# Hands-on: Image Representation Learning with SimCLR

PyTorch Implementation Workflow

CIFAR-10 / STL-10

### 1 Dataset Setup

Load unlabeled dataset for pretraining

🗄️

### 2 Implement SimCLR

Build encoder & projection head

PyTorch

💻

### 3 Data Augmentation

Setup augmentation pipeline

Crop + Color + Blur

🎨

### 4 Train Encoder

Optimize with contrastive loss

NT-Xent Loss

⚡

### 5 Linear Probe

Evaluate on labeled subset

Frozen Features

📊

### 6 Visualize Embeddings

Plot learned representations

t-SNE

📈

🎯 **Final Step:** Compare self-supervised results with supervised baseline performance

## 📊 Example Data and Calculation Process

## 1. Input Images & Augmentation

| Image | Size | Augmentation |
|---|---|---|
| Cat (Original) | 32×32×3 | - |
| Cat View 1 | 32×32×3 | Random Crop + Flip |
| Cat View 2 | 32×32×3 | Color Jitter + Blur |

💡 **Key Point:** Generate two different views from the same image to form a positive pair

## 2. Encoder & Projection Head

**Architecture:**
Input (3×32×32) → ResNet-18 → h (512-dim) → MLP → z (128-dim)

```
# Example embedding vectors
z1 = [0.23, -0.45, 0.67, ..., 0.12]  # 128-dim
z2 = [0.21, -0.43, 0.65, ..., 0.15]  # 128-dim (positive)
z3 = [-0.89, 0.34, -0.12, ..., 0.56] # 128-dim (negative)
```

💡 **Key Point:** Calculate cosine similarity after L2 normalization

## 3. NT-Xent Loss Calculation (Batch Size N=4, Total 8 Samples)

**NT-Xent Loss Formula:**
ℓ(i,j) = -log[ exp(sim(zi, zj)/τ) / Σ exp(sim(zi, zk)/τ) ]
*τ (temperature) = 0.5*

### 📋 Step 1: Calculate Similarity Matrix (8×8)

```
sim(z1, z2) = 0.92 // positive pair
sim(z1, z3) = 0.15 // negative
sim(z1, z4) = -0.23 // negative
sim(z1, z5) = 0.08 // negative
...
```

### 📋 Step 2: Temperature Scaling (τ=0.5)

```
sim(z1, z2)/τ = 0.92/0.5 = 1.84
sim(z1, z3)/τ = 0.15/0.5 = 0.30
sim(z1, z4)/τ = -0.23/0.5 = -0.46
...
```

### 📋 Step 3: Exponential Calculation

```
exp(1.84) = 6.297 // positive (numerator)
exp(0.30) = 1.350 // negative
exp(-0.46) = 0.631 // negative
exp(0.16) = 1.174 // negative
...
Denominator Sum = 6.297 + 1.350 + 0.631 + 1.174 + ... = 12.845
```

### 📋 Step 4: Loss Calculation

```
ℓ(z1, z2) = -log(6.297 / 12.845)
ℓ(z1, z2) = -log(0.490)
ℓ(z1, z2) = 0.713
```

Final Batch Loss (Average)

**Loss = 0.713**

🎯 **Interpretation:** Lower loss indicates that the model has learned good representations where positive pairs are close and negative pairs are far apart.

## 📈 Training Results Example

## ⚡ Self-Supervised Pretraining

| Epoch | Loss | Time |
|---|---|---|
| 1 | 2.456 | 8 min |
| 50 | 1.234 | - |
| 100 | 0.876 | - |
| 200 | 0.623 | 26 hours |

**Settings:** CIFAR-10 unlabeled, batch=256, ResNet-18

## 📊 Linear Probe Evaluation (CIFAR-10 Test)

| Method | Accuracy | Training Data |
|---|---|---|
| **Random Init** | 23.4% | 5,000 labels |
| **SimCLR (ours)** | 78.6% | 5,000 labels |
| **Supervised** | 91.2% | 50,000 labels |

Self-Supervised Improvement

**+55.2%**

## 💻 PyTorch Implementation (NT-Xent Loss)

```python
# SimCLR NT-Xent Loss Implementation
def nt_xent_loss(z_i, z_j, temperature=0.5):
    """
    Args:
        z_i, z_j: [batch_size, embedding_dim] - positive pair
    Returns:
        loss: scalar tensor
    """
    batch_size = z_i.shape[0]

    # L2 normalization
    z_i = F.normalize(z_i, dim=1)
    z_j = F.normalize(z_j, dim=1)
```

```
    # Concatenate: [2*batch_size, embedding_dim]
    z = torch.cat([z_i, z_j], dim=0)

    # Similarity matrix: [2N, 2N]
    similarity_matrix = torch.mm(z, z.T) / temperature

    # Mask to exclude self-similarity
    mask = torch.eye(2 * batch_size, dtype=torch.bool)
    similarity_matrix = similarity_matrix.masked_fill(mask, -9e15)

    # Positive pairs: (i, N+i) and (N+i, i)
    positives = torch.cat([
        torch.diag(similarity_matrix, batch_size),
        torch.diag(similarity_matrix, -batch_size)
    ])

    # NT-Xent loss calculation
    nominator = torch.exp(positives)
    denominator = torch.sum(torch.exp(similarity_matrix), dim=1)
    loss = -torch.log(nominator / denominator)

    return loss.mean()
```

💡 **Key Insight:** Self-supervised learning can learn powerful feature representations without labels, achieving high performance with only a small amount of labeled data.