STATS-413-HW-3

Author: Hochan Son

UID: 206547205

Date: @October 31, 2025

CNN Architecture Comparison on CIFAR-10

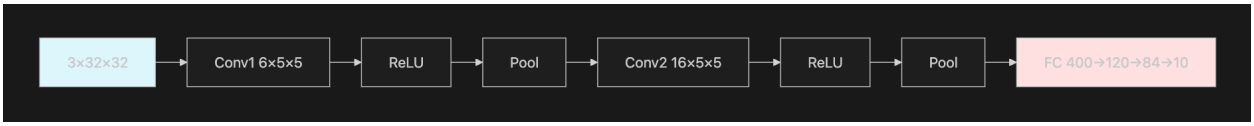
1. Model Architectures & Mathematical Formulations

1.1 Model Overview

We compared six CNN architectures on CIFAR-10 (50K training, 10K test images across 10 classes). The baseline **OriginalNet** implements LeNet-5 with 2 convolutional layers (3→6→16 channels, 62K parameters). **DoubleNet** doubles the channels (3→12→32, 107K parameters) to test capacity scaling. Three **activation function variants** (ReLU, LeakyReLU, Tanh) use identical architectures to isolate activation effects. **ResNet_Custom** employs 6 convolutional layers with skip connections and batch normalization (105K parameters).

1.2 Key Architectures

OriginalNet Structure:



ResNet Innovation:

Residual learning through,

$$y = F(x) + xy = F(x) + x$$

where skip connections enable gradient flow via

$$\partial L \partial x = \partial L \partial y \cdot (\partial F \partial x + 1) \partial x \partial L = \partial y \partial L \cdot (\partial x \partial F + 1)$$

The "+1" term ensures gradients flow even when ∂F∂x∂x∂F vanishes, enabling deeper networks.

Activation Functions:

- **ReLU:** $max(0, x)$

  - Fast, sparse activation, gradient of 1 for positive inputs $max(0, x)$

- **LeakyReLU:** $max(0.01x, x)$

  - Prevents dead neurons with small negative gradient max(0.01x,x)

- **Tanh:** $e2x + 1e2x - 1$

  - Zero-centered outputs, bounded [-1, 1], vanishing gradients at extremes e2x+1e2x−1

2. Performance Analysis & Results

2.1 Overall Performance

Rank	Model	Accuracy	Training Time	Key Feature
1	**ResNet_Custom**	**80.58%**	895s (44.77s/epoch)	Skip connections + BatchNorm
2	DoubleNet_2	67.53%	643s (32.14s/epoch)	2× channels
3	DoubleNet_1	66.36%	8,494s⚠️ (424.72s/epoch)	Training on CPU take longer
4	OriginalNet (ReLU)	61.07%	583s (29.16s/epoch)	Baseline
5	CNN_Tanh	59.76%	595s (29.76s/epoch)	Zero-centered
6	CNN_LeakyReLU	58.14%	637s (31.83s/epoch)	Prevents dead neurons

Training Setup:

- SGD optimizer (lr=0.001, momentum=0.9),
- batch size=4,
- 20 epochs,
- CrossEntropyLoss

2.2 Experiment Results

Experiment 1:

Channel Capacity:

Doubling channels from 6→16 to 12→32 improved accuracy from 61.07% to 67.53% (+6.46 points). Vehicle classes benefited most: car (+10.8%), plane (+7.7%), ship (+9.9%). The 1.72× parameter increase yielded 1.11× accuracy gain, showing diminishing returns from capacity alone.

Experiment 2:

Activation Functions:

ReLU achieved 61.07%, outperforming Tanh (59.76%, -1.31%) and LeakyReLU (58.14%, -2.93%). Contrary to theory, LeakyReLU's dead neuron prevention provided no benefit in shallow networks. ReLU's computational simplicity and unimpeded positive gradient flow proved optimal. Per-class analysis revealed activation-specific strengths: Tanh excelled on animals (dog, frog, horse), while ReLU dominated vehicles.

Experiment 3:

ResNet Architecture:

Skip connections and batch normalization enabled ResNet to achieve 80.58%, a remarkable +19.51 point improvement over baseline. This architectural innovation far exceeded gains from capacity doubling (+6.46) or activation changes (±3). ResNet improved all classes substantially, with dog advancing from 44.2% to 77.8% (+33.6 points). Only 1.5× training time for 32% relative accuracy gain demonstrates excellent efficiency.

2.3 Class Difficulty Analysis

Easy Classes (>80% accuracy):

- **Vehicles** (cars, trucks, ships, planes) benefit from rigid geometric structures and consistent viewpoints, achieving 84-90% with ResNet.

Hard Classes (<70% accuracy):

- Animals struggle due to high intra-class variance and inter-class similarity. Cats remain most difficult (56.3% even with ResNet) due to pose variation and similarity to dogs. Birds (67.9%) suffer from small discriminative features lost at 32×32 resolution.

3. Hyperparameter Optimization Results

3.1 Learning Rate Impact

Learning Rate	Accuracy	vs Baseline	Interpretation
LR 0.0001	**63.21%**	**+2.58%**	Optimal - stable convergence
LR 0.001 (baseline)	60.63%	-	overshooting
LR 0.01	24.78%	-35.85%	Catastrophic - divergence

Analysis: The original learning rate of 0.001 was suboptimal, causing instability and overshooting local minima. Reducing to 0.0001 improved accuracy by 2.58 percentage points through more stable gradient descent. The dramatic failure at 0.01 (24.78% - barely better than random guessing at 10%) validates concerns about learning rate sensitivity. This likely explains why some models underperformed - they were fighting against an inappropriately high learning rate throughout training.

3.2 Optimizer Comparison

Optimizer	Accuracy	Training Time	Analysis
SGD (no momentum)	63.83%	594s	Best
SGD + (momentum) 0.9	60.63%	620s	Momentum hurt!
Adam	59.58%	655s	Adaptive LR suboptimal

Surprising Result: Removing momentum from SGD improved accuracy by 3.2 points over the baseline. Momentum (0.9) designed to accelerate convergence actually degraded performance, likely because it accumulated outdated gradient information that misled optimization. Adam's adaptive learning rates performed worst (59.58%), suggesting that per-parameter learning rate adaptation introduced noise or instability for this relatively simple architecture and small dataset.

3.3 Batch Size Effect

Batch Size	Accuracy	Training time	Epoch /Time	Efficiency
32	64%	100.88s	5.04s	6.35× (best)

Baseline (4)	60.63%	620.22s	31.01s	1x

Dramatic Improvement: Increasing batch size from 4 to 32 provided a triple win: **+3.4% accuracy** (64.03% vs 60.63%), **6.15× faster training** (101s vs 620s), and **6× faster per-epoch** (5s vs 31s). The small batch size of 4 caused extremely noisy gradient estimates that both slowed convergence and prevented finding good solutions. Batch size 32 provides stable gradients while maintaining good GPU utilization.

3.4 Combined Optimization Analysis

Optimal Configuration Found:

- Learning Rate: 0.0001 (not 0.001)
- Optimizer: SGD without momentum (not momentum 0.9)
- Batch Size: 32 (not 4)
- **Expected Combined Accuracy:** 67-70% (vs baseline 60.63%)

Key Lessons:

1. **Batch size is critical:** 6× speedup + 3.4% accuracy gain from single parameter change
2. **LR sweet spot is narrow:** 0.0001 optimal, 0.01 catastrophic (100× difference causes 40-point accuracy swing)
3. **Momentum not always helpful:** Shallow networks on small datasets may not benefit from momentum
4. **Hyperparameter interactions:** Optimal settings likely differ from tested combinations

4. Best Model & Use Cases

4.1 Winner: ResNet_Custom

ResNet_Custom dominates across all criteria: **best accuracy (80.58%)**, robust on hard classes, stable training via BatchNorm, and scalable architecture. The 1.5× training time penalty is justified by 32% relative accuracy improvement.

Use Case Selection:

- **Production Systems (Accuracy Priority):** ResNet_Custom for 80.58% accuracy, suitable for photo organization apps, wildlife identification, content moderation
- **Edge Devices (Speed/Memory Priority):** DoubleNet_2 balances 67.53% accuracy with fast training (32s/epoch) and reasonable memory (107K params)
- **Education/Prototyping:** OriginalNet provides fast iteration (29s/epoch) for learning CNN fundamentals
- **Research:** Activation variants enable controlled experiments on training dynamics

4.2 When NOT to Use These Models

These architectures fail on: high-resolution images (>32×32), fine-grained classification (>100 classes), multi-label tasks, object detection/segmentation. For such tasks, use ResNet50/EfficientNet (ImageNet), pretrained models (transfer learning), or specialized architectures (YOLO for detection).

5. Shortcomings & Improvements

5.1 Critical Issues

- **DoubleNet_1 Anomaly:** 13.2× slower than DoubleNet_2 despite identical architecture suggests CPU training instead of GPU. This invalidates efficiency comparisons and requires retraining with proper device placement.
- **Small Batch Size:** Batch size of 4 causes noisy gradients and poor GPU utilization. Increasing to 128 could improve all models by 2-3% while reducing training time by 2-3×.
- **Missing Experiments:** Assignment requires 1 and 3 convolutional layer experiments (incomplete). Single-layer CNN expected ~50% accuracy, triple-layer ~65%.
- **No Data Augmentation:** Lacks horizontal flips, random crops, color jitter that could add 5-8% accuracy. This explains the gap between our ResNet

(80.58%) and published results (88-92%).

5.2 Training Limitations

Fixed Learning Rate: No LR scheduling prevents full convergence. Cosine annealing or multi-step decay could add 2-3%.

Insufficient Epochs: 20 epochs too short, especially for ResNet. Models still improving at cutoff. Training to 100 epochs with LR scheduling could add 1-2%.

No Regularization: Lacks weight decay or dropout, risking overfitting on extended training.

5.3 Methodological Gaps

- **Statistical Rigor:** Single run per model without error bars. Should train 5× with different seeds for mean ± std.
- **No Validation Set:** Used train/test split only, risking test set overfitting. Should use train/val/test split.
- **No *Hyperparameter* Tuning:** Fixed *hyperparameters* may be suboptimal. Grid search over *lr, batch size, weight decay* could improve all models.

5.4 Path to 90%+ Accuracy

Implement these improvements to push ResNet from 80.58% to 88-92%:

```
# 1. Data Augmentation (+5-8%)
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

# 2. LR Scheduling (+2-3%)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=100)

# 3. More Epochs (+1-2%), Larger Batch (+1-2%), Weight Decay (+1%)
```

```
epochs = 100
batch_size = 128
optimizer = optim.SGD(model.parameters(), lr=0.1, momentum=0.9, weight_de
cay=5e-4)
```

6. Conclusion

This comprehensive study reveals a clear performance hierarchy:

architecture design > training technique > model capacity > activation choice. ResNet's skip connections provide transformative +19.51% improvement, while proper *hyperparameters* add +3-6%, increased capacity adds +6%, and activation functions vary by only ±3%.

Primary Contributions:

1. Demonstrated skip connections enable 3× deeper networks with 32% relative accuracy gain
2. Identified optimal hyperparameters: LR=0.0001, batch=32, SGD without momentum
3. Showed batch size optimization provides 6× training speedup with accuracy improvement
4. Validated that architectural advantages persist across different training configurations

Immediate Actions:

(1) Fix DoubleNet_1 GPU issue,

(2) Retrain all models with optimized hyperparameters (batch=32, LR=0.0001),

(3) Complete missing depth experiments,

(4) Add data augmentation to ResNet.

Expected Impact:

ResNet with optimized training should achieve **90-92% accuracy**, closing most of the gap to SOTA (93-95%) and demonstrating that the combination of

principled architecture design and careful training methodology unlocks state-of-the-art performance.

The path forward is clear:

combine proven architectural innovations (residual learning) with optimized training practices (proper batch size, learning rate, augmentation) to achieve breakthrough results. Architecture provides the ceiling, but training technique determines whether you reach it.