MNIST Handwritten Digits: Reviewing the Convolutional Neural Network and LeNet-5 Architecture

Peter Akioyamen[*]
pakioyam@uwo.ca
Applied Mathematics 3911G
Western University
London, Ontario, Canada

ABSTRACT

The fields of machine learning and deep learning have seen explosive growth in the study of their theory and application within the last 10 years. In this paper we review the convolutional neural network – particularly, we look to reproduce the seminal LeNet-5 architecture and its results on the now widely used MNIST handwritten digits data set. Further, we provide a general overview of some select advancements in the field of deep learning.

1 INTRODUCTION

In 1950, Alan Turing proposed the idea of a “learning machine” which could become artificially intelligent and later, in 1958, the perceptron was conceptualized by Frank Rosenblatt [11]. The amalgamation of the fields of computer science, statistics, linear algebra, and calculus have allowed for the emergence and rapid development of the field now known as machine learning. Machine learning itself can be thought of as an application of artificial intelligence (AI) – specifically, a manifestation of the idea that systems can learn from data, identify patterns, improve themselves, and make decisions with minimal human intervention.

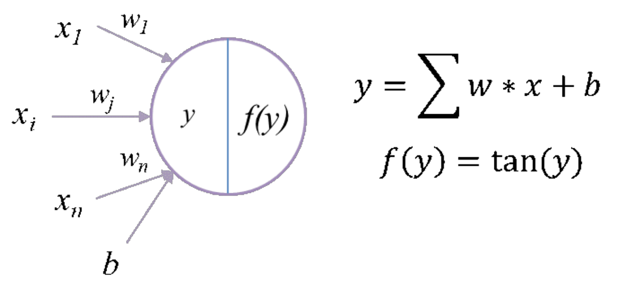


Figure 1: An exemplar perceptron and its computation.

An important framework within the domain of machine learning is the artificial neural network (ANN), which is composed of multiple perceptrons. A perceptron can be thought of as a singular node which computes the summation of the multiplication between inputs, weight terms, and a bias term. A perceptron usually applies a transformation afterward, referred to as an activation function, to add non-linearity to the output (Figure 1). Neural networks are often composed of multiple layers of interconnected perceptrons with different or similar activation functions, allowing for high levels of abstraction. The development of the ANN gave rise to the field of deep learning. Deep learning, a sub-field within machine learning, is concerned with artificial neural network algorithms for representation learning in many capacities. A multilayer perceptron (MLP), one of the most commonly used artificial neural network types, can be seen in Figure 2.

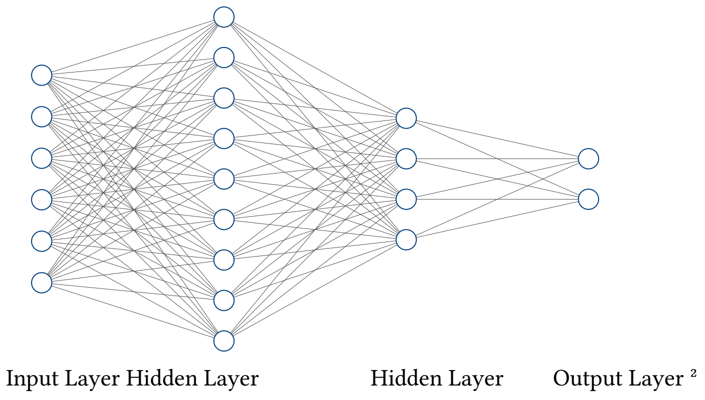


Figure 2: A generic multilayer perceptron.

Many of the discoveries made during the 1950s to the early 2000s in theoretical computer science, probability theory, and other related disciplines formed the foundation for modern deep learning today. Though, if the theoretical understanding of the methods we use today were established long ago, why did the methods go unused in practice for so long? The reality is that there were two prominent constraints limiting application and further development. These constraints have been addressed, in some capacity, only in the last 10–15 years. The first constraint being hardware. Generally, a lack in computational power made it infeasible to train and use these algorithms. Further, the storage capacity and memory access speeds available were either not fit for the algorithms' requirements or were too expensive for individuals to procure. The second constraint was the need for data and its lack thereof. Rapid advances in computer hardware, along with the aggregation of data becoming more commonplace, allowed the field of deep learning to be studied more rigorously than ever before, both in application and theory. These advances promoted the use of deep learning and machine learning for both classification and regression based tasks, among others, in recent years.

In this paper we seek to reproduce the seminal LeNet-5 model and replicate its results classifying handwritten digits from the MNIST database [9]. This data set consists of 60000 training images and 10000 testing images. The classification of handwritten

[*]Departments of Applied Mathematics, Computer Science, and Statistics & Actuarial Science.

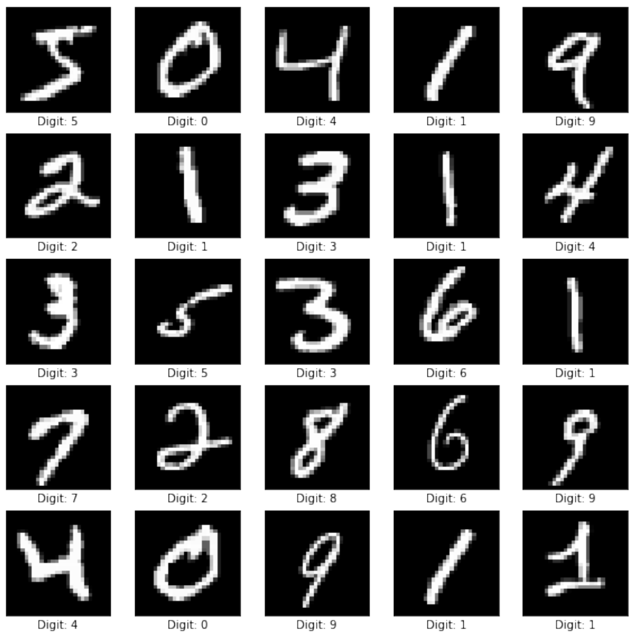


Figure 3: Images from MNIST data set.

digits has important implications for the field of computer vision and is now commonly recognized as the *hello world* equivalent for computer vision and image recognition in deep learning. We seek to build a model which, through training, is able to create an internal representation of the input image and extract essential information which will allow it to then accurately predict or classify the image as a corresponding digit, 0 to 9. Figure 3 shows a set of handwritten digits from the MNIST data set. We first act to reproduce the original model seen in [8] with minimal changes to the model architecture. We then develop an alternate version of the LeNet-5 model implementing some changes to the original architecture which are often preferred in modern deep learning.

The paper is organized as follows, in Section 2 we give an overview of the convolutional neural network and its motivations. Section 3 discusses the original LeNet-5 model while in Section 4 we discuss the high level differences between the two model implementations we use in this paper and the original model used in the primary reference paper. Section 5 shows the results of the models, and finally, we conclude the paper in Section 6, profiling some select topics and models within deep learning.

2 CONVOLUTIONAL NEURAL NETWORKS

A typical multilayer perceptron network used for classification or regression tasks works by taking inputs at each node, summing the multiplication of the inputs and weights and adding a bias, and adding non-linearity to the output via an activation function. The output of one layer acts as the input to the next. The final layer, known as the output layer, will often have an activation function which maps the output to a range that is domain specific – for example, an activation function could be used to produce a binary value corresponding to one of two classes in a binary classification problem, or no activation function may be used in order to preserve the continuous nature of the output in a regression problem. In order to achieve more accurate outputs, we seek to minimize the error given by a selected loss or cost function. To do this, the loss between the output value and the actual value from the data is computed. This loss is then attributed to the many weights and biases throughout the neural network using back-propagation or a related process, and the weights and biases are adjusted by a value, the learning rate. The process is often repeated until the network converges, that is, until a desired loss is achieved or the loss no longer seems to be improving.

Despite being useful in many domains, multilayer perceptron networks have drawbacks. These networks are fully connected, so a perceptron in one layer is connected to every single perceptron in the preceding layer and the succeeding layer. The many weight and bias terms needed in the network can result in redundancy and inefficiencies. As a result, in some use-cases, these networks can be hard to train and may not converge. In the field of computer vision, input data are images which often exhibit spatial correlations. In order to train an MLP to classify image data, the data is flattened to a 1-dimensional vector prior to input. Image data is often stored in a native 2-dimensional or 3-dimensional format; as a result of taking the 1-dimensional vector as input, an MLP network disregards vital spatial information contained in the image data, which is undesirable for learning. The convolutional neural network (CNN) is a model which helps to address the shortcomings of MLPs.

The convolutional neural network, inspired by the work done in [2], is a type of model that is able to account for local connectivity, enabling it to capture the spatial information embedded in image data [8, 10]. The different types of layers used in a convolutional neural network are shown in Figure 4. The layers are sparsely connected and the unique configuration of weights in a filter or kernel allows for parameter sharing. A direct result of sharing these weights in a given layer is that the filter becomes location invariant and learns to extract a specific pattern from images at that point. One layer may be learning to extract fine edges whereas another might be focused on analyzing curvature. This becomes vary useful for object detection and image recognition where specific items in an image may be placed at different locations in an image frame. It is this property that makes CNNs better suited for tasks within computer vision. CNN models consist of at least three unique layer types: convolutional layers, pooling layers, and dense layers. Artificial neural networks also have hyperparameters which must be selected carefully.

2.1 Convolutional Layer

The use of convolutional layers in a neural network model allows for the capturing of the spatial information present in image data. The convolutional layer uses the information in adjacent pixels to down-sample the image into features, which are then contained in the destination layer. Convolutional layers have filters, also known as kernels, which contain the weights in the neural network. These are trainable model parameters which extract specific features at each convolutional layer. An illustration of a filter is provided in Figure 5. Convolutional layers may sometimes add padding to the source layer, increasing its size prior to performing any operations using the filter. This is done to preserve the size of output. Convolutional layers also have a stride, which specifies how many pixels the

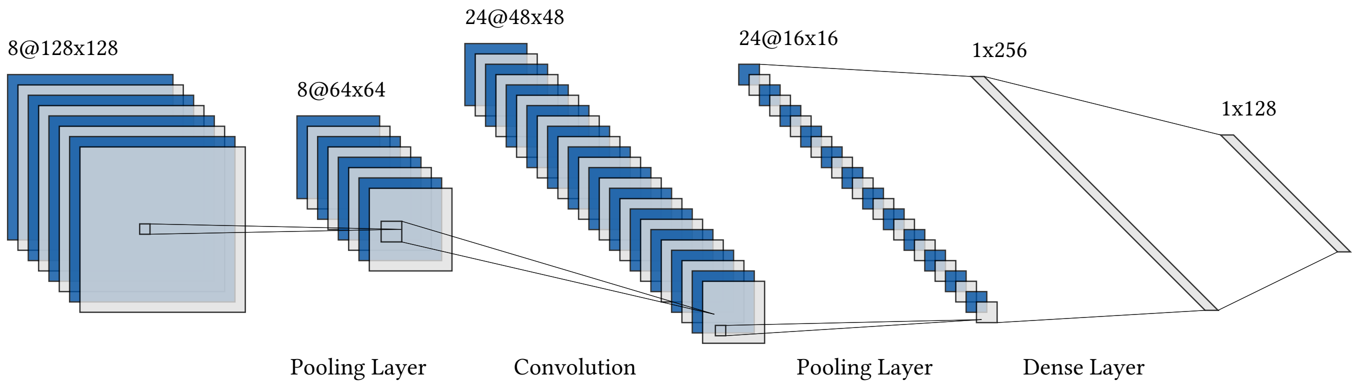


Figure 4: A generic convolutional neural network.

filter show slide before doing the next computation. A stride of 1 indicates that the filter should slide one pixel horizontally and vertically. There may or may not be some type of activation function on these layers depending on the model.

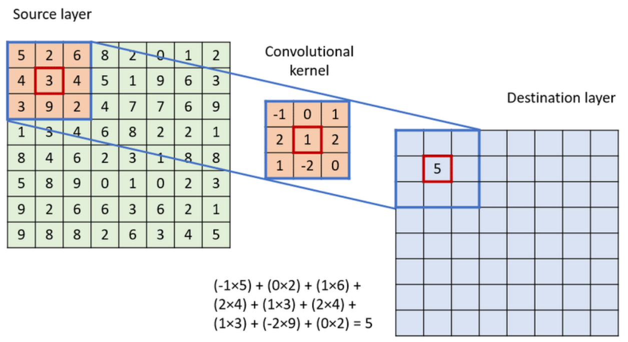


Figure 5: A 3 × 3 convolutional filter operating on a source layer.

2.2 Pooling Layer

The purpose of pooling layers is to reduce the spatial size of the feature extracted by a convolutional layer. The layer also helps reduce over-fitting by producing a more abstract representation of the feature. Instead of using a filter, pooling layers have a sliding kernel which performs a set operation over regions of the image. As a result, these kernels are not trainable model parameters. These kernels also have a stride which is usually specified to be the same size as the kernel used; this results in the pooling operation being applied to mutually exclusive regions of the source layer, unlike the filter in the convolutional layer.

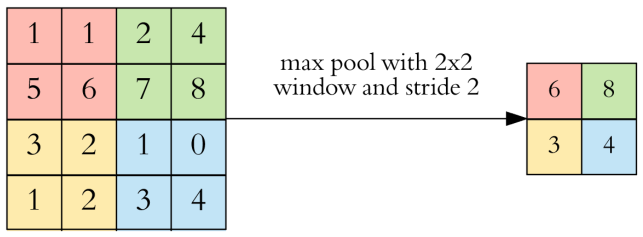


Figure 6: A 2 × 2 max pooling kernel with a stride of 2 operating on a source layer.

Two of the most common pooling operations used are max pooling and average pooling. Max pooling over a region simply extracts the maximum value from the region in the source layer and places it in the destination layer (Figure 6). Average pooling takes the arithmetic mean of all values in the region and places it in the destination layer. It is more common to see max pooling used in modern deep learning models. Again, there may or may not be some type of activation function on these layers depending on the model

2.3 Dense Layer

The dense layers, sometimes referred to as fully connected layers, are the same as those seen in other typical neural network models and in Figure 2. Adding dense layers to a convolutional neural network assists in learning non-linear combinations of the high level features that were extracted by the convolutional layers. Activation functions on these layers provide non-linearity.

2.4 Backward Propagation of Errors

The back-propagation algorithm, short for backward propagation of errors, is widely used to train artificial neural networks [12]. In essence, the algorithm evaluates the expression for the derivative of the loss function as the product of derivatives between each layer contained in the network. If L is the loss function for a given network and w, b represent the weight and bias terms respectively, back-propagation is concerned with the expressions

$$\frac{\partial L}{\partial w} \text{ and } \frac{\partial L}{\partial b}$$

which gives insight into how changing the weights and biases alters the behaviour of the entire model. It provides a way to attribute the loss to the different weights and biases throughout the network, which can than be adjusted in order to optimize the model's loss function.

2.5 Hyperparameters

Unlike model parameters such as the weight and bias terms that are trained in the network, hyperparamters are unique parameters of the actual learning algorithm. These parameters directly influence the ability for the neural network model to learn, and as result, must be chosen and tuned carefully. Despite much research being focused on hyperparameter tuning, there is no model agnostic way to identify the best set of hyperparameters for a given neural network. Much of the tuning of hyperparameters is heuristic, though, past experiences with other models provides some insight into good potential values to choose. Common hyperparameters used to train convolutional neural networks include the number of training epochs, the training batch size, and learning rate used by the optimization algorithm during training. A single epoch is defined as one complete iteration through the entire training data set. The batch size is the number of data instances the model trains on before applying the optimization algorithm to train the model parameters. If the batch size is equal to 1, the loss is computed for a single training instance and model parameters are adjusted according to the loss from that instance; this is done for every single instance in the training data set. If the batch size is $x > 1$, the loss is accumulated over x instances and then the update to the weight and bias terms is applied. If gradient descent is the optimization algorithm used to train the model in the back-propagation process then the following is standard: if $x = 1$ then training is known as stochastic gradient descent, if $1 < x < N$ then training is known as mini-batch gradient descent, and when $x = N$ and N is the total number of training instances the process is known as batch gradient descent. The last hyperparameter that influences the training of the model significantly is the learning rate. This is the rate at which the weights and biases in the model will be updated according to the gradient computed in back-propagation. The learning rate influences how fast a network may converge as well as if it will converge at all, and so it most be chosen carefully according the model architecture and the specific problem being addressed.

3 THE ORIGINAL LENET-5

3.1 Model Parameters

The original LeNet-5 model is comprised of 6 hidden layers, one input layer, and a single output layer (Figure 7). The input layer takes an image of dimension $28 \times 28 \times 1$, that is, an image that is 28 pixels in length, 28 pixels in width, and 1 channel deep indicating it is black and white. The first hidden layer is a convolutional layer which pads the input image with 0s prior to performing any operations. The padding increases the input image dimensions to $32 \times 32 \times 1$, preserving the spatial size of the output after performing operations. This prevents the images from shrinking too fast, which allows deeper convolutional models with more layers to be used if need be. A 5×5 filter is used with a stride of 1 in the layer which results in an output with dimensions $28 \times 28 \times 6$ from the convolutional layer. The next layer is a pooling layer (referred to as sub-sampling in the paper) which takes the $28 \times 28 \times 6$ output from the convolutional layer and uses a 2×2 average pooling sliding window with a stride of 2 in order to produce an output of $14 \times 14 \times 6$. Another convolutional layer is used following this. This layer does not use padding, creating an output of dimensions $10 \times 10 \times 16$, again using a 5×5 filter with a stride of 1. A second pooling layer is used with a 2×2 average pooling sliding window with a stride of 2, producing an output with dimensions $5 \times 5 \times 16$. This output acts as input to one more convolutional layer with a 5×5 filter and stride of 1, resulting in an output of $1 \times 1 \times 120$. The output is then flattened to be a one dimensional vector of length 120. This vector is used as input to the next dense layer of size 84, whose output is then fed into the final layer where a prediction is made. The output layer consists of 10 ouputs, one for each handwritten digit 0–9.

All layers in the original LeNet-5 model except the output layer use a scaled hyperbolic tangent as the activation function:

$$f(\cdot) = 1.7159 \tanh(\cdot)$$

The output layer uses Euclidean Radial Basis Function (RBF) units as activations, with output y_i computed as

$$y_i = \sum_j (x_j - w_{ij})^2$$

where x_j is the output from the perceptron in the previous layer and w_{ij} is the weight connecting perceptron x_j to perceptron y_i. The loss function used in [8] was uniquely developed for the the Euclidean RBF activations used.

3.2 Model Hyperparameters

The LeNet-5 model was trained for a total of 20 epochs. Stochastic gradient descent was used as the optimization algorithm to adjust the trainable model parameters such as the weights and biases. The batch size is set to 1 during the training process, so a single epoch consists of 60000 iterations, one for each training sample. Rather than using a single learning rate through all training epochs, a learning rate schedule was used instead – this can be seen in Table 1.

Table 1: Learning rate schedule

Epoch Range	1–2	3–5	6–8	9–12	13–20
Learning Rate	0.0005	0.0002	0.0001	0.00005	0.00001

3.3 Model Results

The original LeNet-5 model was able to achieve impressive results on the handwritten digit recognition task. After 19 epochs, the training error is approximately 0.35% while the test error is 0.95%. Figure 8 shows the training and testing error from training epoch 1 to 20.

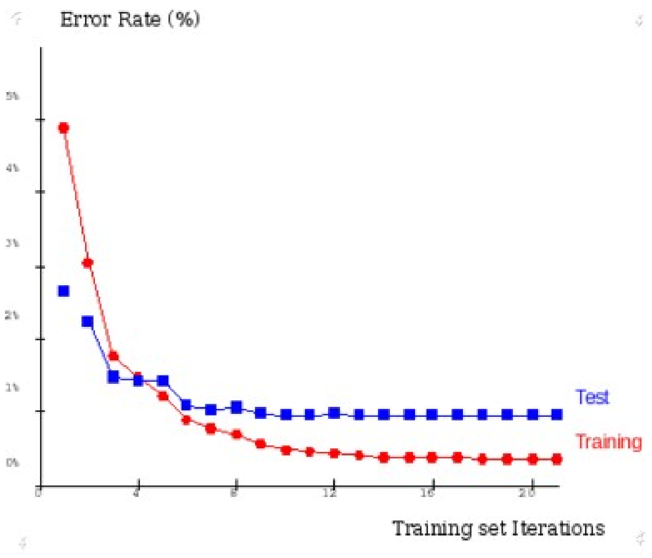


Figure 8: Original LeNet-5 model results.

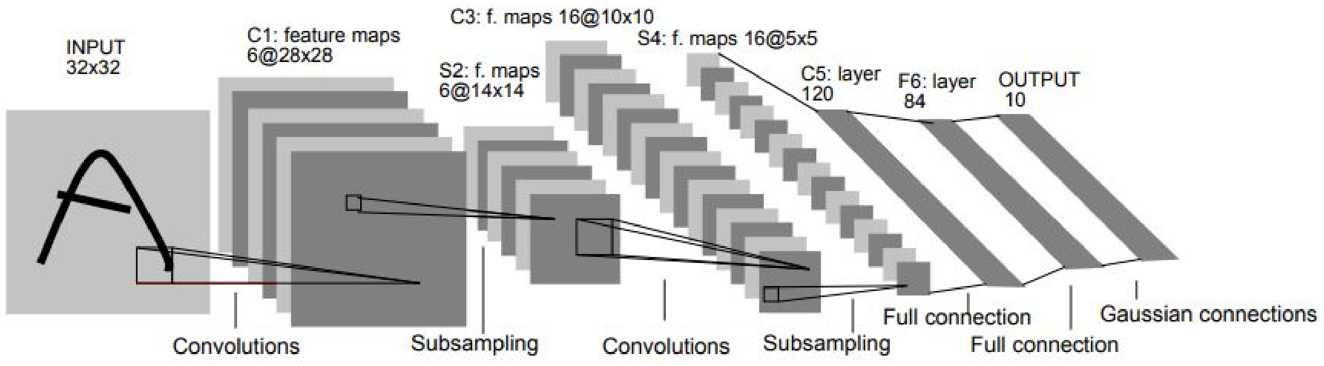


Figure 7: Original LeNet-5 model architecture.

4 MODEL IMPLEMENTATION

The models proposed in this paper, although applied to the same digit classificaiton task, differ from the original LeNet-5 model in a few ways.

4.1 Model 1: LeNet-5 Reproduction

In order to replicate the LeNet-5 architecture, we use the exact same layers as described in Section 3. There are two major deviations from the original model, the first being the activation function used on the final layer, and the second being the loss function used to train the model. A softmax activation function is employed in place of the RBF units used in the original LeNet-5 model. The softmax maps the outputs to a probability distribution where each of the 10 outputs (digits 0–9) takes on a value in $[0, 1]$. The output with the largest probability is the class that is predicted based on the input image. Using softmax simply acts to simplify the activation function and the process of mapping outputs to predictions. The loss function used in this implementation of LeNet-5 is the categorical cross-entropy

$$CE = -\sum_{i}^{C} t_i \log(p_i)$$

where C is the number of classes, t_i is an indicator variable which evaluates to 1 if the i is the true class label for the image and 0 otherwise, and p_i is the probability output of the softmax activation for i^{th} digit. Categorical cross-entropy is often the preferred loss function for multi-class classification tasks.

4.2 Model 2: LeNet-5 with Adam Optimizer

A second LeNet-5 model is implemented in this study. This model incorporates some modern practices used when constructing and training convolutional neural networks. The purpose of developing this model is to assess whether modern alterations provide significant improvements over the original LeNet-5 model in this task.

Similar to the LeNet-5 reproduction, this model also uses the same general architecture and number of layers as the original LeNet-5 model. Softmax activation and a categorical cross-entropy loss are both used as well. The max pooling method is used in the pooling layers instead of average pooling. In place of the scaled hyperbolic tangent activation function, we use the well established and widely used rectified linear unit or ReLu activation function

$$f(\cdot) = \max(0, \cdot)$$

The model's performance is logged over both 10 and 20 epochs to compare the rate of convergence, and the batch size used to train the model is set to 32 images. As a result, it takes 1875 iterations to complete a single training epoch. We use the Adam method as the optimization algorithm in place of stochastic gradient descent [6]. A new learning rate schedule is used with Adam (Table 2).

Table 2: Learning rate schedule

Epoch Range	1–10	11–15	16–20
Learning Rate	0.001	0.0001	0.00001

5 RESULTS

The models implemented in this study were not able to reproduce the exact performance shown by the original LeNet-5 model. Figure 9 displays the error that the LeNet-5 reproduction achieves on the training and testing data over the 20 training epochs completed. By the end of training, the model shows a training error of 0.80% and a testing error of 1.74% which are both significantly higher than the benchmark results referenced in Section 3.3. The behaviour of the training and testing error over the training epoch seem to resemble that found in Figure 8.

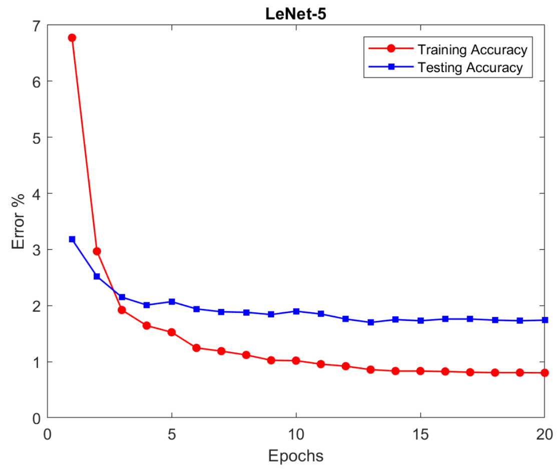


Figure 9: Training and testing results of the LeNet-5 reproduction model.

The LeNet-5 implementation which uses the Adam optimizer shows a better performance than the LeNet-5 reproduction. After the first training epoch the error improves drastically, and following that it seems to plateau showing incremental improvements through the rest of training. The results of this model can be seen in Figure 10. The performance of this model shows very different behaviour than the other models, and this is due to the changes to its architecture, hyperparameters, and optimizer. It seems that this model converges much faster than the others, indicating it is more efficient to train due to the optimizer. We can see that although the training error continues to decrease, the testing error no longer improves by much, so the model is likely over-fitting the training data. The LeNet-5 model with the Adam optimizer is the only model in this study which achieves comparable results to the original model in terms of testing error %. The performance of all models can be seen in Table 3.

Table 3: All model results

	Training Error (%)	Testing Error (%)
Original LeNet-5	0.35	0.95
LeNet-5 Reproduction	0.80	1.74
LeNet-5 with Adam (10 ep.)	0.90	1.52
LeNet-5 with Adam (20 ep.)	0.002	1.00

6 CONCLUSION & SELECTED TOPICS

In this study we implemented two versions of the well known LeNet-5 convolutional neural network model. The first model aims to be a direct reproduction of the original model used in [8]. The second model incorporates elements that are often used in modern deep learning when applied to image classification tasks. Although we were unable to reproduce the exact results achieved by the original LeNet-5 model, the model using an Adam optimizer was able to come close in testing performance. There may be two main reasons as to why the model performance could not be replicated exactly. The first being the stochastic nature of the training process; the weight and bias terms of the neural network are randomly initialized at the beginning of training according to a probability distribution, and this initialization may impact convergence of the model. The second factor may be the loss function. The original LeNet-5 model had used a customized loss function which was developed for the RBF activation units. The minimization of that loss function using gradient descent may have lead to a better model configuration in terms of weight and bias terms than that which was achieved in this study using the categorical cross-entropy loss. Even in light of this, it was shown that the cross-entropy loss along with the Adam optimizer in place of gradient descent led to faster convergence overall. For this specific task, this may not be important since the input data is small and the model itself is not that large, but for a different application within image recognition this faster convergence may be preferred.

To put the problem of handwritten digit recognition into context, a related image recognition system is actually used by the United States Postal Service (USPS). The system is used to automate the scanning of the zip codes written on packages, speeding up sorting time. USPS saw mail volume of approximately 142.6B parcels in the year 2019. Let's assume that if a zip code is misread, its parcel will be shipped to the wrong address. If the model incorporated into this system had an error rate of approximately 1.0%, more than 1.00B parcels would be shipped to incorrect locations. The marginal difference between 0.95% and 1.00% seems small, but the implication of that 0.05% is incredibly large. It is likely that the eventual model used for the task has a lower error rate than those seen in this study as well.

In other application domains unique neural network models have been developed. Just as how a multilayer perceptron network was not best suited to deal with image data in computer vision, a convolutional neural network may not be best suited to deal with sequential data such as time series or natural language. In the following sections we make quick mention of a few other neural network models and their use-cases.

6.1 Long Short-Term Memory Networks

A long short-term memory (LSTM) network is a model which is able to account for the sequential properties of data. They are recurrent in nature which means that the output corresponding to a specific input to the model is fed back into the model along with the next set of input data in order to assist with the next prediction. In this sense, the model has a sort of memory which allows it to capture temporal and sequential relationships. LSTM networks and related models are commonly used with time series data or in the field of natural language processing with text data.

6.2 Generative Adversarial Networks

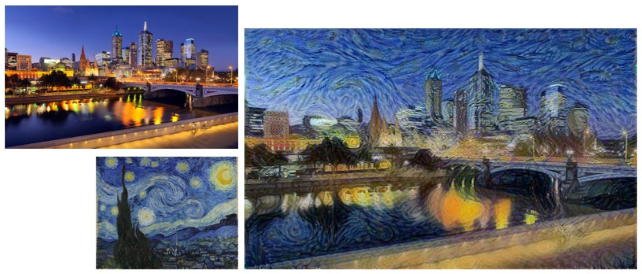
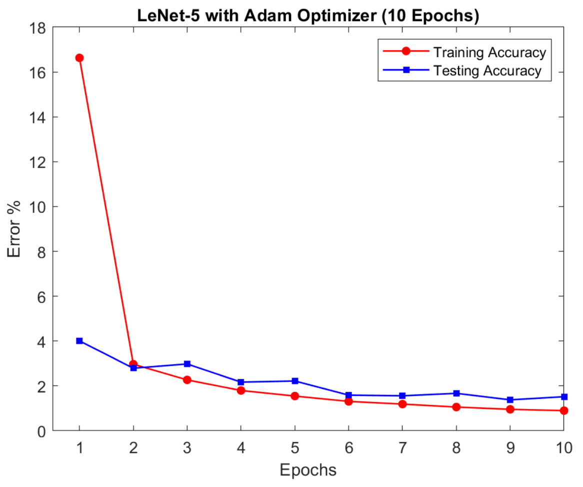
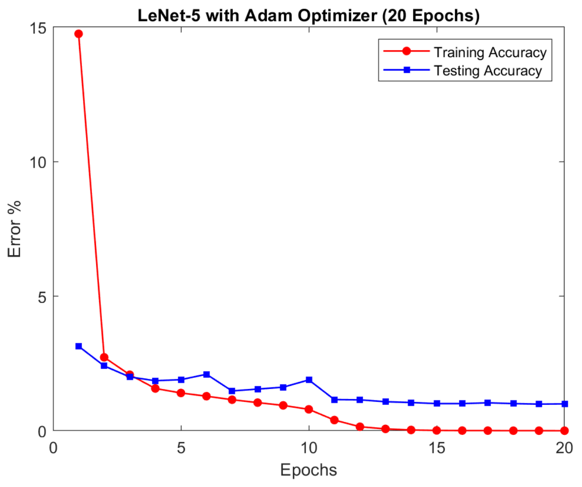


Figure 11: GAN van Gogh, an application of generative adversarial networks.

A generative adversarial network (GANs) can be seen as a framework for neural networks in which they compete to optimize each other at a given task. This competition often happens between two networks in the form of a game (sometimes zero-sum, but not always). GANs are a recent development in the field of deep learning presented in [4]. Usually, one network is a generator whose goal is to learn to model the distribution of the input data such that it can produce synthetic data which seems to be from the true data distribution. The second network is a discriminator whose role is to learn to differentiate between real data from the distribution and the synthetic data which is not actually from the distribution. Through this competition, the generator becomes good enough to produce synthetic data that seems authentic. This has applications in data augmentation and generation, where for some tasks, it may be hard to collect or find enough data to train algorithms. Figure 11 shows an application of GANs used to generate novel artwork



(a) LeNet-5 with Adam trained for 10 epochs.



(b) LeNet-5 with Adam trained for 20 epochs.

Figure 10: Training and testing results of the LeNet-5 model which uses an Adam optimizer.

based on the famous Starry Night piece by Vincent van Gogh [3]. A more recent application of generative adversarial networks can be seen in [1], where researchers used GAN techniques to successfully produce a cryptography system, without the networks having any preconceived notion of cryptography.

6.3 Future Works

The field of deep learning is advancing at a rapid pace, with network architectures and frameworks developed for latent variable modelling, reinforcement learning and robotics, and many other applications. There have been several advances in convolutional neural network models since LeNet-5, such as AlexNet, VGG Net, and Microsoft's ResNet [5, 7, 13]. It seems as though further advances in the field of computer vision may be incredibly impacted by developments in convolutional neural networks as they continue to be studied in depth.

REFERENCES

[1] Martín Abadi and David G Andersen. 2016. Learning to protect communications with adversarial neural cryptography. *arXiv preprint arXiv:1610.06918* (2016).
[2] Kunihiko Fukushima. 1988. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks* 1, 2 (1988), 119–130.
[3] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. 2015. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576* (2015).
[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.
[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[6] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[8] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[9] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/
[10] Yann Lecun, Patrick Haffner, Leon Bottou, and Yoshua Bengio. 1999. Object recognition with gradient-based learning. In *Feature grouping*. Springer.
[11] Frank Rosenblatt. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65, 6 (1958), 386.
[12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.
[13] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).