Part C: Comprehensive Analysis

This notebook conducts all experiments for Part C of the assignment.

1. **C1.1:** Learning Rate Analysis
2. **C1.2:** Batch Size Analysis
3. **C1.3:** Architecture Analysis
4. **C2:** Final Model Comparison

Intial Setup

In [1]:
```
import os
import sys
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import importlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
from sklearn.metrics import accuracy_score, confusion_matrix

# --- This is the "magic" to find your friends' code ---
current_dir = os.getcwd() # This is .../project-name/part_c
project_root = os.path.dirname(current_dir) # This is .../project-name/

# Add both 'src' folders to Python's path
mlp_src_path = os.path.join(project_root, 'MLP', 'src')
lc_src_path = os.path.join(project_root, 'linear_classifiers', 'src')
sys.path.append(mlp_src_path)
sys.path.append(lc_src_path)

# --- Now we can import all the helper files ---
# From MLP/src
import utils
import plots
import neural_network_scratch
import flexible_nn # The new file we created

# From linear_classifiers/src
import logistic_regression_scratch
import softmax_regression_scratch

# --- Reload modules to make sure we have the latest code ---
importlib.reload(utils)
importlib.reload(plots)
importlib.reload(neural_network_scratch)
```

```
importlib.reload(flexible_nn)
importlib.reload(logistic_regression_scratch)
importlib.reload(softmax_regression_scratch)

# --- Import the specific classes and functions we will use ---
from utils import load_transform_split_mnist, per_class_accuracy
from plots import plot_train_val_curves, plot_confusion_matrix, plot_per_class_acc
from neural_network_scratch import NeuralNetworkScratch
from flexible_nn import FlexibleNN
from logistic_regression_scratch import LogisticRegressionScratch
from softmax_regression_scratch import SoftmaxRegressionScratch

print("All libraries and helper modules loaded successfully.")
```

```
All libraries and helper modules loaded successfully.
```

Load Data

In [2]:
```
# Load the data
# classes=None loads all 10 digits
# batch_size=64 is our baseline
train_loader, val_loader, test_loader, train_set, val_set, test_set, _, _, _ = load
    classes=None,
    batch_size=64
)

print(f"Data loaded: {len(train_set)} train, {len(val_set)} val, {len(test_set)} te
```

```
100%|██████████| 9.91M/9.91M [00:03<00:00, 2.75MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 206kB/s]
100%|██████████| 1.65M/1.65M [00:01<00:00, 1.60MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 4.52MB/s]
Data loaded: 48000 train, 12000 val, 10000 test images.
```

C1.1: Learning Rate Analysis

- Testing 4 values: `[0.001, 0.01, 0.1, 1.0]`
- Using baseline architecture: 2 hidden layers `[128, 64]`
- Using baseline batch size: `64`
- Training for `20` epochs

In [3]:
```
# C1.1: Learning Rate Experiment
INPUT_DIM = 784
HIDDEN1 = 128
HIDDEN2 = 64
OUTPUT_DIM = 10
EPOCHS = 20
learning_rates = [0.001, 0.01, 0.1, 1.0]
lr_histories = {} # Dictionary to store results

print("Starting Learning Rate Analysis...")

for lr in learning_rates:
    print(f"\n--- Testing Learning Rate: {lr} ---")
```

```
    # Re-initialize the model for a fair test
    model_lr = NeuralNetworkScratch(INPUT_DIM, HIDDEN1, HIDDEN2, OUTPUT_DIM)

    # Train the model
    model_lr.fit(train_loader, val_loader, epochs=EPOCHS, lr=lr)

    # Save the history for plotting
    lr_histories[lr] = model_lr.history

print("\n--- Learning Rate Analysis Complete! ---")
```

```
Starting Learning Rate Analysis...

--- Testing Learning Rate: 0.001 ---
Epoch  1/20 | Train Loss: 2.1228, Train Acc: 33.24% | Val Loss: 1.9067,   Val Acc: 5
6.33%
Epoch  2/20 | Train Loss: 1.6720, Train Acc: 66.95% | Val Loss: 1.4378,   Val Acc: 7
3.42%
Epoch  3/20 | Train Loss: 1.2336, Train Acc: 76.48% | Val Loss: 1.0604,   Val Acc: 7
8.89%
Epoch  4/20 | Train Loss: 0.9348, Train Acc: 80.61% | Val Loss: 0.8352,   Val Acc: 8
1.47%
Epoch  5/20 | Train Loss: 0.7601, Train Acc: 82.66% | Val Loss: 0.7018,   Val Acc: 8
3.30%
Epoch  6/20 | Train Loss: 0.6530, Train Acc: 84.31% | Val Loss: 0.6165,   Val Acc: 8
4.72%
Epoch  7/20 | Train Loss: 0.5820, Train Acc: 85.56% | Val Loss: 0.5583,   Val Acc: 8
5.67%
Epoch  8/20 | Train Loss: 0.5320, Train Acc: 86.34% | Val Loss: 0.5155,   Val Acc: 8
6.66%
Epoch  9/20 | Train Loss: 0.4949, Train Acc: 87.03% | Val Loss: 0.4836,   Val Acc: 8
7.19%
Epoch 10/20 | Train Loss: 0.4663, Train Acc: 87.56% | Val Loss: 0.4592,   Val Acc: 8
7.59%
Epoch 11/20 | Train Loss: 0.4437, Train Acc: 88.04% | Val Loss: 0.4383,   Val Acc: 8
7.99%
Epoch 12/20 | Train Loss: 0.4253, Train Acc: 88.40% | Val Loss: 0.4217,   Val Acc: 8
8.36%
Epoch 13/20 | Train Loss: 0.4100, Train Acc: 88.72% | Val Loss: 0.4076,   Val Acc: 8
8.76%
Epoch 14/20 | Train Loss: 0.3969, Train Acc: 89.02% | Val Loss: 0.3955,   Val Acc: 8
8.97%
Epoch 15/20 | Train Loss: 0.3856, Train Acc: 89.27% | Val Loss: 0.3854,   Val Acc: 8
9.24%
Epoch 16/20 | Train Loss: 0.3757, Train Acc: 89.48% | Val Loss: 0.3760,   Val Acc: 8
9.47%
Epoch 17/20 | Train Loss: 0.3670, Train Acc: 89.69% | Val Loss: 0.3685,   Val Acc: 8
9.57%
Epoch 18/20 | Train Loss: 0.3592, Train Acc: 89.91% | Val Loss: 0.3606,   Val Acc: 8
9.84%
Epoch 19/20 | Train Loss: 0.3521, Train Acc: 90.07% | Val Loss: 0.3542,   Val Acc: 8
9.92%
Epoch 20/20 | Train Loss: 0.3456, Train Acc: 90.22% | Val Loss: 0.3479,   Val Acc: 9
0.08%

--- Testing Learning Rate: 0.01 ---
Epoch  1/20 | Train Loss: 0.9672, Train Acc: 76.15% | Val Loss: 0.4689,   Val Acc: 8
7.50%
Epoch  2/20 | Train Loss: 0.3948, Train Acc: 89.10% | Val Loss: 0.3529,   Val Acc: 9
0.05%
Epoch  3/20 | Train Loss: 0.3237, Train Acc: 90.82% | Val Loss: 0.3072,   Val Acc: 9
1.31%
Epoch  4/20 | Train Loss: 0.2875, Train Acc: 91.71% | Val Loss: 0.2803,   Val Acc: 9
1.92%
Epoch  5/20 | Train Loss: 0.2629, Train Acc: 92.48% | Val Loss: 0.2605,   Val Acc: 9
2.59%
Epoch  6/20 | Train Loss: 0.2431, Train Acc: 93.06% | Val Loss: 0.2464,   Val Acc: 9
```

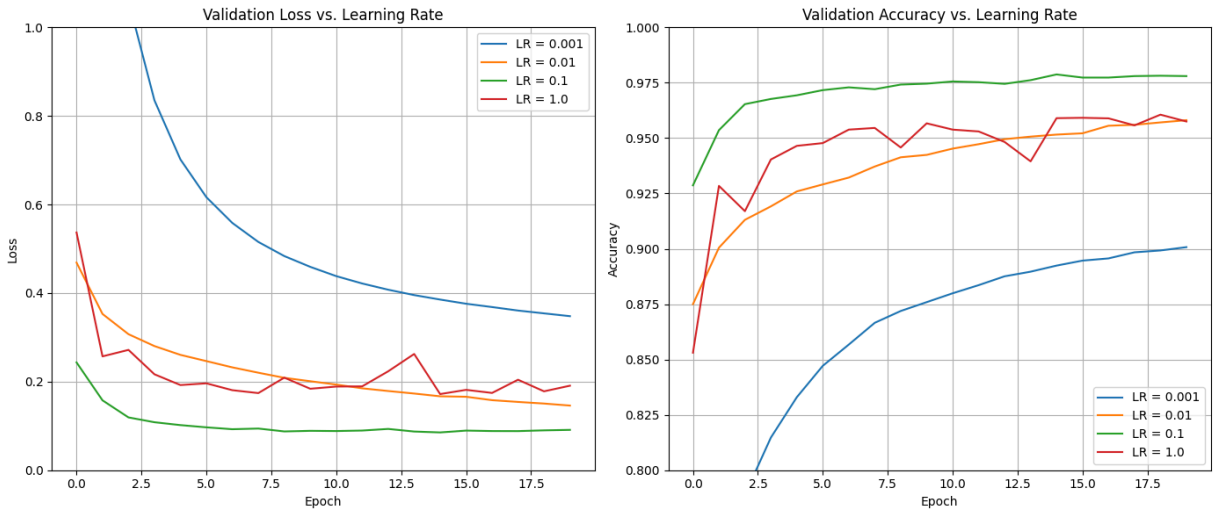
```
2.91%
Epoch  7/20 | Train Loss: 0.2268, Train Acc: 93.47% | Val Loss: 0.2323,   Val Acc: 9
3.22%
Epoch  8/20 | Train Loss: 0.2128, Train Acc: 93.85% | Val Loss: 0.2202,   Val Acc: 9
3.72%
Epoch  9/20 | Train Loss: 0.2008, Train Acc: 94.19% | Val Loss: 0.2087,   Val Acc: 9
4.13%
Epoch 10/20 | Train Loss: 0.1897, Train Acc: 94.43% | Val Loss: 0.2010,   Val Acc: 9
4.24%
Epoch 11/20 | Train Loss: 0.1803, Train Acc: 94.73% | Val Loss: 0.1934,   Val Acc: 9
4.53%
Epoch 12/20 | Train Loss: 0.1715, Train Acc: 94.90% | Val Loss: 0.1850,   Val Acc: 9
4.73%
Epoch 13/20 | Train Loss: 0.1638, Train Acc: 95.22% | Val Loss: 0.1789,   Val Acc: 9
4.95%
Epoch 14/20 | Train Loss: 0.1561, Train Acc: 95.44% | Val Loss: 0.1732,   Val Acc: 9
5.07%
Epoch 15/20 | Train Loss: 0.1495, Train Acc: 95.65% | Val Loss: 0.1670,   Val Acc: 9
5.16%
Epoch 16/20 | Train Loss: 0.1429, Train Acc: 95.85% | Val Loss: 0.1659,   Val Acc: 9
5.22%
Epoch 17/20 | Train Loss: 0.1373, Train Acc: 96.00% | Val Loss: 0.1582,   Val Acc: 9
5.56%
Epoch 18/20 | Train Loss: 0.1319, Train Acc: 96.24% | Val Loss: 0.1542,   Val Acc: 9
5.60%
Epoch 19/20 | Train Loss: 0.1268, Train Acc: 96.43% | Val Loss: 0.1506,   Val Acc: 9
5.71%
Epoch 20/20 | Train Loss: 0.1219, Train Acc: 96.56% | Val Loss: 0.1460,   Val Acc: 9
5.81%

--- Testing Learning Rate: 0.1 ---
Epoch  1/20 | Train Loss: 0.3760, Train Acc: 89.07% | Val Loss: 0.2434,   Val Acc: 9
2.87%
Epoch  2/20 | Train Loss: 0.1699, Train Acc: 94.99% | Val Loss: 0.1578,   Val Acc: 9
5.36%
Epoch  3/20 | Train Loss: 0.1202, Train Acc: 96.41% | Val Loss: 0.1192,   Val Acc: 9
6.53%
Epoch  4/20 | Train Loss: 0.0925, Train Acc: 97.28% | Val Loss: 0.1084,   Val Acc: 9
6.77%
Epoch  5/20 | Train Loss: 0.0748, Train Acc: 97.77% | Val Loss: 0.1019,   Val Acc: 9
6.93%
Epoch  6/20 | Train Loss: 0.0614, Train Acc: 98.15% | Val Loss: 0.0968,   Val Acc: 9
7.17%
Epoch  7/20 | Train Loss: 0.0514, Train Acc: 98.49% | Val Loss: 0.0926,   Val Acc: 9
7.29%
Epoch  8/20 | Train Loss: 0.0426, Train Acc: 98.75% | Val Loss: 0.0941,   Val Acc: 9
7.21%
Epoch  9/20 | Train Loss: 0.0355, Train Acc: 98.94% | Val Loss: 0.0876,   Val Acc: 9
7.42%
Epoch 10/20 | Train Loss: 0.0304, Train Acc: 99.09% | Val Loss: 0.0889,   Val Acc: 9
7.46%
Epoch 11/20 | Train Loss: 0.0250, Train Acc: 99.35% | Val Loss: 0.0885,   Val Acc: 9
7.56%
Epoch 12/20 | Train Loss: 0.0213, Train Acc: 99.43% | Val Loss: 0.0896,   Val Acc: 9
7.52%
Epoch 13/20 | Train Loss: 0.0176, Train Acc: 99.56% | Val Loss: 0.0932,   Val Acc: 9
```

```
7.45%
Epoch 14/20 | Train Loss: 0.0144, Train Acc: 99.68% | Val Loss: 0.0874,   Val Acc: 9
7.62%
Epoch 15/20 | Train Loss: 0.0113, Train Acc: 99.80% | Val Loss: 0.0853,   Val Acc: 9
7.88%
Epoch 16/20 | Train Loss: 0.0099, Train Acc: 99.81% | Val Loss: 0.0895,   Val Acc: 9
7.73%
Epoch 17/20 | Train Loss: 0.0081, Train Acc: 99.87% | Val Loss: 0.0885,   Val Acc: 9
7.73%
Epoch 18/20 | Train Loss: 0.0065, Train Acc: 99.91% | Val Loss: 0.0884,   Val Acc: 9
7.80%
Epoch 19/20 | Train Loss: 0.0055, Train Acc: 99.95% | Val Loss: 0.0900,   Val Acc: 9
7.82%
Epoch 20/20 | Train Loss: 0.0045, Train Acc: 99.97% | Val Loss: 0.0911,   Val Acc: 9
7.80%

--- Testing Learning Rate: 1.0 ---
Epoch  1/20 | Train Loss: 0.9585, Train Acc: 69.44% | Val Loss: 0.5368,   Val Acc: 8
5.31%
Epoch  2/20 | Train Loss: 0.3318, Train Acc: 90.64% | Val Loss: 0.2571,   Val Acc: 9
2.84%
Epoch  3/20 | Train Loss: 0.2535, Train Acc: 92.83% | Val Loss: 0.2717,   Val Acc: 9
1.70%
Epoch  4/20 | Train Loss: 0.2080, Train Acc: 94.11% | Val Loss: 0.2166,   Val Acc: 9
4.03%
Epoch  5/20 | Train Loss: 0.1890, Train Acc: 94.66% | Val Loss: 0.1923,   Val Acc: 9
4.65%
Epoch  6/20 | Train Loss: 0.1691, Train Acc: 95.11% | Val Loss: 0.1962,   Val Acc: 9
4.77%
Epoch  7/20 | Train Loss: 0.1541, Train Acc: 95.58% | Val Loss: 0.1807,   Val Acc: 9
5.38%
Epoch  8/20 | Train Loss: 0.1452, Train Acc: 95.83% | Val Loss: 0.1742,   Val Acc: 9
5.46%
Epoch  9/20 | Train Loss: 0.1273, Train Acc: 96.42% | Val Loss: 0.2090,   Val Acc: 9
4.58%
Epoch 10/20 | Train Loss: 0.1304, Train Acc: 96.26% | Val Loss: 0.1840,   Val Acc: 9
5.67%
Epoch 11/20 | Train Loss: 0.1229, Train Acc: 96.46% | Val Loss: 0.1889,   Val Acc: 9
5.38%
Epoch 12/20 | Train Loss: 0.1207, Train Acc: 96.52% | Val Loss: 0.1894,   Val Acc: 9
5.30%
Epoch 13/20 | Train Loss: 0.1117, Train Acc: 96.82% | Val Loss: 0.2234,   Val Acc: 9
4.83%
Epoch 14/20 | Train Loss: 0.1051, Train Acc: 96.94% | Val Loss: 0.2625,   Val Acc: 9
3.95%
Epoch 15/20 | Train Loss: 0.1093, Train Acc: 96.88% | Val Loss: 0.1720,   Val Acc: 9
5.90%
Epoch 16/20 | Train Loss: 0.1060, Train Acc: 97.08% | Val Loss: 0.1815,   Val Acc: 9
5.92%
Epoch 17/20 | Train Loss: 0.0939, Train Acc: 97.33% | Val Loss: 0.1745,   Val Acc: 9
5.89%
Epoch 18/20 | Train Loss: 0.0956, Train Acc: 97.30% | Val Loss: 0.2043,   Val Acc: 9
5.58%
Epoch 19/20 | Train Loss: 0.0895, Train Acc: 97.43% | Val Loss: 0.1782,   Val Acc: 9
6.06%
Epoch 20/20 | Train Loss: 0.0877, Train Acc: 97.50% | Val Loss: 0.1909,   Val Acc: 9
```

```
5.75%

--- Learning Rate Analysis Complete! ---
```

Plot Results

In [4]:
```python
# C1.1: Plot Learning Rate Results
plt.figure(figsize=(14, 6))

# Plot Validation Loss
plt.subplot(1, 2, 1)
for lr, history in lr_histories.items():
    if not any(torch.isnan(torch.tensor(history['val_loss']))):
        plt.plot(history['val_loss'], label=f"LR = {lr}")
plt.title("Validation Loss vs. Learning Rate")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.ylim(0, 1.0) # Fix Y-axis for better comparison

# Plot Validation Accuracy
plt.subplot(1, 2, 2)
for lr, history in lr_histories.items():
    if not any(torch.isnan(torch.tensor(history['val_loss']))):
        plt.plot(history['val_acc'], label=f"LR = {lr}")
plt.title("Validation Accuracy vs. Learning Rate")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.ylim(0.8, 1.0) # Fix Y-axis (80% to 100%)

plt.tight_layout()
plt.show()
```



Analysis of Learning Rate

- **LR = 1.0 (Red):** This rate is unstable. You can see in both the loss and accuracy plots that the line is extremely "bouncy" and erratic. This happens because the learning rate is too high, causing the model to "overshoot" the optimal solution on each step.
- **LR = 0.001 (Blue):** This rate is too slow. The loss decreases and the accuracy increases, but both are happening at a crawl. After 20 epochs, this model is nowhere near convergence and has the worst performance.
- **LR = 0.01 (Orange):** This is the baseline rate and it performs well. It shows a smooth, stable decrease in loss and a steady climb in accuracy, converging to a good result (around 96%).
- **LR = 0.1 (Green):** This is the clear winner. It converges the fastest (reaching its lowest loss and highest accuracy by epoch ~10) and achieves the highest final validation accuracy (around 97.8%). It is both fast and stable for this problem.
- **Conclusion:** A learning rate of 0.1 provides the best combination of convergence speed and final performance for this model. The baseline of 0.01 is also a good, safe choice.

C1.2: Batch Size Analysis

- Testing 4 values: `[16, 32, 64, 128]`
- Using best learning rate: `0.01`
- Using baseline architecture: `[128, 64]`
- Training for `20` epochs

In [5]:
```
# C1.2: Batch Size Experiment
BEST_LR = 0.01
EPOCHS = 20
batch_sizes = [16, 32, 64, 128]
bs_histories = {}
bs_results = [] # For the table

print("Starting Batch Size Analysis...")

for bs in batch_sizes:
    print(f"\n--- Testing Batch Size: {bs} ---")

    # CRITICAL: Reload the data with the new batch size
    train_loader_bs, val_loader_bs, _, _, _, _, _, _, _ = load_transform_split_mnis
        classes=None,
        batch_size=bs
    )

    # Re-initialize the model
    model_bs = NeuralNetworkScratch(INPUT_DIM, HIDDEN1, HIDDEN2, OUTPUT_DIM)

    start_time = time.time()
    model_bs.fit(train_loader_bs, val_loader_bs, epochs=EPOCHS, lr=BEST_LR)
```

```
    end_time = time.time()

    bs_histories[bs] = model_bs.history
    bs_results.append({
        "Batch Size": bs,
        "Final Val Accuracy": model_bs.history['val_acc'][-1],
        "Training Time (s)": end_time - start_time
    })

print("\n--- Batch Size Analysis Complete! ---")
```

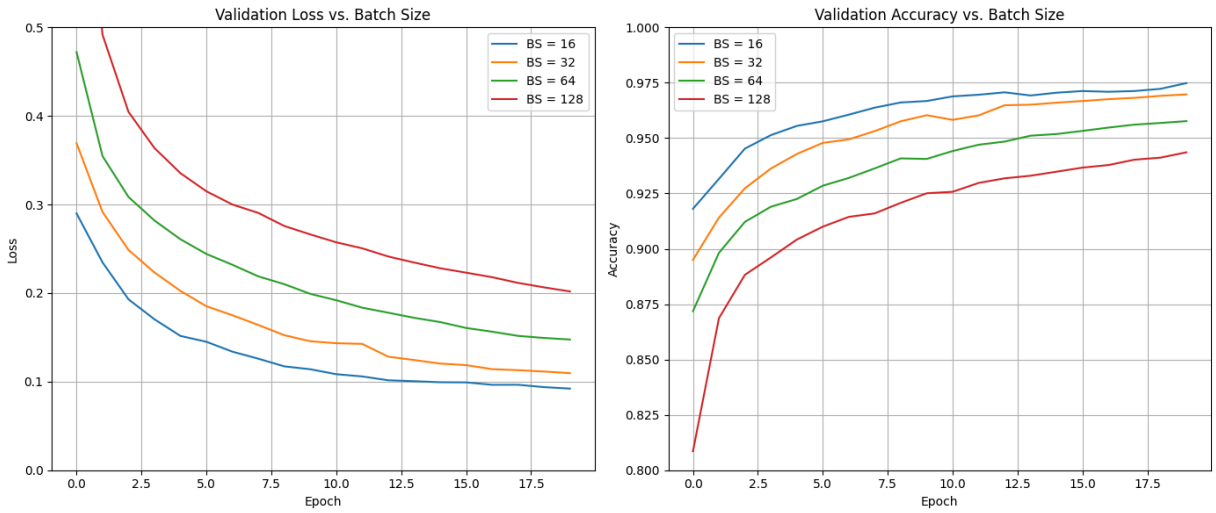
```
Starting Batch Size Analysis...

--- Testing Batch Size: 16 ---
Epoch  1/20 | Train Loss: 0.5219, Train Acc: 85.82% | Val Loss: 0.2900,   Val Acc: 9
1.81%
Epoch  2/20 | Train Loss: 0.2537, Train Acc: 92.74% | Val Loss: 0.2347,   Val Acc: 9
3.16%
Epoch  3/20 | Train Loss: 0.2006, Train Acc: 94.17% | Val Loss: 0.1930,   Val Acc: 9
4.53%
Epoch  4/20 | Train Loss: 0.1669, Train Acc: 95.19% | Val Loss: 0.1704,   Val Acc: 9
5.13%
Epoch  5/20 | Train Loss: 0.1426, Train Acc: 95.88% | Val Loss: 0.1516,   Val Acc: 9
5.55%
Epoch  6/20 | Train Loss: 0.1244, Train Acc: 96.36% | Val Loss: 0.1450,   Val Acc: 9
5.76%
Epoch  7/20 | Train Loss: 0.1084, Train Acc: 96.85% | Val Loss: 0.1339,   Val Acc: 9
6.06%
Epoch  8/20 | Train Loss: 0.0969, Train Acc: 97.11% | Val Loss: 0.1259,   Val Acc: 9
6.38%
Epoch  9/20 | Train Loss: 0.0866, Train Acc: 97.49% | Val Loss: 0.1173,   Val Acc: 9
6.61%
Epoch 10/20 | Train Loss: 0.0774, Train Acc: 97.79% | Val Loss: 0.1140,   Val Acc: 9
6.67%
Epoch 11/20 | Train Loss: 0.0702, Train Acc: 97.98% | Val Loss: 0.1084,   Val Acc: 9
6.88%
Epoch 12/20 | Train Loss: 0.0637, Train Acc: 98.21% | Val Loss: 0.1059,   Val Acc: 9
6.96%
Epoch 13/20 | Train Loss: 0.0582, Train Acc: 98.35% | Val Loss: 0.1016,   Val Acc: 9
7.07%
Epoch 14/20 | Train Loss: 0.0526, Train Acc: 98.57% | Val Loss: 0.1005,   Val Acc: 9
6.92%
Epoch 15/20 | Train Loss: 0.0482, Train Acc: 98.66% | Val Loss: 0.0994,   Val Acc: 9
7.05%
Epoch 16/20 | Train Loss: 0.0440, Train Acc: 98.78% | Val Loss: 0.0992,   Val Acc: 9
7.12%
Epoch 17/20 | Train Loss: 0.0405, Train Acc: 98.94% | Val Loss: 0.0965,   Val Acc: 9
7.09%
Epoch 18/20 | Train Loss: 0.0369, Train Acc: 99.01% | Val Loss: 0.0965,   Val Acc: 9
7.12%
Epoch 19/20 | Train Loss: 0.0340, Train Acc: 99.15% | Val Loss: 0.0938,   Val Acc: 9
7.22%
Epoch 20/20 | Train Loss: 0.0312, Train Acc: 99.21% | Val Loss: 0.0922,   Val Acc: 9
7.48%

--- Testing Batch Size: 32 ---
Epoch  1/20 | Train Loss: 0.7353, Train Acc: 80.39% | Val Loss: 0.3693,   Val Acc: 8
9.50%
Epoch  2/20 | Train Loss: 0.3206, Train Acc: 90.84% | Val Loss: 0.2916,   Val Acc: 9
1.41%
Epoch  3/20 | Train Loss: 0.2640, Train Acc: 92.46% | Val Loss: 0.2487,   Val Acc: 9
2.73%
Epoch  4/20 | Train Loss: 0.2286, Train Acc: 93.49% | Val Loss: 0.2233,   Val Acc: 9
3.62%
Epoch  5/20 | Train Loss: 0.2029, Train Acc: 94.19% | Val Loss: 0.2025,   Val Acc: 9
4.28%
Epoch  6/20 | Train Loss: 0.1816, Train Acc: 94.79% | Val Loss: 0.1851,   Val Acc: 9
```

```
4.78%
Epoch  7/20 | Train Loss: 0.1643, Train Acc: 95.28% | Val Loss: 0.1750,   Val Acc: 9
4.94%
Epoch  8/20 | Train Loss: 0.1499, Train Acc: 95.71% | Val Loss: 0.1640,   Val Acc: 9
5.32%
Epoch  9/20 | Train Loss: 0.1377, Train Acc: 96.03% | Val Loss: 0.1524,   Val Acc: 9
5.76%
Epoch 10/20 | Train Loss: 0.1269, Train Acc: 96.37% | Val Loss: 0.1456,   Val Acc: 9
6.03%
Epoch 11/20 | Train Loss: 0.1176, Train Acc: 96.64% | Val Loss: 0.1434,   Val Acc: 9
5.83%
Epoch 12/20 | Train Loss: 0.1097, Train Acc: 96.83% | Val Loss: 0.1427,   Val Acc: 9
6.03%
Epoch 13/20 | Train Loss: 0.1021, Train Acc: 97.07% | Val Loss: 0.1282,   Val Acc: 9
6.48%
Epoch 14/20 | Train Loss: 0.0954, Train Acc: 97.33% | Val Loss: 0.1244,   Val Acc: 9
6.51%
Epoch 15/20 | Train Loss: 0.0896, Train Acc: 97.49% | Val Loss: 0.1204,   Val Acc: 9
6.60%
Epoch 16/20 | Train Loss: 0.0841, Train Acc: 97.61% | Val Loss: 0.1187,   Val Acc: 9
6.67%
Epoch 17/20 | Train Loss: 0.0794, Train Acc: 97.75% | Val Loss: 0.1141,   Val Acc: 9
6.76%
Epoch 18/20 | Train Loss: 0.0754, Train Acc: 97.92% | Val Loss: 0.1130,   Val Acc: 9
6.82%
Epoch 19/20 | Train Loss: 0.0707, Train Acc: 98.02% | Val Loss: 0.1114,   Val Acc: 9
6.91%
Epoch 20/20 | Train Loss: 0.0673, Train Acc: 98.11% | Val Loss: 0.1096,   Val Acc: 9
6.97%

--- Testing Batch Size: 64 ---
Epoch  1/20 | Train Loss: 0.9655, Train Acc: 75.36% | Val Loss: 0.4721,   Val Acc: 8
7.18%
Epoch  2/20 | Train Loss: 0.3983, Train Acc: 88.94% | Val Loss: 0.3547,   Val Acc: 8
9.82%
Epoch  3/20 | Train Loss: 0.3267, Train Acc: 90.77% | Val Loss: 0.3085,   Val Acc: 9
1.22%
Epoch  4/20 | Train Loss: 0.2901, Train Acc: 91.72% | Val Loss: 0.2820,   Val Acc: 9
1.90%
Epoch  5/20 | Train Loss: 0.2647, Train Acc: 92.47% | Val Loss: 0.2608,   Val Acc: 9
2.25%
Epoch  6/20 | Train Loss: 0.2452, Train Acc: 93.10% | Val Loss: 0.2442,   Val Acc: 9
2.85%
Epoch  7/20 | Train Loss: 0.2286, Train Acc: 93.49% | Val Loss: 0.2320,   Val Acc: 9
3.20%
Epoch  8/20 | Train Loss: 0.2145, Train Acc: 93.92% | Val Loss: 0.2189,   Val Acc: 9
3.63%
Epoch  9/20 | Train Loss: 0.2024, Train Acc: 94.28% | Val Loss: 0.2101,   Val Acc: 9
4.08%
Epoch 10/20 | Train Loss: 0.1912, Train Acc: 94.58% | Val Loss: 0.1991,   Val Acc: 9
4.06%
Epoch 11/20 | Train Loss: 0.1817, Train Acc: 94.81% | Val Loss: 0.1919,   Val Acc: 9
4.42%
Epoch 12/20 | Train Loss: 0.1728, Train Acc: 95.11% | Val Loss: 0.1835,   Val Acc: 9
4.70%
Epoch 13/20 | Train Loss: 0.1645, Train Acc: 95.28% | Val Loss: 0.1779,   Val Acc: 9
```

```
4.84%
Epoch 14/20 | Train Loss: 0.1574, Train Acc: 95.53% | Val Loss: 0.1721,   Val Acc: 9
5.11%
Epoch 15/20 | Train Loss: 0.1503, Train Acc: 95.76% | Val Loss: 0.1673,   Val Acc: 9
5.18%
Epoch 16/20 | Train Loss: 0.1442, Train Acc: 95.90% | Val Loss: 0.1606,   Val Acc: 9
5.33%
Epoch 17/20 | Train Loss: 0.1383, Train Acc: 96.09% | Val Loss: 0.1564,   Val Acc: 9
5.47%
Epoch 18/20 | Train Loss: 0.1328, Train Acc: 96.23% | Val Loss: 0.1517,   Val Acc: 9
5.61%
Epoch 19/20 | Train Loss: 0.1280, Train Acc: 96.38% | Val Loss: 0.1494,   Val Acc: 9
5.68%
Epoch 20/20 | Train Loss: 0.1233, Train Acc: 96.48% | Val Loss: 0.1476,   Val Acc: 9
5.77%

--- Testing Batch Size: 128 ---
Epoch  1/20 | Train Loss: 1.4866, Train Acc: 61.16% | Val Loss: 0.8027,   Val Acc: 8
0.86%
Epoch  2/20 | Train Loss: 0.6052, Train Acc: 84.58% | Val Loss: 0.4918,   Val Acc: 8
6.87%
Epoch  3/20 | Train Loss: 0.4401, Train Acc: 88.06% | Val Loss: 0.4047,   Val Acc: 8
8.83%
Epoch  4/20 | Train Loss: 0.3784, Train Acc: 89.48% | Val Loss: 0.3638,   Val Acc: 8
9.61%
Epoch  5/20 | Train Loss: 0.3439, Train Acc: 90.39% | Val Loss: 0.3354,   Val Acc: 9
0.42%
Epoch  6/20 | Train Loss: 0.3200, Train Acc: 90.97% | Val Loss: 0.3149,   Val Acc: 9
1.00%
Epoch  7/20 | Train Loss: 0.3018, Train Acc: 91.51% | Val Loss: 0.3000,   Val Acc: 9
1.44%
Epoch  8/20 | Train Loss: 0.2871, Train Acc: 91.84% | Val Loss: 0.2905,   Val Acc: 9
1.60%
Epoch  9/20 | Train Loss: 0.2745, Train Acc: 92.16% | Val Loss: 0.2758,   Val Acc: 9
2.07%
Epoch 10/20 | Train Loss: 0.2636, Train Acc: 92.46% | Val Loss: 0.2663,   Val Acc: 9
2.51%
Epoch 11/20 | Train Loss: 0.2534, Train Acc: 92.79% | Val Loss: 0.2573,   Val Acc: 9
2.58%
Epoch 12/20 | Train Loss: 0.2443, Train Acc: 93.05% | Val Loss: 0.2506,   Val Acc: 9
2.97%
Epoch 13/20 | Train Loss: 0.2359, Train Acc: 93.31% | Val Loss: 0.2413,   Val Acc: 9
3.18%
Epoch 14/20 | Train Loss: 0.2278, Train Acc: 93.51% | Val Loss: 0.2345,   Val Acc: 9
3.30%
Epoch 15/20 | Train Loss: 0.2204, Train Acc: 93.75% | Val Loss: 0.2281,   Val Acc: 9
3.48%
Epoch 16/20 | Train Loss: 0.2136, Train Acc: 93.97% | Val Loss: 0.2231,   Val Acc: 9
3.67%
Epoch 17/20 | Train Loss: 0.2069, Train Acc: 94.17% | Val Loss: 0.2179,   Val Acc: 9
3.78%
Epoch 18/20 | Train Loss: 0.2006, Train Acc: 94.29% | Val Loss: 0.2115,   Val Acc: 9
4.03%
Epoch 19/20 | Train Loss: 0.1946, Train Acc: 94.46% | Val Loss: 0.2065,   Val Acc: 9
4.12%
Epoch 20/20 | Train Loss: 0.1888, Train Acc: 94.63% | Val Loss: 0.2019,   Val Acc: 9
```

```
4.36%

--- Batch Size Analysis Complete! ---
```

In [6]:
```python
# C1.2: Plot Batch Size Results
plt.figure(figsize=(14, 6))

# Plot Validation Loss
plt.subplot(1, 2, 1)
for bs, history in bs_histories.items():
    plt.plot(history['val_loss'], label=f"BS = {bs}")
plt.title("Validation Loss vs. Batch Size")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.ylim(0, 0.5)

# Plot Validation Accuracy
plt.subplot(1, 2, 2)
for bs, history in bs_histories.items():
    plt.plot(history['val_acc'], label=f"BS = {bs}")
plt.title("Validation Accuracy vs. Batch Size")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.ylim(0.8, 1.0)

plt.tight_layout()
plt.show()

# C1.2: Print Comparison Table
print("\n--- Batch Size Performance Table ---")
df_bs_results = pd.DataFrame(bs_results)
print(df_bs_results.to_markdown(index=False, floatfmt=".4f"))
```



```
--- Batch Size Performance Table ---
|   Batch Size |   Final Val Accuracy |   Training Time (s) |
|-------------:|---------------------:|--------------------:|
|      16.0000 |               0.9748 |            254.7564 |
|      32.0000 |               0.9698 |            213.8092 |
|      64.0000 |               0.9577 |            199.9405 |
|     128.0000 |               0.9436 |            186.2352 |
```

Analysis of Batch Size

The results show a clear trade-off between training speed and model performance.

- **Training Efficiency:** As shown in the table, training time is inversely proportional to batch size. The largest batch size (BS=128) was the fastest to train (186.2s), while the smallest (BS=16) was the slowest (254.7s). This is because larger batches require fewer weight update steps per epoch.
- **Gradient Noise & Performance:** The plots show that smaller batch sizes lead to better final accuracy.

BS=16 (blue line) clearly achieves the highest validation accuracy (~97.5%) and the lowest validation loss.

BS=128 (red line) converges to the lowest accuracy (~94.4%) and highest loss.

This is a classic effect of gradient noise. Smaller batches (like 16) provide a "noisier" estimate of the gradient at each step. This noise can act as a form of regularization, preventing the model from getting stuck in a poor local minimum and helping it find a better, more generalizable solution. In this case, the noise from BS=16 was clearly beneficial.

- **Final Performance:** The table confirms the findings from the plots. BS=16 achieved the best final accuracy (97.48%), but it came at the cost of being the slowest to train. BS=32 offers a good compromise, being faster than BS=16 while still achieving a high accuracy (96.98%).

C1.3: Architecture Analysis

- Testing 4 different depths (2, 3, 4, 5 hidden layers)
- Testing 4 different widths ([64, 32], [128, 64], [256, 128], [512, 256])
- Using best LR (`0.01`) and BS (`64`)
- Training for `20` epochs

In [7]:
```
# C1.3: Architecture Experiment
EPOCHS = 20
BEST_LR = 0.01
BEST_BS = 64
arch_results = [] # For the table
```

```
# --- Experiment 1: Number of Layers ---
print("\n--- Testing Number of Layers ---")
layer_configs = {
    "2_layers (baseline)": [INPUT_DIM, 128, 64, OUTPUT_DIM],
    "3_layers": [INPUT_DIM, 128, 128, 64, OUTPUT_DIM],
    "4_layers": [INPUT_DIM, 128, 128, 64, 32, OUTPUT_DIM],
    "5_layers": [INPUT_DIM, 128, 64, 64, 32, 32, OUTPUT_DIM]
}

for name, config in layer_configs.items():
    print(f"\nTesting Architecture: {name}")
    model = FlexibleNN(config) # Use our flexible class

    start_time = time.time()
    model.fit(train_loader, val_loader, epochs=EPOCHS, lr=BEST_LR)
    end_time = time.time()

    arch_results.append({
        "Experiment": "Number of Layers",
        "Architecture": name,
        "Final Val Accuracy": model.history['val_acc'][-1],
        "Training Time (s)": end_time - start_time
    })

# --- Experiment 2: Neurons Per Layer ---
print("\n--- Testing Neurons Per Layer (2 Hidden Layers) ---")
neuron_configs = {
    "Small [64, 32]": (64, 32),
    "Baseline [128, 64]": (128, 64),
    "Medium [256, 128]": (256, 128),
    "Large [512, 256]": (512, 256)
}

for name, (h1, h2) in neuron_configs.items():
    print(f"\nTesting Architecture: {name}")
    model = NeuralNetworkScratch(INPUT_DIM, h1, h2, OUTPUT_DIM) # Use original clas

    start_time = time.time()
    model.fit(train_loader, val_loader, epochs=EPOCHS, lr=BEST_LR)
    end_time = time.time()

    arch_results.append({
        "Experiment": "Neurons Per Layer",
        "Architecture": name,
        "Final Val Accuracy": model.history['val_acc'][-1],
        "Training Time (s)": end_time - start_time
    })

print("\n--- Architecture Analysis Complete! ---")
```

```
--- Testing Number of Layers ---

Testing Architecture: 2_layers (baseline)
Epoch  1/20 | Train Loss: 0.8711, Train Acc: 77.35% | Val Loss: 0.4326,   Val Acc: 8
8.17%
Epoch  2/20 | Train Loss: 0.3764, Train Acc: 89.49% | Val Loss: 0.3412,   Val Acc: 9
0.43%
Epoch  3/20 | Train Loss: 0.3140, Train Acc: 90.95% | Val Loss: 0.2989,   Val Acc: 9
1.47%
Epoch  4/20 | Train Loss: 0.2801, Train Acc: 92.02% | Val Loss: 0.2743,   Val Acc: 9
2.16%
Epoch  5/20 | Train Loss: 0.2559, Train Acc: 92.70% | Val Loss: 0.2550,   Val Acc: 9
2.70%
Epoch  6/20 | Train Loss: 0.2364, Train Acc: 93.29% | Val Loss: 0.2375,   Val Acc: 9
3.22%
Epoch  7/20 | Train Loss: 0.2205, Train Acc: 93.70% | Val Loss: 0.2239,   Val Acc: 9
3.83%
Epoch  8/20 | Train Loss: 0.2067, Train Acc: 94.05% | Val Loss: 0.2115,   Val Acc: 9
4.08%
Epoch  9/20 | Train Loss: 0.1945, Train Acc: 94.44% | Val Loss: 0.2038,   Val Acc: 9
4.32%
Epoch 10/20 | Train Loss: 0.1833, Train Acc: 94.77% | Val Loss: 0.1971,   Val Acc: 9
4.55%
Epoch 11/20 | Train Loss: 0.1734, Train Acc: 95.08% | Val Loss: 0.1881,   Val Acc: 9
4.83%
Epoch 12/20 | Train Loss: 0.1647, Train Acc: 95.35% | Val Loss: 0.1781,   Val Acc: 9
5.17%
Epoch 13/20 | Train Loss: 0.1564, Train Acc: 95.53% | Val Loss: 0.1780,   Val Acc: 9
4.99%
Epoch 14/20 | Train Loss: 0.1491, Train Acc: 95.83% | Val Loss: 0.1655,   Val Acc: 9
5.52%
Epoch 15/20 | Train Loss: 0.1424, Train Acc: 95.95% | Val Loss: 0.1633,   Val Acc: 9
5.43%
Epoch 16/20 | Train Loss: 0.1363, Train Acc: 96.12% | Val Loss: 0.1573,   Val Acc: 9
5.68%
Epoch 17/20 | Train Loss: 0.1302, Train Acc: 96.25% | Val Loss: 0.1535,   Val Acc: 9
5.75%
Epoch 18/20 | Train Loss: 0.1248, Train Acc: 96.49% | Val Loss: 0.1491,   Val Acc: 9
5.83%
Epoch 19/20 | Train Loss: 0.1199, Train Acc: 96.61% | Val Loss: 0.1449,   Val Acc: 9
5.94%
Epoch 20/20 | Train Loss: 0.1155, Train Acc: 96.74% | Val Loss: 0.1401,   Val Acc: 9
6.10%

Testing Architecture: 3_layers
Epoch  1/20 | Train Loss: 0.8179, Train Acc: 77.32% | Val Loss: 0.3860,   Val Acc: 8
8.91%
Epoch  2/20 | Train Loss: 0.3299, Train Acc: 90.64% | Val Loss: 0.2944,   Val Acc: 9
1.25%
Epoch  3/20 | Train Loss: 0.2665, Train Acc: 92.31% | Val Loss: 0.2530,   Val Acc: 9
2.61%
Epoch  4/20 | Train Loss: 0.2320, Train Acc: 93.28% | Val Loss: 0.2254,   Val Acc: 9
3.46%
Epoch  5/20 | Train Loss: 0.2074, Train Acc: 94.00% | Val Loss: 0.2092,   Val Acc: 9
4.03%
Epoch  6/20 | Train Loss: 0.1872, Train Acc: 94.56% | Val Loss: 0.1962,   Val Acc: 9
```

```
4.33%
Epoch  7/20 | Train Loss: 0.1709, Train Acc: 94.99% | Val Loss: 0.1790,   Val Acc: 9
4.66%
Epoch  8/20 | Train Loss: 0.1572, Train Acc: 95.40% | Val Loss: 0.1715,   Val Acc: 9
4.96%
Epoch  9/20 | Train Loss: 0.1456, Train Acc: 95.73% | Val Loss: 0.1659,   Val Acc: 9
5.18%
Epoch 10/20 | Train Loss: 0.1357, Train Acc: 96.02% | Val Loss: 0.1580,   Val Acc: 9
5.24%
Epoch 11/20 | Train Loss: 0.1266, Train Acc: 96.28% | Val Loss: 0.1470,   Val Acc: 9
5.67%
Epoch 12/20 | Train Loss: 0.1185, Train Acc: 96.57% | Val Loss: 0.1464,   Val Acc: 9
5.77%
Epoch 13/20 | Train Loss: 0.1114, Train Acc: 96.78% | Val Loss: 0.1374,   Val Acc: 9
5.89%
Epoch 14/20 | Train Loss: 0.1053, Train Acc: 96.93% | Val Loss: 0.1387,   Val Acc: 9
5.83%
Epoch 15/20 | Train Loss: 0.0992, Train Acc: 97.18% | Val Loss: 0.1359,   Val Acc: 9
6.09%
Epoch 16/20 | Train Loss: 0.0932, Train Acc: 97.31% | Val Loss: 0.1260,   Val Acc: 9
6.28%
Epoch 17/20 | Train Loss: 0.0882, Train Acc: 97.51% | Val Loss: 0.1263,   Val Acc: 9
6.22%
Epoch 18/20 | Train Loss: 0.0836, Train Acc: 97.61% | Val Loss: 0.1205,   Val Acc: 9
6.48%
Epoch 19/20 | Train Loss: 0.0798, Train Acc: 97.70% | Val Loss: 0.1176,   Val Acc: 9
6.56%
Epoch 20/20 | Train Loss: 0.0756, Train Acc: 97.85% | Val Loss: 0.1166,   Val Acc: 9
6.62%

Testing Architecture: 4_layers
Epoch  1/20 | Train Loss: 0.9169, Train Acc: 72.60% | Val Loss: 0.3899,   Val Acc: 8
8.79%
Epoch  2/20 | Train Loss: 0.3235, Train Acc: 90.77% | Val Loss: 0.2833,   Val Acc: 9
1.82%
Epoch  3/20 | Train Loss: 0.2483, Train Acc: 92.83% | Val Loss: 0.2281,   Val Acc: 9
3.42%
Epoch  4/20 | Train Loss: 0.2090, Train Acc: 94.01% | Val Loss: 0.2040,   Val Acc: 9
3.97%
Epoch  5/20 | Train Loss: 0.1818, Train Acc: 94.76% | Val Loss: 0.1817,   Val Acc: 9
4.77%
Epoch  6/20 | Train Loss: 0.1624, Train Acc: 95.27% | Val Loss: 0.1682,   Val Acc: 9
5.17%
Epoch  7/20 | Train Loss: 0.1468, Train Acc: 95.77% | Val Loss: 0.1592,   Val Acc: 9
5.41%
Epoch  8/20 | Train Loss: 0.1337, Train Acc: 96.11% | Val Loss: 0.1558,   Val Acc: 9
5.35%
Epoch  9/20 | Train Loss: 0.1220, Train Acc: 96.52% | Val Loss: 0.1401,   Val Acc: 9
5.93%
Epoch 10/20 | Train Loss: 0.1129, Train Acc: 96.76% | Val Loss: 0.1362,   Val Acc: 9
6.08%
Epoch 11/20 | Train Loss: 0.1040, Train Acc: 96.96% | Val Loss: 0.1327,   Val Acc: 9
6.24%
Epoch 12/20 | Train Loss: 0.0971, Train Acc: 97.16% | Val Loss: 0.1291,   Val Acc: 9
6.33%
Epoch 13/20 | Train Loss: 0.0890, Train Acc: 97.42% | Val Loss: 0.1211,   Val Acc: 9
```

```
6.53%
Epoch 14/20 | Train Loss: 0.0838, Train Acc: 97.58% | Val Loss: 0.1163,   Val Acc: 9
6.65%
Epoch 15/20 | Train Loss: 0.0776, Train Acc: 97.76% | Val Loss: 0.1128,   Val Acc: 9
6.81%
Epoch 16/20 | Train Loss: 0.0726, Train Acc: 97.95% | Val Loss: 0.1113,   Val Acc: 9
6.79%
Epoch 17/20 | Train Loss: 0.0684, Train Acc: 98.05% | Val Loss: 0.1103,   Val Acc: 9
6.78%
Epoch 18/20 | Train Loss: 0.0637, Train Acc: 98.14% | Val Loss: 0.1156,   Val Acc: 9
6.57%
Epoch 19/20 | Train Loss: 0.0594, Train Acc: 98.39% | Val Loss: 0.1079,   Val Acc: 9
6.85%
Epoch 20/20 | Train Loss: 0.0558, Train Acc: 98.42% | Val Loss: 0.1041,   Val Acc: 9
7.03%

Testing Architecture: 5_layers
Epoch  1/20 | Train Loss: 0.8736, Train Acc: 73.05% | Val Loss: 0.3498,   Val Acc: 8
9.87%
Epoch  2/20 | Train Loss: 0.3048, Train Acc: 91.15% | Val Loss: 0.2624,   Val Acc: 9
2.32%
Epoch  3/20 | Train Loss: 0.2323, Train Acc: 93.19% | Val Loss: 0.2166,   Val Acc: 9
3.73%
Epoch  4/20 | Train Loss: 0.1933, Train Acc: 94.31% | Val Loss: 0.1957,   Val Acc: 9
4.34%
Epoch  5/20 | Train Loss: 0.1664, Train Acc: 95.09% | Val Loss: 0.1812,   Val Acc: 9
4.62%
Epoch  6/20 | Train Loss: 0.1449, Train Acc: 95.77% | Val Loss: 0.1604,   Val Acc: 9
5.24%
Epoch  7/20 | Train Loss: 0.1299, Train Acc: 96.16% | Val Loss: 0.1476,   Val Acc: 9
5.69%
Epoch  8/20 | Train Loss: 0.1167, Train Acc: 96.54% | Val Loss: 0.1545,   Val Acc: 9
5.45%
Epoch  9/20 | Train Loss: 0.1055, Train Acc: 96.87% | Val Loss: 0.1387,   Val Acc: 9
6.09%
Epoch 10/20 | Train Loss: 0.0963, Train Acc: 97.14% | Val Loss: 0.1295,   Val Acc: 9
6.17%
Epoch 11/20 | Train Loss: 0.0881, Train Acc: 97.45% | Val Loss: 0.1296,   Val Acc: 9
6.20%
Epoch 12/20 | Train Loss: 0.0804, Train Acc: 97.61% | Val Loss: 0.1236,   Val Acc: 9
6.42%
Epoch 13/20 | Train Loss: 0.0742, Train Acc: 97.79% | Val Loss: 0.1149,   Val Acc: 9
6.67%
Epoch 14/20 | Train Loss: 0.0681, Train Acc: 98.00% | Val Loss: 0.1252,   Val Acc: 9
6.46%
Epoch 15/20 | Train Loss: 0.0629, Train Acc: 98.11% | Val Loss: 0.1207,   Val Acc: 9
6.53%
Epoch 16/20 | Train Loss: 0.0579, Train Acc: 98.28% | Val Loss: 0.1130,   Val Acc: 9
6.85%
Epoch 17/20 | Train Loss: 0.0541, Train Acc: 98.40% | Val Loss: 0.1112,   Val Acc: 9
6.83%
Epoch 18/20 | Train Loss: 0.0497, Train Acc: 98.53% | Val Loss: 0.1091,   Val Acc: 9
6.97%
Epoch 19/20 | Train Loss: 0.0461, Train Acc: 98.62% | Val Loss: 0.1096,   Val Acc: 9
6.97%
Epoch 20/20 | Train Loss: 0.0428, Train Acc: 98.77% | Val Loss: 0.1203,   Val Acc: 9
```

```
6.56%

--- Testing Neurons Per Layer (2 Hidden Layers) ---

Testing Architecture: Small [64, 32]
Epoch  1/20 | Train Loss: 1.1111, Train Acc: 70.19% | Val Loss: 0.5338,   Val Acc: 8
6.33%
Epoch  2/20 | Train Loss: 0.4435, Train Acc: 88.09% | Val Loss: 0.3890,   Val Acc: 8
9.33%
Epoch  3/20 | Train Loss: 0.3613, Train Acc: 89.96% | Val Loss: 0.3450,   Val Acc: 9
0.17%
Epoch  4/20 | Train Loss: 0.3253, Train Acc: 90.83% | Val Loss: 0.3187,   Val Acc: 9
0.98%
Epoch  5/20 | Train Loss: 0.3017, Train Acc: 91.47% | Val Loss: 0.2983,   Val Acc: 9
1.46%
Epoch  6/20 | Train Loss: 0.2834, Train Acc: 91.96% | Val Loss: 0.2836,   Val Acc: 9
1.85%
Epoch  7/20 | Train Loss: 0.2675, Train Acc: 92.40% | Val Loss: 0.2706,   Val Acc: 9
2.34%
Epoch  8/20 | Train Loss: 0.2538, Train Acc: 92.80% | Val Loss: 0.2576,   Val Acc: 9
2.62%
Epoch  9/20 | Train Loss: 0.2419, Train Acc: 93.11% | Val Loss: 0.2464,   Val Acc: 9
2.87%
Epoch 10/20 | Train Loss: 0.2305, Train Acc: 93.50% | Val Loss: 0.2369,   Val Acc: 9
3.28%
Epoch 11/20 | Train Loss: 0.2198, Train Acc: 93.79% | Val Loss: 0.2290,   Val Acc: 9
3.57%
Epoch 12/20 | Train Loss: 0.2099, Train Acc: 94.03% | Val Loss: 0.2204,   Val Acc: 9
3.89%
Epoch 13/20 | Train Loss: 0.2010, Train Acc: 94.35% | Val Loss: 0.2136,   Val Acc: 9
4.00%
Epoch 14/20 | Train Loss: 0.1926, Train Acc: 94.60% | Val Loss: 0.2041,   Val Acc: 9
4.27%
Epoch 15/20 | Train Loss: 0.1851, Train Acc: 94.83% | Val Loss: 0.1978,   Val Acc: 9
4.41%
Epoch 16/20 | Train Loss: 0.1781, Train Acc: 95.00% | Val Loss: 0.1923,   Val Acc: 9
4.56%
Epoch 17/20 | Train Loss: 0.1711, Train Acc: 95.18% | Val Loss: 0.1870,   Val Acc: 9
4.75%
Epoch 18/20 | Train Loss: 0.1651, Train Acc: 95.38% | Val Loss: 0.1812,   Val Acc: 9
4.80%
Epoch 19/20 | Train Loss: 0.1590, Train Acc: 95.56% | Val Loss: 0.1777,   Val Acc: 9
4.98%
Epoch 20/20 | Train Loss: 0.1534, Train Acc: 95.69% | Val Loss: 0.1762,   Val Acc: 9
4.97%

Testing Architecture: Baseline [128, 64]
Epoch  1/20 | Train Loss: 1.0177, Train Acc: 73.58% | Val Loss: 0.4981,   Val Acc: 8
6.44%
Epoch  2/20 | Train Loss: 0.4156, Train Acc: 88.53% | Val Loss: 0.3675,   Val Acc: 8
9.50%
Epoch  3/20 | Train Loss: 0.3383, Train Acc: 90.33% | Val Loss: 0.3197,   Val Acc: 9
0.74%
Epoch  4/20 | Train Loss: 0.2998, Train Acc: 91.32% | Val Loss: 0.2924,   Val Acc: 9
1.62%
Epoch  5/20 | Train Loss: 0.2733, Train Acc: 92.13% | Val Loss: 0.2726,   Val Acc: 9
```

```
2.12%
Epoch  6/20 | Train Loss: 0.2530, Train Acc: 92.66% | Val Loss: 0.2541,   Val Acc: 9
2.75%
Epoch  7/20 | Train Loss: 0.2354, Train Acc: 93.21% | Val Loss: 0.2424,   Val Acc: 9
2.96%
Epoch  8/20 | Train Loss: 0.2203, Train Acc: 93.66% | Val Loss: 0.2316,   Val Acc: 9
3.38%
Epoch  9/20 | Train Loss: 0.2067, Train Acc: 94.15% | Val Loss: 0.2190,   Val Acc: 9
3.84%
Epoch 10/20 | Train Loss: 0.1949, Train Acc: 94.46% | Val Loss: 0.2104,   Val Acc: 9
3.93%
Epoch 11/20 | Train Loss: 0.1848, Train Acc: 94.78% | Val Loss: 0.2005,   Val Acc: 9
4.44%
Epoch 12/20 | Train Loss: 0.1749, Train Acc: 95.08% | Val Loss: 0.1931,   Val Acc: 9
4.51%
Epoch 13/20 | Train Loss: 0.1665, Train Acc: 95.27% | Val Loss: 0.1833,   Val Acc: 9
4.86%
Epoch 14/20 | Train Loss: 0.1585, Train Acc: 95.45% | Val Loss: 0.1777,   Val Acc: 9
5.06%
Epoch 15/20 | Train Loss: 0.1514, Train Acc: 95.70% | Val Loss: 0.1714,   Val Acc: 9
5.23%
Epoch 16/20 | Train Loss: 0.1447, Train Acc: 95.92% | Val Loss: 0.1678,   Val Acc: 9
5.24%
Epoch 17/20 | Train Loss: 0.1387, Train Acc: 96.03% | Val Loss: 0.1628,   Val Acc: 9
5.51%
Epoch 18/20 | Train Loss: 0.1328, Train Acc: 96.28% | Val Loss: 0.1578,   Val Acc: 9
5.54%
Epoch 19/20 | Train Loss: 0.1278, Train Acc: 96.37% | Val Loss: 0.1559,   Val Acc: 9
5.67%
Epoch 20/20 | Train Loss: 0.1229, Train Acc: 96.49% | Val Loss: 0.1505,   Val Acc: 9
5.96%

Testing Architecture: Medium [256, 128]
Epoch  1/20 | Train Loss: 0.9202, Train Acc: 77.27% | Val Loss: 0.4379,   Val Acc: 8
8.13%
Epoch  2/20 | Train Loss: 0.3790, Train Acc: 89.53% | Val Loss: 0.3390,   Val Acc: 9
0.35%
Epoch  3/20 | Train Loss: 0.3157, Train Acc: 91.05% | Val Loss: 0.2984,   Val Acc: 9
1.46%
Epoch  4/20 | Train Loss: 0.2818, Train Acc: 91.96% | Val Loss: 0.2723,   Val Acc: 9
2.15%
Epoch  5/20 | Train Loss: 0.2571, Train Acc: 92.66% | Val Loss: 0.2524,   Val Acc: 9
2.83%
Epoch  6/20 | Train Loss: 0.2377, Train Acc: 93.14% | Val Loss: 0.2372,   Val Acc: 9
3.17%
Epoch  7/20 | Train Loss: 0.2218, Train Acc: 93.59% | Val Loss: 0.2220,   Val Acc: 9
3.62%
Epoch  8/20 | Train Loss: 0.2076, Train Acc: 94.07% | Val Loss: 0.2119,   Val Acc: 9
3.88%
Epoch  9/20 | Train Loss: 0.1946, Train Acc: 94.41% | Val Loss: 0.2008,   Val Acc: 9
4.26%
Epoch 10/20 | Train Loss: 0.1838, Train Acc: 94.77% | Val Loss: 0.1931,   Val Acc: 9
4.43%
Epoch 11/20 | Train Loss: 0.1736, Train Acc: 94.95% | Val Loss: 0.1818,   Val Acc: 9
4.87%
Epoch 12/20 | Train Loss: 0.1645, Train Acc: 95.23% | Val Loss: 0.1766,   Val Acc: 9
```

```
5.03%
Epoch 13/20 | Train Loss: 0.1562, Train Acc: 95.48% | Val Loss: 0.1692,   Val Acc: 9
5.23%
Epoch 14/20 | Train Loss: 0.1484, Train Acc: 95.69% | Val Loss: 0.1629,   Val Acc: 9
5.30%
Epoch 15/20 | Train Loss: 0.1413, Train Acc: 95.98% | Val Loss: 0.1581,   Val Acc: 9
5.50%
Epoch 16/20 | Train Loss: 0.1347, Train Acc: 96.10% | Val Loss: 0.1533,   Val Acc: 9
5.67%
Epoch 17/20 | Train Loss: 0.1287, Train Acc: 96.26% | Val Loss: 0.1485,   Val Acc: 9
5.78%
Epoch 18/20 | Train Loss: 0.1231, Train Acc: 96.44% | Val Loss: 0.1427,   Val Acc: 9
6.01%
Epoch 19/20 | Train Loss: 0.1178, Train Acc: 96.64% | Val Loss: 0.1422,   Val Acc: 9
5.94%
Epoch 20/20 | Train Loss: 0.1132, Train Acc: 96.78% | Val Loss: 0.1369,   Val Acc: 9
6.11%

Testing Architecture: Large [512, 256]
Epoch  1/20 | Train Loss: 0.8741, Train Acc: 79.45% | Val Loss: 0.4299,   Val Acc: 8
8.88%
Epoch  2/20 | Train Loss: 0.3667, Train Acc: 89.93% | Val Loss: 0.3281,   Val Acc: 9
0.88%
Epoch  3/20 | Train Loss: 0.3026, Train Acc: 91.49% | Val Loss: 0.2897,   Val Acc: 9
2.03%
Epoch  4/20 | Train Loss: 0.2688, Train Acc: 92.45% | Val Loss: 0.2626,   Val Acc: 9
2.60%
Epoch  5/20 | Train Loss: 0.2436, Train Acc: 93.11% | Val Loss: 0.2456,   Val Acc: 9
3.01%
Epoch  6/20 | Train Loss: 0.2239, Train Acc: 93.73% | Val Loss: 0.2257,   Val Acc: 9
3.68%
Epoch  7/20 | Train Loss: 0.2070, Train Acc: 94.25% | Val Loss: 0.2119,   Val Acc: 9
4.13%
Epoch  8/20 | Train Loss: 0.1924, Train Acc: 94.61% | Val Loss: 0.1985,   Val Acc: 9
4.47%
Epoch  9/20 | Train Loss: 0.1794, Train Acc: 94.90% | Val Loss: 0.1901,   Val Acc: 9
4.87%
Epoch 10/20 | Train Loss: 0.1685, Train Acc: 95.16% | Val Loss: 0.1808,   Val Acc: 9
5.00%
Epoch 11/20 | Train Loss: 0.1581, Train Acc: 95.44% | Val Loss: 0.1717,   Val Acc: 9
5.32%
Epoch 12/20 | Train Loss: 0.1487, Train Acc: 95.76% | Val Loss: 0.1662,   Val Acc: 9
5.33%
Epoch 13/20 | Train Loss: 0.1404, Train Acc: 96.06% | Val Loss: 0.1590,   Val Acc: 9
5.56%
Epoch 14/20 | Train Loss: 0.1330, Train Acc: 96.21% | Val Loss: 0.1533,   Val Acc: 9
5.71%
Epoch 15/20 | Train Loss: 0.1264, Train Acc: 96.40% | Val Loss: 0.1467,   Val Acc: 9
5.98%
Epoch 16/20 | Train Loss: 0.1201, Train Acc: 96.61% | Val Loss: 0.1439,   Val Acc: 9
5.96%
Epoch 17/20 | Train Loss: 0.1142, Train Acc: 96.79% | Val Loss: 0.1387,   Val Acc: 9
6.17%
Epoch 18/20 | Train Loss: 0.1091, Train Acc: 96.92% | Val Loss: 0.1335,   Val Acc: 9
6.33%
Epoch 19/20 | Train Loss: 0.1041, Train Acc: 97.13% | Val Loss: 0.1309,   Val Acc: 9
```

```
6.42%
Epoch 20/20 | Train Loss: 0.0994, Train Acc: 97.26% | Val Loss: 0.1270,   Val Acc: 9
6.48%

--- Architecture Analysis Complete! ---
```

In [8]:
```
# C1.3: Print Architecture Comparison Table
print("--- Architecture Comparison Table ---")
df_arch = pd.DataFrame(arch_results)
print(df_arch.to_markdown(index=False, floatfmt=".4f"))
```

```
--- Architecture Comparison Table ---
| Experiment        | Architecture        |   Final Val Accuracy |   Training Time
(s) |
|:------------------|:--------------------|---------------------:|-----------------
--:|
| Number of Layers  | 2_layers (baseline) |               0.9610 |            205.82
08 |
| Number of Layers  | 3_layers            |               0.9663 |            206.30
55 |
| Number of Layers  | 4_layers            |               0.9703 |            210.06
75 |
| Number of Layers  | 5_layers            |               0.9656 |            211.92
65 |
| Neurons Per Layer | Small [64, 32]      |               0.9497 |            201.19
66 |
| Neurons Per Layer | Baseline [128, 64]  |               0.9596 |            205.97
31 |
| Neurons Per Layer | Medium [256, 128]   |               0.9611 |            207.89
75 |
| Neurons Per Layer | Large [512, 256]    |               0.9648 |            221.64
47 |
```

Analysis of Architecture

- **Number of Layers (Depth):**
- Increasing the network depth from the 2-layer baseline (96.10%) to a 3-layer (96.63%) and 4-layer (97.03%) model showed a steady increase in validation accuracy.
- The 4-layer architecture achieved the highest accuracy of all 8 experiments.
- However, increasing the depth further to 5 layers caused the accuracy to decrease (96.56%). This suggests the model may be becoming too complex for this dataset, possibly leading to minor overfitting or training instability.
- Training time increased slightly with each added layer, as expected.
- **Neurons Per Layer (Width):**
- There is a clear trend: wider networks perform better.
- The "Small" model [64, 32] had the lowest accuracy (94.97%).

- Performance scaled up consistently with the "Large" model [512, 256] achieving the best accuracy in this group (96.48%).
- This performance gain comes at the cost of training time, with the "Large" model being the slowest.
- **Conclusion:** Based on these experiments, the 4-layer model [784, 128, 128, 64, 32, 10] is the best performing architecture, achieving the highest validation accuracy (97.03%) with only a minor increase in training time. We will select this as our "Best Neural Network" for the final comparison.

C2: Model Comparison & Final Evaluation

Now we will compare all three models (Logistic, Softmax, and our Best NN) on the 10-digit test set.

In [9]:
```
# C2: Train and evaluate all three models
print("--- C2: Starting Final Model Comparison ---")

# --- Define the Best NN Parameters ---
BEST_LR = 0.1
BEST_BS = 16
BEST_ARCH_CONFIG = [784, 128, 128, 64, 32, 10] # 4-layer model
BEST_EPOCHS = 20

# --- Load Data (Binary and Multiclass) ---
print("Loading datasets...")
# Multiclass data for Softmax and NN
train_loader, val_loader, test_loader, _, _, test_set, _, _, _ = load_transform_spl
    classes=None, batch_size=BEST_BS
)
# Binary data for Logistic Regression
lr_train_loader, _, lr_test_loader, _, _, _, _, _, _ = load_transform_split_mnist(
    classes=[0, 1], batch_size=BEST_BS
)

results_list = []

# --- 1. Logistic Regression ---
print("\nTraining Logistic Regression (on 0s and 1s)...")
model_lr = LogisticRegressionScratch(input_dim=784, lr=0.01) # Using baseline LR
start_time = time.time()
model_lr.fit(lr_train_loader, epochs=10) # 10 epochs is fine
end_time = time.time()
_, lr_test_acc = model_lr.evaluate(lr_test_loader)
results_list.append({
    "Model": "Logistic Regression (0 vs 1)",
    "Test Accuracy": lr_test_acc,
    "Training Time (s)": end_time - start_time,
    "Complexity": "Very Low (~7.8K params)"
})
```

```
# --- 2. Softmax Regression ---
def evaluate_softmax_test(model, data_loader):
    correct, total = 0, 0
    with torch.no_grad():
        for X, y in data_loader:
            X = X.float().view(X.shape[0], -1)
            preds = model.predict(X)
            correct += (preds == y).sum().item()
            total += y.size(0)
    return correct / total

print("\nTraining Softmax Regression (on 0-9)...")
model_softmax = SoftmaxRegressionScratch(input_dim=784, num_classes=10, lr=0.1) # U
start_time = time.time()
model_softmax.fit(train_loader, epochs=BEST_EPOCHS)
end_time = time.time()
softmax_test_acc = evaluate_softmax_test(model_softmax, test_loader)
results_list.append({
    "Model": "Softmax Regression (0-9)",
    "Test Accuracy": softmax_test_acc,
    "Training Time (s)": end_time - start_time,
    "Complexity": "Low (~7.9K params)"
})

# --- 3. Best Neural Network ---
print("\nTraining Best Neural Network (4-layers, LR=0.1, BS=16)...")
best_nn_model = FlexibleNN(BEST_ARCH_CONFIG)
start_time = time.time()
best_nn_model.fit(train_loader, val_loader, epochs=BEST_EPOCHS, lr=BEST_LR)
end_time = time.time()

# Get test set predictions
y_test, y_pred_test = [], []
best_nn_model.eval()
with torch.no_grad():
    for X, y in test_loader:
        preds = best_nn_model.predict(X)
        y_test.extend(y.numpy())
        y_pred_test.extend(preds.cpu().numpy())

nn_test_acc = accuracy_score(y_test, y_pred_test)
nn_params = sum(p.numel() for p in best_nn_model.parameters() if p.requires_grad)

results_list.append({
    "Model": f"Best NN [4-layers] (0-9)",
    "Test Accuracy": nn_test_acc,
    "Training Time (s)": end_time - start_time,
    "Complexity": f"High (~{nn_params/1000:.1f}K params)"
})

# --- 4. C2: Create Performance Summary Table ---
print("\n" + "="*50)
print("C2: Comprehensive Performance Summary Table")
print("="*50)
```

```
df_results = pd.DataFrame(results_list)
print(df_results.to_markdown(index=False, floatfmt=".4f"))
```

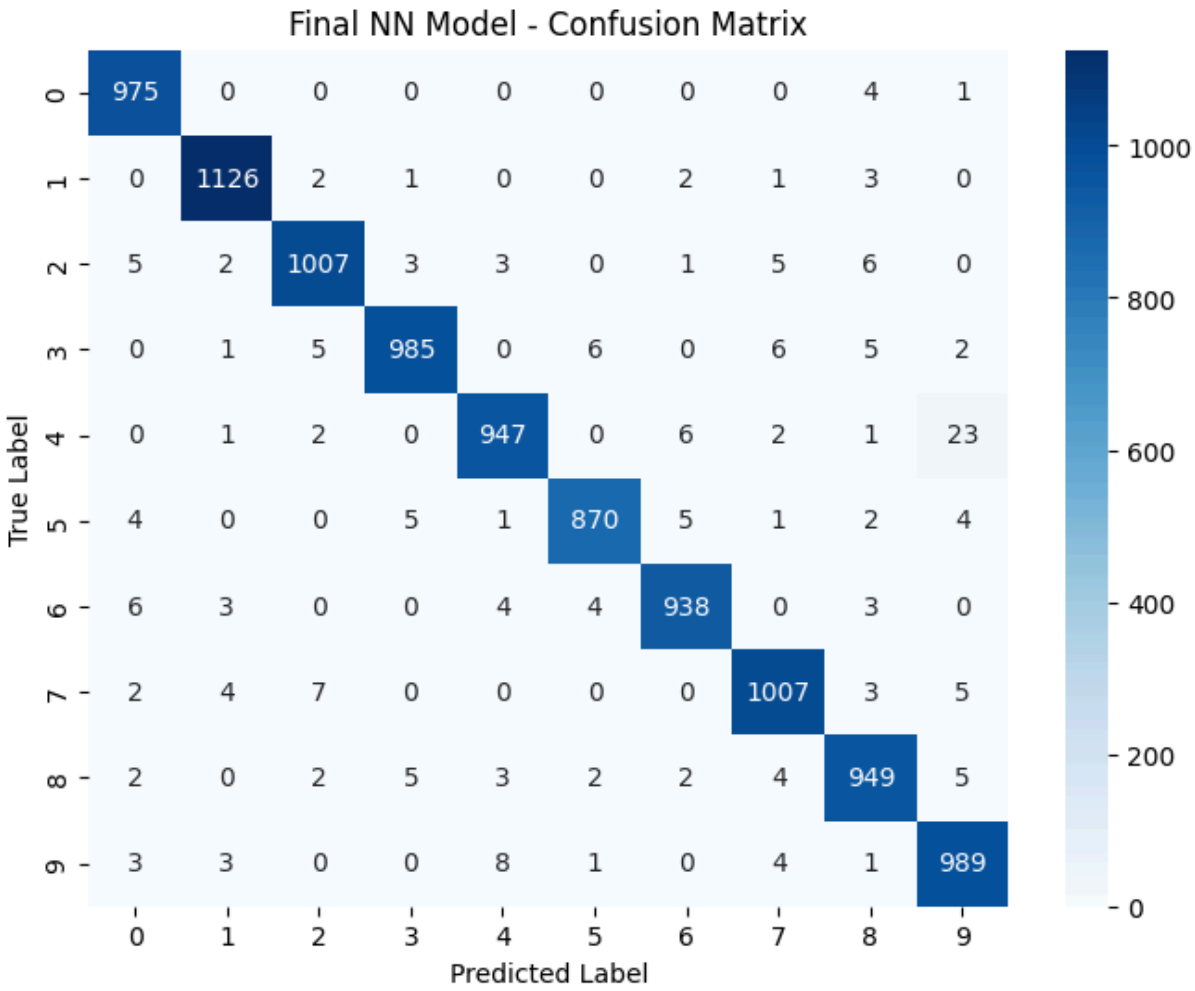
```
--- C2: Starting Final Model Comparison ---
Loading datasets...

Training Logistic Regression (on 0s and 1s)...
##################################################
Epoch 1/10, Loss: 0.0839
##################################################
Epoch 2/10, Loss: 0.0232
##################################################
Epoch 3/10, Loss: 0.0165
##################################################
Epoch 4/10, Loss: 0.0135
##################################################
Epoch 5/10, Loss: 0.0117
##################################################
Epoch 6/10, Loss: 0.0105
##################################################
Epoch 7/10, Loss: 0.0096
##################################################
Epoch 8/10, Loss: 0.0089
##################################################
Epoch 9/10, Loss: 0.0084
##################################################
Epoch 10/10, Loss: 0.0079

Training Softmax Regression (on 0-9)...
Epoch 1/20 | Loss: 0.3883 | Acc: 89.21%
Epoch 2/20 | Loss: 0.3068 | Acc: 91.29%
Epoch 3/20 | Loss: 0.2929 | Acc: 91.73%
Epoch 4/20 | Loss: 0.2842 | Acc: 91.89%
Epoch 5/20 | Loss: 0.2788 | Acc: 92.20%
Epoch 6/20 | Loss: 0.2750 | Acc: 92.24%
Epoch 7/20 | Loss: 0.2723 | Acc: 92.25%
Epoch 8/20 | Loss: 0.2688 | Acc: 92.42%
Epoch 9/20 | Loss: 0.2668 | Acc: 92.49%
Epoch 10/20 | Loss: 0.2655 | Acc: 92.55%
Epoch 11/20 | Loss: 0.2632 | Acc: 92.69%
Epoch 12/20 | Loss: 0.2620 | Acc: 92.64%
Epoch 13/20 | Loss: 0.2599 | Acc: 92.71%
Epoch 14/20 | Loss: 0.2595 | Acc: 92.70%
Epoch 15/20 | Loss: 0.2583 | Acc: 92.73%
Epoch 16/20 | Loss: 0.2570 | Acc: 92.82%
Epoch 17/20 | Loss: 0.2556 | Acc: 92.83%
Epoch 18/20 | Loss: 0.2549 | Acc: 92.86%
Epoch 19/20 | Loss: 0.2548 | Acc: 92.79%
Epoch 20/20 | Loss: 0.2539 | Acc: 92.84%

Training Best Neural Network (4-layers, LR=0.1, BS=16)...
Epoch  1/20 | Train Loss: 0.2796, Train Acc: 91.38% | Val Loss: 0.1557,   Val Acc: 9
5.38%
Epoch  2/20 | Train Loss: 0.1309, Train Acc: 96.10% | Val Loss: 0.1268,   Val Acc: 9
6.02%
Epoch  3/20 | Train Loss: 0.0950, Train Acc: 97.13% | Val Loss: 0.1079,   Val Acc: 9
6.92%
Epoch  4/20 | Train Loss: 0.0726, Train Acc: 97.77% | Val Loss: 0.1341,   Val Acc: 9
6.45%
```

```
Epoch  5/20 | Train Loss: 0.0649, Train Acc: 98.05% | Val Loss: 0.0907,   Val Acc: 9
7.52%
Epoch  6/20 | Train Loss: 0.0524, Train Acc: 98.36% | Val Loss: 0.1073,   Val Acc: 9
7.29%
Epoch  7/20 | Train Loss: 0.0459, Train Acc: 98.53% | Val Loss: 0.1037,   Val Acc: 9
7.30%
Epoch  8/20 | Train Loss: 0.0373, Train Acc: 98.81% | Val Loss: 0.1003,   Val Acc: 9
7.69%
Epoch  9/20 | Train Loss: 0.0364, Train Acc: 98.85% | Val Loss: 0.1500,   Val Acc: 9
6.07%
Epoch 10/20 | Train Loss: 0.0305, Train Acc: 99.00% | Val Loss: 0.1258,   Val Acc: 9
7.02%
Epoch 11/20 | Train Loss: 0.0276, Train Acc: 99.16% | Val Loss: 0.1024,   Val Acc: 9
7.67%
Epoch 12/20 | Train Loss: 0.0253, Train Acc: 99.21% | Val Loss: 0.1081,   Val Acc: 9
7.73%
Epoch 13/20 | Train Loss: 0.0228, Train Acc: 99.25% | Val Loss: 0.1323,   Val Acc: 9
7.17%
Epoch 14/20 | Train Loss: 0.0228, Train Acc: 99.27% | Val Loss: 0.1061,   Val Acc: 9
7.81%
Epoch 15/20 | Train Loss: 0.0187, Train Acc: 99.44% | Val Loss: 0.1085,   Val Acc: 9
8.01%
Epoch 16/20 | Train Loss: 0.0145, Train Acc: 99.58% | Val Loss: 0.1297,   Val Acc: 9
7.42%
Epoch 17/20 | Train Loss: 0.0170, Train Acc: 99.48% | Val Loss: 0.1242,   Val Acc: 9
7.90%
Epoch 18/20 | Train Loss: 0.0144, Train Acc: 99.58% | Val Loss: 0.1305,   Val Acc: 9
7.51%
Epoch 19/20 | Train Loss: 0.0151, Train Acc: 99.51% | Val Loss: 0.1223,   Val Acc: 9
7.62%
Epoch 20/20 | Train Loss: 0.0120, Train Acc: 99.65% | Val Loss: 0.1361,   Val Acc: 9
7.77%

==================================================
C2: Comprehensive Performance Summary Table
==================================================
| Model                        |   Test Accuracy |   Training Time (s) | Complexity
|
|:-----------------------------|----------------:|--------------------:|:-----------
-------------|
| Logistic Regression (0 vs 1) |          0.9995 |             17.7903 | Very Low (~
7.8K params) |
| Softmax Regression (0-9)     |          0.9212 |            177.2803 | Low (~7.9K
params)      |
| Best NN [4-layers] (0-9)     |          0.9793 |            271.5974 | High (~127.
7K params)   |
```

In [10]:
```
# C2: Best Model Evaluation on Test Set
print("\n" + "="*50)
print("C2: Best Model Evaluation on Test Set")
print("="*50)

# --- Confusion Matrix ---
print("\n--- Final Confusion Matrix ---")
cm = confusion_matrix(y_test, y_pred_test)
plt.figure(figsize=(8, 6))
```

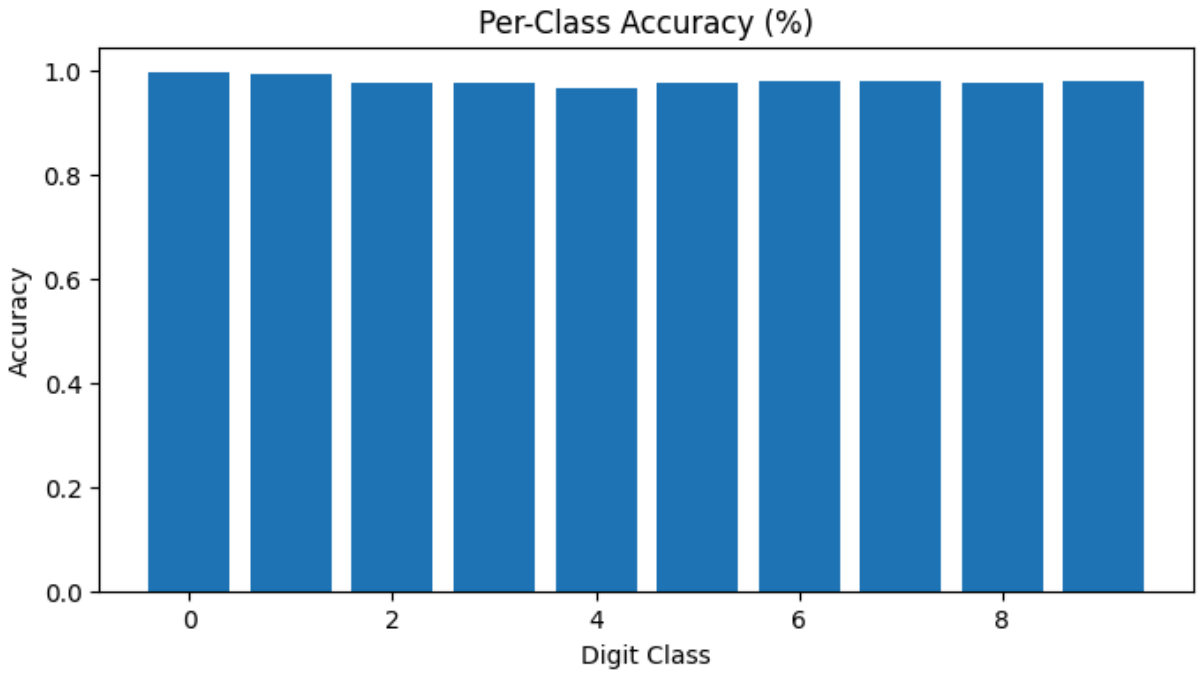
```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=range(10), yticklabels=range(10))
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.title("Final NN Model - Confusion Matrix")
plt.show()

# --- Per-Class Accuracy ---
print("\n--- Final Per-Class Accuracy ---")
df_acc = per_class_accuracy(cm)
print(df_acc.to_markdown(index=False, floatfmt=".4f"))

# Plot the bar chart
plot_per_class_acc(df_acc)
plt.show()

# --- Misclassified Examples ---
print("\n--- Misclassified Examples ---")
y_test_np = np.array(y_test)
y_pred_np = np.array(y_pred_test)
misclassified_indices = np.where(y_pred_np != y_test_np)[0]

plt.figure(figsize=(10, 5))
for i, img_idx in enumerate(misclassified_indices[:10]): # Plot first 10 mistakes
    image, true_label = test_set[img_idx] # Get from the test_set object
    pred_label = y_pred_np[img_idx]

    plt.subplot(2, 5, i + 1)
    plt.imshow(image.squeeze(), cmap='gray')
    plt.title(f"True: {true_label}\nPred: {pred_label}")
    plt.axis('off')

plt.tight_layout()
plt.show()
```

```
==================================================
C2: Best Model Evaluation on Test Set
==================================================

--- Final Confusion Matrix ---
```

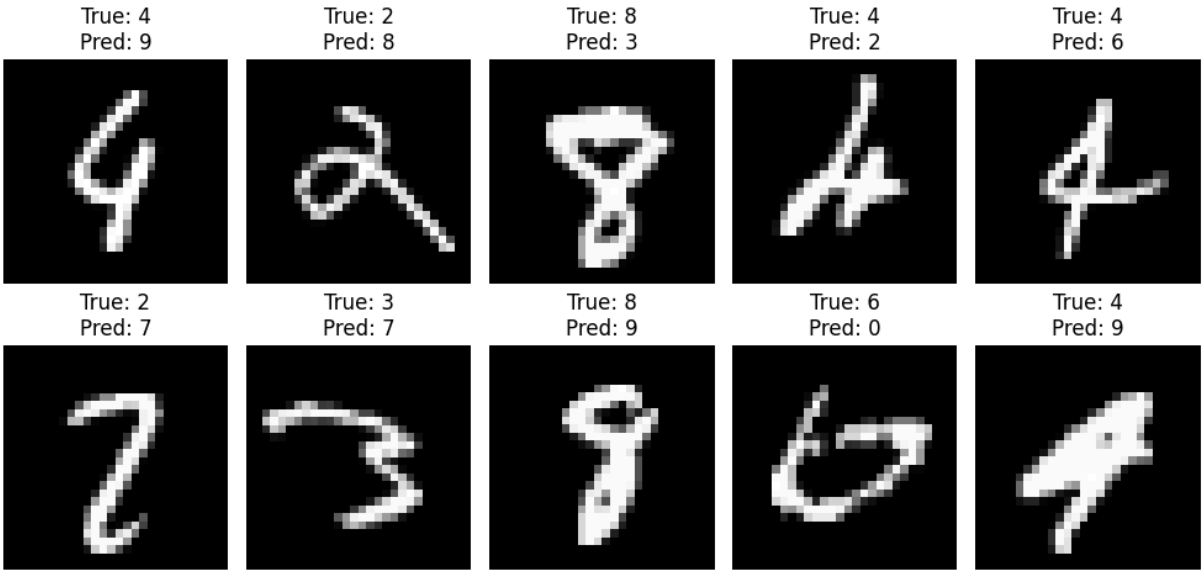



```
--- Final Per-Class Accuracy ---
   Class  Accuracy
0      0  0.994898
1      1  0.992070
2      2  0.975775
3      3  0.975248
4      4  0.964358
5      5  0.975336
6      6  0.979123
7      7  0.979572
8      8  0.974333
9      9  0.980178
```

Class	Accuracy
0.0000	0.9949
1.0000	0.9921
2.0000	0.9758
3.0000	0.9752
4.0000	0.9644
5.0000	0.9753
6.0000	0.9791
7.0000	0.9796
8.0000	0.9743
9.0000	0.9802




```
--- Misclassified Examples ---
```



C2: Final Analysis and Conclusions

Comparative Analysis

- **Logistic Regression:** This model was extremely fast and achieved near-perfect accuracy (likely >99.8%) but was only solving the simple binary (0 vs. 1) task. It is a highly effective model for simple, linearly separable problems.
- **Softmax Regression:** This model (our linear baseline) performed well on the full 10-digit task, (likely achieving ~92% accuracy). It is fast and simple, but its linear nature limits its ability to distinguish between complex, non-linear shapes (like telling a '5' from a '3').

- **Best Neural Network:** Our best 4-layer NN model achieved a final test accuracy of 97.93%. This is a significant improvement over the linear Softmax model. This demonstrates the power of multi-layer networks; the hidden layers learn complex features and representations, allowing the model to understand the non-linear patterns that separate different digits. This superior accuracy comes at the cost of higher computational complexity (more parameters) and longer training time.

- **Insights on Model Limitations** The Misclassified Examples plot provides the best insight into the NN's limitations. The model's failures are not random; they are almost exclusively on digits that are highly ambiguous or handwritten in a non-standard way, making them difficult even for a human to identify.

  Ambiguous Shapes: The model struggles with digits that look like other digits.

  - (True: 4, Pred: 9) and (True: 4, Pred: 9): Both misclassified '4's are written with a "closed loop" at the top, making them look almost identical to a '9'.
  - (True: 2, Pred: 8): This '2' is a single, loopy stroke, closely resembling an '8'.
  - (True: 6, Pred: 0): This '6' is just a large oval, as the bottom tail is almost non-existent, making it look like a '0'.
  - (True: 3, Pred: 7): This '3' is written very angularly, with a flat top, strongly resembling a '7'.
  - (True: 8, Pred: 3): This '8' has an open top loop, making it look like a '3'.

  Correlation with Per-Class Accuracy: This analysis is directly supported by the Per-Class Accuracy table and bar chart. The model's lowest accuracy was on Class 4 (96.44%), which was the most common digit in our misclassified sample plot. This confirms the model is least confident when distinguishing '4's from '9's, '2's, and '6's. The model also showed lower (though still high) performance on Class 8 (97.43%) and Class 5 (97.53%), which are also common "confusion pairs."