**Group Members: Jalen Spivey, EJ Binitie, Samuel Holison**

**Group Number: 17**

**Georgia State University**

**Name of Guide: Dr. Yue Wang**

**04/25/2025**

**Deep Learning Project:**

**Building Multi-Layer Perceptrons and**

**Convolutional Neural Networks (CNNs)**

**(1)Introduction**

* 1. **Background & Motivation**  
     Deep learning models like Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) have become the workhorses of image classification. However, most libraries “hide” the math. Our goal was to **manually implement** both an MLP and a simple CNN in NumPy, train and evaluate them on MNIST, and then wrap them in a Flask app to demonstrate real-time comparison.
  2. **Problem Definition**

Build from-scratch forward and backward passes for (a) a fully-connected MLP and (b) a single-filter CNN.

Train both on the MNIST dataset, compare their convergence and test accuracy.

Expose “Train” buttons in a web interface so a user can launch and monitor each model.

1. **System Overview**

**Data loading** via torchvision → NumPy arrays

**Model classes** (MLP, CNN) with forward(), backward(), and train() methods

**Training loops** in main() printing losses and final accuracy

**Flask app** with endpoints to trigger and poll each model in a background thread

**(3) Methodology**

**3.1Algorithm Design & Logic Flow**

* 1. **MLP**
     + Input: flattened 28×28 images → hidden layer → output logits → softmax
     + Activations: sigmoid in hidden layer, softmax at output
     + Loss: cross-entropy
     + Update: gradient descent from manually derived gradients
  2. **CNN**
     + Input: 28×28 image → single set of learnable 3×3 kernels → ReLU → flatten → FC layer → softmax
     + Loss & update analogous to MLP but with convolution gradients
  3. **Implementation Steps**
  4. **Convolution operation**

def convolve2d(img, kernel):

Hk, Wk = kernel.shape

H, W = img.shape

out = np.zeros((H-Hk+1, W-Wk+1))

for i in range(H-Hk+1):

for j in range(W-Wk+1):

out[i,j] = np.sum(img[i:i+Hk, j:j+Wk] \* kernel)

return out

* 1. **MLP forward & backward**

# Forward

z1 = x.dot(W1) + b1

a1 = sigmoid(z1)

z2 = a1.dot(W2) + b2

y\_hat = softmax(z2)

# Backward (derivation)

# dz2 = y\_hat - y\_true\_onehot

dW2 = a1.T.dot(dz2) / m

db2 = np.sum(dz2, axis=0) / m

da1 = dz2.dot(W2.T)

dz1 = da1 \* a1 \* (1 - a1)

dW1 = x.T.dot(dz1) / m

db1 = np.sum(dz1, axis=0) / m

* 1. **CNN forward & backward**

# Forward

conv\_out[f] = convolve2d(img, kernel[f])

relu\_out = np.maximum(0, conv\_out)

flat = relu\_out.reshape(batch, -1)

logits = flat.dot(W\_fc) + b\_fc

probs = softmax(logits)

# Backward

# d\_logits = probs - y\_onehot

dW\_fc = flat.T.dot(d\_logits) / batch

db\_fc = np.sum(d\_logits, axis=0) / batch

d\_flat = d\_logits.dot(W\_fc.T)

d\_relu = d\_flat.reshape(relu\_out.shape)

d\_conv = d\_relu \* (conv\_out > 0)

# gradient wrt kernels

for f in filters:

for i in batch:

d\_kernel[f] += convolve2d(x[i], d\_conv[i,f])

d\_kernel /= batch

* 1. **Putting it together** in train() methods: compute loss, call backward(), update weights.
* **Code Snippets**
  1. **MLP.forward() & backward()** (see above)
  2. **CNN.forward() & backward()** (see above)

**(4) Evaluation**

* 1. **Experimental Settings**
  + **Datasets**: MNIST train (60 000), test (10 000)
  + **Batch size**: 128
  + **Epochs**: MLP → 10; CNN → 5
  + **Learning rate**: 0.01
  1. **Demo Setup**

Flask endpoints /run\_mlp and /run\_cnn start training threads

/check\_\*\_status returns JSON with {status, result:{accuracy,…}}

* 1. **Observations & Results**

| **Model** | **Epochs** | **Final Train Loss** | **Test Accuracy** |
| --- | --- | --- | --- |
| MLP | 10 | ~0.18 | ~0.93 |
| CNN | 5 | ~0.12 | ~0.96 |

**CNN** converges faster and reaches higher accuracy, thanks to spatial weight sharing.

**MLP** takes more epochs and still lags behind, illustrating the power of convolution.

**(5) Learning Outcomes**

* 1. **Tasks Accomplished**
  + Manual NumPy implementations of MLP and CNN forward/backward
  + Empirical comparison on MNIST
  + Interactive Flask demo showing real-time training status
  1. **Team Contributions**

*Samuel*: designed and coded the CNN class, wrote convolution/backprop loops

*Amodou*: implemented MLP logic, sigmoid/backprop derivation, and cross-entropy loss

*Jalen*: Flask integration, threading, and front-end templates

1. **Unaccomplished Tasks / Failures**

Didn’t implement multiple filters or pooling layers due to time constraints

GUI could be polished further (progress bars, live charts)

1. **Lessons Learned**

Manually deriving gradients deepens understanding of deep learning internals

Thread safety in Flask requires careful state management

Clear division of tasks and frequent check-ins streamline teamwork