Group Equivariant Convolutional Neural Network

for Translation-Invariant Digit Recognition

1 Overview

This project implements a **group equivariant convolutional neural network** that leverages mathematical group theory to achieve robust translation invariance for MNIST digit recognition. Unlike standard CNNs that learn translation invariance implicitly through data augmentation, this architecture enforces equivariance through explicit group-theoretic design.

1.1 Key Innovation

The network decomposes the 28×28 input space using a **quotient group structure** G/H, where:

- $G = \mathbb{Z}_{28} \times \mathbb{Z}_{28}$ is the full translation group
- $H = 7\mathbb{Z}_{28} \times 7\mathbb{Z}_{28}$ is a subgroup of grid translations
- $|G/H| = 16$ represents 16 cosets, each corresponding to a 7×7 spatial region

2 Mathematical Framework

2.1 Quotient Group Decomposition

Definition The translation group G acts on the 28×28 image space. We define a normal subgroup H consisting of translations by multiples of 7:

$$H = \{(7k, 7m) : k, m \in \mathbb{Z}_{28}\}$$

Quotient Group The quotient group G/H has exactly 16 cosets:

$$G/H = \{gH : g \in G\} \quad \text{with} \quad |G/H| = \frac{|G|}{|H|} = \frac{28 \times 28}{7 \times 7} = 16$$

Each coset is represented by a **representative element** $(7h, 7w)$ where $h, w \in \{0, 1, 2, 3\}$.

2.2 Group Equivariance

Equivariance Condition A function $\Phi : X \to Y$ is equivariant with respect to group actions if:

$$\Phi(T_g x) = \rho(g)\Phi(x), \quad \forall g \in G$$

where:

- T_g is the action of g on the input space
- $\rho(g)$ is the corresponding action on the feature space

In our implementation, $\rho(g)$ is realized through **cyclic permutations** of the 16 feature maps.

3 Architecture Components

3.1 Layer 0: Quotient Group Projection

Purpose Project the input image onto the quotient group space G/H by averaging over local translations within each coset.

Algorithm For each representative (r_h, r_w):

1. Apply convolution: $f = \text{ReLU}(\text{Conv}(x))$
2. For each offset $(o_h, o_w) \in \{0, \ldots, 6\}^2$ (49 offsets):
   - Compute shifted feature: $f_{shifted} = T_{(r_h+o_h, r_w+o_w)} f$
3. Average: $\bar{f}_{(r_h,r_w)} = \frac{1}{49} \sum_o f^{(o)}_{shifted}$

Output 16 feature maps, one per coset representative.

Mathematical Significance This operation induces local translation invariance within each 7×7 region while preserving global spatial structure through the quotient group.

3.2 Equivariant Layers

Design Principle Each equivariant layer maintains the group structure by applying shared convolutions to all cyclic permutations of the 16 feature maps.

Algorithm

1. Input: list of 16 feature maps $\{f_0, f_1, \ldots, f_{15}\}$
2. For each permutation index $p \in \{0, \ldots, 15\}$:
   (a) Circularly permute: $\{f_p, f_{p+1}, \ldots, f_{15}, f_0, \ldots, f_{p-1}\}$
   (b) Concatenate: $x_{cat} = \text{concat}(f_p, f_{p+1}, \ldots)$
   (c) Convolve: $g_p = \text{ReLU}(\text{Conv}(x_{cat}))$
3. Output: list of 16 new feature maps $\{g_0, g_1, \ldots, g_{15}\}$

Equivariance Guarantee The cyclic permutation corresponds to the group action on G/H, ensuring:

$$\Phi(T_g x) = \sigma(g)\Phi(x)$$

where $\sigma(g)$ permutes the 16 output features accordingly.

3.3 Cyclic Shift Operation

The fundamental operation is the **cyclic shift**:

$$\text{cyclic\_shift}(x, h, w)[i, j] = x[(i - h) \mod 28, (j - w) \mod 28]$$

This implements the torus topology where opposite edges wrap around, making the translation group action well-defined.

4 Training Strategy

4.1 Multi-Position Parallel Supervision

Motivation To prevent the model from learning only a subset of spatial positions, we supervise all 16 quotient group positions simultaneously.

Loss Function Given a batch of images with digit labels y:

1. Forward pass produces 16 logit vectors: $\{z_0, \ldots, z_{15}\}$, each $\in \mathbb{R}^{10}$
2. Concatenate and reshape: $Z \in \mathbb{R}^{B \times 16 \times 10}$
3. Compute cross-entropy for all positions:

$$\mathcal{L} = \frac{1}{16B} \sum_{i=1}^{B} \sum_{p=0}^{15} \text{CrossEntropy}(z_p^{(i)}, y_i)$$

Implementation

```
# Reshape logits: [B, 16*10] -> [B, 16, 10]
logits_reshaped = logits.view(B, 16, num_classes)

# Flatten to [B*16, 10] and repeat labels 16 times
digit_loss = F.cross_entropy(
    logits_reshaped.view(-1, num_classes),
    labels.repeat_interleave(16)
)
```

4.2 Random Translation Augmentation

Data-Level Robustness In addition to architectural equivariance, we apply random cyclic translations during training:

```
shifts_h = torch.randint(0, 28, (batch_size,))
shifts_w = torch.randint(0, 28, (batch_size,))
x = cyclic_shift(x_orig, shifts_h, shifts_w)
```

Dual Guarantee

- **Architectural**: Equivariance enforced by group-theoretic layer design
- **Data-level**: Robustness enhanced via random translation augmentation

5 Experimental Validation

5.1 Test A: Translation Robustness

We evaluate model performance under various translation ranges:

Translation Range	Test Shifts
Small (0-5 pixels)	(1,1), (2,1), (3,3), (4,2), (5,3)
Medium (7-12 pixels)	(8,9), (11,5), (9,7), (12,3), (10,8)
Large (18-24 pixels)	(21,12), (15,5), (18,18), (24,7)
Random	5 random translations

Expected Result The equivariant CNN should maintain high accuracy across all translation ranges, while the standard CNN's performance degrades significantly with larger shifts.

5.2 Test B: Mathematical Equivariance Verification

Principle For a base translation (a, b), adding grid translations $(7k, 7m)$ should produce equivalent results due to quotient group structure.

Test Protocol

1. Choose base shift: $(a, b) = (3, 5)$
2. Apply additional grid shifts: $(0, 7), (14, 0), (7, 7), (21, 7), (28, 0), (28, 14)$
3. Measure digit accuracy for each total shift: $(a + 7k, b + 7m)$
4. Compute standard deviation of accuracies

Success Criterion Low standard deviation (< 0.01) indicates strong equivariance maintenance.

6 Theoretical Insights

6.1 Why Quotient Groups?

Continuous vs Discrete Natural images have continuous translation symmetry, but digital images are discrete. The quotient group G/H provides a principled way to:

- Discretize the continuous translation space
- Reduce computational complexity from $28^2 = 784$ to 16
- Maintain mathematical equivariance properties

6.2 Local Invariance, Global Equivariance

Layer 0 Averaging By averaging over 49 local translations within each 7×7 region, we achieve:

$$\text{Local invariance}: \quad f(T_h x) \approx f(x) \quad \text{for } h \in H$$

Equivariant Layers Subsequent layers maintain equivariance with respect to the quotient group:

$$\text{Global equivariance}: \quad \Phi(T_g x) = \rho(g)\Phi(x) \quad \text{for } g \in G/H$$

This design balances between full translation invariance (losing spatial information) and translation sensitivity (poor generalization).

7 Output and Comparison

```
1. Standard CNN
================================================================================
Number of parameters: 29,738

Training standard CNN...
--------------------------------------------------------------------------------
Epoch 0: Loss=0.2185, Val Acc=0.9792, Time=5.9s
Epoch 1: Loss=0.0587, Val Acc=0.9829, Time=5.8s
Training completed! Best Val Acc: 0.9829


2. Equivariant CNN
================================================================================
  Number of representatives: 16
  Kernel size: 49
  Layer 0 computation: 16 × 49 = 784
Number of parameters: 237,098

Training Equivariant CNN
--------------------------------------------------------------------------------
  Batch 0/500, DigitLoss: 2.3028
  Batch 100/500, DigitLoss: 1.8818
  Batch 200/500, DigitLoss: 1.7136
  Batch 300/500, DigitLoss: 1.4546
  Batch 400/500, DigitLoss: 0.9423
Epoch 0: AvgLoss=1.4652, ValDigitAcc=0.8165, Time=45.6min
--------------------------------------------------------------------------------
  Batch 0/500, DigitLoss: 0.5460
  Batch 100/500, DigitLoss: 0.5531
  Batch 200/500, DigitLoss: 0.5093
  Batch 300/500, DigitLoss: 0.3974
  Batch 400/500, DigitLoss: 0.4598
Epoch 1: AvgLoss=0.4017, ValDigitAcc=0.9182, Time=52.9min
--------------------------------------------------------------------------------
Training completed! Best digit accuracy: 0.9182
```

```
3. Test A: Robustness to different ranges of translation
================================================================================

Test Category            Standard CNN    Equivariant CNN Improvement
--------------------------------------------------------------------------------
Small shifts (0-5 pixels) 0.5287         0.9215          +0.3928
Medium shifts (7-12 pixels) 0.0350         0.9135          +0.8784
Large shifts (18-24 pixels) 0.1041         0.9076          +0.8035
Random translations       0.1272         0.9102          +0.7830

================================================================================
4. Test B: Equivariance property verification (only digit accuracy)
================================================================================

Verifying Equivariance Property (base translation=(3, 5)):
--------------------------------------------------------------------------------
  Shift( 3,12) [+grid(0,7)]: Digit accuracy=0.9100
  Shift(17, 5) [+grid(14,0)]: Digit accuracy=0.9064
  Shift(10,12) [+grid(7,7)]: Digit accuracy=0.9083
  Shift(24,12) [+grid(21,7)]: Digit accuracy=0.9120
  Shift(31, 5) [+grid(28,0)]: Digit accuracy=0.9228
  Shift(31,19) [+grid(28,14)]: Digit accuracy=0.9199

  Digit recognition accuracy: 0.9132 ± 0.0060
  Conclusion: ✓ Equivariance maintained well

Verifying Equivariance Property (base translation=(2, 4)):
--------------------------------------------------------------------------------
  Shift( 2,11) [+grid(0,7)]: Digit accuracy=0.9097
  Shift(16, 4) [+grid(14,0)]: Digit accuracy=0.9095
  Shift( 9,11) [+grid(7,7)]: Digit accuracy=0.9049
  Shift(23,11) [+grid(21,7)]: Digit accuracy=0.9116
  Shift(30, 4) [+grid(28,0)]: Digit accuracy=0.9204
  Shift(30,18) [+grid(28,14)]: Digit accuracy=0.9129

  Digit recognition accuracy: 0.9115 ± 0.0047
  Conclusion: ✓ Equivariance maintained well
```

8 Running the Code

8.1 Requirements

```
pip install torch torchvision numpy
```

8.2 Execution

```
python ml.py
```

8.3 Expected Output

1. Training logs for standard CNN and equivariant CNN
2. Robustness comparison across different translation ranges
3. Equivariance property verification with statistical measures