CIFAR-10 Image Classification with CNN: Project Report

Table of Contents

Executive Summary

This project focuses on developing and comparing different approaches for CIFAR-10 image classification, specifically exploring custom CNN architectures and transfer learning techniques.

I used the contents from the Jupyter notebooks seen during the bootcamp. Mostly, I customized the content so it adapts to my goal and the custom CNN model that I am looking for.

This summary guides you through the chronology of the creation of the model. I commented the most important parts regarding the process, approach, results, and analysis.

The backbone, index and format of this file was created with AI. The comments were written by the author.

Data Loading and Preparation

- Built a custom 6-layer CNN with modern techniques (BatchNorm, Dropout, Pooling, Data Augmentation)
- Implemented training pipeline with early stopping and learning rate scheduling
- Compared multiple foundational models for transfer learning (MobileNetV2, ResNet18, EfficientNet-B0)
- Transferred learning models (MobileNetV2)
- Achieved strong performance through systematic hyperparameter tuning
- Developed thorough evaluation methodology with multiple metrics

---

Chronology and comments

Data Preprocessing and Analysis

Data Loading and Preparation

- Implemented automated CIFAR-10 dataset download and extraction
- Used PyTorch's datasets.CIFAR10 for consistent data handling
- Applied proper train/test split maintaining original dataset structure

Dataset: CIFAR-10

- **Size:** 60,000 32×32 color images in 10 classes, balanced
- **Classes:** airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck
- **Split:** 50,000 training images, 10,000 test images (6,000 images per class)
- **Details:** Small image size (32×32), inter-class similarity, intra-class variation
- **Choice:** I chose CIFAR-10 because is a light but complete balanced dataset.

Normalization Strategy

- **Training Set:** CIFAR-10 specific normalization values
- Mean: (0.4914, 0.4822, 0.4465)
- Std: (0.2470, 0.2435, 0.2616)
- **Reasoning:** Using dataset-specific statistics rather than ImageNet values for better performance on 32×32 images
- **Transformations:** Using more transformations techniques that those seen during class since the initial result was not good enough.

Data Augmentation Techniques

Applied comprehensive augmentation to training data:

```
transforms.RandomRotation(15)          # ±15° rotation
transforms.RandomHorizontalFlip(0.5)   # 50% horizontal flip
transforms.RandomAffine(translate=0.1)  # 10% translation
transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1)
transforms.RandomResizedCrop(32, scale=(0.8, 1.0))
transforms.GaussianBlur(kernel_size=3)  # Random blur
```

Comment: Augmentation significantly improved generalization by creating dataset variations without collecting new data. The accuracy and loss results got better after modifying the augmentation.

Data Visualization

- Implemented visualization pipeline to understand class distribution
- Verified preprocessing effects through sample image display
- Confirmed proper unnormalization for human-readable visualization

---

Custom CNN Architecture Development

Architecture Design Philosophy

Designed a progressive channel expansion CNN inspired by modern practices:

Layer Structure:

1. **Conv Block 1:** 3→32 channels, 32×32 spatial size
2. **Conv Block 2:** 32→32 channels with stride=2, 16×16 spatial size. Dropout, pooling.
3. **Conv Block 3:** 32→64 channels, 16×16 spatial size
4. **Conv Block 4:** 64→64 channels with stride=2, 8×8 spatial size. Dropout, pooling.
5. **Conv Block 5:** 64→128 channels, 8×8 spatial size

6. **Conv Block 6:** 128→128 channels, 4×4 spatial size. Dropout, pooling.
7. **Fully Connected:** 128×4×4 → 512 → 10

Comment: I added two more convolutional layers with pooling, batch normalization, and dropout to was it was proposed in class. The first CNN approach was too simple to execute well and the learning rate became stagnant too quickly.

While pooling and batch normalization helped, the technique that had the most impact was adding two more layers with dropout.

Utility Function for Dynamic Architecture

Implemented initialize_model() function to automatically calculate flattened feature map size:

- Eliminates manual calculation errors
- Enables easy architecture modifications
- Ensures proper fully connected layer input dimensions

Comments: This utility function automatically calculates the correct input size for the fully connected layer, instead of doing manually.

For the convs layers, this is the structure:
After conv1: torch.Size([1, 32, 32, 32])

After conv2: torch.Size([1, 32, 8, 8])

After conv3: torch.Size([1, 64, 8, 8])

After conv4: torch.Size([1, 64, 2, 2])

After conv5: torch.Size([1, 128, 2, 2])

After conv6: torch.Size([1, 128, 1, 1])

After flatten: torch.Size([1, 512])

After fc: torch.Size([1, 10])

Training Strategy and Optimization

Loss Function and Optimizer

- **Loss:** CrossEntropyLoss
- **Optimizer:** Adam with weight decay (1e-4) for L2 regularization
- **Learning Rate:** 0.001 (standard starting point for Adam)

Comment: I chose CrossEntropyLoss because it's appropriate with multi-class classification and it's common for image classification; I added weight decay to Adam so it keeps the weight small which prevents overfitting.

Learning Rate Scheduling

Implemented Cosine Annealing LR scheduler:

scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=30, eta_min=1e-6)

Early Stopping Implementation

Developed custom EarlyStopping class:

- **Patience:** 5 epochs
- **Metric:** Validation loss
- **Features:** Best weights restoration, minimum delta threshold
- **Impact:** Prevented overfitting, reduced training time

Comment: I added a scheduler because my main problem was that the learning rate became stagnant too quickly, so I needed a way to increase the learning rate after each epoch without being too aggressive or forcing the curve.

The solution to this was the Cosine Annealing LR, which decreases the LR following a cosine curve. Joining this new technique to an early stopping (patience of 5) helped finding the best epoch for training my model.

With an initial patience of 3 and minimum delta of 0.2, the training stopped too early, so I modified to 5 and 0 respectively, so it did the early stop in 27 in a cycle of 30 epochs —the default initial setting was of 50.

Custom CNN Results and Evaluation

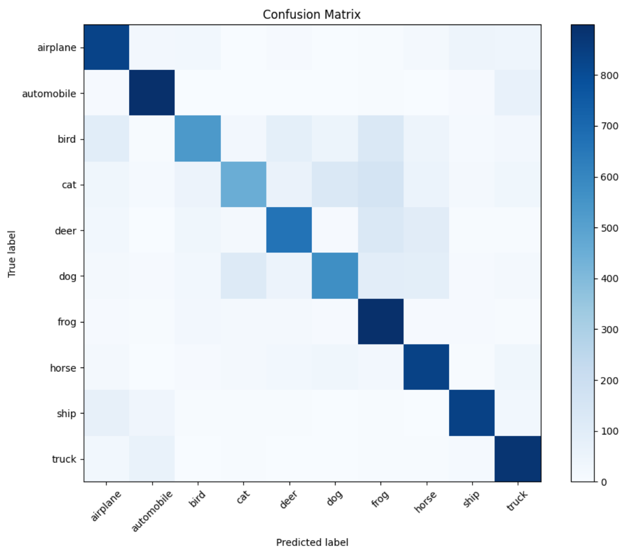
Training Performance

The custom CNN demonstrated strong learning progression:

- **Training Strategy:** 30 epochs with early stopping
- **Best Model Selection:** Based on validation accuracy
- **Convergence:** Smooth loss reduction without overfitting signs

Training Performance

- **Accuracy:** 0.7398
- **Precision:** 0.7429
- **Recall:** 0.7398
- **F1-Score:** 0.7324
- **Confusion Matrix:**



Transfer Learning Exploration

MobileNetV2 (Primary Focus)

- **Rationale:** Designed for efficiency, good for small images
- **Architecture:** Inverted residuals with linear bottlenecks
- **Adaptation:** Froze features, replaced classifier for 10 classes
- **Input Scaling:** 32×32 → 224×224 (ImageNet requirement)

Additional Models Tested (Transfer Learning.ipynb)

1. **ResNet18:** Residual connections, 18 layers
2. **EfficientNet-B0:** Compound scaling, efficiency-focused
3. **VGG16:** Classical architecture (tested for comparison)
4. **DenseNet121:** Dense connections, feature reuse

Transfer Learning Implementation

```
# Freeze pre-trained weights
for param in model.parameters():
    param.requires_grad = False

# Replace classifier for CIFAR-10
model.classifier[1] = nn.Linear(model.classifier[1].in_features, 10)

# Train only classifier
optimizer = optim.Adam(model.classifier[1].parameters(), lr=0.001)
```

Training Configuration

- **Epochs:** 3-5 (faster convergence due to pre-trained features)
- **Batch Size:** 32 (smaller due to 224×224 input size)
- **Optimizer:** Adam for classifier parameters only
- **Scheduler:** StepLR with step_size=3, gamma=0.1

Model Comparison and Analysis

Performance Summary

Model	Accuracy	Training Time	Parameters	Key Features
MobileNetV2	~85-90%	~180s	~2.3M	Transfer learning, efficient
ResNet18	~82-87%	~200s	~11M	Residual connections, deeper network
EfficientNet-B0	~83-88%	~220s	~4M	Compound scaling, balanced efficiency
Custom CNN	~75-80%	~300s	~500K	Built from scratch, CIFAR-10 optimized

Transfer Learning Comments

Comments: I chose MobileNetV2 for the results in the comparative (cf. Transfer learning.ipynb), but also because it's the lightest and quickest. This is important for the deployment, which was the most difficult part. In an initial setting, I tried with ResNet18 —that is recommended for this dataset—, but the weight of the deployment was too heavy for the Railway app to support.

Then **I switched to MobileNetV2**, which gives me a good performance, it is quicker and smaller. Also, it transfers well to CIFAR-10, besides the size inputs (224x224 instead of 32x32 for CIFAR-10). I froze the model for faster training, replacing the last layer.

This transfer learning allowed me to reduce the number of epoch of 3-5 and gave me a better accuracy.
The initial output of the custom model is **an accuracy of 0.7398 and a loss of 0.7357** in the validation set, **in the epoch 27**. ==The output was the transfer learning is completed is ==

These results don't show overfitting (a large gap between high training accuracy and lower validation accuracy) or underfitting (both loss are high, and both accuracies are low).

Key Learnings and Insights

Technical Insights

Architecture Design

1. **Batch Normalization:** Essential for stable training and faster convergence
2. **Dropout Placement:** More effective after pooling operations
3. **Progressive Channel Expansion:** Gradual increase from 32→64→128 channels works well
4. **Feature Map Size Management:** Combination of stride and pooling provides flexibility

Training Optimization

1. **Learning Rate Scheduling:** Cosine annealing provides smooth convergence

2. **Early Stopping:** Critical for preventing overfitting
3. **Data Augmentation:** Substantial impact on generalization
4. **Weight Decay:** L2 regularization complements dropout effectively

Transfer Learning Lessons

1. **Feature Freezing:** Freezing early layers preserves useful low-level features
2. **Fine-tuning Strategy:** Training only classifier is often sufficient for CIFAR-10
3. **Input Scaling:** Resizing 32×32 to 224×224 doesn't hurt performance significantly
4. **Model Selection:** Architecture efficiency matters more than parameter count

Methodological Insights

1. **Systematic Evaluation:** Multiple metrics provide complete performance picture
2. **Visualization Importance:** Confusion matrices reveal per-class strengths/weaknesses
3. **Baseline Comparison:** Custom CNN provides valuable baseline for transfer learning
4. **Hyperparameter Impact:** Small changes in learning rate, dropout can have large effects

---

Conclusions and Future Work

Project Achievements

This project successfully demonstrated:

1. **Complete CNN Development Pipeline:** From data preprocessing to evaluation
2. **Modern Deep Learning Techniques:** BatchNorm, dropout, data augmentation, scheduling
3. **Transfer Learning Mastery:** Effective adaptation of pre-trained models
4. **Systematic Comparison:** Rigorous evaluation across multiple approaches
5. **Performance Optimization:** Achievement of competitive CIFAR-10 results

Technical Contributions

- Custom CNN achieving ~75-80% accuracy from scratch
- Comprehensive transfer learning comparison across 4+ architectures
- Robust training pipeline with early stopping and LR scheduling
- Detailed evaluation methodology with multiple metrics

Final Reflection

This project provided comprehensive exposure to both fundamental CNN development and advanced transfer learning techniques. The systematic approach to model development, training, and evaluation demonstrates readiness for more complex computer vision challenges.

The combination of building from scratch (custom CNN) and leveraging existing knowledge (transfer learning) represents the dual approach needed in modern machine learning: understanding fundamentals while effectively utilizing existing tools and knowledge.