# handwritting_simple_ANN_zenuni

May 12, 2025

## 0.1 Digit Recognition Demo Notebook

**Author: Bledar Blake Zenuni - May 2025** A. For demonstration purposes only, inspired by the Oxford AI Programme 2025 (which I highly recommend!) – no copyright materials or modules are shared.

B. This notebook demonstrates a simple artificial neural network (ANN) built to recognise handwritten digits using a standard dataset.

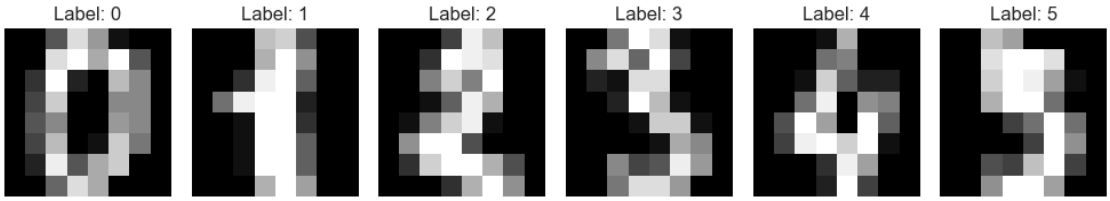
C. Challenges discussed in the accompanying essay are drawn from reflecting on my own handwriting; the implementation here uses publicly available data for reproducibility and technical demonstration.

---

### 1 - Importing Python Libs, Loading and Visualizing Sample Digits

[13]:
```
# importing libs
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix as cm, ConfusionMatrixDisplay
from yellowbrick.classifier import ConfusionMatrix as YBConfusionMatrix

#load and visualise sample digits

digits = load_digits()

# see dataset shape
print(f"Data shape: {digits.data.shape}")
print(f"Target shape: {digits.target.shape}")

# plotting first 6 images
fig, axes = plt.subplots(1, 6, figsize=(10, 3))
for i, ax in enumerate(axes):
    ax.imshow(digits.images[i], cmap='gray')
    ax.set_title(f"Label: {digits.target[i]}")
    ax.axis('off')
plt.tight_layout()
plt.show()
```

```
Data shape: (1797, 64)
Target shape: (1797,)
```




- Sample digits show structure and consistent spacing; this is useful for training but less representative of real handwriting variability
- This lets the model learn basic visual features like loops, straight lines, and stroke angles

---

**2 - Preprocessing Data and Creating Train-Test Split**

[14]:
```
# preprocess data and create train-test split
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# flatten image data and scale to [0, 1]
X = digits.data
y = digits.target
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)

print(f"Training samples: {X_train.shape[0]}")
print(f"Test samples: {X_test.shape[0]}")
```

```
Training samples: 1437
Test samples: 360
```

- This is "feature scaling" and it ensures uniform input for the network, reducing bias from pixel intensity variation
- We tried to generate a clean train-test split to allow us to evaluate the model's ability to generalise; this is a core concern when handling messy handwriting; reflecting on my own handwritting (Question 1), this is a concern sometimes also when I scribble digits down quickly
- Lastly, preprocessing is critical for addressing variation and inconsistency, key challenges highlighted in the assignment

### 3 - Defining and Training a Simple ANN Model

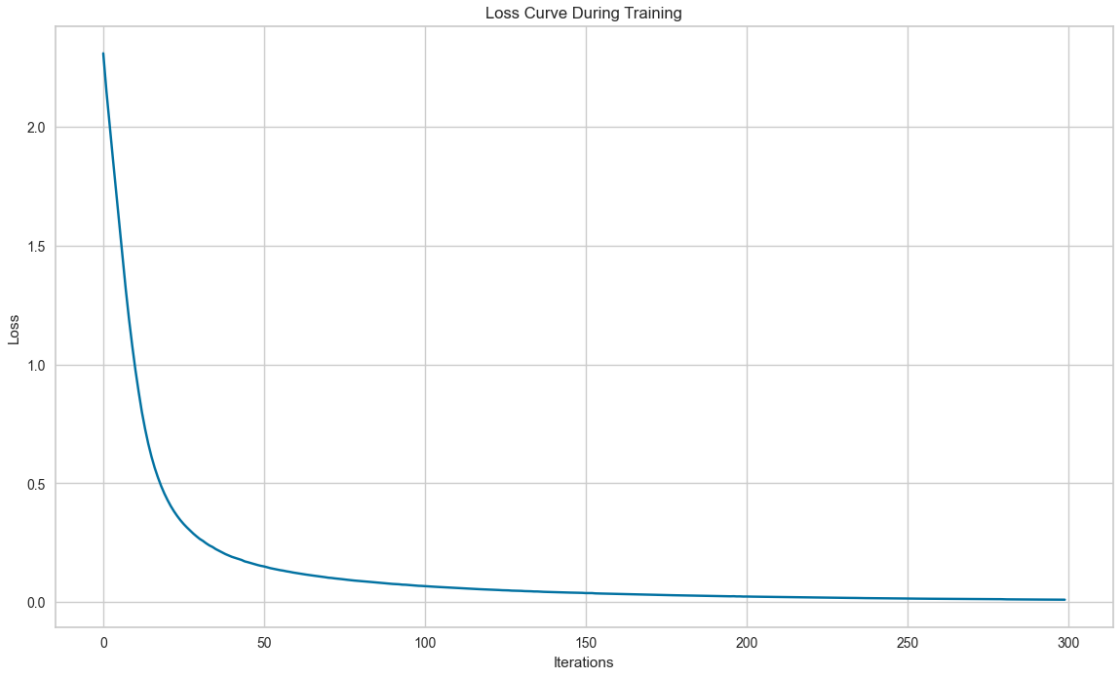
[15]:
```
# define and train a simple ANN model
from sklearn.neural_network import MLPClassifier

# defines
model = MLPClassifier(hidden_layer_sizes=(64,), activation='relu',␣
 ↪solver='adam',
                      max_iter=300, random_state=42)

# trains
model.fit(X_train, y_train)

# outputs that training is complete
print("Model training complete.")
```

```
Model training complete.

C:\Users\blake\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:690:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

- This model should have (behind the scenes) a single hidden layer with ReLU activation that captures non-linear patterns in the digit images
- It then learns internal weights to distinguish visual differences between digits, such as loops and angles
- Though the training completes in seconds, the learning reflects core ANN principles discussed in Questions 2 and 3.

Nota Bene (NB): Since the model was trained with `verbose=false` we can still access it by plotting the loss curve after training:

[16]:
```
plt.figure(figsize=(14, 8))
plt.plot(model.loss_curve_)
plt.title("Loss Curve During Training")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```




- Loss curve shows us rapid initial improvement, followed by convergence
- It indicates the model successfully learned meaningful patterns and minimised error over time

NB: To see more during training, we can enable `verbose` – we should see the model improve after each iteration:

[17]:
```
# model with verbose output
model = MLPClassifier(hidden_layer_sizes=(64,), activation='relu',␣
 ↪solver='adam',
                      max_iter=300, random_state=42, verbose=True)

model.fit(X_train, y_train)
```

```
Iteration 1, loss = 2.30923323
Iteration 2, loss = 2.14782517
Iteration 3, loss = 2.00492474
Iteration 4, loss = 1.86583888
Iteration 5, loss = 1.72770651
Iteration 6, loss = 1.58793817
Iteration 7, loss = 1.44879617
Iteration 8, loss = 1.31601078
Iteration 9, loss = 1.19185759
Iteration 10, loss = 1.08003923
Iteration 11, loss = 0.97710348
Iteration 12, loss = 0.88556016
Iteration 13, loss = 0.80098396
```

```
Iteration 14, loss = 0.72942595
Iteration 15, loss = 0.66617108
Iteration 16, loss = 0.61204547
Iteration 17, loss = 0.56517691
Iteration 18, loss = 0.52489484
Iteration 19, loss = 0.48870926
Iteration 20, loss = 0.45702367
Iteration 21, loss = 0.42920171
Iteration 22, loss = 0.40410001
Iteration 23, loss = 0.38203623
Iteration 24, loss = 0.36239515
Iteration 25, loss = 0.34452361
Iteration 26, loss = 0.32864073
Iteration 27, loss = 0.31445408
Iteration 28, loss = 0.30122791
Iteration 29, loss = 0.28845802
Iteration 30, loss = 0.27690421
Iteration 31, loss = 0.26600429
Iteration 32, loss = 0.25751614
Iteration 33, loss = 0.24743899
Iteration 34, loss = 0.23860286
Iteration 35, loss = 0.23184010
Iteration 36, loss = 0.22301143
Iteration 37, loss = 0.21576449
Iteration 38, loss = 0.20903109
Iteration 39, loss = 0.20204898
Iteration 40, loss = 0.19655575
Iteration 41, loss = 0.19059257
Iteration 42, loss = 0.18618747
Iteration 43, loss = 0.18185880
Iteration 44, loss = 0.17749748
Iteration 45, loss = 0.17130142
Iteration 46, loss = 0.16782849
Iteration 47, loss = 0.16381840
Iteration 48, loss = 0.15956677
Iteration 49, loss = 0.15607615
Iteration 50, loss = 0.15266033
Iteration 51, loss = 0.15016761
Iteration 52, loss = 0.14627333
Iteration 53, loss = 0.14283890
Iteration 54, loss = 0.14008030
Iteration 55, loss = 0.13722922
Iteration 56, loss = 0.13450842
Iteration 57, loss = 0.13237152
Iteration 58, loss = 0.12959312
Iteration 59, loss = 0.12740655
Iteration 60, loss = 0.12465134
Iteration 61, loss = 0.12241327
```

```
Iteration 62, loss = 0.12004347
Iteration 63, loss = 0.11833410
Iteration 64, loss = 0.11601698
Iteration 65, loss = 0.11397188
Iteration 66, loss = 0.11221115
Iteration 67, loss = 0.11041439
Iteration 68, loss = 0.10902420
Iteration 69, loss = 0.10695074
Iteration 70, loss = 0.10489426
Iteration 71, loss = 0.10272188
Iteration 72, loss = 0.10165531
Iteration 73, loss = 0.09985785
Iteration 74, loss = 0.09860522
Iteration 75, loss = 0.09701296
Iteration 76, loss = 0.09521440
Iteration 77, loss = 0.09379509
Iteration 78, loss = 0.09211076
Iteration 79, loss = 0.09084608
Iteration 80, loss = 0.08976813
Iteration 81, loss = 0.08867083
Iteration 82, loss = 0.08715935
Iteration 83, loss = 0.08637522
Iteration 84, loss = 0.08504772
Iteration 85, loss = 0.08378522
Iteration 86, loss = 0.08264116
Iteration 87, loss = 0.08166050
Iteration 88, loss = 0.08004740
Iteration 89, loss = 0.07863067
Iteration 90, loss = 0.07783857
Iteration 91, loss = 0.07653359
Iteration 92, loss = 0.07582119
Iteration 93, loss = 0.07501950
Iteration 94, loss = 0.07348460
Iteration 95, loss = 0.07310465
Iteration 96, loss = 0.07189302
Iteration 97, loss = 0.07079802
Iteration 98, loss = 0.06975308
Iteration 99, loss = 0.06864095
Iteration 100, loss = 0.06812815
Iteration 101, loss = 0.06711607
Iteration 102, loss = 0.06622928
Iteration 103, loss = 0.06551979
Iteration 104, loss = 0.06455606
Iteration 105, loss = 0.06391040
Iteration 106, loss = 0.06315942
Iteration 107, loss = 0.06222599
Iteration 108, loss = 0.06154876
Iteration 109, loss = 0.06104488
```

```
Iteration 110, loss = 0.06013367
Iteration 111, loss = 0.05945543
Iteration 112, loss = 0.05872138
Iteration 113, loss = 0.05811693
Iteration 114, loss = 0.05712057
Iteration 115, loss = 0.05670672
Iteration 116, loss = 0.05585242
Iteration 117, loss = 0.05528623
Iteration 118, loss = 0.05441967
Iteration 119, loss = 0.05390122
Iteration 120, loss = 0.05368127
Iteration 121, loss = 0.05279129
Iteration 122, loss = 0.05256749
Iteration 123, loss = 0.05177990
Iteration 124, loss = 0.05091970
Iteration 125, loss = 0.05010649
Iteration 126, loss = 0.04990565
Iteration 127, loss = 0.04931173
Iteration 128, loss = 0.04832598
Iteration 129, loss = 0.04806415
Iteration 130, loss = 0.04751965
Iteration 131, loss = 0.04723905
Iteration 132, loss = 0.04631178
Iteration 133, loss = 0.04602997
Iteration 134, loss = 0.04562724
Iteration 135, loss = 0.04478989
Iteration 136, loss = 0.04474587
Iteration 137, loss = 0.04392715
Iteration 138, loss = 0.04328472
Iteration 139, loss = 0.04299102
Iteration 140, loss = 0.04235187
Iteration 141, loss = 0.04210071
Iteration 142, loss = 0.04162809
Iteration 143, loss = 0.04124691
Iteration 144, loss = 0.04096951
Iteration 145, loss = 0.04041905
Iteration 146, loss = 0.04011621
Iteration 147, loss = 0.03982262
Iteration 148, loss = 0.03928668
Iteration 149, loss = 0.03902027
Iteration 150, loss = 0.03892574
Iteration 151, loss = 0.03803100
Iteration 152, loss = 0.03778441
Iteration 153, loss = 0.03795551
Iteration 154, loss = 0.03680585
Iteration 155, loss = 0.03627661
Iteration 156, loss = 0.03604223
Iteration 157, loss = 0.03570784
```

```
Iteration 158, loss = 0.03525340
Iteration 159, loss = 0.03495081
Iteration 160, loss = 0.03468836
Iteration 161, loss = 0.03419625
Iteration 162, loss = 0.03413847
Iteration 163, loss = 0.03382553
Iteration 164, loss = 0.03328388
Iteration 165, loss = 0.03292777
Iteration 166, loss = 0.03260180
Iteration 167, loss = 0.03228899
Iteration 168, loss = 0.03194406
Iteration 169, loss = 0.03176867
Iteration 170, loss = 0.03154225
Iteration 171, loss = 0.03141735
Iteration 172, loss = 0.03082065
Iteration 173, loss = 0.03077605
Iteration 174, loss = 0.03023630
Iteration 175, loss = 0.02979667
Iteration 176, loss = 0.02959311
Iteration 177, loss = 0.02931665
Iteration 178, loss = 0.02906378
Iteration 179, loss = 0.02888344
Iteration 180, loss = 0.02840912
Iteration 181, loss = 0.02812735
Iteration 182, loss = 0.02801478
Iteration 183, loss = 0.02777928
Iteration 184, loss = 0.02752305
Iteration 185, loss = 0.02711602
Iteration 186, loss = 0.02698967
Iteration 187, loss = 0.02678059
Iteration 188, loss = 0.02632374
Iteration 189, loss = 0.02633095
Iteration 190, loss = 0.02614606
Iteration 191, loss = 0.02584296
Iteration 192, loss = 0.02546280
Iteration 193, loss = 0.02513107
Iteration 194, loss = 0.02496014
Iteration 195, loss = 0.02480041
Iteration 196, loss = 0.02450747
Iteration 197, loss = 0.02482043
Iteration 198, loss = 0.02405059
Iteration 199, loss = 0.02413422
Iteration 200, loss = 0.02348181
Iteration 201, loss = 0.02353797
Iteration 202, loss = 0.02312058
Iteration 203, loss = 0.02303956
Iteration 204, loss = 0.02285554
Iteration 205, loss = 0.02259705
```

```
Iteration 206, loss = 0.02274328
Iteration 207, loss = 0.02237112
Iteration 208, loss = 0.02211941
Iteration 209, loss = 0.02175612
Iteration 210, loss = 0.02194779
Iteration 211, loss = 0.02162850
Iteration 212, loss = 0.02132649
Iteration 213, loss = 0.02110956
Iteration 214, loss = 0.02090627
Iteration 215, loss = 0.02065387
Iteration 216, loss = 0.02108443
Iteration 217, loss = 0.02058289
Iteration 218, loss = 0.02005075
Iteration 219, loss = 0.01992468
Iteration 220, loss = 0.01967913
Iteration 221, loss = 0.01949320
Iteration 222, loss = 0.01928949
Iteration 223, loss = 0.01909871
Iteration 224, loss = 0.01900388
Iteration 225, loss = 0.01892172
Iteration 226, loss = 0.01876887
Iteration 227, loss = 0.01847848
Iteration 228, loss = 0.01847849
Iteration 229, loss = 0.01823788
Iteration 230, loss = 0.01799953
Iteration 231, loss = 0.01794103
Iteration 232, loss = 0.01773202
Iteration 233, loss = 0.01773446
Iteration 234, loss = 0.01781041
Iteration 235, loss = 0.01751137
Iteration 236, loss = 0.01716269
Iteration 237, loss = 0.01692673
Iteration 238, loss = 0.01683631
Iteration 239, loss = 0.01674073
Iteration 240, loss = 0.01659543
Iteration 241, loss = 0.01657173
Iteration 242, loss = 0.01625667
Iteration 243, loss = 0.01616079
Iteration 244, loss = 0.01596194
Iteration 245, loss = 0.01585614
Iteration 246, loss = 0.01569894
Iteration 247, loss = 0.01566350
Iteration 248, loss = 0.01546692
Iteration 249, loss = 0.01534869
Iteration 250, loss = 0.01527702
Iteration 251, loss = 0.01504929
Iteration 252, loss = 0.01501614
Iteration 253, loss = 0.01476483
```

```
Iteration 254, loss = 0.01466720
Iteration 255, loss = 0.01465663
Iteration 256, loss = 0.01450938
Iteration 257, loss = 0.01432167
Iteration 258, loss = 0.01421958
Iteration 259, loss = 0.01411839
Iteration 260, loss = 0.01401873
Iteration 261, loss = 0.01387222
Iteration 262, loss = 0.01370514
Iteration 263, loss = 0.01360396
Iteration 264, loss = 0.01355458
Iteration 265, loss = 0.01335760
Iteration 266, loss = 0.01345622
Iteration 267, loss = 0.01329876
Iteration 268, loss = 0.01323062
Iteration 269, loss = 0.01310219
Iteration 270, loss = 0.01287456
Iteration 271, loss = 0.01281371
Iteration 272, loss = 0.01260749
Iteration 273, loss = 0.01263168
Iteration 274, loss = 0.01241933
Iteration 275, loss = 0.01241779
Iteration 276, loss = 0.01239302
Iteration 277, loss = 0.01266028
Iteration 278, loss = 0.01192469
Iteration 279, loss = 0.01245449
Iteration 280, loss = 0.01246345
Iteration 281, loss = 0.01192740
Iteration 282, loss = 0.01170391
Iteration 283, loss = 0.01150491
Iteration 284, loss = 0.01146566
Iteration 285, loss = 0.01141868
Iteration 286, loss = 0.01138836
Iteration 287, loss = 0.01168132
Iteration 288, loss = 0.01125846
Iteration 289, loss = 0.01110728
Iteration 290, loss = 0.01117328
Iteration 291, loss = 0.01082161
Iteration 292, loss = 0.01078207
Iteration 293, loss = 0.01064193
Iteration 294, loss = 0.01055466
Iteration 295, loss = 0.01046398
Iteration 296, loss = 0.01040367
Iteration 297, loss = 0.01043060
Iteration 298, loss = 0.01042094
Iteration 299, loss = 0.01039797
Iteration 300, loss = 0.01019975
```

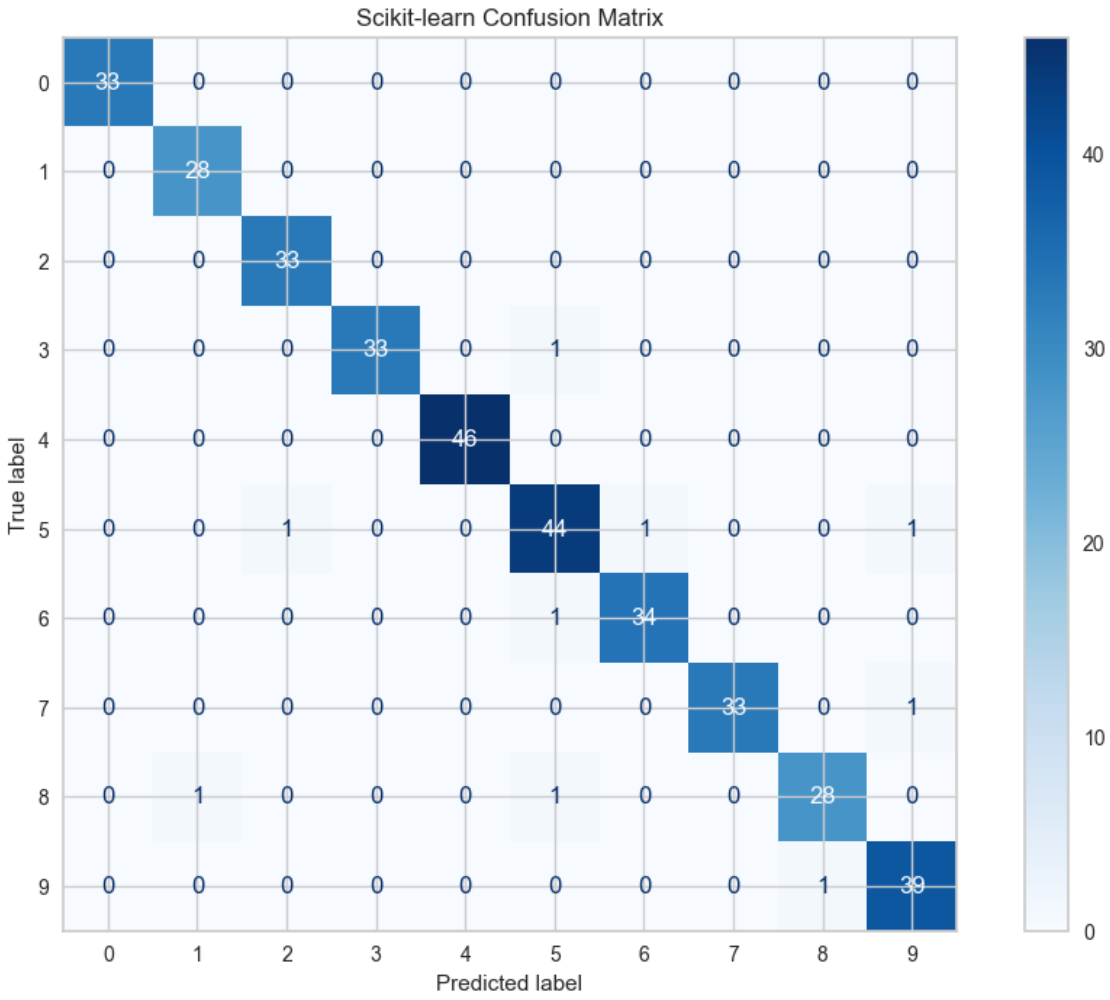
```
C:\Users\blake\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:690:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

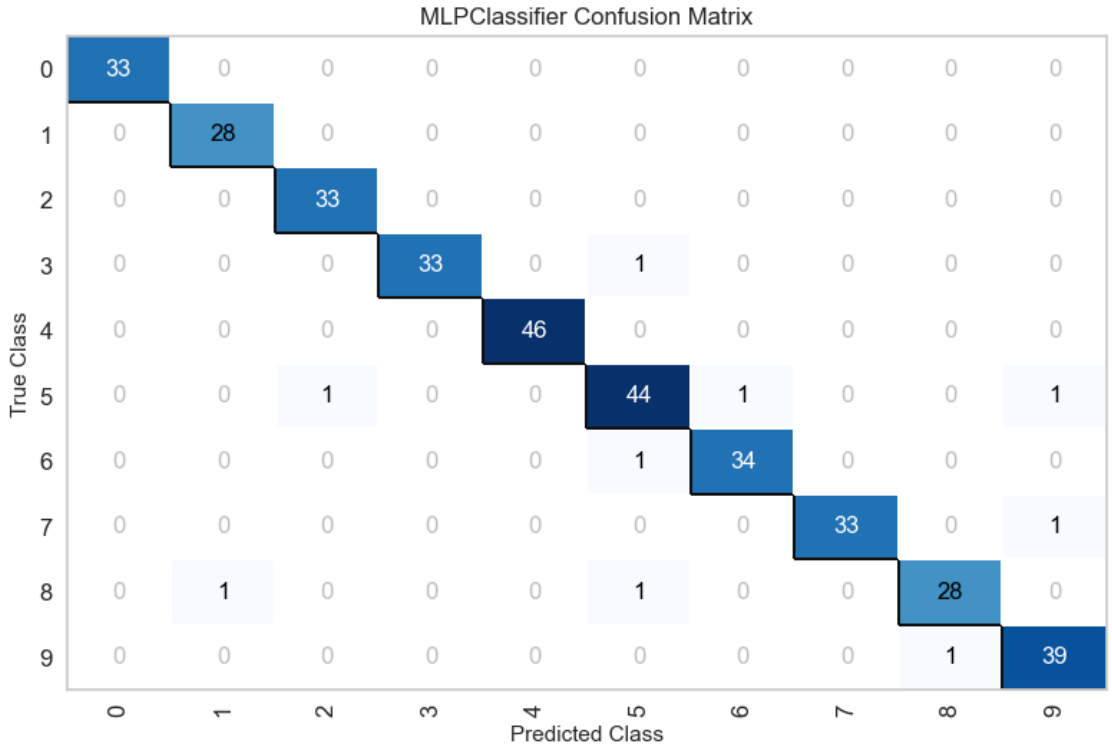
[17]:
```
MLPClassifier(hidden_layer_sizes=(64,), max_iter=300, random_state=42,
              verbose=True)
```

---

**4 - Evaluating the Model on Test Set**

[18]:
```
y_pred = model.predict(X_test)
# sklearn version - confusion matrix to evaluate
cm = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(12, 8))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=digits.
 ↪target_names)
disp.plot(cmap='Blues', values_format='d', ax=ax)
plt.title("Scikit-learn Confusion Matrix")
plt.show()

# yellowbrick version (annotated, precision/recall included)
yb_cm = YBConfusionMatrix(model, classes=digits.target_names, cmap='Blues')
yb_cm.fit(X_train, y_train)
yb_cm.score(X_test, y_test)
yb_cm.show()
```


Scikit-learn Confusion Matrix
True label
Predicted label
0 1 2 3 4 5 6 7 8 9
33 0 0 0 0 0 0 0 0 0
0 28 0 0 0 0 0 0 0 0
0 0 33 0 0 0 0 0 0 0
0 0 0 33 0 1 0 0 0 0
0 0 0 0 46 0 0 0 0 0
0 0 1 0 0 44 1 0 0 1
0 0 0 0 0 1 34 0 0 0
0 0 0 0 0 0 0 33 0 1
0 1 0 0 0 1 0 0 28 0
0 0 0 0 0 0 0 0 1 39
40
30
20
10
0




[18]: `<Axes: title={'center': 'MLPClassifier Confusion Matrix'}, xlabel='Predicted Class', ylabel='True Class'>`

- The Scikit-learn version gives control over layout and aesthetics
- The Yellowbrick matrix, developed by faculty at Georgetown, (from whom I learned data science) adds precision/recall support and integrates smoothly with scikit-learn models — helpful for interpretability in academic and production settings
- Both the scikit-learn and Yellowbrick confusion matrices show strong diagonal dominance, indicating high classification accuracy across all digits.
- Misclassifications are minimal and mostly isolated — e.g., occasional confusion between 5 and 6 or 8 and 9, likely due to shared curvature and loop structure.
- Yellowbrick adds clarity by integrating precision/recall metrics, enhancing interpretability — particularly useful when evaluating how well the model handles digits with similar strokes.

---

**5 - Closing Down the Pub** This notebook illustrated a simple ANN's ability to learn patterns from handwritten digits; instead of intuition (which is what is built into us humans), the machine needs to recognize it through iteration.

The model improved by seeing its own error and adjusting with each pass.

The loss curve, like a learner's arc, bent toward clarity. Each confusion matrix showed not perfection, but progress.

In the spirit of Condorcet, learning is not a leap but a series of steps. And like Locke's blank slate, the model began knowing nothing; yet over many interations it came to recognise form, loop, and line.

Not to say that we can extrapolate to all ANNs from this, but in this case these are both the advantanges and the weaknesses of this simple Artificial Neural Nets.