Apple Species Classification Using CNN and Transfer Learning

Dataset Discussion

This project utilized the Fruits360 dataset, published on Kaggle by Horea Mureșan. It features thousands of labeled fruit images captured under controlled lighting and backgrounds to ensure consistency. For this work, the dataset was filtered to include only apple species.

Such data is valuable for training image classifiers in educational, agricultural, and commercial applications—like fruit recognition tools for farms, grocery stores, or educational platforms.

Problem Statement

The core task was to classify different species of apples from images—a supervised classification problem. It is particularly difficult because many apple species look nearly identical, differing only in subtle features like hue or texture.

I approached the project with a working knowledge of convolutional neural networks (CNNs), training strategies, and transfer learning. The goal was to evaluate and compare model performances, including a custom CNN and pre-trained architectures.

Dataset Exploration

Before training, I manually explored the dataset. The images were standardized, but many apple species appeared highly similar. For instance, Red Delicious and Braeburn apples often look identical. I removed non-apple categories, excluded apple types with few examples, renamed labels for clarity, and organized the data into training, validation, and test splits.

Sample images were plotted using matplotlib to visually confirm label quality. Some apples had distinctive colors (e.g., Granny Smith's green, Pink Lady's soft red), while others were harder to distinguish even by eye.

Technical Approach

I implemented and evaluated four models:

- A custom CNN built with Keras
- MobileNetV2 (pre-trained on ImageNet)
- ResNet50
- EfficientNetB0

The CNN included convolutional and pooling layers, followed by dropout and dense layers. It was trained on 100x100 images. MobileNetV2 also used 100x100 input and began with pre-trained weights. ResNet50 and EfficientNetB0 required 224x224 images.

All models used the Adam optimizer, an initial learning rate of 0.001 (reduced via ReduceLROnPlateau), and categorical cross-entropy with label smoothing. To reduce overfitting, I added dropout, batch normalization, and used early stopping based on validation accuracy. Transfer learning models were fine-tuned by freezing and unfreezing layers strategically.

Data was split as follows:

- Training: 60%
- Validation: 20%
- Test: 20%

Performance Analysis
MobileNetV2 outperformed all others, achieving over 85% validation accuracy. In contrast, the custom CNN overfit quickly—performing well on training data but poorly on the validation set, especially for visually similar species. Learning curves showed a growing gap between training and validation performance after several epochs.

Despite their potential, ResNet50 and EfficientNetB0 underperformed. Their complexity may have been excessive for this dataset, and upscaling from 100x100 to 224x224 likely diluted subtle visual cues. I tried data augmentation and fine-tuned only the top layers, but results didn't improve meaningfully.

I also experimented with an ensemble approach by averaging predictions from MobileNet, ResNet, and EfficientNet. However, the poor performance of the latter two models prevented any boost in overall accuracy.

Reflection and Conclusion
This project successfully built and evaluated deep learning models for apple classification. MobileNetV2 proved to be the best solution, balancing generalization with performance. While deeper models like ResNet50 and EfficientNetB0 are powerful, they demand more data and resolution than this dataset provided.

Overfitting was a key issue for the CNN. However, MobileNetV2—with careful fine-tuning and preprocessing—largely avoided it. I iterated through several design changes, including image resolution, augmentation strategies, and ensemble experiments, before landing on the final approach.

While not perfect, the project made meaningful progress. In the future, expanding the dataset or using attention mechanisms might further improve accuracy. More broadly, this work strengthened my understanding of transfer learning, data preparation, and model evaluation—skills applicable across a wide range of machine learning tasks.

Time spent

3/22 - 2.0 hours - created repo and initialized README
3/23 - 0.5 hours - installed dataset
3/23 - 1.5 hours - reviewed dataset
3/23 - 1.0 hour - cleaned and renamed data
3/24 - 1.5 hours - researched CNNs and transfer learning
3/24 - 2.0 hours - set up project structure and organized data
3/25 - 1.0 hour - read about model architectures
3/26 - 0.5 hours - reviewed performance metrics
3/27 - 1.0 hour - watched image classification tutorials
3/28 - 1.0 hour - reviewed dataset examples and CNN guides
4/16 - 2.0 hours - tested models and analyzed performance
4/16 - 2.0 hours - built ensemble logic
4/16 - 2.5 hours - resized data and trained models
4/17 - 1.5 hours - debugged preprocessing issues
4/17 - 2.0 hours - integrated MobileNetV2 and tuned layers
4/17 - 2.5 hours - fine-tuned models and adjusted learning rate
4/18 - 2.0 hours - built initial CNN
4/18 - 1.0 hour - created prediction pipeline
4/18 - 1.5 hours - analyzed validation accuracy
4/18 - 2.0 hours - added data augmentation
4/20 - 1.0 hour - wrote results and conclusions

Total time spent: 32 hours

Time by category:

- Prep Work: 5.0 hours
- Research: 5.0 hours
- ML Modeling: 20.0 hours
- N/A: 2.0 hours