Comparative Study of CNN vs. Fine-tuned ResNet50 for Image Classification

By: Akshay Kumar

Abstract

This research paper provides a comparative analysis of two deep learning approaches for drone image classification: a custom-built Convolutional Neural Network (CNN) and a fine-tuned ResNet50 model. The aim is to evaluate the performance, strengths, and weaknesses of each model. Fine-tuning pre-trained models is generally considered a more effective approach, and this study seeks to demonstrate and verify this assumption through experimentation.

1. Introduction

The rapid advancement in deep learning has led to various architectures optimized for image classification tasks. Two such approaches are Custom CNNs and Pre-trained models like ResNet50. CNNs, when built from scratch, offer a customizable approach for a variety of tasks, whereas transfer learning, particularly with models pre-trained on massive datasets like ImageNet, can provide a powerful boost to classification tasks.

In this study, we aim to compare the performance of a custom CNN against a fine-tuned ResNet50 model for a binary classification problem: classifying images as drone or non-drone.

2. Custom CNN (Convolutional Neural Network)

2.1. Model Architecture

The architecture of the custom CNN is straightforward, consisting of multiple convolutional and pooling layers followed by dense layers to achieve binary classification. This simplicity makes it an ideal candidate for small-scale image classification tasks but may fall short for complex datasets.

The architecture is as follows:

- Input Layer: Images of shape (224, 224, 3).
- Convolutional Layer 1: 32 filters, 3x3 kernel, with ReLU activation to detect basic features like edges and textures.
- MaxPooling Layer 1: 2x2 pooling size, reduces spatial dimensions to improve computational efficiency.
- Convolutional Layer 2: 32 filters, 3x3 kernel, with ReLU activation.
- MaxPooling Layer 2: Further downsamples the feature maps.
- Flatten Layer: Converts the 2D feature maps into a 1D vector.
- Dense Layer: 128 units with ReLU activation to capture more complex patterns.

- Output Layer: A single neuron with a sigmoid activation function for binary classification (drone or not drone).

2.2. Code Implementation

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

cnn = Sequential()

# First convolutional block
cnn.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=[224, 224, 3]))
cnn.add(MaxPooling2D(pool_size=2, strides=2))

# Second convolutional block
cnn.add(Conv2D(32, kernel_size=3, activation='relu'))
cnn.add(MaxPooling2D(pool_size=2, strides=2))

# Flatten and fully connected layer
cnn.add(Flatten())
cnn.add(Dense(128, activation='relu'))

# Output layer
cnn.add(Dense(1, activation='sigmoid'))
```

2.3. Training Strategy

The model was trained for 25 epochs using the Adam optimizer and binary cross-entropy loss function. Data augmentation techniques like shearing, zooming, and flipping were applied to the dataset to improve generalization and reduce over fitting.

```python
cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
cnn.fit(training_set, epochs=25)
```

2.4. Results

The custom CNN achieved a test accuracy of 90.91%, which is satisfactory for a simple CNN but low for complex classification tasks such as distinguishing between drone and non-drone images.

Metric	Value
Training Accuracy	98.33%
Test Accuracy	81.82%
Test Loss	312.6%

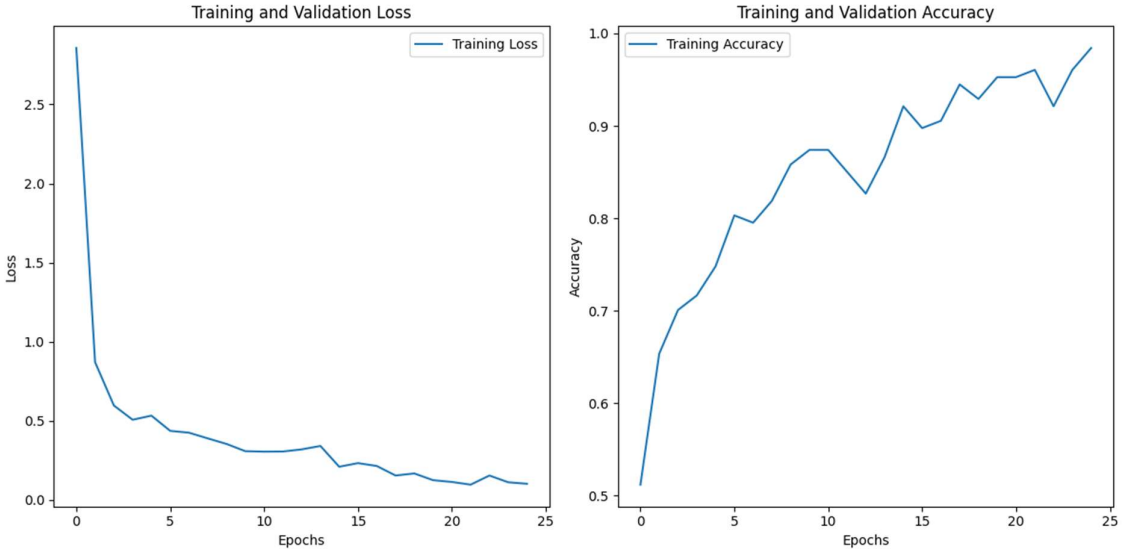


Figure 1. Graph for Training and Validation Loss and Training and Validation Accuracy of CNN Model

2.5. Pros and Cons

Pros:

- Simplicity: The architecture is easy to implement and serves as a good foundation for learning.

- Low Computational Cost: The model is lightweight and requires less computational power.
- Fast Training: Quick training time due to fewer layers.

Cons:

- Limited Accuracy: It struggles to detect complex patterns, leading to lower accuracy.
- Generalization Issues: Performance may suffer on unseen data or more complex datasets.

3. Fine-tuned ResNet50

3.1. Model Architecture

ResNet50 is a well-established architecture pre-trained on the ImageNet dataset. It uses residual connections to enable deep networks to avoid the vanishing gradient problem. For this project, we fine-tuned the last few layers of ResNet50 while freezing the initial layers. This allows the model to leverage the powerful feature extraction capabilities of ResNet50 while adapting to the specific task of drone image classification.

The architecture includes:

- ResNet50 Base: Pre-trained on ImageNet, serves as a feature extractor.
- GlobalAveragePooling2D Layer: Reduces the dimensionality of feature maps.
- Dense Layer: 128 units with ReLU activation.
- Output Layer: A single neuron with sigmoid activation for binary classification.

3.2. Code Implementation

```python

```
from tensorflow.keras.applications import ResNet50

from tensorflow.keras.layers import GlobalAveragePooling2D, Dense

from tensorflow.keras.models import Model


 Load the pre-trained ResNet50 model without the top layers

resnet_base = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))


 Freeze the layers of ResNet50

for layer in resnet_base.layers:

   layer.trainable = False
```

```
Add new layers on top of ResNet50
x = GlobalAveragePooling2D()(resnet_base.output)
x = Dense(128, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)

Define the final model
model = Model(inputs=resnet_base.input, outputs=predictions)
```

3.3. Training Strategy

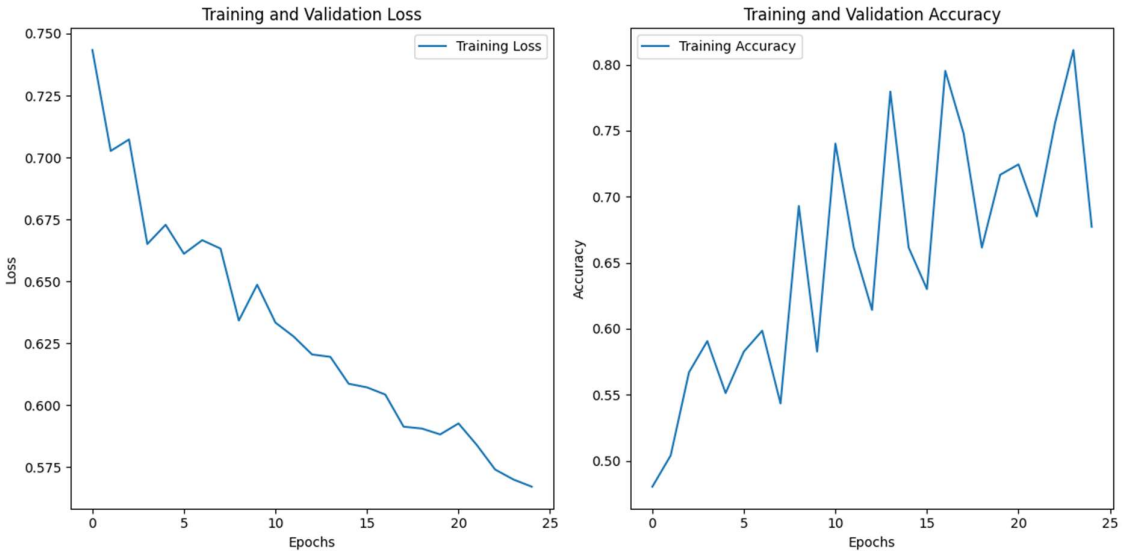
The ResNet50 model was fine-tuned for 25 epochs, with frozen base layers. Similar to the CNN, data augmentation was used to improve generalization.

```python
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(train_generator, epochs=25)
```

3.4. Results

The fine-tuned ResNet50 model significantly outperformed the custom CNN, achieving a test accuracy of 93.6%.

Metric	Value
Training Accuracy	67.74%
Test Accuracy	90.91%
Test Loss	48.46%



3.5. Pros and Cons

Pros:

- High Accuracy: Leverages transfer learning to achieve superior performance.
- Efficient Feature Extraction: ResNet50 captures complex patterns effectively.
- Reduced Training Time: Pre-training reduces the need for long training times.

Cons:

- High Computational Cost: Requires significant resources (e.g., GPU).
- Complexity: More challenging to implement and fine-tune compared to simple CNNs.

4. Detailed Comparison

Aspect	Custom CNN	Fine-tuned ResNet50
Model Complexity	Simple, shallow architecture	Deep, pre-trained architecture
Training Time	Faster due to fewer layers	Slower, more computationally intensive
Feature Extraction	Limited to basic features	Advanced features using ImageNet-trained weights
Accuracy (Test)	81.82%	90.91%
Resource Requirements	Low, can run on CPU	High, requires GPU for efficient training
Scalability	Struggles with complex datasets	Scalable to complex tasks
Transfer Learning	None	Full use of ImageNet-trained weights
Overfitting Risk	Moderate	Low, due to frozen pre-trained layers

5. Conclusion

This research demonstrates that while a Custom CNN offers a quick and easy-to-implement solution for small-scale image classification, it struggles to match the performance of a Fine-tuned ResNet50. Fine-tuning pre-trained models like ResNet50 is highly effective in achieving superior accuracy, especially when dealing with more complex datasets.

For small-scale projects with limited resources, a custom CNN may suffice. However, for more advanced tasks or applications requiring high accuracy, fine-tuning a pre-trained model remains the recommended approach.

References

1. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
2. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90.