 0.9951
Val Loss: 0.1629 Acc: 0.9434
Epoch 10/10
Train Loss: 0.0135 Acc: 0.9962
Val Loss: 0.1588 Acc: 0.9463
Best Validation Accuracy: 0.9608
Evaluating model: ResNet
Confusion Matrix:
[[336  16]
 [ 28 311]]
              precision    recall  f1-score   support

     Smoking       0.92      0.95      0.94       352
 Not Smoking       0.95      0.92      0.93       339

    accuracy                           0.94       691
   macro avg       0.94      0.94      0.94       691
weighted avg       0.94      0.94      0.94       691


Training model: AlexNet
Epoch 1/10
Train Phase:  61%|██████    | 210/345 [01:22<00:51,  2.60it/s,
loss=0.54]/usr/local/lib/python3.10/dist-packages/PIL/Image.py:10
54: UserWarning: Palette images with Transparency expressed in
bytes should be converted to RGBA images
 warnings.warn(
Train Loss: 0.4712 Acc: 0.7815
Val Loss: 0.3231 Acc: 0.8810
Epoch 2/10
Train Loss: 0.3205 Acc: 0.8656
Val Loss: 0.3021 Acc: 0.8853
Epoch 3/10
Train Loss: 0.2271 Acc: 0.9064
Val Loss: 0.2640 Acc: 0.8955
Epoch 4/10
Train Loss: 0.1510 Acc: 0.9435
Val Loss: 0.2508 Acc: 0.9057
Epoch 5/10
Train Loss: 0.0941 Acc: 0.9648
Val Loss: 0.2318 Acc: 0.9245
Epoch 6/10
Train Loss: 0.0719 Acc: 0.9711
```

```
Val Loss: 0.2327 Acc: 0.9303
Epoch 7/10
Train Loss: 0.0579 Acc: 0.9787
Val Loss: 0.2888 Acc: 0.9129
Epoch 8/10
Train Loss: 0.1073 Acc: 0.9573
Val Loss: 0.2154 Acc: 0.9289
Epoch 9/10
Train Loss: 0.0480 Acc: 0.9824
Val Loss: 0.2396 Acc: 0.9318
Epoch 10/10
Train Loss: 0.0372 Acc: 0.9875
Val Loss: 0.2590 Acc: 0.9260
Best Validation Accuracy: 0.9318
Evaluating model: AlexNet
Confusion Matrix:
[[317  35]
 [ 25 314]]
              precision    recall  f1-score   support

     Smoking       0.93      0.90      0.91       352
 Not Smoking       0.90      0.93      0.91       339

    accuracy                           0.91       691
   macro avg       0.91      0.91      0.91       691
weighted avg       0.91      0.91      0.91       691


Training model: MobileNetV2
Epoch 1/10
Train Phase:  92%|█████████ | 318/345 [02:26<00:13,  2.07it/s,
loss=0.573]/usr/local/lib/python3.10/dist-packages/PIL/Image.py:1
054: UserWarning: Palette images with Transparency expressed in
bytes should be converted to RGBA images
  warnings.warn(
Train Loss: 0.5498 Acc: 0.7417
Val Loss: 0.4182 Acc: 0.8374
Epoch 2/10
Train Loss: 0.3962 Acc: 0.8283
Val Loss: 0.3387 Acc: 0.8737
Epoch 3/10
Train Loss: 0.3099 Acc: 0.8732
Val Loss: 0.2540 Acc: 0.9071
Epoch 4/10
Train Loss: 0.2506 Acc: 0.9034
Val Loss: 0.2189 Acc: 0.9216
Epoch 5/10
Train Loss: 0.2026 Acc: 0.9221
Val Loss: 0.2320 Acc: 0.9071
Epoch 6/10
Train Loss: 0.1566 Acc: 0.9417
Val Loss: 0.1932 Acc: 0.9187
Epoch 7/10
Train Loss: 0.1125 Acc: 0.9602
Val Loss: 0.1794 Acc: 0.9347
Epoch 8/10
```

```
Train Loss: 0.1133 Acc: 0.9599
Val Loss: 0.1989 Acc: 0.9318
Epoch 9/10
Train Loss: 0.0851 Acc: 0.9706
Val Loss: 0.1721 Acc: 0.9390
Epoch 10/10
Train Loss: 0.0619 Acc: 0.9804
Val Loss: 0.1861 Acc: 0.9318
Best Validation Accuracy: 0.9390
Evaluating model: MobileNetV2
Confusion Matrix:
[[337  15]
 [ 40 299]]
              precision    recall  f1-score   support

     Smoking       0.89      0.96      0.92       352
 Not Smoking       0.95      0.88      0.92       339

    accuracy                           0.92       691
   macro avg       0.92      0.92      0.92       691
weighted avg       0.92      0.92      0.92       691
```

Adapun penjelasan detail dari masing-masing *syntax* tersebut adalah sebagai berikut.

```
model_data = {}
```

Menyimpan hasil training dan evaluasi dari masing-masing model.

```
for model in models_list:
    model = model.to(device)
```

Memindahkan masing-masing model ke GPU.

```
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
criterion = nn.BCEWithLogitsLoss()
```

Menggunakan *Stochastic Gradient Descent* (SGD) sebagai metode optimasi dengan `learning rate = 0,001` dan momentum 0,9 selama proses optimasi. Selain itu, menggunakan *Binary Cross Entropy with Logits* sebagai fungsi *loss*.

```
model, train_losses, val_losses, train_accuracies, val_accuracies
= train_model(
    model, dataloaders, criterion, optimizer, num_epochs=10,
device=device
)
```

Melakukan training dengan 10 *epoch*.

```
test_loss, test_acc = evaluate_model(model, dataloaders['test'],
criterion, device)
```

Mengevaluasi model dengan menghitung `test_lost` dan `test_acc`.

```
model_data[model.__class__.__name__] = {
    'train_losses': train_losses,
    'val_losses': val_losses,
    'train_accuracies': train_accuracies,
    'val_accuracies': val_accuracies,
    'test_loss': test_loss,
    'test_accuracy': test_acc
}
```

Menyimpan hasil pelatihan yang terdiri dari proses *epoch, train loss, train acc, val loss, val acc*, *test loss,* dan *test acc*.

2.8 Evaluasi dari Smoking Detector

a. Memuat model yang sudah dilatih

```
print("Memuat model...")
model = load_model("smoking.model.h5")
print("Model berhasil dimuat.")
```

Memanggil model smoking.model.h5 yang sudah di *training*.

b. Memproses gambar dalam batch

```
batch_size = 16

# Fungsi untuk memproses gambar dalam batch
def process_and_predict_batch(image_paths, labels):
    global correct_predictions, total_predictions

    # Membaca gambar
    images = []
    for img_path in image_paths:
        if not os.path.exists(img_path):
            print(f"File tidak ditemukan: {img_path}")
            continue

        # Memuat gambar
        image = cv2.imread(img_path)

        # Memeriksa apakah gambar berhasil dimuat
        if image is None:
            print(f"Error memuat gambar: {img_path}")
            continue

        # Mengubah ke RGB jika model menggunakan RGB
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
        image = cv2.resize(image, (32, 32))  # Sesuaikan dengan
input model
        image = image.astype("float32") / 255.0  # Normalisasi
pixel ke rentang [0, 1]
        images.append(img_to_array(image))

    if len(images) == 0:
        print("Tidak ada gambar yang dapat diproses dalam batch
ini.")
        return

    # Menambahkan dimensi batch
    images = np.array(images)

    # Melakukan prediksi
    predictions = model.predict(images)

    # Menentukan kelas prediksi (smoking atau not smoking)
    predicted_classes = np.argmax(predictions, axis=1)  # 0
untuk smoking, 1 untuk not smoking

    # Membandingkan prediksi dengan label yang sebenarnya
    correct_predictions += np.sum(predicted_classes ==
labels[:len(predicted_classes)])  # Membatasi panjang labels
    total_predictions += len(predicted_classes)

   # Menyimpan semua prediksi dan label untuk evaluasi
    all_preds.extend(predicted_classes)
    all_labels.extend(labels[:len(predicted_classes)])
```

Menentukan banyaknya *batch* atau banyaknya gambar yang diproses dalam setiap iterasi, yaitu sebesar 12 gambar. Proses dimulai dengan membaca data gambar yang sudah didefinisikan pada proses awal. Gambar dimuat menggunakan OpenCV, diubah ke dalam format RGB, menyesuaikan ukuran menjadi 32x32, dna menormalisasikan *pixel*. Gambar akan dilakukan prediksi dengan model yang sudah ada. Mengevaluasi proses prediksi dengan menghitung prediksi yang benar dan total prediksi. Menyimpan semua prediksi dan label untuk di evaluasi.

c. Mengambil masing-masing gambar secara acak dari data smoking dan not smoking

```
smoking_images = os.listdir(smoking_path)
notsmoking_images = os.listdir(notsmoking_path)
```

Mengambil gambar smoking dan not smoking secara acak dari data gambar.

d. Menyusun gambar dan label dalam batch

```
image_paths = []
```

```
labels = []
```

Membuat list kosong untuk menyimpan gambar dan label yang akan di proses dalam *batch*

e. Menambahkan gambar dari folder smoking dan not smoking

```
for img_file in smoking_sample:
    img_path = os.path.join(smoking_path, img_file)
    image_paths.append(img_path)
    labels.append(0)  # 0 untuk smoking

for img_file in notsmoking_sample:
    img_path = os.path.join(notsmoking_path, img_file)
    image_paths.append(img_path)
    labels.append(1)  # 1 untuk not smoking
```

Menambahkan gambar *smoking* dan *not smoking* pada `image_path`. Menambahkan label 0 untuk *smoking* dan 1 untuk *not smoking*.

f. Mengonversi label

```
labels = np.array(labels)
```

Mengubah label menjadi *array numpy*.

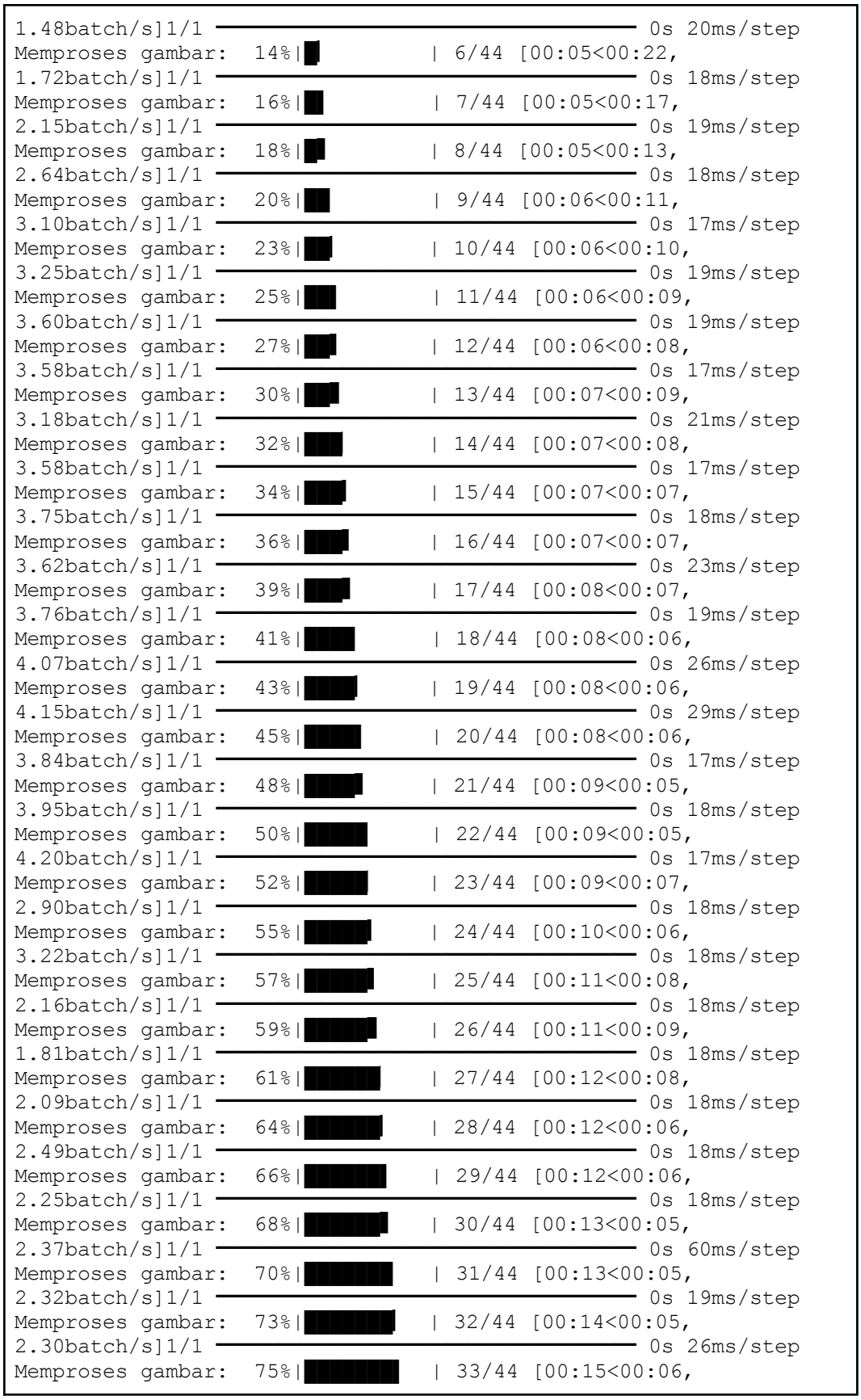
g. Menambahkan tqdm untuk membuat progress bar

```
correct_predictions = 0
total_predictions = 0
all_preds = []  # Menyimpan semua prediksi
all_labels = []  # Menyimpan semua label sebenarnya

for i in tqdm(range(0, len(image_paths), batch_size),
desc="Memproses gambar", unit="batch"):
    batch_image_paths = image_paths[i:i + batch_size]
    batch_labels = labels[i:i + batch_size]
    process_and_predict_batch(batch_image_paths, batch_labels)
```

Output:

```
Memproses gambar:   0%|          | 0/44 [00:00<?, ?batch/s]1/1
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
Memproses gambar:   2%|▏         | 1/44 [00:03<02:34,
3.59s/batch]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
Memproses gambar:   5%|▍         | 2/44 [00:03<01:08,
1.63s/batch]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
Memproses gambar:   7%|▋         | 3/44 [00:04<00:43,
1.05s/batch]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
Memproses gambar:   9%|▉         | 4/44 [00:04<00:30,
1.30batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Memproses gambar:  11%|█         | 5/44 [00:05<00:26,
```


1.48batch/s]1/1 ━━ 0s 20ms/step
Memproses gambar: 14%| | 6/44 [00:05<00:22,
1.72batch/s]1/1 ━━ 0s 18ms/step
Memproses gambar: 16%| | 7/44 [00:05<00:17,
2.15batch/s]1/1 ━━ 0s 19ms/step
Memproses gambar: 18%| | 8/44 [00:05<00:13,
2.64batch/s]1/1 ━━ 0s 18ms/step
Memproses gambar: 20%| | 9/44 [00:06<00:11,
3.10batch/s]1/1 ━━ 0s 17ms/step
Memproses gambar: 23%| | 10/44 [00:06<00:10,
3.25batch/s]1/1 ━━ 0s 19ms/step
Memproses gambar: 25%| | 11/44 [00:06<00:09,
3.60batch/s]1/1 ━━ 0s 19ms/step
Memproses gambar: 27%| | 12/44 [00:06<00:08,
3.58batch/s]1/1 ━━ 0s 17ms/step
Memproses gambar: 30%| | 13/44 [00:07<00:09,
3.18batch/s]1/1 ━━ 0s 21ms/step
Memproses gambar: 32%| | 14/44 [00:07<00:08,
3.58batch/s]1/1 ━━ 0s 17ms/step
Memproses gambar: 34%| | 15/44 [00:07<00:07,
3.75batch/s]1/1 ━━ 0s 18ms/step
Memproses gambar: 36%| | 16/44 [00:07<00:07,
3.62batch/s]1/1 ━━ 0s 23ms/step
Memproses gambar: 39%| | 17/44 [00:08<00:07,
3.76batch/s]1/1 ━━ 0s 19ms/step
Memproses gambar: 41%| | 18/44 [00:08<00:06,
4.07batch/s]1/1 ━━ 0s 26ms/step
Memproses gambar: 43%| | 19/44 [00:08<00:06,
4.15batch/s]1/1 ━━ 0s 29ms/step
Memproses gambar: 45%| | 20/44 [00:08<00:06,
3.84batch/s]1/1 ━━ 0s 17ms/step
Memproses gambar: 48%| | 21/44 [00:09<00:05,
3.95batch/s]1/1 ━━ 0s 18ms/step
Memproses gambar: 50%| | 22/44 [00:09<00:05,
4.20batch/s]1/1 ━━ 0s 17ms/step
Memproses gambar: 52%| | 23/44 [00:09<00:07,
2.90batch/s]1/1 ━━ 0s 18ms/step
Memproses gambar: 55%| | 24/44 [00:10<00:06,
3.22batch/s]1/1 ━━ 0s 18ms/step
Memproses gambar: 57%| | 25/44 [00:11<00:08,
2.16batch/s]1/1 ━━ 0s 18ms/step
Memproses gambar: 59%| | 26/44 [00:11<00:09,
1.81batch/s]1/1 ━━ 0s 18ms/step
Memproses gambar: 61%| | 27/44 [00:12<00:08,
2.09batch/s]1/1 ━━ 0s 18ms/step
Memproses gambar: 64%| | 28/44 [00:12<00:06,
2.49batch/s]1/1 ━━ 0s 18ms/step
Memproses gambar: 66%| | 29/44 [00:12<00:06,
2.25batch/s]1/1 ━━ 0s 18ms/step
Memproses gambar: 68%| | 30/44 [00:13<00:05,
2.37batch/s]1/1 ━━ 0s 60ms/step
Memproses gambar: 70%| | 31/44 [00:13<00:05,
2.32batch/s]1/1 ━━ 0s 19ms/step
Memproses gambar: 73%| | 32/44 [00:14<00:05,
2.30batch/s]1/1 ━━ 0s 26ms/step
Memproses gambar: 75%| | 33/44 [00:15<00:06,

```
1.61batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
Memproses gambar:  77%|███████▋   | 34/44 [00:15<00:06,
1.67batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
Memproses gambar:  80%|███████▉   | 35/44 [00:16<00:04,
1.98batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
Memproses gambar:  82%|████████   | 36/44 [00:16<00:03,
2.28batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
Memproses gambar:  84%|████████▎  | 37/44 [00:16<00:03,
2.02batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
Memproses gambar:  86%|████████▌  | 38/44 [00:17<00:02,
2.21batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
Memproses gambar:  89%|████████▊  | 39/44 [00:17<00:02,
1.90batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
Memproses gambar:  91%|█████████  | 40/44 [00:18<00:02,
1.69batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Memproses gambar:  93%|█████████▎ | 41/44 [00:20<00:02,
1.21batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Memproses gambar:  95%|█████████▌ | 42/44 [00:20<00:01,
1.44batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Memproses gambar:  98%|█████████▊ | 43/44 [00:20<00:00,
1.73batch/s]1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 816ms/step
Memproses gambar: 100%|██████████| 44/44 [00:21<00:00,
2.03batch/s]
```

Menghitung banyak prediksi benar dan total prediksi juga menampilkan progress bar untuk memantau progress pengolahan data.

h. Menghitung akurasi, confusion matrix, dan classification report

```
accuracy = (correct_predictions / total_predictions) * 100
print(f"Akurasi: {accuracy:.2f}%")

cm = confusion_matrix(all_labels, all_preds)
print("Confusion Matrix:")
print(cm)

report = classification_report(all_labels, all_preds,
target_names=["Smoking", "Not Smoking"])
print("Classification Report:")
print(report)
```

Output:

```
Akurasi: 52.10%
Confusion Matrix:
[[ 71 268]
 [ 63 289]]
Classification Report:
              precision    recall  f1-score   support

     Smoking       0.53      0.21      0.30       339
 Not Smoking       0.52      0.82      0.64       352
```

```
    accuracy                           0.52       691
   macro avg       0.52      0.52      0.47       691
weighted avg       0.52      0.52      0.47       691
```

Menghitung nilai akurasi dengan membagi jumlah prediksi benar dengan total prediksi keseluruhan dan mengalikan dengan 100 agar menghasilkan akurasi berupa persen dengan menampilkan 2 angka desimal. Menghitung *confusion matrix* dan juga *classification report* yang terdiri dari *precision, recall, f1-score*, dan *support*.

2.9 Graphical User Interface (GUI)

```
# -- coding: utf-8 --
import torch
import torchvision.transforms as transforms
from torch.autograd import Variable
import numpy as np
import argparse
import imutils
import time
import cv2
import os
from imutils.video import VideoStream
from tkinter import Tk, Button, filedialog
from torchvision.models import resnet50  # Menggunakan ResNet50
import torch.nn as nn

# Argument parser untuk filter deteksi wajah
ap = argparse.ArgumentParser()
ap.add_argument("-c", "--face_confidence", type=float, default=0.5,
                help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# Memuat detektor wajah
print("[INFO] Loading face detector...")
protoPath = os.path.sep.join(["face_detector", "deploy.prototxt"])
modelPath = os.path.sep.join(["face_detector",
"res10_300x300_ssd_iter_140000.caffemodel"])
net = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

# Informasi awal
print("[INFO] Loading smoking detector...")

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

# Muat arsitektur model
model = resnet50(pretrained=False)
model.fc = nn.Linear(2048, 1)  # Output layer: 1 neuron untuk binary
classification

# Muat state dict
state_dict = torch.load("detectorsmoking3.pth", map_location=device)
```

```
state_dict.pop('fc.weight', None)  # Hapus parameter fc jika ada konflik
state_dict.pop('fc.bias', None)

# Load state dict ke model
model.load_state_dict(state_dict, strict=False)
model = model.to(device)
model.eval()

print("[INFO] Model berhasil dimuat dan siap untuk evaluasi.")

# Preprocessing pipeline untuk gambar
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225]),
])

# Fungsi untuk memproses video (kamera atau file video)
def process_video(video_source):
    print("[INFO] Starting video stream...")
    if isinstance(video_source, str):  # File video
        vs = cv2.VideoCapture(video_source)
    else:  # Kamera
        vs = VideoStream(src=0).start()
        time.sleep(2.0)

    while True:
        # Ambil frame dari video stream
        frame = vs.read() if isinstance(video_source, int) else
vs.read()[1]
        if frame is None:
            break
        frame = imutils.resize(frame, width=600)

        # Ambil dimensi frame dan konversikan ke blob untuk deteksi
wajah
        (h, w) = frame.shape[:2]
        blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,
                                     (300, 300), (104.0, 177.0, 123.0))

        # Proses blob melalui network untuk mendapatkan deteksi wajah
        net.setInput(blob)
        detections = net.forward()

        # Loop melalui deteksi wajah
        for i in range(0, detections.shape[2]):
            # Ambil confidence dari prediksi
            confidence = detections[0, 0, i, 2]

            # Filter deteksi yang lemah
            if confidence > args["face_confidence"]:
                # Ambil koordinat (x, y) untuk bounding box wajah
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")
```

```
                # Pastikan bounding box tetap berada dalam dimensi frame
                startX = max(0, startX)
                startY = max(0, startY)
                endX = min(w, endX)
                endY = min(h, endY)

                # Ekstrak wajah dari ROI dan preprocessing
                face = frame[startY:endY, startX:endX]
                face = transform(face)
                face = face.unsqueeze(0)  # Menambahkan dimensi batch
                face = face.to(device)

                # Pass wajah melalui model deteksi merokok
                with torch.no_grad():
                    output = model(face)
                    predicted = torch.sigmoid(output).item()  # Konversi
ke probabilitas
                    label = "Smoking" if predicted > 0.55 else "Not
Smoking"

                # Gambar label dan bounding box pada frame
                cv2.putText(frame, label, (startX, startY - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255),
2)
                cv2.rectangle(frame, (startX, startY), (endX, endY),
                              (0, 0, 255), 2)

        # Menampilkan frame secara realtime
        cv2.imshow("SMOKING_DETECTOR", frame)

        # Jika tombol 'q' ditekan, keluar dari loop
        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
            break

    # Menutup semua window dan menghentikan video stream
    if isinstance(video_source, str):
        vs.release()
    else:
        vs.stop()
    cv2.destroyAllWindows()

# Fungsi untuk membuka kamera
def open_camera():
    process_video(0)

# Fungsi untuk memilih file video
def upload_video():
    file_path = filedialog.askopenfilename(filetypes=[("Video Files",
".mp4;.avi;*.mov")])
    if file_path:
        process_video(file_path)

# Fungsi untuk memilih file gambar
def upload_image():
```

```
    file_path = filedialog.askopenfilename(filetypes=[("Image Files",
".jpg;.jpeg;.png;.bmp;*.tiff")])
    if file_path:
        # Membaca gambar yang diunggah
        image = cv2.imread(file_path)
        image = imutils.resize(image, width=600)

        # Ambil dimensi frame dan konversikan ke blob untuk deteksi
wajah
        (h, w) = image.shape[:2]
        blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0,
                                     (300, 300), (104.0, 177.0, 123.0))

        # Proses blob melalui network untuk mendapatkan deteksi wajah
        net.setInput(blob)
        detections = net.forward()

        # Loop melalui deteksi wajah
        for i in range(0, detections.shape[2]):
            # Ambil confidence dari prediksi
            confidence = detections[0, 0, i, 2]

            if confidence > args["face_confidence"]:
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")
                startX = max(0, startX)
                startY = max(0, startY)
                endX = min(w, endX)
                endY = min(h, endY)

                # Ekstrak wajah dari ROI dan prediksi
                face = image[startY:endY, startX:endX]
                face = transform(face).unsqueeze(0).to(device)

                with torch.no_grad():
                    output = model(face)
                    predicted = torch.sigmoid(output).item()
                    label = "Smoking" if predicted > 0.5 else "Not
Smoking"

                cv2.putText(image, label, (startX, startY - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255),
2)
                cv2.rectangle(image, (startX, startY), (endX, endY), (0,
0, 255), 2)

        cv2.imshow("Smoking Detection", image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

# Membuat antarmuka pengguna dengan tkinter
root = Tk()
root.title("Smoking Detector")
root.geometry("300x200")

btn_camera = Button(root, text="Open Camera", command=open_camera,
```

```
width=20, height=2)
btn_camera.pack(pady=10)

btn_upload = Button(root, text="Upload Video", command=upload_video,
width=20, height=2)
btn_upload.pack(pady=10)

btn_upload_image = Button(root, text="Upload Image",
command=upload_image, width=20, height=2)
btn_upload_image.pack(pady=10)

root.mainloop()
```

Adapun penjelasan detail dari masing-masing *syntax* adalah sebagai berikut

```
ap = argparse.ArgumentParser()
ap.add_argument("-c", "--face_confidence", type=float, default=0.5,
help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

Pembuatan argumen parser yang berfungsi untuk memberikan nilai ambang batas melalui baris perintah yang akan mengatur seberapa percaya detektor wajah terhadap prediksi deteksi wajah.

```
print("[INFO] loading face detector...")
protoPath = os.path.sep.join(["face_detector", "deploy.prototxt"])
modelPath = os.path.sep.join(["face_detector",
"res10_300x300_ssd_iter_140000.caffemodel"])
net = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
```

Memuat file yang digunakan untuk mendeteksi wajah pada gambar/video. Dengan bantuan file tersebut, sistem akan mendeteksi wajah dan akan dilanjutkan dengan model klasifikasi yang telah di *training* sebelumnya.

```
print("[INFO] loading smoking detector...")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = resnet50(pretrained=False)
model.fc = nn.Linear(2048, 1)
state_dict = torch.load("detectorsmoking_edisi2.pth",
map_location=device)
state_dict.pop('fc.weight', None)
model.load_state_dict(state_dict, strict=False)
model = model.to(device)
model.eval()
print("[INFO] Model berhasil dimuat dan siap untuk evaluasi.")
```

Dimulai dengan mendeteksi apakah terdapat GPU untuk mempercepat proses deteksi perokok. Memanggil model yang menjadi model terbaik dari beberapa percobaan model. Karena model sudah tersedia, maka `pretrained = False`. Model ini nantinya digunakan untuk mendeteksi dengan terdapat 2 label, yaitu *smoking* dan *non smoking*.

Memuat data model hasil *training* yang digunakan untuk mendeteksi perokok. Dilanjut dengan menghapus beberapa informasi yang tidak lagi dibutuhkan. Mengisi model dengan data model yang sudah dilatih sebelumnya agar dapat mendeteksi perokok. Untuk mempercepat proses deteksi, model dijalankan pada GPU yang sudah di cek pada tahap awal.

```
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]),
])
```

Transformasi data dengan mengubah ukuran menjadi 224x224, mengubah menjadi bentuk tensor, dan dilakukan normalisasi data gambar dengan nilai rata-rata dan *standar deviasi* dari masing-masing *channel* RGB (*Red, Green, Blue*).

```
def process_video(video_source):
    print("[INFO] starting video stream...")
    if isinstance(video_source, str):  # File video
        vs = cv2.VideoCapture(video_source)
    else:  # Kamera
        vs = VideoStream(src=0).start()
        time.sleep(2.0)
```

Menggunakan `VideoCapture` untuk memproses video yang akan dideteksi atau `VideoStream` jika ingin memproses secara langsung melalui tangkapan kamera.

```
# Fungsi untuk membuka kamera
def open_camera():
    process_video(0)

# Fungsi untuk memilih file video
def upload_video():
    file_path = filedialog.askopenfilename(filetypes=[("Video Files",
".mp4;.avi;*.mov")])
    if file_path:
        process_video(file_path)

# Fungsi untuk memilih file gambar
def upload_image():
    file_path = filedialog.askopenfilename(filetypes=[("Image Files",
".jpg;.jpeg;.png;.bmp;*.tiff")])
    if file_path:
        # Membaca gambar yang diunggah
        image = cv2.imread(file_path)
        image = imutils.resize(image, width=600)
```

Mendefinisikan fungsi untuk membuka kamera, mengunggah gambar, dan mengunggah video.

```
blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0, (300,
300), (104.0, 177.0, 123.0))
net.setInput(blob)
detections = net.forward()
```

Mengubah gambar yang ditangkap melalui video ataupun kamera menjadi `blob`. Deteksi wajah akan dilakukan melalui model deteksi wajah yang sudah dimuat sebelumnya.

```
for i in range(0, detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if confidence > args["face_confidence"]:
        # Ekstrak wajah dan lakukan preprocessing
        face = frame[startY:endY, startX:endX]
        face = transform(face)
        face = face.unsqueeze(0)  # Menambahkan dimensi batch
        face = face.to(device)

        # Lakukan prediksi
        with torch.no_grad():
            output = model(face)
            _, predicted = torch.max(output, 1)
            label = le.classes_[predicted.item()]

        # Gambar bounding box dan label pada frame
        cv2.putText(frame, label, (startX, startY - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), (0, 0,
255), 2)
```

Untuk setiap wajah yang terdeteksi menggunakan model pertama (deteksi wajah), akan diekstrak dan diteruskan ke deteksi menggunakan model kedua (deteksi perokok). Hasil prediksi akan ditampilkan dengan *bounding box* sesuai dengan label hasil prediksi, yaitu *smoking* dan *no smoking*.

```
root = Tk()
root.title("Smoking Detector")
root.geometry("300x150")

btn_camera = Button(root, text="Open Camera", command=open_camera,
width=20, height=2)
btn_camera.pack(pady=10)

btn_upload = Button(root, text="Upload Video", command=upload_video,
width=20, height=2)
btn_upload.pack(pady=10)
```

```
btn_upload_image = Button(root, text="Upload Image",
command=upload_image, width=20, height=2)
btn_upload_image.pack(pady=10)

root.mainloop()
```

Membuat antar muka dengan *Tkinter* untuk menampilkan opsi *open camera*, *upload video*, atau *upload gambar*.

BAB III
HASIL DAN PEMBAHASAN

3.1 Pengujian Model

Pada penelitian yang dilakukan menggunakan metode CNN untuk klasifikasi deteksi perokok, dilakukan pengujian dengan menggunakan metode CNN dasar yaitu AlexNet. MobileNetV2, dan ResNet50, dan juga *Smoking Method* sebagai pembanding tambahan. Evaluasi model dilakukan menggunakan *PyTorch* yang kemudian dibandingkan untuk menentukan model terbaik untuk penelitian ini. Hasil pelatihan dan validasi yang dilakukan pada masing-masing arsitektur juga perlu diperhatikan untuk menentukan model terbaik.

3.1.1 CNN Arsitektur AlexNet

Pada pengujian model yang menggunakan metode CNN arsitektur AlexNet didapatkan informasi berdasarkan hasil pelatihan yang dilakukan pada Tabel 3.1 Hasil Pelatihan dan Validasi untuk Model CNN AlexNet.

Tabel 3.1 Hasil Pelatihan dan Validasi untuk Model CNN AlexNet

Epoch	Accuracy	Loss	Val_Accuracy	Val_Loss
1	79.01%	0.4533	84.01%	0.3555
2	87.33%	0.3069	85.03%	0.3742
3	91.16%	0.2188	89.97%	0.2671
4	94.37%	0.1455	90.70%	0.2511
5	96.17%	0.1087	91.13%	0.2559
6	97.71%	0.0645	91.57%	0.2438
7	97.71%	0.0637	90.99%	0.3059
8	99.04%	0.0336	91.42%	0.3186
9	98.75%	0.0419	92.15%	0.2836
10	98.53%	0.0392	90.84%	0.3707

Berdasarkan Tabel 3.1 bahwa hasil pelatihan model CNN dengan arsitektur AlexNet menunjukkan peningkatan performa signifikan pada awal *epoch*, tetapi mulai *overfitting* setelah *epoch* ke-6. Pada *epoch* 1, akurasi pelatihan mencapai 79.01% dengan *train loss* 0.4533, sementara akurasi validasi sebesar 84.01% dengan *validation loss* 0.3555. Performanya terus

meningkat hingga *epoch* 6, dengan *train loss* 0.0645, *validation loss* 0.2438, dan akurasi validasi terbaik 91.57%. Namun, setelah itu *validation loss* mulai meningkat meski *train loss* tetap rendah, menandakan *overfitting*. Sehingga performa terbaik tercapai pada *epoch* 9 dengan akurasi validasi 92.15%.

3.1.2 CNN Arsitektur MobileNetV2

Pada pengujian model yang menggunakan metode CNN arsitektur MobileNetV2 didapatkan informasi berdasarkan pelatihan yang dilakukan, dapat ditemukan pada Tabel 3.2 Hasil Pelatihan dan Validasi untuk Model CNN MobileNetV2.

Tabel 3.2 Hasil Pelatihan dan Validasi untuk Model CNN MobileNetV2

Epoch	Accuracy	Loss	Val_Accuracy	Val_Loss
1	81.25%	0.4041	0.2733	90.41%
2	90.18%	0.2374	91.57%	0.2159
3	94.55%	0.1518	88.23%	0.3116
4	96.24%	0.1032	92.01%	0.2321
5	96.88%	0.0895	92.30%	0.2425
6	97.35%	0.0739	92.88%	0.2563
7	98.17%	0.0528	93.17%	0.2854
8	98.26%	0.0471	92.30%	0.3184
9	98.57%	0.0400	92.15%	0.3307
10	97.64%	0.0737	93.17%	0.2592

Berdasarkan Tabel 3.2 bahwa hasil pelatihan metode CNN arsitektur MobileNetV2 selama 10 *epoch* menunjukkan peningkatan performa yang signifikan pada awal pelatihan, diikuti dengan kecenderungan *overfitting* setelah *epoch* ke-6. Pada *epoch* pertama, model berhasil mencapai akurasi pelatihan sebesar 81.25% dan akurasi validasi 90.41%, menandakan model dapat mengenali pola dasar dengan baik. Akurasi terus meningkat hingga *epoch* ke-6, di mana *validation accuracy* mencapai 92.88% dengan *train accuracy* 97.35%, menunjukkan performa yang optimal pada data validasi. Puncak akurasi validasi terbaik mencapai pada *epoch* ke-7 yaitu mencapai *validation accuracy* terbaik sebesar 93.17%.

3.1.3. CNN Arsitektur ResNet50

Pada pengujian model yang menggunakan metode CNN arsitektur ResNet50 didapatkan informasi berdasarkan pelatihan yang dilakukan dapat ditemukan pada Tabel 3.3 Hasil Pelatihan dan Validasi untuk Model CNN ResNet50.

Tabel 3.3 Hasil Pelatihan dan Validasi untuk Model CNN ResNet50

Epoch	Accuracy	Loss	Val_Accuracy	Val_Loss
1	83.35%	0.3737	90.70%	0.2471
2	94.75%	0.1515	91.86%	0.2204
3	97.88%	0.0639	93.31%	0.2215
4	98.24%	0.0554	93.90%	0.2346
5	98.47%	0.0432	93.17%	0.2434
6	98.58%	0.0396	93.75%	0.2464
7	99.24%	0.0250	94.19%	0.2780
8	98.84%	0.0356	93.30%	0.2666
9	97.97%	0.0593	94.04%	0.2235
10	99.11%	0.0279	95.06%	0.2712

Berdasarkan Tabel 3.3 bahwa hasil pelatihan model CNN dengan arsitektur ResNet50 menunjukkan peningkatan performa yang konsisten selama 10 *epoch*. Pada *epoch* pertama, model mencatat *train accuracy* 83.35% dan *validation accuracy* 90.70%. Di *epoch* kedua, *train accuracy* meningkat menjadi 94.75%, diikuti dengan peningkatan *validation accuracy* menjadi 91.86%. Pada *epoch* ketiga, model mencapai 97.88% untuk *train accuracy* dan 93.31% untuk *validation accuracy*, menunjukkan kemampuan generalisasi yang baik. *Train accuracy* tetap di atas 98% di *epoch* keempat dan kelima, sementara *validation accuracy* stabil sekitar 93%-94%. Pada *epoch* ke-10, model mencapai *train accuracy* 99.11% dan *validation accuracy* terbaik 95.06%. Meskipun *train accuracy* sangat tinggi, fluktuasi *validation loss* tidak mengindikasikan *overfitting* signifikan. Secara keseluruhan, model menunjukkan kemampuan generalisasi yang kuat dengan performa optimal pada *epoch* ke-10.

3.1.4. Smoking Method

Smoking Method merupakan model yang sudah tersedia dalam bentuk model terlatih yang disimpan (*pre-trained* model) dan langsung diimplementasikan tanpa melalui tahapan *training* dan *validation* lagi.

3.2 Evaluasi Model

Setelah melakukan pengujian terhadap model dengan memperhatikan hasil dari pelatihan dan validasi pada setiap arsitektur CNN. Penelitian ini melakukan evaluasi lanjutan untuk mengukur kinerja model yang telah dilatih. Evaluasi ini dilakukan melalui dua metode yaitu *Confusion Matrisx* dan *Classification Report*.

3.2.1 Confusion Matrix

Pada *confusion matrix* yang dihasilkan dari pengujian model menunjukkan distribusi prediksi model terhadap kelas-kelas yang ada. *Matriks* ini memberikan gambaran visual mengenai seberapa banyak prediksi yang benar dalam hal ini *True Positive* dan *True Negative* serta kesalahan prediksi yaitu *False Positive* dan *False Negative*.

1. CNN Arsitektur AlexNet

   Pada tahap hasil evaluasi model CNN arsitektur AlexNet ini didapatkan informasi berdasarkan Confusion Matrix yang dapat dilihat pada Tabel 3.4 Hasil Confusion Matrix CNN AlexNet.

Tabel 3. 4 Hasil Evaluasi Model CNN AlexNet

```
[[317  35]
 [ 25 314]]
```

Berdasarkan Tabel 3.4 bahwa hasil evaluasi menggunakan *confusion matrix* pada model dengan arsitektur AlexNet. *Confusion matrix* di atas memiliki empat komponen utama yang merepresentasikan jumlah prediksi benar dan salah dalam dua kelas:

- *True Positive* (TP):

  Nilai 314 pada baris kedua dan kolom kedua menunjukkan jumlah sampel positif yang diprediksi benar oleh model (klasifikasi benar sebagai positif).
- *True Negative* (TN):

  Nilai 317 pada baris pertama dan kolom pertama menunjukkan jumlah sampel negatif yang diprediksi benar oleh model (klasifikasi benar sebagai negatif).
- *False Positive* (FP):

  Nilai 35 pada baris pertama dan kolom kedua menunjukkan jumlah sampel negatif yang salah diprediksi sebagai positif (prediksi salah).
- *False Negative* (FN):

  Nilai 25 pada baris kedua dan kolom pertama menunjukkan jumlah sampel positif yang salah diprediksi sebagai negatif (prediksi salah).

2. CNN Arsitektur MobileNetV2

Pada tahap hasil evaluasi model CNN arsitektur MobileNetV2 ini didapatkan informasi berdasarkan Confusion Matrix yang dapat dilihat pada Tabel 3.5 Hasil Confusion Matrix CNN MobileNetV2.

Tabel 3.5 Hasil Evaluasi Model CNN MobileNetV2

```
[[337 15]
[ 40 299]]
```

Berdasarkan Tabel 3.5 bahwa hasil evaluasi menggunakan *confusion matrix* pada model dengan arsitektur MobileNetV2. *Confusion matrix* di atas memiliki empat komponen utama yang merepresentasikan jumlah prediksi benar dan salah dalam dua kelas:

- *True Positive* (TP):
  Nilai 299 pada baris kedua dan kolom kedua menunjukkan jumlah sampel positif yang diprediksi benar oleh model (klasifikasi benar sebagai positif).
- *True Negative* (TN):
  Nilai 337 pada baris pertama dan kolom pertama menunjukkan jumlah sampel negatif yang diprediksi benar oleh model (klasifikasi benar sebagai negatif).
- *False Positive* (FP):
  Nilai 15 pada baris pertama dan kolom kedua menunjukkan jumlah sampel negatif yang salah diprediksi sebagai positif (prediksi salah).
- *False Negative* (FN):
  Nilai 50 pada baris kedua dan kolom pertama menunjukkan jumlah sampel positif yang salah diprediksi sebagai negatif (prediksi salah)

3. CNN Arsitektur ResNet50

Pada tahap hasil evaluasi model CNN arsitektur ResNet50 ini didapatkan informasi berdasarkan *Confusion Matrix* yang dapat dilihat pada Tabel 3.6 Hasil Confusion Matrix CNN ResNet50.

Tabel 3.6 Hasil Evaluasi Model CNN ResNet50

```
[[336 16]
[28 311]]
```

Berdasarkan Tabel 3.6 bahwa hasil evaluasi menggunakan confusion matrix pada model dengan arsitektur ResNet50. Confusion matrix di atas memiliki empat komponen utama yang merepresentasikan jumlah prediksi benar dan salah dalam dua kelas:

- *True Positive* (TP):

  Nilai 311 pada baris kedua dan kolom kedua menunjukkan jumlah sampel positif yang diprediksi benar oleh model (klasifikasi benar sebagai positif).
- *True Negative* (TN):

  Nilai 336 pada baris pertama dan kolom pertama menunjukkan jumlah sampel negatif yang diprediksi benar oleh model (klasifikasi benar sebagai negatif).
- *False Positive* (FP):

  Nilai 16 pada baris pertama dan kolom kedua menunjukkan jumlah sampel negatif yang salah diprediksi sebagai positif (prediksi salah).
- *False Negative* (FN):

  Nilai 28 pada baris kedua dan kolom pertama menunjukkan jumlah sampel positif yang salah diprediksi sebagai negatif (prediksi salah)

4. Smoking Method

Pada tahap hasil evaluasi model smoking method ini didapatkan informasi berdasarkan Confusion Matrix yang dapat dilihat pada Tabel 3.7 Hasil Confusion Matrix Smoking Method.

Tabel 3.7 Hail Confusion Matrix Smoking Method

```
[[ 71 268]
 [ 63 289]]
```

Berdasarkan Tabel 3.7 diatas bahwa hasil evaluasi menggunakan confusion matrix pada model dengan smoking method. Confusion matrix di atas memiliki empat komponen utama yang merepresentasikan jumlah prediksi benar dan salah dalam dua kelas:

- *True Positive* (TP):

  Nilai 71 menunjukkan kasus yang benar-benar positif (positif sebenarnya) dan diprediksi sebagai positif
- *True Negative* (TN):

  Nilai 289 menunjukkan kasus yang benar-benar negatif (negatif sebenarnya) dan diprediksi sebagai negatif.
- *False Positive* (FP):

  Nilai 63 menunjukkan kasus yang sebenarnya negatif tetapi diprediksi sebagai positif (kesalahan tipe I).
- *False Negative* (FN):

Nilai 268 menunjukkan kasus yang sebenarnya positif tetapi diprediksi sebagai negatif (kesalahan tipe II)

3.2.2 Classification Report

Pada *classification report* yang dihasilkan dari pengujian model memberikan evaluasi tambahan seperti *precision, recall,* dan *f1-score* untuk masing-masing kelas. *Precision* dimaksudkan untuk mengukur proporsi positif yang benar. *Recall* dimaksudkan untuk mengukur proporsi kasus aktual yang benar-benar dikenali oleh model. *F1-Score* merupakan harmonisasi dari *precision* dan *recall*. *Support* menunjukkan jumlah sampel aktual dari setiap kelas.

1. CNN Arsitektur AlexNet

Pada evaluasi model yang menggunakan metode CNN arsitektur AlexNet didapatkan informasi berdasarkan *Classification Report* yang dapat dilihat pada Tabel 3.8 Hasil Classification Report pada Model CNN AlexNet.

Tabel 3.8 Hasil Classification Report AlexNet

	Precision	recall	f1-score	support
Smoking	0.93	0.90	0.91	352
Not Smoking	0.90	0.93	0.91	339
accuracy			0.91	691
macro avg	0.91	0.91	0.91	691
weighted avg	0.91	0.91	0.91	691

Berdasarkan Tabel 3.8 metode CNN arsitektur AlexNet memiliki akurasi tinggi (91%) dan performa yang seimbang di kedua kelas.

- Kelas Smoking: Memiliki precision tinggi (93%), sehingga jarang terjadi kesalahan dalam memprediksi individu yang benar-benar merokok.
- Kelas Not Smoking: Memiliki recall tinggi (93%), menunjukkan model mampu mendeteksi sebagian besar individu yang benar-benar tidak merokok.

2. CNN Arsitektur MobileNetV2

Pada evaluasi model yang menggunakan metode CNN arsitektur MobileNetV2 didapatkan informasi berdasarkan Classification Report yang dapat dilihat pada Tabel 3.9 Hasil Classification Report pada Model CNN MobileNetV2.

Tabel 3.9 Hasil Classification Report MobileNetV2

	Precision	recall	f1-score	support

Smoking	0.89	0.96	0.92	352
Not Smoking	0.95	0.88	0.92	339
accuracy			0.92	691
macro avg	0.92	0.92	0.92	691
weighted avg	0.92	0.92	0.92	691

Berdasarkan Tabel 3.9 metode CNN arsitektur MobileNetV2 memiliki akurasi tinggi (92%) dan performa yang seimbang di kedua kelas.

- Kelas Smoking: Memiliki recall tinggi (96%), sehingga model sangat baik dalam mendeteksi orang yang benar-benar merokok.
- Kelas Not Smoking: Memiliki precision tinggi (95%), sehingga model jarang membuat kesalahan dalam memprediksi seseorang sebagai tidak merokok

3. CNN Arsitektur ResNet50

Pada evaluasi model yang menggunakan metode CNN arsitektur ResNet50 didapatkan informasi berdasarkan Classification Report yang dapat dilihat pada Tabel 3.10 Hasil Classification Report pada Model CNN ResNet50.

Tabel 3.10 Hasil Classification Report ResNet50

	Precision	recall	f1-score	support
Smoking	0.92	0.95	0.94	352
Not Smoking	0.95	0.92	0.93	339
accuracy			0.94	691
macro avg	0.94	0.94	0.94	691
weighted avg	0.94	0.94	0.94	691

Berdasarkan Tabel 3.10 metode CNN arsitektur ResNet50 memiliki akurasi tinggi (94%) dan performa yang seimbang di kedua kelas

- Kelas Smoking: Memiliki recall tinggi (95%), sehingga model sangat baik dalam mendeteksi individu yang benar-benar merokok.
- Kelas Not Smoking: Memiliki precision tinggi (95%), sehingga model jarang membuat kesalahan dalam memprediksi individu sebagai tidak merokok

4. Smoking Method

Pada evaluasi model yang menggunakan metode smoking method didapatkan informasi berdasarkan Classification Report yang dapat dilihat pada Tabel 3.11 Hasil Classification Report pada Model Smoking Method.

Tabel 3.11 Hasil Classification Smoking Method

	Precision	recall	f1-score	support
Smoking	0.53	0.21	0.30	339
Not Smoking	0.52	0.82	0.64	352
accuracy			0.52	691
macro avg	0.52	0.52	0.47	691
weighted avg	0.52	0.52	0.47	691

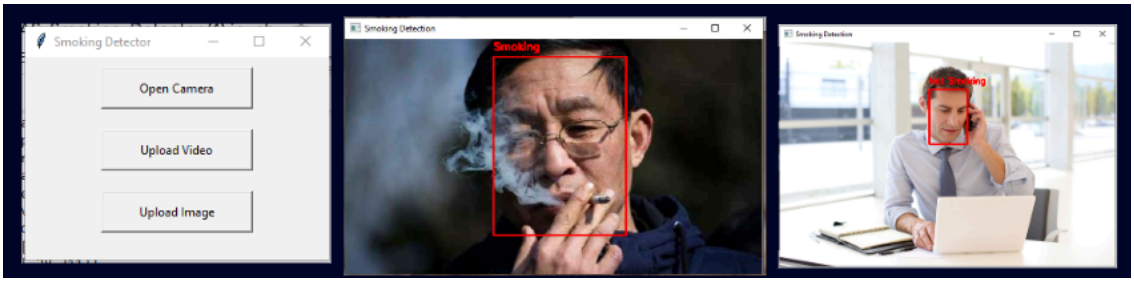
Berdasarkan Tabel 3.11, *Smoking Method* memiliki akurasi rendah (52%) dan performa yang tidak seimbang di kedua kelas:

a. Kelas Smoking:
   - Recall rendah (21%), menunjukkan model tidak mampu mendeteksi sebagian besar individu yang benar-benar merokok
   - Precision sedang (53%), artinya sekitar setengah dari prediksi "Smoking" adalah benar, namun masih banyak kesalahan prediksi.

b. Kelas Not Smoking:
   - Recall tinggi (82%), menunjukkan model cukup baik dalam mendeteksi individu yang benar-benar tidak merokok.
   - Precision rendah (52%), artinya model cukup sering salah memprediksi individu sebagai "Not Smoking" padahal sebenarnya merokok. Walaupun model lebih baik pada kelas ini, tingkat kesalahan prediksi tetap cukup signifikan.

3.2.5 Hasil GUI

Berdasarkan hasil yang diperoleh dalam tahap evaluasi bahwa ResNet50 merupakan model yang terbaik berdasarkan hasil evaluasi yang menunjukkan akurasi tinggi dan test loss yang rendah, pengembangan GUI (*Graphical User Interface*) untuk deteksi perokok dapat dilakukan dengan memanfaatkan model ini. Penggunaan ResNet50 dalam aplikasi GUI memungkinkan sistem untuk melakukan deteksi perokok dengan lebih akurat dan efisien. Dengan akurasi mencapai 94.05%, model ini mampu mengenali pola atau fitur penting dalam data yang berhubungan dengan perilaku perokok, seperti citra wajah atau perilaku yang teridentifikasi. Dapat ditemukan pada Gambar 3.1 Hasil GUI Deteksi Perokok.



Gambar 3.1 Hasil GUI Deteksi Perokok

BAB IV
KESIMPULAN

Berdasarkan hasil yang diperoleh dari evaluasi ketiga model, dapat disimpulkan bahwa model ResNet50 memberikan performa terbaik dengan akurasi sebesar 94.05% dan test loss 0.2975. Dengan hasil ini, ResNet50 menonjol sebagai model yang paling andal di antara ketiga model yang diuji, yaitu AlexNet dan MobileNetV2. MobileNetV2, meskipun sedikit lebih rendah dari ResNet50, mencapai akurasi 91.44% dengan test loss 0.3079, yang masih menunjukkan kinerja yang cukup baik, bahkan lebih unggul dibandingkan dengan AlexNet. Sementara itu, AlexNet memperoleh akurasi 91.73% dan test loss 0.3348, yang meskipun berada dalam rentang performa yang baik, tetap tidak dapat mengungguli ResNet50 dalam hal akurasi dan kemampuan prediksi. Di sisi lain, *smoking method* hanya mencapai akurasi 52.19%, yang jauh lebih rendah dibandingkan dengan model *deep learning* lainnya. Hal ini menunjukkan bahwa *smoking method* kurang cocok digunakan untuk proyek klasifikasi ini, karena performanya yang sangat terbatas dalam membedakan kelas-kelas yang ada. Dengan demikian, berdasarkan perbandingan performa, ResNet50 direkomendasikan sebagai model terbaik untuk digunakan dalam proyek deteksi perokok. Setelah mengetahui bahwa ResNet50 merupakan model yang terbaik, maka pengembangan GUI *(Graphical User Interface)* untuk deteksi perokok dapat dilakukan dengan memanfaatkan model ResNet50, untuk menyediakan aplikasi yang lebih akurat dan efisien dalam identifikasi perokok.