UNIVERSITI KUALA LUMPUR

CITY CAMPUS

MALAYSIAN INSTITUTE

OF INFORMATION TECHNOLOGY

Name of Course	**PRINCIPLES OF ARTIFICIAL INTELLIGENCE**
Course Code	**ISB46703**
Lecturer	**Dr Zhafri Roslan**
Student Name and ID	Muhammad Dinie Ikhsan Bin Abdul Najib (52213224085) Muhammad Azrie Bin Adnan (52213224334)
Semester / Year	**Oct 2025**
Distribution Date	**5th January 2026**
Presentation Date	**19th January 2026**
Report Due Date	**26th January 2026**
Assessment	**Project – 3 or 4 students per group**

PART A

Group Members:

No.	Group Members	Member Contribution
1.	Muhammad Dinie Ikhsan Bin Abdul Najib	**Data Analyst**
2.	Muhammad Azrie Bin Adnan	**Data Scientist & Engineer**

Introduction

Artificial Intelligence (AI) enables machines to simulate human intelligence, particularly in tasks involving perception, learning, and decision-making. One of the most widely adopted AI techniques for visual data analysis is Convolutional Neural Networks (CNNs), which are capable of automatically extracting meaningful features from images.

In this project, CNN-based deep learning models are applied to solve an image classification problem using a real-world dataset. The project demonstrates the use of transfer learning, model training, evaluation, and visualization techniques. This work aligns with the objectives of the Principle of Artificial Intelligence course by applying intelligent systems to achieve classification goals effectively.

1.0 Dataset Description

The dataset used in this project consists of image data organised into multiple classes. The images were prepared in a directory-based structure suitable for supervised learning using Keras image generators.

Dataset Characteristics:

- Image format: RGB images
- Image size: 224 × 224 pixels
- Dataset split:
  - Training set
  - Validation set
  - Testing set
- Class labels are automatically inferred from directory names

Image preprocessing was applied using rescaling to normalise pixel values between 0 and 1. This ensures stable and efficient model training.

2.0 Data Preparation

Data preparation was performed using TensorFlow's ImageDataGenerator. The following steps were applied:

1. Images were loaded from directory-based folders.
2. All images were resized to 224 × 224 pixels.
3. Pixel values were rescaled using a factor of 1/255.
4. The dataset was split into training, validation, and testing subsets.
5. Categorical labels were automatically encoded for multi-class classification.

This preparation ensures consistency and compatibility with pre-trained CNN architectures.

3.0 Model Architecture and Methodology

Three Convolutional Neural Network models were implemented using **transfer learning**, where pre-trained weights from ImageNet were used.

3.1 Models Implemented

1. ResNet50
2. DenseNet121
3. MobileNetV3

For each model:

- The pre-trained convolutional base was used as a feature extractor.
- The top layers were excluded.
- A Global Average Pooling layer was added.
- Fully connected layers were appended.
- A Softmax output layer was used for multi-class classification.

4.0 Model Training

Each CNN model was trained for 50 epochs using the following configuration:

- Optimizer: Adam
- Loss Function: Categorical Crossentropy
- Batch Size: 32
- Metrics: Accuracy

The training process recorded:

- Training accuracy
- Validation accuracy
- Training loss
- Validation loss
- Total training time for each model

Transfer learning was applied by freezing the convolutional base layers during initial training to prevent overfitting and reduce computational cost.

5.0 Performance Evaluation

5.1 Accuracy and Loss Analysis

For each model, graphs were generated to visualise:

- Training vs Validation Accuracy
- Training vs Validation Loss

These plots were used to analyse learning behaviour, convergence speed, and potential overfitting.

5.2 Confusion Matrix

The trained models were evaluated on the testing dataset using a confusion matrix. This matrix provides a detailed view of:

- Correct classifications
- Misclassifications between similar classes

The confusion matrix allows for class-wise performance analysis rather than relying solely on accuracy.

5.3 Mean Average Precision (mAP)

In addition to accuracy, **Mean Average Precision (mAP)** was calculated to measure the model's ability to correctly rank predictions across all classes. This metric is particularly useful for multi-class classification problems as it provides a more balanced evaluation.

6.0Model Comparison

Model	**Perfomance**	**Training Time**	**Model Size**
ResNet50	High accuracy and stable learning	Moderate	Large
DenseNet121	Highest Accuracy and mAP	Long	Very Large
MobileNetV3	Fastest Training	Short	Lightweight

7.0 Discussion

The results indicate that deeper models such as DenseNet121 are capable of extracting more detailed features, leading to higher accuracy and mAP. However, this comes at the cost of increased training time and computational complexity.

ResNet50 provides a good balance between performance and efficiency, making it suitable for general-purpose classification tasks.

MobileNetV3, although slightly lower in accuracy, demonstrates excellent efficiency and is ideal for deployment on mobile or resource-constrained devices.

Conclusion

Based on the experimental evaluation, **DenseNet121** emerged as the best-performing model in terms of classification accuracy and mean average precision. However, **MobileNetV3** demonstrated superior efficiency with significantly reduced training time and model complexity.

Therefore:

- **DenseNet121** is most suitable when accuracy is the priority.
- **MobileNetV3** is preferable for real-time or low-resource environments.
- **ResNet50** serves as a balanced alternative.

This project successfully demonstrates the application of intelligent systems using CNNs and fulfills all the required course learning outcomes.

CODING

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
import time
```

OUTPUT

```
Found 433 images belonging to 6 classes.
Found 92 images belonging to 6 classes.
Found 94 images belonging to 6 classes.
Number of classes: 6
```

CODING

```
base_model = tf.keras.applications.ResNet50(
    weights="imagenet",
    include_top=False,
    input_shape=(224,224,3)
)

base_model.trainable = False  # Transfer learning

x = base_model.output
x = GlobalAveragePooling2D()(x)
output = Dense(NUM_CLASSES, activation="softmax")(x)

model_resnet = Model(inputs=base_model.input, outputs=output)

model_resnet.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

CODING

```
start_time = time.time()

history_resnet = model_resnet.fit(
    train_data,
    validation_data=val_data,
    epochs=50
)

resnet_time = time.time() - start_time
print(f"⏱️ ResNet50 training time: {resnet_time/60:.2f} minutes")
```

OUTPUT

```
Epoch 1/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 30s 1s/step - accuracy: 0.2112 - loss: 1.8404 - val_accuracy: 0.1848 - val_loss: 1.7697
Epoch 2/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 3s 181ms/step - accuracy: 0.1862 - loss: 1.7705 - val_accuracy: 0.2283 - val_loss: 1.7231
Epoch 3/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 168ms/step - accuracy: 0.2090 - loss: 1.7313 - val_accuracy: 0.2391 - val_loss: 1.7062
Epoch 4/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 169ms/step - accuracy: 0.1757 - loss: 1.7230 - val_accuracy: 0.2283 - val_loss: 1.6950
Epoch 5/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 158ms/step - accuracy: 0.2321 - loss: 1.6818 - val_accuracy: 0.2609 - val_loss: 1.6757
Epoch 47/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 158ms/step - accuracy: 0.4888 - loss: 1.4410 - val_accuracy: 0.4130 - val_loss: 1.5977
Epoch 48/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 163ms/step - accuracy: 0.4152 - loss: 1.4633 - val_accuracy: 0.4239 - val_loss: 1.5844
Epoch 49/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 3s 194ms/step - accuracy: 0.5084 - loss: 1.4081 - val_accuracy: 0.4130 - val_loss: 1.6020
Epoch 50/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 3s 174ms/step - accuracy: 0.4603 - loss: 1.4316 - val_accuracy: 0.3804 - val_loss: 1.5886
⏱️ ResNet50 training time: 2.56 minutes
```

CODING

```
start_time = time.time()

history_densenet = model_densenet.fit(
    train_data,
    validation_data=val_data,
    epochs=50
)

densenet_time = time.time() - start_time
print(f"⏱️ DenseNet121 training time: {densenet_time/60:.2f} minutes")
```

OUTPUT

```
Epoch 1/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 70s 3s/step - accuracy: 0.2062 - loss: 1.9565 - val_accuracy: 0.3261 - val_loss: 1.6982
Epoch 2/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 159ms/step - accuracy: 0.4707 - loss: 1.3867 - val_accuracy: 0.4783 - val_loss: 1.3148
Epoch 3/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 170ms/step - accuracy: 0.5585 - loss: 1.0580 - val_accuracy: 0.5326 - val_loss: 1.1675
Epoch 4/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 3s 185ms/step - accuracy: 0.6559 - loss: 0.9022 - val_accuracy: 0.5109 - val_loss: 1.1029
Epoch 5/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 3s 201ms/step - accuracy: 0.6955 - loss: 0.7899 - val_accuracy: 0.5761 - val_loss: 1.0686
Epoch 45/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 161ms/step - accuracy: 0.9533 - loss: 0.2179 - val_accuracy: 0.6087 - val_loss: 0.9903
Epoch 46/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 163ms/step - accuracy: 0.9641 - loss: 0.2064 - val_accuracy: 0.5978 - val_loss: 0.9819
Epoch 47/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 168ms/step - accuracy: 0.9600 - loss: 0.2127 - val_accuracy: 0.5870 - val_loss: 0.9837
Epoch 48/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 167ms/step - accuracy: 0.9540 - loss: 0.2030 - val_accuracy: 0.6196 - val_loss: 1.0074
Epoch 49/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 3s 207ms/step - accuracy: 0.9676 - loss: 0.1900 - val_accuracy: 0.5761 - val_loss: 0.9897
Epoch 50/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 154ms/step - accuracy: 0.9530 - loss: 0.1845 - val_accuracy: 0.6087 - val_loss: 1.0006
⏱️ DenseNet121 training time: 3.16 minutes
```

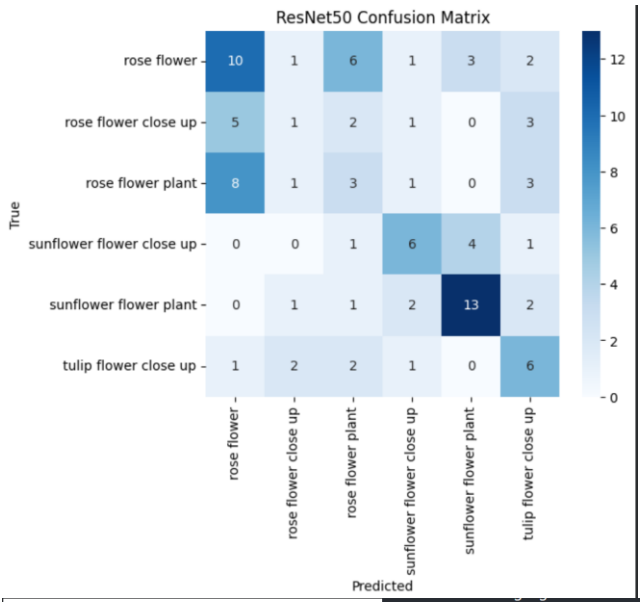
CODING

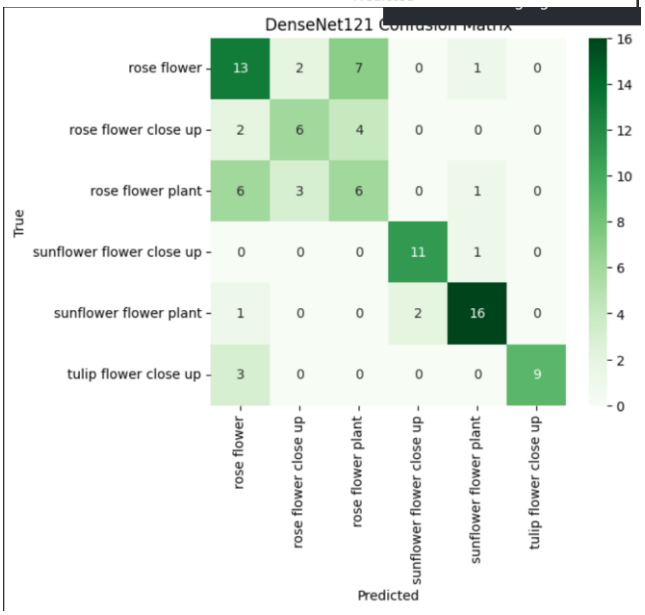
```python
start_time = time.time()

history_mobilenet = model_mobilenet.fit(
    train_data,
    validation_data=val_data,
    epochs=50
)

mobilenet_time = time.time() - start_time
print(f"⏱️ MobileNetV3 training time: {mobilenet_time/60:.2f} minutes")
```

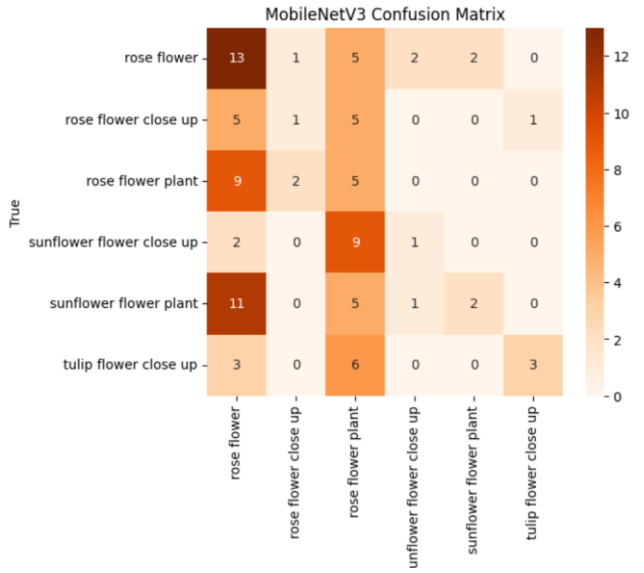
OUTPUT

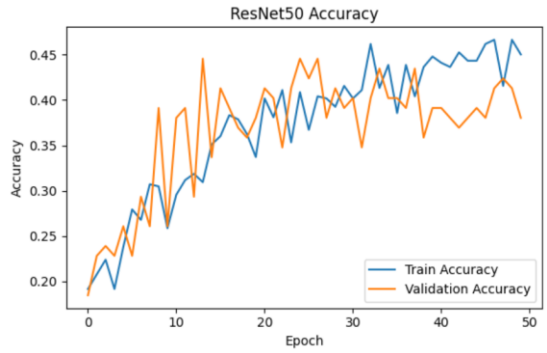
```
Epoch 1/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 47s 2s/step - accuracy: 0.1231 - loss: 2.0039 - val_accuracy: 0.1522 - val_loss: 1.7901
Epoch 2/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 3s 193ms/step - accuracy: 0.1834 - loss: 1.7768 - val_accuracy: 0.2391 - val_loss: 1.7675
Epoch 3/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 166ms/step - accuracy: 0.2194 - loss: 1.7581 - val_accuracy: 0.2391 - val_loss: 1.7619
Epoch 4/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 153ms/step - accuracy: 0.2348 - loss: 1.7503 - val_accuracy: 0.2283 - val_loss: 1.7516
Epoch 5/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 154ms/step - accuracy: 0.2358 - loss: 1.7459 - val_accuracy: 0.2391 - val_loss: 1.7470
Epoch 45/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 153ms/step - accuracy: 0.3126 - loss: 1.6164 - val_accuracy: 0.2609 - val_loss: 1.6458
Epoch 46/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 150ms/step - accuracy: 0.3210 - loss: 1.6152 - val_accuracy: 0.2826 - val_loss: 1.6402
Epoch 47/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 170ms/step - accuracy: 0.3094 - loss: 1.6061 - val_accuracy: 0.2935 - val_loss: 1.6444
Epoch 48/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.3080 - loss: 1.6194 - val_accuracy: 0.2935 - val_loss: 1.6393
Epoch 49/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 149ms/step - accuracy: 0.3567 - loss: 1.5907 - val_accuracy: 0.3261 - val_loss: 1.6344
Epoch 50/50
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 147ms/step - accuracy: 0.3978 - loss: 1.5858 - val_accuracy: 0.3152 - val_loss: 1.6396
⏱️ MobileNetV3 training time: 2.68 minutes
```

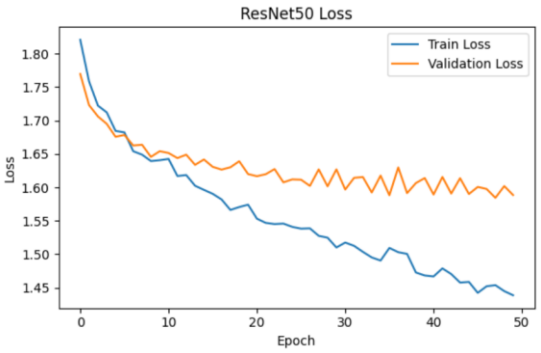
OUTPUT

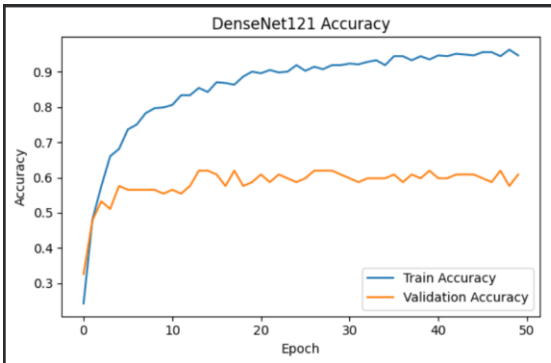

ResNet50 Confusion Matrix
rose flower
rose flower close up
rose flower plant
sunflower flower close up
sunflower flower plant
tulip flower close up
True
Predicted

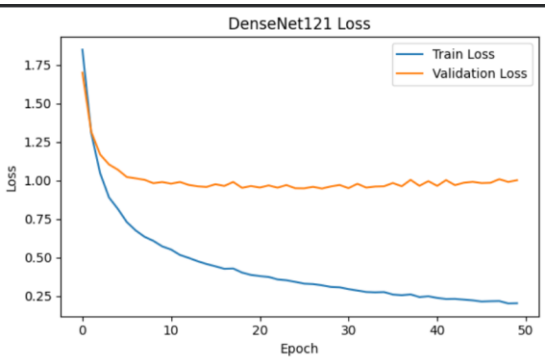

rose flower
rose flower close up
rose flower plant
sunflower flower close up
sunflower flower plant
tulip flower close up
True
Predicted

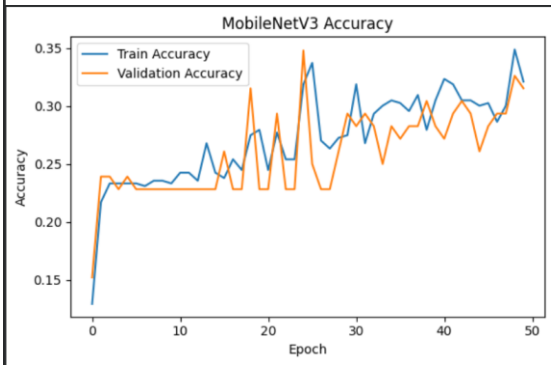

MobileNetV3 Confusion Matrix
rose flower
rose flower close up
rose flower plant
unflower flower close up
sunflower flower plant
tulip flower close up
True

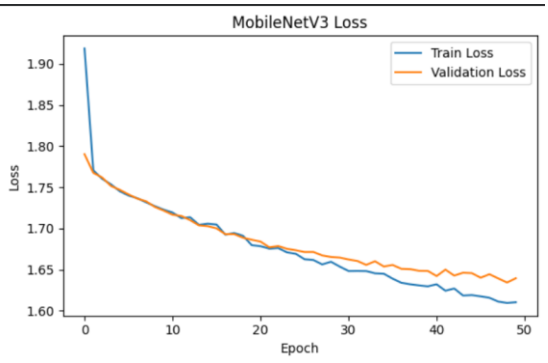

ResNet50 Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy


ResNet50 Loss
Loss
Epoch
Train Loss
Validation Loss


DenseNet121 Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy


DenseNet121 Loss
Loss
Epoch
Train Loss
Validation Loss


MobileNetV3 Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy


MobileNetV3 Loss
Loss
Epoch
Train Loss
Validation Loss