NYU

# Final Project: PCA + MLP vs CNN for Object Localization

ECE-4563 Machine Learning Project — NYU Tandon

**PRESENTED BY** **Max Ma, Yanbo Wang**

## Problem Statement

- Object localization is a fundamental computer vision problem
- Goal: predict the bounding box of an object in an image
- We study localization of a single MNIST digit placed on a larger canvas
- Compare a **classical ML pipeline** with an **end-to-end deep learning approach**

## Dataset & Task Definition

- Synthetic **MNIST-on-Canvas** dataset
- Input: 64 × 64 grayscale image
- Output: bounding box parameters

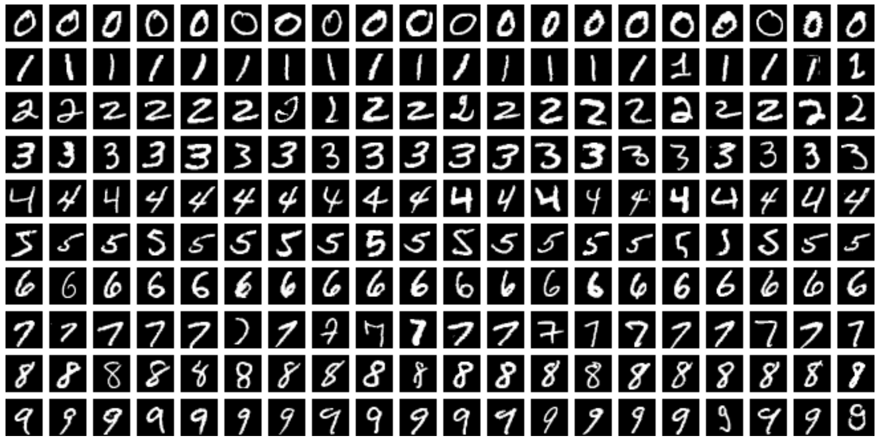


Figure 1: Sample images from MNIST test dataset

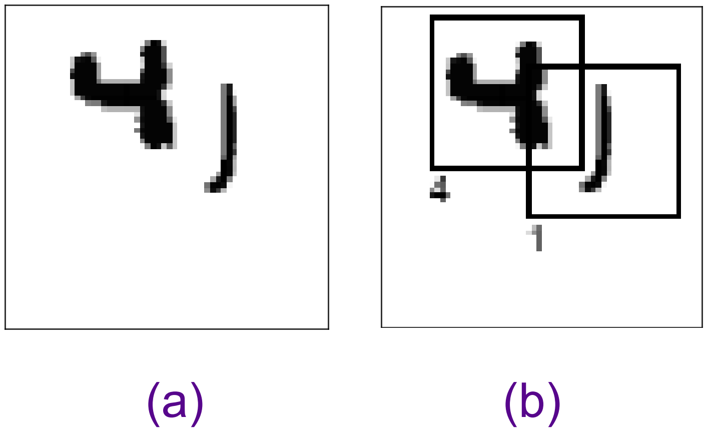


Figure 2: (a) Input, (b) Expected Output

# Problem Formulation

**Training Data:**

$$\{(X_i, y_i)\}_{i=1}^{N}$$

$$where\ x_i \in \mathbb{R}^{64\times64}, y_i = (x_c, y_c, w, h) \in [0,1]^4$$

**We choose MSE as the Loss function:**

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\|\widehat{\mathrm{y}_\mathrm{i}} - \mathrm{y}_\mathrm{i}\|_2^2$$

# Training Procedure

## 1. Data Splitting and Preparation:

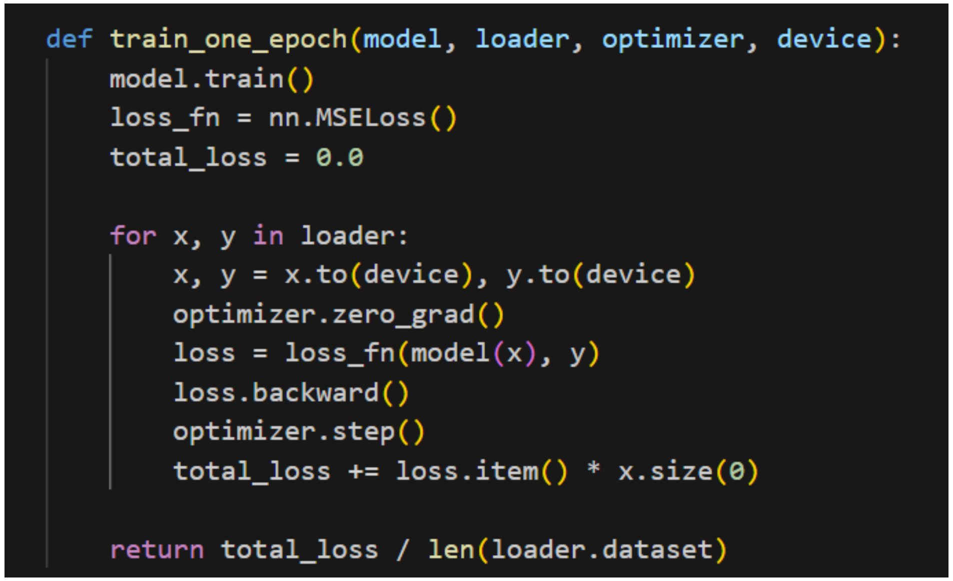
- Dataset is split into a pool set and a held-out test set
- Pool set is used for training and validation
- Test set is used only for final evaluation
- Prevents information leakage during model selection

```
# Hold-out TEST set (never used during CV)
X_pool, X_test, y_pool, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

## Training Procedure

### 2. Model Training (MLP / CNN):

- Models are trained using supervised learning
- Mini-batch gradient descent is used
- Loss function: Mean Squared Error (MSE)
- Optimizer: Adam
- Parameters updated via backpropagation



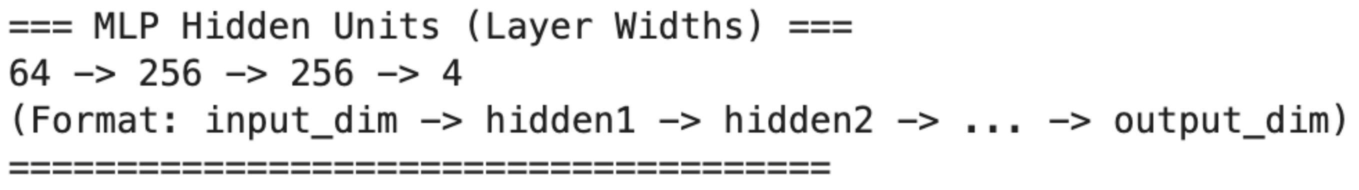
```python
def train_one_epoch(model, loader, optimizer, device):
    model.train()
    loss_fn = nn.MSELoss()
    total_loss = 0.0

    for x, y in loader:
        x, y = x.to(device), y.to(device)
        optimizer.zero_grad()
        loss = loss_fn(model(x), y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item() * x.size(0)

    return total_loss / len(loader.dataset)
```

# PCA + MLP

- Input 64×64 image was flattened to a 4069 dim-vector and standardized before entering MLP regressor
- MLP regressor predicts bounding box
- Sigmoid output activation applied for enforcing valid box range



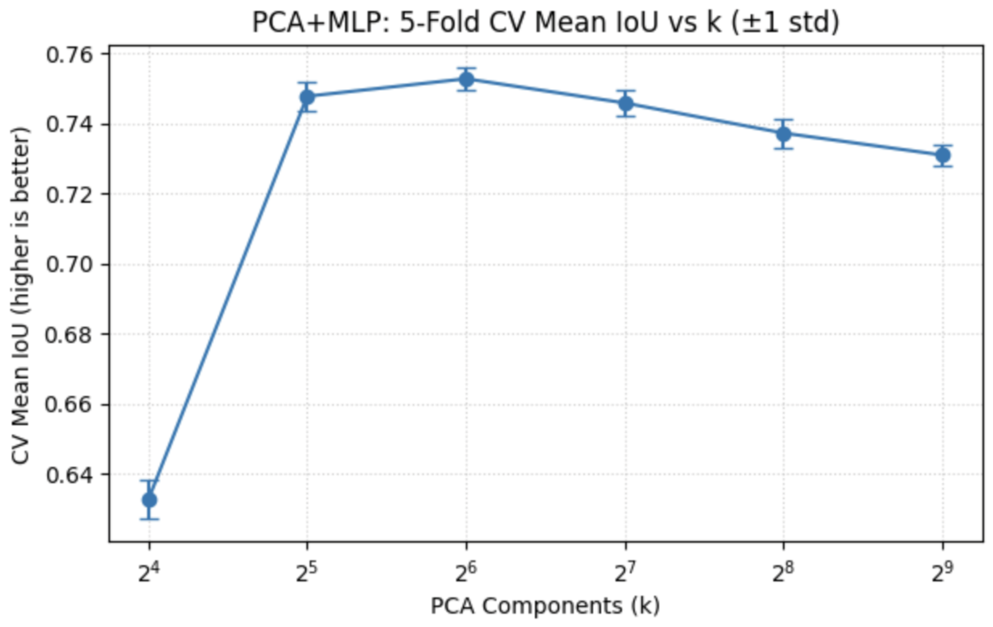
```
=== MLP Hidden Units (Layer Widths) ===
64 -> 256 -> 256 -> 4
(Format: input_dim -> hidden1 -> hidden2 -> ... -> output_dim)
======================================
```

MLP layer widths (example k=64)

# PCA Component Selection

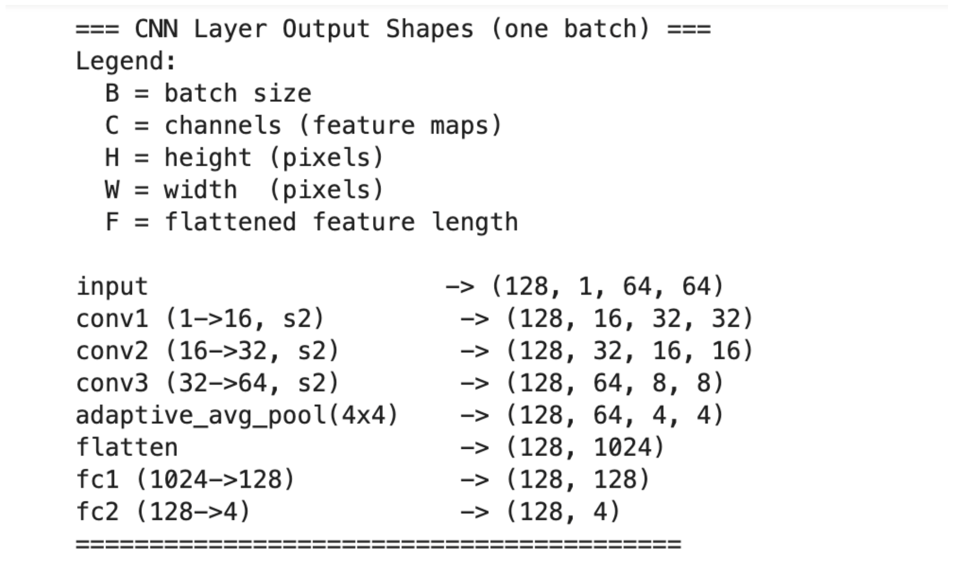
- A 5-Fold cross-validation was applied on training pool for selecting best dimensions from 16-512 with the highest IoU
- K-Fold cross-validation applied only on training set
- Best dimension count: k = 64

Best k by mean IoU: k = 64



# End-to-End CNN

- Constructed with 3 convolution blocks, 1 average pooling on 4 by 4 region and 2 linear layers
- Directly regressed the bounding box from the input image
- Trained with MSE loss



```
=== CNN Layer Output Shapes (one batch) ===
Legend:
  B = batch size
  C = channels (feature maps)
  H = height (pixels)
  W = width  (pixels)
  F = flattened feature length

input                    -> (128, 1, 64, 64)
conv1 (1->16, s2)         -> (128, 16, 32, 32)
conv2 (16->32, s2)        -> (128, 32, 16, 16)
conv3 (32->64, s2)        -> (128, 64, 8, 8)
adaptive_avg_pool(4x4)    -> (128, 64, 4, 4)
flatten                   -> (128, 1024)
fc1 (1024->128)           -> (128, 128)
fc2 (128->4)              -> (128, 4)
=========================================
```

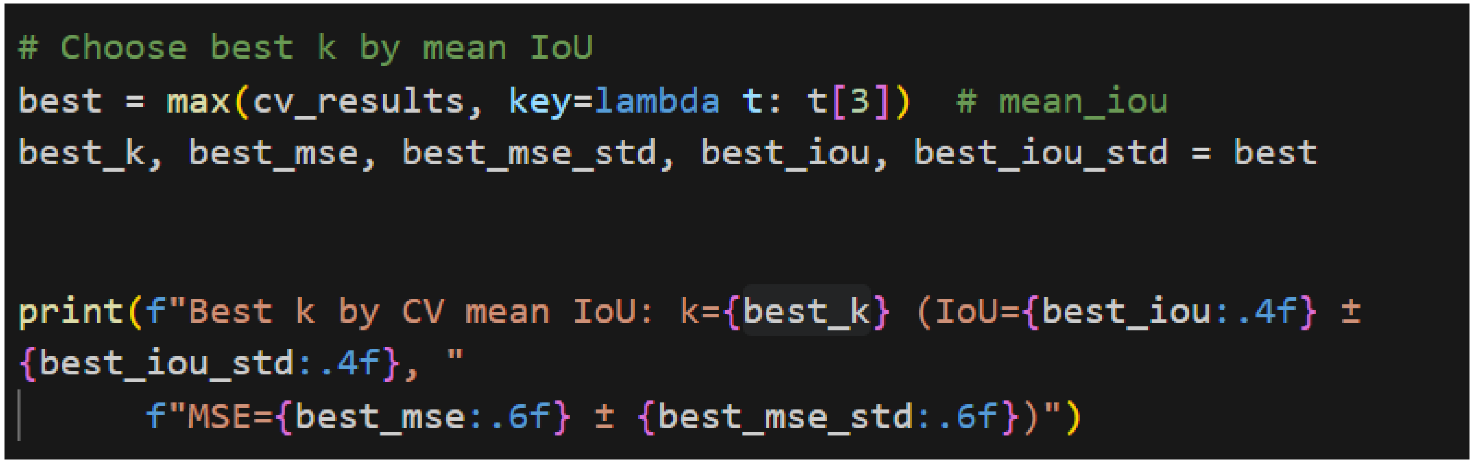
CNN tensor shapes for one batch (B,C,H,W)

# Training Procedure

## 3. Cross-Validation for PCA+MLP:

- PCA dimension is treated as a hyperparameter
- K-fold cross-validation is performed on the pool set
- Standardization and PCA are fit only on training folds
- Mean IoU across folds is used to select optimal PCA dimension

$$k^* = \arg\max_k \overline{\mathrm{IoU}}_k$$



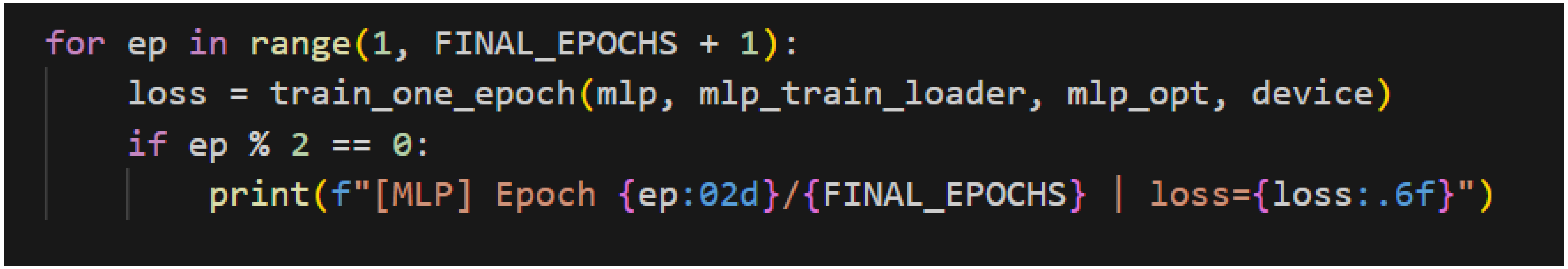
```
# Choose best k by mean IoU
best = max(cv_results, key=lambda t: t[3])  # mean_iou
best_k, best_mse, best_mse_std, best_iou, best_iou_std = best


print(f"Best k by CV mean IoU: k={best_k} (IoU={best_iou:.4f} ±
{best_iou_std:.4f}, "
      f"MSE={best_mse:.6f} ± {best_mse_std:.6f})")
```

# Training Procedure

## 4. Final Training on Full Pool Set:

- Final model is retrained using the entire pool set
- Same training procedure used as in cross-validation
- Fixed number of training epochs



```python
for ep in range(1, FINAL_EPOCHS + 1):
    loss = train_one_epoch(mlp, mlp_train_loader, mlp_opt, device)
    if ep % 2 == 0:
        print(f"[MLP] Epoch {ep:02d}/{FINAL_EPOCHS} | loss={loss:.6f}")
```

## Evaluation Results

- **PCA + MLP achieves competitive localization accuracy**
- **CNN achieves higher mean IoU on test set**
- **CNN benefits from spatial feature learning**
- **Both models evaluated on held-out test data**

**Table 1: Table for final MSE/IoU result for two models**

| Model | Test MSE (↓) | Mean IoU (↑) |
|---|---|---|
| PCA + MLP (k=64) | 0.000941 | 0.7825 |
| CNN Regressor | 0.000683 | 0.8134 |

# Visual Evaluation Results

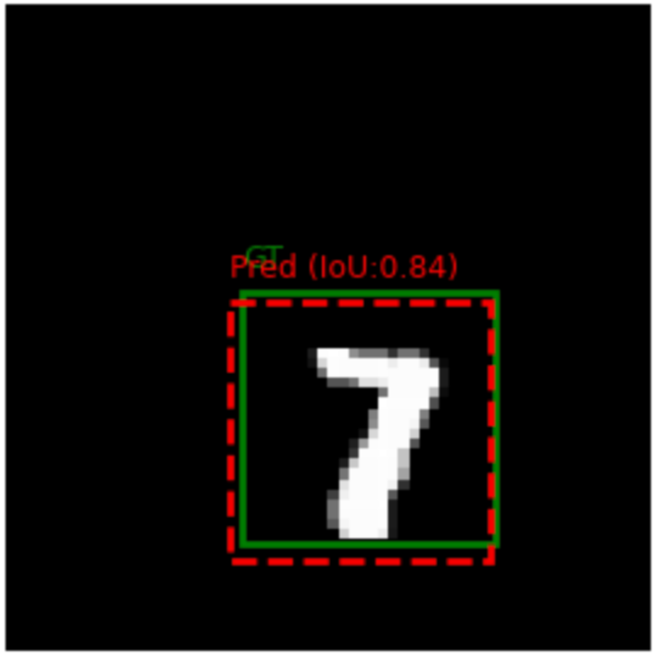


Figure 3: Sample for PCA+MLP Model Prediction VS. Ground Truth

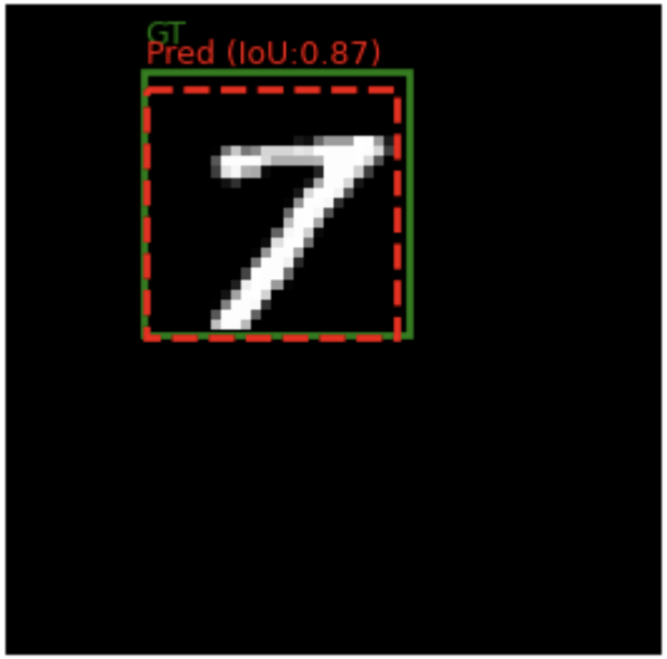


Figure 4: Sample for CNN Model Prediction VS. Ground Truth

## Conclusion

- PCA + MLP offers interpretability and efficiency
- CNN provides superior end-to-end performance
- Cross-validation is critical for fair model selection
- Demonstrates trade-offs between classical ML and deep learning