

Tugas 4 IF4073 Pemrosesan Citra Digital

Sistem Pengenalan Jenis Kendaraan



Disusun Oleh:

Jason Rivalino 13521008

Arlene Chrysantha Gunardi 13521059

Juan Christopher Santoso 13521116

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG**

2024

Daftar Isi

Daftar Isi.....	1
Daftar Gambar.....	2
1. Deskripsi Masalah.....	3
2. GUI.....	3
3. Program.....	4
3.1. Program Utama.....	4
3.2. Dataset.....	4
3.3. Program Pengenalan Jenis Kendaraan dengan Metode Konvensional.....	6
3.4. Program Pengenalan Jenis Kendaraan dengan Pembelajaran Mendalam.....	16
4. Hasil Eksekusi Program.....	24
4.1. Metode Konvensional.....	24
4.2. Pembelajaran Mendalam.....	31
5. Diskusi dan Analisis Hasil Eksekusi Program.....	34
6. Kesimpulan.....	36
7. Komentar dan Refleksi.....	36
8. Referensi.....	37
9. Lampiran.....	37

Daftar Gambar

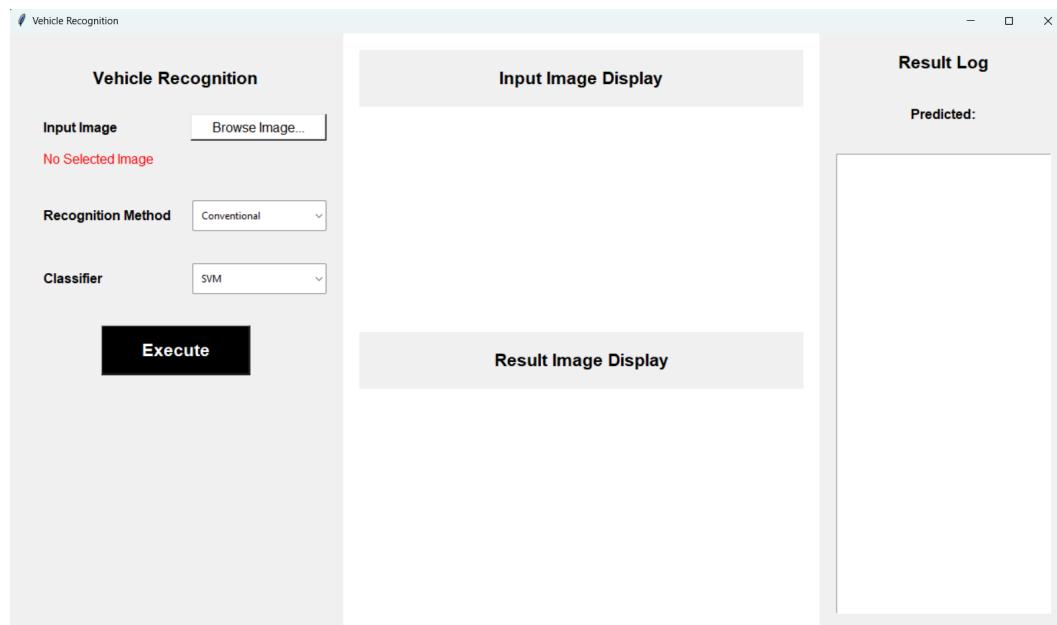
Gambar 1.1. GUI Program.....	3
Gambar 3.1. Contoh dataset (Sumber: https://www.kaggle.com/code/eyadgk/vehicle-rec-via-efficientnet-100-acc-full-guide/notebook)	5
Gambar 3.2. Struktur folder dari dataset_training untuk pelabelan.....	5
Gambar 3.3. Hasil Classification Report untuk pelatihan model dengan SVM.....	14
Gambar 3.4. Hasil Classification Report untuk pelatihan model dengan KNN.....	14
Gambar 3.5. Rangkuman struktur model.....	22
Gambar 4.1. Hasil eksekusi program untuk citra mobil dengan metode konvensional SVM (1).	24
Gambar 4.2. Hasil eksekusi program untuk citra mobil dengan metode konvensional SVM (2).	24
Gambar 4.3. Hasil eksekusi program untuk citra mobil dengan metode konvensional SVM (3).	25
Gambar 4.4. Hasil eksekusi program untuk citra mobil dengan metode konvensional KNN (4).	25
Gambar 4.5. Hasil eksekusi program untuk citra bus dengan metode konvensional SVM (1)....	26
Gambar 4.6. Hasil eksekusi program untuk citra bus dengan metode konvensional SVM (2)....	26
Gambar 4.7. Hasil eksekusi program untuk citra bus dengan metode konvensional KNN (3)....	27
Gambar 4.8. Hasil eksekusi program untuk citra bus dengan metode konvensional SVM (4)....	27
Gambar 4.9. Hasil eksekusi program untuk citra truk dengan metode konvensional KNN (1)....	28
Gambar 4.10. Hasil eksekusi program untuk citra truk dengan metode konvensional SVM (2)..	28
Gambar 4.11. Hasil eksekusi program untuk citra truk dengan metode konvensional SVM (3)..	29
Gambar 4.12. Hasil eksekusi program untuk citra truk dengan metode konvensional KNN (4)..	29
Gambar 4.13. Hasil eksekusi program untuk citra motor dengan metode konvensional KNN (1)... 30	
Gambar 4.14. Hasil eksekusi program untuk citra motor dengan metode konvensional SVM(2)	30
Gambar 4.15. Hasil eksekusi program untuk citra mobil dengan pembelajaran mendalam (1)....	31
Gambar 4.16. Hasil eksekusi program untuk citra mobil dengan pembelajaran mendalam (2)....	31
Gambar 4.17. Hasil eksekusi program untuk citra bus dengan pembelajaran mendalam (1).....	32
Gambar 4.18. Hasil eksekusi program untuk citra bus dengan pembelajaran mendalam (2).....	32
Gambar 4.19. Hasil eksekusi program untuk citra truk dengan pembelajaran mendalam (1).....	33
Gambar 4.20. Hasil eksekusi program untuk citra truk dengan pembelajaran mendalam (2).....	33
Gambar 4.21. Hasil eksekusi program untuk citra sepeda motor dengan pembelajaran mendalam (1).....	34
Gambar 4.22. Hasil eksekusi program untuk citra sepeda motor dengan pembelajaran mendalam (2).....	34
Gambar 5.1. Evaluasi model pembelajaran mendalam.....	36

1. Deskripsi Masalah

Tugas 4 IF4073 Pemrosesan Citra Digital menugaskan pembuatan sistem pengenalan kendaraan (*vehicle recognition*) dengan menggunakan teknik-teknik pengolahan citra. Pengerajan program untuk pengenalan kendaraan dibuat dengan bahasa pemrograman Python. Terdapat dua metode yang diimplementasikan, yaitu metode konvensional dengan menggunakan teknik-teknik pengolahan citra (deteksi tepi, segmentasi, dll.) dan pembelajaran mendalam dengan arsitektur CNN.

2. GUI

Berikut adalah tangkapan layar untuk tampilan GUI.



Gambar 1.1. GUI Program

GUI program dapat dibagi menjadi 3 bagian besar, yaitu:

1. Bagian *panel* yang tertera di sebelah kiri berfungsi untuk menyediakan konfigurasi dan menerima *input* dari pengguna. *Input* yang diberikan adalah citra masukan, metode pengenalan, dan jenis *classifier* yang digunakan (khusus untuk metode konvensional).
2. Bagian *display* yang tertera di bagian tengah berfungsi untuk menampilkan citra masukan dan luaran proses pengenalan kendaraan.

3. Bagian *logging* yang tertera di sebelah kanan berfungsi untuk menampilkan hasil pengenalan kendaraan yang sudah dilakukan oleh sistem.

3. Program

3.1. Program Utama

Program utama terdiri dari algoritma dan sistem yang mengatur tampilan (GUI) dan integrasi kepada sistem pengenalan, baik konvensional maupun pembelajaran mendalam. Program utama ini sekaligus menjadi *interface* yang digunakan pengguna untuk menjalankan sistem yang ada.

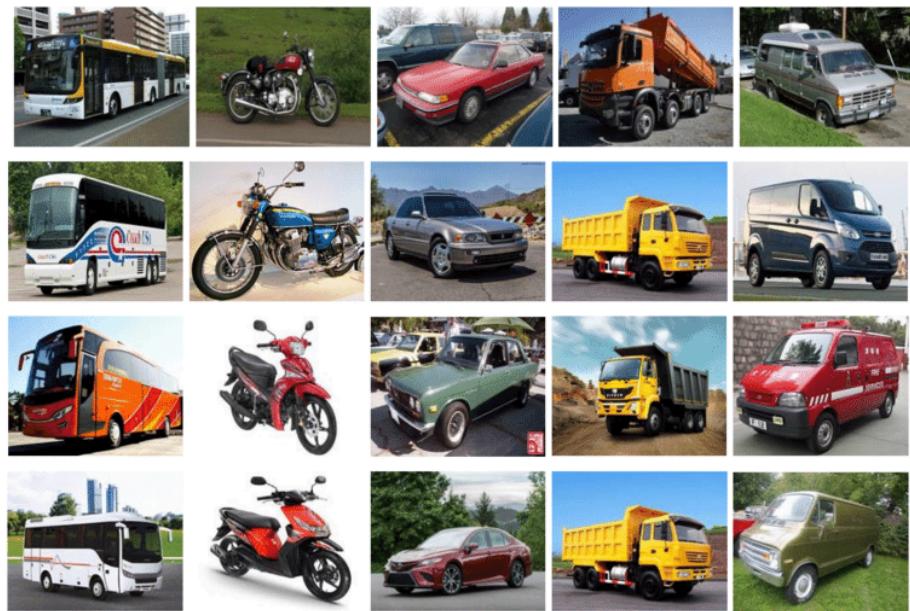
Program utama ini juga mengatur konfigurasi terkait sistem yang dikembangkan, antara lain:

1. Ukuran dari tampilan GUI adalah 1280 x 720 piksel untuk mula-mula. Namun, pengguna dapat mengubah ukuran tampilan layar jika diinginkan.
2. Ukuran citra yang digunakan adalah 480 x 270. Ukuran ini dipilih karena memiliki rasio dimensi 16:9.
3. Jenis *classifier* yang digunakan pada proses pengenalan kendaraan metode konvensional dibagi menjadi 2, yakni SVM dan KNN. Pada GUI, pengguna juga dapat memilih jenis *classifier* yang ingin digunakan.
4. Pengguna dapat memberikan *input* berupa citra melalui tombol yang tertera pada GUI. GUI akan menampilkan citra *input* yang dimasukkan pengguna.
5. Untuk melakukan proses pengenalan kendaraan, pengguna dapat menekan tombol *execute* yang tertera. Hasil dari proses adalah ditampilkannya citra dengan kendaraan yang terdeteksi serta ditampilkannya kelas kendaraan sebagai hasil dari pengenalan.

3.2. Dataset

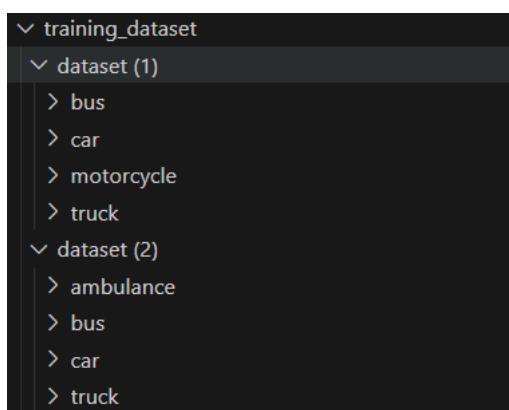
Dataset citra kendaraan yang digunakan untuk pelatihan merupakan *dataset* berlabel. Terdapat dua sumber *dataset* yang digunakan. *Dataset* pertama terdiri dari kumpulan citra bus, mobil (*car*), sepeda motor (*motorcycle*), dan truk (*truck*), masing-masing memiliki 100 buah citra. *Dataset* kedua terdiri dari kumpulan citra ambulans (*ambulance*) sebanyak 120 buah, bus sebanyak 166 buah, mobil (*car*)

sebanyak 145 buah, dan truk (*truck*) sebanyak 125 buah citra. Contoh citra *dataset* dapat dilihat pada Gambar 3.1. Adapun sumber *dataset* terlampir pada bagian Referensi.



Gambar 3.1. Contoh *dataset* (Sumber:
<https://www.kaggle.com/code/evadgk/vehicle-rec-via-efficientnet-100-acc-full-guide/notebook>)

Struktur direktori untuk pelabelan kendaraan terdiri atas lima jenis kendaraan yaitu *bus*, *motorcycle*, *truck*, *car*, dan *ambulance*. Untuk penjelasan lebih detail terkait *dataset* sudah dijelaskan pada bagian sebelumnya. Gambar 3.2 menunjukkan struktur *folder* dari *dataset_training* untuk pelabelan.



Gambar 3.2. Struktur *folder* dari *dataset_training* untuk pelabelan

3.3. Program Pengenalan Jenis Kendaraan dengan Metode Konvensional

Program pengenalan jenis kendaraan dengan metode konvensional secara umum memanfaatkan sejumlah pra-pemrosesan citra di awal untuk memproses citra agar dapat mengenali pola tertentu yang terdapat di dalam citra. Kemudian, akan dilakukan pembelajaran mesin menggunakan dua metode pilihan, yaitu KNN dan SVM. Proses pembelajaran dibagi menjadi dua tahap, yaitu pelatihan (*train*) terhadap *dataset* untuk membentuk model dan prediksi berbagai jenis kendaraan masukan menggunakan model yang sudah dilatih. Hasil keluaran akhir yang didapatkan berupa prediksi kelas dari citra kendaraan masukan beserta dengan nilai akurasinya.

Berikut merupakan penjelasan lebih detail terkait proses pengenalan jenis kendaraan dengan metode konvensional:

1. Proses pelatihan terhadap *dataset* yang dimiliki (*train*)

Proses pelatihan untuk pembentukan model dilakukan pada fungsi `train.py` dengan memanggil sejumlah fungsi untuk melakukan pelatihan *dataset* untuk pembelajaran dengan metode KNN dan SVM. Berikut adalah kode programnya.

```
from conventional.classification.knn import train_knn
from conventional.classification.svm import train_svm
from deep_learning.model import load_model
from constant import DATASET_PATH

if __name__ == "__main__":
    print(f"[TRAINING] Start training SVM model")
    train_svm(DATASET_PATH)

    print(f"[TRAINING] Start training KNN model")
    train_knn(DATASET_PATH)
```

2. Proses pelatihan *dataset* untuk pembentukan model

Terdapat beberapa tahapan yang dilalui selama proses pelatihan *dataset* untuk dapat membentuk model yang akan digunakan untuk prediksi yaitu sebagai berikut:

a. Memuat *dataset* dari berbagai jenis kendaraan

Pemuatan dataset dilakukan dengan memanggil fungsi `load_dataset` untuk mendapatkan nilai fitur dan label dengan kode berikut.

```
features, labels = load_dataset(dataset_path)
```

Fungsi ini mengekstraksi setiap citra kendaraan yang terdapat di dalam *dataset* untuk mendapatkan nilai fitur (total dari keseluruhan citra) dan label (jenis kendaraan). Pelabelan kendaraan ditentukan berdasarkan penamaan direktori dari *folder dataset* tempat berbagai citra kendaraan ditempatkan. Berikut merupakan kode programnya.

```
import os
import numpy as np
from PIL import Image
from conventional.preprocess_image import preprocess_image
from constant import IMAGE_HEIGHT, IMAGE_WIDTH

# Assuming preprocess_image returns features and hog_image
def load_dataset(dataset_paths, amount_each_class: int = 200):
    features = []
    labels = []

    for dataset_path in dataset_paths:
        print(f'[PROCESSING DATASET] {dataset_path}')
        if not os.path.isdir(dataset_path):
            print(f"Dataset path '{dataset_path}' does not exist or is not a directory.")
            continue

        for label in os.listdir(dataset_path): # Loop
            label_path = os.path.join(dataset_path, label)
            if os.path.isdir(label_path):
                for image_file in
os.listdir(label_path)[:amount_each_class]:
                    image_path = os.path.join(label_path,
image_file)
                    print(f'[EXTRACTING FEATURE]
```

```

{image_path})

    try:
        # Load and preprocess the image
        image =
Image.open(image_path).convert('RGB')    # Ensure 3 channels
                                                image = image.resize((IMAGE_WIDTH,
IMAGE_HEIGHT))    # Resize image

        # Extract features using
preprocess_image
            feature_vector, _ =
preprocess_image(image, target_feature_size=1000)
            features.append(feature_vector)
            labels.append(label)    # Label is
the folder name

        except Exception as e:
            print(f"Error processing
{image_path}: {e}")

        # Logging statistics
        print(f"Processed {len(features)} images in total.")
        for label in set(labels):
            print(f"Number of images for label '{label}' :
{labels.count(label)}")

    return np.array(features), np.array(labels)

```

Selain itu, pada proses pemuatan *dataset*, dilakukan juga pra-pemrosesan terhadap berbagai citra yang ada untuk menentukan pola yang terdapat pada citra.

b. Pra-pemrosesan citra

Pra-pemrosesan citra dilakukan untuk menentukan pola yang terdapat pada citra, sehingga nanti dapat diproses untuk membantu pembentukan model latih. Tahap ini dibagi menjadi dua bagian besar, yakni *thresholding* dan *pre-processing*. Bagian *thresholding* dilakukan untuk mempersiapkan citra agar dapat lebih mudah dikenali pada bagian *pre-processing*. Tahap-tahap pada bagian *thresholding* mencakup proses pengubahan citra ke citra

grayscale, pelaksanaan *blur*, pengenalan tepi dengan Canny, dan pengenalan segmentasi menggunakan metode Otsu. *Output* dari proses ini adalah citra yang sudah diproses dengan *thresholding*. Berikut adalah kode programnya.

```
def thresholding_image(image):
    # Convert to grayscale
    img = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

    # Apply Gaussian blur
    img = cv2.GaussianBlur(img, (5, 5), 0)

    # Thresholding using otsu
    _, otsu_thresh = cv2.threshold(img, 0, 255,
cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    kernel_close = np.ones((5, 5), np.uint8)
    otsu_new = cv2.morphologyEx(otsu_thresh,
cv2.MORPH_CLOSE, kernel_close)

    # Thresholding using edges
    edges = cv2.Canny(image, threshold1=275,
threshold2=300)

    res = cv2.bitwise_xor(edges, otsu_new)
    res = cv2.medianBlur(res, 3)
    kernel_res = np.ones((3, 3), np.uint8)
    res = cv2.morphologyEx(res, cv2.MORPH_CLOSE,
kernel_res)
    return res
```

Beberapa pra-pemrosesan yang diimplementasikan adalah sebagai berikut:

- i. Mengubah masukan citra ke dalam bentuk *array*.
- ii. *Resize* citra untuk mengubah ukuran menjadi 480 x 270.
- iii. Memanggil fungsi *thresholding*.
- iv. Implementasi fitur HOG dengan sejumlah parameter sebagai berikut
 - *img*: citra masukan dalam format *array*.
 - *orientations*: jumlah *bin* yang terdapat dalam HOG yang menentukan orientasi gradien (nilai yang digunakan: 9).

- `pixels_per_cell`: ukuran sel yang ada di dalam piksel untuk perhitungan HOG (nilai yang digunakan: (8,8)).
 - `cells_per_block`: jumlah sel yang terdapat dalam satu blok untuk normalisasi lokal (nilai yang digunakan: (2,2)).
 - `block_norm`: metode normalisasi blok untuk perbaikan konsistensi fitur di seluruh citra. Metode yang digunakan adalah L2-Hys untuk menambahkan *clipping* nilai agar stabil.
 - `transform_sqrt`: transformasi akar digunakan pada intensitas piksel untuk menstabilkan perubahan besar dalam intensitas pencahayaan citra.
 - `visualize`: untuk mengembalikan citra berupa `HOG_image` yang merepresentasikan HOG secara visual.
- v. Membuat *bounding box* untuk menentukan *region* wilayah dari masukan citra kendaraan. Implementasi *bounding box* memanfaatkan *threshold* yaitu untuk pembuatan *mask* berdasarkan nilai yang lebih besar dari 0.9 akan menjadi 1 sedangkan nilai lain berubah menjadi 0 untuk menentukan area gambar HOG dengan fitur signifikan. Setelah mendapatkan nilai *bounding box*, *region* yang termasuk di dalamnya (kendaraan yang dideteksi) akan digambar dengan bentuk persegi panjang untuk menentukan batas kendaraannya.
- vi. Melakukan penyesuaian fitur HOG untuk memotong fitur dan mengisi kekurangan menggunakan *padding* agar sesuai dengan ukuran target.
- vii. Mengembalikan gambar yang sudah dilakukan proses HOG dan telah dibatasi oleh *bounding box*.

Berikut adalah kode programnya.

```
def preprocess_image(pil_image, target_feature_size):

    # Convert PIL image to numpy array
    img_init = np.array(pil_image)
```

```

# Resize the image
img_resized = cv2.resize(img_init, (IMAGE_WIDTH,
IMAGE_HEIGHT)) # Use consistent size

thresholded_img = thresholding_image(img_resized)

# Extract HOG features and visualization
hog_features, hog_image = hog(
    thresholded_img,
    orientations=9,
    pixels_per_cell=(8, 8),
    cells_per_block=(2, 2),
    block_norm='L2-Hys',
    transform_sqrt=True,
    visualize=True
)

# Find the coordinates of the bounding box for HOG
features
maxval_hog = npamax(hog_image)
binary_mask = (hog_image >
maxval_hog*0.1).astype(np.uint8) # Create a binary mask
coords = cv2.findNonZero(binary_mask) # Find non-zero
coordinates

# Determine bounding box based on non-zero pixels
if coords is not None:
    x, y, w, h = cv2.boundingRect(coords) # Get
bounding box
    top_left = (x, y)
    bottom_right = (x + w, y + h)
else:
    top_left = (0, 0)
    bottom_right = (IMAGE_WIDTH, IMAGE_HEIGHT) # Default to full image if no features

cv2.rectangle(binary_mask, top_left, bottom_right, 1,
2)

bounded_img = img_resized.copy()

```

```

        cv2.rectangle(bounded_img, top_left, bottom_right,
(255,0,0), 2)

        # Adjust the features to match the target size
        if len(hog_features) > target_feature_size:
            hog_features = hog_features[:target_feature_size]
        # Trim excess features
        else:
            hog_features = np.pad(hog_features, (0,
target_feature_size - len(hog_features)), mode='constant')
        # Pad with zeros

    return hog_features, bounded_img

```

3. Proses pelatihan *dataset* untuk pembentukan model

Pelatihan yang dilakukan dalam *dataset* secara umum dilakukan dengan melakukan pembagian gambar yang termasuk ke dalam data latih dan yang termasuk ke dalam data uji. Terdapat 80% citra yang digunakan untuk data latih dan 20% citra yang digunakan untuk data uji dengan kode sebagai berikut.

```

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    features, labels, test_size=0.2, stratify=labels,
random_state=42
)

```

Setelah dilakukan pembagian terhadap *dataset*, proses selanjutnya adalah pembelajaran menggunakan dua jenis metode yaitu *Support Vector Machine* (SVM) dan *K-Nearest Neighbor* (KNN) dengan detail dari kedua metode tersebut adalah sebagai berikut:

a. Pelatihan *dataset* dengan metode SVM

Proses awal yang dilakukan adalah dengan melakukan *hyperparameter tuning* dengan sejumlah parameter antara lain sebagai berikut:

- *c*: parameter regulasi untuk pengaturan margin dan mengurangi *error*
- *kernel*: fungsi untuk pemetaan data pada ruang dimensi yang lebih tinggi

- `gamma`: parameter penentu seberapa jauh pengaruh dari satu titik data
- `degree`: parameter untuk menentukan derajat polinomial untuk *kernel poly*

b. Pelatihan *dataset* dengan metode KNN

Proses awal yang dilakukan adalah dengan melakukan *hyperparameter tuning* dengan sejumlah parameter antara lain sebagai berikut:

- `n_neighbors`: banyaknya tetangga dalam model KNN (nilai antara 3,5,7,9)
- `weights`: pemberian bobot pada tetangga (*uniform* atau *distance*)
- `metric`: metode pengukuran jarak antar titik (Euclidean, Manhattan, atau Minkowski)

Proses berikutnya menggunakan `GridSearchCV` untuk mencoba kombinasi semua *hyperparameter tuning*. Kemudian dilakukan proses *fit* untuk mencari konfigurasi terbaik berdasarkan parameter. Model SVM kemudian akan diambil berdasarkan kombinasi parameter terbaik yang didapat. Berikut merupakan kode programnya:

```
SVM:  
grid = GridSearchCV(SVC(probability=True), param_grid,  
cv=10, scoring='accuracy', n_jobs=-1)  
  
KNN:  
grid = GridSearchCV(KNeighborsClassifier(), param_grid,  
cv=5, n_jobs=-1)
```

Selanjutnya akan dilakukan evaluasi model dengan membuat prediksi pada data uji serta menampilkan hasil visualisasi data berupa klasifikasi dan *confusion matrix* berdasarkan hasil prediksi. Kemudian model pelatihan akan disimpan ke dalam bentuk *file pkl* untuk penggunaan prediksi masukan kendaraan selanjutnya. Berikut merupakan kode programnya.

```
grid.fit(X_train, y_train)  
    print(f"Best Parameters: {grid.best_params_}")  
    svm = grid.best_estimator_  
    print(f"Training Time: {time.time() - start_time:.2f}")
```

```

seconds")

# Evaluation
y_pred = svm.predict(X_test)
print(classification_report(y_test, y_pred,
zero_division=0))
print(confusion_matrix(y_test, y_pred))

# Save Model
with open(svm_model_file_path, 'wb') as f:
    pickle.dump(svm, f)

```

Berdasarkan hasil prediksi dari evaluasi citra, berikut merupakan rincian detail dari *classification report* untuk pelatihan model dengan metode SVM dan KNN (Gambar 3.3 dan Gambar 3.4).

SVM Report				
	precision	recall	f1-score	support
bus	0.38	0.25	0.30	20
car	0.30	0.70	0.42	20
motorcycle	0.17	0.05	0.08	20
truck	0.07	0.05	0.06	20
accuracy			0.26	80
macro avg	0.23	0.26	0.22	80
weighted avg	0.23	0.26	0.22	80

Gambar 3.3. Hasil *Classification Report* untuk pelatihan model dengan SVM

KNN Report				
	precision	recall	f1-score	support
bus	0.33	0.15	0.21	20
car	0.50	0.05	0.09	20
motorcycle	0.27	0.85	0.40	20
truck	0.40	0.10	0.16	20
accuracy			0.29	80
macro avg	0.37	0.29	0.22	80
weighted avg	0.37	0.29	0.22	80

Gambar 3.4. Hasil *Classification Report* untuk pelatihan model dengan KNN

4. Fungsi prediksi berdasarkan pembentukan model

Fungsi prediksi dilakukan dengan menerima masukan citra yang kemudian akan dilakukan pra-pemrosesan dengan cara yang sama seperti citra pada *dataset* untuk mencari nilai fiturnya. Program juga akan memuat model pelatihan dari *file pkl* yang telah disimpan dari proses pelatihan sebelumnya. Dari nilai fitur yang didapatkan, akan dilakukan prediksi terhadap kelas kendaraan untuk citra masukan beserta nilai akurasinya. Berikut merupakan kode programnya.

SVM:

```
def predict_svm(img):
    feature, bounded_image = preprocess_image(img,
target_feature_size=1000)

    # Load Model
    with open(svm_model_file_path, 'rb') as f:
        svm = pickle.load(f)

    # Preprocess new image
    feature = feature.reshape(1, -1)

    # Predict new image
    proba = svm.predict_proba(feature)
    classes = svm.classes_

    return proba, classes, bounded_image
```

KNN:

```
def predict_knn(img):
    feature, bounded_image = preprocess_image(img,
target_feature_size=1000)

    # Load Model
    with open(knn_model_file_path, 'rb') as f:
        knn = pickle.load(f)

    # Preprocess new image
    feature = feature.reshape(1, -1)

    # Predict new image
```

```
    proba = knn.predict_proba(feature)
    classes = knn.classes_

    return proba, classes, bounded_image
```

3.4. Program Pengenalan Jenis Kendaraan dengan Pembelajaran Mendalam

Program pengenalan jenis kendaraan dengan pembelajaran mendalam menggunakan model *pre-trained* dengan arsitektur CNN, khususnya EfficientNet (EfficientNetB3). Program terdiri dari beberapa tahap, mulai dari membangkitkan direktori *dataset* latih, membuat *dataframe* dari *dataset*, memisahkan *dataset* latih, validasi, dan uji, membuat *image data generator*, membuat struktur model, memuat bobot model *pre-trained* ke dalam model, dan terakhir memprediksi kelas menggunakan model yang sudah dilatih.

Program utama untuk membangun dan memuat bobot model terdapat pada fungsi `load_model(data_dir)` berikut. Fungsi ini memanggil fungsi-fungsi lainnya, mulai dari fungsi `generate_data_paths(data_dir)`, hingga fungsi `create_model_structur(train_gen)` dan memuat bobot melalui fungsi `load_weights`. Terakhir, model disimpan dalam file bernama `cnn_model.pkl`.

```
import pickle
from deep_learning.preprocessing import *

cnn_model_file_path = os.path.join(os.getcwd(), 'src',
'deep_learning', 'model', 'cnn_model.pkl')

def load_model(data_dir):
    # Load Model
    filepaths, labels = generate_data_paths(data_dir)
    df = create_df(filepaths, labels)

    train_df, valid_df, test_df = split_dataset(df)
    train_gen, valid_gen, test_gen =
    generate_image_data(train_df, valid_df, test_df)
```

```

model = create_model_structure(train_gen)
model.load_weights(
    './src/deep_learning/my_model_weights.h5')

# Save Model
with open(cnn_model_file_path, 'wb') as f:
    pickle.dump(model, f)

```

Pertama-tama, program akan memuat *dataset* dan membangkitkan daftar direktoriya melalui fungsi `generate_data_paths(data_dir)` untuk membangkitkan direktori *dataset* beserta dengan labelnya. Fungsi ini mengembalikan daftar *file path* dan daftar label. Berikut adalah kode programnya.

```

import os

def generate_data_paths(data_dir):
    filepaths = []
    labels = []
    folds = os.listdir(data_dir)
    for fold in folds:
        foldpath = os.path.join(data_dir, fold)
        filelist = os.listdir(foldpath)
        for file in filelist:
            fpath = os.path.join(foldpath, file)
            filepaths.append(fpath)
            labels.append(fold)
    return filepaths, labels

```

Selanjutnya, program akan membuat *dataframe* dari *dataset* yang ada melalui fungsi `create_df(filepaths, labels)`, lalu memisahkan *dataset* latih, validasi, dan uji melalui fungsi `split_dataset(df)`. *Dataset* latih terdiri dari 82 citra mobil, 74 citra sepeda motor, 79 citra bus, dan 85 citra truk. *Dataset* validasi terdiri dari 8 citra mobil, 21 citra sepeda motor, 11 citra bus, dan 8 citra truk. *Dataset* uji terdiri dari 10 citra mobil, 5 citra sepeda motor, 10 citra bus, dan 7 citra truk. Berikut merupakan kode programnya.

```

import pandas as pd
import tensorflow as tf

```

```

from sklearn.model_selection import train_test_split

def create_df(filepaths, labels):
    Fseries = pd.Series(filepaths, name= 'filepaths')
    Lseries = pd.Series(labels, name='labels')
    df = pd.concat([Fseries, Lseries], axis= 1)
    return df

def split_dataset(df):
    train_df, dummy_df = train_test_split(df, train_size= 0.8,
shuffle= True, random_state= 123)
    valid_df, test_df = train_test_split(dummy_df, train_size=
0.6, shuffle= True, random_state= 123)
    return train_df, valid_df, test_df

```

Kemudian, program akan mempersiapkan *image data generator* yang akan digunakan pada proses pelatihan, validasi, dan pengujian, sehingga model lebih *robust* untuk memprediksi berbagai masukan. Ini dilakukan dengan fungsi `generate_image_data(train_df, valid_df, test_df)`. Konfigurasi yang digunakan untuk ukuran *batch* pemrosesan adalah 16, ukuran citra 224 x 224, kanal 3 warna (RGB), dan bentuk citra direpresentasikan dengan ukuran dan kanal warnanya. Kemudian, ukuran *batch* untuk data uji dibagi secara merata untuk total banyaknya sampel pengujian dan ukurannya kurang dari atau sama dengan 80. Dengan konfigurasi tersebut, dibuat *generator* untuk pelatihan, validasi, dan pengujian, masing-masing dengan augmentasi berupa rotasi, penggeseran, kecerahan, *zoom*, dan *flip* agar model terlatih oleh *dataset* yang beragam. Berikut merupakan kode programnya.

```

import tensorflow as tf
from tensorflow.keras.preprocessing.image import
ImageDataGenerator

def generate_image_data(train_df, valid_df, test_df):
    batch_size = 16
    img_size = (224, 224)
    channels = 3
    img_shape = (img_size[0], img_size[1], channels)

```

```

    ts_length = len(test_df)
    test_batch_size = max(sorted([ts_length // n for n in
range(1, ts_length + 1) if ts_length%n == 0 and ts_length/n <=
80]))
    test_steps = ts_length // test_batch_size

    def scalar(img):
        return img

    tr_gen = ImageDataGenerator(
        preprocessing_function= scalar,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        brightness_range=[0.4,0.6],
        zoom_range=0.3,
        horizontal_flip=True,
        vertical_flip=True)

    ts_gen = ImageDataGenerator(
        preprocessing_function= scalar,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        brightness_range=[0.4,0.6],
        zoom_range=0.3,
        horizontal_flip=True,
        vertical_flip=True)

    train_gen = tr_gen.flow_from_dataframe(train_df,
        x_col= 'filepaths',
        y_col= 'labels',
        target_size= img_size,
        class_mode=
'categorical',
        color_mode= 'rgb',
        shuffle= True,
        batch_size= batch_size)

```

```

valid_gen = ts_gen.flow_from_dataframe(valid_df,
                                         x_col= 'filepaths',
                                         y_col= 'labels',
                                         target_size= img_size,
                                         class_mode=
                                         'categorical',
                                         color_mode= 'rgb',
                                         shuffle= True,
                                         batch_size= batch_size)

test_gen = ts_gen.flow_from_dataframe(test_df,
                                         x_col= 'filepaths',
                                         y_col= 'labels',
                                         target_size= img_size,
                                         class_mode=
                                         'categorical',
                                         color_mode= 'rgb',
                                         shuffle= False,
                                         batch_size=
                                         test_batch_size)

return train_gen, valid_gen, test_gen

```

Setelah membuat *generator*, program membuat struktur model melalui fungsi `create_model_structure(train_gen)`. Ini menggunakan *generator* latih yang sebelumnya dikembalikan dari fungsi `generate_image_data`. Model menggunakan arsitektur *pre-trained* EfficientNetB7. Model menggunakan *global max pooling* untuk *feature extraction*. Bobot model dasar dibekukan untuk mempertahankan pengetahuan yang telah dilatih sebelumnya. Selain itu, fungsi ini menambahkan lapisan khusus, termasuk normalisasi *batch*, lapisan *dense* dengan regularisasi L2 dan L1 untuk mencegah *overfitting*, *dropout* untuk regularisasi lebih lanjut, dan lapisan *dense* akhir dengan aktivasi *softmax* untuk klasifikasi multikelas. Model ini dikompilasi dengan pengoptimal Adamax dan *categorical cross entropy loss*, sehingga siap untuk pelatihan. Adapun rangkumgan (*summary*) mengenai struktur model dapat dilihat pada Gambar 3.5. Berikut merupakan kode programnya.

```

import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,
BatchNormalization
from tensorflow.keras.optimizers import Adamax
from tensorflow.keras import regularizers

def create_model_structure(train_gen):
    img_size = (224, 224)
    channels = 3
    img_shape = (img_size[0], img_size[1], channels)
    class_count = len(list(train_gen.class_indices.keys())) # to
define number of classes in dense layer

    # Create pre-trained model (efficientnetb3)
    base_model =
tf.keras.applications.efficientnet.EfficientNetB7(include_top=
False, weights= "imagenet", input_shape= img_shape, pooling=
'max')
    base_model.trainable = False

    model = Sequential([
        base_model,
        BatchNormalization(axis= -1, momentum= 0.99, epsilon=
0.001),
        Dense(128, kernel_regularizer= regularizers.l2(0.016),
activity_regularizer= regularizers.l1(0.006),
                bias_regularizer= regularizers.l1(0.006),
activation= 'relu'),
        Dropout(rate= 0.45, seed= 123),
        Dense(class_count, activation= 'softmax')
    ])

    model.compile(Adamax(learning_rate= 0.001), loss=
'categorical_crossentropy', metrics= ['accuracy'])

    return model

```

Layer (type)	Output Shape	Param #
efficientnetb7 (Functional)	(None, 2560)	64,097,687
batch_normalization (BatchNormalization)	(None, 2560)	10,240
dense (Dense)	(None, 128)	327,808
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516

Total params: 64,436,251 (245.80 MB)

Trainable params: 333,444 (1.27 MB)

Non-trainable params: 64,102,807 (244.53 MB)

Gambar 3.5. Rangkuman struktur model

Terakhir, untuk melakukan prediksi label kelas terhadap citra masukan berdasarkan model yang telah dilatih, diperlukan fungsi `predict_class(image_path, model, class_labels)`. Pertama, model yang sudah dilatih (`cnn_model.pkl`) akan dimuat terlebih dahulu, kemudian citra masukan dibaca dari direktori citranya, dikonversi menjadi *array*, menambahkan dimesi *batch*, dan melakukan pra-pemrosesan citra masukan untuk normalisasi terhadap model. Fungsi akan mengembalikan label hasil prediksi model, misalnya *bus*, *car*, *motorcycle*, atau *truck*.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.efficientnet import
preprocess_input

cnn_model_file_path = os.path.join(os.getcwd(), 'src',
'deep_learning', 'model', 'cnn_model.pkl')

def predict_class(image_path, model, class_labels):
    # Load Model
    with open(cnn_model_file_path, 'rb') as f:
```

```

model = pickle.load(f)

# Predict
img = image.load_img(image_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

prediction = model.predict(img_array)
predicted_class_index = np.argmax(prediction)

predicted_class_label = class_labels[predicted_class_index]

return predicted_class_label

```

Terakhir, dilakukan evaluasi terhadap model, yaitu untuk *loss* dan akurasi. Berikut adalah kode programnya.

```

# Evaluate

ts_length = len(test_df)
test_batch_size = max(sorted([ts_length // n for n in range(1,
ts_length + 1) if ts_length % n == 0 and ts_length/n <= 80]))
test_steps = ts_length // test_batch_size

train_score = model.evaluate(train_gen, steps= test_steps,
verbose= 1)
valid_score = model.evaluate(valid_gen, steps= test_steps,
verbose= 1)
test_score = model.evaluate(test_gen, steps= test_steps,
verbose= 1)

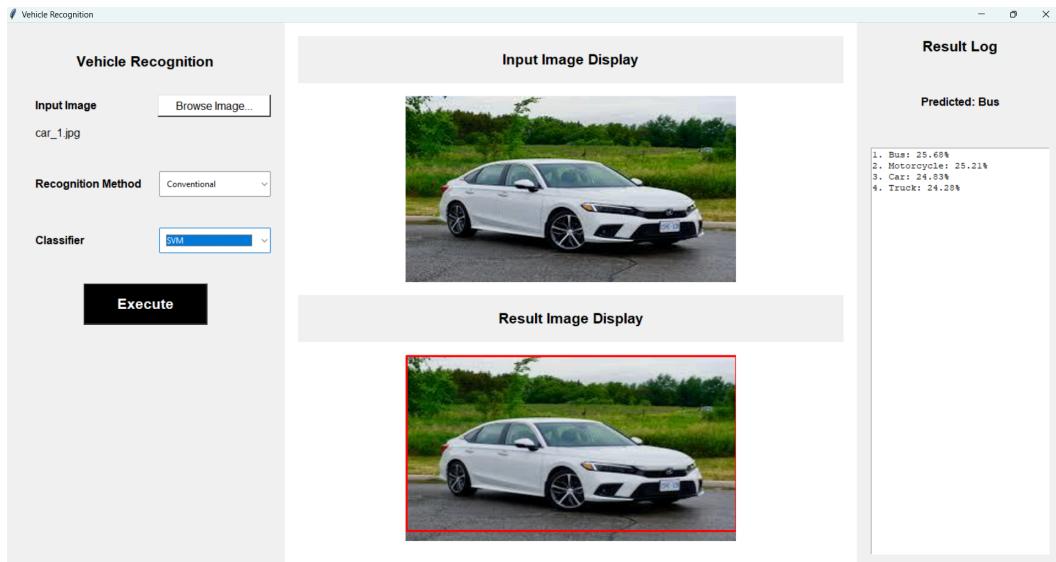
print("Train Loss: ", train_score[0])
print("Train Accuracy: ", train_score[1])
print('-' * 20)
print("Validation Loss: ", valid_score[0])
print("Validation Accuracy: ", valid_score[1])
print('-' * 20)
print("Test Loss: ", test_score[0])
print("Test Accuracy: ", test_score[1])

```

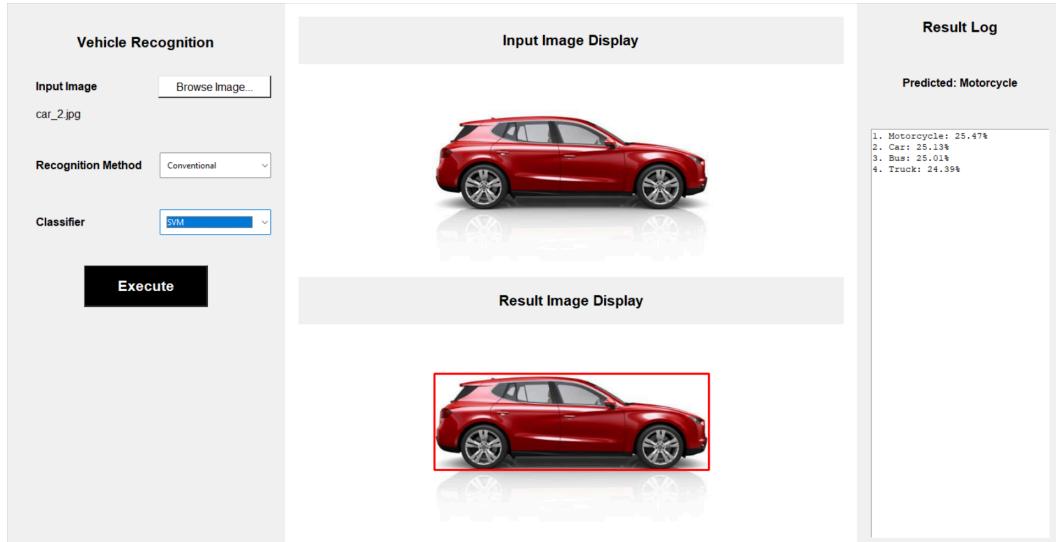
4. Hasil Eksekusi Program

4.1. Metode Konvensional

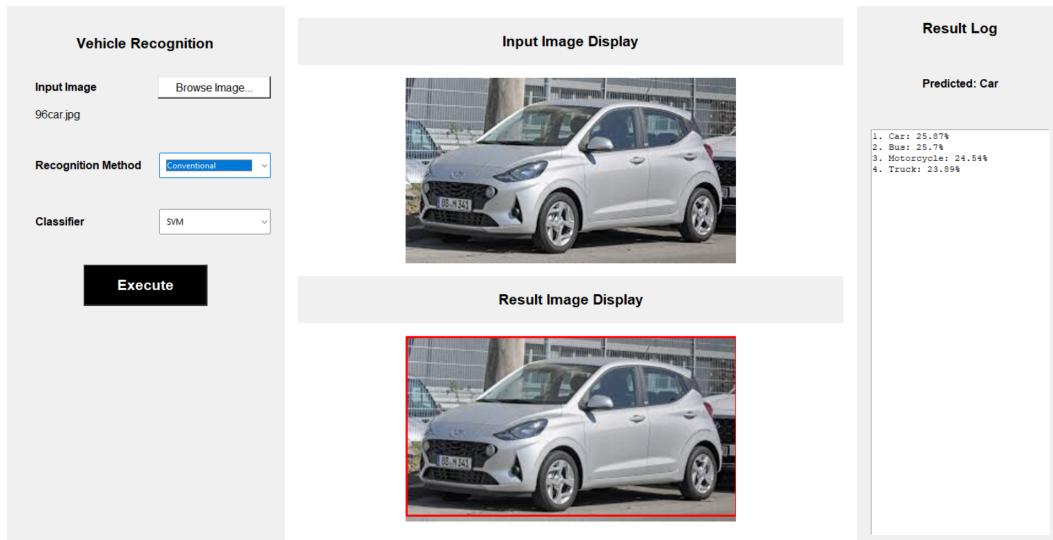
Berikut adalah hasil eksekusi program dengan metode konvensional.



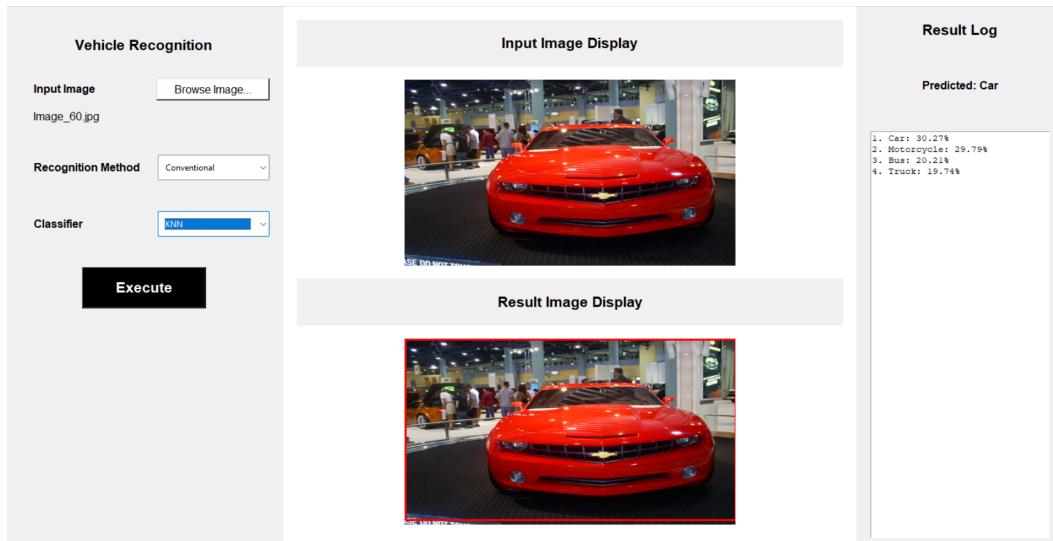
Gambar 4.1. Hasil eksekusi program untuk citra mobil dengan metode konvensional SVM (1)



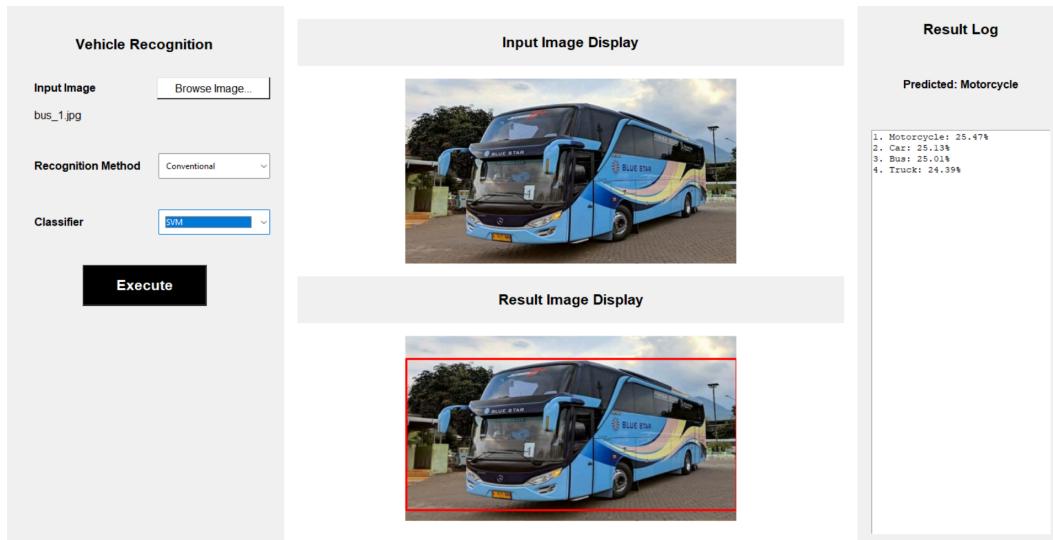
Gambar 4.2. Hasil eksekusi program untuk citra mobil dengan metode konvensional SVM (2)



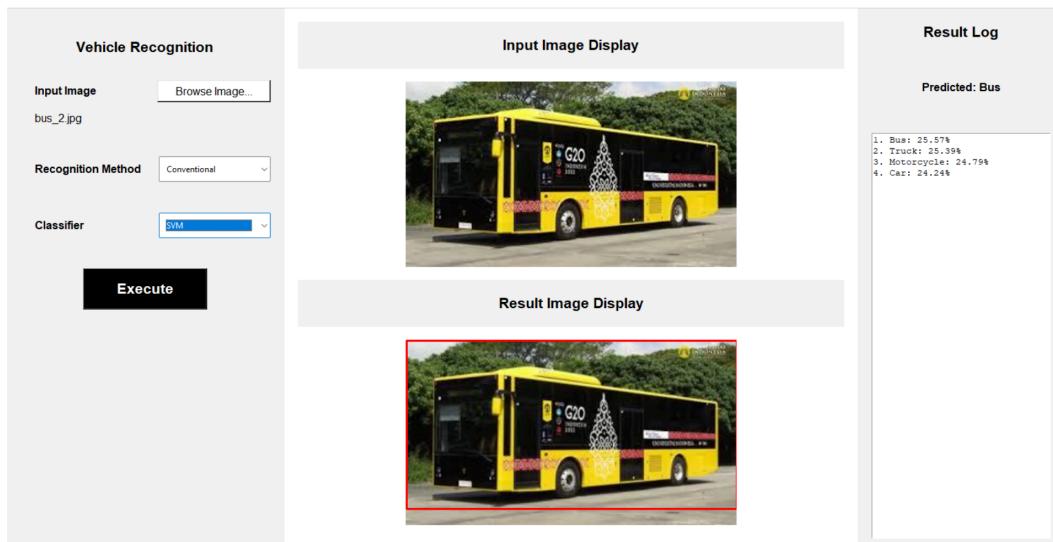
Gambar 4.3. Hasil eksekusi program untuk citra mobil dengan metode konvensional SVM (3)



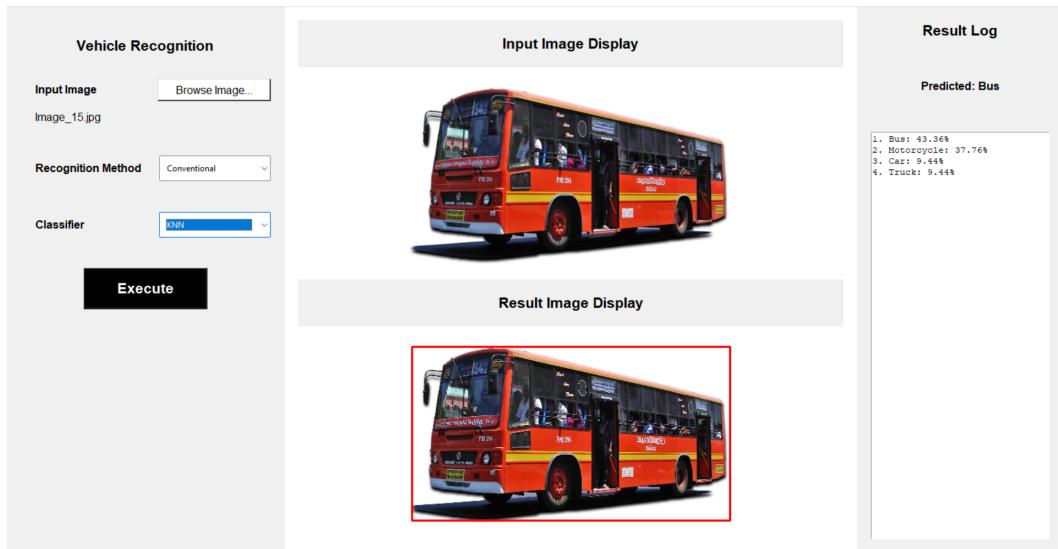
Gambar 4.4. Hasil eksekusi program untuk citra mobil dengan metode konvensional KNN (4)



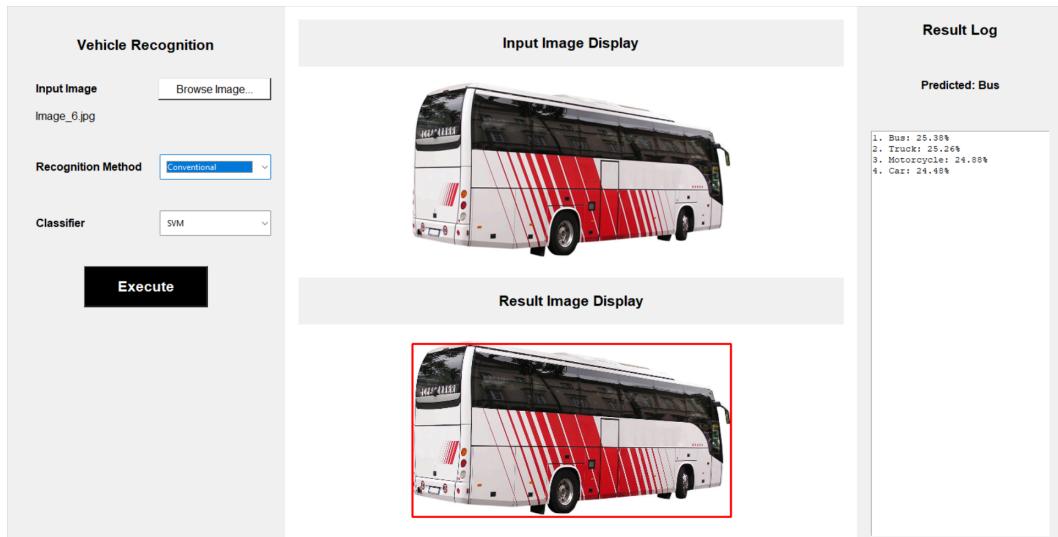
Gambar 4.5. Hasil eksekusi program untuk citra bus dengan metode konvensional SVM (1)



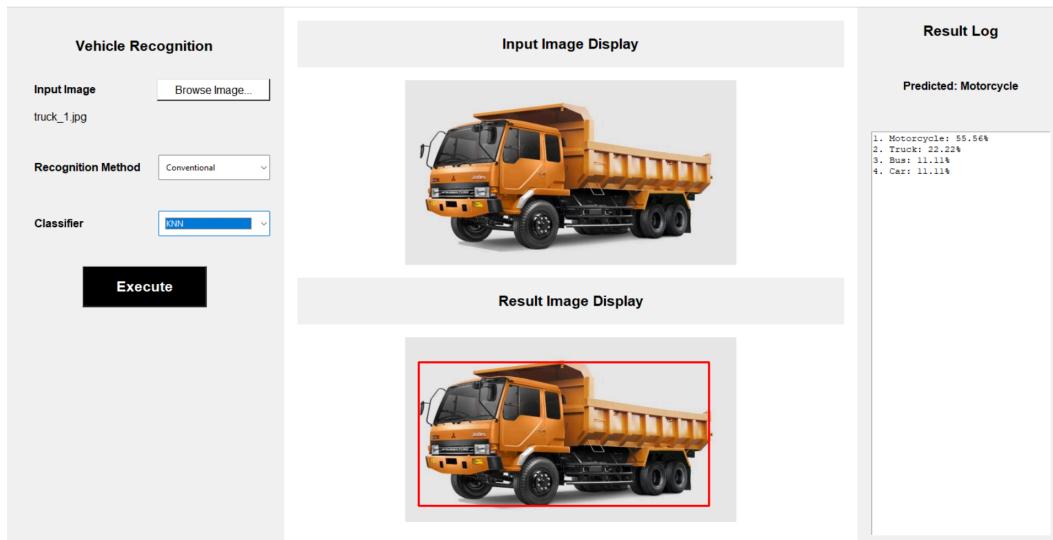
Gambar 4.6. Hasil eksekusi program untuk citra bus dengan metode konvensional SVM (2)



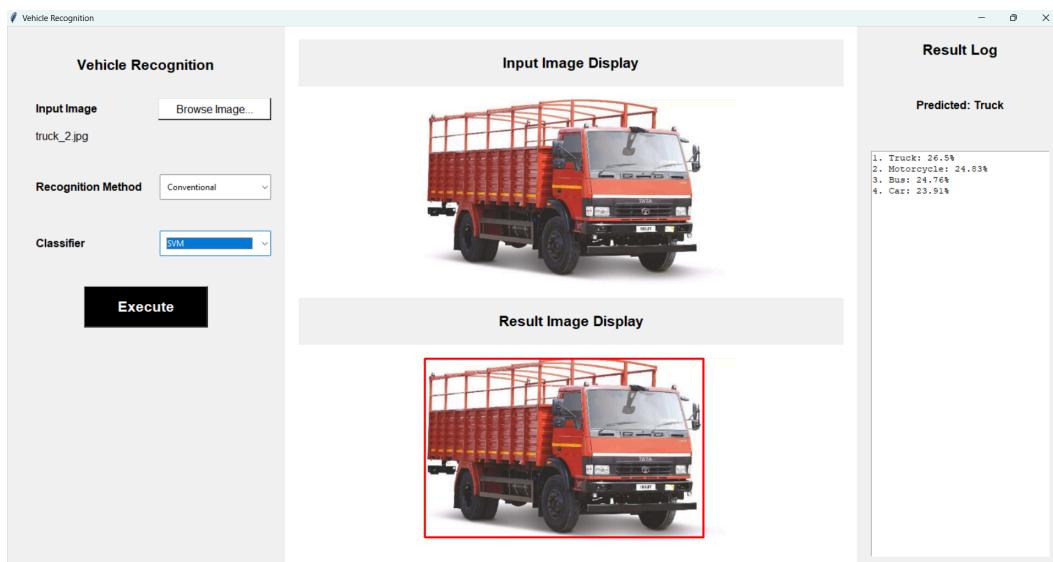
Gambar 4.7. Hasil eksekusi program untuk citra bus dengan metode konvensional KNN (3)



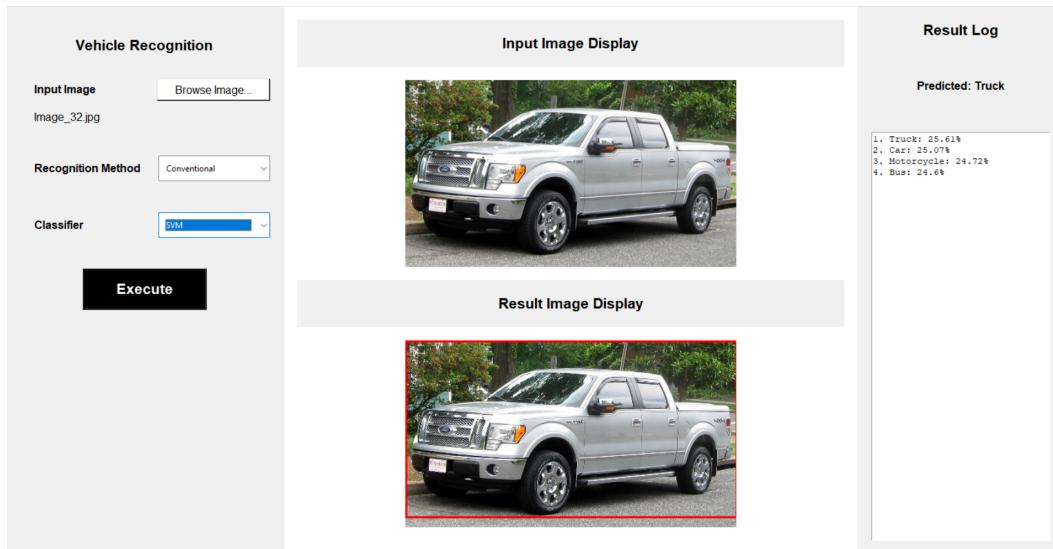
Gambar 4.8. Hasil eksekusi program untuk citra bus dengan metode konvensional SVM (4)



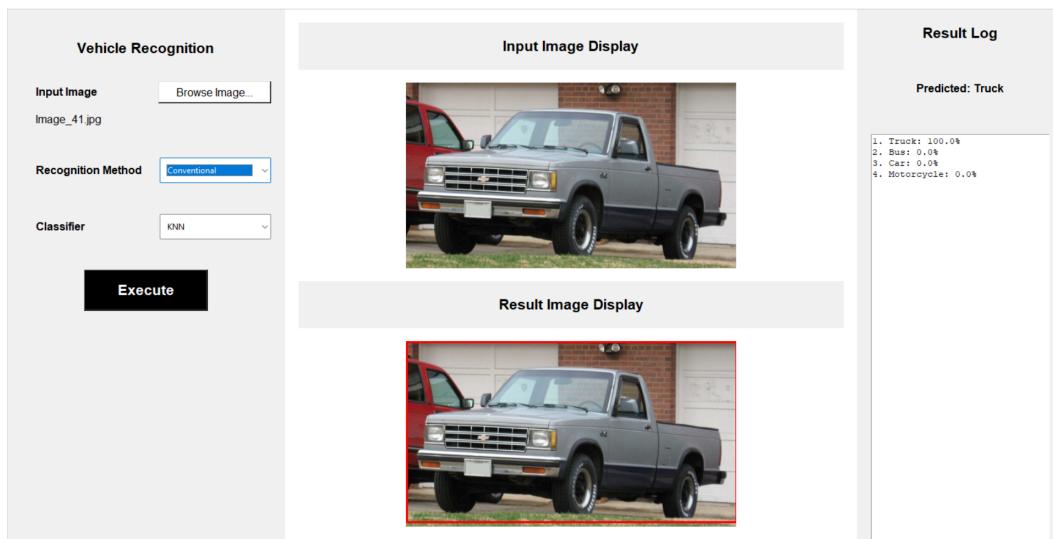
Gambar 4.9. Hasil eksekusi program untuk citra truk dengan metode konvensional KNN (1)



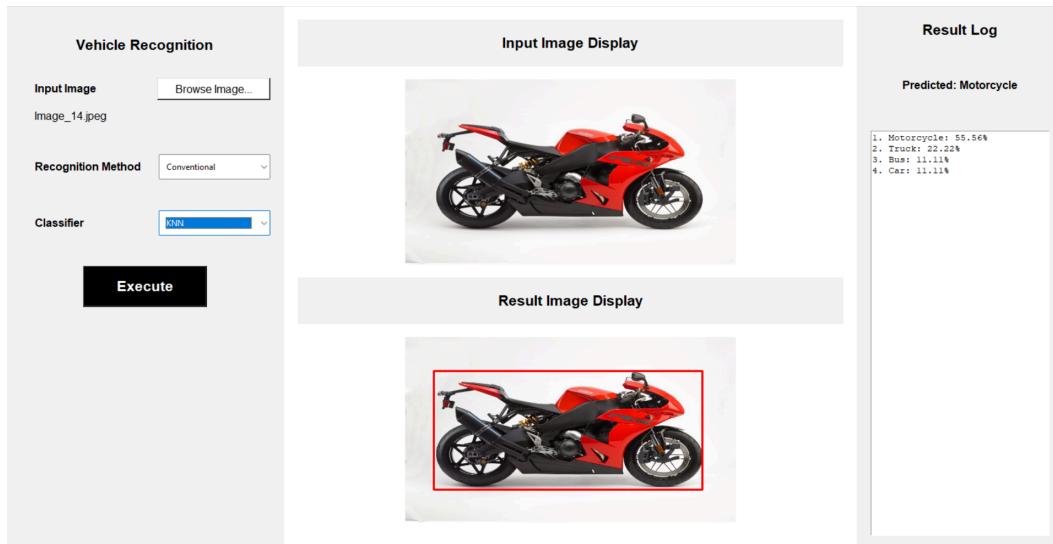
Gambar 4.10. Hasil eksekusi program untuk citra truk dengan metode konvensional SVM (2)



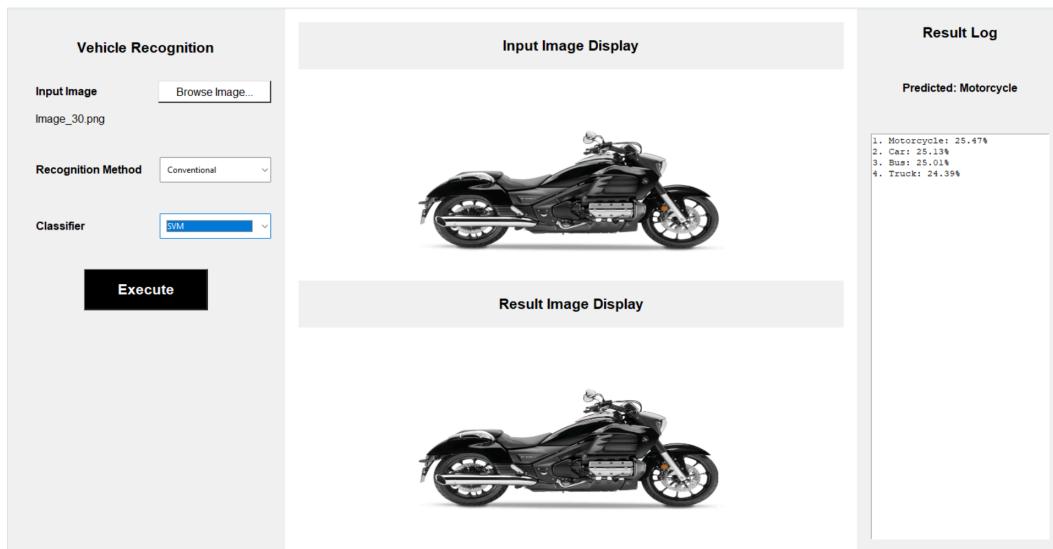
Gambar 4.11. Hasil eksekusi program untuk citra truk dengan metode konvensional SVM (3)



Gambar 4.12. Hasil eksekusi program untuk citra truk dengan metode konvensional KNN (4)



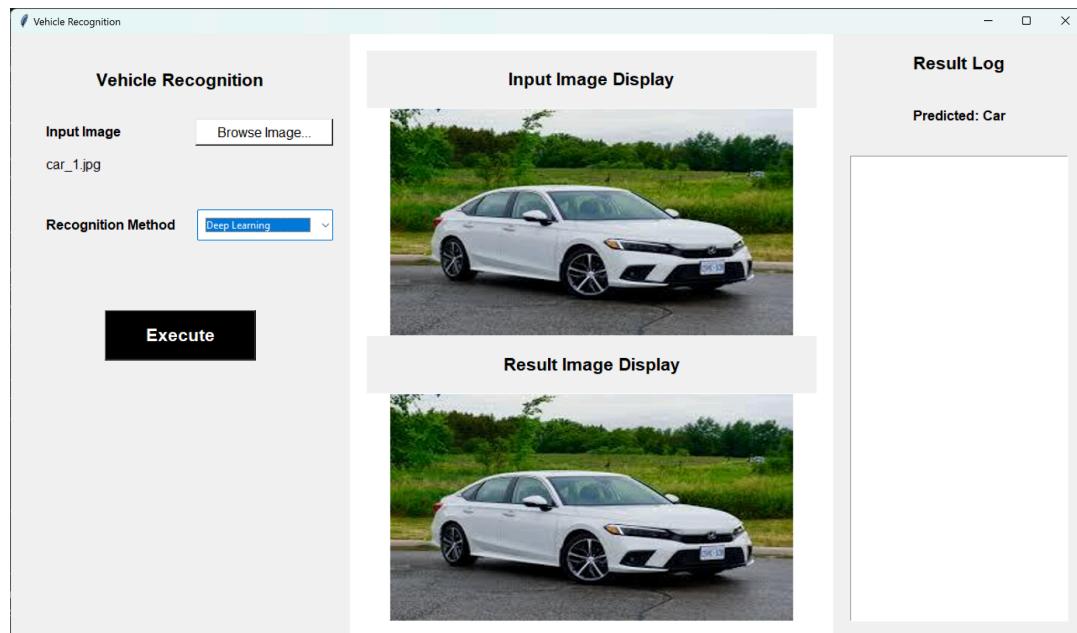
Gambar 4.13. Hasil eksekusi program untuk citra motor dengan metode konvensional KNN (1)



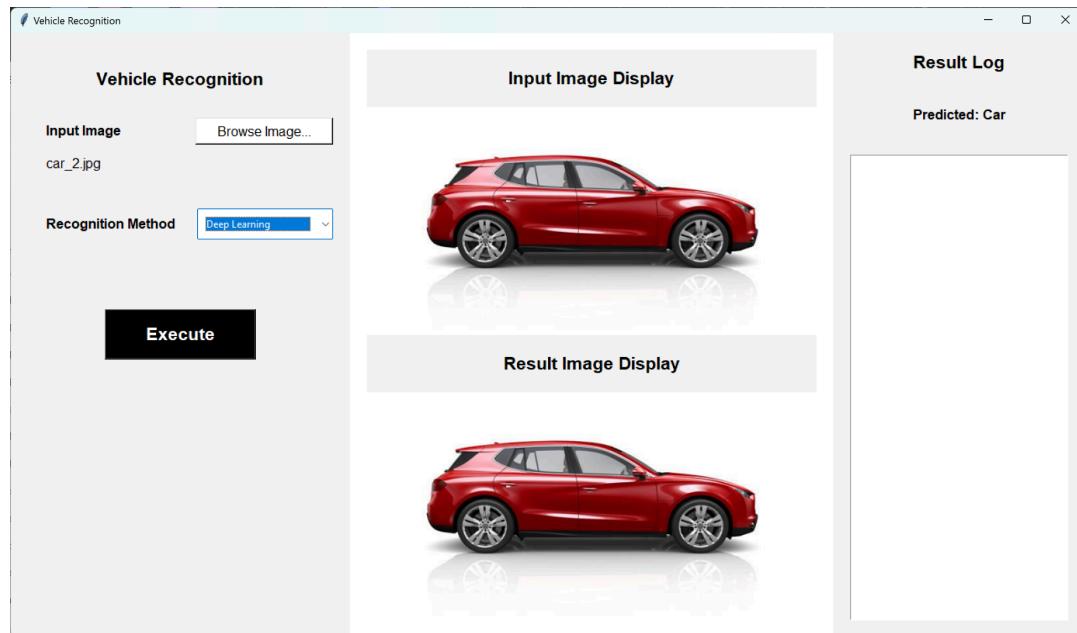
Gambar 4.14. Hasil eksekusi program untuk citra motor dengan metode konvensional SVM(2)

4.2. Pembelajaran Mendalam

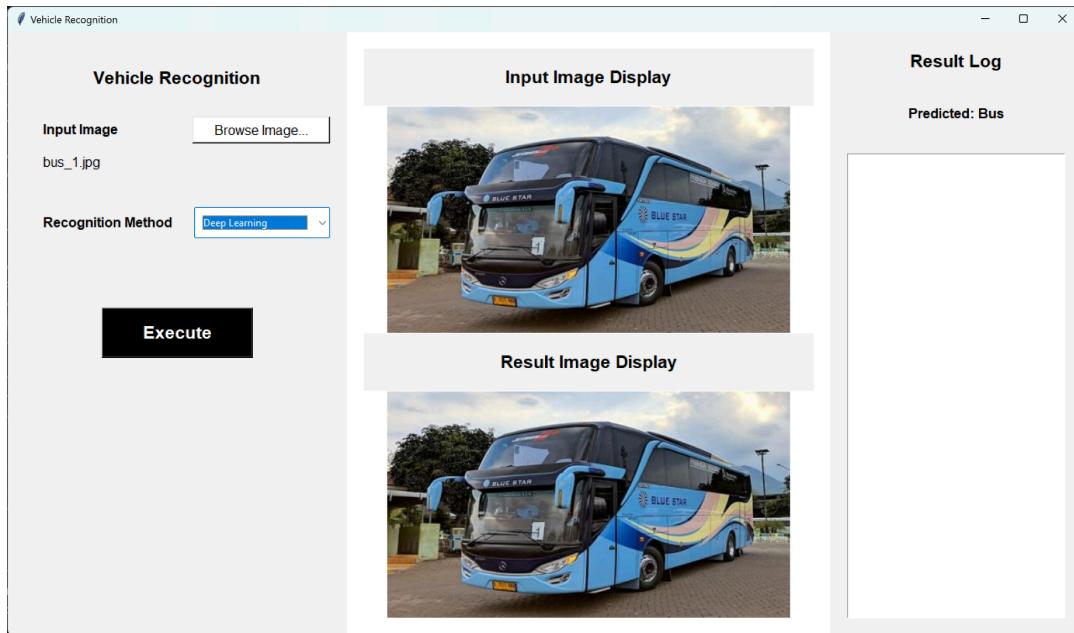
Berikut adalah hasil eksekusi program dengan pembelajaran mendalam.



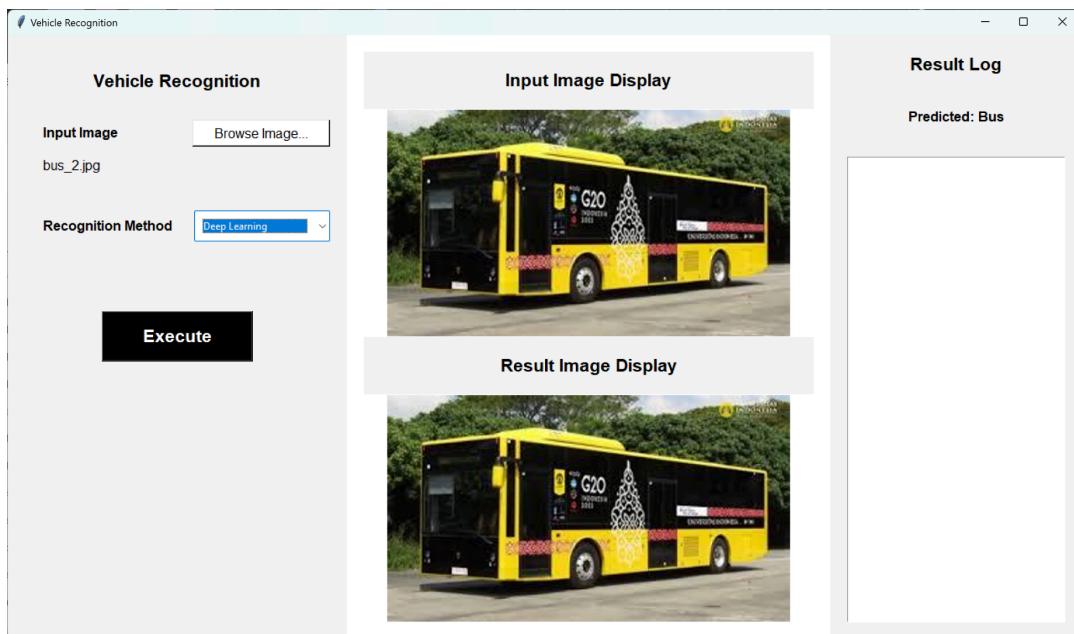
Gambar 4.15. Hasil eksekusi program untuk citra mobil dengan pembelajaran mendalam (1)



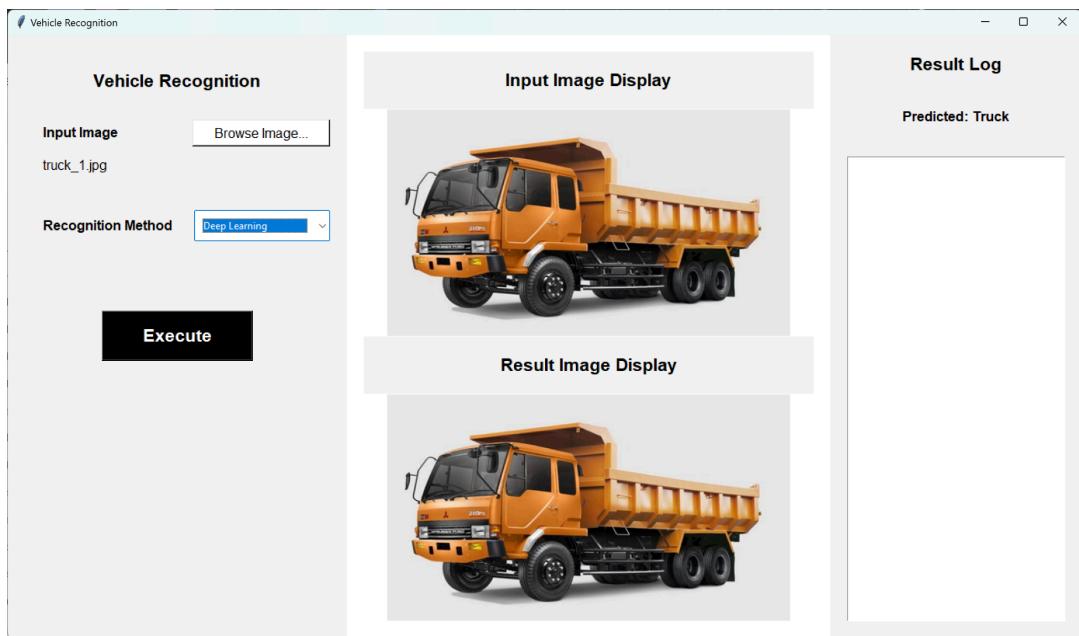
Gambar 4.16. Hasil eksekusi program untuk citra mobil dengan pembelajaran mendalam (2)



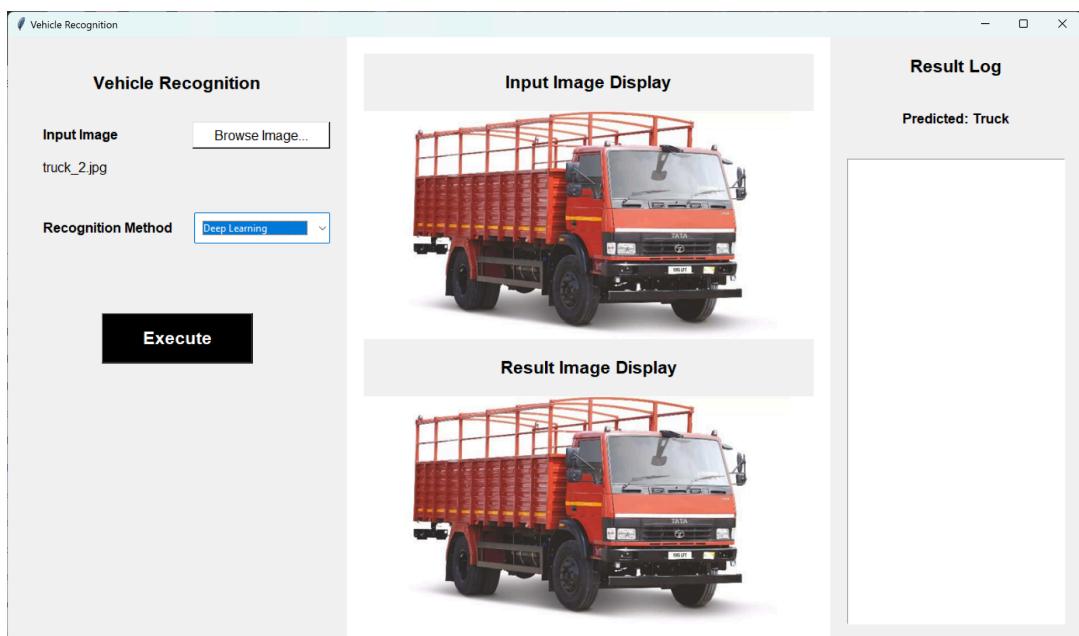
Gambar 4.17. Hasil eksekusi program untuk citra bus dengan pembelajaran mendalam (1)



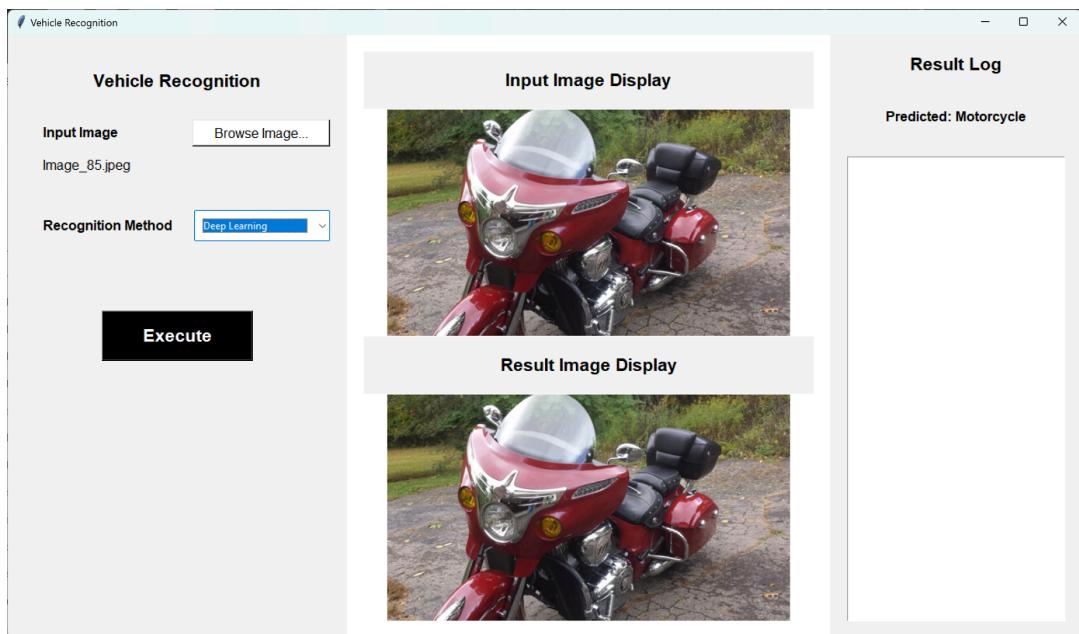
Gambar 4.18. Hasil eksekusi program untuk citra bus dengan pembelajaran mendalam (2)



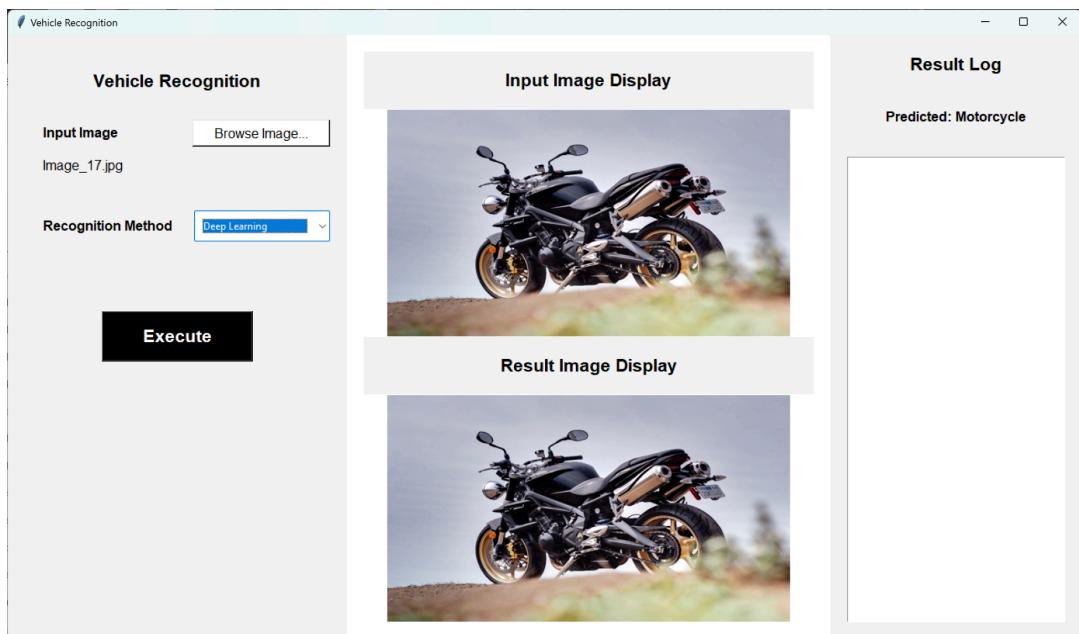
Gambar 4.19. Hasil eksekusi program untuk citra truk dengan pembelajaran mendalam (1)



Gambar 4.20. Hasil eksekusi program untuk citra truk dengan pembelajaran mendalam (2)



Gambar 4.21. Hasil eksekusi program untuk citra sepeda motor dengan pembelajaran mendalam (1)



Gambar 4.22. Hasil eksekusi program untuk citra sepeda motor dengan pembelajaran mendalam (2)

5. Diskusi dan Analisis Hasil Eksekusi Program

Pengenalan kendaraan dengan pembelajaran konvensional masih memberikan hasil yang belum terlalu akurat secara umum dengan masih banyaknya prediksi kesalahan yang ada dalam mengidentifikasi kesesuaian terhadap masukan kendaraan. Hal ini dapat terjadi

dikarenakan sejumlah faktor seperti pengenalan pola yang masih kurang optimal, pengaturan *hyperparameter tuning* pembelajaran yang kurang tepat, hingga faktor perbedaan jumlah *dataset* dari kendaraan sehingga menyebabkan pembelajaran untuk pembentukan model menjadi tidak maksimal dan menyebabkan kesalahan prediksi.

Berdasarkan hasil dari *classification report* yang didapat sebelumnya juga menunjukkan bahwa nilai *precision*, *recall*, dan *f1-score* untuk semua jenis label kendaraan masih rendah. Hal ini juga yang menyebabkan hasil akurasi dari prediksi kendaraan masih belum bagus dan hanya berkisar pada 30% saja. Ini menunjukkan pelatihan model secara konvensional masih kurang baik dalam memberikan hasil prediksi untuk kendaraan. Selain itu, pembelajaran dengan metode konvensional menghasilkan prediksi yang lebih baik dengan metode SVM jika dibandingkan dengan metode KNN. Hal ini dapat terjadi karena metode SVM lebih mampu menangani data dengan dimensi lebih tinggi serta memiliki kemampuan generalisasi yang lebih baik.

Sementara itu, pengenalan kendaraan dengan pembelajaran mendalam menghasilkan prediksi yang lebih akurat jika dibandingkan dengan metode konvensional. Hal ini disebabkan karena pembelajaran mendalam dapat menangkap pola yang lebih kompleks, sehingga lebih efektif dalam mengidentifikasi dan membedakan berbagai jenis kendaraan. Selain itu, arsitektur EfficientNet untuk model dipilih karena optimal dalam memanfaatkan sumber daya komputasi, sehingga menghasilkan model yang efisien tanpa mengorbankan performa.

Gambar 5.1 menunjukkan akurasi dan *loss* model dengan arsitektur CNN (EfficientNet). Kinerja model dipengaruhi oleh beberapa faktor, seperti kualitas data pelatihan, teknik augmentasi data, serta parameter dan hiperparameter yang digunakan selama pelatihan. Dengan demikian, pembelajaran mendalam memberikan solusi yang lebih akurat untuk pengenalan jenis kendaraan dibandingkan dengan metode konvensional.

```
1/1 ━━━━━━━━━━ 14s 14s/step - accuracy: 0.5000 - loss: 7.3269
1/1 ━━━━━━ 3s 3s/step - accuracy: 0.4375 - loss: 7.1751
1/1 ━━━━━━ 6s 6s/step - accuracy: 0.6250 - loss: 9.1115
Train Loss: 7.326911449432373
Train Accuracy: 0.5
-----
Validation Loss: 7.175079345703125
Validation Accuracy: 0.4375
-----
Test Loss: 9.11153793334961
Test Accuracy: 0.625
```

Gambar 5.1. Evaluasi model pembelajaran mendalam

6. Kesimpulan

Berdasarkan pengeroaan tugas ini, terdapat beberapa kesimpulan yang didapatkan antara lain sebagai berikut:

1. Sistem pengenalan kendaraan yang menggunakan pembelajaran mendalam dengan arsitektur CNN memberikan hasil yang lebih akurat jika dibandingkan dengan sistem pengenalan kendaraan dengan metode konvensional.
2. Secara umum, pembelajaran sistem pengenalan kendaraan dengan metode konvensional SVM memberikan hasil yang lebih akurat jika dibandingkan dengan metode KNN.
3. Nilai akurasi model sistem pengenalan kendaraan dengan metode konvensional dapat dipengaruhi oleh banyak faktor, seperti ukuran jumlah *dataset*, pengaturan *hyperparameter tuning*, hingga optimasi pra-pemrosesan untuk pengenalan pola pada gambar.

7. Komentar dan Refleksi

Pengerjaan tugas ini memberikan pembelajaran baru mengenai sistem pengenalan objek menggunakan metode konvensional dan pembelajaran mendalam (CNN) yang lebih modern. Terdapat beberapa ilmu baru yang didapatkan, seperti pemrosesan citra untuk pengenalan pola, pelatihan model dengan menggunakan pembelajaran mesin, hingga pemanfaatan metode pembelajaran mendalam, khususnya arsitektur CNN, untuk pengenalan pola yang lebih baik. Pengeroaan tugas ini memberikan pemahaman tentang algoritma dan metode yang digunakan dalam pengenalan objek, sekaligus menjadi motivasi untuk terus mendalami berbagai pendekatan canggih dalam bidang ini. Harapan

untuk ke depannya adalah agar dapat mengembangkan kemampuan lebih jauh lagi untuk mengimplementasikan teknologi pengenalan objek yang inovatif dan relevan di masa yang akan datang.

8. Referensi

1. *Dataset (1):*
<https://www.kaggle.com/code/kaggleashwin/dataset-collection-for-vehicle-type-recognition>
2. *Dataset (2):*
<https://www.kaggle.com/datasets/rohan300557/vehicle-detection>
3. *Pre-trained model:*
<https://www.kaggle.com/code/eyadgk/vehicle-rec-via-efficientnet-100-acc-full-guide/notebook>
4. <https://www.geeksforgeeks.org/hog-feature-visualization-in-python-using-skimage/>

9. Lampiran

Tautan repositori GitHub: <https://github.com/arleenchr/VehicleRecognition>