CIFAR-10 Image Classification with Transfer Learning (TensorFlow)

---

1. Data Preprocessing

Dataset: CIFAR-10

- 60,000 32×32 color images in 10 classes
- 50,000 for training, 10,000 for testing

Preprocessing Steps

- Resized images to **160x160** to match `MobileNetV2` input
- Normalized pixel values to range [0, 1]
- One-hot encoded labels
- Efficient data loading using `tf.data.Dataset` with:
  - Shuffling
  - Batching
  - Prefetching

Visualization

Sample images and labels were visualized using `matplotlib` to verify correct loading and distribution across classes.

---

2. Model Architecture

a) Custom CNN (from scratch)

We first implemented a custom CNN for CIFAR-10:

- Conv2D (32 filters, 3x3) + ReLU + MaxPooling
- Conv2D (64 filters, 3x3) + ReLU + MaxPooling
- Conv2D (128 filters, 3x3) + ReLU + MaxPooling
- Flatten + Dense(256, ReLU) + Dropout(0.5)
- Output: Dense(10, softmax)

This model served as a baseline before applying transfer learning.

b) Transfer Learning with MobileNetV2

- Base model: `MobileNetV2(weights='imagenet', include_top=False)`

- Added custom head:
  - Global Average Pooling
  - Dense(128, activation='relu')
  - Dropout(0.3)
  - Dense(10, activation='softmax')

  Total layers: 158 (154 from base + 4 custom)

---

3. Model Training

⚙ Phase 1: Initial Training

- Frozen base model
- Optimizer: Adam (lr=0.001)
- Loss: Categorical crossentropy
- Batch Size: 32
- Early stopping on validation loss

Phase 2: Fine-Tuning

- Unfroze last 50 layers of the base model
- Recompiled with lower learning rate: `1e-5`
- Trained for 10 epochs with early stopping

---

4. Model Evaluation

Metrics

- Final Accuracy: **90.24%**
- Final Loss: 0.33
- Precision, Recall, F1-Score calculated with `sklearn`
- Confusion matrix visualized with `seaborn`

  Model performed well across all classes. Most errors occurred in similar categories (e.g., automobile vs truck).

---

5. Transfer Learning

Model Choice: MobileNetV2

- Chosen for lightweight architecture and strong performance
- Better suited for CIFAR-10 compared to heavier models like VGG16
- Also tested fine-tuning: unfreezing last 50 layers for improved adaptation

Result:

- Accuracy increased from ~85% (frozen) to **90%+** after fine-tuning

---

6. Code Quality

Standards Used:

- **Ruff** for linting and formatting (configured via `pyproject.toml`)
- Modular, readable code with:
  - Model setup
  - Training logic
  - Inference functions
  - Deployment scripts

Code is PEP8-compliant, clean, and well-documented.

---

7. Report

Contents:

- Project objective and dataset description
- Step-by-step architecture explanation
- Preprocessing and training strategy
- Evaluation metrics with charts
- Transfer learning justification
- Deployment approach and insights

Report is written in Markdown and easily convertible to PDF or HTML.

---

8. Model Deployment

a) Gradio App (Primary)

- Interactive UI with image upload + prediction
- Shows top 3 predicted classes with confidence scores
- Local and Hugging Face Spaces ready
- Launch with: `python gradio_app.py`

b) Flask API (Optional)

- `POST /predict` endpoint for image classification
- Accepts single image, returns class probabilities in JSON
- Deployable via **Render.com** using `Procfile`

Final Performance Summary

| Model | Accuracy | Notes | |---|---|---| | Custom CNN | ~75% | 3-layer CNN from scratch | | MobileNetV2 (frozen) | ~85% | Base model frozen | | MobileNetV2 (fine-tuned) | **90.2%** | Last 50 layers unfrozen |

Insights & Learnings

- Transfer learning significantly reduces training time and improves results on small datasets
- Fine-tuning only part of the base model helps avoid overfitting
- Gradio is an efficient way to deploy and demo models without heavy backend work
- Code quality tools like Ruff elevate your project to production-ready status

---

Future Improvements

- Add data augmentation (flips, crops, etc.)
- Compare results from VGG16 or EfficientNet
- Improve Gradio UI layout and style
- Add batch prediction functionality
- Containerize with Docker for easier deployment

Tools Used

| Category | Tools | |---|---| | Deep Learning | TensorFlow, Keras | | Visualization | Matplotlib, Seaborn | | Linting/Format | Ruff | | Deployment | Gradio, Flask, Render, Hugging Face | | Project Tools | Git, VS Code, Google Colab |

Author

Your Name Here
GitHub: yourusername
Email: your.email@example.com