Image Classification using a Convolutional Neural Network

Sarthak Sachdeva
College of Computer and Information Science
Northeastern University
Boston, MA

Abstract

Image classification consists of assigning to an input image a label from a fixed set of categories. It involves the analysing of numerical properties of various features from an image and organising them. Despite its simplicity, image classification is one of the core problems and projects of Computer Vision, having a large variety of practical applications such as remote sensing, segmentation, and applications that involve image recognition. In order to classify a set of data into different classes or categories, the relationship between the data and the classes into which they are classified must be well understood. This can be achieved with Machine Learning, which enables computers to learn, without the specific content of the learning being programmed, only the method of its acquisition. In this project, we used Neural Networks, which are derived from study into biological nervous systems, that are composed of many simple nonlinear computational elements (called neurons). These neurons are connected by links with variable weights. Neural Networking allows the simultaneous exploration of multiple hypotheses in parallel, resulting in high computational efficiency. Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. CNN's differ in that they make the explicit assumption that its inputs are images, allowing for the encoding of certain properties into the architecture.

Introduction

Our day-to-day functioning make vision and our brain's organising of it seem easy. We can very easily distinguish and tell apart a dog and a horse, recognise each others' faces, and read signs. But we didn't always have such an easy time: we learned through trial and error with the correction of our parents and others in our environment in order to internalise and generate robust symbol-recognising capabilities so that trial-and-error is minimised and virtually eliminated as we develop and mature. In the last few years the field of machine learning has made tremendous progress by attempting to parallel and computationally recreate such processes. In particular, we've found that a model called a Convolutional Neural Network can achieve reasonable performance on otherwise difficult visual recognition tasks. Neural Networks provide a great degree of robustness or fault tolerance because of the many processing neurons, each of which is responsible for a small portion of the task. These nets perform a variety of tasks, one of which is known as supervised learning. This is a decision-making process that requires the net to identify the class or category which best represents an input pattern.

In this paper, we used a dataset of handwritten images of numerical digits and predicted the class to which the image belongs. We fed about 50,000 images to our machine, with a label associated with every digit, each corresponding to the number it represents. Our images have multiple variations of the same digit so as to have a diverse set of training data. This training data goes through various layers of the Convolutional Network, where filters (or weights) are multiplied with each pixel in the input image. For each position of the filter, the dot-product is calculated between the filter and the image pixels under the filter, which results in a single pixel in the output image. So moving the filter across the entire input image results in a new image being generated. This happens for two convolutional layers before it reaches the fully-connected layer. The 3D feature

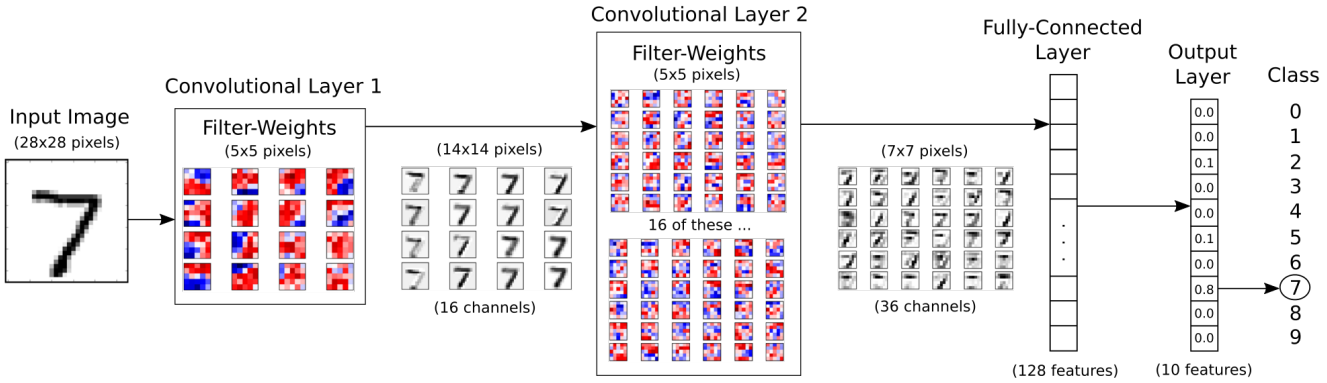


Fig.1 Overview of the Classification Process

map of the image is squashed with a sigmoid after which we use Softmax regression.

To assign probabilities to an object being one of several different things, Softmax regression is used, giving us a list of values between 0 and 1 that add up to 1, representing the full range of probabilistic probabilities. First we added up the evidence of our input being in certain classes, and then we converted that evidence into a probability distribution using Softmax. This distribution corresponds to each of the 10 classes for the images. Fig. 1 depicts an overview of the classification process.

The remainder of this paper is organised as follows. Section 2 contains the methodology, information about the dataset, libraries and the techniques being used. Section 3 contains the details about implementation of the model. Section 4 contains the model results. Section 5 presents the analysis of our models and Section 6 finally concludes the paper.

Methodology

In this paper, we discuss our machine learning algorithm's ability to successfully recognise and classify an image after being trained with over 50,000 images. Our goal is to predict the correct class for a particular image. Hence, in this section we present our approach to collect the data and discuss the modelling techniques used.

Data Source

For this project, we have used the MNIST dataset[1]. The data is split into three parts: 55,000 data points of training data, 10,000 points of test data, and 5,000 points of validation data. For this project, we are just using the training and the test data. Every MNIST data point has two parts: an image of a handwritten digit and a corresponding label. Each image is 28 pixels by 28 pixels which we have downsampled to 14 pixels by 14 pixels using 2x2 max-pooling.

Libraries Used

In order to successfully carry out the classification, we use various libraries :

- *Matplotlib*[2] is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.
- *NumPy*[3] is a library that adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- *scikit-learn*[4] is a machine learning library that features various classification, regression and clustering algorithms. It is designed to interoperate with NumPy.
- *TensorFlow*[5] is a library for machine learning across a range of tasks. It is used for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning which humans use. It is the most important library we have used in this project.

Modeling Techniques

As mentioned in the proposal, at first, we attempted the implementation of k-Nearest Neighbour (k-NN) classifier to classify the images. However, this model yielded poor results with low accuracy and took a long time for the computation in comparison with CNN. We, therefore, have attempted to implement Neural Networks in order to classify our input image.

Neural Networks : A neural network (NN) [6] is an information processing paradigm which is inspired by the way biological nervous systems, such as the brain, process information.The key element of this paradigm is the structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. Neural Networks are useful because :

- They have the ability to learn how to do tasks based on the data given for training or initial experience.

- They work well with identifying and extracting patterns from complex data for classification tasks especially when training data is small.

In order to be more accurate, we have used a Convolutional Neural Network for this project. They work by moving small filters across the input image. This means the filters are re-used for recognizing patterns throughout the entire input image, making CNNs much more powerful than Fully-Connected networks with the same number of variables. In a CNN, each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. A simple *ConvNet* is a sequence of three layers : Convolutional Layer, Pooling Layer and Fully-Connected Layer. Note that each convolutional block comprises of down-sampling and ReLU. Down sampling is done by a process known as max-pooling[7]. It is a sample-based discretization process that helps us to reduce the dimensionality of an image, allowing for assumptions to be made about features contained in the sub-regions binned. This is done in part to help over-fitting by providing an abstracted form of the representation. Also, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation invariance to the internal representation. Max pooling is done by applying a *max filter* to non-overlapping sub-regions of the initial representation.

In order to minimize our loss function, we implemented the gradient descent [8] algorithm with back propagation using Sigmoid [9] and ReLU [10] functions as the activation functions.

***function**: inputs ,weights, iterations$_{max}$, learn$_{rate}$*
Step 1: Initialize weights
Step 2:
. for (i in 1 to iterations$_{max}$)
. HiddenLayers = Activation(inputs, weights)
. Output$_i$ = ForwardPropagation(HiddenLayers , Network)
. BackPropagateError(HiddenLayers , Output$_i$)
. Update Weights$_i$ (HiddenLayers , Output$_i$, learn$_{rate}$)
Step 3: return weights

TensorFlow

TensorFlow is extremely efficient and even more so relative to NumPy because it knows the entire computation graph that must be executed, while NumPy only knows the computation of a single mathematical operation at a time. It can also automatically calculate the gradients that are needed to optimize the variables of the graph so as to make the model perform better. This is because the graph is a combination of simple mathematical expressions so the gradient of the entire graph can be calculated using the chain-rule for derivatives.

A TensorFlow graph consists of - placeholder variables used for inputting data to the graph, variables that are going to be optimized so as to make the convolutional network perform better, mathematical formulas for the convolutional network, a cost measure that can be used to guide the optimization of the variables and an optimization method which updates the variables.

Implementation

The dataset is loaded with all the images and class-labels associated with them. Each class-label is one-hot encoded, which converts our dataset to binary values 1 or 0 that make it easier to process.

The first convolutional layer takes an image as input and creates a number of different filters. These filters are matrices of pixels values that it learns from the weights via matrix multiplication (Fig 2). The input to a convolutional layer is a 4-dimensional tensor with the dimensions being image number, x & y axes of each image and channels of each image. (The channels may either be colour-channels or filter-channels if the input is provided from a prior convolutional layer).

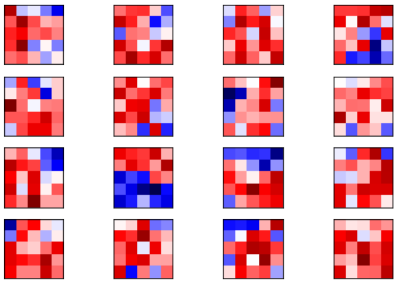


Fig. 2 Filter weights for the first convolutional layer

The output for this is a 4-dimensional tensor with similar dimensions. Each image is 14 pixels wide and 14 pixels high, and there are 16 different channels, one channel for each of the filters. We use 2x2 max pooling to down-sample the resolution of these images (Fig 3).

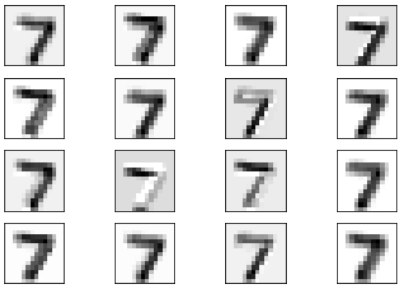


Fig. 3 Result of the First Convolutional Layer

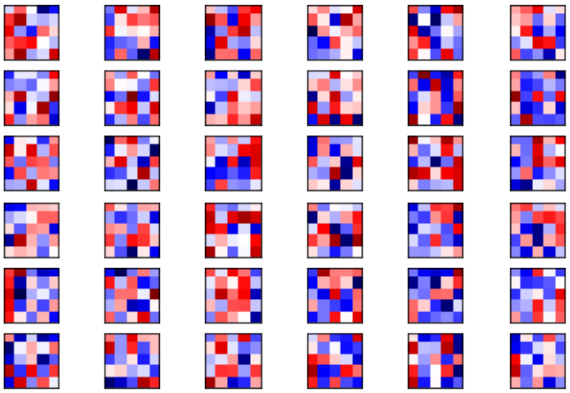


Fig 4. The filter weights for the second convolutional layer

The second convolutional layer then takes the output from the first layer as input. This layer has a filter weight for each of its 16 input channels. (Fig 4). The number of

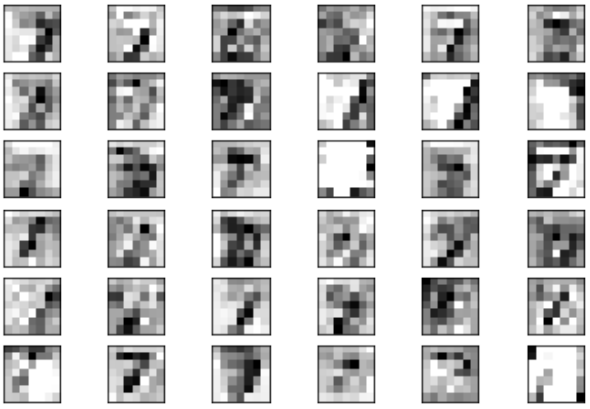


Fig.5 Result of the second convolutional layer

input channels corresponds to the number of filters in the first convolutional layer. This time the output has each image having a width and height of 7 pixels. (Fig 5) This is again done using 2x2 max pooling.

We add two fully-connected layers after the convolution layers. The convolutional layers produce an output tensor with 4 dimensions. In order to process this output, we reduce the 4-dimensional tensor to 2-dimensions which can be used as input to the fully-connected layer. This 2-dimensional vector is further reduced to a scalar quantity which is our probability.

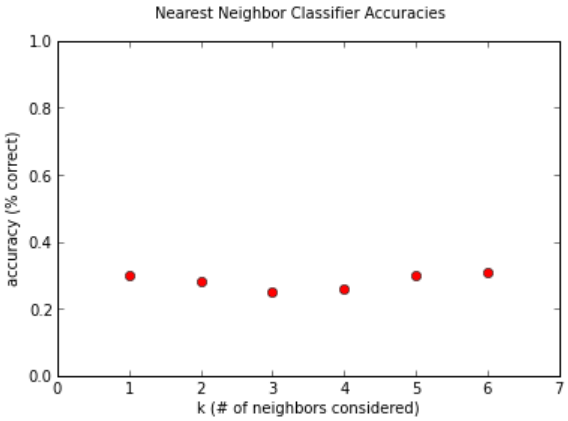
The first fully connected layer takes the output from the second convolutional layer as input in the form of the 2D vector. The output of this layer is used as input to the final fully-connected layer that estimates how likely it is that the input image belongs to each of the 10 classes. However, these estimates are somewhat rough and difficult to interpret because the numbers may be very small or large. In order to fix this, we normalize them so that each element is limited between zero and one and the 10 elements sum to one. This is calculated using the softmax function, which outputs a probability. This probability is mapped to one of 10 classes.

The mapping is done by an optimizer and a loss function. This loss function is known as cross-entropy. Cross-entropy measures error at the softmax layer. This outputs an error between the real and the expected value. The error is minimised by gradient descent, which determines the direction we want to update our weights in, which in this case are filter values. In order to use gradient descent, we use chain rule. Here, we take the derivative of each layer, *back-propagating* our loss and updating our weights recursively.

Once the back propagation is done and we have minimised our error, we get a probability distribution that finally allows us carry on our mapping and determine the class of the image.

Results

At first, we used k-NN classification as it is among the simplest of all machine learning algorithms. However, it wasn't very accurate (about 38%). The implementation was



terribly inefficient. For instance, if you have N training points in your data set, computing the distances to each point will take O(N) time, and then getting the first k values by sorting the data set based on distance will take another O(N*log*N) time, so you'll spend O(NN) time for *each* prediction you want to make. This was certainly suboptimal. The results for k-NN accuracies for varying values of k are represented below in the figure above.

On the other hand, using Neural Networks yielded a classification accuracy of 98%, which was significantly high and more optimal. Upon the first iteration, the accuracy on the test-set was very low(7.6%) because the model variables had only been initialized and not optimized at all, so it just classified the images randomly.
After one optimization iteration, the classification accuracy did not improve much either, as the learning-rate for the optimizer was set very low.
However, with increasing number of optimization iterations, the model remarkably improved the classification accuracy. A tabular representation of the results with respect to the number of iterations is underneath :

No. of Iterations	1	100	1000	10,000
Percentage Accuracy	10.1%	62.8%	93.2%	98.7%

In order to further depict the performance of our classification model, we use a confusion matrix[11], which is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one. Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class (or vice versa). [12]

```
[[ 973    0    1    0    0    1    1    1    3    0]
 [   0 1128    3    0    0    0    1    0    3    0]
 [   2    1 1017    1    1    0    0    2    8    0]
 [   1    0    0 1002    0    4    0    0    3    0]
 [   0    0    1    0  969    0    1    1    3    7]
 [   1    0    1    4    0  884    1    1    0    0]
 [   7    2    0    0    2    6  939    0    2    0]
 [   0    1   12    2    0    0    0 1005    3    5]
 [   3    0    2    2    1    1    0    1  963    1]
 [   1    3    0    6    4    2    0    3    3  987]]
```

Fig. Confusion Matrix for our classification model

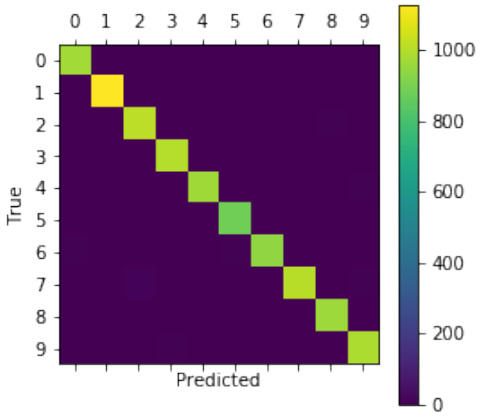
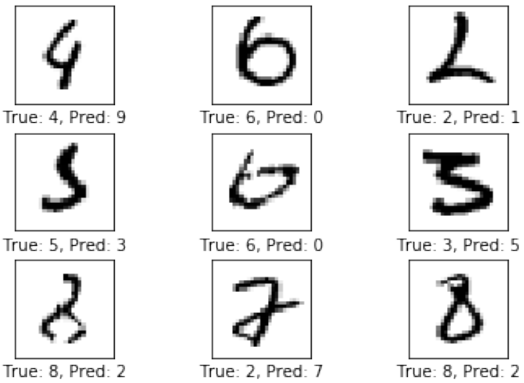


Fig. A more visual representation of our Confusion Matrix. Note that the classes for the digits are illustrated on top and on the left.

Model Analysis & Conclusion

In the previous section, we discovered that Neural Networks are more adept than k-nearest neighbour algorithm when it comes to classification. Although, k-NN yields smooth decision regions, the learning capacity of neural networks is better.

An accuracy measure of 98.7% by CNNs is extremely efficient. The 1.3% error exists because people are likely to write digits differently. Our model also provided us with an illustration of example errors. The figure for the same is below :



Future work

For this project, we implemented two modelling techniques for image classification but we concluded that Convolutional Neural Networks are relatively better to carry out the task.

In future works we would like to implement more complex modelling techniques such as a combination of Neural Networks and k-NN, Genetic K-means, weighted k-NN and Neuro Fuzzy algorithms.

References

[1] LeCun, Yann; Corinna Cortes; Christopher J.C. Burges. "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges″

[2] Whitaker, Jeffrey. "The Matplotlib Basemap Toolkit User's Guide (v. 1.0.5)". *Matplotlib Basemap Toolkit documentation*.

[3] David Ascher; Paul F. Dubois; Konrad Hinsen; Jim Hugunin; Travis Oliphant (1999). "Numerical Python”

[4] Fabian Pedregosa; Gaël Varoquaux; Alexandre Gramfort; Vincent Michel; Bertrand Thirion; Olivier Grisel; Mathieu Blondel; Peter Prettenhofer; Ron Weiss; Vincent Dubourg; Jake Vanderplas; Alexandre Passos; David Cournapeau (2011). "Scik-

it-learn: Machine Learning in Python". *Journal of Machine Learning Research*. 12: 2825–2830

[5] Dean, Jeff; Monga, Rajat; et al. (November 9, 2015). "TensorFlow: Large-scale machine learning on heterogeneous systems" (PDF). *TensorFlow.org*. Google Research.

[6] Haykin, S. and Network, N. (2004). A comprehensive foundation. *Neural Networks* , 2(2004)

[7] Justin Johnson & Andrej Karpathy, http://cs231n.github.io/convolutional-networks/

[8] Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning* , pages 89–96. ACM.

[9] Ito, Y. (1992). Approximation of continuous functions on rd by linear combinations of shifted rotations of a sigmoid function with and without scaling. *Neural Networks* , 5(1):105–115.

[10] Ba, J. and Frey, B. (2013). Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems* , pages 3084–3092.

[11] Stehman, Stephen V. (1997). "Selecting and interpreting measures of thematic classification accuracy". *Remote Sensing of Environment*. 62 (1): 77–89.

[12] Powers, David M W (2011). "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation". *Journal of Machine Learning Technologies*. 2 (1): 37–63

[13] Magnus Erik Hvass Pedersen , http://www.hvass-labs.org/