Animal Classification with EfficientNet

Introduction

The goal of this project is to build an image classification model that identifies animal species from photos. Using a transfer learning approach, we leverage a pre-trained EfficientNet backbone and fine-tune it for our specific set of animal classes. The model is trained on the "Animals" dataset (Anto Benedetti) which contains images of five species: horse, lion, dog, elephant, and cat. The purpose is to achieve high classification accuracy while keeping the model compact and efficient. By using a state-of-the-art EfficientNet architecture, we aim to exploit its known advantages (high accuracy, fewer parameters, scalable design) in a multi-class animal recognition task.

Dataset Overview

The dataset is organized into class-labeled folders. There are **5 classes** (horse, lion, dog, elephant, cat) with roughly 2600–2800 training images per class (total ≈13.5K images). Each class folder contains JPG images of that animal. The data was split 80/20 into training and validation sets (stratified by class). Input images are preprocessed by resizing to (**224 x 224)** pixels (the input size expected by the EfficientNet model) and then passed through the EfficientNet-specific `preprocess_input` function (normalizing pixel values to the range and scale used by EfficientNet). No additional data augmentation (such as flips or rotations) was applied in this setup; the data pipeline shuffled the training set and batched it (batch size = 32) for the TensorFlow `tf.data` pipeline.

Model Architecture

The **base model** is EfficientNetB3 (pre-trained on ImageNet) with `include_top=False` (Base model layers are freezed to **prevent training into overfitting)**. The EfficientNet family is chosen because of its proven **state-of-the-art accuracy with a small model size**. EfficientNet uses a *compound scaling* method that simultaneously scales network width, depth, and input resolution in a balanced way. In practice, EfficientNet models (from B0 up to B7) provide a flexible range of model sizes: B0 is the baseline and B1–B7 are progressively larger variants obtained by applying the compound scaling coefficients. This means we can choose a model that fits our computational budget while still benefiting from the EfficientNet architecture. EfficientNet has been shown to achieve better accuracy than prior models (e.g. ResNet, NASNet, Inception) using **orders of magnitude fewer parameters and FLOPs**. For example, the largest EfficientNet-B7 reached ~84.4% top-1 accuracy on ImageNet while being *6.1× faster* and *8.4× smaller* than the previous best model. EfficientNet's efficiency and strong transfer-

learning performance make it well-suited for this task. In fact, scaled EfficientNet variants have achieved new state-of-the-art results on several transfer-learning benchmarks, using on average ~9.6× fewer parameters than previous models. These factors – high accuracy, few parameters, and good transferability – justify using EfficientNet as the backbone.

In our network, the **EfficientNetB3 base** is frozen (weights not trainable), and we add a small classification head. Specifically, the final convolutional output of EfficientNet is passed through a **GlobalAveragePooling2D** layer to produce a feature vector. We then apply a **Dropout (rate=0.4)** for regularization, and a final **Dense** layer with softmax activation to output probabilities for the 5 classes. Thus the model architecture is:

Input(224×224×3)
→ EfficientNetB3 (frozen, no top)
→ GlobalAveragePooling2D
→ Dropout(0.4)
→ Dense(5, softmax)

This keeps the number of trainable parameters small (only the head), while leveraging the rich feature extraction learned by EfficientNet. The chosen modifications (pooling and dropout) are standard for image classification heads and help reduce overfitting.

Training Setup

The model was compiled with the **Adam optimizer** (default learning rate), categorical cross-entropy loss, and tracked **accuracy**. Key hyperparameters and training details include:

- **Epochs:** Up to 70, with early stopping (monitoring validation loss with patience=5, restoring best weights).
- **Batch size:** 32.
- **Optimizer:** Adam (adaptive gradient, default LR≈0.001).
- **Loss function:** Categorical cross-entropy (multi-class classification).
- **Regularization:** Dropout 0.4 in the head, and the EfficientNet layers are frozen to prevent overfitting on the small dataset.
- **Data processing:** Images resized to 224×224 and normalized by `EfficientNet.preprocess_input`. No explicit data augmentation was used beyond this preprocessing and random shuffling.

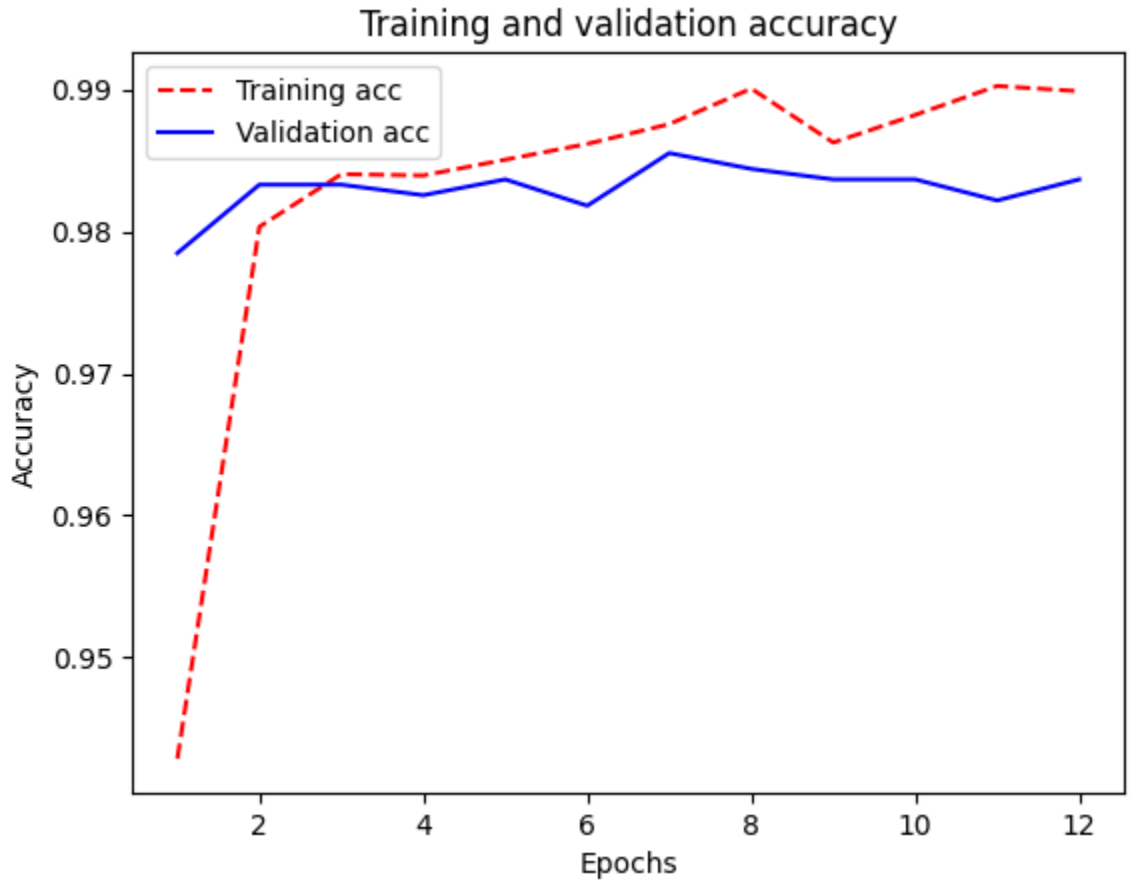
- **Early stopping:** The training was halted when validation loss stopped improving for 5 consecutive epochs, preventing overfitting.

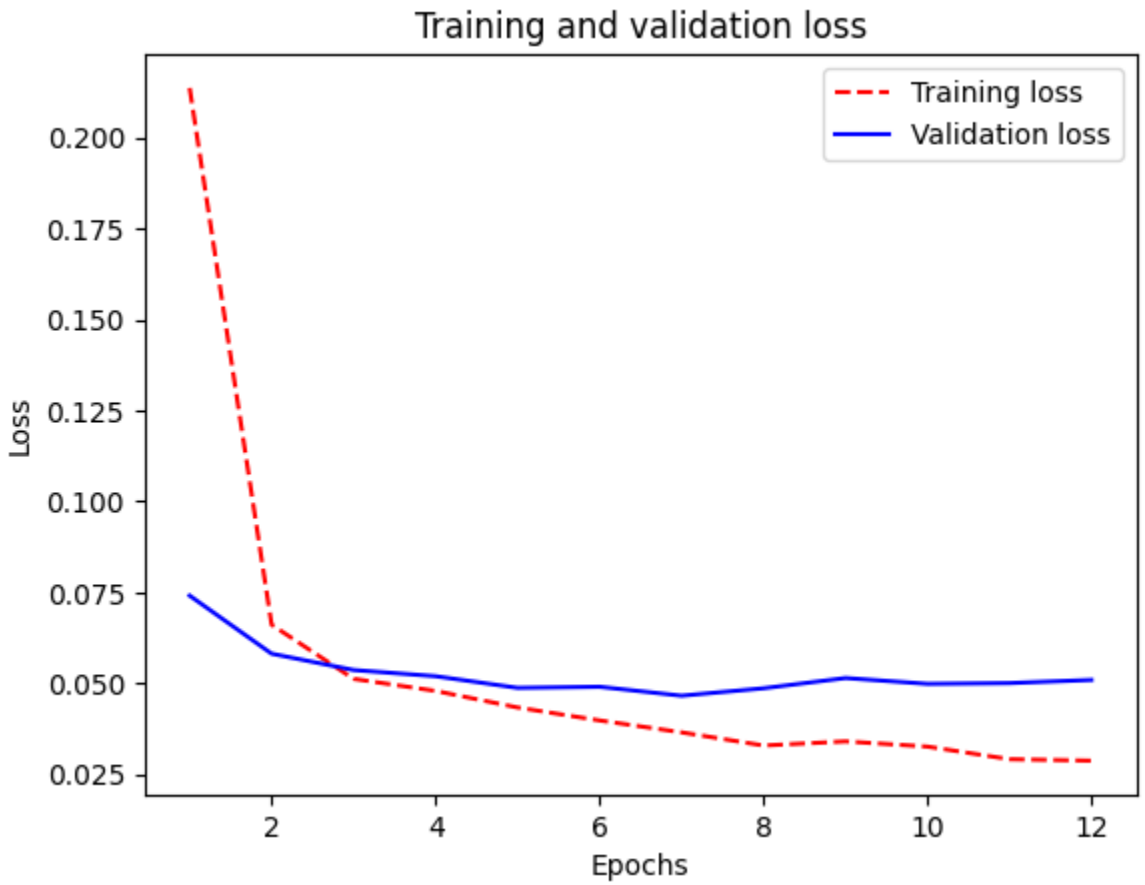
Training was conducted on GPU hardware (the code detected available GPUs) to accelerate computation. The TensorFlow `tf.data` pipeline was used for efficient data loading, batching, and prefetching.

Evaluation and Results

After training, we analyze the learning curves and final accuracy. The training and validation loss/accuracy per epoch are shown below. The model **rapidly improves** in early epochs. For instance, training accuracy jumps from ~85.9% to ~98.0% after the first two epochs, and validation accuracy rises into the high 97–98% range. Training accuracy eventually reaches ~98.9% with a very low loss (~0.03), while validation accuracy stabilizes around 98.4% and validation loss around 0.05 (see Figure below). Early stopping occurred around epoch 7–12 when the validation metrics plateaued.

Figure: Training and validation curves for loss (red) and accuracy (blue) over epochs.





In the final evaluation, the model achieved **~98.5% accuracy** on the validation set (the best validation accuracy observed). The confusion matrix was not explicitly computed, but sample predictions show that the model correctly identifies examples of each class (predicted labels for sample images: “dog”, “cat”, “lion”, “elephant”, etc. matched the true class). Overall, the high training and validation accuracies, along with low losses, indicate the model successfully learned to distinguish the five animal species with very few errors.

Conclusion

The EfficientNet-based classifier performed very well on the Animals dataset. Using EfficientNet’s compound-scaled architecture gave high accuracy with a relatively small model (few trainable parameters). The high validation accuracy (~98%) confirms that the model generalized well. The choice of EfficientNet is justified by its strong performance-per-parameter tradeoff and proven transfer-learning capabilities. In summary, the project demonstrates that a transfer-learning approach with EfficientNet achieves state-of-the-art-level results on a modestly-sized multi-class animal dataset. Future improvements could include fine-tuning some

EfficientNet layers, adding data augmentation, or ensembling multiple models to potentially boost performance further.

Sources: Model and dataset details are from the provided training notebook. EfficientNet design and performance characteristics are supported by the original research and summaries.