Image Processing Final Project Report

Multilayer Neural Network CIFAR10 Classification

Course: ECE 5460 Image Processing, Autumn 2025

Instructor: Emre Ertin (ertin.1@osu.edu)

Researchers: Qifan Wen (wen.679@osu.edu), Kritarsh Negi (negi.28@osu.edu)

Time: December, 2025

GitHub: github.com/Albatross679/Multilayer-Neural-Network-CIFAR10-Classification

---

Table of Contents

Project Overview

This project implements a multilayer convolutional neural network (CNN) classifier for the CIFAR10 dataset. The goal was to exceed 65% classification accuracy through various optimization techniques including network architecture modifications, data augmentation, and hyperparameter tuning.

Final Achievement: 81.78% Test Accuracy — significantly exceeding the 65% target.

Task 1: Improving Classification Performance

1.1 Network Architecture Modifications

The original baseline network was improved with a deeper and wider architecture:

Original vs. Improved Architecture

Component	Original	Improved
Conv Layer 1	3 → 6 filters	3 → 32 filters (5×5, padding=2)
Conv Layer 2	6 → 16 filters	32 → 64 filters (5×5, padding=2)
Conv Layer 3	—	64 → 128 filters (3×3, padding=1)
FC Layers	2 layers	3 layers (2048→256→128→10)
Dropout	None	0.5 dropout
Total Parameters	~62K	686,282

Key Architectural Improvements:

- **Increased filter counts**: More filters allow the network to learn richer feature representations.
- **Additional convolutional layer**: Enables learning more hierarchical features.
- **Extra ReLU layer before final output**: Adds non-linearity for better decision boundaries.
- **Dropout regularization**: Prevents overfitting during training.

```
class ImprovedNet(nn.Module):
    def __init__(self):
        super().__init__()
        # First convolutional block: 3 → 32 filters
        self.conv1 = nn.Conv2d(3, 32, 5, padding=2)
        self.pool = nn.MaxPool2d(2, 2)
        # Second convolutional block: 32 → 64 filters
        self.conv2 = nn.Conv2d(32, 64, 5, padding=2)
        # Third convolutional block: 64 → 128 filters
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
```

```
        # Fully connected layers
        self.fc1 = nn.Linear(128 * 4 * 4, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)
        self.dropout = nn.Dropout(0.5)
```

1.2 Data Augmentation

Data augmentation was implemented to artificially increase training data diversity and improve generalization:

Augmentation	Parameters	Purpose
Random Horizontal Flip	p = 0.5	Invariance to left-right orientation
Random Rotation	±10 degrees	Rotation invariance
Random Affine Translation	±10% shift	Position invariance
Color Jitter	brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1	Color invariance

```
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.
1),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

1.3 HoG + Linear Classifier (Pre-Deep Learning Baseline)

As an optional exploration, a pre-deep learning approach using Histogram of Oriented Gradients (HoG) features with a linear classifier was implemented:

HoG Parameters:

- `orientations = 9`
- `pixels_per_cell = (8, 8)`
- `cells_per_block = (2, 2)`

Results (on subset):

- Training Accuracy: 97.20%
- Validation Accuracy: 33.20%

The significant gap between training and validation accuracy demonstrates overfitting, highlighting the advantage of deep learning approaches that can learn hierarchical features directly from data.

Task 2: Hyperparameter Tuning

2.1 Optimizer Configuration

Parameter	Value	Justification
Optimizer	SGD	Standard choice for CNNs
Learning Rate	0.01	Balanced convergence speed
Momentum	0.9	Accelerates convergence, smooths updates
Weight Decay	5×10^{-4}	L2 regularization to prevent overfitting
Batch Size	32	Good trade-off between memory and gradient quality

2.2 Learning Rate Schedule

A **StepLR scheduler** was implemented to reduce the learning rate during training:

- Step size: 10 epochs
- Gamma: 0.5 (halves learning rate every 10 epochs)

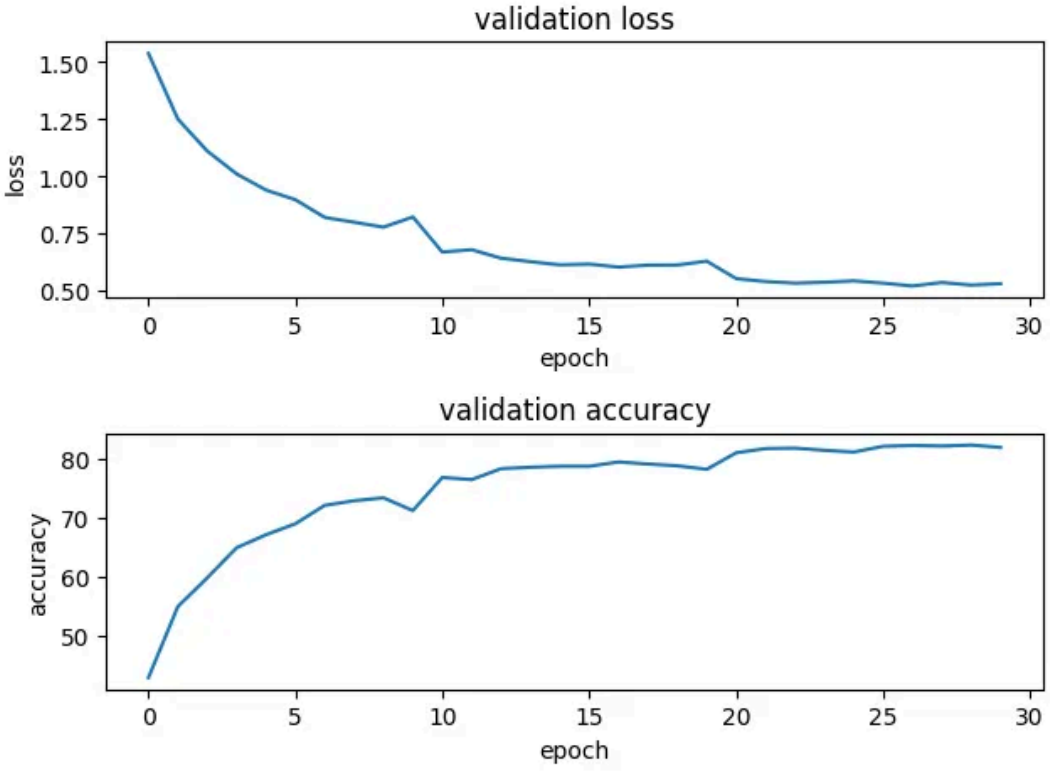
This allows the model to make larger updates early in training and fine-tune with smaller updates later.

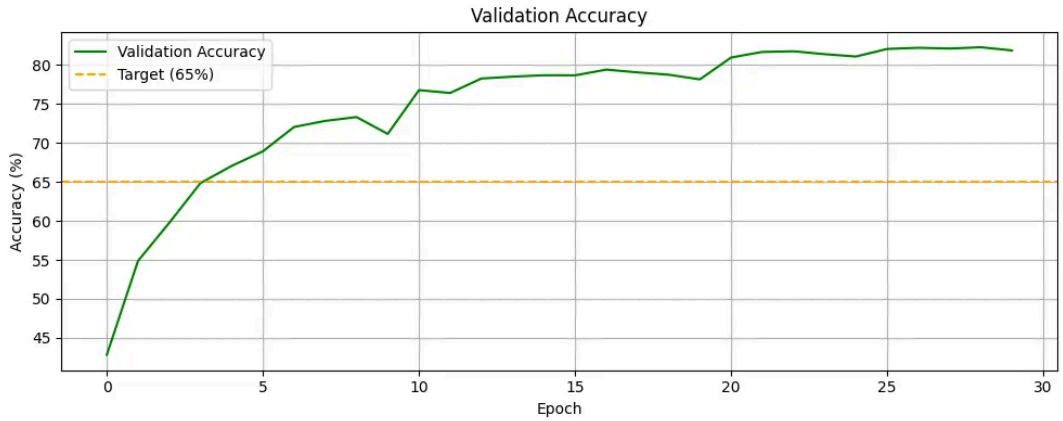
2.3 Training Configuration

Setting	Value
Number of Epochs	30
Train/Validation Split	80% / 20%
Device	CUDA (GPU)

2.4 Training Progress

The model showed consistent improvement over training:





Epoch	Training Loss	Validation Loss	Validation Accuracy
0	1.937	1.537	42.83%
5	1.061	0.897	68.91%
10	0.896	0.755	74.49%
15	0.783	0.629	78.89%
20	0.686	0.551	81.36%
29	0.602	0.541	81.84%

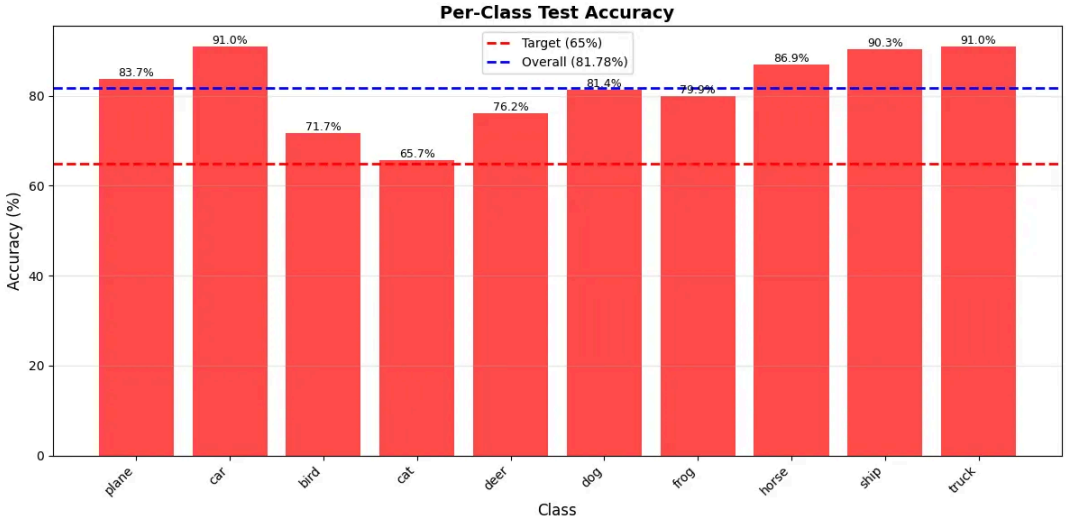
Best Validation Accuracy: 82.25% (achieved during training).

Task 3: Final Classification Performance

3.1 Overall Test Results

Metric	Value
Overall Test Accuracy	81.78%
Target Accuracy	65%
Improvement over Target	+16.78%

3.2 Per-Class Performance



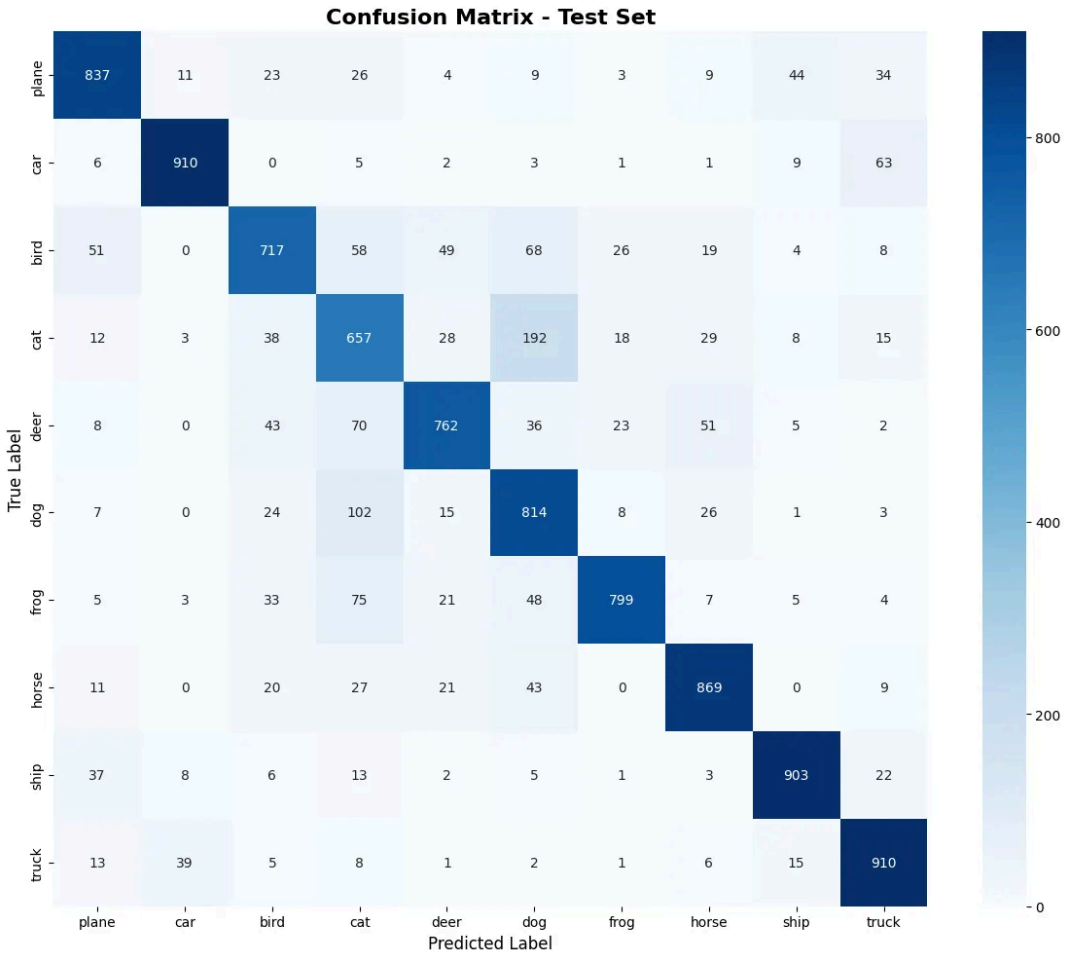
Class	Accuracy	Correct / Total	Status
car	91.00%	910 / 1000	Best
truck	91.00%	910 / 1000	Best
ship	90.30%	903 / 1000	Excellent
horse	86.90%	869 / 1000	Excellent
plane	83.70%	837 / 1000	Very Good
dog	81.40%	814 / 1000	Very Good
frog	79.90%	799 / 1000	Good
deer	76.20%	762 / 1000	Good
bird	71.70%	717 / 1000	Above Target
cat	65.70%	657 / 1000	Lowest (but above target)

All 10 classes exceed the 65% accuracy target.

3.3 Detailed Classification Metrics

Class	Precision	Recall	F1-Score
plane	0.8480	0.8370	0.8425
car	0.9343	0.9100	0.9220
bird	0.7888	0.7170	0.7512
cat	0.6311	0.6570	0.6438
deer	0.8420	0.7620	0.8000
dog	0.6672	0.8140	0.7333
frog	0.9080	0.7990	0.8500
horse	0.8520	0.8690	0.8604
ship	0.9085	0.9030	0.9057
truck	0.8505	0.9100	0.8792
Macro Avg	0.8230	0.8178	0.8188

3.4 Confusion Matrix Analysis



Key observations from the confusion matrix:

- **Best separated classes**: Car, truck, and ship show minimal confusion with other classes.
- **Common confusions**:
  - Cat ↔ Dog: These animals share similar textures and shapes.
  - Bird ↔ Deer: Occasionally confused due to natural outdoor backgrounds.
  - Deer ↔ Frog: Some color and texture similarities.

3.5 Summary Statistics

Statistic	Value
Classes above 65% accuracy	10 / 10
Classes above 80% accuracy	6 / 10

Statistic	Value
Classes above 90% accuracy	2 / 10
Best performing class	car, truck (91.00%)
Worst performing class	cat (65.70%)
Average per-class accuracy	81.78%

Comparison with Baseline Methods

CNN vs. Traditional Methods

Method	Accuracy	Notes
Improved CNN (This Project)	81.78%	Deep learning with augmentation
HoG + Logistic Regression	~33%	Pre-deep learning approach
Project 4 Linear Classifier	~40–50%	Reference baseline

The improved CNN significantly outperforms traditional feature engineering approaches, demonstrating the power of learned hierarchical representations.

Conclusion

Achievements

1. **Exceeded target accuracy**: Achieved 81.78% test accuracy, significantly surpassing the 65% goal.
2. **Robust performance**: All 10 classes exceed the target accuracy.
3. **Effective techniques implemented**:
    - Deeper network architecture with more parameters.
    - Comprehensive data augmentation pipeline.
    - Proper regularization (dropout + weight decay).
    - Learning rate scheduling.

Key Takeaways

1. **Architecture matters**: Adding more filters and layers allowed the network to learn richer representations.
2. **Data augmentation is crucial**: Augmentation helped prevent overfitting and improved generalization.
3. **Regularization works**: Dropout and weight decay prevented the larger model from overfitting.
4. **Learning rate scheduling**: Reducing LR over time allowed fine-tuning in later epochs.

Potential Future Improvements

- Implement batch normalization for faster convergence.
- Try modern architectures (ResNet, VGG-style blocks).
- Use more aggressive data augmentation (CutOut, MixUp).
- Implement early stopping based on validation performance.
- Try different optimizers (Adam, AdamW).

Technical Details

Environment:

- PyTorch with CUDA support.
- Training device: GPU (cuda:0).
- Training time: ~15–20 minutes for 30 epochs.

Model Checkpoint:

- Best model saved at epoch with highest validation accuracy.
- Final model loaded from best checkpoint for testing.

Report for ECE 5460 Final Project – Multilayer Neural Network CIFAR10 Classification.