Comparison of neural network models for digit recognition

Mohammed Khalil

July 7th, 2025

The goal of this project is to compare a few neural network architectures on a standard dataset (the MINST dataset) for digit recognition. We use the following algorithms: 1) single-layer network using the MLPClassifier from the scikit-learn library, 2) multi-layer perceptron (MLP) using keras, and 3) convolutional neural network (CNN). The results of this project could help inform our choices of which algorithms to use for more complicated datasets.

Dataset

The "*Modified National Institute of Standards and Technology*" (MNIST) dataset contains 70,000 handwritten digits (0-9), with each image being 28 x 28 pixels. Each pixel takes a value between 0-255 representing the grayscale intensity.

The dataset can be directly loaded from Keras using:

```
(X_train, y_train), (X_test, y_test) = tensorflow.keras.datasets.mnist.load_data()
```

Where the training dataset contains 60,000 images, while the test dataset contains 10,000 images. However, to reduce the training time, we restrict the training dataset to 20,000 images only.

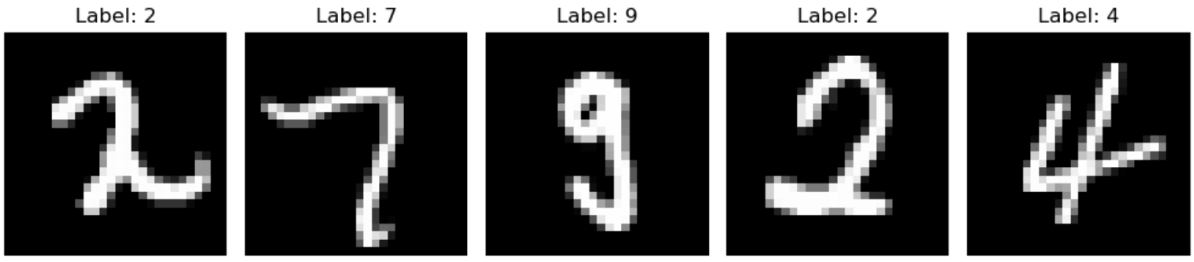
Then, we normalize the data so it takes values between 0-1 to improve the convergence during training.

```
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255
```

For the MLP model in scikit-learn, we reshape each image to be a vector of length 28 * 28 = 784, but when using keras, we add a flatten layer and directly use the 28 x 28 images. For the CNN model, it is useful to reshape the inputs to be matrices of shape 28 x 28 x 1 using np.expand_dims(X_train, -1), and we convert the labels y to binary classes using

```
y_train = tensorflow.keras.utils.to_categorical(y_train, 10)
y_test = tensorflow.keras.utils.to_categorical(y_test, 10)
```

The following are 5 random examples from the dataset



Single-layer network with scikit-learn

We use the MLPClassifier() from scikit-learn with RandomizedSearchCV() to search for best hyper-parameters. To reduce the computational cost, we only use 20000 examples from the train data and pick a few options for the parameter values.

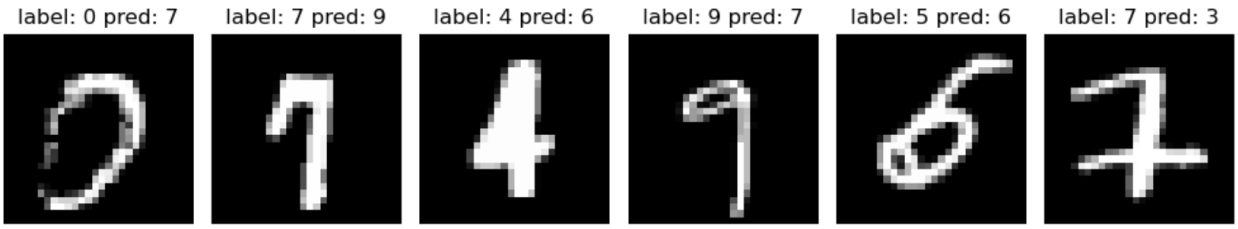
```
parameters = {'hidden_layer_sizes':[100, 200, 300],
              'alpha': [0.001, 0.01],
              'max_iter': [200, 400, 600],
              'learning_rate_init':[0.001, 0.01]}

model = MLPClassifier()
clf = RandomizedSearchCV(estimator=model, param_distributions=parameters, cv=5, n_jobs=-1)
clf.fit(X_train_vec[:20000], y_train[:20000])
```

The best accuracy is found for the parameters:

'max_iter': 600, 'learning_rate_init': 0.001, 'hidden_layer_sizes': 300, 'alpha': 0.01

Rerunning the model with best parameters leads to an accuracy score of **0.983**, and training the model took 20 min CPU time and 50 sec wall time on an Intel Ultra 9 285K processor.

Out of the 10,000 test set, only 167 were misclassified. The following images are a few examples of wrong predictions:



Multi-layer network with keras

Next, we use the Sequential() model from keras with 2 hidden layers structured as follows:

```
model = Sequential([
    Flatten(input_shape=(28, 28)), # Reshapes 2D input (28x28 pixels) into a 1D array
    Dense(256, activation='relu'), # 2 hidden layers with ReLu activation
    Dense(128, activation='relu'),
    Dense(10, activation='softmax'), # output with softmax for classification
])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 256)	200,960
dense_1 (Dense)	(None, 128)	32,896
dense_2 (Dense)	(None, 10)	1,290

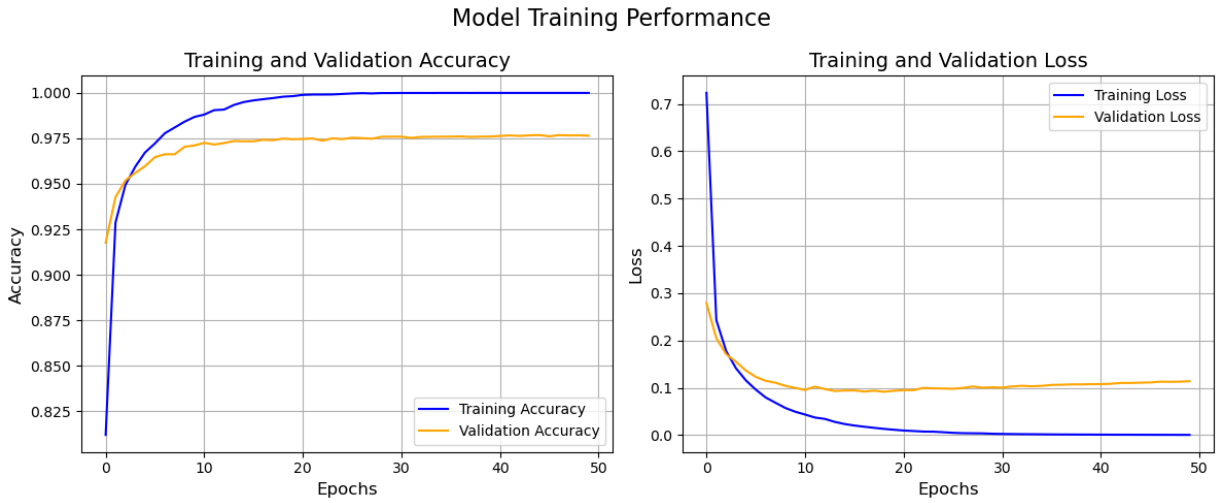
Total params: 235,146 (918.54 KB)

Trainable params: 235,146 (918.54 KB)

Non-trainable params: 0 (0.00 B)

We compiled the model using the Adam optimizer with learning_rate 0.001, and the sparse_categorical_ crossentropy loss function. We trained the model over 50 epochs, with batch_size 1000.

The accuracy score of the resulting model is **0.977** after training wall time of only 20 sec. The following plots show the accuracy and loss after each epoch:



Convolutional neural network

For the CNN architecture, we use two convolutional layers followed by one dense layer. In between, we use MaxPooling2D() to downsample the images and Dropout() to prevent overfit.

```
model = Sequential(
    [
        Input(shape=input_shape),
        Conv2D(32, kernel_size=(3, 3), activation="relu"),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(64, kernel_size=(3, 3), activation="relu"),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dropout(0.5),
        Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_1 (Flatten)	(None, 1600)	0
dropout (Dropout)	(None, 1600)	0
dense_3 (Dense)	(None, 10)	16,010

```
Total params: 34,826 (136.04 KB)
Trainable params: 34,826 (136.04 KB)
Non-trainable params: 0 (0.00 B)
```

The accuracy of the model is **0.993** and the training time took 40 sec.

Key findings and conclusions

The following table summarizes the results of the three models considered in this project:

Model	Accuracy score	Training time (wall time)
Single-layer network with scikit-learn	0.983	50 sec
Multi-layer network with keras	0.977	20 sec
Convolutional neural network	0.993	40 sec

We conclude that the CNN model leads to the best accuracy, but is computationally more expensive than training a single instance of an MLP model in Keras, which is significantly faster than using scikit-learn. However, it is often easier to choose the parameters of the CNN model than running a GridSearch over many options with the MLP classifier. Therefore, we recommend using CNN for image classification.

Next steps

1. Use RandomizedSearchCV() with more options for the parameters to optimize both the MLPClassifier() and the keras Sequential() models.
2. Optimize the parameters of the convolutional layers of the CNN model.
3. Investigate other neural network architectures.