Handwritten Digit Recognition

Spandan Ghosh, Shoumik Bhattacharya, Arghya Mukherjee

Abstract

Recently, handwritten-digit recognition has been a subject of considerable interest among machine-learning researchers primarily due to the evolution of various algorithms in the areas of Machine Learning, Deep Learning and Computer Vision. In this report, we compare the outcomes of Stochastic Gradient Descent, Decision Tree and Logistic Regression — three of the most widely used machine-learning algorithms. Using these, we were able to get the accuracy of 86.87% using Stochastic Gradient Descent, 87.68% using Decision Tree and 92.38% using Logistic Regression

Index Terms — Machine Learning, Handwritten digit recognition, Stochastic Gradient Descent, Decision Tree, Logistic Regression

1. Introduction

Handwritten-digit recognition, a topic of research for decades, refers to the ability of computer systems to identify handwritten digits and characters from varied sources like letters, documents, images, emails, and so on. Signature verification, postal-address interpretation from envelopes, and bank-check processing continue to be some of the research areas in this domain. Various machine-learning based classification techniques [1] like K-Nearest Neighbors, SVM Classifier, Random Forest Classifier, etc. have been devised and used for this recognition purpose, but these approaches, even though having accuracies of over 95%, are not enough to serve the purposes of many real-world applications. For instance, if a letter with recipient name “Kamal” is posted, and our classification system misinterprets it as “Amal”, there lies the possibilities of the letter being returned back to the sender, or of the letter being delivered to an unintended person “Amal”, residing close to “Kamal”. In this context, the use of deep-learning techniques comes into picture. The last decade has witnessed the employment of deep learning [3] for image processing, handwritten character and digit recognition, object detection, etc. Also, several machine-learning tools (like Scikit-learn, Scipy-Ndimage etc.) and deep-learning tools (like PyBrain, Keras, Theano, Tensorflow, TFLearn etc.) have been developed. These tools make the applications significantly more robust and accurate; artificial-neural networks, closely resembling the human brain, act as a key ingredient in handwritten-digit identification and image processing in general.

2. Dataset

Our dataset MNIST [2], a subset of the much larger NIST dataset, comprises images of 70,000 handwritten digits, divided into 60,000 training instances and 10,000 testing instances. An image in the MNIST dataset is present in the form of a two-dimensional array consisting of 28 x 28 values along with its label. The entire data is distributed across four files, namely: 1. *train-images-idx3-ubyte* containing training-set images, 2. *train-labels-idx1-ubyte* containing training-set labels, 3. *t10k-images-idx3-ubyte* containing testing-set images, and 4. *t10k-labels-idx1-ubyte* containing testing-set labels.

Data in the training-set-images file (*train-images-idx3-ubyte*) has the following representation [2]:

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

Data in the training-set-labels file (train-labels-idx1-ubyte) has the following representation [2]:

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	10000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
.....			
xxxx	unsigned byte	??	label

Data in the testing-set-images file (t10k-images-idx3-ubyte) has the following representation [2]:

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	10000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

Data in the testing-set-labels file (t10k-labels-idx1-ubyte) has the following representation [2]:

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	10000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
.....			
xxxx	unsigned byte	??	label

Pixels are organized in row-major order. Pixel values range from 0 to 255; 0 stands for background (white), 255 stands for foreground (black). The labels values are integers from 0 to 9.

3. Literature Review

Anuj Dutt[9] has evaluated the performance of SVM, KNN, RFC . Yann LeCun[2],[3] has described the MNIST in great detail. He has also done a comprehensive review of performance of multiple classification algorithms. Ali, S., Shaukat[5] & Issam Bendib[6] demonstrates improved CNN architectures.

4. Classification Using Machine Learning Algorithms

We have used the following three machine-learning based classifiers for our baseline model: A. Stochastic Gradient Descent, B. Decision Tree, C. Logistic Regression

A. Stochastic Gradient Descent

Below is the algorithm for Stochastic Gradient Descent:

$$\text{repeat until convergence: } \{$$
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j} := 0...\text{n}$$
$$\}$$

Here, $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$, is the hypothesis. $x^{(i)}$ represents the feature vector (of length m + 1) and $y^{(i)}$ represents the target value corresponding to the i^{th} training instance. α is a hyperparameter indicating the learning rate, and there are altogether n training instances. Our task is to find an optimal value of each θ_j that minimizes the cost function

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Figure 1 shows a sample predicted output using the above algorithm, and figure 2 depicts the corresponding confusion matrix that resulted from our validation data.

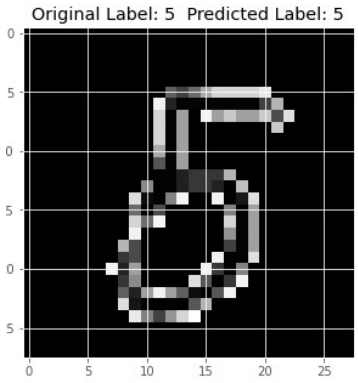

Fig 1: A sample predicted output using stochastic gradient descent

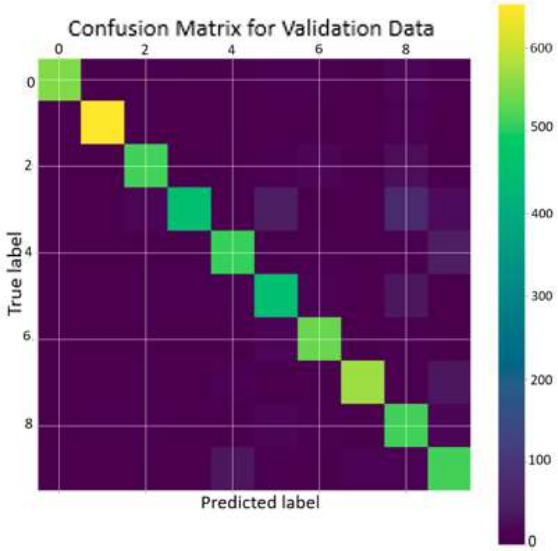

Fig 2: Confusion matrix for stochastic gradient descent

Further, an accuracy of 86.87% was achieved on the test data using stochastic gradient descent.

B. Decision Tree

A decision tree, by sorting the given instances down from the root to some leaf node, provide a classification of the instances. Each tree node features a test of some attribute of the instance, and each branch leaving that node corresponds to one of the feasible values of this attribute. Attribute selection at each stage is usually performed using a metric called information gain, using which there is a bias of preferring those attributes that have many distinct values. This bias may give rise to a problem called overfitting of data, which can be overcome using gain ratio as the attribute-section parameter.

Figure 3[7] portrays a sample decision tree for deciding whether to buy a car or not based on color, model and mileage attributes.

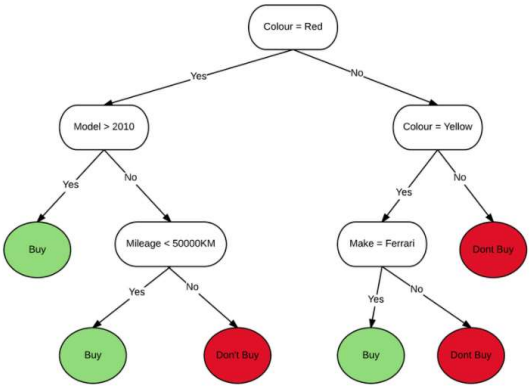

Fig 3: A sample decision tree

Figure 4 depicts the confusion matrix that resulted from running the decision-tree model on our validation data. In this case, a slightly higher accuracy of 87.68% was achieved on the test data.

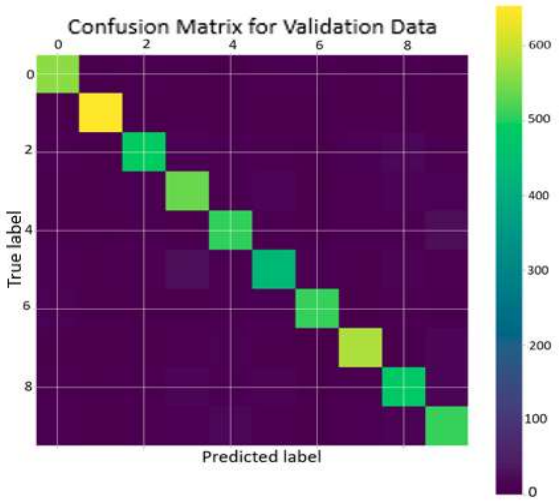

Fig 4: Confusion matrix for decision-tree classifier

C. Logistic Regression

In logistic regression, we derive a hypothesis function which provides the probability of an instance belonging to a certain class.

The formula for softmax coding used for multinomial Logistic Regression is given by

$$\Pr(Y = k|X = x) = \frac{e^{\beta_{k0}+\beta_{k1}x_1+\cdots+\beta_{kp}x_p}}{\sum_{l=1}^{K} e^{\beta_{l0}+\beta_{l1}x_1+\cdots+\beta_{lp}x_p}}.$$

where K denotes the number of classes and Pr(Y=k|X=x) denotes the probability of x belonging to class k

Figure 5 depicts the confusion matrix for validation data. In this case, an accuracy of 92.38% was achieved on the test data.

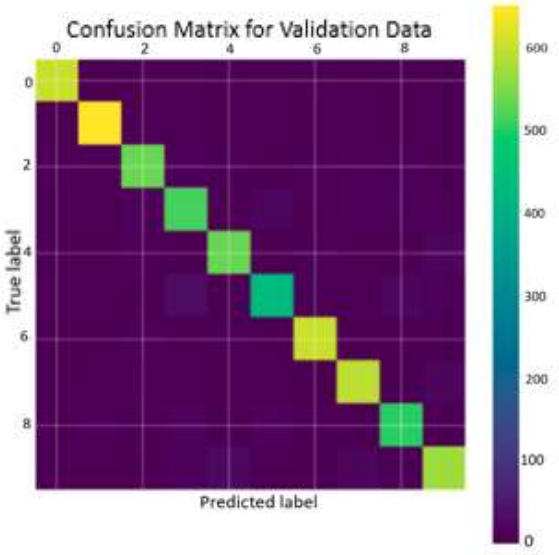

Fig 5: Confusion matrix for logistic-regression classifier

D. Support Vector Machine (SVM)

Support-vector machines are supervised machine-learning algorithms capable of performing both classification and regression analysis. SVMs construct one or more hyperplanes in n-dimensional space; the best separation is obtained by the hyperplane having the largest distance to the nearest training-data point of any class (known as functional margin). Given an understandable margin of dissociation between classes, SVM performs reasonably better than most other machine-learning algorithms. Although being productive for data instances with large number of features, it is not suitable for large datasets.

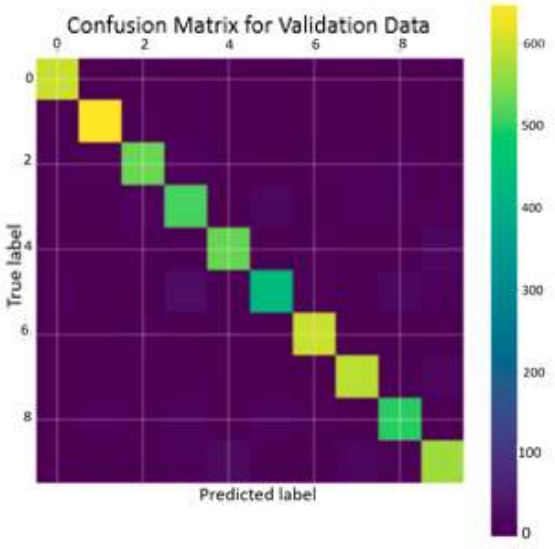

Fig 6: Confusion matrix for support-vector-machine classifier

Figure 6 depicts the confusion matrix for validation data. In this case, an accuracy of 97.87% was obtained on the test data.

E. Naive Bayes

Naive Bayes is a probabilistic machine-learning model, based on the Bayes theorem, that is being used for classification task Using Bayes theorem, we can find the probability of occurrence of an event A, given that even B has occurred.

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

A, B = events
$P(A|B)$ = probability of A given B is true
$P(B|A)$ = probability of B given A is true
$P(A), P(B)$ = the independent probabilities of A and B

In this model, it is assumed that the all features are independent. Nave Bayes classifier can be of three types, namely, multinomial naive Bayes, Bernoulli naive Bayes and Gaussian naive Bayes. Here, we are making use of multinomial naive Bayes.

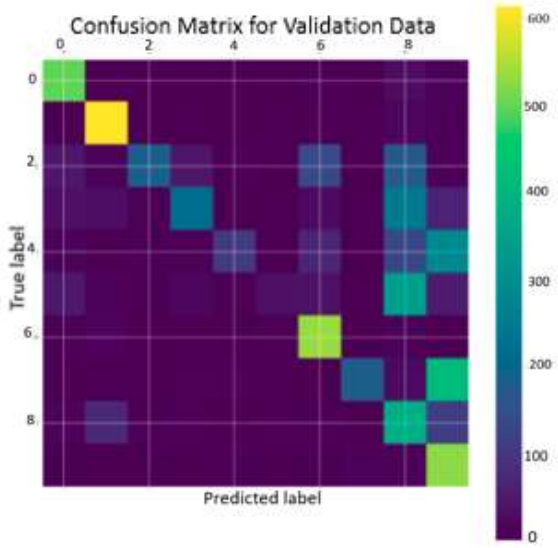

Fig 7: Confusion matrix for naive-Bayes classifier

Figure 7 shows the confusion matrix for validation data. In this case, an accuracy of 55.49% was attained on the test data.

The fact that naïve-Bayes algorithm results in poor predictions here is revealed by the presence of multiple spurious squares (squares other than the ones appearing along the principal diagonal of the matrix) in the confusion matrix of Figure 7.

F. K-Nearest Neighbors (KNN)

K-Nearest Neighbor is a non-parametric, supervised machine learning algorithm that assumes each training instance to be a point in n-dimensional space; here, n is the number of features.

Algorithm:

1. Initialize K to a preferred number of neighbors.
2. For each instance in the data
3. Determine the distance metric between the query instance and the current instance.
4. Add distance and index of the example to an ordered collection.
5. Sort the ordered collection of distances and indices in ascending order of the distances.

6. Select first K entries from the collection thus sorted.
7. Obtain labels of the chosen K entries.
8. Return mode of the K labels.

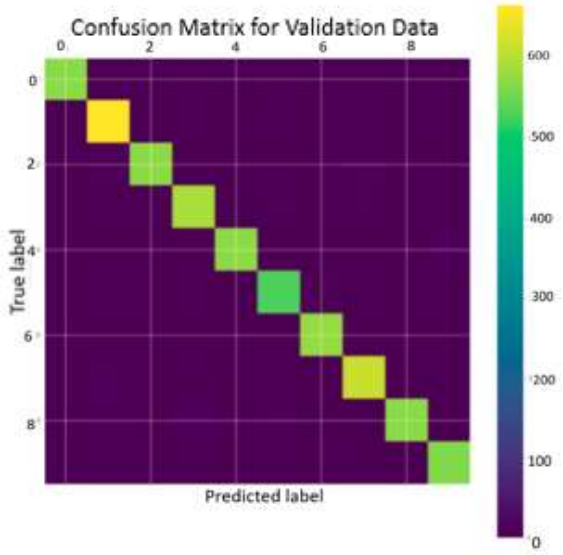

Fig 8: Confusion matrix for k-nearest-neighbors classifier

Figure 8 depicts the confusion matrix for validation data. In this case, an accuracy of 96.82% was observed on the test data.

KNN is simple, easy to implement, and is suitable for both regression and classification. However, being an instance-based learning method, it suffers from the problem of high cost of classifying new instances. Also, a difficulty arises if many irrelevant attributes are present, something often referred to as the curse of dimensionality.

G. Random Forest

Random forest classifier is an ensemble learning method suitable for both classification and regression tasks; it operates by constructing a large set of decision trees at training time. The central idea is that a multitude of relatively uncorrelated models (trees) operating as a committee will outperform any of the constituent models taken individually. Random forest has the advantage of making use of parallelism resulting in faster computation time. Moreover, it is suitable for high dimensional data and is less sensitive to outliers.

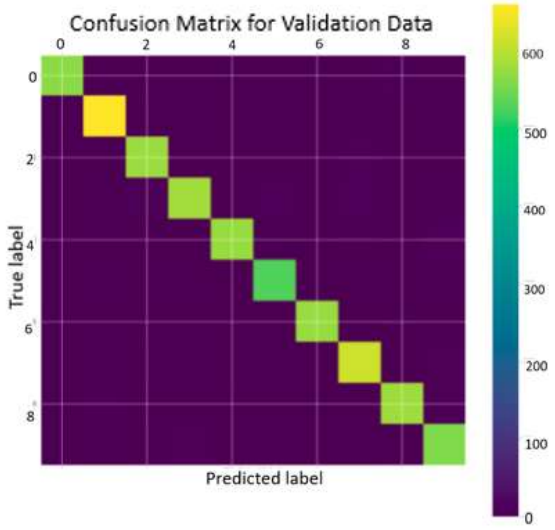

Fig 9: Confusion matrix for random-forest classifier

Figure 9 depicts the confusion matrix for validation data. In this case, an accuracy of 96.83% was attained on the test data.

H. Neural Network

Artificial Neural Networks (ANN) are computational algorithms, intended for simulating the behavior of human brain comprising numerous neurons. An ANN typically consists of three layers: an input layer, one or more hidden layers, and an output layer. Neurons, the building block of ANNs, take inputs, multiply each input by weights, and add the weighted inputs thus obtained along with a bias. The resulting sums are finally passed through an activation function (which is often a sigmoid function).

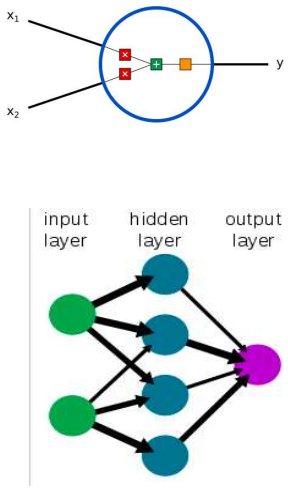

Fig 10: Sample neuron and neural-network[8]

Neural networks have several desirable properties, such as:

a) The input features can be highly correlated or independent,

b) The target function may be discrete valued, continues valued, or even a vector of several discrete or continuous valued features,

c) The training instances may be noisy.

However, longer training times are often demanded by this model.

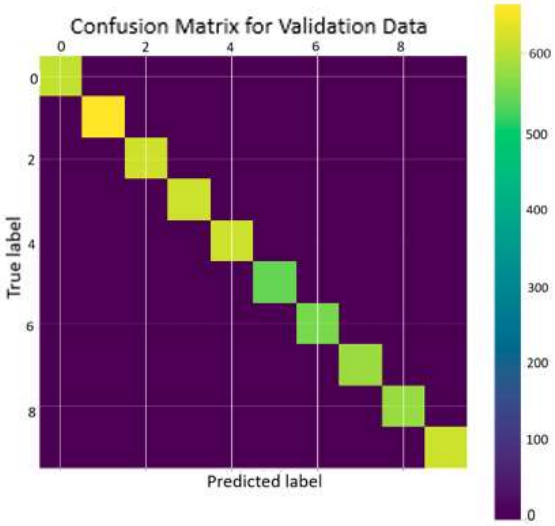

Fig 11: Confusion matrix for neural-network classifier

Figure 11 shows the confusion matrix for validation data. In this case, an accuracy of 99.24% was obtained on the test data.

5. Performance Metrics

Comparison of Different Performance Metrics on Validation Data			
Model	**Accuracy**	**Precision**	**Recall**
Stochastic Gradient Descent	85.28%	86.71%	84.81%
Decision Tree	86.97%	86.71%	84.81%
Logistic Regression	91.60%	91.45%	91.37%
Support Vector Machine	97.70%	97.69%	97.70%
Naive Bayes	54.53%	67.04%	54.82%
K-nearest Neighbors	97.35%	97.40%	97.33%
Random Forest	96.90%	96.90%	96.89%
Neural Network	99.85%	99.85%	99.85%

Comparison of Different Performance Metrics on Test Data			
Model	**Accuracy**	**Precision**	**Recall**
Stochastic Gradient Descent	86.87%	88.20%	86.58%
Decision Tree	87.68%	87.48%	87.52%
Logistic Regression	92.38%	92.30%	92.26%
Support Vector Machine	97.87%	97.87%	97.86%
Naive Bayes	55.49%	69.07%	54.75%
K-nearest Neighbors	96.82%	96.87%	96.79%
Random Forest	96.83%	96.82%	96.80%
Neural Network	99.24%	99.24%	99.24%

Comparison of Training Times	
Model	**Training Time (secs.)**
Stochastic Gradient Descent	154.96
Decision Tree	21.83
Logistic Regression	33.40
Support Vector Machine	257.90
Naive Bayes	.93
K-nearest Neighbors	.016
Random Forest	49.94
Neural Network	483.05

Fig 12: Comparative analysis of various performance metrics

The first two tables clearly reveal that naïve Bayes, with an accuracy of around 55%, performed poorly among all the machine-learning models. On the other hand, neural network achieved the best performance, at the expense significant time and complexity overheads. Logistic regression, support vector machine, k-nearest neighbors, and random forest are some other algorithms that achieved accuracies of over 90%.

6. Conclusion

Considering only the baseline models (i.e. models A, B and C), error rate of Logistic Regression is minimum. However, if all the eight models are taken into account, the best result is obtained using Neural Network. Therefore, it is the most suitable predictor. However, as mentioned before, a perfect accuracy of 100% is demanded by several real-world applications, something that necessitates the search for an even better performing machine-learning model.

7. Code & Data Links

https://drive.google.com/drive/folders/1XGcNyb_Nv2y4LB1-g6hpa2uIAiMuVXCJ?usp=sharing

8. Contribution of each member

Spandan Ghosh: Studied various models and prepared the baseline models, performed data analysis, and prepared a framework for implementation of other models.

Shoumik Bhattacharya: Implemented the baseline models, identified and studied the models that can be applied to the problem, and designed the presentation by studying various research papers.

Arghya Mukherjee: Implemented baseline, performed the literature survey, and prepared the detailed project report. Also performed an analysis of the results.

References

[1] Mahnoor Javed, Nov 22, 2020, “The Best Machine Learning Algorithm for Handwritten Digits Recognition”.

[2] Yann LeCun, “Comparison of Learning Algorithms for Handwritten Digit Recognition”.

[3] Yann LeCun, “Learning algorithms for classification: A comparison on Handwritten digit recognition”.

[4] Aditi Jain, Jul 9, 2020, “Deep Learning — Handwritten Digit Recognition using Python and MNIST”.

[5] Ali, S., Shaukat, Z., Azeem, M. et al, SN Appl. Sci. 1, 1125 (2019), “An efficient and improved scheme for handwritten digit recognition based on convolutional neural network”.

[6] Issam Bendib, Abdeljalil Gattal, Guedri Marouane, October 2020. “Handwritten Digit Recognition Using Deep CNN”.

[7] Victor Zhou, Mar 6, 2019, “Machine Learning for Beginners: An Introduction to Neural Networks”.

[8] Anuj Dutt, AashiDutt, July 7, 2017, “Handwritten Digit Recognition Using Deep Learning”.

[9] Jason Brownlee, May 8, 2019. In Deep Learning for Computer Vision, “How to Develop a CNN for MNIST Handwritten Digit Classification”