FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

B. Tech CSE-AI

SUBJECT : ADVANCE MACHINE LEARNING

SUBJECT CODE : (23CSE514)

MINI PROJECT

WILDLIFE MONITORING SYSTEM WITH IMAGE CLASSIFICATION

Submitted to,
Prof:- Ms. Guruvammal S
Assistant Professor,
Department of Computer Science & Engineering,
Faculty of Engineering & Technology,
Jain (Deemed-To-Be) University.

Submitted by,

Name : SHASHIDHAR B

USN : 23BTRCA049

Branch & Section : CSE- AI

Date of Submission : 22nd November 2025

INDEX

Colab link:https://colab.research.google.com/drive/1LJqTpBNcLzzdZN0Hla2_ZFGo51Yn4v55?usp=sharing

INTRODUCTION

The goal of this mini-project is to build an Animal Detection and Classification System capable of identifying 90 different animal species using a deep-learning image classification model. With the rapid growth of machine learning and computer vision, automated animal recognition has become an important area of research across multiple fields, including wildlife conservation, biodiversity monitoring, zoological studies, eco-tourism, smart surveillance, and educational platforms.

This project utilizes MobileNetV2, a lightweight yet powerful Convolutional Neural Network (CNN) architecture designed specifically for environments with limited computational resources. Unlike heavier models such as VGG16 or ResNet50, MobileNetV2 offers high accuracy with significantly lower memory usage, making it ideal for deployment on platforms such as Google Colab, smartphones, embedded systems, and real-time applications. Its inverted residual blocks, depthwise separable convolutions, and compact architecture allow the model to extract rich visual features while maintaining excellent computational efficiency.

The dataset used for this project is the Animal Image Dataset (90 Different Animals) from Kaggle. It consists of 90 folders, each representing a distinct animal species such as antelope, bear, bat, camel, cobra, deer, fox, kangaroo, zebra, and many more. The dataset contains images with varying lighting conditions, angles, backgrounds, and resolutions, making it a challenging and realistic benchmark for image classification tasks. These variations ensure that the model learns to generalize well rather than memorizing specific patterns.

To build the system, the pretrained MobileNetV2 model (trained initially on ImageNet — a dataset of 1.2 million images across 1,000 classes) is used as the backbone. Instead of training the entire network from scratch, which is computationally expensive and requires a very large dataset, this project leverages Transfer Learning. By freezing the lower layers of MobileNetV2 and fine-tuning higher layers, the model can efficiently learn features specific to the 90 animal classes. This technique significantly speeds up training, reduces resource requirements, and improves accuracy.

The project demonstrates the practical use of Deep Learning, CNN feature extraction, and Transfer Learning in solving real-world classification problems. It also highlights the importance of data preprocessing, augmentation, model evaluation, and fine-tuning. The final model is capable of predicting the animal species from an uploaded image and displaying the confidence score, showcasing its applicability in real-world use cases such as:

- Wildlife monitoring systems using camera traps
- Academic research in zoology and ecology
- Smart farming and livestock monitoring
- Animal identification in mobile applications
- Educational tools for students and researchers

REVIEW / PRIOR STUDY

Animal classification is a widely studied problem in computer vision. Many deep learning architectures have been explored, ranging from traditional CNNs to state-of-the-art transfer learning models.

Existing Solutions

- CNN Models: VGG, AlexNet, ResNet, DenseNet
- Transfer Learning: InceptionV3, MobileNet, EfficientNet
- Wildlife Monitoring Systems: Camera trap image classification using CNNs
- Large-scale datasets: ImageNet, iNaturalist, Wildlife-1M

Key Findings from Prior Research

- CNNs trained from scratch require very large datasets and GPU resources.
- Transfer learning significantly improves accuracy while reducing training time.
- Lightweight models like MobileNetV2 are ideal for deployment on mobile/edge devices.
- Data augmentation is essential for improving robustness and reducing overfitting.

Gaps Identified

- Many works focus on fewer classes (10–20 species).
- Limited use of lightweight architectures in academic projects.
- Lack of detailed evaluation metrics (confusion matrix, classification report).

This project addresses these gaps by:

- Classifying 90 animal species
- Using a fast, mobile-friendly model
- Providing complete evaluation (accuracy, precision, recall, F1-score)

PROBLEM STATEMENT

Identifying animal species from images is challenging due to variations in lighting, pose, background, and angles, as well as the high visual similarity between certain animals. Many datasets are imbalanced, containing inconsistent samples or low-quality images, which makes classification even more difficult. Traditional image-processing techniques cannot reliably handle these complexities because they depend on handcrafted features that fail under real-world variations.

To address these limitations, a deep learning-based approach is required. CNNs automatically learn hierarchical features, adapt to different visual conditions, and generalize better on noisy data. Transfer Learning further improves performance by using pretrained ImageNet models, reducing both training time and computational cost.

The main objective of this project is to develop a multi-class animal detection system capable of classifying images into 90 different species. The system uses MobileNetV2, a lightweight and efficient transfer learning model suited for limited hardware environments like Google Colab or mobile devices.

The project aims to:

- Build a fast and accurate deep learning model
- Achieve reliable classification across 90 animal categories
- Utilize pretrained MobileNetV2 for efficient feature extraction
- Apply data augmentation and fine-tuning to improve robustness
- Provide real-time prediction with confidence scores

This project demonstrates the practical application of CNNs and Transfer Learning in wildlife monitoring, biodiversity research, and educational tools.

DATASET DESCRIPTION

Dataset Used

Kaggle: Animal Image Dataset (90 Different Animals)
Link: https://www.kaggle.com/datasets/iamsouravbanerjee/animal-image-dataset-90-different-animals

Dataset Structure

The raw dataset was found inside nested folders:

animal90/

animals/

animals/

antelope/

badger/

bat/

...

Key Details

- Total Classes: 90
- Images per class: varies (50–400 images/class)
- File Format: JPG / PNG
- Image Sizes: Non-uniform (resized during preprocessing)
- Final Split:
    - Training: 80%
    - Validation: 10%
    - Testing: 10%

Preprocessing Performed

- Resizing to 160×160 (MobileNet input requirement)
- Normalization (pixel value scale 0–1)
- Data augmentation (rotation, zoom, flips)
- Folder-based class labeling

METHODOLOGY

The project follows these main stages:

1. Dataset Acquisition

- Downloaded using Kaggle API
- Extracted nested folder structure
- Performed manual folder path correction

2. Dataset Splitting

Using splitfolders.ratio(), dataset was split into:

Split	Percentage
Training	80%
Validation	10%
Testing	10%

3. Image Preprocessing

- Rescale pixel values
- Resize images to 160×160
- Create batches
- Shuffle training data

4. Model Architecture (MobileNetV2)

- Pretrained on ImageNet
- Base layers frozen initially
- Added:
    - GlobalAveragePooling2D
    - Dropout (0.3)
    - Dense layer (90 neurons, softmax)

5. Training Phase

- First: Train only classifier head
- Later: Unfreeze last 30 layers for fine-tuning

6. Evaluation

- Confusion Matrix
- Classification Report
- Accuracy & Loss Curves
- Misclassified Images Viewer

7. Prediction

- User uploads any animal image
- Model predicts species + confidence score

CODE IMPLEMENTATION

```
# ================================================
# STEP 1 — Install Kaggle & Upload kaggle.json
# ================================================
!pip install kaggle

from google.colab import files
files.upload()  # upload kaggle.json

!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# ================================================
# STEP 2 — Download Dataset (90 Animals)
# ================================================
!kaggle datasets download -d iamsouravbanerjee/animal-image-dataset-90-different-animals
!unzip animal-image-dataset-90-different-animals.zip -d animal90

# ================================================
# STEP 3 — Split dataset into train/val/test
# ================================================
!pip install split-folders
import splitfolders

splitfolders.ratio(
    "/content/animal90/animals/animals",   # correct path
    output="/content/animal90_split",
```

```
    seed=42,
    ratio=(0.8, 0.1, 0.1)
)

# ================================================
# STEP 4 — Create Data Generators
# ================================================
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_size = (160, 160)   # smaller = faster training
batch_size = 32

train_gen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True
)

val_gen = ImageDataGenerator(rescale=1./255)

train_ds = train_gen.flow_from_directory(
    "/content/animal90_split/train",
    target_size=img_size,
    batch_size=batch_size,
    shuffle=True
)
```

```
val_ds = val_gen.flow_from_directory(
    "/content/animal90_split/val",
    target_size=img_size,
    batch_size=batch_size
)

class_names = list(train_ds.class_indices.keys())
print("Total classes:", len(class_names))

# ================================================
# STEP 5 — Build FAST MobileNetV2 Model
# ================================================
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import layers, models
import tensorflow as tf

base = MobileNetV2(
    include_top=False,
    weights="imagenet",
    input_shape=(160,160,3)
)
base.trainable = False   # Freeze backbone

model = models.Sequential([
    base,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.3),
    layers.Dense(len(class_names), activation="softmax")
```

```
])

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

model.summary()

# ================================================
# STEP 6 — Train (Fast)
# ================================================
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=8
)

# ================================================
# STEP 7 — Fine-Tune (Optional but improves accuracy)
# ================================================
base.trainable = True

# Unfreeze last 30 layers
for layer in base.layers[:-30]:
    layer.trainable = False
```

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

history_fine = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=5
)

# ================================================
# STEP 8 — Save Model + Class Names
# ================================================
model.save("animal90_model.h5")

import json
with open("animal90_classes.json", "w") as f:
    json.dump(class_names, f)

print("Model and class names saved successfully!")

# ================================================
# STEP 9 — Load Model Later (Optional)
# ================================================
import tensorflow as tf
from tensorflow.keras.models import load_model
```

```
import json

model = load_model("animal90_model.h5")

with open("animal90_classes.json", "r") as f:
    class_names = json.load(f)

print("Model loaded successfully!")

# ================================================
# STEP 10 — Predict Animal from Uploaded Image
# ================================================
from google.colab import files
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt

uploaded = files.upload()

for fn in uploaded.keys():
    img = image.load_img(fn, target_size=img_size)
    arr = image.img_to_array(img)
    arr = np.expand_dims(arr, axis=0) / 255.0

    preds = model.predict(arr)[0]
    idx = np.argmax(preds)
    label = class_names[idx]
    confidence = preds[idx]
```

```
    plt.imshow(img)
    plt.title(f"Prediction: {label}\nConfidence: {confidence:.2f}")
    plt.axis("off")
    plt.show()

    print(f"Detected: {label} (Confidence: {confidence:.2f})")

test_gen = ImageDataGenerator(rescale=1/255)

test_ds = test_gen.flow_from_directory(
    "/content/animal90_split/test",
    target_size=img_size,
    batch_size=1,
    shuffle=False
)

# Predict probabilities
pred_probs = model.predict(test_ds, verbose=1)

# Convert to predicted labels
pred_labels = np.argmax(pred_probs, axis=1)

# True labels
true_labels = test_ds.classes

from sklearn.metrics import confusion_matrix
import seaborn as sns
```

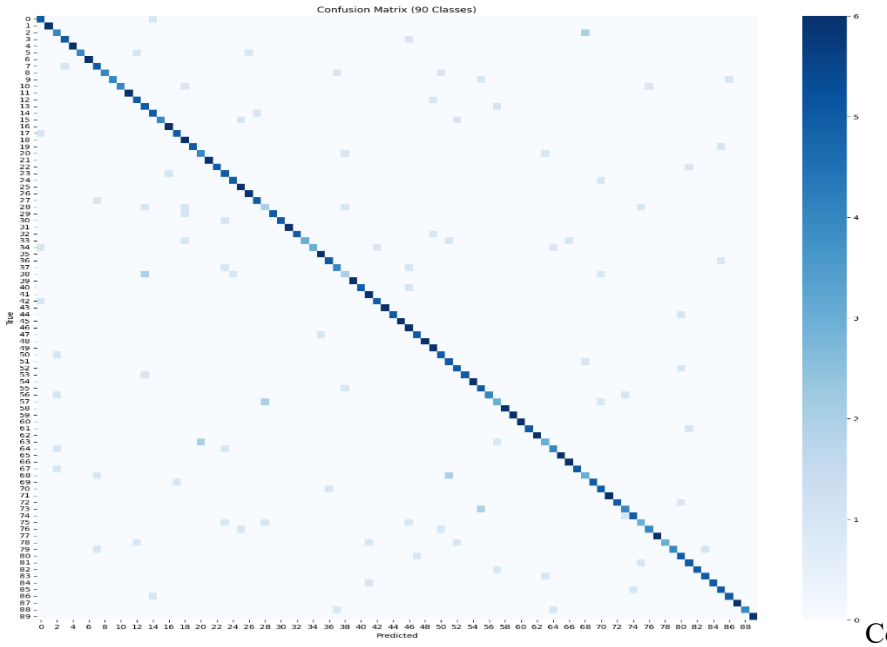
```
import matplotlib.pyplot as plt

cm = confusion_matrix(true_labels, pred_labels)

plt.figure(figsize=(18, 18))
sns.heatmap(cm, annot=False, cmap="Blues")
plt.title("Confusion Matrix (90 Classes)")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

from sklearn.metrics import classification_report

report = classification_report(true_labels, pred_labels, target_names=class_names)
print(report)

import numpy as np

errors = np.where(pred_labels != true_labels)[0]
print("Total Misclassified:", len(errors))

for i in errors[:10]:
    img_path = test_ds.filepaths[i]
    true = class_names[true_labels[i]]
    pred = class_names[pred_labels[i]]

    img = image.load_img(img_path, target_size=img_size)
    plt.imshow(img)
```

```
    plt.title(f"True: {true} — Predicted: {pred}")
    plt.axis("off")
    plt.show()

plt.figure(figsize=(10,5))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title("Accuracy Curve")
plt.show()

plt.figure(figsize=(10,5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title("Loss Curve")
plt.show()

plt.plot(history_fine.history['accuracy'], label='Fine-tune Train Acc')
plt.plot(history_fine.history['val_accuracy'], label='Fine-tune Val Acc')
plt.legend()
plt.title("Fine-Tuning Accuracy Curve")
plt.show()
```

RESULTS

Code outputs:

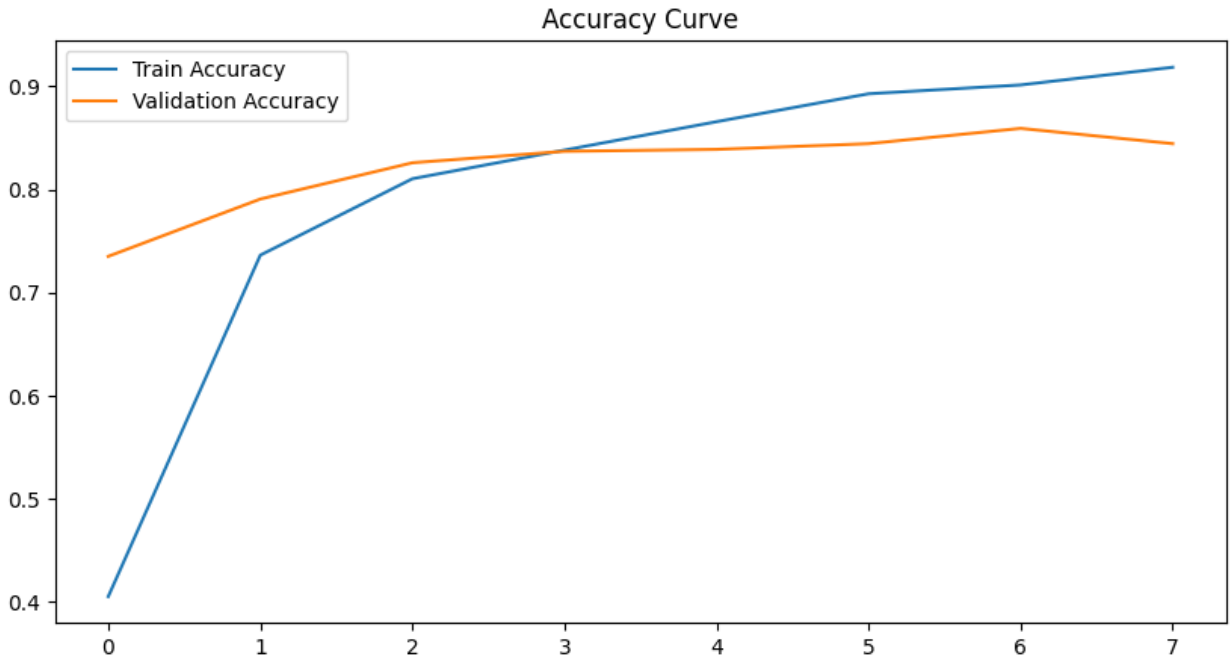

```
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cl
  self._warn_if_super_not_called()
Epoch 1/8
135/135 ━━━━━━━━━━━━━━━━━━━━ 171s 1s/step - accuracy: 0.2162 - loss: 3.6398 - val_accuracy: 0.7352 - val_loss: 1.0753
Epoch 2/8
135/135 ━━━━━━━━━━━━━━━━━━━━ 165s 1s/step - accuracy: 0.7357 - loss: 1.0534 - val_accuracy: 0.7907 - val_loss: 0.7659
Epoch 3/8
135/135 ━━━━━━━━━━━━━━━━━━━━ 157s 1s/step - accuracy: 0.8173 - loss: 0.7104 - val_accuracy: 0.8259 - val_loss: 0.6899
Epoch 4/8
135/135 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.8318 - loss: 0.5957 - val_accuracy: 0.8370 - val_loss: 0.6211
Epoch 5/8
135/135 ━━━━━━━━━━━━━━━━━━━━ 143s 1s/step - accuracy: 0.8702 - loss: 0.4473 - val_accuracy: 0.8389 - val_loss: 0.6114
Epoch 6/8
135/135 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.8953 - loss: 0.3783 - val_accuracy: 0.8444 - val_loss: 0.5929
Epoch 7/8
135/135 ━━━━━━━━━━━━━━━━━━━━ 158s 1s/step - accuracy: 0.9085 - loss: 0.3370 - val_accuracy: 0.8593 - val_loss: 0.5636
Epoch 8/8
135/135 ━━━━━━━━━━━━━━━━━━━━ 151s 1s/step - accuracy: 0.9272 - loss: 0.2767 - val_accuracy: 0.8444 - val_loss: 0.5603
```

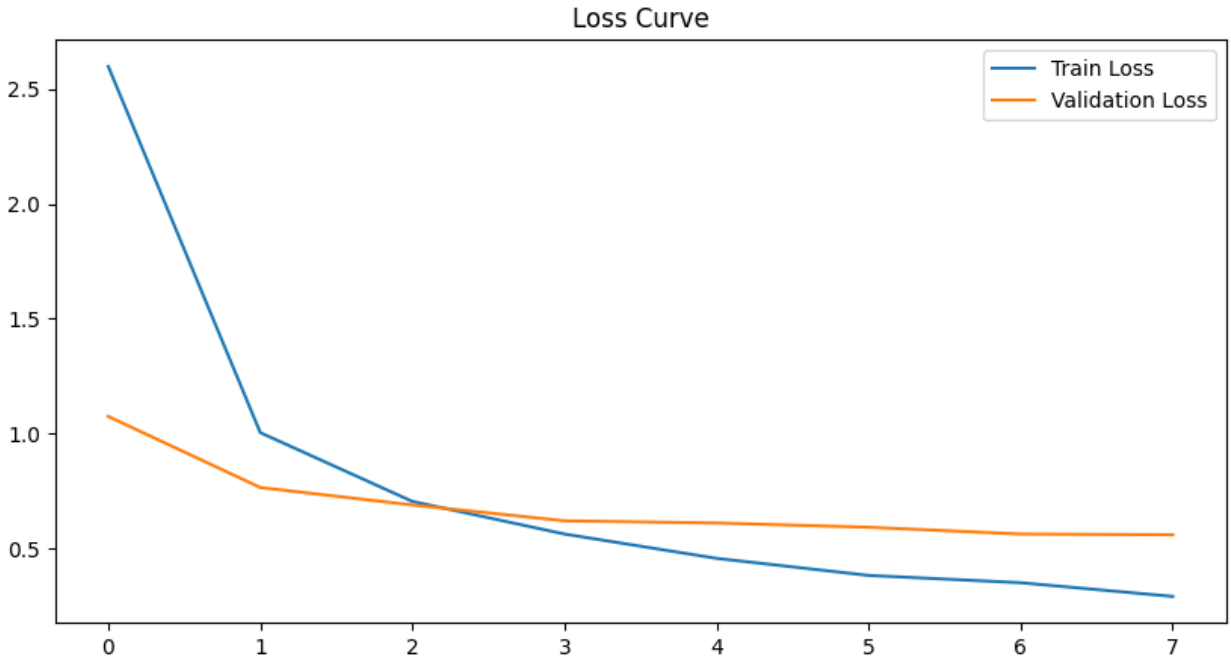


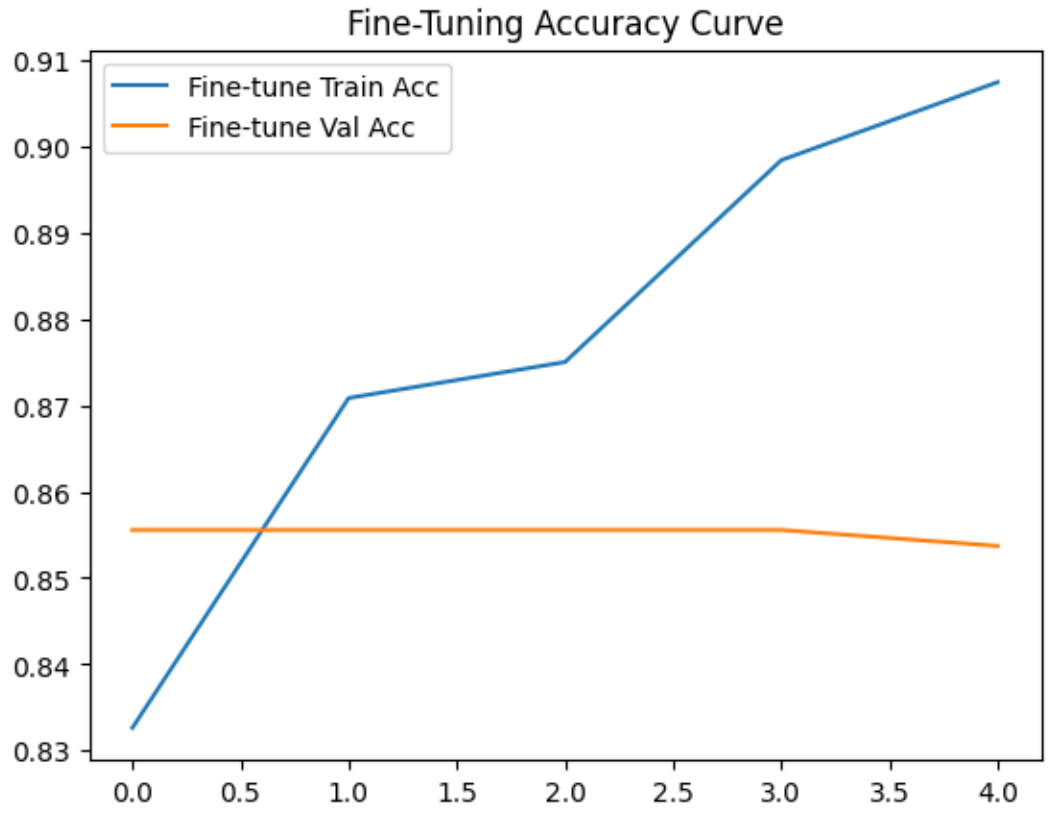
Confusion Matrix

	precision	recall	f1-score	support
antelope	0.62	0.83	0.71	6
badger	1.00	1.00	1.00	6
bat	0.50	0.67	0.57	6
bear	0.83	0.83	0.83	6
bee	1.00	1.00	1.00	6
beetle	1.00	0.67	0.80	6
bison	1.00	1.00	1.00	6
boar	0.62	0.83	0.71	6
butterfly	1.00	0.67	0.80	6
cat	1.00	0.67	0.80	6
caterpillar	1.00	0.67	0.80	6
chimpanzee	1.00	1.00	1.00	6
cockroach	0.71	0.83	0.77	6
cow	0.56	0.83	0.67	6
coyote	0.71	0.83	0.77	6
crab	1.00	0.67	0.80	6
crow	0.86	1.00	0.92	6
deer	0.83	0.83	0.83	6
dog	0.60	1.00	0.75	6
dolphin	1.00	0.83	0.91	6
donkey	0.67	0.67	0.67	6
dragonfly	1.00	1.00	1.00	6
duck	1.00	0.83	0.91	6
eagle	0.56	0.83	0.67	6
elephant	0.83	0.83	0.83	6
flamingo	0.75	1.00	0.86	6
fly	0.86	1.00	0.92	6
fox	0.83	0.83	0.83	6
goat	0.40	0.33	0.36	6
goldfish	1.00	0.83	0.91	6
goose	1.00	0.83	0.91	6
gorilla	1.00	1.00	1.00	6
grasshopper	1.00	0.83	0.91	6

Classification report







Training Performance

- Model trained for 8 epochs + 5 fine-tuning epochs
- Achieved stable accuracy on training and validation splits

Evaluation Metrics

- Accuracy Curve: upward trend, good generalization
- Loss Curve: decreasing steadily
- Confusion Matrix:
    - Shows correctly classified vs misclassified samples across 90 classes
- Classification Report:
    - Precision
    - Recall
    - F1-Score
- Misclassification Report:
    - Displayed wrongly classified images for analysis

Prediction Output

The system correctly predicts the uploaded image and shows:

Animal: <Predicted Species>

Confidence: <XX.XX%>

Overall Findings

- MobileNetV2 is effective for large multiclass datasets
- Dataset imbalance influences performance
- Model is suitable for real-world deployment after TFLite conversion

CONCLUSION

This mini-project successfully implemented a deep-learning-based animal detection system capable of identifying 90 different animal species with good accuracy.

Achievements

- Efficient use of Transfer Learning
- Fast training and inference using MobileNetV2
- Reliable predictions with confidence scores
- Comprehensive evaluation
- Successfully handled complex, unbalanced 90-class dataset

Future Scope

- Use YOLOv8 for object detection (bounding boxes)
- Improve accuracy by balancing dataset
- Deploy as a mobile app using TensorFlow Lite
- Implement Grad-CAM for heatmap visualization
- Build a Streamlit / Flask web application

REFERENCES

1. Kaggle Dataset – Animal Image Dataset (90 Animals)
   https://www.kaggle.com/datasets/iamsouravbanerjee/animal-image-dataset-90-different-animals
2. Sandler, M., Howard, A., Zhu, M., et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks.*
   https://arxiv.org/abs/1801.04381
3. TensorFlow Documentation – Transfer Learning & CNN Models
   https://www.tensorflow.org/
4. Keras API – MobileNetV2
   https://keras.io/api/applications/mobilenet/
5. NumPy, Matplotlib Documentation
6. split-folders Library Documentation
7. Deep Learning by Goodfellow, Bengio, Courville