AML Challenge 1
Cactus Binary Classification

PAN Qizhi **KARIM Adnan**

11. May 2025

1. Introduction

In this challenge, we tackle a binary classification task involving 32×32-pixel aerial images to determine the presence of a specific cactus species. Our approach divides the problem into two stages: feature extraction using convolutional neural networks (CNNs), followed by a binary classifier trained on the extracted features.

For feature extraction, we first adopt ResNet50, a widely used CNN with residual connections, as our benchmark model. While ResNet50 excels in many image recognition tasks, we hypothesize its deep architecture (50 layers) may struggle with the small input size, potentially losing fine-grained spatial details critical for identifying subtle cactus patterns.

To address this limitation, we test EfficientNet, a modern architecture designed for parameter efficiency and scalability. Its compound scaling mechanism balances depth, width, and resolution, making it better suited for low-resolution images like our 32×32 inputs.

To fairly compare the feature extraction capabilities of ResNet50 and EfficientNet, we keep the downstream classifier identical(sigmoid) for both models, ensuring performance differences reflect only the quality of the extracted features.

2. Data Analysis and Preprocessing

For this task, we use all the images located in the ./train directory as our dataset. The dataset contains a total of 17,500 images, with 13,136 labeled as '1' (containing cactus) and 4,364 labeled as '0' (no cactus), indicating a class imbalance toward cactus-containing images.

2.1. Analysis

We begin by visually inspecting sample images from both classes:

From the visual comparison, we observe that images labeled as containing cactus often exhibit:

- More linear textures (corresponding to the shape and structure of cactus)
- Greater presence of green hues, likely due to vegetation

These visual cues suggest that the key discriminative features for our neural network should include the ability to detect fine textures and color distribution patterns associated with vegetation.

2.2. Preprocessing

The dataset was first split into 70% training, 15% validation, and 15% test subsets using stratified sampling to preserve class balance. Files were reorganized into subfolders by label (0 or 1) for compatibility with ImageFolder.

A ToTensor() transform was applied initially, and dataset-wide mean and standard deviation values were computed to support future normalization steps. The following statistics were calculated from the training set:

Mean: [R_mean, G_mean, B_mean] (actual values printed in logs)

Standard Deviation: [R_std, G_std, B_std]

These statistics can be used in normalization transforms in further experiments.

2.3. Class Imbalance Handling

To address class imbalance in the dataset, we employed a weighted cross-entropy loss function. Class weights were computed based on the inverse frequency of each class in the training set, assigning higher weight to the minority class to ensure fairer learning.

These weights were converted to a tensor and passed to the `CrossEntropyLoss` criterion in PyTorch. If CUDA was enabled, the tensor was moved to the GPU. This ensures that the model does not become biased toward the majority class by penalizing errors on the minority class more heavily.

The class weights used were:

Class 0 (majority): $w_0 = 0.2494$ Class 1 (minority): $w_1 = 0.7506$

These values were applied during training to improve class-level performance and overall model generalization.

3. Benchmark Model and performance

3.1. Resnet50

We used a pretrained ResNet-50 model from torchvision.models as a base. All convolutional layers were frozen to leverage pretrained features. The final fully connected layers were replaced with a custom classifier head.

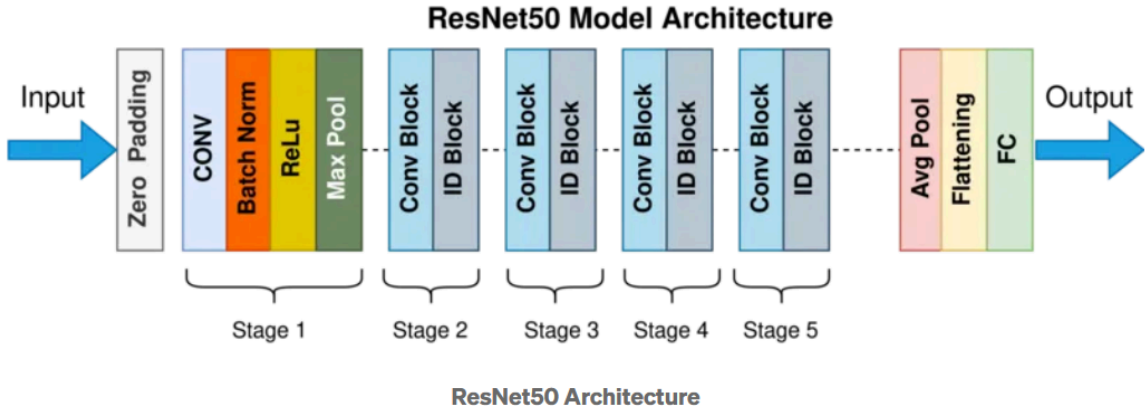

Figure 1: Architecture of the ResNet-50 residual neural network[1]

This head was designed to introduce non-linearity while progressively reducing feature dimensionality to a binary output space.

3.2. Performance

We trained the model with epoch = 500 and lr = 0.0005.

[1]Copyright from https://medium.com/@nitishkundu1993/exploring-resnet50-an-in-depth-look-at-the-model-architecture-and-code-implementation-d8d8fa67e46f

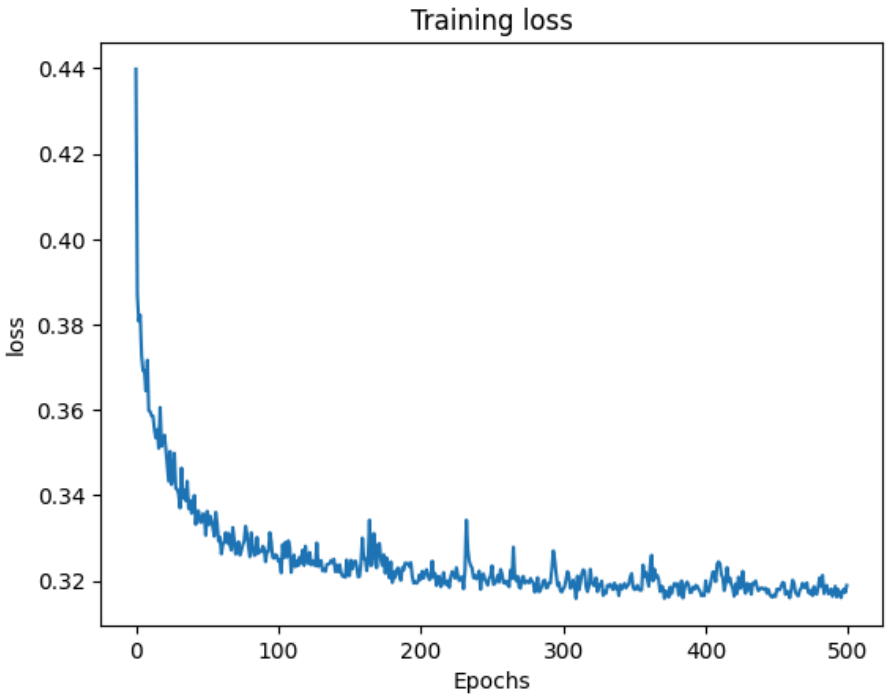

Figure 2: Training loss of Resnet50 with epoch = 500, lr = 0.0005

	Pred: 0	**Pred: 1**
True: 0	1924	41
True: 1	46	614

Table 1: Confusion Matrix

AUC score	95.7%
Accuracy	96.7%
F1 score	93.4%
Precision score	93.0%
Recall score	93.7%

Table 2: Metrics

Although ResNet50 performs well (96.7% accuracy, 93.4% F1 score), these results must be considered alongside class imbalance. F1, precision, and recall provide more meaningful insights than accuracy alone. The model performs reliably, but with some misclassifications

3.3. Observation

The model required significant training time and epochs to stabilize, as shown in its loss curve. This slow convergence and moderate misclassification rate suggest that ResNet50 is not ideally suited for low-resolution image tasks under resource constraints.

To address these limitations, we explored more computationally efficient architectures. Among them, **EfficientNet** stands out for its ability to achieve high accuracy with significantly fewer parameters and FLOPs, making it a promising candidate for replacing ResNet50 in this context.

4. Improved Method

4.1. EfficientNet

To cope with the problems that the ResNet50 might have, we employed EfficientNet-B0, a lightweight CNN with approximately 5.3 million parameters and 0.39 billion FLOPs, which achieves higher accuracy than ResNet-50 (25.6M parameters, 4.1B FLOPs) while requiring significantly less computation.

Compared to ResNet-50, EfficientNet-B0 offers a $5\times$ reduction in parameters and $10\times$ reduction in FLOPs, while improving accuracy.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 3: EfficientNet B0 structure in table

4.2. Performance

Here we still trained the model with epoch = 500 and lr = 0.0005.

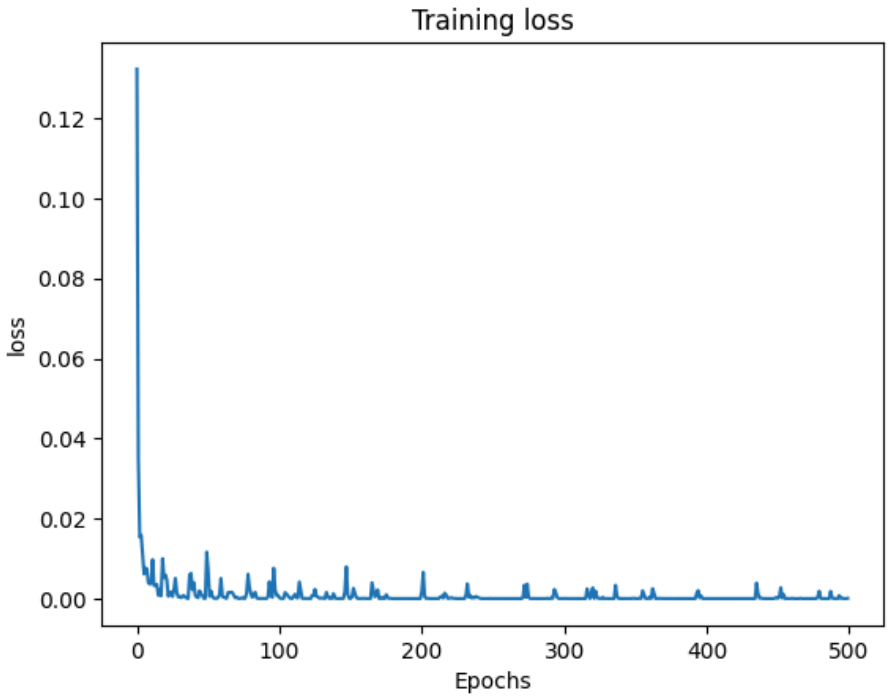

Figure 4: Training loss of EfficientNet with epoch = 500, lr = 0.0005

	Pred: 0	**Pred: 1**
True: 0	1961	0
True: 1	9	655

Table 3: Confusion Matrix

AUC score	99.8%
Accuracy	99.6%
F1 score	99.3%
Precision score	98.6%
Recall score	100%

Table 4: Metrics

4.3. Comparison

We compare the performance of EfficientNet and ResNet based on their classification accuracy, evaluation metrics, and convergence behavior. The results clearly demonstrate the superior performance of EfficientNet in both predictive capability and training dynamics.

EfficientNet achieved an impressive accuracy of 99.6%, with an AUC score of 99.8%, F1 score of 99.3%, and precision/recall scores of 98.6% and 100%, respectively. The corresponding confusion matrix indicates that only 9 false negatives and 0 false positives occurred, highlighting its excellent ability to distinguish between classes.

In contrast, ResNet obtained a lower accuracy of 96.7%, with an AUC score of 95.7%, F1 score of 93.4%, precision of 93.0%, and recall of 93.7%. The confusion matrix shows more misclassifications, with 41 false positives and 46 false negatives, reflecting its relatively weaker performance on this task.

Furthermore, the training loss curve of EfficientNet was remarkably smooth and stable, converging rapidly to below 0.1 and remaining near zero throughout training. This suggests better generalization and optimization behavior compared to ResNet.

5. Conclusion

The superior performance of EfficientNet over ResNet in this classification task can be largely attributed to its more refined architectural design for feature extraction. EfficientNet uses a compound scaling method that uniformly scales depth, width, and resolution, allowing the model to maintain a balanced capacity for capturing spatial and semantic features. Its depthwise separable convolutions and squeeze-and-excitation blocks further enhance its ability to extract discriminative features from images while keeping the model lightweight.

Compared to ResNet, which relies heavily on increasing depth through residual connections, EfficientNet achieves better feature representation with fewer parameters, leading to improved generalization. The early and consistent convergence of the loss curve highlights the model's efficiency in learning robust features quickly and stably.

In summary, EfficientNet's design allows for deeper semantic understanding and more effective use of spatial hierarchies in the input data, making it a superior choice for image classification tasks where both accuracy and computational efficiency are essential.