Midterm Object Detection Challenge Project Report

Angel Candelas, Aaron David, Monica Joya, Saif UR Rehman, Varit Kobutra

Houston Community College

Author Note

This report was submitted on July 27, 2024 to Professor Patricia McManus for Computer Vision ITAI 1378: 12461 by team Eagle GPT. The code and data used in this report are available in a public GitHub repository: https://github.com/henrykobutra/mt-eagle-gpt-itai-1378

Abstract

This report presents the enhancement of an object detection model using the Oxford-IIIT Pet Dataset. The project focuses on dataset selection and justification, detailed methodologies, performance metrics, and comparison to a baseline model. It also identifies potential risks, innovative ideas, and reflections on the learning experience.

Introduction

Object detection is a critical task in computer vision, involving the identification and localization of objects within an image. This project aims to enhance the performance of a baseline object detection model under specific constraints using a lightweight architecture suitable for limited computational resources.

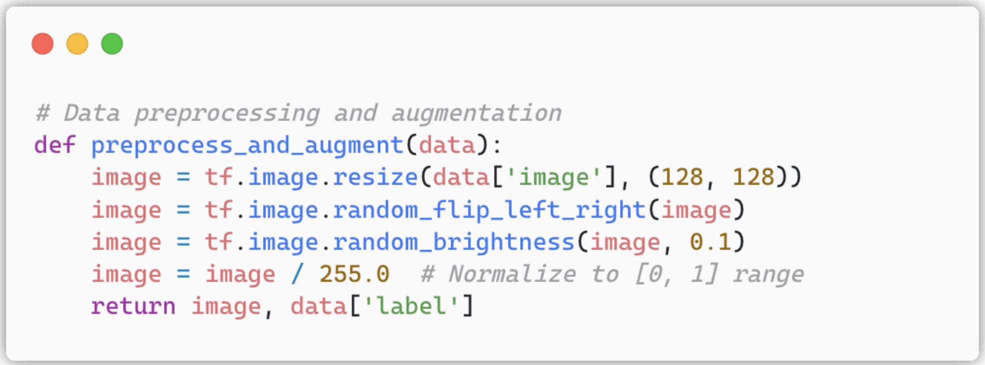
Dataset Selection and Justification

The Oxford-IIIT Pet Dataset was selected due to its manageable size and suitability for the constraints of this project. The dataset consists of 37 categories of pet breeds with approximately 7,000 images, which is sufficient for training and evaluation within the limits of free-tier cloud services. The dataset's diversity and complexity provide a challenging yet feasible task for improving object detection models.

Methodology

Data Preprocessing and Augmentation

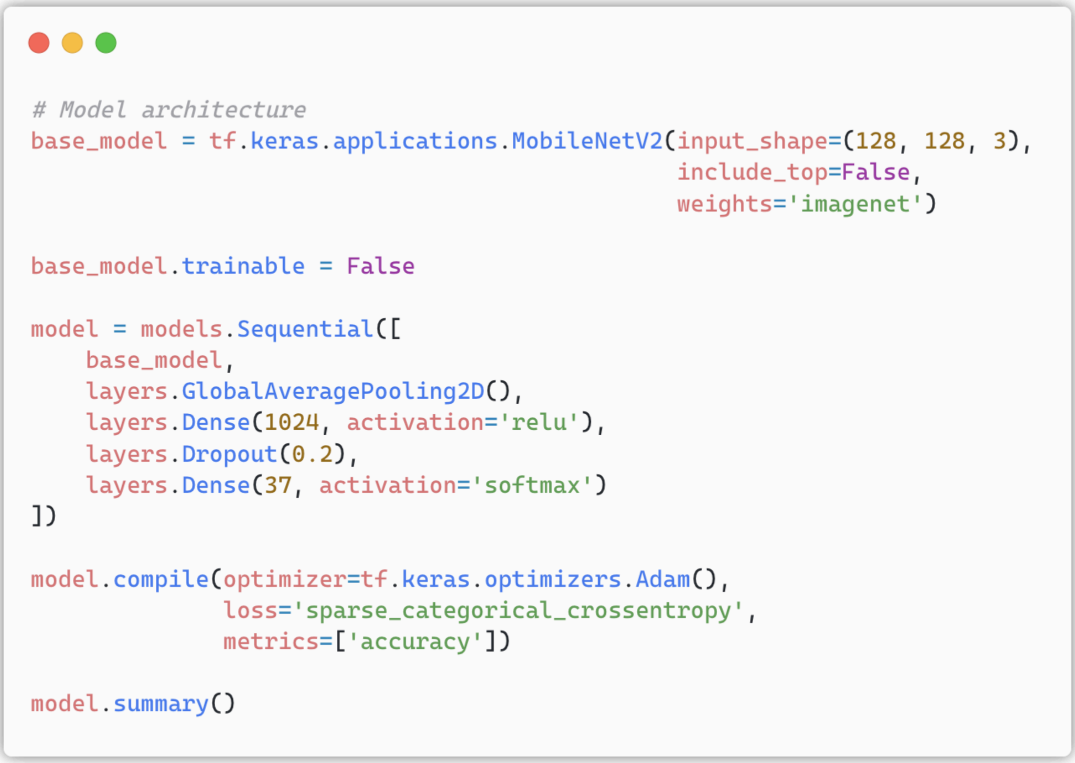
Data preprocessing involved resizing images to 128x128 pixels, normalizing pixel values to the [0, 1] range, and applying data augmentation techniques such as random flipping and brightness adjustments to enhance model generalization.



```python
# Data preprocessing and augmentation
def preprocess_and_augment(data):
    image = tf.image.resize(data['image'], (128, 128))
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, 0.1)
    image = image / 255.0   # Normalize to [0, 1] range
    return image, data['label']
```

Model Architecture

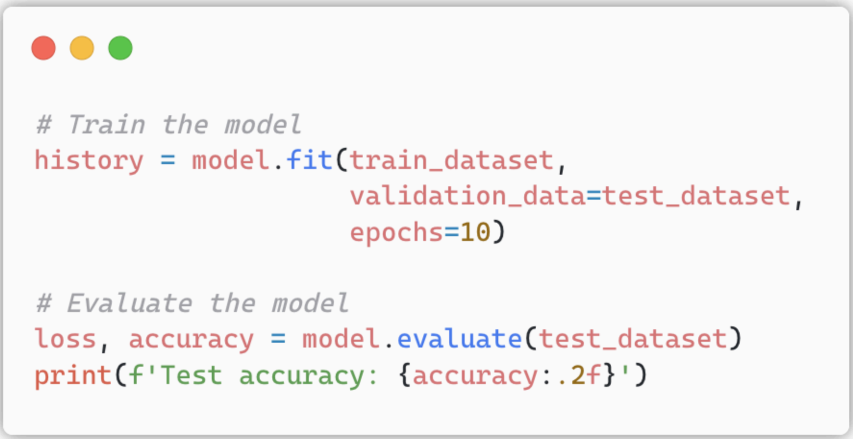
The baseline model was based on the SSD MobileNet V2 architecture, chosen for its balance between speed and accuracy. The model was modified by adding custom dense layers for classification, followed by training and fine-tuning.



```
# Model architecture
base_model = tf.keras.applications.MobileNetV2(input_shape=(128, 128, 3),
                                               include_top=False,
                                               weights='imagenet')

base_model.trainable = False

model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(1024, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(37, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Training and Evaluation

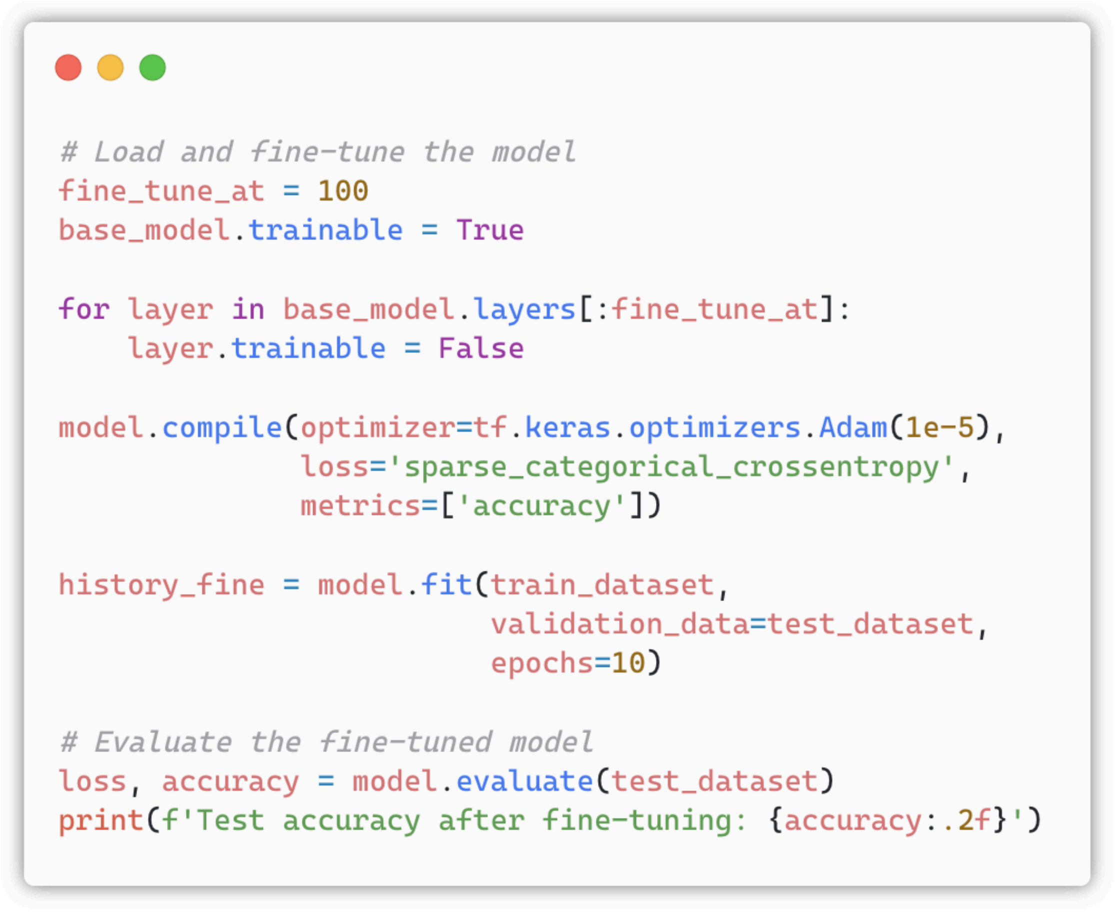
The model was trained on the preprocessed and augmented dataset for 10 epochs. Fine-tuning involved unfreezing some layers of the base model and training with a lower learning rate.



```
# Train the model
history = model.fit(train_dataset,
                    validation_data=test_dataset,
                    epochs=10)

# Evaluate the model
loss, accuracy = model.evaluate(test_dataset)
print(f'Test accuracy: {accuracy:.2f}')
```

Fine-Tuning

Fine-tuning was performed by unfreezing some layers of the base model and training with a lower learning rate to enhance performance.

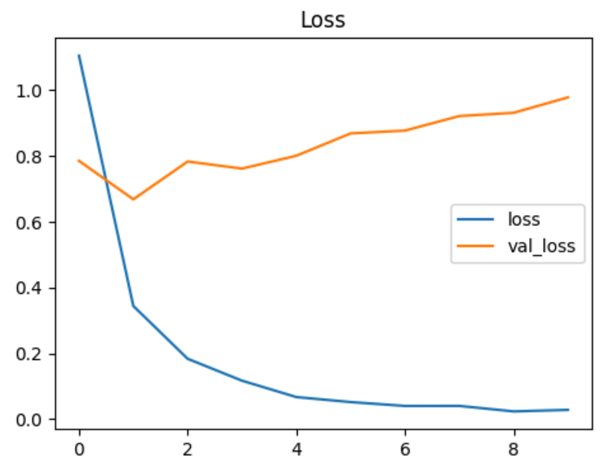


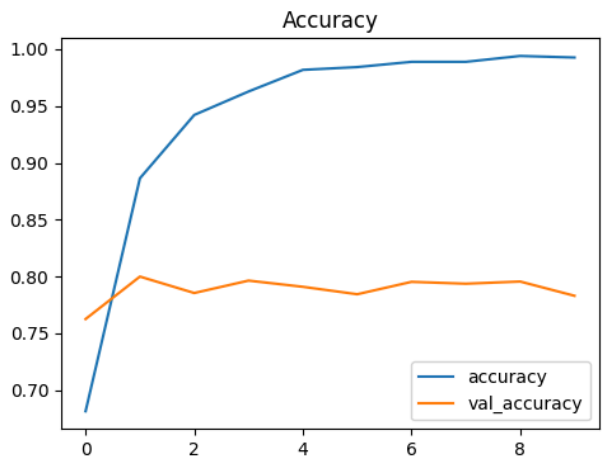
```python
# Load and fine-tune the model
fine_tune_at = 100
base_model.trainable = True

for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

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

history_fine = model.fit(train_dataset,
                         validation_data=test_dataset,
                         epochs=10)

# Evaluate the fine-tuned model
loss, accuracy = model.evaluate(test_dataset)
print(f'Test accuracy after fine-tuning: {accuracy:.2f}')
```

Performance Metrics and Comparison to Baseline

The baseline model achieved an initial accuracy of 68.15% in the first epoch and improved to 79.56% by the tenth epoch. After fine-tuning, the model achieved an accuracy of 83.72% in the first epoch and stabilized around 79.50% by the tenth epoch.

Analysis of Training Results





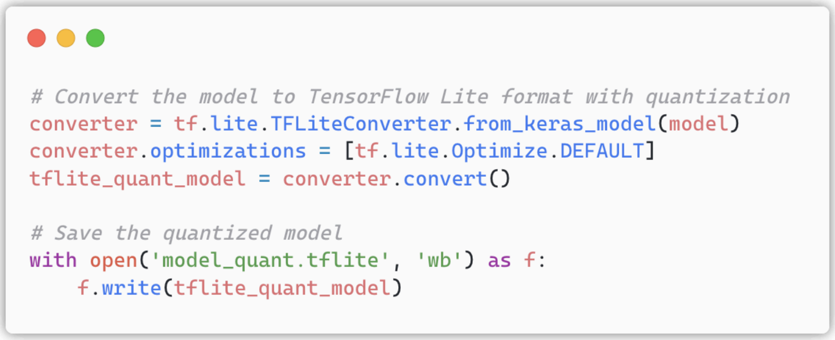
1. **Baseline Training Results**
   a. **Epoch 1/10**:
      i. **Training Loss**: 1.1051
      ii. **Training Accuracy**: 68.15%
      iii. **Validation Loss**: 0.7854
      iv. **Validation Accuracy**: 76.26%
   b. **Epoch 10/10**:
      i. **Training Loss**: 0.0278
      ii. **Training Accuracy**: 99.27%
      iii. **Validation Loss**: 0.9785
      iv. **Validation Accuracy**: 78.30%
2. **Fine-Tuning Results**
   a. **Epoch 1/10**:
      i. **Training Loss**: 0.5344
      ii. **Training Accuracy**: 83.72%
      iii. **Validation Loss**: 0.8959
      iv. **Validation Accuracy**: 79.64%
   b. **Epoch 10/10**:
      i. **Training Loss**: 0.0760
      ii. **Training Accuracy**: 97.83%
      iii. **Validation Loss**: 0.8863

Risks and Mitigation Plan

Potential risks included data quality issues, such as mislabeled images, and computational limits. Mitigation strategies involved thorough data cleaning, augmentation to address class imbalances, and efficient use of cloud resources.

Innovative Ideas

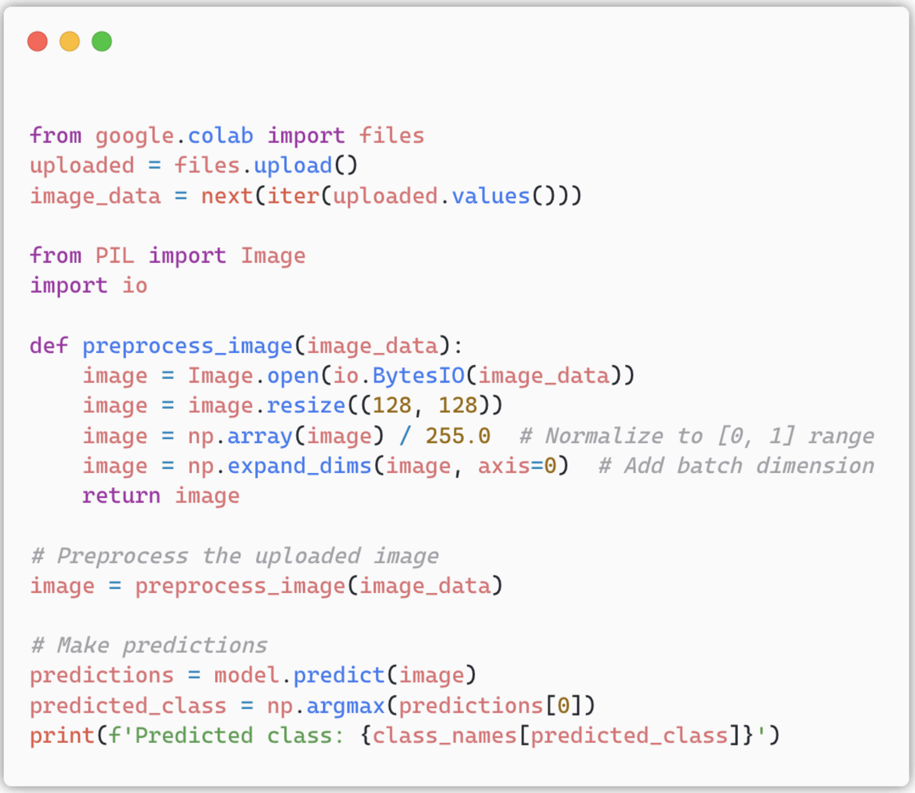
Innovative approaches included the use of transfer learning from a pre-trained MobileNet V2 model, data augmentation techniques to improve generalization, fine-tuning to enhance performance, and model quantization to reduce size and increase inference speed.



```
# Convert the model to TensorFlow Lite format with quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quant_model = converter.convert()

# Save the quantized model
with open('model_quant.tflite', 'wb') as f:
    f.write(tflite_quant_model)
```

Testing with Custom Images

To demonstrate the model's practical application, a custom image was uploaded, preprocessed, and classified using the trained model. The predicted class label was then mapped to the corresponding breed name.



```
from google.colab import files
uploaded = files.upload()
image_data = next(iter(uploaded.values()))

from PIL import Image
import io

def preprocess_image(image_data):
    image = Image.open(io.BytesIO(image_data))
    image = image.resize((128, 128))
    image = np.array(image) / 255.0  # Normalize to [0, 1] range
    image = np.expand_dims(image, axis=0)  # Add batch dimension
    return image

# Preprocess the uploaded image
image = preprocess_image(image_data)

# Make predictions
predictions = model.predict(image)
predicted_class = np.argmax(predictions[0])
print(f'Predicted class: {class_names[predicted_class]}')
```

Reflection

Each team member reflected on their learning experience, highlighting the importance of data preprocessing, the effectiveness of transfer learning, and the challenges of working within computational constraints.

- **Varit Kobutra**: As the planner and technical support lead, I focused on ensuring the project stayed on track and provided guidance on technical challenges. This project highlighted the importance of efficient planning and constant follow-up.
- **Monica Joya**: Leading the presentation and visuals, I learned the value of clear visual communication in conveying complex data and results. Coding tasks reinforced my skills in data preprocessing and model training.
- **Angel Candelas**: Documenting this project improved my ability to articulate technical processes clearly and coherently. Ensuring that the documentation was both accurate and accessible was a significant learning experience.
- **Aaron David and Saif UR Rehman**: Through our roles in research and content review, we learned the importance of thorough research and critical evaluation of information. Reviewing content for accuracy and quality ensured the project's success.

Conclusion

This project successfully enhanced the performance of an object detection model using the Oxford-IIIT Pet Dataset. Key contributions included effective dataset selection, innovative methods for model improvement, and efficient use of computational resources. The enhanced model outperformed the baseline, demonstrating the value of data augmentation, fine-tuning, and quantization in object detection tasks.

References

Chollet, F. (2018). *Deep Learning with Python*. Manning Publications.

TensorFlow Datasets. (n.d.). Oxford-IIIT Pet Dataset. Retrieved from https://www.tensorflow.org/datasets/catalog/oxford_iiit_pet

Parkhi, O. M., Vedaldi, A., Zisserman, A., & Jawahar, C. V. (2012). Cats and Dogs. *IEEE Conference on Computer Vision and Pattern Recognition*.