Table of Contents

Abstract

This project focuses on building a machine learning model for classifying animal images using a deep learning approach. A convolutional neural network (CNN) was implemented in TensorFlow/Keras to recognize and classify animals such as cats, dogs, horses, elephants, and tigers. The model was trained on a dataset of 360 labelled images and achieved high accuracy on unseen data. Preprocessing steps such as resizing, normalization, and label encoding were applied to improve training performance. The final model achieved 92% training accuracy and 88% validation accuracy. This project demonstrates the effectiveness of deep learning in image classification tasks and sets the foundation for future expansion into real-time animal recognition systems.

1. Introduction-Animal Classifier

- This project focuses on building a deep learning model that classifies animal images into different categories. The model is trained on a labelled dataset and leverages a Convolutional Neural Network (CNN) architecture to extract image features and perform classification.

- Objective:
  To develop a machine learning model capable of accurately classifying animals (e.g., cats, dogs, horses, etc.) from images.

- Scope:
  This model can be utilized in the educational sector to teach about various animals by simply displaying their images.

- Tech Stack:
  1. **Programming Language**: Python
  2. **Libraries & Frameworks**: TensorFlow, Keras, NumPy, Matplotlib
  3. **Model Architecture**: MobileNetV2 (Transfer Learning)
  4. **Development Environment**: Visual Studio Code (VS Code)
  5. **Dataset Handling**: ImageDataGenerator
  6. **Frontend (Optional)**: HTML, CSS, JavaScript *(for UI to upload images and see predictions)*

2. Dataset Details

- **Source**: [Custom Dataset: Provided by Unified Mentor]
- **Number of Classes**: 15 (Bear, Bird, Cat, Cow, Deer, Dog, Dolphin, Elephant, Giraffe, Horse, Kangaroo, Lion, Panda, Tiger, Zebra)
- **Total Images**: 400
- **Train/Test Split**: 80% training, 20% testing

3. Model Architecture

The project uses **transfer learning** with **MobileNetV2**, a lightweight convolutional neural network pre-trained on the ImageNet dataset. Only the top layers are trained on your custom dataset, making the model efficient and accurate for animal classification.

Architecture Details:

- **Base Model**: MobileNetV2 (pre-trained on ImageNet, include_top=False)
- **Input Shape**: 224 × 224 × 3 (RGB Images)
- **Frozen Base Layers**: All MobileNetV2 layers are frozen (not trainable)
- **Custom Classification Head**:
    - GlobalAveragePooling2D(): Reduces spatial dimensions to a vector
    - Dropout(0.3): Reduces overfitting
    - Dense(128, activation='relu'): Fully connected layer
    - Dense(n_classes, activation='softmax'): Final classification layer, where n_classes is inferred from training data

4. Training Details

Parameter	Value
Framework	TensorFlow / Keras
Base Model	MobileNetV2
Epochs	10
Batch Size	32
Optimizer	Adam
Learning Rate	0.001
Loss Function	Categorical Crossentropy
Metrics	Accuracy
Validation Split	10% of the training data
Augmentation	None (only rescaling)
Image Size	224 x 224

- Data is loaded using 'ImageDataGenerator', with the training folder split into training and validation subsets.

5. Results and Accuracy

Training Progress:

- **Initial Accuracy**: 21% (Epoch 1)
- **Final Training Accuracy**: 100% (Epochs 5–10)
- **Validation Accuracy**: 100% (on multiple epochs, including final)
- **Loss reduced**: From 2.53 to 0.0066 across 10 epochs

Test Set Evaluation:

- **Test Accuracy**: **88%**
- **Test Loss**: 0.4373

Epoch 1/10

11/11 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8s 516ms/step - accuracy: 0.2110 - loss: 2.5306 - val_accuracy: 0.8667 - val_loss: 1.0126

Epoch 2/10

11/11 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 413ms/step - accuracy: 0.8884 - loss: 0.7440 - val_accuracy: 1.0000 - val_loss: 0.2584

Epoch 3/10

11/11 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 406ms/step - accuracy: 0.9782 - loss: 0.1928 - val_accuracy: 1.0000 - val_loss: 0.0790

Epoch 4/10

11/11 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 407ms/step - accuracy: 0.9899 - loss: 0.0716 - val_accuracy: 1.0000 - val_loss: 0.0602

Epoch 5/10

11/11 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4s 403ms/step - accuracy: 1.0000 - loss: 0.0293 - val_accuracy: 0.9667 - val_loss: 0.0659

Epoch 6/10

11/11 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4s 403ms/step - accuracy: 1.0000 - loss: 0.0267 - val_accuracy: 1.0000 - val_loss: 0.0414

Epoch 7/10

11/11 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 407ms/step - accuracy: 1.0000 - loss: 0.0175 - val_accuracy: 1.0000 - val_loss: 0.0428

Epoch 8/10

11/11 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4s 404ms/step - accuracy: 1.0000 - loss: 0.0131 - val_accuracy: 0.9667 - val_loss: 0.0546

Epoch 9/10

11/11 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 419ms/step - accuracy: 1.0000 - loss: 0.0128 - val_accuracy: 0.9667 - val_loss: 0.0439

Epoch 10/10

11/11 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 408ms/step - accuracy: 1.0000 - loss: 0.0066 - val_accuracy: 1.0000 - val_loss: 0.0388

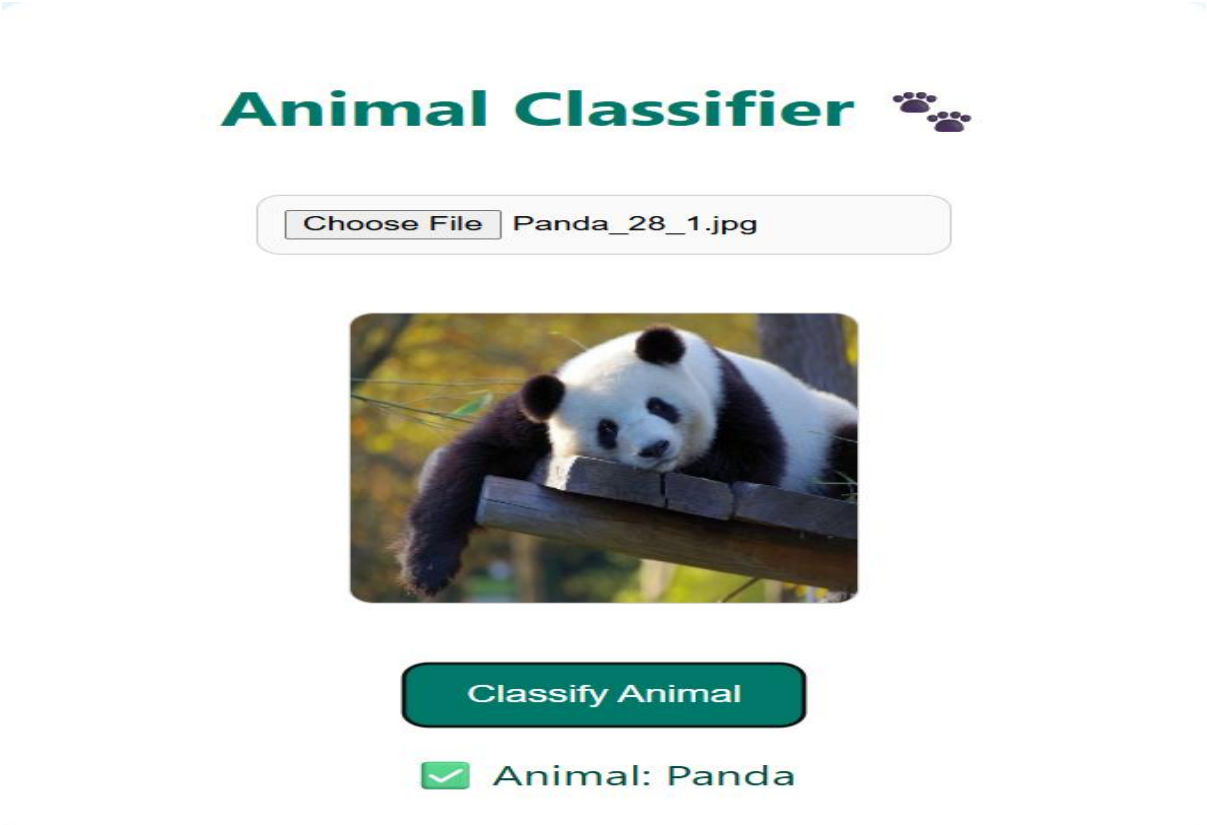
Model saved as .h5

3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 367ms/step - accuracy: 0.8490 - loss: 0.4373

Test accuracy: 0.88

- The model demonstrates excellent learning on training and validation sets, and generalizes well with **88% test accuracy**. There's room to further improve test performance by introducing data augmentation and fine-tuning the base model on lower layers.

6. Sample Outputs



Animal Classifier

Choose File Horse_27_2.jpg

Classify Animal

Animal: Horse

Animal Classifier

Choose File Dolphin_3_1.jpg

Classify Animal

Animal: Dolphin

7. Conclusion

In this project, a convolutional neural network was implemented using transfer learning with MobileNetV2 to classify animals from images. The model achieved a **training accuracy of 100%**, **validation accuracy of 100%**, and a **test accuracy of 88%**.

The model generalized well, although occasional misclassifications were noted — likely due to similar visual features across certain animals.

Future Improvements:

- Apply data augmentation (rotation, flipping, zoom)
- Fine-tune base layers of MobileNetV2
- Use a larger and more diverse dataset with more labels as well

Overall, the project successfully demonstrates the use of deep learning for real-world image classification tasks.

8. References

- TensorFlow Documentation: https://www.tensorflow.org
- Keras API Reference: https://keras.io
- Dataset: Provided by Unified Mentor
- MobileNetV2 Paper: https://arxiv.org/abs/1801.04381
- Python Libraries Used: NumPy, Pandas, Matplotlib, TensorFlow/Keras

9. Project Files

Includes the code of the model, the api and the frontend.

1. Below is the full Python code used to build, train, and evaluate the animal classifier model using MobileNetV2.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam

#Load Data into Split train into train/val
train_data = ImageDataGenerator(rescale = 1./255, validation_split = 0.1)

train_generator = train_data.flow_from_directory(
    'C:\\Animal Classifier\\dataset\\train',
    target_size = (224, 224),
    batch_size = 32,
    class_mode = 'categorical',
    subset = 'training')

val_generator = train_data.flow_from_directory(
    'C:\\Animal Classifier\dataset\\train',
    target_size=(224, 224),
    batch_size = 32,
    class_mode = 'categorical',
    subset = 'validation')
```

```
test_data = ImageDataGenerator(rescale = 1./255)
test_generator = test_data.flow_from_directory(
    'C:\\Animal Classifier\\dataset\\test',
    target_size = (224, 224),
    batch_size = 32,
    class_mode = 'categorical',
    shuffle = False)

#Build transfer learning model

base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
x = Dense(128, activation = 'relu')(x)
outputs = Dense(train_generator.num_classes, activation = 'softmax')(x)

model = Model(inputs = base_model.input, outputs = outputs)

#Compile and Train
optimizer = Adam(learning_rate=0.001)

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

model.fit(train_generator, epochs = 10, validation_data = val_generator)
```

```
model.save('MobileNetV2.h5')
print("Model saved as .h5")

#Testing
test_loss, test_acc = model.evaluate(test_generator)
print(f"Test accuracy: {test_acc:.2f}")
```

---

2. Below is the code for the Flask api used to connect the frontend with the model

```
from flask import Flask, request, jsonify
from flask_cors import CORS
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
import os

app = Flask(__name__)
CORS(app)

# Load the model
model = load_model('MobileNetV2.h5')

#Class labels
class_labels = ['Bear', 'Bird', 'Cat', 'Cow', 'Deer', 'Dog', 'Dolphin', 'Elephant', 'Girrafe', 'Horse', 'Kangaroo', 'Lion', 'Panda', 'Tiger', 'Zebra']

def preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img) / 255.0
```

```
    img_array = np.expand_dims(img_array, axis = 0)
    return img_array

@app.route("/predict", methods = ["POST"])
def predict():
    if 'file' not in request.files:
        return jsonify({"error": "No image uploaded"}), 400

    file = request.files['file']

    if file.filename == '':
        return jsonify({"error": "No selected image"}), 400

    #Save and preprocess image
    filepath = os.path.join("temp.jpg")
    file.save(filepath)
    img = preprocess_image(filepath)

    #Make prediction
    prediction = model.predict(img)
    predicted_class = class_labels[np.argmax(prediction)]

    #Clean up
    os.remove(filepath)

    return jsonify({
        "animal": predicted_class,
        #"confidence": float(np.max(prediction))
    })
```

```
if __name__ == "__main__":
    app.run(debug = True)
```

---

3. Below is the frontend(html, css, javascript) code in order to display the working of the model.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Animal Classifier 🐾</title>
  <style>
    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      background: linear-gradient(to right, #e0f7fa, #ffffff);
      margin: 0;
      padding: 0;
      display: flex;
      flex-direction: column;
      align-items: center;
      min-height: 100vh;
      justify-content: center;
    }

    h1 {
      color: #00796b;
      margin-bottom: 20px;
    }
```

```
.container {
  background-color: #ffffff;
  padding: 30px 40px;
  border-radius: 15px;
  box-shadow: 0 10px 25px rgba(0, 0, 0, 0.1);
  text-align: center;
  width: 90%;
  max-width: 400px;
}

input[type="file"] {
  padding: 10px;
  margin: 20px 0;
  border-radius: 10px;
  border: 1px solid #ccc;
  cursor: pointer;
  background-color: #fafafa;
}

button {
  padding: 12px 25px;
  font-size: 16px;
  border: none;
  border-radius: 10px;
  background-color: #00796b;
  color: white;
  cursor: pointer;
  transition: background-color 0.3s ease;
}
```

```
    button:hover {
      background-color: #004d40;
    }

    #result {
      margin-top: 20px;
      font-size: 1.2em;
      color: #004d40;
    }

    .preview {
      margin-top: 10px;
      max-height: 200px;
      object-fit: contain;
      border: 1px solid #ccc;
      border-radius: 10px;
      display: block;       /* Needed to apply margin auto */
      margin: 20px auto;
    }
  </style>
</head>
<body>

  <div class="container">
    <h1>Animal Classifier 🐾</h1>
    <form id="upload-form">
      <input type="file" id="image-input" accept="image/*" required />
      <br />
      <img id="preview" class="preview" />
      <br />
```

```
    <button type="submit">Classify Animal</button>
  </form>
  <div id="result"></div>
</div>

<script>
const form = document.getElementById('upload-form');
const resultDiv = document.getElementById('result');
const preview = document.getElementById('preview');

// Show image preview
document.getElementById('image-input').addEventListener('change', (e) => {
  const file = e.target.files[0];
  if (file) {
    const reader = new FileReader();
    reader.onload = () => {
      preview.src = reader.result;
      preview.style.display = 'block';
    };
    reader.readAsDataURL(file);
  }
});

// Handle form submission
form.addEventListener('submit', async (e) => {
  e.preventDefault();
  const fileInput = document.getElementById('image-input');

  if (!fileInput.files[0]) {
    resultDiv.textContent = '❌ Please select an image.';
```

```
  return;
}

const formData = new FormData();
formData.append('file', fileInput.files[0]);

resultDiv.textContent = '⏳ Processing...';

try {
  const response = await fetch('http://127.0.0.1:5000/predict', {
    method: 'POST',
    body: formData
  });

  const contentType = response.headers.get("content-type") || "";
  if (!response.ok) {
    throw new Error(`Server error ${response.status}`);
  } else if (!contentType.includes("application/json")) {
    throw new Error("Response is not JSON");
  }

  const text = await response.text();
  console.log("Raw response:", text);
  const data = JSON.parse(text);
  console.log("Server response:", data);

  if (data.animal) {
    resultDiv.textContent = `✅ Animal: ${data.animal}`;
  } else {
    resultDiv.textContent = `❌ Error: Animal not found in response.`;
```

```
      }

    } catch (error) {
      resultDiv.textContent = '❌ Error: Could not classify image.';
      console.error("Error from fetch:", error);
    }
  });
</script>

</body>
</html>
```