A Report

On

Convolutional Neural Networks

With a Java Implementation

By

Yash Gupta   2019A7PS1138P

Prepared in Partial Fulfillment of the Course

BITS F464 Machine Learning

Birla Institute of Technology and Science (BITS), Pilani

November, 2022

Table of Contents

Introduction

In 2019, a group of quants at Wall Street were able to realize an informational advantage of 4% to 5% around Walmart's quarterly financial statements release, which according to Panos Patatoukas - a professor at UC Berkeley - if added up, can become a staggering profit [1]. You might be expecting some very complex financial model leveraging options, high-frequency arbitrage, or convertible bonds; however, it was something entirely unexpected, or rather novel. They used satellite images of the parking lots to estimate the sales in advance for the Walmart outlets across the United States!

This warrants some thought. First, satellite data - while abundant - can be very difficult to analyze with the more traditional statistical and machine learning techniques. Second, even if a more recent method - say a fully-connected neural network - is used, it can take a staggering amount of time for just a county, let alone the entire country. So what's the secret recipe behind their lucrative strategy? CNNs or the convolutional neural networks - used by many but understood by few.

Computer Vision can be defined as the subfield of Artificial Intelligence where the goal is to build a computer replicating the visual intelligence of the human brain [2]. It has gained unprecedented attention in the preceding decade - the Google Lens is now ubiquitous, slowly winning over clients from its traditional textual-query based search engine counterpart. And much of this has been possible due to the efficient, intuitive, and robust solutions provided by CNNs.

However, saying that CNNs are restricted primarily to computer vision would be gravely undermining their significance. They can also be commonly found in models for forecasting of equity prices using multivariate time series of financial ratios, stock indices, and past opening/closing prices. Natural language processing applications commonly employ them to extract meaningful features and reduce the burden for the more computationally expensive sequence based recurrent layers [3]. Video data which has been daunting to process - let alone classify or tag - has been made tractable using CNNs. In the present data driven world, "data" is the veritable *"oil"* [4] and CNNs make it simpler for us to harness this oil to fuel our applications.

The objective of this report is to provide a high level theoretical and application-oriented overview of CNNs. The major operations underlying the working of CNNs would be touched upon and their implications will be discussed. Afterwards, CNNs shall be applied to the time-old yet significant problem of recognition of handwritten digits using the MNIST dataset. The report shall provide a brief walkthrough of the Java implementation of the network and present the results obtained on training and testing the said network.

First, however, let us briefly see what CNNs are - how do they get their name and how do they work their veritable marvels!

1. An Overview of CNNs

The section briefly describes what is meant by the term "convolution" in the name Convolutional Neural Network, how it fits into the operation of CNNs, and why these networks are used so ubiquitously. A rationale founded in human biology would be given to convince the reader of the intuitiveness of CNNs. The oft-companion pooling operation and the mechanics of padding and strides will hence be touched upon. Let us begin with the term convolution.

1.1 Convolution

The convolution operation is best understood with an example (adapted from [5]). Say we were observing the location of an ant on a 1D track. We have a real-valued continuous function $x(t)$ describing its position with respect to time t. If the function contains noise, we might want to take a weighted average of our observations obtained through the function - the weights being higher for times closer to our time of interest. If a denotes the age of the observation, the smoothed estimate of the position of the ant will be

$$s(t) = \int x(a)\, w(t - a)\, da$$

This operation on functions of real-valued arguments is called convolution and is sometimes expressed as
$s(t) = (x * w)(t)$, where x is often called the *input* and w is called the *kernel*. The output s is called the *feature map.*

If t is a discrete variable - as is often the case in computer science - the operation becomes

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)\, w(t - a)$$

With multidimensional input, say a 2D image I, we may write the operation as follows using a 2D kernel:

$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n)$, which can using the commutative property of convolution, can be written as

$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n)$, which can be interpreted as a sliding window of the kernel over the input image as implemented in most machine learning algorithms.

A note on disambiguation

The commutative property of convolution rigorously holds when we flip the kernel relative to the input, but if we do not flip the kernel, we obtain another related function called *cross-correlation,* which may be expressed as

$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n)$.

This is referred to as convolution by most machine learning libraries and practitioners. This report will follow the same convention.

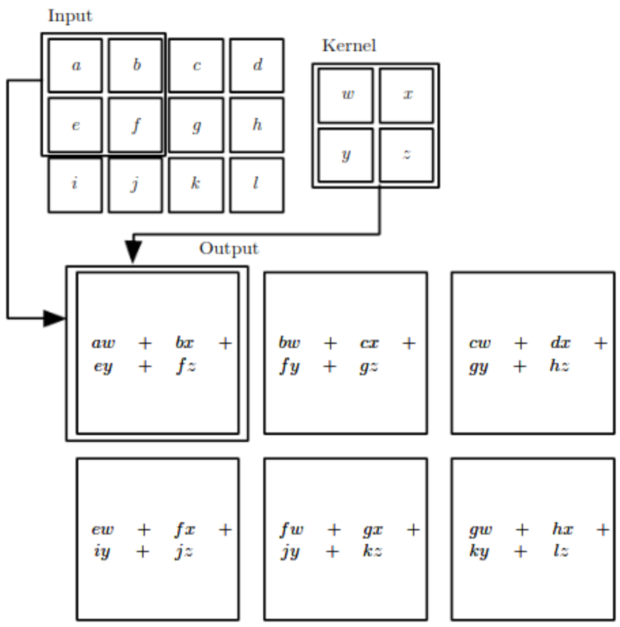


Figure 1.1 The cross-correlation operation in 2D.
Image Credit: Please refer [5].

Finally, discrete convolution - or more appropriately cross-correlation - can be regarded as matrix multiplication, as depicted in Figure 1.1. The flipping of kernel here equates to the rotation of the matrix about its center by an angle of π radians.

1.2 The Convolutional Layer

Before moving on to the rationale or motivation behind a CNN, let us first understand the working of a convolutional layer in practice. It may help us better appreciate the myriad benefits a CNN offers.

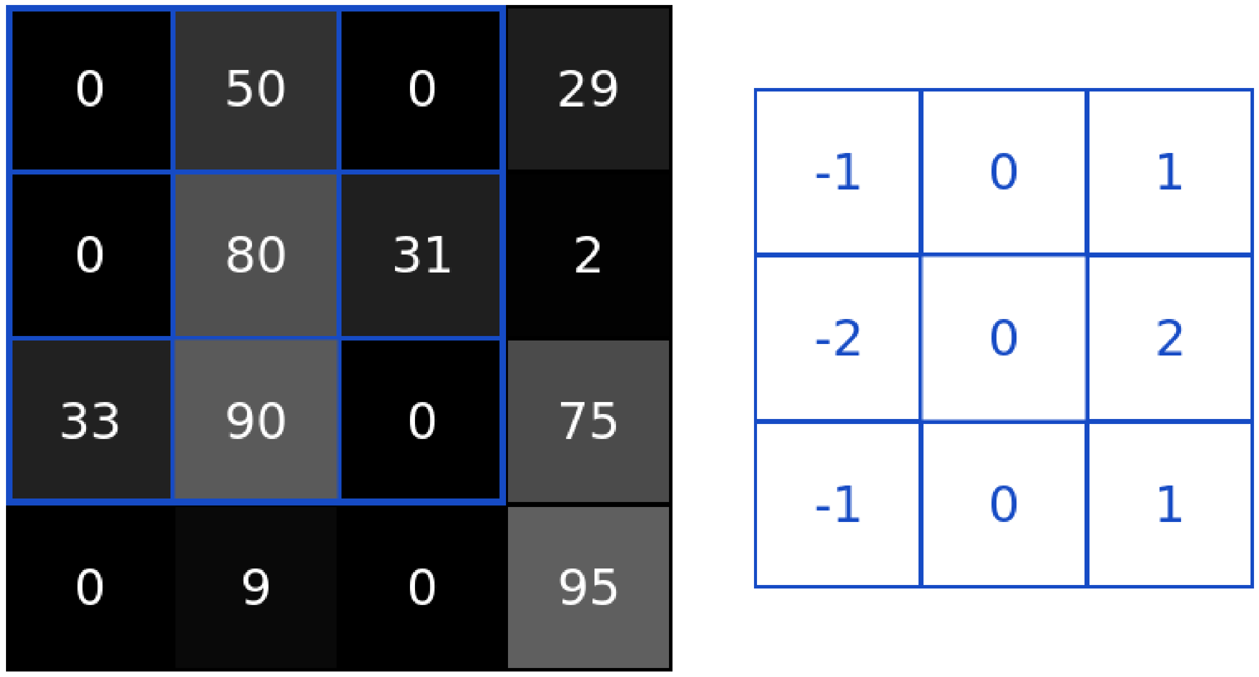


Figure 1.2 An example input and kernel for illustrating convolutional layer’s operation.
Image Credit: Victor Zhou, Intro to CNNs Part-1.

Consider the input matrix - or image - on the left and the kernel - sometimes called *filter* in machine learning jargon - depicted in Figure 1.2. A typical convolutional layer applies the convolution operation on the input using the filter as follows:

- Apply convolution operation on the currently selected window (blue window in this case) by performing an element-wise matrix multiplication followed by summation.
  In this case, -1(0) + 0(50) + 1(0) - 2(0) + 0(80) + 2(31) - 1(33) + 0(90) + 1(0) = 29.
- Save this answer in the resulting feature map.

- Move on to the next window by moving left-right (or up-down) as required using a sliding window operation. Repeat the same convolution step on the new selected window and so on to obtain the feature map shown in Figure 1.3.

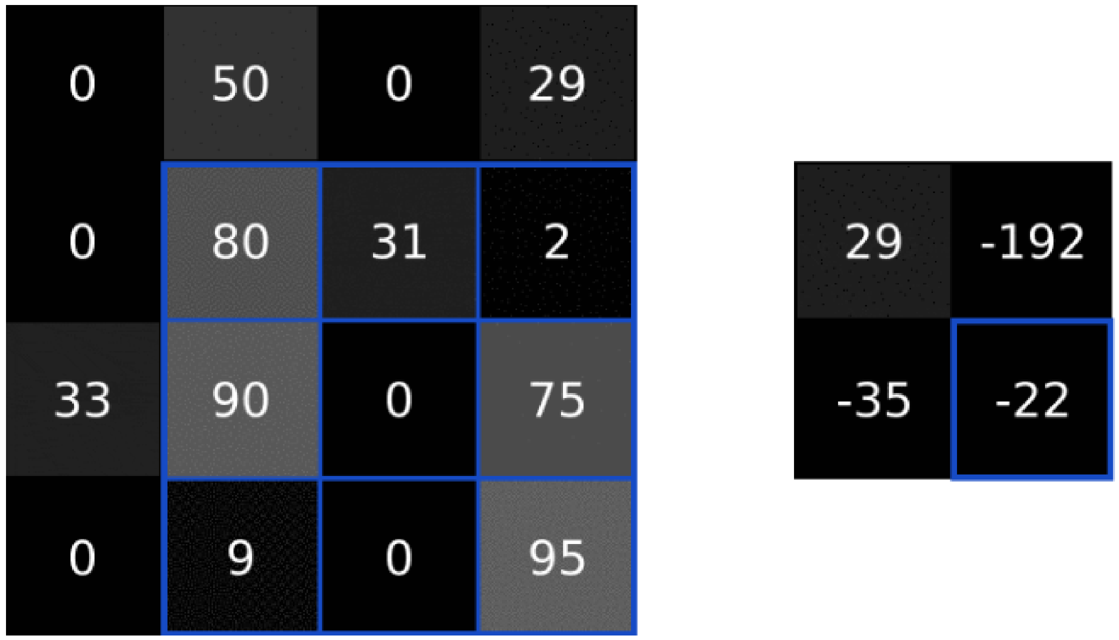


Figure 1.3 Input on the left and output on the right.
Image Credit: Victor Zhou, Intro to CNNs Part-1.

1.3 Intuition Behind Convolutional Layer

Now that we understand how the convolutional layer works, a natural question arises - why do it the way we did it? To see the point behind convolution, liken this operation to the working of a Sobel filter, or an edge detection filter. Figure 1.4 shows the application of a Sobel filter to a 2D image. The horizontal lines are evident in the output on the right. A high-valued, or bright, pixel in the output image indicates that there is a strong edge there in the input image [6].

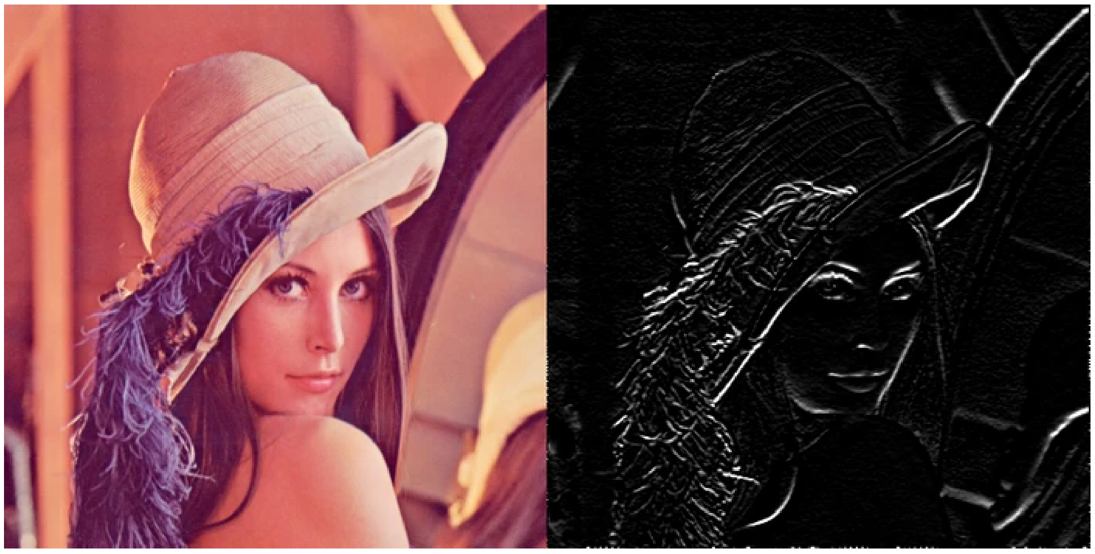


Figure 1.4 Original image and the image after horizontal edge detector.
Image Credit: Victor Zhou, Intro to CNNs Part-1.

According to the bottom-up approach of human visual perception in Psychology, our ability to recognize specific patterns, such as letters of the alphabet, arises from simpler capacities to recognize and correctly combine lower level features like lines, edges, corners, and angles. The pattern recognition is constructed from simpler perceptual abilities through a series of discrete steps [7]. Drawing a neuroscientific rationale from the preceding, we surmise that by emulating an action similar to Sobel filters, convolution in effect achieves feature extraction similar to the human visual cortex [5]. To summarize, convolution helps us look for *specific localized image features* - like edges and corners - that we use later in the network [6].

1.4 Padding and Strides

Note that in Figure 1.3, the points in the corners of the input image have been used only once in the convolution operations. The points at the edges of the input images have been used less number of times as compared to the points in the center. Further, notice that the image shrinks when using a filter of size more than 1.

The shrinkage of the image limits how many times we may apply the convolution. Furthermore, the pixels at the boundaries and vertices of the input image are used less, that is, a lot of information gets wasted, in effect. To avoid these problems we consider *padding* - that is, adding layers of zeros to our input image, as shown in Figure 1.5.

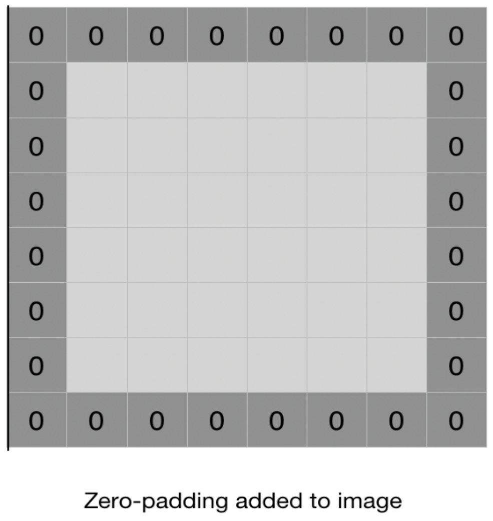


Figure 1.5 Zero padding added to the original image.
Image Credit: GFG - Introduction to Padding.

Two important classes of padding are

- ***Valid padding:*** We apply the convolution filters to only the valid pixels of the input, that it is equivalent to no padding at all.
- ***Same padding:*** The padding is chosen in such a way that the output feature map has the same dimensions as the original input image.

Another concept commonly used in conjunction is the *stride*, or the size of the step when sliding the filter over the input image. Figure 1.3 gives an example of a convolutional layer with a stride of 1. Figure 1.6 shows the operation with a stride of 2.

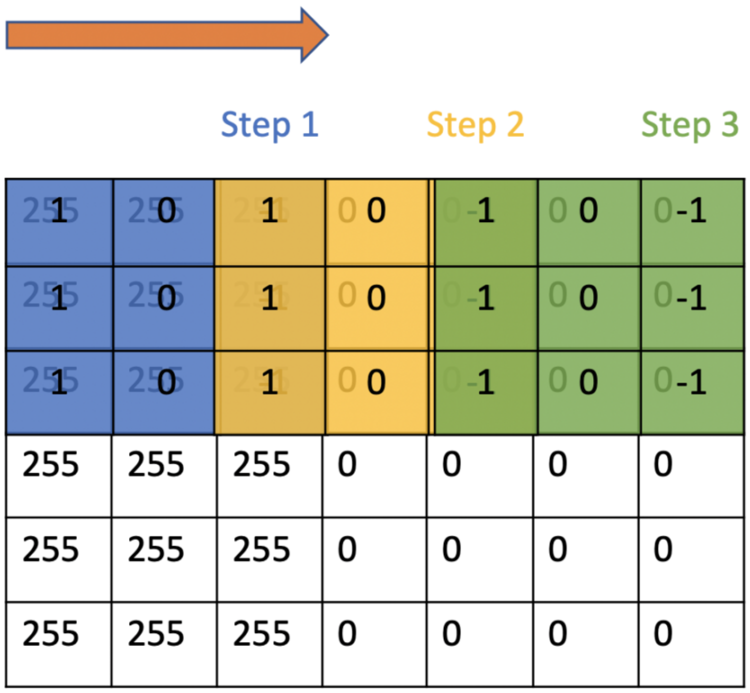


Figure 1.6 Convolution with stride = 2
Image Credit: Programmathically: Understanding Padding and Stride.

In general, if the image dimensions are $n_H \times n_W$, the filter dimensions are $f \times f$, the stride size is s, and the padding is p, the output dimensions will be $n_H^{out} \times n_W^{out} = \left\lfloor \frac{n_H+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W+2p-f}{s} + 1 \right\rfloor$, where $\lfloor x \rfloor$ denotes the floor function. Further, if there are n_c filters, then the depth of the output would also be n_C.

1.5 Pooling

Often used in conjunction with the convolutional layer is a pooling layer, which uses a pooling function to replace the output of a net at a certain location with the summary statistic of the nearby outputs [5]. The three types of pooling layers seen commonly in practice are

- ***Maxpooling:*** The output is replaced with the maximum value within the pool, that is, $x^{out} = max_{x^{in} \in pool\ window} x^{in}$.

- ***Minpooling:*** The output is replaced with the minimum value within the pool, that is, $x^{out} = min_{x^{in} \in pool\ window} x^{in}$.
- ***Averagepooling:*** The output is replaced with the arithmetic (or in some case a weighted) mean of the values within the pool, that is, $x^{out} = \frac{1}{(pool\ size)^2} \sum_{x^{in} \in pool\ window} x^{in}$.

Consider the operation of a maxpooling layer in Figure 1.7. In the selected blue pool, the maximum value (of 33, 90, 0, and 0) - 90 - is chosen to be taken to the output matrix. Also notice that the dimensions of the image gets halved here. In general, pooling divides the input's width and height by the pool size (2 in Figure 1.7) [6].

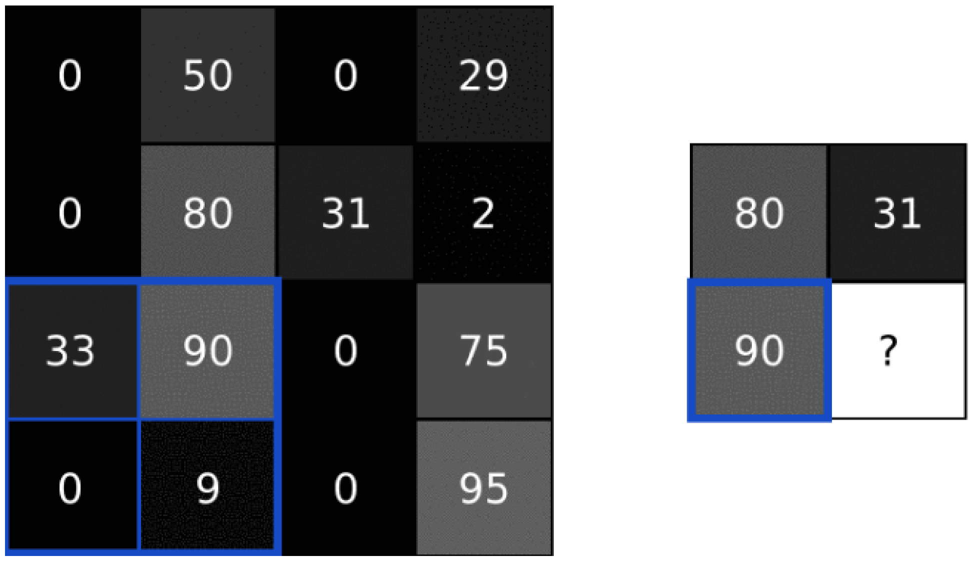


Figure 1.7 The input (left) and output of maxpooling (right)
Image Credit: Victor Zhou, Intro to CNNs Part-1.

An obvious consideration behind using a pooling layer would be to reduce the computational needs and the storage requirements by effectively slashing the number of features for the later layers [9]. The neighboring pixels in images tend to have similar values, so much of the information is redundant, allowing us to discard much of the pixels without losing out vital information [6]. Another benefit of pooling is *invariance* - especially, *translation invariance* - discussed later in the report.

1.6 A Representative CNN

As a typical example consider the network used by McGinnity et al [9] for the purpose of classifying 6 hand gestures as different letters in the Peruvian sign language, which implements their proposal depicted in Figure 1.8 for this multiclass classification problem.

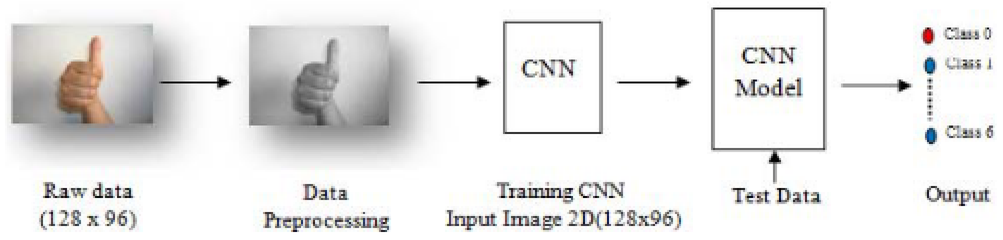
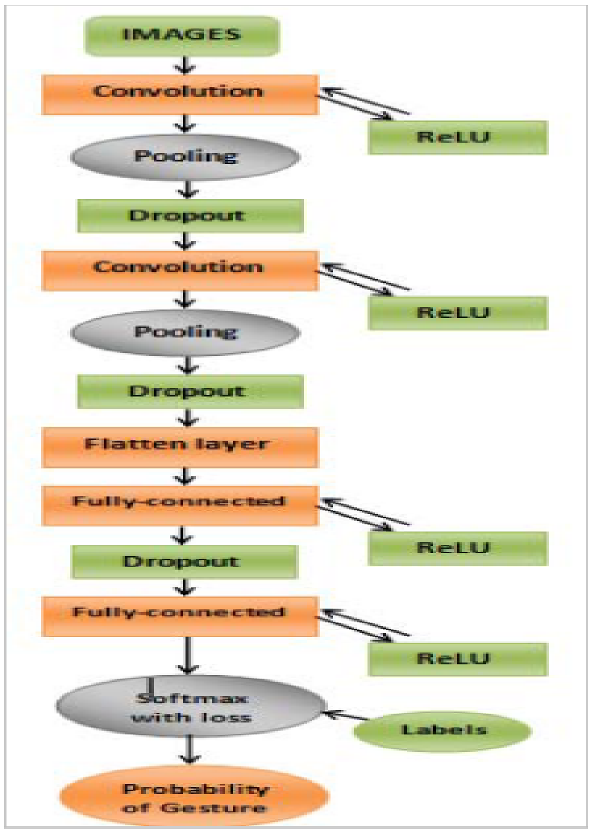


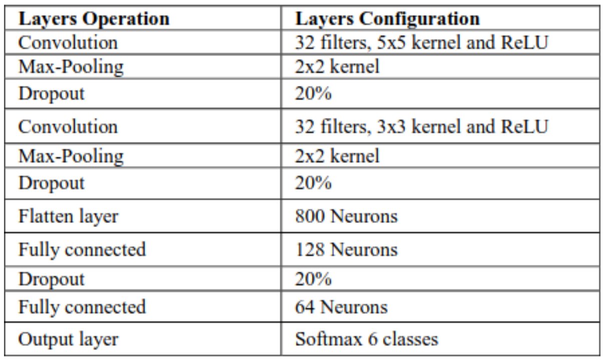
Figure 1.7 The proposed hand gesture recognition model.
Image Credit: Please refer [8]

They fed the preprocessed images (RGB channels reduced to blackonly-grayscale) to a convolutional layer, which was maxpooled for computation cost benefits. Dropout layers were used to reduce overfitting. The process or the convolutional block - of ReLu activated convolutional layers, maxpooling, and dropout - was repeated once more. Up to this stage, the images have been converted to feature rasters, but are 2D images nonetheless.

A flatten layer was used to vectorize the matrix and feed it into a fully connected network. Finally, a softmax layer was used to output probabilities for each class, as is the norm for multiclass classification problems.

The benefits offered by convolutional layers become evident when they realized a test-time accuracy of 0.95 with far less runtime cost and overfit compared to a traditional fully-connected network. These benefits are further elaborated upon in the following subsection.





Layers Operation	Layers Configuration
Convolution	32 filters, 5x5 kernel and ReLU
Max-Pooling	2x2 kernel
Dropout	20%
Convolution	32 filters, 3x3 kernel and ReLU
Max-Pooling	2x2 kernel
Dropout	20%
Flatten layer	800 Neurons
Fully connected	128 Neurons
Dropout	20%
Fully connected	64 Neurons
Output layer	Softmax 6 classes

Figure 1.7 CNN Architecture (left) with Details (right)
Image Credit: Please refer [8]

1.7 Motivation for CNNs

The motivation for using CNNs - and hence their widespread benefits - derives from the following special benefits offered by them which make them more amenable compared to their fully-connected counterparts, while still retaining the versatility and power of neural networks.

Sparsity of connections

When processing an image, there might be thousands, or even millions of pixels, while the number of meaningful features - such as edges - might be of the order of tens or hundreds, thereby requiring a lower number of kernels. Hence, if we store fewer parameters, we may improve the memory and runtime requirements of the model. In general, with m inputs and n outputs, matrix multiplication in a fully-connected layer would require $m \times n$ parameters, or weights, and hence any algorithms using it would be $O(m \times n)$ in runtime. If the we limit, the number of connections to k, we can obtain an $O(k \times n)$ runtime with the corresponding number of features in a sparsely connected approach [5]. Although the connections are sparse, the nature of convolution - sliding in particular - allows the later layers to interact with a much larger portion of the input image.

Parameter Sharing

It is a term for the use of the same parameter for more than one function in a model [5]. Unlike a traditional neural network using each element of the weight matrix exactly once when computing the output of the layer, each member of the kernel is used at almost every position of the input (as shown in Figure 1.8) thereby reducing the number of parameters to just k. Although the runtime of the forward propagation still remains $O(k \times n)$, the memory requirements are down to $k << m \times n$.

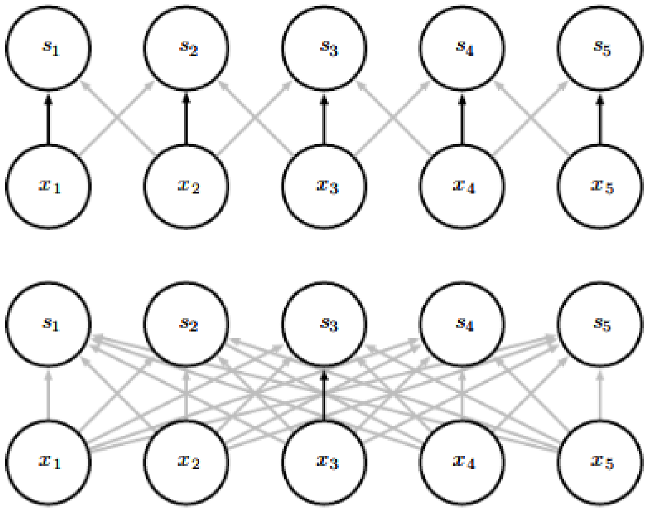


Figure 1.8 The black arrow or parameter is shared multiple times in the convolutional layer (top) and used just once in the fully connected layer (bottom).
Image Credit: Please refer [5]

Equivariance

The parameter sharing in convolution allows *equivariance to translation*, that is, the output changes the same as input for a translational change. Mathematically, the function f and g are equivariant if $f(g(x)) = g(f(x))$. This allows us to keep related features, such as edges in a person's face, close together regardless of the translational shifts in the input images.

Invariance

Pooling allows approximate invariance to small translations in the input, that is, small translations do not affect most values in the pooled output. Consider Figure 1.9 as an example - only half the outputs of the maxpooling layer get changed by a shift of 1 in the input. It can be useful when checking whether a feature is present exactly where it is supposed to be.

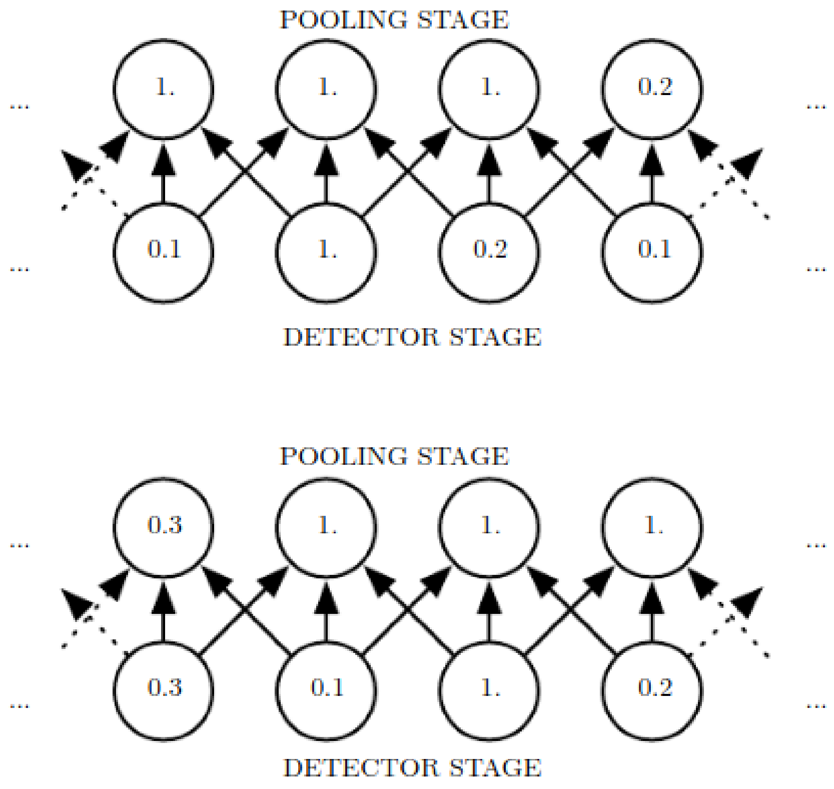


Figure 1.9 Example of Translation Invariance.
Image Credit: Please refer [5]

2. Applications of CNNs

The section briefly enumerates some of the more prominent applications of CNNs in the AI landscape today, particularly those in the field of computer vision. It also briefly touches upon the prospects of CNNs in the fields of time series analysis and natural language processing.

2.1 Image Classification

Image classification refers to the task of assigning a label (from a given set of possible labels) to a given image. It could be a binary classification problem - for example, calling an image a “Cat” or “Not a Cat” - or a multiclass classification problem - for example the hand gesture recognition problem in the preceding section. It is one of the earliest applications of CNNs in the field of computer vision.

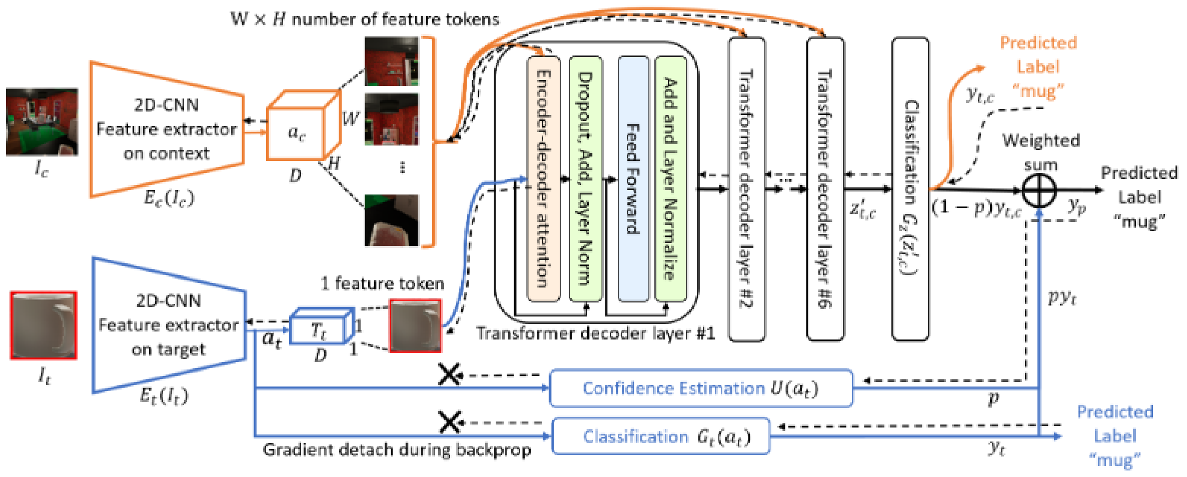

W × H number of feature tokens
2D-CNN Feature extractor on context
I_c
$E_c(I_c)$
a_c
W
H
D
1 feature token
2D-CNN Feature extractor on target
I_t
$E_t(I_t)$
a_t
T_t
Encoder-decoder attention
Dropout, Add, Layer Norm
Feed Forward
Add and Layer Normalize
Transformer decoder layer #1
Transformer decoder layer #2
Transformer decoder layer #6
$z'_{t,c}$
Classification $G_z(z'_{t,c})$
$y_{t,c}$
Predicted Label “mug”
Weighted sum
$(1-p)y_{t,c}$
y_p
py_t
Predicted Label “mug”
Confidence Estimation $U(a_t)$
p
Classification $G_t(a_t)$
y_t
Gradient detach during backprop
Predicted Label “mug”

Figure 2.1 Context aware image classification involving convolutional feature extractors.
Image Credit: *Please refer [10].*

The more recent development has been the incorporation of contextual clues for image classification, much like human brains do. We would be more likely to call an object a “coffee mug” if it is placed on a table than if it is floating in the air! Different convolutional layers are used to extract features pertaining to the object and its background or context, and the confidence of our prediction regarding the object is modified as per the contextual information [10], as depicted in Figure 2.1.

2.2 Object Localization and Detection

The problem of object localization involves locating where within the image the object is and drawing a bounding box around it. Object detection goes a step further to classify and detect all objects in the image, that is, assign a class to each object in the image and draw a bounding box around it. These tasks have found applications in

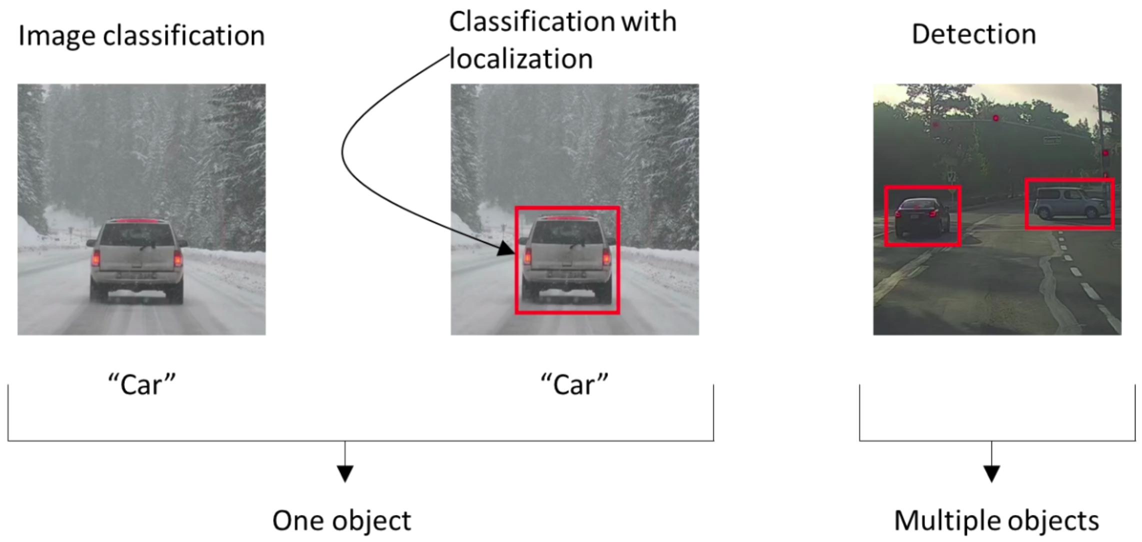


Figure 2.2 Localization and Object Detection.
Image Credit: Andrew Ng, Deep Learning Specialization.

The problem is usually tackled by using a sliding window approach to capture the object in a bounding box followed by a classification. CNNs have found application in object detection, particularly the classification part of the problem. This technology has been employed by recording footfall, industrial quality checks, facial recognition, text detection, and logo detection.

2.3 Image Segmentation

There are two prominent classes of image segmentation problems in literature: the semantic segmentation and the instance segmentation. *Semantic segmentation* involves labeling each pixel of the image with the corresponding class of what is being represented, while *instance segmentation* goes a step further to classify each instance of the class in the image separately, as shown in Figure 2.3.

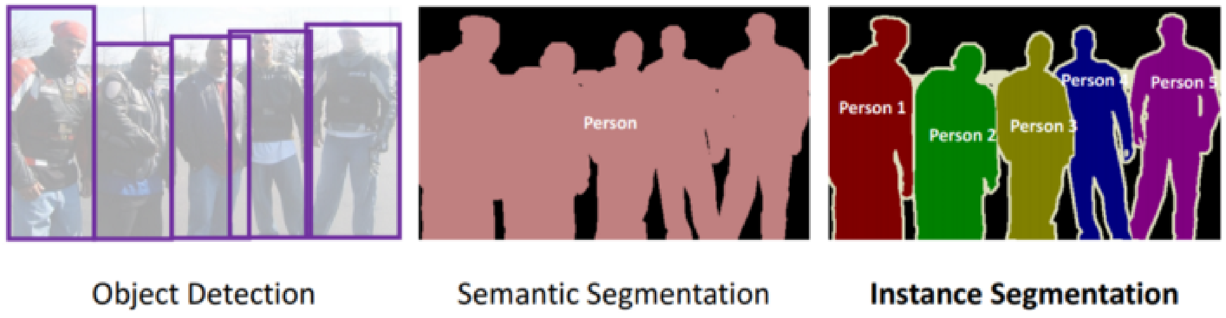


Figure 2.3 Object detection, semantic segmentation, and instance segmentation.
Image Credit: Harshall Lamba, Understanding Semantic Segmentation with U-Net.

The techniques find applications in medical image analysis (for instance, recognizing and demarcating a tumor), self-driving cars (to detect lane markings, traffic signs, other cars, and pedestrians), remote sensing (for instance, monitoring forest cover and water bodies), and mining and exploration (such as the famous salt deposit segmentation problem).

To see how CNNs might be useful in image segmentation, consider Figure 2.4, which shows the celebrated U-Net.

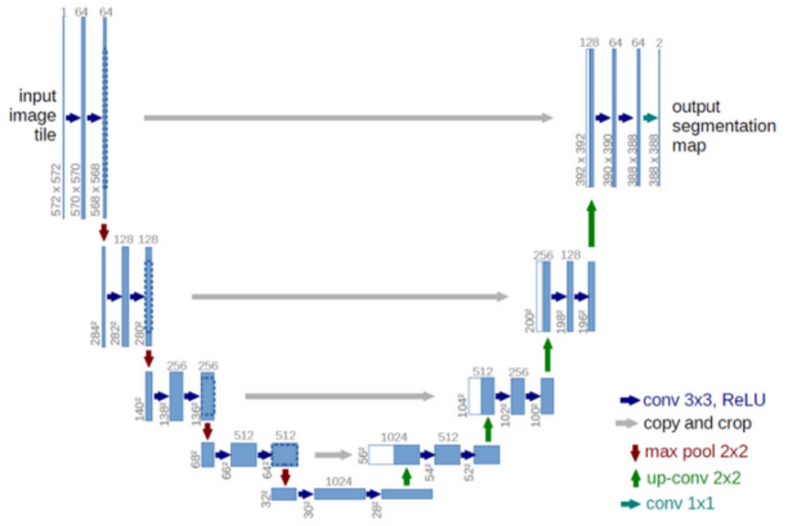


Figure 2.4 The U-Net architecture.
Image Credit: *Please refer [12].*

The U-Net takes an image along a contraction path (or a series of encoders) each of which is a stack of convolutional and maxpooling layers. This contraction is followed by a symmetric expanding path (or a series of decoders) which use transposed convolutions. Note the absence of fully-connected layers - it makes sense since the output has to be of the same dimensions (atleast in two dimensions) as the input image for the segmentation problem.

2.4 Neural Style Transfer

The neural style transfer can be understood as merging two images: a *content image* and a *style image* to create a *generated image*. The goal is to create an artificial system based on deep neural networks that creates artistic images of high perceptual quality. The algorithm uses neural representations to separate and recombine content and style of arbitrary images, thereby creating novel art and paving the way for an algorithmic understanding of how humans create and perceive artistic imagery [13]. The convolutional layers are used to extract meaningful features from the images - the shallowers layers detecting lower-level features like edges and simple textures and deeper layers detecting higher-level features like complex textures, object classes, and art styles. Using the features from the style, we perturb the content to generate the desired art, as shown in Figure 2.5.

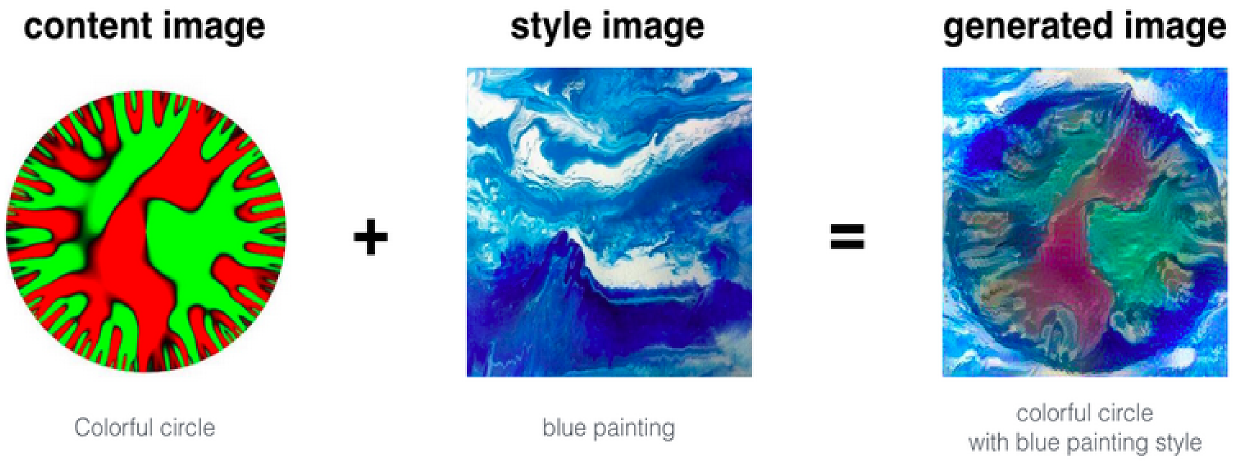
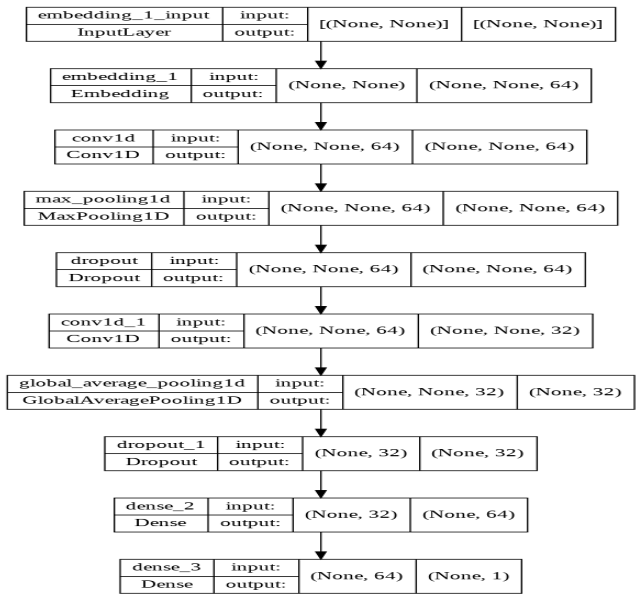


Figure 2.5 Neural Style Transfer in summary.
Image Credit: Coursera Deep Learning Specialization.

2.5 Natural Language Processing

The CNNs have crept into the field of natural language processing, where they have been employed for sentiment analysis - can be posed as a classification problem, each sentiment being the label to be assigned to the given text - since they can easily extract catchphrases in the text commonly associated with the expressed emotions.

A simple CNN based architecture has been applied to sentiment analysis of IMDb movie reviews, where they were able to achieve an accuracy of 0.90, comparable to that of LSTMs *ceteris paribus* [14], the architecture being shown in Figure 2.6.


embedding_1_input | InputLayer | input: | output: | [(None, None)] | [(None, None)]
embedding_1 | Embedding | input: | output: | (None, None) | (None, None, 64)
conv1d | Conv1D | input: | output: | (None, None, 64) | (None, None, 64)
max_pooling1d | MaxPooling1D | input: | output: | (None, None, 64) | (None, None, 64)
dropout | Dropout | input: | output: | (None, None, 64) | (None, None, 64)
conv1d_1 | Conv1D | input: | output: | (None, None, 64) | (None, None, 32)
global_average_pooling1d | GlobalAveragePooling1D | input: | output: | (None, None, 32) | (None, 32)
dropout_1 | Dropout | input: | output: | (None, 32) | (None, 32)
dense_2 | Dense | input: | output: | (None, 32) | (None, 64)
dense_3 | Dense | input: | output: | (None, 64) | (None, 1)
Figure 2.6 A CNN for sentiment analysis.
Image Credit: Coursera Deep Learning Specialization.

2.6 Time Series Classification

The convolutional layers in conjunction with LSTMs have been successfully applied to MVTS classification problems, such as the human activity recognition problem [15], where they involve over the time axis, extracting features which might otherwise be invisible to the human eye. This has led to the deep learning based approaches gradually replacing the traditional DTW based approaches for time series classification, especially for complex multivariate IoT data.

3. The Problem: Handwritten Digit Recognition

The section provides a walkthrough of the problem taken up for the present assignment, the dataset used for training and testing, the architecture chosen, the mathematics of the algorithms implemented, brief overview of the forward and backward propagation stages, and the results obtained on training and testing the network.

3.1 The Problem Statement

The problem taken up in this section is a classic example of the image classification set of problems. Given an image of a handwritten digit, our goal is to assign a label to the image, the label being the digit represented by the image. The problem will be posed as a machine learning problem, that is, we shall train a classification model on a set of examples - handwritten digit images - with known true labels and test it on another set of instances, or images, where we do not reveal the true labels to the model.

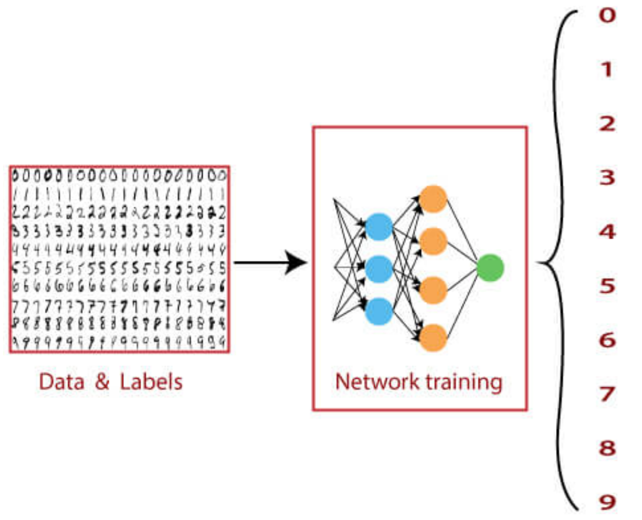


Figure 3.1 The Handwritten Digit Recognition Problem.
Image Credit: Javatpoint, The Tensorflow MNIST Dataset.

Specifically, we would be using a simple convolutional neural network as our classification model. It may be noted that the problem is a multiclass classification problem where we have more than two classes. Therefore, we would have for the output of our model the probabilities of the image being any particular digit, thereafter, choosing the digit with the highest probability as the assigned label.

To pose the problem more rigorously, we define a loss function which is to be minimized during the course of training:
$L = - ln(p_c)$, where p_c is the predicted probability for the correct class *c*. This loss function is sometimes termed the cross-entropy loss. We wish to minimize our classification error - especially the testing error - by minimizing the loss in the course of training.

3.2 The Dataset and Data Preprocessing

The classic MNIST data set has been used for its simplicity and moderate memory requirements, making it suitable for a non-parallelized moderate-load personal-use CPU environment. There are 70,000 images of handwritten digits, each 28×28 pixels (784 pixels total) and one channel (black-only grayscale). There are ten class labels associated with the images: 0, 1, 2, …, 9 (Table 3.1). The dataset has been constructed from scanned documents available under the National Institute of Standards and Technology (NIST).

Table 3.1 The MNIST Dataset

Example Image										
Label	0	1	2	3	4	5	6	7	8	9

By default, the images have been split into two partitions: 60,000 for training and 10,000 for testing. However, due to computational requirements involved, only 40,000 images for training and 10,000 images for testing have been taken up for the present implementation, thereby achieving a 5 : 1 train : test ratio.

The images have been stored in a `.png` format under their respective digit directories, which themselves have been stored under the parent training and testing directories. Java modules have been developed to read images directly from these directory structures and convert them into the familiar matrix representation.

Images have been normalized to the interval $[-\ 0.5,\ 0.5]$ for faster and more reliable convergence of the network.

3.3 The Network Architecture

A simple network architecture has been adopted, incorporating all the essential components - convolution, pooling, flattening, and probability assignment - yet nothing too complex, the reason being the ease of experimenting such a simple network on a personal-use machine and the clarity of backpropagation mechanics for such an architecture. The broad form of the network may be seen in Figure 3.3.

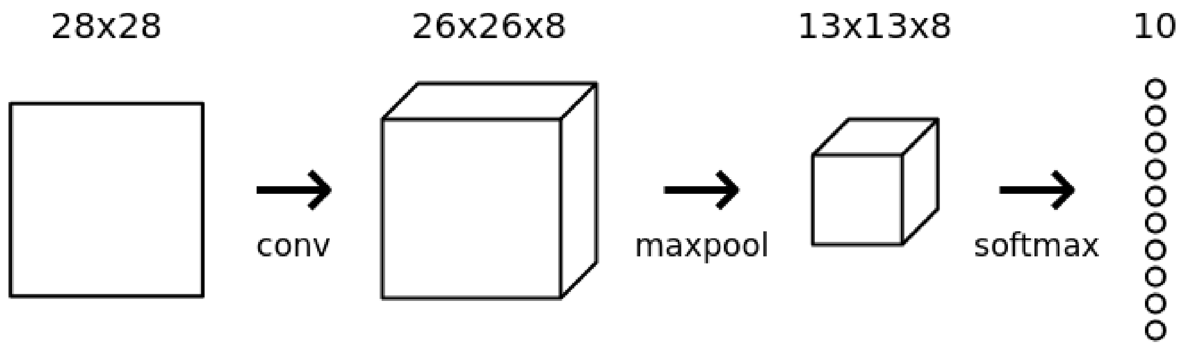


Figure 3.2 The Network Architecture Chosen
Image Credit: Victor Zhou, Intro to CNNs in Keras

A softmax layer follows the maxpooling layer since we want to assign a probability to each of the 10 classes. Also note that the functionality of the traditional `tensorflow.keras.layers.Flatten()` has been incorporated into the fully-connected flatten layer itself.

3.4 The Forward Propagation

Let us walk through the network as an image passes during its forward pass:

The Convolutional Layer `(Conv3x3.java)`
A 28×28 pixel normalized image is passed to the convolutional layer, which applies 8 (customizable) 3×3 filters to the image with a valid padding and a single-step stride. The result is a $26 \times 26 \times 8$ tensor to be passed to the pooling layer.

The Maxpooling Layer `(MaxPooling.java)`

This layer applies the maxpooling operation on a 2×2 (customizable) pool window at a time, reducing the tensor shape to $13 \times 13 \times 8$ for the output.

The Fully Connected Layer `(Dense.java)`

The $13 \times 13 \times 8$ tensor is flattened to a 1352 sized vector (1×1352 matrix in implementation to maintain equivalence between vectors and matrices in the custom-made matrix operations modules). This leads to the use of a weight matrix of size 1352×10 and a bias vector of size 1×10 to give a 1×10 vector output. Note that a *softmax* activation has been applied in this layer, that is $z = [z_0, z_1, ..., z_9]^T = W^T x + b$ is converted to $a = softmax(z)$, where, $a_i = \frac{e^{z_i}}{\sum_{i=0}^{9} e^{z_i}}$ for $i \in [0 .. 9]$. The class with the highest probability ($arg\,max_i\, a_i$) is chosen as the assigned label.

3.5 The Backward Propagation

Since our objective is to minimize the loss function, we propagate the gradients from the loss back to the parameters of each layer and update them as follows with gradient descent (learning rate $\eta = 5 \times 10^{-3}$ and loss clipping constant $\epsilon = 10^{-6}$ for our training but customizable with the `compile()` method of the class `CNN`).

The Fully Connected Layer `(Dense.java)`

$\frac{\partial L}{\partial a_i} = 0,\ i \neq c$ and $\frac{\partial L}{\partial a_i} = \frac{-1}{a_i},\ i = c$, where *c* is the correct label.

$\frac{\partial a_c}{\partial z_i} = \frac{-e^{z_c} e^{z_i}}{(\sum_{j=0}^{9} e^{z_j})^2},\ i \neq c$ and $\frac{\partial a_c}{\partial z_i} = \frac{e^{z_c}(\sum_{j=0}^{9} e^{z_j} - e^{z_c})}{(\sum_{j=0}^{9} e^{z_j})^2},\ i = c.$

$\frac{\partial z}{\partial w} = input$, $\frac{\partial z}{\partial b} = 1$, and $\frac{\partial z}{\partial input} = w$.

Use chain rule to get $\frac{\partial L}{\partial w}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial input}$.

Update the parameters as $w^{+} = w - \eta\frac{\partial L}{\partial w}$ and $b^{+} = b - \eta\frac{\partial L}{\partial w}$.
The gradient $\frac{\partial L}{\partial input}$ is backpropagated to the previous layer where it serves as $\frac{\partial L}{\partial output}$.

The Maxpooling Layer (`MaxPooling.java`)
There are no weights this time, and an input cell which is not the maximum value in a pooling window does not affect the output - and hence the loss. Therefore, $\frac{\partial L}{\partial input} = 0$ for non-max pixels and $\frac{\partial output}{\partial input} = 1 \Rightarrow \frac{\partial L}{\partial input} = \frac{\partial output}{\partial input}$ for the maximum pixel, which is backpropagated to the previous layer where it serves as $\frac{\partial L}{\partial output}$.

The Convolutional Layer (`Conv3x3.java`)
To update the filter values, use $\frac{\partial output[i,j]}{\partial filter[r,c]} = img(i + r, j + c)$ and $\frac{\partial L}{\partial filter[r,c]} = \sum_{i}\sum_{j}\frac{\partial L}{\partial output[i,j]} \bullet \frac{\partial output[i,j]}{\partial filter[r,c]}$. Then perform
$filter^{+}[r, c] = filter[r, c] - \eta\frac{\partial L}{\partial filter[r,c]}$.
There is no gradient that needs to passed backward in this layer, for it is the first layer.

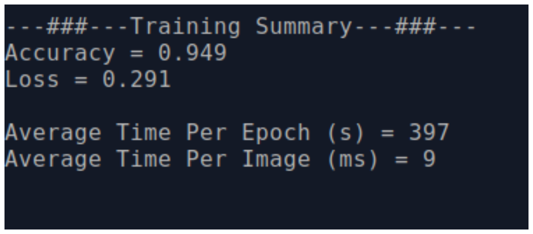
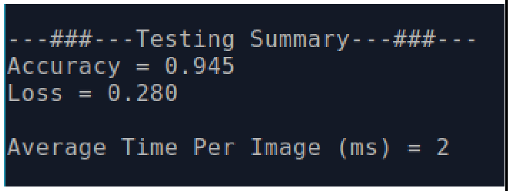
3.6 The Results

After training for 15 epochs on 40,000 training images with a $\eta = 5 \times 10^{-3}$ and $\epsilon = 10^{-6}$, the results shown in Table 3.2 were obtained with 8,000 testing images. Note that the training and testing accuracies are quite close (0.95 vs 0.945), indicating that the model did *not* suffer from an overfit. This may be attributed to the relatively large number of images available for the problem.

Also notice that the training time per image is significantly higher than the testing time per image (9 ms vs 2 ms). A plausible explanation for this observation can be the higher runtime requirements for the backward propagation phase as compared to the forward propagation phase (which makes sense when we consider the numerous matrix multiplications involved in the former).

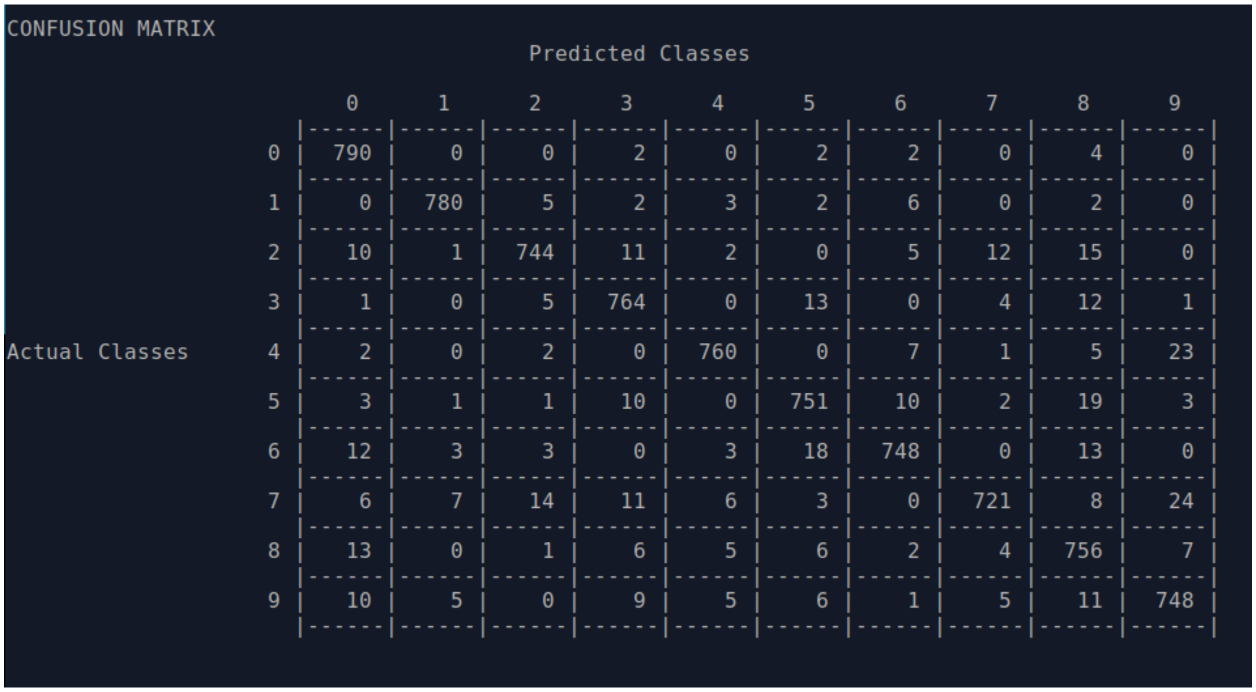
Table 3.2 Results Summarized

Metric	Training	Testing

Summary	 ---###---Training Summary---###---
	Accuracy = 0.949
	Loss = 0.291
	
	Average Time Per Epoch (s) = 397
	Average Time Per Image (ms) = 9
		 ---###---Testing Summary---###---
		Accuracy = 0.945
		Loss = 0.280
		
		Average Time Per Image (ms) = 2

Accuracy	0.95	0.945
Average Time Per Image	9 ms	2 ms

To further ensure that the model was producing correct results for most testing instances, a confusion matrix was plotted as shown in Figure 3.4. Note the larger values are along the diagonal, indicating the correct label assignments.



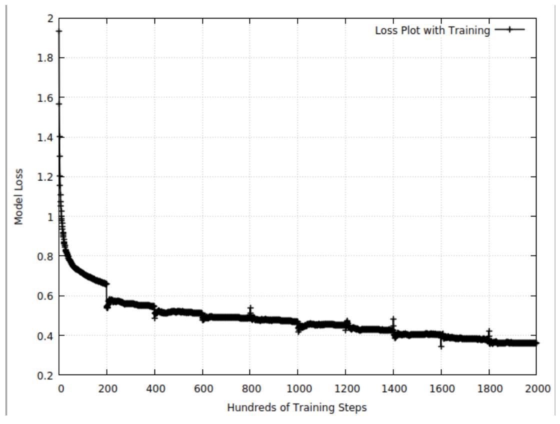
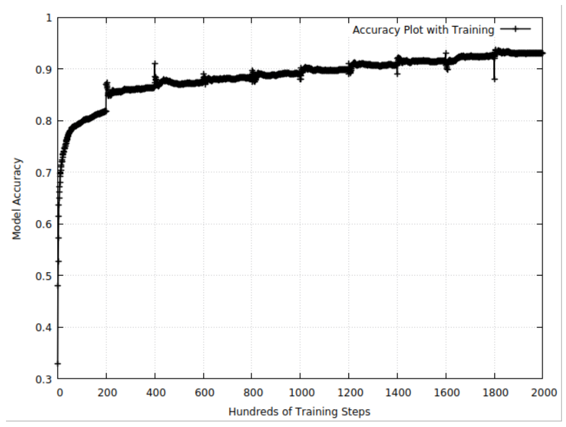
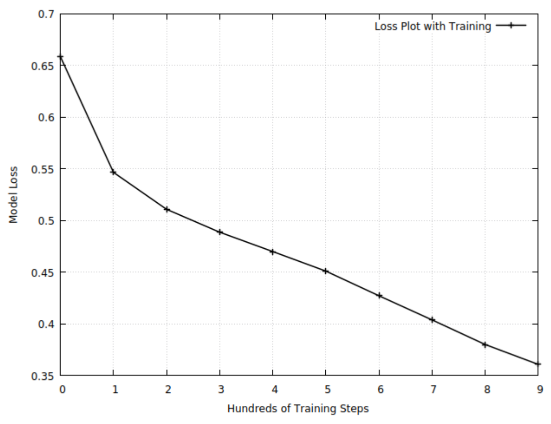
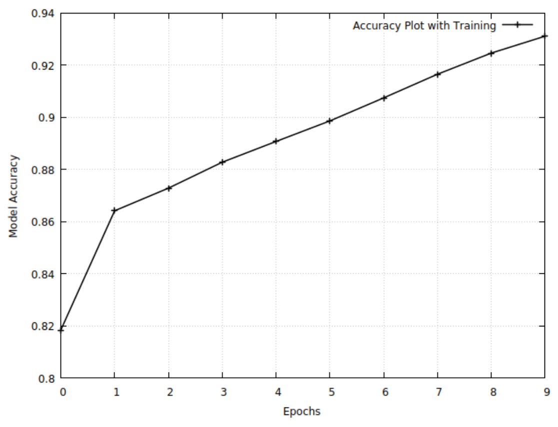
```
CONFUSION MATRIX
                                     Predicted Classes

                        0      1      2      3      4      5      6      7      8      9
                    |------|------|------|------|------|------|------|------|------|------|
                  0 |  790 |    0 |    0 |    2 |    0 |    2 |    2 |    0 |    4 |    0 |
                    |------|------|------|------|------|------|------|------|------|------|
                  1 |    0 |  780 |    5 |    2 |    3 |    2 |    6 |    0 |    2 |    0 |
                    |------|------|------|------|------|------|------|------|------|------|
                  2 |   10 |    1 |  744 |   11 |    2 |    0 |    5 |   12 |   15 |    0 |
                    |------|------|------|------|------|------|------|------|------|------|
                  3 |    1 |    0 |    5 |  764 |    0 |   13 |    0 |    4 |   12 |    1 |
                    |------|------|------|------|------|------|------|------|------|------|
Actual Classes    4 |    2 |    0 |    2 |    0 |  760 |    0 |    7 |    1 |    5 |   23 |
                    |------|------|------|------|------|------|------|------|------|------|
                  5 |    3 |    1 |    1 |   10 |    0 |  751 |   10 |    2 |   19 |    3 |
                    |------|------|------|------|------|------|------|------|------|------|
                  6 |   12 |    3 |    3 |    0 |    3 |   18 |  748 |    0 |   13 |    0 |
                    |------|------|------|------|------|------|------|------|------|------|
                  7 |    6 |    7 |   14 |   11 |    6 |    3 |    0 |  721 |    8 |   24 |
                    |------|------|------|------|------|------|------|------|------|------|
                  8 |   13 |    0 |    1 |    6 |    5 |    6 |    2 |    4 |  756 |    7 |
                    |------|------|------|------|------|------|------|------|------|------|
                  9 |   10 |    5 |    0 |    9 |    5 |    6 |    1 |    5 |   11 |  748 |
                    |------|------|------|------|------|------|------|------|------|------|
```

Figure 3.3 The Confusion Matrix
Image Credit: Yash Gupta, *Please use run.sh to replicate the results.*

To see the behavior of the model during training, the model accuracy and loss were plotted against the training steps (in multiples of 100) and the epochs, as shown in Table 3.3. Note that since *stochastic gradient descent* has been used, each training step corresponds to one image.

Table 3.3 The Training Behavior (graphs generated with the Shell GNU-plot command)

$y - axis \rightarrow$ / $x - axis \downarrow$	Model Loss	Model Accuracy
Training Steps (in hundreds)		
Epochs		

Notice how loss decays consistently with the epochs, indicating that the learning rate has been set right and that the model is not suffering from major artifacts. The small fluctuations in the loss versus training steps plot may be attributed to 'the updation with one image at a time' property of stochastic gradient descent.

3.7 Future Steps and Conclusion

While test error rates as low as 0.5% have been recorded in the literature for the MNIST handwritten digit classification problem, our model performs satisfactorily when viewed in comparison to an almost equivalent `Keras` implementation, where it achieves a 0.964 accuracy on the testing partition after training with *all* the training images (notice that we used only ⅔ of the training images). Still, the model might benefit from inclusion of additional layers (going deeper) with appropriate overfitting handling mechanisms (regularization and dropouts, for example).

The training and testing time can be brought down significantly by implementing parallel computing techniques for the matrix operations modules (`MatrixOps.java`) as done by `Tensorflow` and `Numpy` open-source developers. Provisions can be made to handle multichannel images, such as colored images and multispectral images. Finally, the network can be made more friendly by introducing customizable filter sizes, padding, and strides (also allowing room for greater hyperparameter optimization).

Nonetheless, it is believed that the present network has helped immensely in demonstrating the mechanics of a typical convolutional network along with the nitty-gritties of matrix and image operations. Such an understanding can greatly benefit future development with high-level (more abstracted) APIs for more complex tasks, such as those alluded to in Section 2.

References

[1] Prashant, "How satellite imagery is helping hedge funds outperform," *International Banker*, 26-Jun-2020. [Online]. Available: https://internationalbanker.com/brokerage/how-satellite-imagery-is-helping-hedge-funds-outperform/. [Accessed: 19-Nov-2022].

[2] Connectdreams, "Beginner's Guide to Computer Vision", *Medium,* 18-Nov-2016. [Online] Available: https://medium.com/readers-writers-digest/beginners-guide-to-computer-vision-23606224b720. [Accessed: 19-Nov-2022].

[3] Z. Jianqiang, G. Xiaolin and Z. Xuejun, "Deep Convolution Neural Networks for Twitter Sentiment Analysis," in *IEEE Access*, vol. 6, pp. 23253-23260, 2018, doi: 10.1109/ACCESS.2017.2776930.

[4]"Joe Kaeser: Data is the 21st century's oil, says Siemens CEO Joe Kaeser," *The Economic Times*. [Online]. Available: https://economictimes.indiatimes.com/magazines/panache/data-is-the-21st-centurys-oil-says-siemens-ceo-joe-kaeser/articleshow/64298125.cms?from=mdr. [Accessed: 19-Nov-2022].

[5] Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

[6] Zhou, "An Introduction to Convolutional Neural Networks Part 1", *Victor Zhou's Blog*, 10-Nov-2019. [Online]. Available: https://victorzhou.com/blog/intro-to-cnns-part-1/. [Accessed: 10-Nov-2021].

[7] Baron, R., Misra, G. (2016). Psychology, 5th Edition, Pearson India Education Services. ISBN: 978-93-325-5854-0.

[8] Andrew Ng, "Course 3: Convolution Neural Networks", *The Coursera Deep Learning Specialization*. [Accessed: 02-March-2022]

[9] A. A. Alani, G. Cosma, A. Taherkhani and T. M. McGinnity, "Hand gesture recognition using an adapted convolutional neural network with data augmentation," *2018 4th International Conference on Information Management (ICIM)*, 2018, pp. 5-12, doi: 10.1109/INFOMAN.2018.8392660.

[10] P. Bomatter, M. Zhang, D. Karev, S. Madan, C. Tseng, and G. Kreiman, "When Pigs Fly: Contextual Reasoning in synthetic and natural scenes," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[11] Stewart M., “Localization and Object Detection with Deep Learning”, *TowardsDataScience, 25-Mar-2019.* [Online]. Available: https://towardsdatascience.com/localization-and-object-detection-with-deep-learning-67b5aca67f22 . [Accessed 20-Nov-2022]

[12] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *Lecture Notes in Computer Science*, pp. 234–241, 2015.

[13] L. Gatys, A. Ecker, and M. Bethge, “A neural algorithm of artistic style,” *Journal of Vision*, vol. 16, no. 12, p. 326, 2016.

[14] M. R. Haque, S. Akter Lima and S. Z. Mishu, "Performance Analysis of Different Neural Networks for Sentiment Analysis on IMDb Movie Reviews," *2019 3rd International Conference on Electrical, Computer & Telecommunication Engineering (ICECTE)*, 2019, pp. 161-164, doi: 10.1109/ICECTE48615.2019.9303573.

[15] UCI Machine Learning Repository: Human Activity Recognition using Smartphones Data Set. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones. [Accessed: 20-Nov-2022].

Dataset Used

The MNIST Database of Handwritten Digits: The dataset is available in public domain as .gz files at the link: http://yann.lecun.com/exdb/mnist/. The .png format data used in the present implementation can be found at the link: https://www.kaggle.com/datasets/jidhumohan/mnist-png.

Appendix: Source Code Files

Following is a list of source code files - containing the implementation of the network, various auxiliary functions, plotting utilities, and a consolidated runtime script - attached with the present report.

Java Files

- `Driver.java`: main function instantiating a CNN object and training and testing it, while also saving the epoch- and step-wise loss and accuracy
- `CNN.java`: the CNN constructor and training and testing modules
- `Conv3x3.java`: the convolutional layer - both forward and backward props
- `MaxPooling.java`: the pooling layer - both forward and backward props
- `Dense.java`: the fully connected softmax layer - both forward and backward props
- `ImageOps.java`: modules to open and process images
- `MatrixOps.java`: modules to emulate `NumPy` operations like adding matrices, matrix dot products, flattening matrices, and the like
- `PrettyPrint.java`: printing results on to console or required output files in a human friendly format

GNUPlot Files

- `plotEpochAcc.p`: plotting the training accuracy against epochs
- `plotStepAcc.p`: plotting the training accuracy against hundreds of steps
- `plotEpochLoss.p`: plotting the training loss against epochs
- `plotStepLoss.p`: plotting the training loss against hundreds of steps

Shell Scripts

- `run.sh`: an all-in-one script to compile all relevant Java files, run the driver, and plot the results obtained