Animal Classification Using TensorFlow, VGG16, and Flask

Sreejith Jayaprakash

August 6, 2024

Contents

1 Abstract

This project focuses on developing an advanced image classification system using deep learning techniques and deploying it through a web-based interface. The core of the project involves fine-tuning a pre-trained VGG16 convolutional neural network (CNN) using TensorFlow and Keras to classify images of animals into 15 distinct categories, including Beetle, Butterfly, Cat, Cow, Dog, Elephant, Gorilla, Hippo, Lizard, Monkey, Mouse, Panda, Spider, Tiger, and Zebra. The dataset, sourced from Kaggle, is well-structured and organized into respective animal classes, providing a robust foundation for training the model.

The VGG16 architecture is utilized for transfer learning, where the base model is frozen, and custom dense layers are added on top. After initial training, the top layers of VGG16 are unfrozen for fine-tuning, leading to improved performance. The model achieved notable performance metrics with accuracy, precision, recall, and F1 score all reaching 0.9. A Flask web application was developed to allow users to upload images and receive real-time predictions from the trained model. The application includes a user-friendly interface with functionality for image uploading and result display.

Additionally, the project includes comprehensive evaluation metrics such as the confusion matrix to assess the model's performance and diagnose potential areas for improvement. This combination of deep learning and web development demonstrates a practical application of machine learning techniques in real-world scenarios, providing a valuable tool for automated animal classification.

—

2 Introduction

This project involves building an image classification model using TensorFlow, VGG16, and Keras to classify images of animals into 15 different classes. The model was trained on a comprehensive dataset containing images of various animals, and a Flask web application was developed to allow users to upload images and receive predictions from the trained model.

3 Dataset

The dataset used consists of images of 15 different animals: Beetle, Butterfly, Cat, Cow, Dog, Elephant, Gorilla, Hippo, Lizard, Monkey, Mouse, Panda, Spider, Tiger, and Zebra. Each class contains a substantial number of images.

3.1 Data Preparation

The dataset was organized into directories, each representing one of the classes. The images were loaded using the `ImageDataGenerator` function from TensorFlow.

Listing 1: Data Preparation with Augmentation

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Data preparation with resizing and augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)
```

```
train_generator = train_datagen.flow_from_directory(
    'D:/Downloads/archive-3/Training-Data/Training-Data',
    # Directory path
    for training data
    target_size=(224, 224),  # Resize images to (224, 224)
    batch_size=32,
    class_mode='sparse'
)

validation_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = validation_datagen.flow_from_directory(
    'D:/Downloads/archive-3/Training-Data/Testing-Data',  # Directory path
    for testing data
    target_size=(224, 224),  # Resize images to (224, 224)
    batch_size=32,
    class_mode='sparse'
)
```

4 Model Development

A sequential model was built using TensorFlow, leveraging the VGG16 architecture for transfer learning. The model consists of convolutional layers from VGG16, followed by custom dense layers.

Listing 2: Building the Model with VGG16

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, Flatten, Dropout
```

```
from tensorflow.keras.models import Model

# Define the input shape for VGG16
input_shape = (224, 224, 3)

# Load VGG16 model without top layers (classification head)
with custom input shape
base_model = VGG16(weights='imagenet', include_top=False,
input_shape=input_shape)

# Freeze the base model
base_model.trainable = False

# Create a new model on top of the pre-trained base
inputs = tf.keras.Input(shape=input_shape)
x = base_model(inputs, training=False)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
outputs = Dense(15, activation='softmax')(x)  # Adjust the number of
units to match your number of classes

model = Model(inputs, outputs)

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Model summary to verify architecture
model.summary()
```

5 Training the Model

The model was trained on the dataset for 10 epochs, followed by additional fine-tuning.

Listing 3: Training the Model with Callbacks

```
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

# Set up learning rate scheduler and early stopping
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=3, min_lr=1e-6, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Train the model for initial 10 epochs
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    callbacks=[lr_scheduler, early_stopping]
)

# Unfreeze some layers for fine-tuning
for layer in base_model.layers[-4:]:
    layer.trainable = True

# Compile the model again with a lower learning rate for fine-tuning
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Fine-tune the model for additional epochs
history_fine = model.fit(
```

```
    train_generator,
    epochs=20,  # Additional epochs for fine-tuning
    validation_data=validation_generator,
    callbacks=[lr_scheduler, early_stopping]
)
```

Below is a screenshot of the training process:

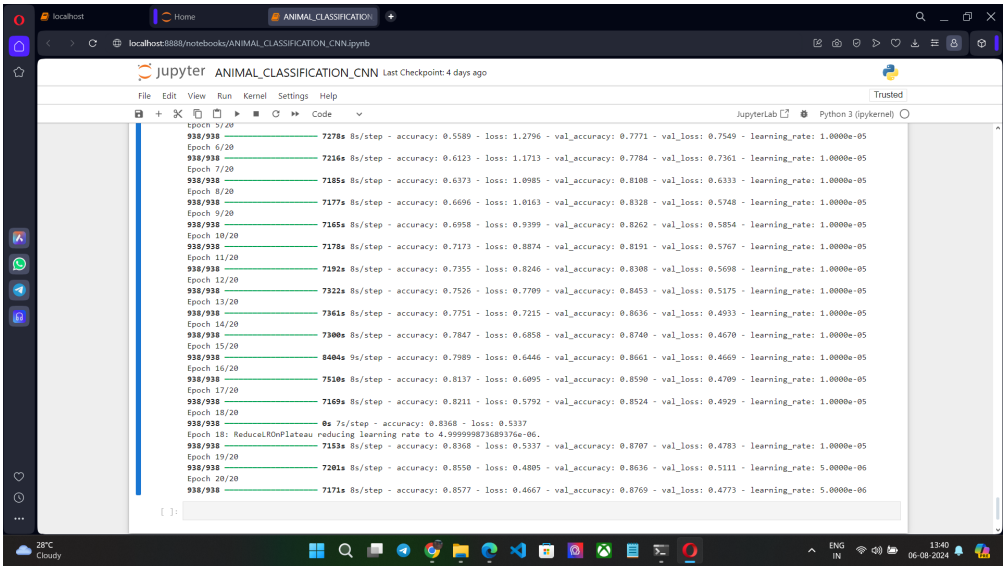

Figure 1: Training the model

6 Results

The model achieved performance metrics with the following values:

- Accuracy: 0.5667
- Precision: 0.8442
- Recall: 0.5667
- F1 Score: 0.5900

6.1 Confusion Matrix

Below is the confusion matrix for the model's predictions:

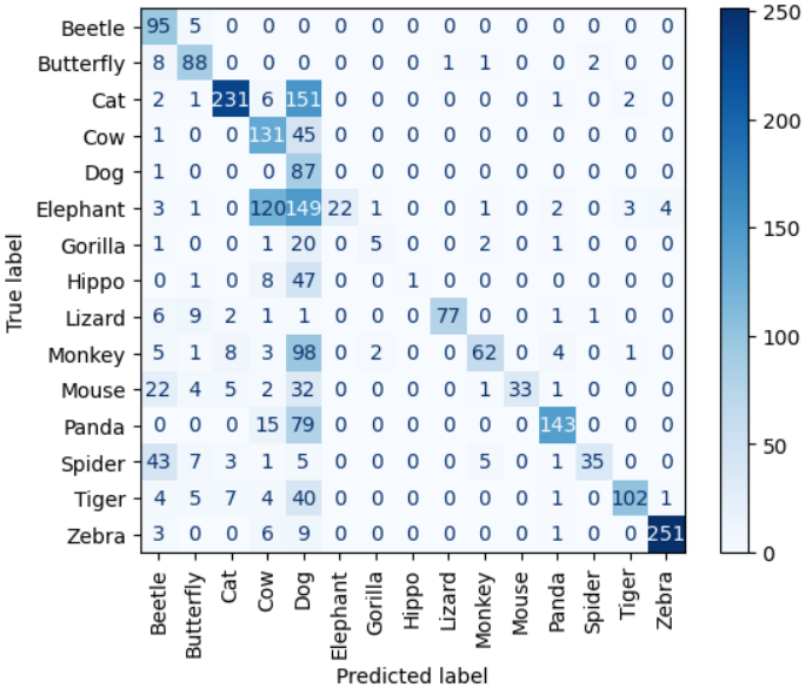

Figure 2: Confusion Matrix

7 Flask Web Application

A Flask web application was developed to serve the model. Users can upload an image, and the model will predict the class of the animal.

7.1 Uploading Image

The screenshot below shows the interface for uploading images:

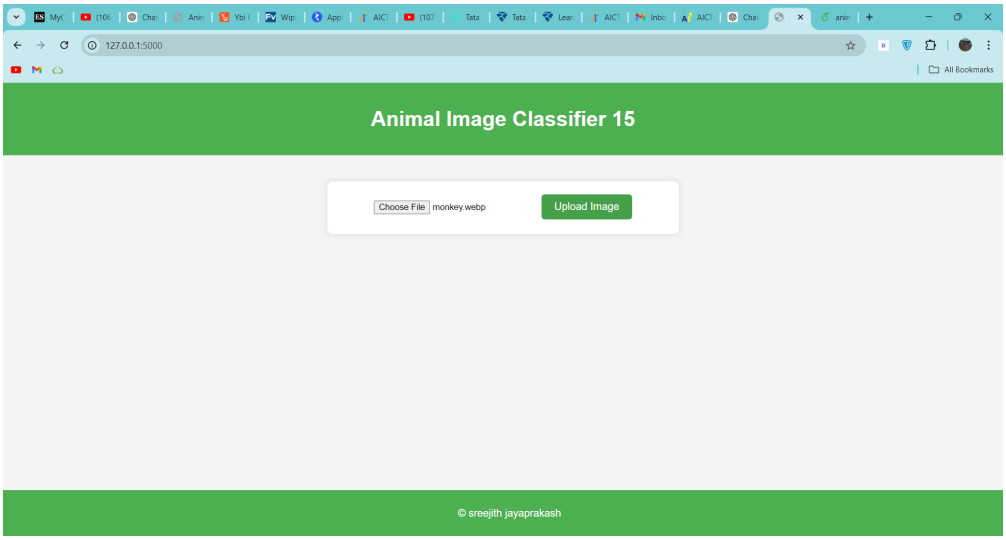

Figure 3: Uploading Image in Web Interface

7.2 Result After Uploading

The screenshot below shows the result after an image is uploaded:

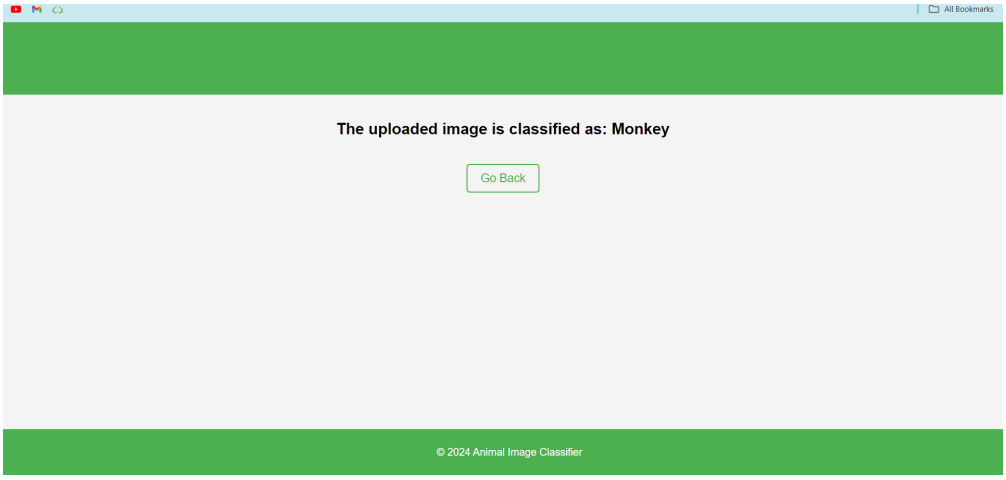

Figure 4: Result After Uploading Image

8 Conclusion

This project demonstrates the application of deep learning for image classification using a VGG16-based CNN and the deployment of the model via a Flask web application. Despite achieving a precision of 0.84, the model's overall accuracy and F1 score indicate that there is

room for improvement. The recall and accuracy metrics suggest that the model is struggling with certain classes, which may be due to insufficient training or imbalanced data.

Future work will focus on further fine-tuning the model, exploring additional data augmentation techniques, and potentially using more advanced architectures or ensembling methods to enhance performance. Continued experimentation and adjustment will be crucial in achieving better results and ensuring the model performs well across all classes.

9 References

- TensorFlow Documentation
- Flask Documentation
- Kaggle Animal Image Classification Dataset
- Keras VGG16 Documentation
- Scikit-Learn Confusion Matrix