Classical and Neural Classifiers on MNIST Dataset

Brock Badeaux
CSC 4444

October 25, 2025

Abstract

This report implements and compares five standard classifiers on handwritten digits. KNN, Bernoulli Naïve Bayes, a linear classifier, an MLP, and a small CNN using a simple pipeline (normalize to $[0, 1]$, flatten to 784 for non-CNN models; keep 28×28 for CNN) and a consistent 80:20 train/test split. For KNN, cosine distance reliably beat Euclidean with best performance around $k \in \{1, 3, 5\}$. Bernoulli Naïve Bayes worked best with threshold τ=0.5 and small Laplace α (e.g., 10^{-6}). The linear model converged quickly (roughly ∼50 epochs) under SGD; for the MLP ($784 \to 256 \to 128 \to 10$), cross-entropy loss clearly outperformed MSE, with a moderate learning rate giving the best trade-off. The CNN (two conv blocks with BN/Dropout, followed by a three-layer head) achieved the highest test accuracy overall, saturating in roughly ∼100 epochs with SGD. Across models, the hardest classes were **5** and **8**, with recurring confusions **5→3** and **9↔4**. Key take away is that even simple baselines are strong on MNIST, but the right distance metric/loss and convolutional structure make a noticeable difference.

1 Project Description

The problem being addressed in this paper is one that has been repeated multiple times in the past. That is, the classification of handwritten digits through common methods of classification. More importantly these various methods will give insight into the nature of character recognition and classification methods. While this problem, and its solutions, have been extensively deployed; this report serves an exercise to the author (myself) and hopefully gives insight to its readers in the future.

2 Dataset

The source for the MNIST dataset is currently widely available online, however the exact source of this dataset was an attached zip file in an email received from Professor Lao. This zip file contains 10 folders labeled 0-9 with each folder containing roughly 10,000 .png 28x28 grayscale images.

Figure 1: Example digits from dataset.

To extract and label data a custom data loader takes advantage of the file structure and naming to return corresponding image and label NumPy arrays. After this a data pre-processor normalizes

pixel values to $[0, 1]$ and flattens images to 784 Dimension vectors when needed. Finally an $80:20$ split partitioning training and test sets is used throughout every model consistently.

3 AI Techniques (Methods)

While each method is slightly different, the ultimate goal for each model is to classify a digit given a data point (image). This report showcases five common classification models KNN (K-Nearest Neighbor), Bernoulli Naïve Bayes, Linear Classifier, Multilayer Perceptron, and finally CNN (Convolutional Neural Network). While each model is similar in its goal, it will become clear that each model differs in its fundamental structure, hyperparameters (the parameters that are decided upon by the model creator), training time, inference time, and accuracy. The first three models will be expressed mathematically then generic implementation details are outlined. While the MLP and CNN will cover a higher level architecture for brevity.

3.1 K-Nearest Neighbor

The K-Nearest Neighbor model is unique in that it does not have a traditional training cycle and instead the bulk of compute time is done at inference time. During inference time a distance metric is used to find the nearest neighbors to a given sample point and a majority vote is used as prediction. For the MNIST dataset the data-points are represented as $\mathbf{x} \in \mathbb{R}^{28\times 28}$. For this model, and every model besides the CNN flattening and normalized data is needed or preferred. By using the previously mentioned data preprocessor data points are represented as $\mathbf{x} \in \mathbb{R}^{784}$ and stored in $\mathbf{X} \in \mathbb{R}^{N\times D}$. Where N is the number of training samples and D is the 784 normalized features. Using a 80:20 partition of training and testing:

$$\mathbf{X}_{\text{tr}} \in \mathbb{R}^{48{,}000\times 784} \text{ and } \mathbf{X}_{\text{te}} \in \mathbb{R}^{12{,}000\times 784}$$

There are many distance metrics for K-Nearest Neighbors, with the most natural being Euclidean distance. This can be represented as $\|\mathbf{X}_{\text{te}} - \mathbf{X}_{\text{tr}}\|_2$.
With misaligned dimensions its is not possible to take advantage of NumPy's vectorization. However it is possible to compute a distance matrix D where

$$\mathbf{X}_{tr} \in R^{N_{tr}\times D}, \mathbf{X}_{te} \in \mathbb{R}^{N_{te}\times D}, D_{ij} = \|x_j^{(tr)} - x_j^{(te)}\|$$

With s_{tr} as the training squared-norm vector and s_{te} as the test squared-norm row vector. The squared distance matrix gro [2] is

$$D^2 = \mathbf{s}_{tr}\mathbf{1}_{N_{te}}^\top + \mathbf{1}_{N_{te}}\mathbf{s}_{te} - 2X_{tr}X_{te}^\top \in \mathbb{R}^{N_{tr}\times N_{te}}$$

There are also angular metrics such as cosine distance Prabhakaran [8]. For two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^D$, the cosine similarity and distance are defined by

$$\cos\theta(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|_2 \, \|\mathbf{b}\|_2}, \qquad d_{\cos}(\mathbf{a}, \mathbf{b}) = 1 - \cos\theta(\mathbf{a}, \mathbf{b}).$$

The cosine similarity matrix $\mathbf{C}$ and the cosine distance matrix $\mathbf{D}_{\cos}$ is

$$\mathbf{C} = \mathbf{P} \oslash \mathbf{N}, \qquad \mathbf{D}_{\cos} = \mathbf{1}_{N_{tr}\times N_{te}} - \mathbf{C},$$

where $\oslash$ denotes element-wise division and $\mathbf{1}_{N_{tr}\times N_{te}}$ is the all–ones matrix of size $N_{tr} \times N_{te}$.

For each test point (each column of $\mathbf{D}$), partial selection is used to identify the indices of the k smallest distances without fully sorting the column. Stacking into

$$\mathbf{I} \in \mathbb{N}^{k \times N_{te}}, \quad \text{where } \mathbf{I}_{:,j} \text{ are the } k \text{ nearest training indices to } x_j^{(te)}.$$

For each test point j, collect the neighbor labels $\{ y_i^{(tr)} : i \in \mathbf{I}_{:,j} \}$ and predict by majority vote using a fixed tie–breaking rule if needed (e.g., smallest label).

3.2 Naïve Bayes

Naïve Bayes classifier relies on the assumption that pixels are conditionally independent given class $c \in \{0, \ldots, 9\}$. Intuitively this sounds slightly wrong because nearby pixels should have some sort of correspondence; however looking into this idea would no longer be Naïve... To make this a Bernoulli Naïve classifier Brownlee [5]scikit-learn developers [9]leorrose [7] take binarized pixels using a threshold τ,

$$\bar{x}_j = \begin{cases} 1, & x_j \geq \tau, \\ 0, & x_j < \tau, \end{cases} \qquad \tau = 0.5$$

with this altered dataset take the Bernoulli likelihood with Laplace smoothing:

$$\Pr(\bar{x}_j = 1 \mid c) = \hat{\theta}_{c,j}, \qquad \hat{\theta}_{c,j} = \frac{N_{c,j}^{(1)} + \alpha}{N_c + 2\alpha}, \ \alpha > 0.$$

Where N is the count. By Bayes' Theorem the posterior

$$\Pr(c|\mathbf{x}) \propto \Pr(c) \prod_j \hat{\theta}_{c,j}^{x_j} (1 - \hat{\theta}_{c,j})^{1 - x_j}.$$

For a more numerically stable result taking the log of posterior yields

$$\log \Pr(c \mid \mathbf{x}) \propto \log \Pr(c) + \sum_j x_j \log \hat{\theta}_{c,j} + (1 - x_j) \log(1 - \hat{\theta}_{c,j}).$$

From here the predicted label is the maximum argument for the posterior.

3.3 Linear Classifier

Consider $\hat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b}$. Where $\mathbf{x}$ represents a sample data point, $\mathbf{W}$ is an arbitrary weight matrix, $\mathbf{b}$ is an arbitrary bias and $\hat{\mathbf{y}}$ is a score. With $\mathbf{y}$ as our ground truth label and choosing L2 loss function with one-hot encoded output

$$\mathcal{L} = \frac{1}{2N} \sum_{i=1}^{N} \|(\mathbf{W}\mathbf{x}_i + \mathbf{b}) - \mathbf{y}_i\|_2^2.$$

For a batch of N samples, with D features and C classes $X \in \mathbb{R}^{N \times D}$, $Y \in \mathbb{R}^{N \times C}$, $\hat{Y} = X\mathbf{W}^T + \mathbf{1_N}\mathbf{b}^T$, the gradient han [3]cs2 [1] becomes

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{1}{N} \left(\hat{Y} - Y \right)^\top X \in \mathbb{R}^{C \times D}$$

and

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \frac{1}{N} (\hat{\mathbf{Y}} - \mathbf{Y})^\top \mathbf{1}_N \in \mathbb{R}^{C \times 1}$$

With this update weights with learning rate η

$$W \leftarrow W - \eta \frac{\partial \mathcal{L}}{\partial W}$$
$$\mathbf{b} \leftarrow \mathbf{b} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}}$$

To train this model, initialize W and $\mathbf{b}$ (zero initialization is acceptable for a single linear). Train for n epochs with mini-batches: at each epoch, shuffle the data, then for each batch perform forward $\rightarrow$ loss $\rightarrow$ backward $\rightarrow$ parameter update with learning rate η, for a total of $\frac{N_{\text{tr}}}{N_{\text{batch}}}$ updates per epoch.

After training, predictions are obtained by a forward pass with the learned $W, \mathbf{b}$, producing logits $\hat{Y} \in \mathbb{R}^{N_{\text{te}} \times C}$. The predicted class for sample i is

$$\hat{y}_i = \arg \max_{c \in \{1,\ldots,C\}} \hat{Y}_{i,c},$$

i.e., an argmax over classes for each row.

3.4 Multilayer Perceptron

Multilayer perceptrons aim to add non linearity by use of multiple hidden linear layers with various activation functions (ReLU, Softmax, tanh., etc.). The choice of layer size, activation function, and loss function is mostly empirical however there are strong defaults for MNIST. The architecture tested for the MLP is $784 \rightarrow 256 \rightarrow 128 \rightarrow 10$ with ReLU between each layer. For this architecture cross-entropy if favored over L2 loss. For optimizer either Adam or SGD can be used. After fitting; the prediction function simply sets the model to evaluation mode and produces returns a list of predictions. Kuo [6]

3.5 Convolutional Neural Network

The final method, and best suited for MNIST, is the Convolutional Neural Network. Convloutional layers perform a convolution over a grid via a kernal and have input and output channels. Multiple convolutions can be strung together in the hopes of capturing more fundamental features such as edges and curves while further layers might pick up on more complex features. To reduce the chance of overfitting batch normilization and dropout is used. Finally a fully connected (deep) layer is used to classify these features that the convolution learns. Artley [4]

Block 1: Conv3×3(1→32) – ReLU – BN_{32}; Conv3×3(32→32) – ReLU – BN_{32}; MaxPool2; Dropout(0.25)

Block 2: Conv3×3(32→64) – ReLU – BN_{64}; Conv3×3(64→64) – ReLU – BN_{64}; MaxPool2; Dropout(0.25)

Head: Flatten; FC(3136→512) – ReLU – BN_{512} – Dropout(0.25); FC(512→1024) – ReLU – BN_{1024} – Dropout(0.5); FC(1024→10)

Table 1: Tested hyperparameters

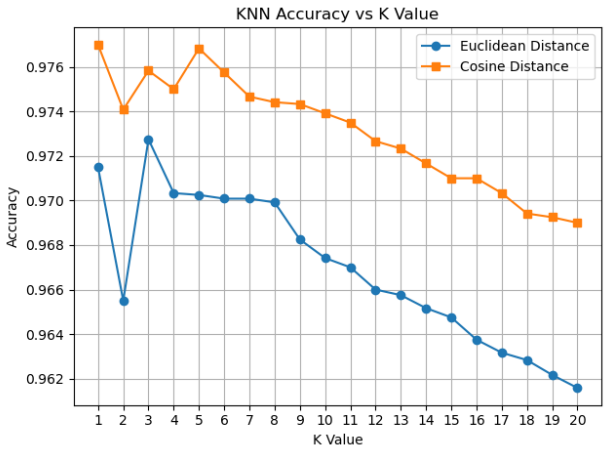
Model	Settings
KNN	$k \in \{1, \ldots, 20\}$; distance $\in$ {Euclidean, Cosine}
Naïve Bayes	$\alpha \in \{10^{-6}, 10^{-3}, 10^{-1}, 0.5\}$; threshold $\tau \in \{0.30, 0.40, 0.45, 0.50, 0.55, 0.60\}$
Linear	learning rate $\eta \in \{0.001, 0.01\}$; epochs $\in$ {10, 50, 100, 200, 500}
MLP	$LR \in \{0.01, 0.1, 0.2\}$; weight decay $\in \{0.0, 10^{-5}\}$; optimizer $\in$ {SGD, Adam}
CNN	AdamW/SGD; weight decay 10^{-4}; cosine LR; batch size 256

4 Experiments

5 Results

5.1 Learning Curves

5.1.1 KNN



Cosine distance consistently outperforms Euclidean. The best region is $k \in \{1, 3, 5\}$

5.1.2 Naïve Bayes (Bernoulli)

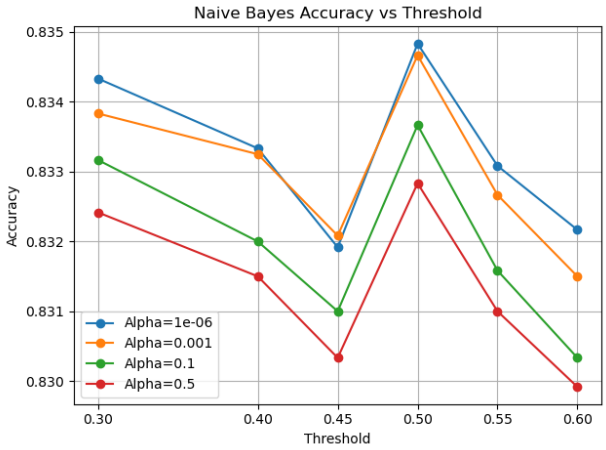


Figure 2: Naïve Bayes accuracy across thresholds τ and Laplace α.

Best setting at τ=0.5 and α=10^{-6}, lower α seems to correlate with higher accuracy with diminishing returns after 10^{-3}

5.1.3 Linear Classifier

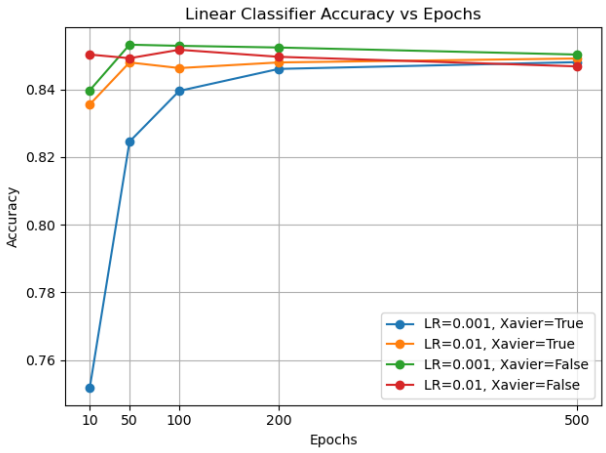


Figure 3: Linear classifier accuracy vs. epochs and init settings.

Zero (standard) initialization with η as used in training converges quickly; about 50 epochs works best, with diminishing returns for additional epochs.

5.1.4 MLP

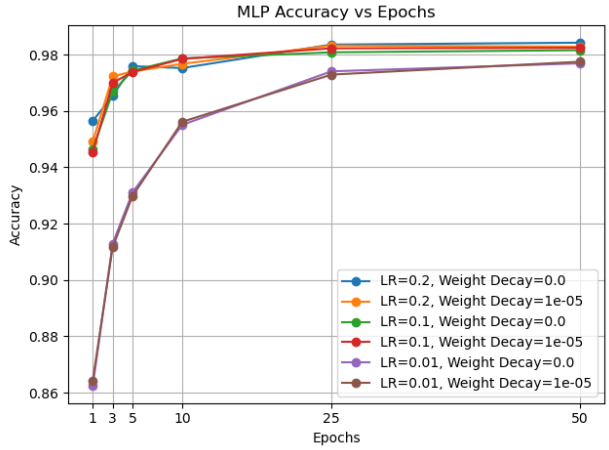


Figure 4: MLP accuracy vs. epochs for several learning rates and weight decay.

Best performance at LR $= 0.1$ around 25 epochs; further training gives diminishing returns. Weight decay (10^{-5}) did not help on this setup.

5.2 Loss comparison (MLP).

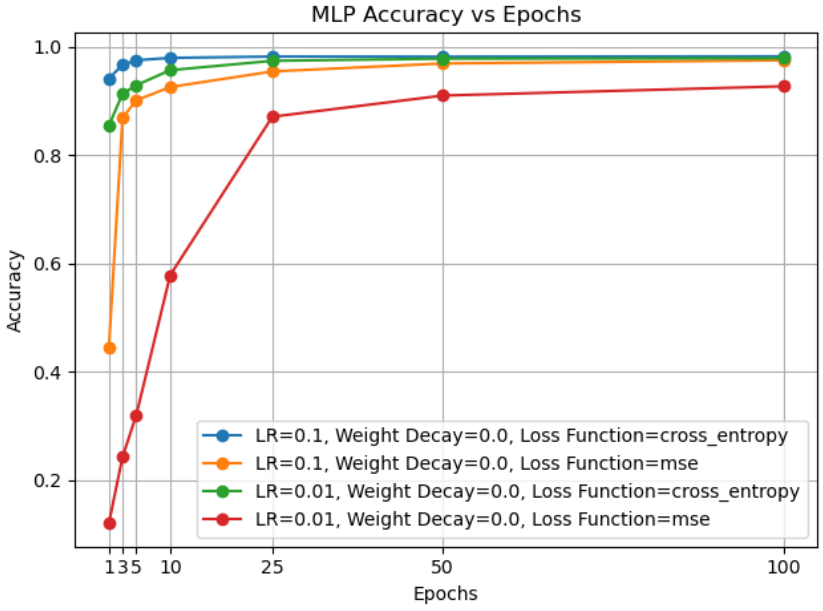


Figure 5: MLP: Cross-entropy vs. MSE loss.

Cross-entropy clearly outperforms MSE for classification.

5.2.1 CNN

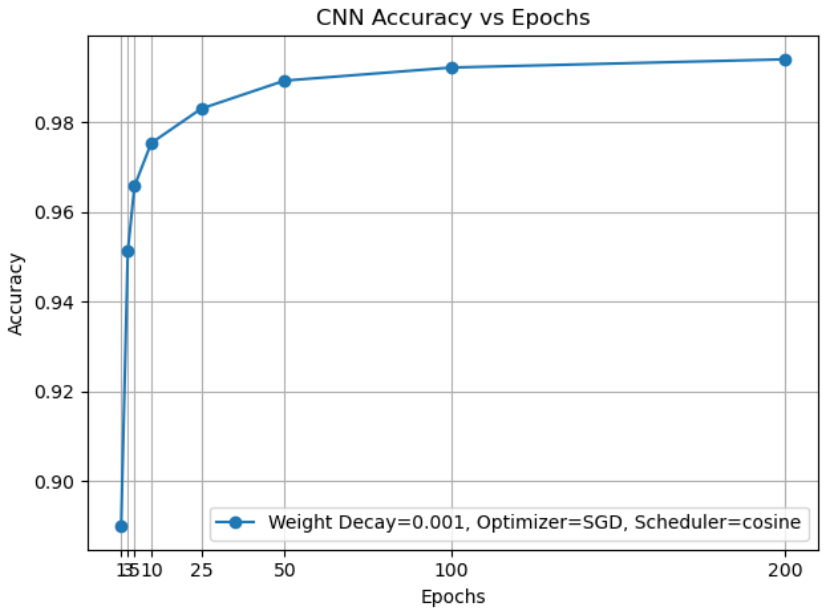
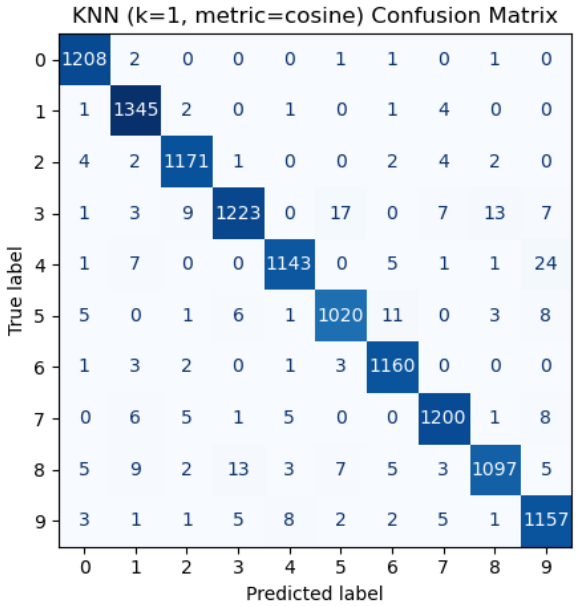


Figure 6: CNN accuracy vs. epochs (SGD, cosine schedule, wd 10^{-3}).

CNN attains the best overall accuracy; with SGD it needs ~ 100 epochs to saturate. Adam may reach similar accuracy in fewer epochs (not fully explored here).

5.3 Confusion Matrices & Error Analysis

5.3.1 KNN (k=1, cosine)



Worst class: **5** was frequently misclassified as **3**. **8** is also often confused with **3**. Most prominent pairwise confusion: **9** ↔ **4**.

5.3.2 Naïve Bayes (Bernoulli)

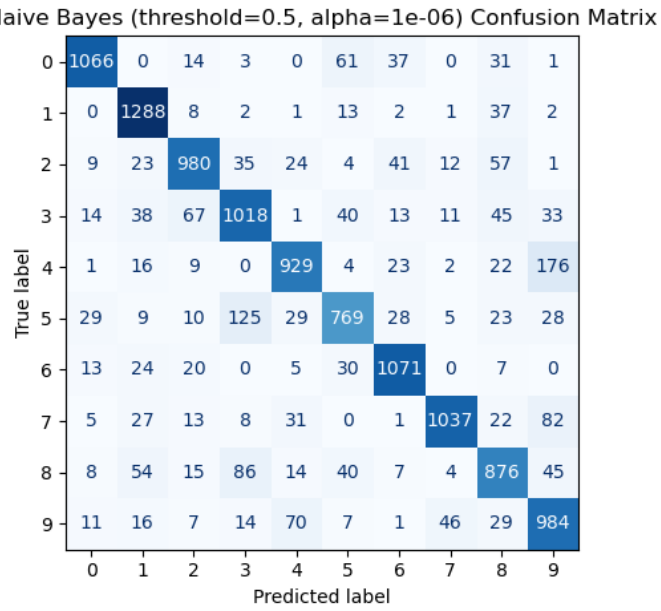


Figure 7: Naïve Bayes confusion matrix (τ=0.5, α=10^{-6}).

Weakest on **5** and **8**. Strong confusions: **4**→**9** and **5**→**3**.

5.3.3 Linear Classifier

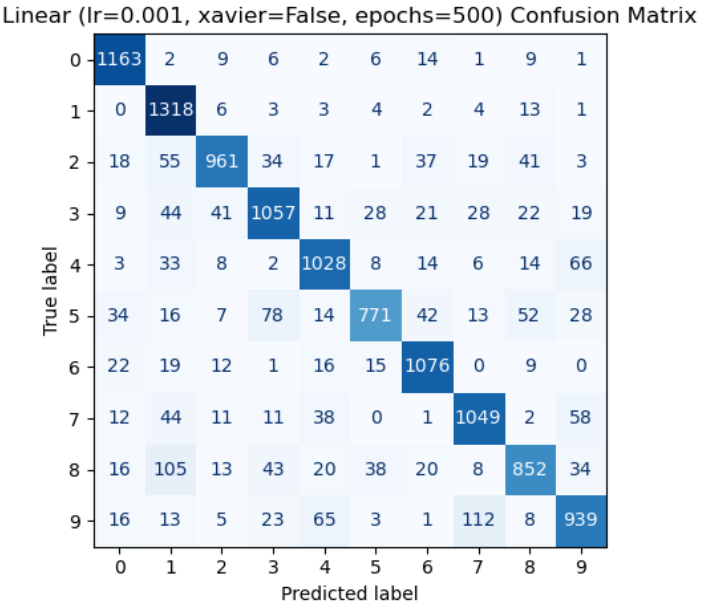


Figure 8: Linear model confusion matrix (lr 10^{-3}, no Xavier, 500 ep.).

Weakest on **5** and **8**. Most notable confusion: **9**→**7**.

5.3.4 MLP

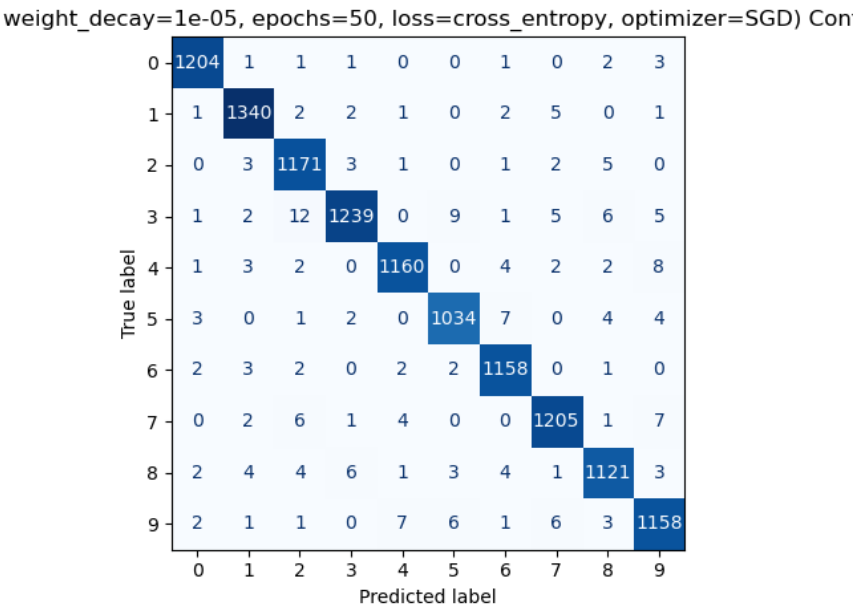


Figure 9: MLP confusion matrix (CE loss, SGD).

Weakest on **5**; most common confusion is **5**→**3**.

5.3.5 CNN

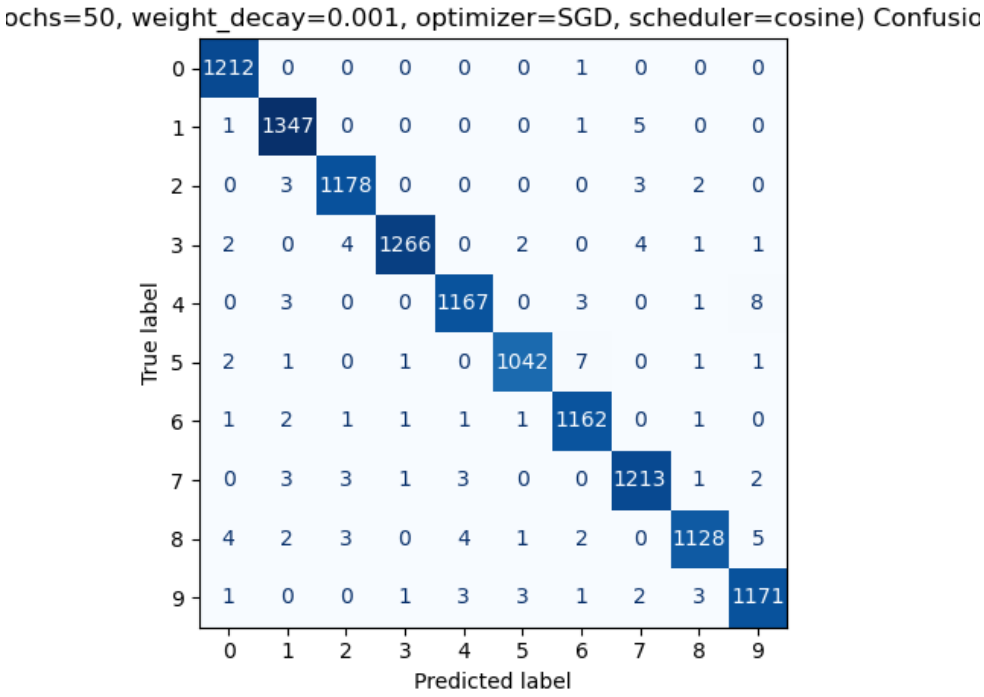


Figure 10: CNN confusion matrix (SGD + cosine, wd 10^{-3}).

Remaining errors concentrate on **5** and **8** (hardest classes overall with 2 consistently third).

A Public Repo

`https://github.com/BrockBadeaux14/Classical-and-Neural-Classifiers-on-MNIST-Dataset`

References

[1] Cs231n notes: Linear classifiers and backpropagation. `https://cs231n.stanford.edu/handouts/linear-backprop.pdf`. Accessed: 2025-10-24.

[2] Csc2515 tutorial 1: k-nearest neighbours. `https://nbviewer.org/url/www.cs.toronto.edu/~rgrosse/courses/csc2515_2019/tutorials/tut1/tut1_knn.ipynb`. Accessed: 2025-10-24.

[3] Matrix calculus (notes). `https://www.doc.ic.ac.uk/~ahanda/referencepdfs/MatrixCalculus.pdf`. Accessed: 2025-10-24.

[4] Brendan Artley. Mnist keras simple cnn (99.6%). `https://medium.com/@BrendanArtley/mnist-keras-simple-cnn-99-6-731b624aee7f`. Accessed: 2025-10-24.

[5] Jason Brownlee. Naive bayes classifier from scratch in python. `https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/`. Accessed: 2025-10-24.

[6] Yu-Cheng (Morton) Kuo. Ml14: Pytorch — mlp on mnist. https://medium.com/analytics-vidhya/ml14-f03f75254934, 2020. Published on Analytics Vidhya (Medium), Accessed: 2025-10-24.

[7] leorrose. Mnist digit recognition — bernoulli naive bayes (kaggle notebook). https://www.kaggle.com/code/leorrose/mnist-digit-recognition-bernoulli-naive-bayes. Accessed: 2025-10-24.

[8] Selva Prabhakaran. Cosine similarity — definition, formula and python examples. https://www.machinelearningplus.com/nlp/cosine-similarity/. Accessed: 2025-10-24.

[9] scikit-learn developers. Naive bayes — scikit-learn user guide. https://scikit-learn.org/stable/modules/naive_bayes.html. Accessed: 2025-10-24.