Handwritten Digit Recognition System using Machine Learning in Python

A Project report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

Submitted by

317126510008	B Sri Sahithi
317126510030	K Sri Pragnya
317126510031	K Rama Krishna
318126510L18	M Lokesh

Under the guidance of

Mrs. SVSS Lakshmi

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(UGC AUTONOMOUS)

(*Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade)*

Sangivalasa, Bheemili mandal, Visakhapatnam dist.(A.P)

2017-2021

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

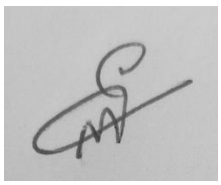
(Affiliated to Andhra University)

SANGIVALASA, VISAKHAPATNAM – 531162

2017-2021

BONAFIDE CERTIFICATE

This is to certify that this project report "**HANDWRITTEN DIGITS RECOGNITIONUSING ML"** is the bona fide work of **B. Sri Sahithi (317126510008), K. Sri Pragnya (317126510030), K. Rama Krishna (317126510031), M. Lokesh (318126510l11) of IV/IV CSE** carried out the project work under my supervision.



S.V.S.S. LAKSHMI
ASSISTANT PROFESSOR
CSE
ANITS

Dr.R.SIVARANJINI
HEAD OF THE DEPARTMENT
CSE
ANITS

ACKNOWLEDGMENT

An endeavour over a long period can be successful with the advice and support of many well-wishers. We take this opportunity to express our gratitude and appreciation to all of them. We owe our tributes to Dr.R.Sivaranjani, Head of the Department, Computer Science & Engineering, ANITS, for her valuable support and guidance during the period of project implementation. We wish to express our sincere thanks and gratitude to our project guide Mrs. S.V.S.S Lakshmi Assistant Professor, Computer Science and Engineering, ANITS, for the simulating discussions, in analysing problems associated with our project work and for guiding us throughout the project. Project meetings were highly informative. We express our warm and sincere thanks for the encouragement, untiring guidance and the confidence they had shown in us. We are immensely indebted for their valuable guidance throughout our project. We also thank all the staff members of CSE department for their valuable advices. We also thank supporting staff for providing resources as and when required.

B.SRI SAHITHI	317126510008	B Sahithi
K.SRI PRAGNYA	317126510030	K S Pragnya
K.RAMAKRISHNA	317126510031	K Rama Krishna
M.LOKESH	318126510L11	M Lokesh

TABLE OF FIGURES

TABLE OF CONTENTS

1. Abstract

Handwritten character recognition is one of the practically important issues in pattern recognition applications. The main purpose of this project is to build an automatic handwritten digit recognition method for the recognition of handwritten digit strings.

To accomplish the recognition task, first, the digits will be segmented into individual digits. Then, a digit recognition module is employed to classify each segmented digit completing the handwritten digit string recognition task.

The applications of digit recognition include postal mail sorting, bank check processing, form data entry, etc. The heart of the problem lies within the ability to develop an efficient algorithm that can recognize handwritten digits and which is submitted by users by the way of a scanner, tablet, and other digital devices.

2. INTRODUCTION

2.1 Introduction

Machine learning and deep learning plays an important role in computer technology and artificial intelligence. With the use of deep learning and machine learning, human effort can be reduced in recognizing, learning, predictions and many more areas.

This article presents recognizing the handwritten digits (0 to 9) from the famous MNIST dataset, comparing classifiers like KNN, PSVM, NN and convolution neural network on basis of performance, accuracy, time, sensitivity, positive productivity, and specificity with using different parameters with the classifiers.

To make machines more intelligent, the developers are diving into machine learning and deep learning techniques. A human learns to perform a task by practicing and repeating it again and again so that it memorizes how to perform the tasks. Then the neurons in his brain automatically trigger and they can quickly perform the task they have learned. Deep learning is also very similar to this. It uses different types of neural network architectures for different types of problems.

For example – object recognition, image and sound classification, object detection, image segmentation, etc.

The handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different flavors. The handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image.

2.1.1 Digit Recognition System:

Digit recognition system is the working of a machine to train itself or recognizing the digits from different sources like emails, bank cheque, papers, images, etc. and in different real-world scenarios for online handwriting recognition on computer tablets or system, recognize number plates of , numeric entries in forms filled up by hand and so on.

2.2 Problem Statement:

The goal of this project is to create a model that will be able to recognize and determine the handwritten digits from its image by using the concepts of Convolution Neural Network. Though the goal is to create a model which can recognize the digits, it can be extended to letters and an individual's handwriting. The major goal of the proposed system is understanding Convolutional Neural Network, and applying it to the handwritten recognition system.

3. LITERATURE SURVEY

An early notable attempt in the area of character recognition research is by Grimsdale in 1959. The origin of a great deal of research work in the early sixties was based on an approach known as analysis-by-synthesis method suggested by Eden in 1968. The great importance of Eden's work was that he formally proved that all handwritten characters are formed by a finite number of schematic features, a point that was implicitly included in previous works.

This notion was later used in all methods in syntactic (structural) approaches of character recognition.

1. **K. Gaurav, Bhatia P. K. ,** his paper deals with the various pre-processing techniques involved in the character recognition with different kind of images ranges from a simple handwritten form based documents and documents containing colored and complex background and varied intensities.In this, different preprocessing techniques like skew detection and correction, image enhancement techniques of contrast stretching, binarization, noise removal techniques, normalization and segmentation, morphological processing techniques are discussed.
2. **Sandhya Arora** , used four feature extraction techniques namely, intersection, shadow feature, chain code histogram and straight line fitting features.

   Shadow features are computed globally for character image while intersection features, chain code histogram features and line fitting features are computed by dividing the character image into different segments.

   On experimentation with a dataset of 4900 samples the overall recognition rate observed was 92.80% for Devanagari characters.

3. **Brakensiek, J. Rottland, A. Kosmala, J. Rigoll**, in their paper a system for off-line cursive handwriting recognition is described which is based on Hidden Markov Models (HMM) using discrete and hybrid modelling techniques.

   Handwriting recognition experiments using a discrete and two different hybrid approaches, which consist of a discrete and semi-continuous structures, are compared.

   It is found that the recognition rate performance can be improved of a hybrid modelling technique for HMMs, which depends on a neural vector quantizer (hybrid MMI), compared to discrete and hybrid HMMs, based on tired mixture structure (hybrid - TP), which may be caused by a relative small data set.

4. **R. Bajaj, L. Dey, S. Chaudhari** , employed three different kinds of features, namely, the density features, moment features and descriptive component features for classification of Devanagari Numerals. They proposed multi classifier connectionist architecture for increasing the recognition reliability and they obtained 89.6% accuracy for handwritten Devanagari numerals.
5. **G. Pirlo and D. Impedovo** in his work on , presented a new class of membership functions, which are called Fuzzymembership functions (FMFs), for zoning-based classification. These FMFs can be easily adapted to the specific characteristics of a classification problem in order to maximize classification performance. In this research, a realcoded genetic algorithm is presented to find, in a single optimization procedure, the optimal FMF, together with the optimal zoning described by Voronoi tessellation. The experimental results, which are carried out in the field of handwritten digit and character recognition, indicate that optimal FMF performs better than other membership functions based on abstract level, ranked-level, and measurement-level weighting models, which can be found in the literature.
6. **Sushree Sangita Patnaik and Anup Kumar Panda** May 2011 , this paper proposes the implementation of particle swarm optimization (PSO) and bacterial foraging optimization (BFO) algorithms which are intended for optimal harmonic compensation by minimizing the undesirable losses occurring inside the APF itself. The efficiency and effectiveness of the implementation of two approaches are compared for two different conditions of supply. The total harmonic distortion (THD) in the source current which is a measure of APF performance is reduced drastically to nearly 1% by employing BFO. The results demonstrate that BFO outperforms the conventional and PSO based approaches by ensuring excellent functionality of APF and quick prevail over harmonics in the source current even under unbalanced supply.
7. **M. Hanmandlu, O.V. Ramana Murthy** have presented in their study the recognition of handwritten Hindi and English numerals by representing them in the form of exponential membership functions which serve as a fuzzy model. The recognition is carried out by modifying the exponential membership functions fitted to the fuzzy sets. These fuzzy sets are derived from features consisting of normalized distances obtained using the Box approach. The membership function is modified by two structural parameters that are estimated by optimizing the entropy subject to the attainment of membership function to unity. The overall recognition rate is found to be 95% for Hindi numerals and 98.4% for English numerals.

8. **Renata F. P. Neves** have proposed SVM based offline handwritten digit recognition. Authors claim that SVM outperforms the Multilayer perceptron classifier. Experiment is

carried out on NIST SD19 standard dataset. Advantage of MLP is that it is able to segment non-linearly separable classes. However, MLP can easily fall into a region of local minimum, where the training will stop assuming it has achieved an optimal point in the error surface. Another hindrance is defining the best network architecture to solve the problem, considering the number of layers and the number of perceptron in each hidden layer. Because of these disadvantages, a digit recognizer using the MLP structure may not produce the desired low error rate.

4. UML DIAGRAMS

A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of **visually representing a system** along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system. UML is an acronym that stands for **Unified Modelling Language**. Simply put, UML is a modern approach to modelling and documenting software. In fact, it's one of the most popular business process modelling techniques. It is based on **diagrammatic representations** of software components. As the old proverb says: "a picture is worth a thousand words". By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

4.1 Class Diagram:

Class diagram model class structure and contents using design elements such as classes, packages and objects. Class diagram describes 3 perspectives when designing a system Conceptual, Specification, Implementation. Classes are composed of three things: name, attributes and operations. Class diagrams also display relations such as containment, inheritance, associations etc. The association relationship is most common relationship in a class diagram. The association shows the relationship between instances of classes. The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction. UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as −

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

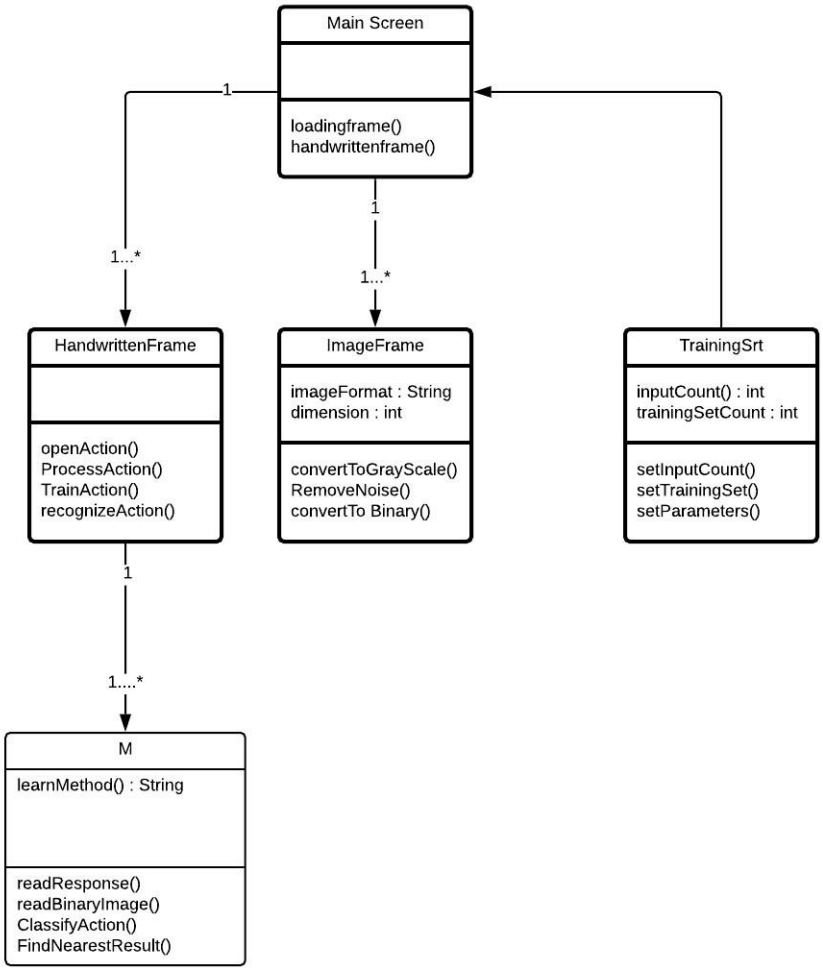


Figure 1 Class Diagram

4.2 Use Case Diagram:

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware. UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components. Most of the UML diagrams are used to handle logical components of the system.

A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. A use-case diagram can help provide a higher-level view of the system. Use-Case provide the simplified and graphical representation of what the system must actually do.

In software and systems engineering, a use case is a list of actions or event steps typically defining the interactions between a role known in the Unified Modeling Language (UML) as an actor and a system to achieve a goal. The actor can be a human or other external system. In systems engineering, use cases are used at a higher level than within software engineering. The detailed requirements may then be captured in the Systems Modeling Language. Use case analysis is an important and valuable requirement analysis technique that has been widely used in modern software engineering.

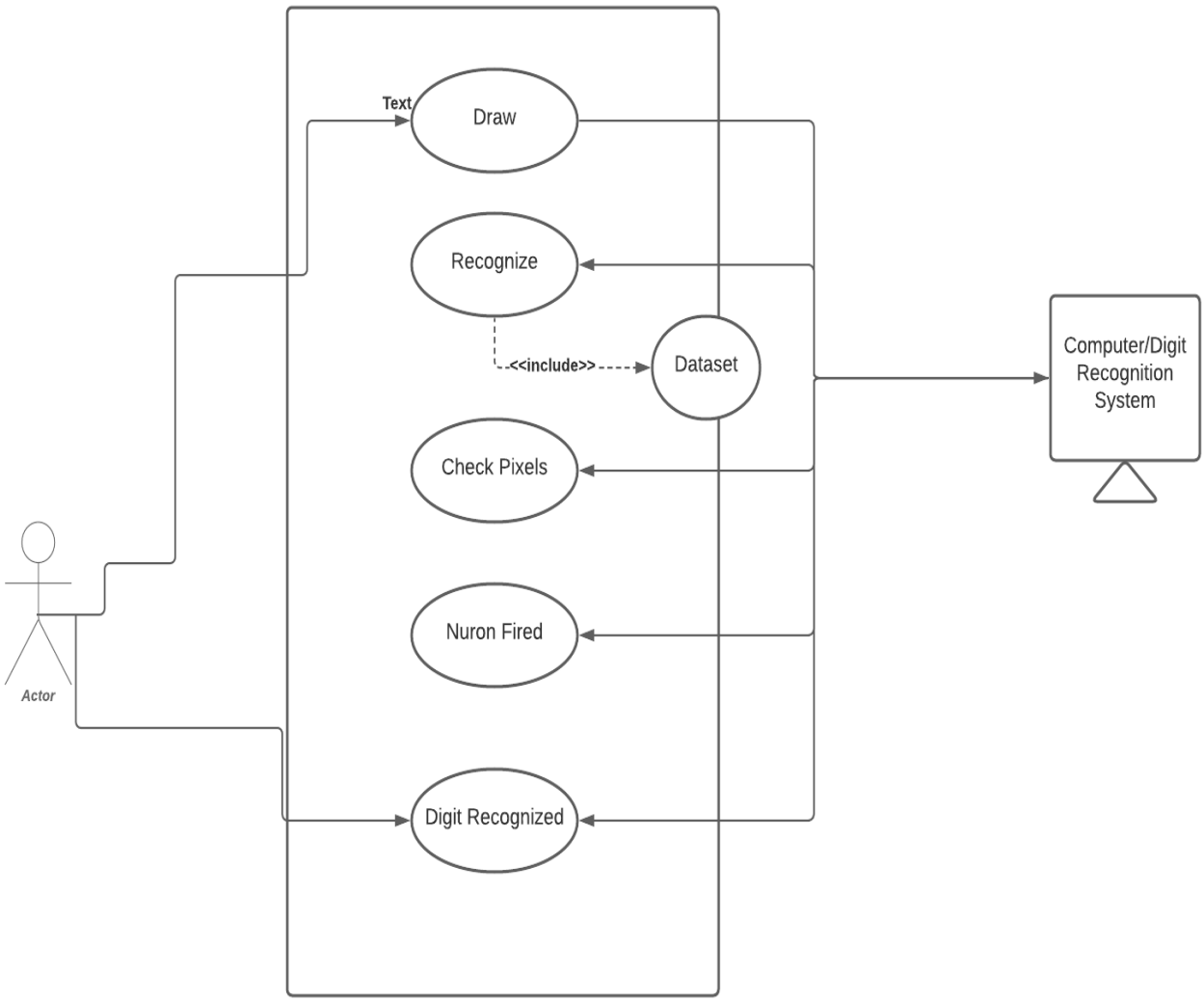


Figure 2 Use Case Diagram

5. METHODOLOGY

5.1 Basic steps in constructing a Machine Learning model:

5.1.1 - Data Collection

- The quantity & quality of your data dictate how accurate our model is
- The outcome of this step is generally a representation of data (Guo simplifies to specifying a table) which we will use for training
- Using pre-collected data, by way of datasets from Kaggle, UCI, etc., still fits into this step

5.1.2 - Data Preparation

- Wrangle data and prepare it for training
- Clean that which may require it (remove duplicates, correct errors, deal with missing values, normalization, data type conversions, etc.)
- Randomize data, which erases the effects of the particular order in which we collected and/or otherwise prepared our data
- Visualize data to help detect relevant relationships between variables or class imbalances (bias alert!), or perform other exploratory analysis
- Split into training and evaluation sets

5.1.3 - Choose a Model

- Different algorithms are for different tasks; choose the right one

5.1.4 - Train the Model

- The goal of training is to answer a question or make a prediction correctly as often as possible
- Linear regression example: algorithm would need to learn values for m (or W) and b (x is input, y is output)
- Each iteration of process is a training step

5.1.5 - Evaluate the Model

- Uses some metric or combination of metrics to "measure" objective performance of model
- Test the model against previously unseen data

- This unseen data is meant to be somewhat representative of model performance in the real world, but still helps tune the model (as opposed to test data, which does not)
- Good train/eval split? 80/20, 70/30, or similar, depending on domain, data availability, dataset particulars, etc.

5.1.6 - Parameter Tuning

- This step refers to *hyperparameter* tuning, which is an "artform" as opposed to a science
- Tune model parameters for improved performance
- Simple model hyperparameters may include: number of training steps, learning rate, initialization values and distribution, etc.

5.1.7 - Make Predictions

- Using further (test set) data which have, until this point, been withheld from the model (and for which class labels are known), are used to test the model; a better approximation of how the model will perform in the real world

5.2 Methodologies for Handwritten Digit Recognition System

We used MNIST as a primary dataset to train the model, and it consists of 70,000 handwritten raster images from 250 different sources out of which 60,000 are used for training, and the rest are used for training validation. Our proposed method mainly separated into stages, preprocessing, Model Construction, Training & Validation, Model Evaluation & Prediction. Since the loading dataset is necessary for any process, all the steps come after it.

Figure 3 Steps in System development

5.2.1 Import the libraries:

Libraries required are Keras, Tensor flow, Numpy, Pillow, Tkinkter.

1. Keras:

Keras is a powerful and easy-to-use free open source Python library for developing and evaluating **deep learning** models**.**

It wraps the efficient numerical computation libraries **Theano** and **TensorFlow** and allows you to define and train neural network models in just a few lines of code. It uses libraries such as Python, C#, C++ or standalone machine learning toolkits. Theano and TensorFlow are very powerful libraries but difficult to understand for creating neural networks.

Keras is based on minimal structure that provides a clean and easy way to create deep learning models based on TensorFlow or Theano. Keras is designed to quickly define deep learning models. Well, Keras is an optimal choice for deep learning applications.

2. TensorFlow:

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of **TensorFlow.** TensorFlow tutorial is designed for both beginners and professionals. Our tutorial provides all the basic and advanced concept of machine learning and deep learning concept such as deep neural network, image processing and sentiment analysis. TensorFlow is one of the famous deep learning frameworks, developed by **Google** Team. It is a free and open source software library and designed in **Python** programming language, this tutorial is designed in such a way that we can easily implements deep learning project on TensorFlow in an easy and efficient way. Unlike other numerical libraries intended for use in Deep Learning like **Theano**, **TensorFlow** was designed for use both in research and development and in production systems. It can run on single CPU systems, GPUs as well as mobile devices and largescale distributed systems of hundreds of machines.

3. Numpy:

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. Numpy which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc. It is an open source project and you can use it freely. NumPy stands for Numerical Python. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called **ndarray**, it provides a lot of supporting functions that make working with **ndarray** very easy. Arrays are very frequently used in data science, where speed and resources are very important.

4. Pillow:

Pillow is a free and open source library for the Python programming language that allows you to easily create &s manipulate digital images. Pillow is built on top of PIL (Python Image Library). PIL is one of the important modules for image processing in Python. However, the PIL module is not supported since 2011 and doesn't support python 3.

Pillow module gives more functionalities, runs on all major operating system and support for python 3. It supports wide variety of images such as "jpeg", "png", "bmp", "gif", "ppm", "tiff". You can do almost anything on digital images using pillow module. Apart from basic image processing functionality, including point operations, filtering images using built-in convolution kernels, and color space conversions.

5. Tkinkter:

Tkinter is the standard **GUI library** for Python. Python when combined with Tkinter provides a fast and easy way to create **GUI applications**. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

We need to import all the modules that we are going to need for training our model. The Keras library already contains some datasets and MNIST is one of them. So we can easily import the dataset through Keras. The mnist.load_data() method returns the training data, its labels along with the testing data and its labels.

5.3 Loading The Data Set:

5.3.1 MNIST Data Set:

Modified National Institute of Standards and Technology (MNIST) is a large set of computer vision dataset which is extensively used for training and testing different systems. It was created from the two special datasets of National Institute of Standards and Technology (NIST) which holds
binary images of handwritten digits. The training set contains handwritten digits from 250 people, among them 50% training dataset was employees from the Census Bureau and the rest of it was from high school students. However, it is often attributed as the first datasets among
other datasets to prove the effectiveness of the neural networks.

Figure 4 MNIST Data Set

The database contains 60,000 images used for training as well as few of them can be used for cross-validation purposes and 10,000 images used for testing. All the digits are grayscale and positioned in a fixed size where the intensity lies at the center of the image with 28×28 pixels. Since all the images are 28×28 pixels, it forms an array which can be flattened into 28*28=784 dimensional vector. Each component of the vector is a binary value which describes the intensity of the pixel.

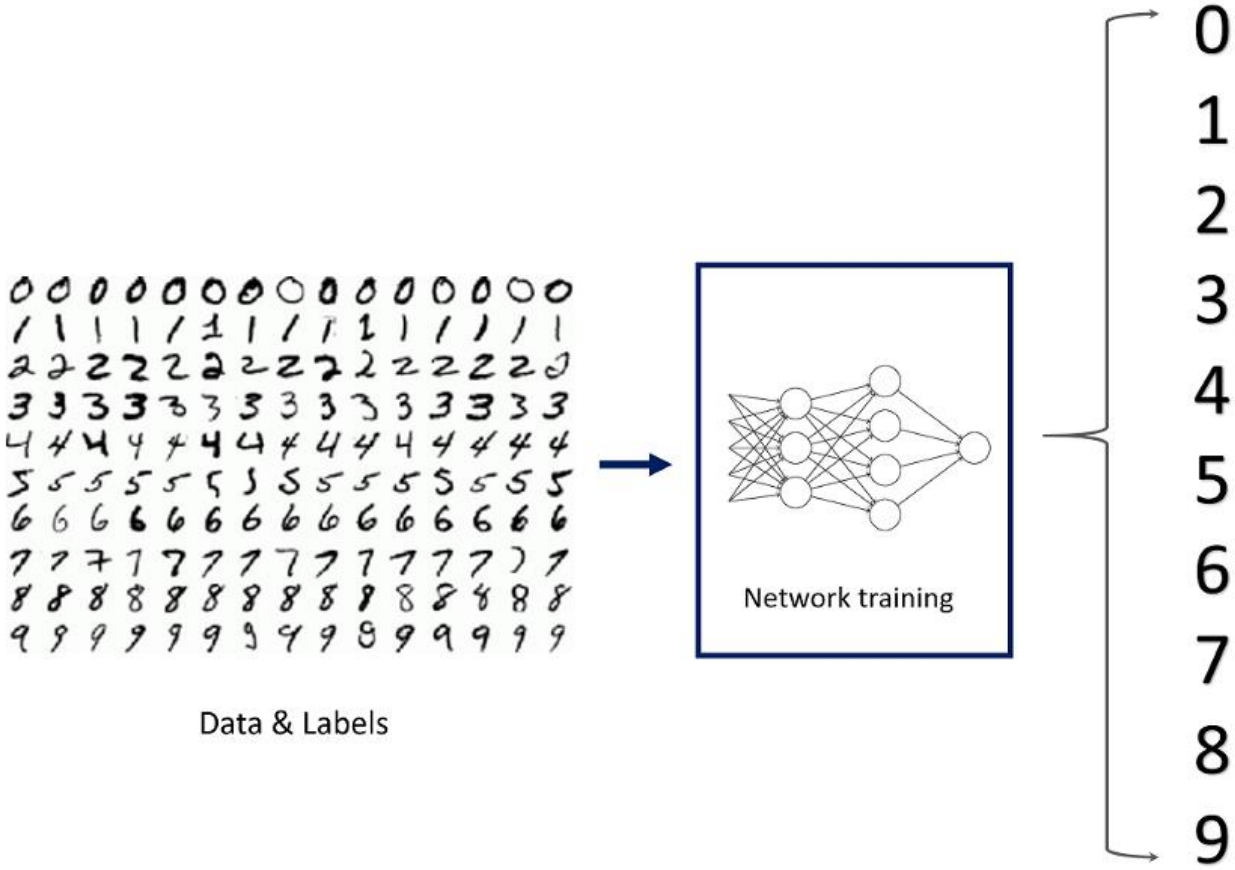


Figure 5 MNIST Examole

5.4 Pre-Processing

Data pre-processing plays an important role in any recognition process. Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format. To shape the input images in a form suitable for segmentation pre-processing is used. Data preprocessing is a necessary step before building a model with these features. It usually happens in stages.

- Data quality assessment
- Data cleaning
- Data transformation
- Data reduction

5.4.1 Data quality assessment:

A Data Quality Assessment is a distinct phase within the data quality life-cycle that is used to verify the source, quantity and impact of any data items that breach pre-defined data quality rules. The Data Quality Assessment can be executed as a one-off process or repeatedly as part of an ongoing data quality assurance initiative.

The quality of your data can quickly decay over time, even with stringent data capture methods cleaning the data as it enters your database. People moving house, changing phone numbers and passing away all mean the data you hold can quickly become out of date.

A Data Quality Assessment helps to identify those records that have become inaccurate, the potential impact that inaccuracy may have caused and the data's source. Through this assessment, it can be rectified and other potential issues identified.

5.4.2 Data cleaning:

Data cleaning is one of the important parts of machine learning. It plays a significant part in building a model. It surely isn't the fanciest part of machine learning and at the same time, there aren't any hidden tricks or secrets to uncover. However, proper data cleaning can make or break your project. Professional data scientists usually spend a very large portion of their time on this step. Because of the belief that, "Better data beats fancier algorithms". If we have a well-cleaned dataset, we can get desired results even with a very simple algorithm, which can prove very beneficial at times. Obviously, different types of data will require different types of cleaning. However, this systematic approach can always serve as a good starting point.

5.4.3 Data transformation

In fact, by cleaning and smoothing the data, we have already performed data modification. However, by data transformation, we understand the methods of turning the data into an appropriate format for the computer to learn from. Data transformation is the process in which data is taken from its raw, siloed and normalized source state and transform it into data that's joined together, dimensionally modelled, de-normalized, and ready for analysis. Without the right technology stack in place, data transformation can be time-consuming, expensive, and tedious. Nevertheless, transforming the data will ensure maximum data quality which is imperative to gaining accurate analysis, leading to valuable insights that will eventually empower data-driven decisions.

Building and training models to process data is a brilliant concept, and more enterprises have adopted, or plan to deploy, machine learning to handle many practical applications. But for models to learn from data to make valuable predictions, the data itself must be organized to ensure its analysis yield valuable insights.

5.4.4 Data reduction:

Data reduction is a process that reduced the volume of original data and represents it in a much smaller volume. Data reduction techniques ensure the integrity of data while reducing the data. The time required for data reduction should not overshadow the time saved by the data mining on the reduced data set.

Data Reduction Techniques:

Techniques of data deduction include dimensionality reduction, numerosity reduction and data compression.

1. Dimensionality Reduction:
    a. Wavelet Transform
    b. Principal Component Analysis
    c. Attribute Subset Selection

2. Numerosity Reduction:
    a. Parametric
    b. Non-Parametric

3. Data Compression:

When you work with large amounts of data, it becomes harder to come up with reliable solutions. Data reduction can be used to reduce the amount of data and decrease the costs of analysis. After loading

the data, we separated the data into X and y where X is the image, and y is the label corresponding to X. The first layer/input layer for our model is convolution. Convolution takes each pixel as a neuron, so we need to reshape the images such that each pixel value is in its own space, thus converting a 28x28 matrix of greyscale values into 28x28x1 tensor. With the right dimensions for all the images, we can split the images into train and test for further steps.

After loading the data, we separated the data into X and y where X is the image, and y is the label corresponding to X.

The first layer/input layer for our model is convolution.

Convolution takes each pixel as a neuron, so we need to reshape the images such that each pixel value is in its own space, thus converting a 28x28 matrix of greyscale values into 28x28x1 tensor. With the right dimensions for all the images, we can split the images into train and test for further steps.

5.5 Data Encoding:

This is an optional step since we are using the cross-categorical entropy as loss function. We have to specify the network that the given labels are categorical in nature. The raw data can contain various different types of data which can be both structured and unstructured and needs to be processed in order to bring to form that is usable in the Machine Learning models. Since machine learning is based on mathematical equations, it would cause a problem when we keep categorical variables as is. Many algorithms support categorical values without further manipulation, but in those cases, it's still a topic of discussion on whether to encode the variables or not. After the identification of the data types of the features present in the data set, the next step is to process the data in a way that is suitable to put to Machine Learning models. The three popular techniques of converting Categorical values to Numeric values are done in two different methods.

1. Label Encoding.
2. One Hot Encoding.
3. Binary Encoding.

Encoding variability describes the variation of encoding of individually inside a category. When we talk about the variability in one hot encoding, the variability depends on the time of implementation in which it decides the number of categories to take that do have sufficient impact on the target. Other encoding methodologies do show a significant variability which is identified at the time of validation.

5.6 Model Construction

Now, comes the fun part where we finally get to use the meticulously prepared data for model building. Depending on the data type (qualitative or quantitative) of the target variable (commonly referred to as the Y variable) we are either going to be building a classification (if Y is qualitative) or regression (if Y is quantitative) model.

5.6.1 Learning Algorithms : Machine learning algorithms could be broadly categorised to one of three types:

1. Supervised learning — In supervised learning, each example is a *pair* consisting of an input object (typically a vector) and a desired output value (also called the *supervisory signal*). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances.

It is a machine learning task that establishes the mathematical relationship between input X and output Y variables. Such X, Y pair constitutes the labeled data that are used for model building in an effort to learn how to predict the output from the input. Supervised learning problems can be further grouped into regression and classification problems.

- **Classification**: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
- **Regression**: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

2. Unsupervised learning — is a machine learning task that makes use of only the input X variables. Such X variables are unlabeled data that the learning algorithm uses in modeling the inherent structure of the data. Unsupervised learning problems can be further grouped into clustering and association problems.

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.

- **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

4. Reinforcement learning — Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behaviour or path it should take in a specific situation.

   Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task.

   In the absence of a training dataset, it is bound to learn from its experience. It is a machine learning task that decides on the next course of action and it does this by learning through trial and error in an effort to maximize the reward.

   - Input: The input should be an initial state from which the model will start
   - Output: There are many possible output as there are variety of solution to a particular problem
   - Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
   - The model keeps continues to learn.
   - The best solution is decided based on the maximum reward.

5.6.2 MODELS THAT CAN BE USED FOR THE PROJECT:

1. SUPPORT VECTOR MACHINE:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat.

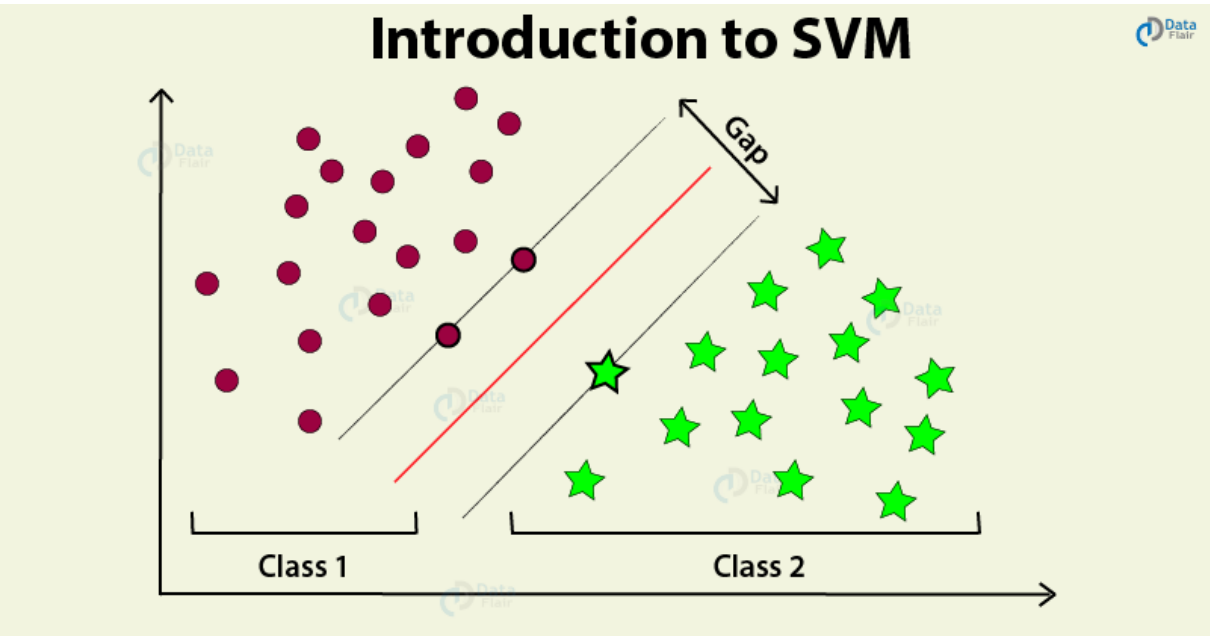


Figure 6 SVM working graph

The followings are important concepts in SVM −

- **Support Vectors** − Datapoints that are closest to the hyperplane is called support vectors. Separating line will be defined with the help of these data points.
- **Hyperplane** − As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.
- **Margin** − It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

The main goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH) and it can be done in the following two steps −

- First, SVM will generate hyperplanes iteratively that segregates the classes in best way.
- Then, it will choose the hyperplane that separates the classes correctly.

Pros of SVM classifiers

- SVM classifiers offers great accuracy and work well with high dimensional space. SVM classifiers basically use a subset of training points hence in result uses very less memory.

Cons of SVM classifiers

- They have high training time hence in practice not suitable for large datasets. Another disadvantage is that SVM classifiers do not work well with overlapping classes.

2. K-NN ALGORITHM:

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

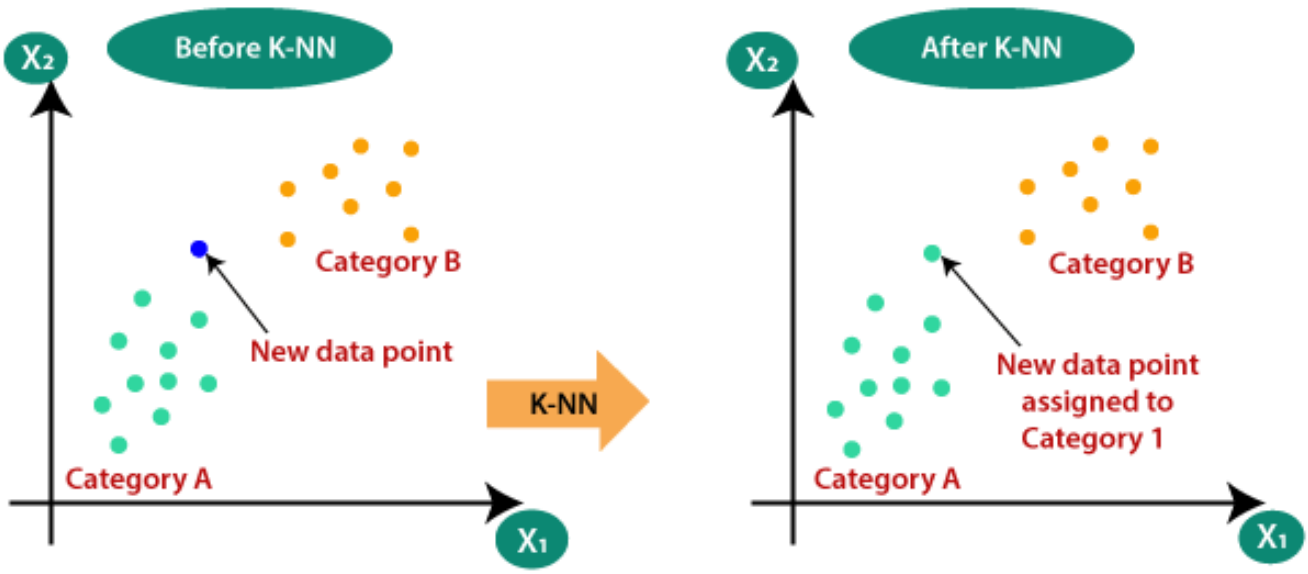


Figure 7 KNN

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbours
- **Step-2:** Calculate the Euclidean distance of **K number of Neighbours**
- **Step-3:** Take the K nearest Neighbours as per the calculated Euclidean distance.
- **Step-4:** Among these k Neighbours, count the number of the data points in each category.

- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data.
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Steps to implement the K-NN algorithm:

- Data Pre-processing step
- Fitting the K-NN algorithm to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

This is pseudocode for implementing the KNN algorithm from scratch:

1. Load the training data.
2. Prepare data by scaling, missing value treatment, and dimensionality reduction as required.
3. Find the optimal value for K:
4. Predict a class value for new data:
    1. Calculate distance (X, Xi) from i =1, 2, 3,….,n. where X= new data point, Xi= training data, distance as per your chosen distance metric.
    2. Sort these distances in increasing order with corresponding train data.
    3. From this sorted list, select the top ‘K’ rows.

- Find the most frequent class from these chosen ‘K’ rows. This will be your predicted class.

After data encoding, the images and labels are ready to be fitted into our model. We need to define a **Convolutional Neural Network Model**.

3.CONVOLUTION NEURAL NETWORK:

In simpler words, CNN is an artificial neural network that specializes in picking out or detect patterns and make sense of them. Thus, CNN has been most useful for image classification. A CNN model has various types of filters of different sizes and numbers. These filters are essentially what helps us in detecting the pattern. The convolutional neural network, or CNN for short, is a specialized type of neural network model designed for working with two-dimensional image data, although they can be used with one-dimensional and three-dimensional data.

Central to the convolutional neural network is the convolutional layer that gives the network its name. This layer performs an operation called a "*convolution*".

A CNN model generally consists of convolutional and pooling layers. It works better for data that are represented as grid structures, this is the reason why CNN works well for image classification problems. The dropout layer is used to deactivate some of the neurons and while training, it reduces offer fitting of the model. Our model is composed of feature extraction with convolution and binary classification. Convolution and max pooling are carried out to extract the features in the image, and a 32 3x3 convolution filters are applied to a 28x28 image followed by a max-pooling layer of 2x2 pooling size followed by another convolution layer with 64 3x3 filters.

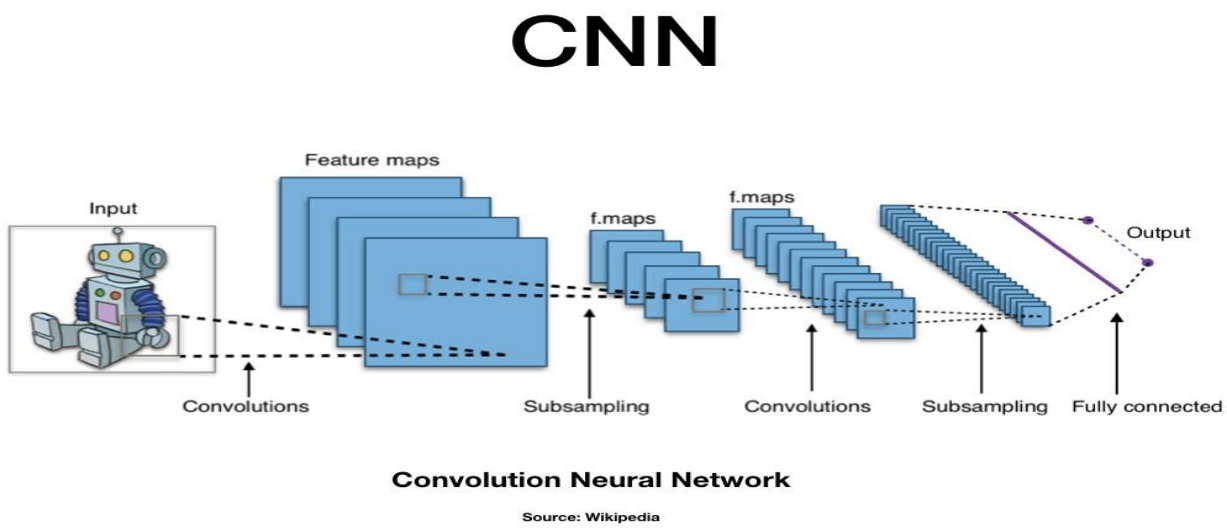


Figure 8 CNN Architecture

In the end, we obtain 7x7 images to flatten. Flatten layer will flatten the 7x7 images into a series of 128 values that will be mapped to a dense layer of 128 neurons that are connected to the categorical output layer of 10 neurons.

The filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product. A dot product is the element-wise multiplication between the filter-sized patch of the input and filter, which is then summed, always resulting in a single value. Because it results in a single value, the operation is often referred to as the *"*scalar product*"* Using a filter smaller than the input is intentional as it allows the same filter (set of weights) to be multiplied by the input array multiple times at different points on the input. Specifically, the filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right, top to bottom.

The output from multiplying the filter with the input array one time is a single value. As the filter is applied multiple times to the input array, the result is a two-dimensional array of output values that represent a filtering of the input. As such, the two-dimensional output array from this operation is called a *"*feature map*"*.

Convolution in Computer Vision:

The idea of applying the convolutional operation to image data is not new or unique to convolutional neural networks; it is a common technique used in computer vision.

Historically, filters were designed by hand by computer vision experts, which were then applied to an image to result in a feature map or output from applying the filter then makes the analysis of the image easier in some way. The network will learn what types of features to extract from the input. Specifically, training under stochastic gradient descent, the network is forced to learn to extract features from the image that minimize the loss for the specific task the network is being trained to solve, e.g. extract features that are the most useful for classifying images as dogs or cats.

Worked Example of Convolutional Layers

The Keras deep learning library provides a suite of convolutional layers.

We can better understand the convolution operation by looking at some worked examples with contrived data and handcrafted filters.

The one-dimensional convolutional layer and a two-dimensional convolutional layer example to both make the convolution operation concrete and provide a worked example of using the Keras layers.

- Convolutional neural networks apply a filter to an input to create a feature map that summarizes the presence of detected features in the input.
- Filters can be handcrafted, such as line detectors, but the innovation of convolutional neural networks is to learn the filters during training in the context of a specific prediction problem.

WORKING OF CNN:

Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value.

The behaviour of each neuron is defined by its weights. When fed with the pixel values, the artificial neurons of a CNN pick out various visual features.

When you input an image into a ConvNet, each of its layers generates several activation maps. Activation maps highlight the relevant features of the image. Each of the neurons takes a patch of pixels as input, multiplies their colour values by its weights, sums them up, and runs them through the activation function.

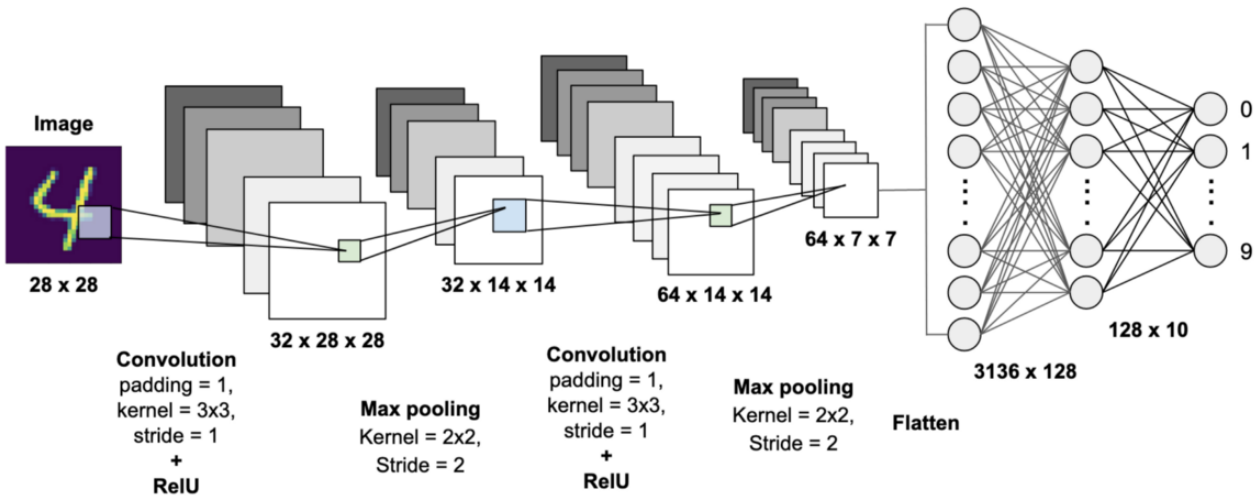


Figure 9 CNN for handwritten digit recognition

The first (or bottom) layer of the CNN usually detects basic features such as horizontal, vertical, and diagonal edges. The output of the first layer is fed as input of the next layer, which extracts more complex features, such as corners and combinations of edges. As you move deeper into the convolutional neural network, the layers start detecting higher-level features such as objects, faces, and more.

The operation of multiplying pixel values by weights and summing them is called "convolution" (hence the name convolutional neural network). A CNN is usually composed of several convolution layers, but it also contains other components. The final layer of a CNN is a classification layer, which takes the output of the final convolution layer as input (remember, the higher convolution layers detect complex objects).

Based on the activation map of the final convolution layer, the classification layer outputs a set of confidence scores (values between 0 and 1) that specify how likely the image is to belong to a "class." For instance, if you have a ConvNet that detects cats, dogs, and horses, the output of the final layer is the possibility that the input image contains any of those animals.

After selecting the model the following process is done:

The model type that we will be using is Sequential. Sequential is the easiest way to build a model in Keras. It allows you to build a model layer by layer.

We use the 'add()' function to add layers to our model.

Our first 2 layers are Conv2D layers. These are convolution layers that will deal with our input images, which are seen as 2-dimensional matrices.

64 in the first layer and 32 in the second layer are the number of nodes in each layer. This number can be adjusted to be higher or lower, depending on the size of the dataset. In our case, 64 and 32 work well, so we will stick with this for now.

Kernel size is the size of the filter matrix for our convolution. So a kernel size of 3 means we will have a 3x3 filter matrix. Refer back to the introduction and the first image for a refresher on this.

Activation is the activation function for the layer. The activation function we will be using for our first 2 layers is the ReLU, or Rectified Linear Activation. This activation function has been proven to work well in neural networks.

Our first layer also takes in an input shape. This is the shape of each input image, 28,28,1 as seen earlier on, with the 1 signifying that the images are greyscale.

In between the Conv2D layers and the dense layer, there is a 'Flatten' layer. Flatten serves as a connection between the convolution and dense layers.

'Dense' is the layer type we will use in for our output layer. Dense is a standard layer type that is used in many cases for neural networks.

We will have 10 nodes in our output layer, one for each possible outcome (0–9).

The activation is 'softmax'. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.

5.7 Training & Validation

After the construction of the model the model has to be compiled to train it with the available data set. Optimizers are used to compile the model. Compiling the model takes three parameters: optimizer, loss and metrics. Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function.

The optimizer controls the learning rate. We will be using 'adam' as our optmizer. Adam is generally a good optimizer to use for many cases. The adam optimizer adjusts the learning rate throughout training. The learning rate determines how fast the optimal weights for the model are calculated. A smaller learning rate may lead to more accurate weights (up to a certain point), but the time it takes to compute the weights will be longer.

We will use 'categorical_crossentropy' for our loss function. This is the most common choice for classification. A lower score indicates that the model is performing better. To make things even easier to interpret, we will use the 'accuracy' metric to see the accuracy score on the validation set when we train the model. The idea behind training and testing any data model is to achieve maximum learning rate and maximum validation. Better Learning rate and better validation can be achieved by increasing the train and test data respectively.

Once the model is successfully assembled, then we can train the model with training data for 100 iterations, but as the number of iteration increases, there is a chance for overfitting. Therefore we limit the training up to 98% accuracy, as we are using real-world data for prediction, test data was used to validate the model.

Different optimizers used in Neural Networks are:

1. Gradient Descent
2. Stochastic Gradient Descent (SGD)
3. Mini Batch Stochastic Gradient Descent (MB-SGD)
4. SGD with momentum
5. Nesterov Accelerated Gradient (NAG)
6. Adaptive Gradient (AdaGrad)
7. AdaDelta
8. RMSprop
9. Adam

5.7.1 ADAM Optimizer

Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm. The **Adam optimization algorithm** is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper (poster) titled "Adam: A Method for Stochastic Optimization".
The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically:

- **Adaptive Gradient Algorithm** (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer

vision problems). Adaptive Moment Estimation is most popular today. ADAM computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients vt like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients mt, similar to momentum

- **Root Mean Square Propagation** (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

Properties of Adam:

1. Actual step size taken by the Adam in each iteration is approximately bounded the step size hyper-parameter. This property add intuitive understanding to previous unintuitive learning rate hyper-parameter.

2. Step size of Adam update rule is invariant to the magnitude of the gradient, which helps a lot when going through areas with tiny gradients (such as saddle points or ravines). In these areas SGD struggles to quickly navigate through them.

3. Adam was designed to combine the advantages of Adagrad, which works well with sparse gradients, and RMSprop, which works well in on-line settings. Having both of these enables us to use Adam for broader range of tasks. Adam can also be looked at as the combination of RMSprop and SGD with momentum.

Why ADAM?

1. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.
2. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.
3. Adam is relatively easy to configure where the default configuration parameters do well on most problems.

5.8 Model Evaluation & Prediction:

For real-world image classification prediction, we need to do a little image pre-processing on the real-world images as model training was done with greyscale raster images. The steps of image pre-processing are :

1. Loading image
2. Convert the image to greyscale
3. Resize the image to 28x28
4. Converting the image into a matrix form
5. Reshape the matrix into 28x28x1

After pre processing, we predict the label of the image by passing the pre-processed image through the neural network. The output we get is a list of 10 activation values 0 to 9, respectively. The position having the highest value is the predicted label for the image.

These structures are called as Neural Networks. It teaches the computer to do what naturally comes to humans. Deep learning, there are several types of models such as the Artificial Neural Networks (ANN), Autoencoders, Recurrent Neural Networks (RNN) and Reinforcement Learning. But there has been one particular model that has contributed a lot in the field of computer vision and image analysis which is the Convolutional Neural Networks (CNN) or the ConvNet.
CNNs are a class of Deep Neural Networks that can recognize and classify particular features from images and are widely used for analyzing visual images. Their applications range from image and video recognition, image classification, medical image analysis, computer vision and natural language processing.

Methods for evaluating a model's performance are divided into 2 categories: namely, holdout and Cross-validation. Both methods use a test set (i.e data not seen by the model) to evaluate model performance. It's not recommended to use the data we used to build the model to evaluate it. This is because our model will simply remember the whole training set, and will therefore always predict the correct label for any point in the training set. This is known as overfitting.

Holdout:

The purpose of holdout evaluation is to test a model on different data than it was trained on. This provides an unbiased estimate of learning performance.
In this method, the dataset is *randomly* divided into three subsets:

1. **Training set** is a subset of the dataset used to build predictive models.

2. **Validation set** is a subset of the dataset used to assess the performance of the model built in the training phase. It provides a test platform for fine-tuning a model's parameters and selecting the best performing model. Not all modeling algorithms need a validation set.

3. **Test set**, or unseen data, is a subset of the dataset used to assess the likely future performance of a model. If a model fits to the training set much better than it fits the test set, overfitting is probably the cause.

The holdout approach is useful because of its speed, simplicity, and flexibility. However, this technique is often associated with high variability since differences in the training and test dataset can result in meaningful differences in the estimate of accuracy.

Cross-Validation:

Cross-validation is a technique that involves partitioning the original observation dataset into a training set, used to train the model, and an independent set used to evaluate the analysis.

The most common cross-validation technique is k-fold cross-validation, where the original dataset is partitioned into k equal size subsamples, called folds. The k is a user-specified number, usually with 5 or 10 as its preferred value. This is repeated k times, such that each time, one of the k subsets is used as the test set/validation set and the other k-1 subsets are put together to form a training set. The error estimation is averaged over all k trials to get the total effectiveness of our model.

6. CNN ARCHITECTURE

CNNs are a class of Deep Neural Networks that can recognize and classify particular features from images and are widely used for analyzing visual images. Their applications range from image and video recognition, image classification, medical image analysis, computer vision and natural language processing.

The term 'Convolution" in CNN denotes the mathematical function of convolution which is a special kind of linear operation wherein two functions are multiplied to produce a third function which expresses how the shape of one function is modified by the other. In simple terms, two images which can be represented as matrices are multiplied to give an output that is used to extract features from the image.

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernals), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.

6.1 Basic Architecture

There are two main parts to a CNN architecture

- A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction
- A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.

6.1 CNN Layers:

The multiple occurring of these layers shows how deep our network is, and this formation is known as the deep neural network.

- Input: raw pixel values are provided as input.
- Convolutional layer: Input layers translates the results of the neuron layer. There is a need to specify the filter to be used. Each filter can only be a 5*5 window that slides over input data and gets pixels with maximum intensities.
- Rectified linear unit [ReLU] layer: provided activation function on the data taken as an image. In the case of back propagation, ReLU function is used which prevents the values of pixels from changing.
- Pooling layer: Performs a down-sampling operation in volume along the dimensions (width, height).

- Fully connected layer: score class is focused, and a maximum score of the input digits is found.

As we go deeper and deeper in the layers, the complexity is increased a lot. But it might be worth going as accuracy may increase but unfortunately, time consumption also increases.

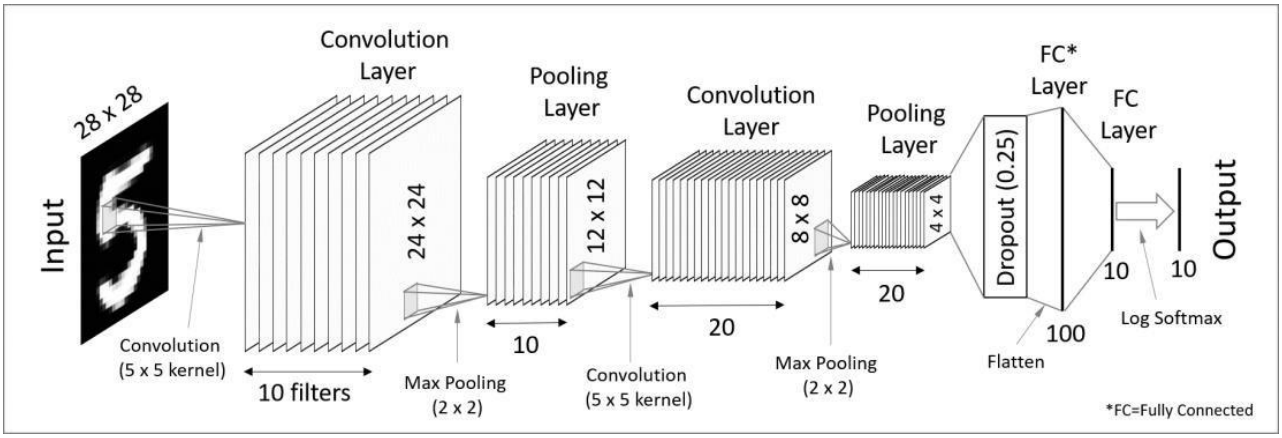


Figure 10 CNN Architecture For Handwritten Digit Recognition

1. Convolutional Layer

This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).

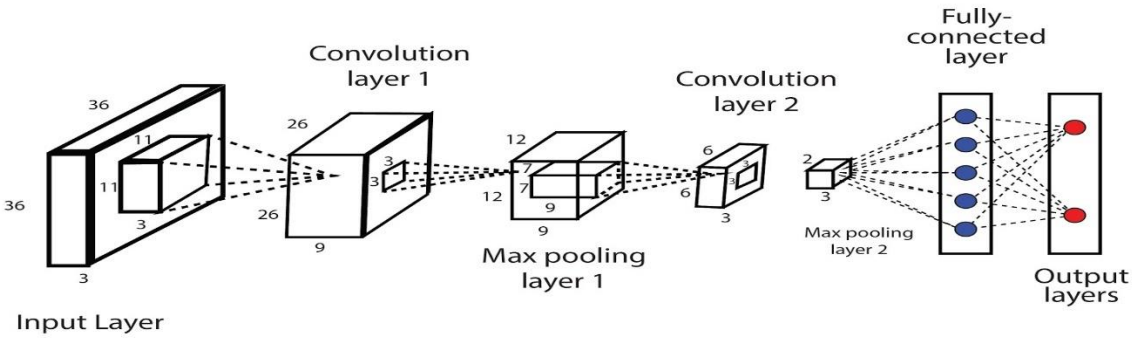


Figure 11 Convolutional Layer

The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

2. Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations.

In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

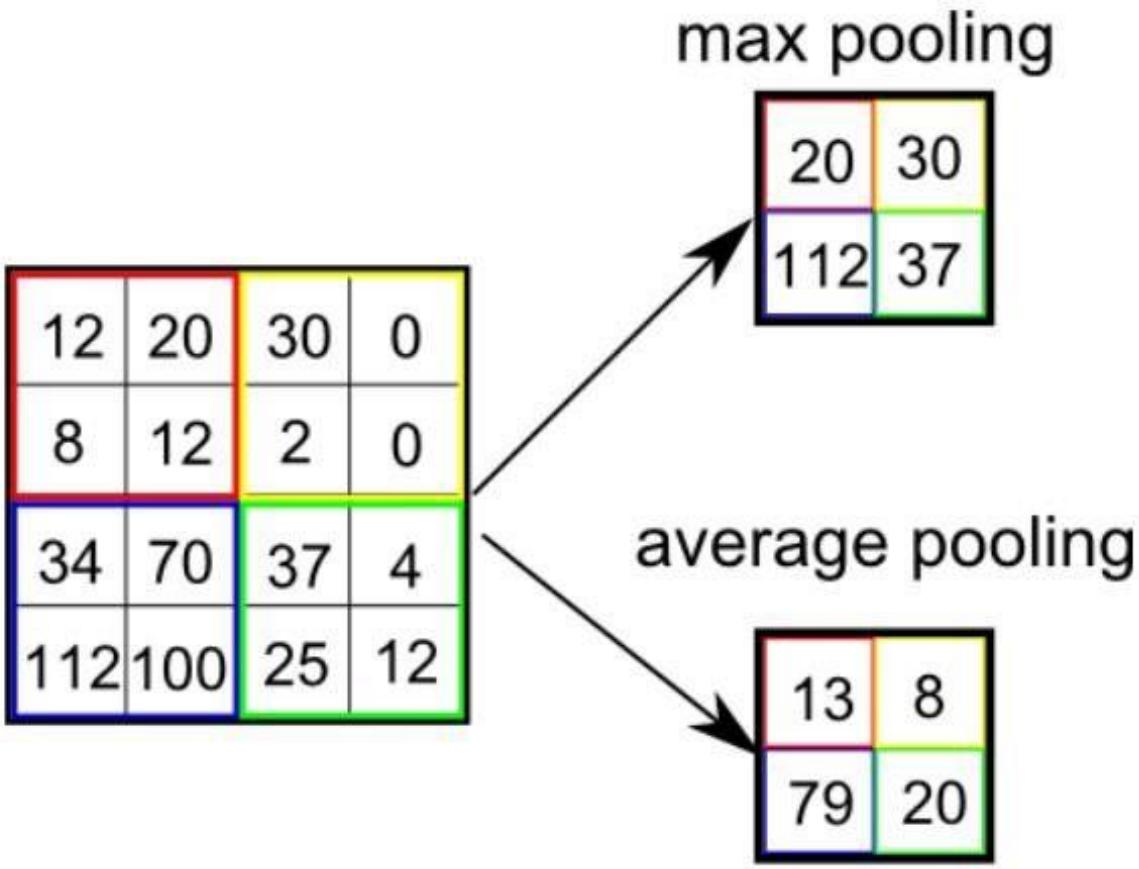


Figure 12 Pooling Layer

3. Fully Connected Layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

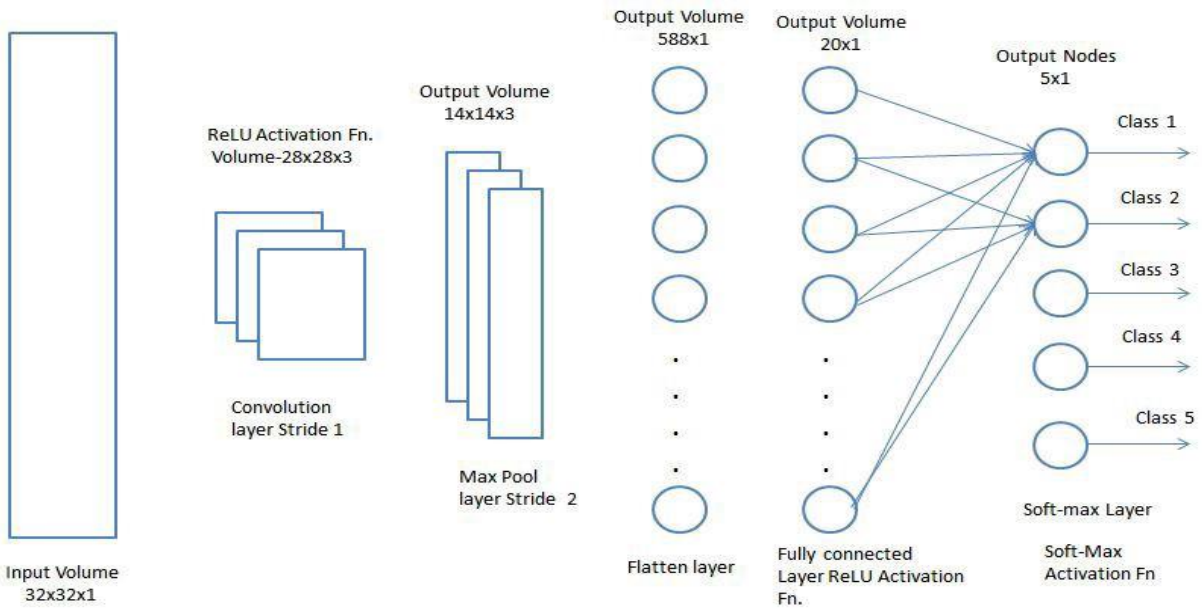


Figure 13 Fully Connected Layer

In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.

4. Dropout

Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model's performance when used on a new data.

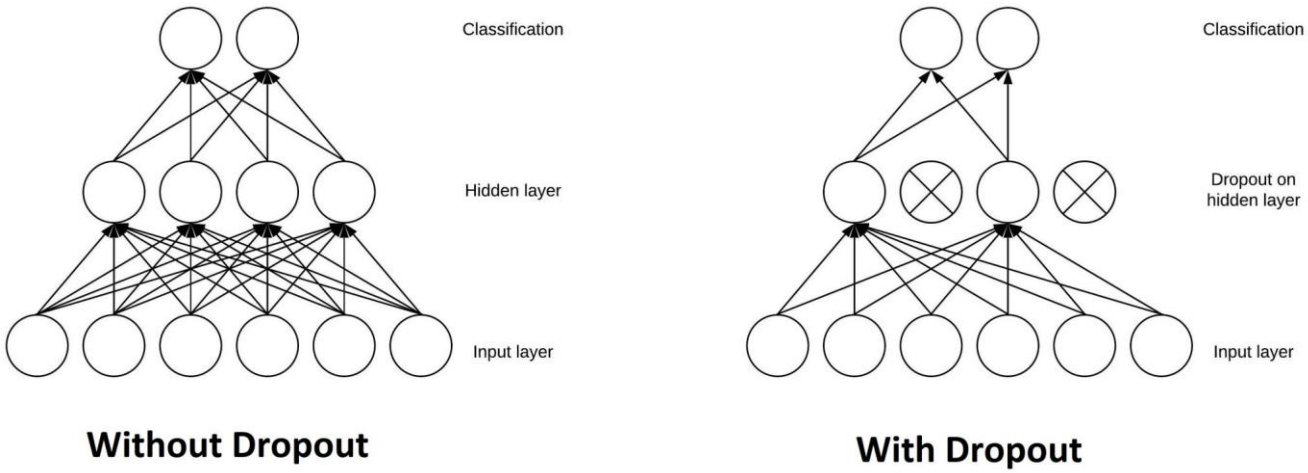


Figure 14 Dropout layer

To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

5. Activation Functions

An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.
Sometimes the activation function is called a "*transfer function*." If the output range of the activation function is limited, then it may be called a "*squashing function*." Many activation functions are nonlinear and may be referred to as the "*nonlinearity*" in the layer or the network design.
The choice of activation function has a large impact on the capability and performance of the neural network, and different activation functions may be used in different parts of the model.

Technically, the activation function is used within or after the internal processing of each node in the network, although networks are designed to use the same activation function for all nodes in a layer.

A network may have three types of layers: input layers that take raw input from the domain, **hidden layers** that take input from another layer and pass output to another layer, and **output layers** that make a prediction.
All hidden layers typically use the same activation function. The output layer will typically use a different activation function from the hidden layers and is dependent upon the type of prediction required by the model.

Activation functions are also typically differentiable, meaning the first-order derivative can be calculated for a given input value. This is required given that neural networks are typically trained using the backpropagation of error algorithm that requires the derivative of prediction error in order to update the weights of the model.

There are many different types of activation functions used in neural networks, although perhaps only a small number of functions used in practice for hidden and output layers.

Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network.
It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred an for a multi-class classification, generally softmax us used.

- A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction
- A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.

Applications:

1. Object detection: With CNN, we now have sophisticated models like R-CNN, Fast R-CNN, and Faster R-CNN that are the predominant pipeline for many object detection models deployed in autonomous vehicles, facial detection, and more.

2. Semantic segmentation: In 2015, a group of researchers from Hong Kong developed a CNN-based Deep Parsing Network to incorporate rich information into an image segmentation model. Researchers from UC Berkeley also built fully convolutional networks that improved upon state-of-the-art semantic segmentation.

6.2 Why ConvNets over Feed-Forward Neural Nets?

An image is nothing but a matrix of pixel values, right? So why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes?

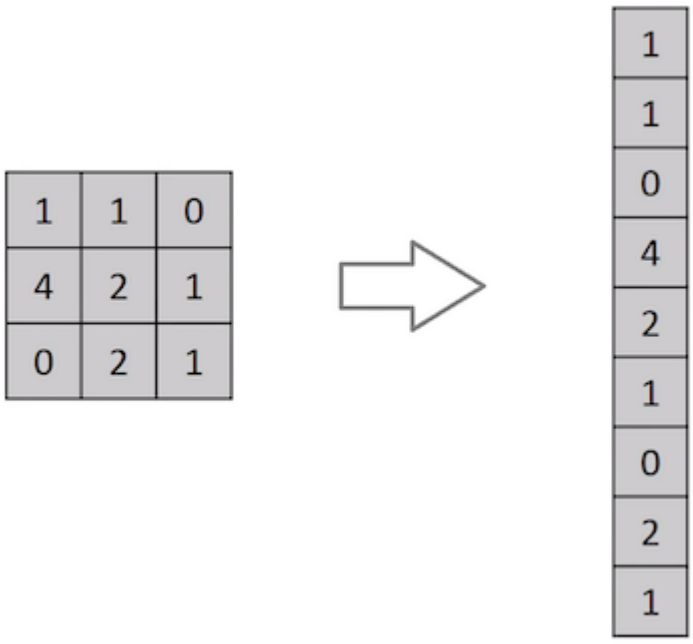


Figure 15 Flattening of a 3x3 image matrix into a 9x1 vector

In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images

having pixel dependencies throughout. A ConvNet is able to **successfully capture the Spatial and Temporal dependencies** in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

Convolutional neural network is better than a feed-forward network since CNN has features parameter sharing and dimensionality reduction. Because of parameter sharing in CNN, the number of parameters is reduced thus the computations also decreased. The main intuition is the learning from one part of the image is also useful in another part of the image. Because of the dimensionality reduction in CNN, the computational power needed is reduced.

All the layers of a CNN have multiple convolutional filters working and scanning the complete feature matrix and carry out the dimensionality reduction. This enables CNN to be a very apt and fit network for image classifications and processing.

7. EXPERIMENTAL ANALYSIS AND RESULTS

7.1 System Configuration:

7.1.1 Software requirements**:**

These are the software configurations used:

Operating system: windows 10.

IDE: Jupyter Notebook.

Python: Python is an interpreted, high-level, general purpose programming language created by Guido Van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant Whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Jupyter Notebook: Jupyter is a free, open-source, interactive web tool known as a computational notebook, which researchers can use to combine software code, computational output, explanatory text and multimedia resources in a single document. Computational notebooks have been around for decades, but Jupyter in particular has exploded in popularity over the past couple of years. This rapid uptake has been aided by an enthusiastic community of user–developers and a redesigned architecture that allows the notebook.

7.1.2 Hardware requirements:

These are the Hardware interfaces used Processor: Intel Pentium 4 or equivalent

RAM: Minimum of 256 MB or higher HDD: 10 GB or higher

Monitor: 15'' or 17'' color monitor

Mouse: Scroll or optical mouse

Keyboard: Standard 110 keys keyboard

7.2 Learning Curves:

A learning curve is a concept that graphically depicts the relationship between the cost and output over a defined period of time, normally to represent the repetitive task of an employee or worker. The learning curve was first described by psychologist Hermann Ebbinghaus in 1885 and is used as a way to measure production efficiency and to forecast costs. In the visual representation of a learning curve, a steeper slope indicates initial learning translates into higher cost savings, and subsequent learnings result in increasingly slower, more difficult cost savings.

A learning curve is just a **plot showing the progress over the experience of a specific metric related to learning during the training** of a machine learning model. They are just a mathematical representation of the learning process.

According to this, we'll have a measure of time or progress in the x-axis and a measure of error or performance in the y-axis.

We use these charts to monitor the evolution of our model during learning so we can diagnose problems and optimize the prediction performance.

We often see these two types of learning curves appearing in charts:

- *Optimization Learning Curves*: Learning curves calculated on the metric by which the parameters of the model are being optimized, such as loss or Mean Squared Error
- *Performance Learning Curves*: Learning curves calculated on the metric by which the model will be evaluated and selected, such as accuracy, precision, recall, or F1 score

Accuracy and **Loss** are the two most well-known and discussed metrics in machine learning.

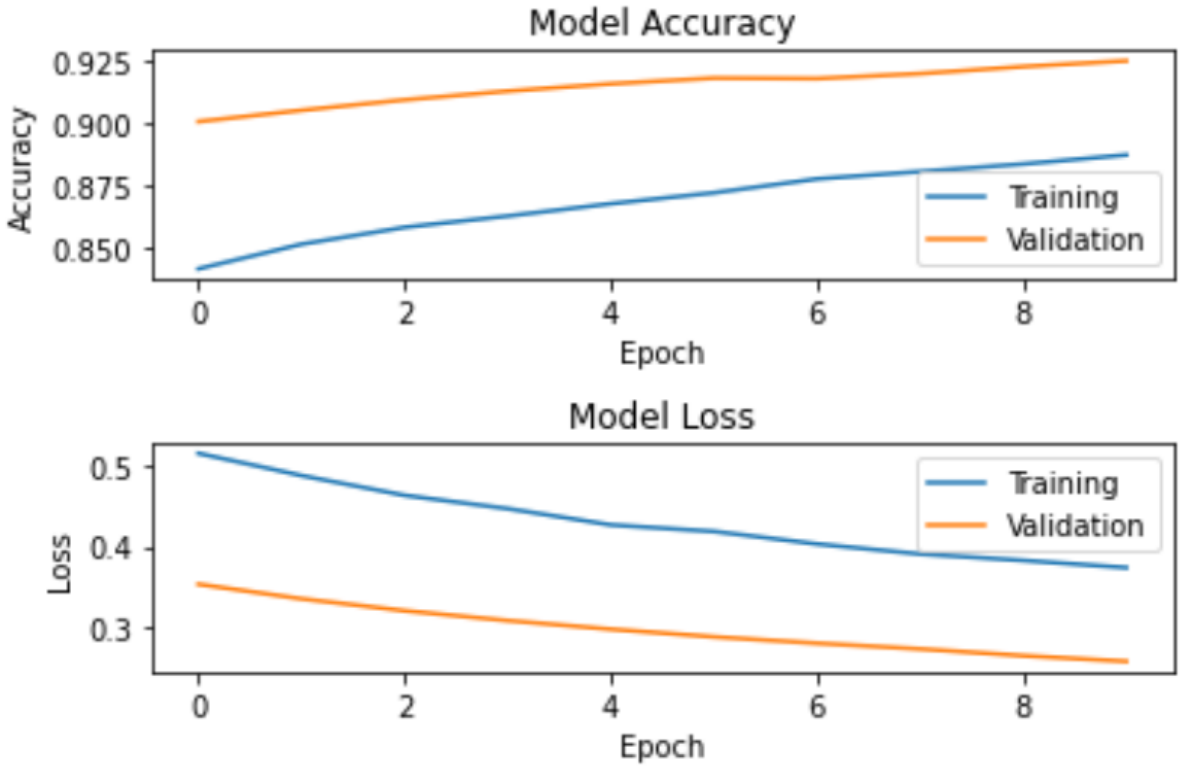


Figure 16 Accuracy curve and Loss Curve

From the above curve we can say that accuracy during training and validation has increased with increase in number of epochs and loss has been subsequently decreases during training and validation.

7.2.1 Accuracy curve:

Accuracy is a method for measuring a classification model's performance. It is typically expressed as a percentage. Accuracy is the count of predictions where the predicted value is equal to the true value. It is binary (true/false) for a particular sample. Accuracy is often graphed and monitored during the training phase though the value is often associated with the overall or final model accuracy. Accuracy is easier to interpret than loss.

7.2.2 Loss Curve:

A loss function, also known as a cost function, takes into account the probabilities or uncertainty of a prediction based on how much the prediction varies from the true value. This gives us a more nuanced view into how well the model is performing.

Unlike accuracy, loss is not a percentage — it is a summation of the errors made for each sample in training or validation sets. Loss is often used in the training process to find the "best" parameter values for the model (e.g. weights in neural network). During the training process the goal is to minimize this value. The most common loss functions are **log loss** and **cross-entropy loss** (which yield the same result when calculating error rates between 0 and 1), as well as **mean squared error**, and **likelihood loss.** Unlike accuracy, loss may be used in both classification and regression problems.

One of the most used plots to debug a neural network is a Loss curve during training. It gives us a snapshot of the training process and the direction in which the network learns.

7.3 Sample Code:

7.3.1 Import the libraries and load the dataset**:**

First, we are going to import all the modules that we are going to need for training our model. The Keras library already contains some datasets and MNIST is one of them. So we can easily import the dataset and start working with it.

The **mnist.load_data()** method returns us the training data, its labels and also the testing data and its labels.

```
import keras from keras.datasets
import mnist from keras.models
import Sequential from keras.layers
import Dense, Dropout, Flatten from
keras.layers import Conv2D,
MaxPooling2D from keras import
backend as K
(x_train, y_train), (x_test, y_test) = mnist.load_data()
print(x_train.shape, y_train.shape)
```

```
: import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
print(x_train.shape, y_train.shape)
```

```
(60000, 28, 28) (60000,)
```

Figure 17 Output Training and Test Data Shape

7.3.2 Preprocess the data:

The image data cannot be fed directly into the model so we need to perform some operations and **process the data** to make it ready for our neural network. The dimension of the training data is (60000,28,28). The CNN model will require one more dimension so we reshape the matrix to shape (60000,28,28,1).

```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1) x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
#convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes) y_test = keras.utils.to_categorical(y_test, num_classes)
x_train = x_train.astype('float32') x_test = x_test.astype('float32') x_train /= 255 x_test /= 255 print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, 10
y_test = keras.utils.to_categorical(y_test, 10)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

Figure 18 Output Train and Test Samples

7.3.3 Create the model:

Now we will create our CNN model in Python data science project. A CNN model generally consists of convolutional and pooling layers.

It works better for data that are represented as grid structures, this is the reason why CNN works well for image classification problems.

The dropout layer is used to deactivate some of the neurons and while training, it reduces offer fitting of the model. We will then compile the model with the **Adadelta** optimizer.

batch_size = 128 num_classes = 10 epochs = 10 model = Sequential() model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape)) model.add(Conv2D(64, (3, 3), activation='relu')) model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Dropout(0.25)) model.add(Flatten()) model.add(Dense(256, activation='relu')) model.add(Dropout(0.5))

```
model.add(Dense(num_classes,
activation='softmax'))
model.compile(loss=keras.losses.categorica
l_crossentropy,optimizer=keras.optimiz
ers.Adadelta(),metrics=['accuracy'])
```

7.3.4 Train the Model :

The model.fit() function of Keras will start the training of the model. It takes the training data, validation data, epochs, and batch size.It takes some time to train the model. After training, we save the weights and model definition in the ‘mnist.h5’ file.

```
hist =
model.fit(x_tra
in,
y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_te
st, y_test))
print("The model has successfully trained")
model.save('mnist.h5')
print("Saving the model as mnist.h5")
```

Epochs Running:

```
Epoch 1/20
469/469 [==============================] - 101s 216ms/step - loss: 2.2880 - accuracy: 0.1356 - val_l
oss: 2.2662 - val_accuracy: 0.2760
Epoch 2/20
469/469 [==============================] - 101s 215ms/step - loss: 2.2487 - accuracy: 0.2489 - val_l
oss: 2.2179 - val_accuracy: 0.4613
Epoch 3/20
469/469 [==============================] - 101s 215ms/step - loss: 2.1971 - accuracy: 0.3559 - val_l
oss: 2.1498 - val_accuracy: 0.5518
Epoch 4/20
469/469 [==============================] - 100s 214ms/step - loss: 2.1202 - accuracy: 0.4506 - val_l
oss: 2.0485 - val_accuracy: 0.6127
Epoch 5/20
469/469 [==============================] - 210089s 448s/step - loss: 2.0066 - accuracy: 0.5147 - val
_loss: 1.8987 - val_accuracy: 0.6651
Epoch 6/20
469/469 [==============================] - 85s 181ms/step - loss: 1.8419 - accuracy: 0.5789 - val_lo
ss: 1.6888 - val_accuracy: 0.7164
Epoch 7/20
469/469 [==============================] - 85s 181ms/step - loss: 1.6308 - accuracy: 0.6277 - val_lo
```

```
Epoch 8/20
469/469 [==============================] - 81s 172ms/step - loss: 1.3985 - accuracy: 0.6658 - val_lo
ss: 1.1777 - val_accuracy: 0.7929
Epoch 9/20
469/469 [==============================] - 82s 175ms/step - loss: 1.1975 - accuracy: 0.6927 - val_lo
ss: 0.9706 - val_accuracy: 0.8133
Epoch 10/20
469/469 [==============================] - 86s 183ms/step - loss: 1.0412 - accuracy: 0.7176 - val_lo
ss: 0.8206 - val_accuracy: 0.8254
Epoch 11/20
469/469 [==============================] - 81s 172ms/step - loss: 0.9298 - accuracy: 0.7370 - val_lo
ss: 0.7162 - val_accuracy: 0.8383
Epoch 12/20
469/469 [==============================] - 81s 173ms/step - loss: 0.8469 - accuracy: 0.7548 - val_lo
ss: 0.6423 - val_accuracy: 0.8478
Epoch 13/20
469/469 [==============================] - 82s 176ms/step - loss: 0.7867 - accuracy: 0.7659 - val_lo
ss: 0.5874 - val_accuracy: 0.8547
Epoch 14/20
469/469 [==============================] - 82s 175ms/step - loss: 0.7387 - accuracy: 0.7794 - val_lo
```

```
Epoch 15/20
469/469 [==============================] - 80s 172ms/step - loss: 0.6978 - accuracy: 0.7894 - val_lo
ss: 0.5134 - val_accuracy: 0.8682
Epoch 16/20
469/469 [==============================] - 80s 171ms/step - loss: 0.6640 - accuracy: 0.7997 - val_lo
ss: 0.4868 - val_accuracy: 0.8718
Epoch 17/20
469/469 [==============================] - 82s 176ms/step - loss: 0.6379 - accuracy: 0.8063 - val_lo
ss: 0.4648 - val_accuracy: 0.8762
Epoch 18/20
469/469 [==============================] - 81s 172ms/step - loss: 0.6115 - accuracy: 0.8147 - val_lo
ss: 0.4463 - val_accuracy: 0.8800
Epoch 19/20
469/469 [==============================] - 81s 173ms/step - loss: 0.5938 - accuracy: 0.8207 - val_lo
ss: 0.4306 - val_accuracy: 0.8837
Epoch 20/20
469/469 [==============================] - 81s 173ms/step - loss: 0.5776 - accuracy: 0.8251 - val_lo
ss: 0.4172 - val_accuracy: 0.8856
The model has successfully trained
Saving the model as mnist.h5
```

Figure 19 Epochs

7.3.5 Evaluate the model :

We have 10,000 images in our dataset which will be used to evaluate how good our model works. The testing data was not involved in the training of the data therefore, it is new data for our model. The MNIST dataset is well balanced so we can get around 99% accuracy.

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.41715335845947266
Test accuracy: 0.8855999708175659
```

Figure 20 Output Test Loss and Test Accuracy

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 9, 64)          36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 100)               102500
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 159,254
Trainable params: 159,254
Non-trainable params: 0
```

Figure 21 Model Summary

7.3.6 Create GUI to predict digits:

Now for the GUI, we have created a new file in which we build an interactive window to draw digits on canvas and with a button, we can recognize the digit. The Tkinter library comes in the Python standard library. We have created a function predict_digit() that takes the image as input and then uses the trained model to predict the digit.Then we create the App class which is responsible for building the GUI for our app. We create a canvas where we can draw by capturing the mouse event and with a button, we trigger the predict_digit() function and display the results. Here's the full code for our gui_digit_recognizer.py file:

```
from keras.models import load_model
from tkinter import *
import tkinter as tk
import win32gui
from PIL import ImageGrab, Image
import numpy as np

model = load_model('mnist.h5')

def predict_digit(img):
    #resize image to 28x28 pixels
    img = img.resize((28,28))
    #convert rgb to grayscale
    img = img.convert('L')
    img = np.array(img)
    #reshaping to support our model input and normalizing
    img = img.reshape(1,28,28,1)
    img = img/255.0
    #predicting the class
    res = model.predict([img])[0]
    return np.argmax(res), max(res)

class App(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)

        self.x = self.y = 0

        # Creating elements
        self.canvas = tk.Canvas(self, width=300, height=300, bg = "white", cursor="cross")
        self.label = tk.Label(self, text="Thinking..", font=("Helvetica", 48))
        self.classify_btn = tk.Button(self, text = "Recognise", command =
self.classify_handwriting)
        self.button_clear = tk.Button(self, text = "Clear", command = self.clear_all)
```

```
        # Grid structure
        self.canvas.grid(row=0, column=0, pady=2, sticky=W, )
        self.label.grid(row=0, column=1,pady=2, padx=2)
        self.classify_btn.grid(row=1, column=1, pady=2, padx=2)
        self.button_clear.grid(row=1, column=0, pady=2)

        #self.canvas.bind("<Motion>", self.start_pos)
        self.canvas.bind("<B1-Motion>", self.draw_lines)

    def clear_all(self):
        self.canvas.delete("all")

    def classify_handwriting(self):
        HWND = self.canvas.winfo_id() # get the handle of the canvas
        rect = win32gui.GetWindowRect(HWND) # get the coordinate of the canvas
        im = ImageGrab.grab(rect)

        digit, acc = predict_digit(im)
        self.label.configure(text= str(digit)+', '+ str(int(acc*100))+'%')

    def draw_lines(self, event):
        self.x = event.x
        self.y = event.y
        r=8
        self.canvas.create_oval(self.x-r, self.y-r, self.x + r, self.y + r, fill='black')

app = App()
mainloop()
```

7.4 OUTPUT :

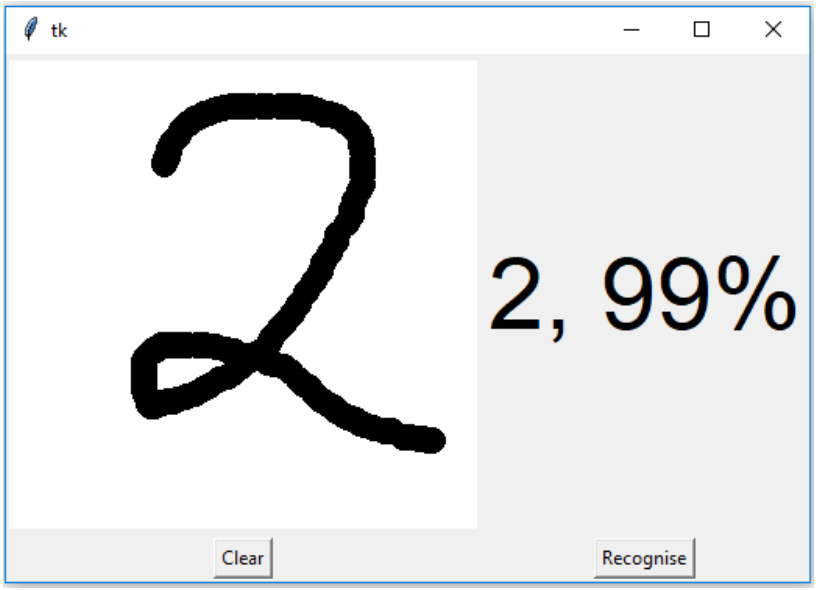


Figure 22 Output 'digit 2'

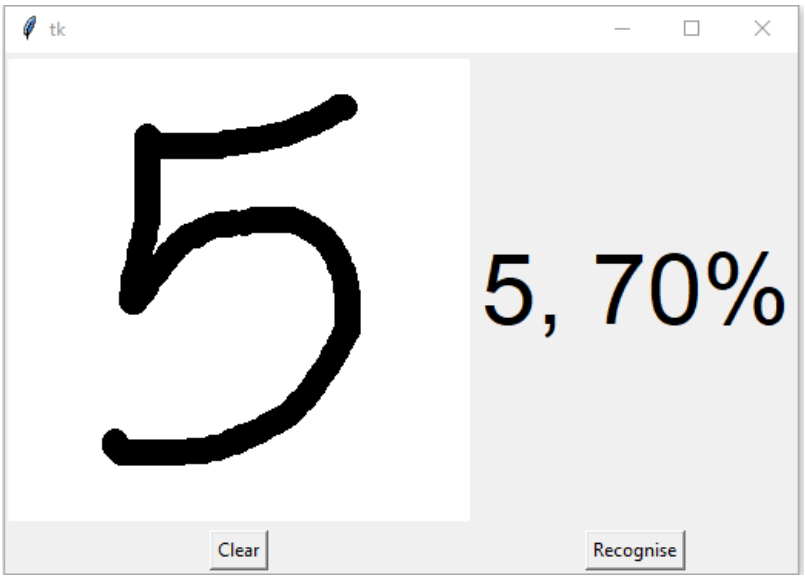


Figure 23 Output 'digit 5'

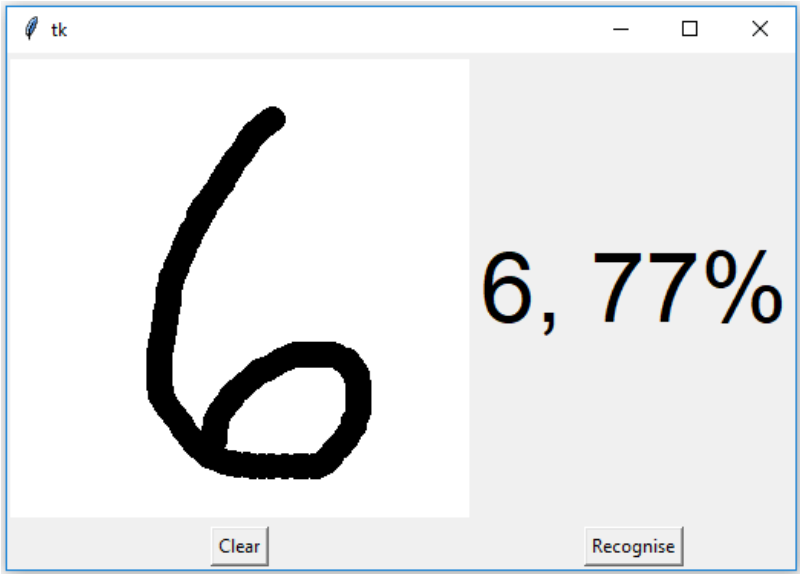


Figure 24 Output 'digit 6'

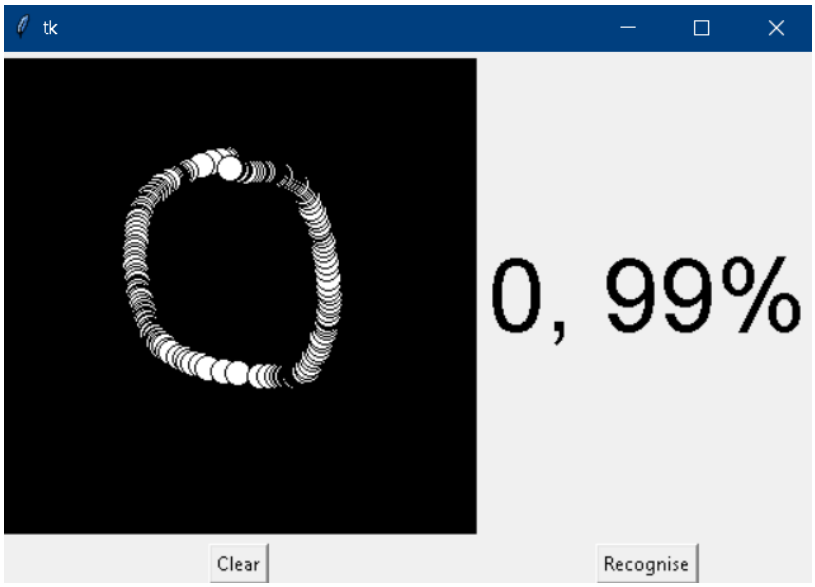


Figure 25 Output 'digit 0'

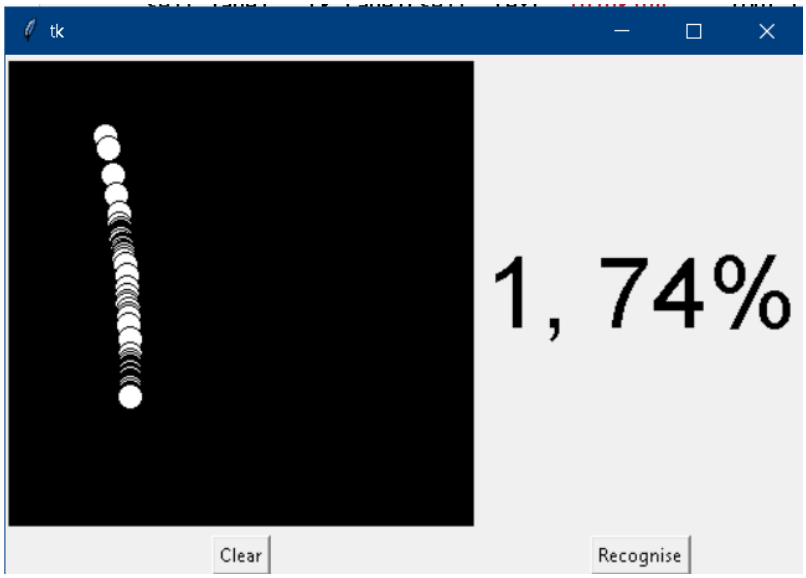


Figure 26 Output 'digit 1'

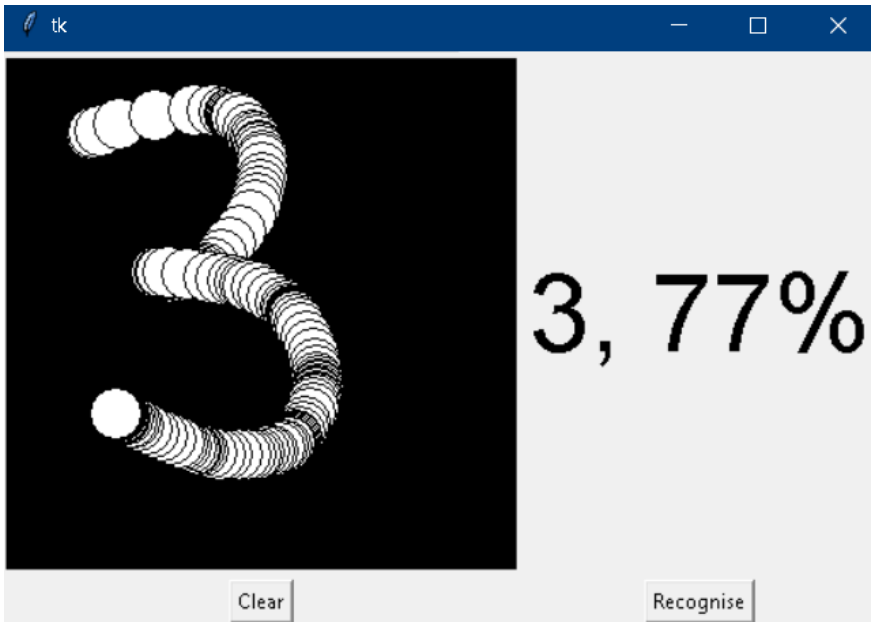


Figure 27 Output 'digit 3'

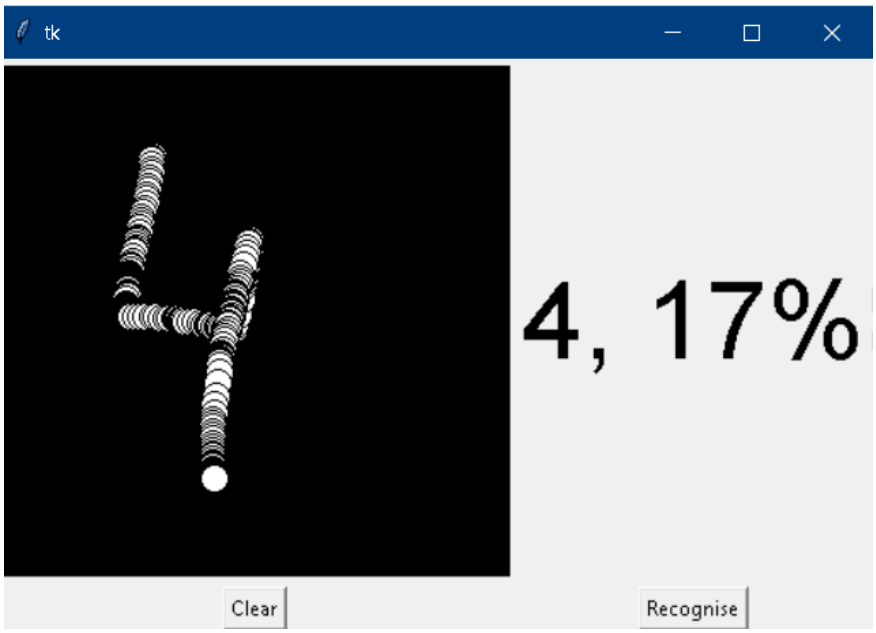


Figure 28 Output 'digit 4'

8. APPENDIX

Python:

Python is an interpreted, high-level, general purpose programming language created by Guido Van Rossum and first released in 1991, Python's design philosophy emphasizes code Readability with its notable use of significant White space. Its language constructs and object oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Keras :

Keras is a powerful and easy-to-use free open source Python library for developing and evaluating **deep larning** models**.**

It wraps the efficient numerical computation libraries **Theano** and **TensorFlow** and allows you to define and train neural network models in just a few lines of code. It uses libraries such as Python, C#, C++ or standalone machine learning toolkits. Theano and TensorFlow are very powerful libraries but difficult to understand for creating neural networks.

Keras is based on minimal structure that provides a clean and easy way to create deep learning models based on TensorFlow or Theano. Keras is designed to quickly define deep learning models. Well, Keras is an optimal choice for deep learning applications.

Steps for creating a keras model:

1)First we must define a network model.

2)Compile it, which transforms the simple sequence of layers into a complex group of matrix operations.

3)Train or fit the network.

To import: from keras.models import Sequential

fromkeras.layers import Dense, Activation, Dropout

TensorFlow:

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of **TensorFlow.** TensorFlow tutorial is designed for both beginners and professionals. Our tutorial provides all the basic and advanced concept of machine learning and deep learning concept such as deep neural network, image processing and sentiment analysis.

TensorFlow is one of the famous deep learning frameworks, developed by **Google** Team. It is a free and open source software library and designed in **Python** programming language, this tutorial is designed in such a way that we can easily implements deep learning project on TensorFlow in an easy and efficient way. Unlike other numerical libraries intended for use in Deep Learning like **Theano**, **TensorFlow** was designed for use both in research and development and in production systems. It can run on single CPU systems, GPUs as well as mobile devices and largescale distributed systems of hundreds of machines.

Numpy:

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. Numpy which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc. It is an open source project and you can use it freely. NumPy stands for Numerical Python. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called **ndarray**, it provides a lot of supporting functions that make working with **ndarray** very easy. Arrays are very frequently used in data science, where speed and resources are very important.

Pillow:

Pillow is a free and open source library for the Python programming language that allows you to easily create &s manipulate digital images. Pillow is built on top of PIL (Python Image Library). PIL is one of the important modules for image processing in Python. However, the PIL module is not supported since 2011 and doesn't support python 3.

Pillow module gives more functionalities, runs on all major operating system and support for python 3. It supports wide variety of images such as "jpeg", "png", "bmp", "gif", "ppm", "tiff". You can do almost anything on digital images using pillow module. Apart from basic image processing functionality, including point operations, filtering images using built-in convolution kernels, and color space conversions.

Tkinkter:

Tkinter is the standard **GUI library** for Python. Python when combined with Tkinter provides a fast and easy way to create **GUI applications**. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

We need to import all the modules that we are going to need for training our model. The Keras library already contains some datasets and MNIST is one of them. So we can easily import the dataset through Keras. The mnist.load_data() method returns the training data, its labels along with the testing data and its labels.

Jupyter Notebook:

JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is extensible and modular: write plugins that add new components and integrate with existing ones.

Machine Learning:

Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

Deep Learning:

Deep learning is an artificial intelligence (AI) function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network.

Neural Networks:

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature.

9.CONCLUSION AND FUTURE WORK

9.1 Conclusion:

Our project HANDWRITTEN DIGIT RECOGNITION deals with identifying the digits.

The main purpose of this project is to build an automatic handwritten digit recognition method for the recognition of handwritten digit strings.

In this project, different machine learning methods, which are SVM (Support Vector Machine), ANN (Artificial Neural Networks), and CNN (Convolutional Neural Networks) architectures are used to achieve high performance on the digit string recognition problem.

9.2 Future Work :

The proposed system takes 28x28 pixel sized images as input. The same system with further modifications and improvements in the dataset and the model can be used to build Handwritten Character Recognition System which recognizes human handwritten characters and predicts the output.

10. REFERENCES

1. https://www.tensorflow.org/learn
2. https://www.geeksforgeeks.org/python-tkinter-tutorial/
3. https://medium.com/the-andela-way/applying-machine-learning-to-recognize handwritten-characters-babcd4b8d705
4. https://nanonets.com/blog/handwritten-character-recognition/
5. https://www.geeksforgeeks.org/python-tkinter-tutorial/
6. https://medium.com/the-andela-way/applying-machine-learning-to-recognize handwritten-characters-babcd4b8d705
7. https://nanonets.com/blog/handwritten-character-recognition/
8. https://www.tensorflow.org/learn
9. R. Alhajj and A. Elnagar, "Multiagents to separating handwritten connected digits," in IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, vol. 35, no. 5, pp. 593-602, Sept. 2005. https://doi.org/10.1109/TSMCA.2005.843389
10. https://towardsdatascience.com/useful-plots-to-diagnose-your-neural-network-521907fa2f45
11. https://towardsdatascience.com/the-best-machine-learning-algorithm-for-handwritten-digits-recognition-2c6089ad8f09
12. https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python
13. https://www.analyticsvidhya.com/blog/2021/01/building-a-cnn-model-with-95-accuracy/
14. https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/
15. https://www.kdnuggets.com/2018/11/top-python-deep-learning-libraries.html
16. https://analyticsindiamag.com/7-types-classification-algorithms/
17. https://towardsdatascience.com/image-recognition-with-machine-learning-on-python-image-processing-3abe6b158e9a

ISSN 2278-3091
Volume 9 No.2, March -April 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse66922020.pdf
https://doi.org/10.30534/ijatcse/2020/66922020

Effective Handwritten Digit Recognition using Deep Convolution Neural Network

Yellapragada SS Bharadwaj[1], Rajaram P[2], Sriram V.P[3], Sudhakar S[4], Kolla Bhanu Prakash[5]

[1] Dept. of CSE, Koneru Lakshmaiah Education Foundation, India,
saisrinivasabharadwaj@gmail.com
[2]Dept. of CSE, AKT Memorial College of Engineering and Technology, Kallakurichi, Tamil Nadu, India,
rajaramnov82@gmail.com
[3]Dept. of Management Studies, Acharya Bangalore B School (ABBS), Bengaluru,India
dr.vpsriram@gmail.com
[4]Dept. of CSE, Sree Sakthi Engineering College, Coimbatore, Tamil Nadu India,
sudhasengan@gmail.com
[5]Dept. of CSE, Koneru Lakshmaiah Education Foundation, India,
drkbp@kluniversity.in

ABSTRACT

This paper proposed a simple neural network approach towards handwritten digit recognition using convolution. With machine learning algorithms like KNN,SVM/SOM, recognizing digits is considered as one of the unsolvable tasks due to its distinctiveness in the style of writing. In this paper, Convolution Neural Networks are implemented with an MNIST dataset of 70000 digits with 250 distinct forms of writings. The proposed method achieved 98.51% accuracy for real-world handwritten digit prediction with less than 0.1 % loss on training with 60000 digits while 10000 under validation.

Key words: Convolution neural networks, MNIST dataset, TensorFlow, OCR, Segmentation, Cross-Validation

1. INTRODUCTION

Advancements in the field of computer vision using deep neural networks attract attention; thus, many A.I. practitioners are moving towards it.[1] One of the influencing projects that opted for deep learning is OCR (Object Character Recognition). OCR is a mechanism that converts printed or documented letters into encoded text. Intuitively OCR tool scans a document to extract the information and store it in a digital document. There are two major ways of implementing OCR, one by recognizing the patterns in the characters, and the other one is through segmentation.[1][10] Either way, this problem comes under the radar of machine learning. Handwritten digit recognition (HDR) is a snippet of OCR where instead of taking the whole character's data, HDR detects digits. Comparing to OCR, HDR is light and faster. In fields like medical, banking, student management, and taxation process, HDR possesses great flexibility.[3]

A human brain can easily interpret a sophisticated image and can extract data from it, but for a computer, image is a collection of pixels which are nothing but a list of combinations of numbers ranging from 0 to 255, blends being RGB, B.W., greyscale, etc. Information in an image can be extracted from the pixel values. A human eye contains light-sensitive cells that can segment the image into partitions which help to interpret the information like shape, size, and color which are called features, these features are later sent to the visual cortex that analyses the characteristics and maps it to our memory for identification. Thus our brain is capable of identifying an unseen image.

Neural Networks are a branch of machine learning inspired by the human brain. It adapts layered architecture where the layers represent mathematical functions, and each neutron is a carrier of weights. Layers that lie between input and output layers are called hidden layers, which can be changed based on the task. For image classification CNN (Convolution Neural Networks) performs better, and for speech recognition, Long short-term memory network delivers better.[4] Intuitively convolution is a process of applying various filters on images to highlight the details in it. Each binary classifier called perceptron, many perceptron mapping from each neuron to every other neuron of forwarding layers together forms a neural network.[2] Consider an input greyscale image of size 28x28, and the input layer consists of 784 neurons, each with a value of grey level corresponding to

the pixels, and the weight carried by that neuron is called activation. The last layer, aka the output layer, contains neurons equal to the number of target classes; in case of the handwritten digit, recognition number of categories are ten ranging from 0 to 9 with probability values.[3][4] The number of input neurons and output neurons depends on the task, but the hidden layers are arbitrary and are not dependent on the task. That's why they are hidden layers, but the propagations between the hidden layers depend on the activation of the previous layers.[11] [12] Intuitively, the pattern of activations in the presentation layer causes some specific patterns in the next layer; thus, the highest activation neuron is the network's choice of class. In this paper, we are implementing convolution neural network architecture with relu and sigmoid activation functions to predict a real-world handwritten digit by training the network with the MNIST dataset.[10]

2. RELATED WORK

Many techniques have been developed to recognize handwritten digits; most of the A.I. practitioners use this to test their model's performance. In the past decades, a segmentation-based approach was used to solve this problem later with the advancements in machine learning a segmentation less approach was introduced. Even though the implementation changes, the issue still remains the same and open for anyone to solve.
Bailing Zhang [5] utilizes the ASSOM technique to provide numerical stability that can predict precise digits. Their main idea was to use the SOM clustering algorithm with autoencoder neural networks in a nonlinear approach. The modularity of SOM helped in extracting several features in a digit. For each digit, individual ASSOM was constructed and compared with ten several construction-related errors to minimize the misclassification. Their model shows promising results, even with small training samples.

Saleh Aly [3] proposed a technique for handwritten numerical strings of arbitrary length recognition using SVM and PCA, addressing the major challenge in word detection, which is overlapping characters. Their method uses hybrid PCA called PCANet for segmentation and SVM for segmentation classification together called PCA-SVMNet. Their experiment shows high efficiency in recognizing unknown handwritten number classification without any segmentation method applied [15]

Yue Yin; Wei Zhang [1] have concluded that out of all neural network implementations CNN method is valid for OCR based image classification systems. They claimed that OCR had become a preliminary technique in the field of computer vision; they state that if an image classification model performs well in OCR, then it can be used for any image classification systems.

Mahmoud M. Abu Ghosh [8] compared CNN, DNN, DBN approaches to determine which neural network is useful in the field of computer vision, specifically in image classification like OCR. Their analysis claims that DNN's(Deep Belief Network) accuracy beats other neural networks with 98.08% accuracy but falls behind CNN in terms of execution time. they concluded that any algorithm would only have 1 or 2 percent error rate towards the similarity of digits [13] [14]

A.K. Jain [9] have presented an approximation based KNN classification algorithm for handprinted digit recognition. He performed matching on two character's deforming edges and dissimilarities. Their proposed work is on patterns in low-dimensional space where the scaling is 2000 times lesser results 99.25% accuracy.

R.Alhajj [10] has applied a completely new approach called the agent-oriented approach. In a sentence, they appoint agents to each character where their only purpose is to identify hills and valleys, which are nothing but blacks and whites. The ability of agents to socialize with each other is highlighting features compared with any other image classification techniques. Overlapping digits are identified based on the cut-points, which is nothing but the intersection of agent paths. The results of their methods are surprisingly higher when compared to many other ANN-based approaches at about 97% accuracy.

3. METHODOLOGY

In this paper, we used MNIST as a primary dataset to train the model, and it consists of 70,000 handwritten raster images from 250 different sources out of which 60,000 are used for training, and the rest are used for training validation. MNIST data is represented in the IDX file format and are look like in figure 1. Our proposed method mainly separated into stages, as shown in Figure 2, pre-processing, Data Encoding, Model Construction, Training & Validation, Model Evaluation & Prediction. Since the loading dataset is necessary for any process, all the steps come after it.

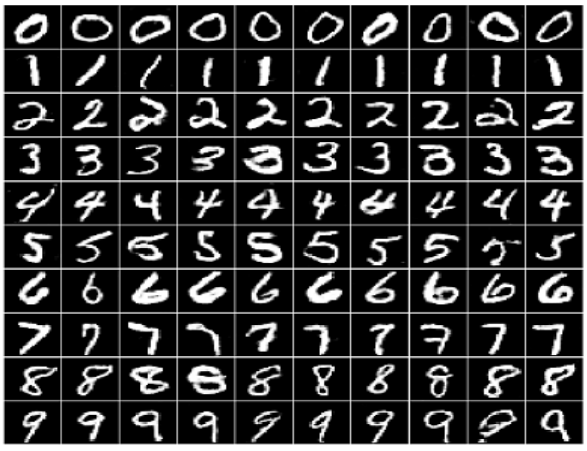


Figure 1: Sample MNIST data

3.1 Pre-Processing

After loading the data, we separated the data into X and y where X is the image, and y is the label corresponding to X. As shown in figure 3, the first layer/input layer for our model is convolution. Convolution takes each pixel as a neuron, so we need to reshape the images such that each pixel value is in its own space, thus converting a 28x28 matrix of greyscale values into 28x28x1 tensor. With the right dimensions for all the images, we can split the images into train and test for further steps [19][20][21].

Figure 2: Flowchart

3.2 Data Encoding

This is an optional step since we are using the cross-categorical entropy as loss function; we have to specify the network that the given labels are categorical in nature.

3.3 Model Construction

After data encoding, the images and labels are ready to be fitted into our model [22] [23]. Summary of the model can be seen in Figure 4

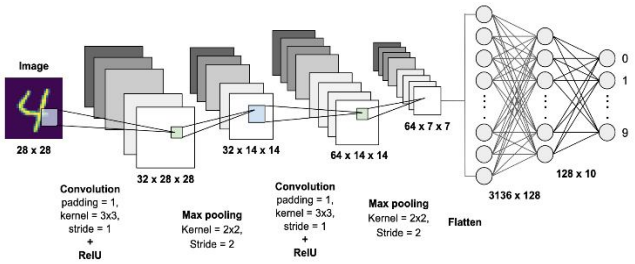


Figure 3: Proposed model

Our model is composed of feature extraction with convolution and binary classification. Convolution and max-pooling are carried out to extract the features in the image, and a 32 3x3 convolution filters are applied to a 28x28 image followed by a max-pooling layer of 2x2 pooling size followed by another convolution layer with 64 3x3 filters. In the end, we obtain 7x7 images to flatten. Flatten layer will flatten the 7x7 images into a series of 128 values that will be mapped to a dense layer of 128 neurons that are connected to the categorical output layer of 10 neurons.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               204928
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 225,034
Trainable params: 225,034
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: Model Summary

3.4 Training & Validation

After building the model [24], we compiled a model with adam optimizer and particular cross-entropy loss function, which are standard in making a convolution neural network. Once the model is successfully assembled, then we can train the model with training data for 100 iterations, but as the number of iteration increases, there is a chance for overfitting. Therefore we limit the training up to 98% accuracy, as we are using real-world data for prediction, test data was used to validate the model [16].

3.5 Model Evaluation & Prediction

For real-world image classification prediction, we need to do a little image pre-processing on the real-world images as model training was done with greyscale raster images. The steps of image pre-processing are,

1. Loading image
2. Convert the image to greyscale
3. Resize the image to 28x28
4. Converting the image into a matrix form
5. Reshape the matrix into 28x28x1

After pre-processing, we predict the label of the image by passing the pre-processed image through the neural network. The output we get is a list of 10 activation values 0 to 9, respectively. The position having the highest value is the predicted label for the image [18].

4. RESULTS AND DISCUSSION

Our model is built to work on real-world data, and real-world images are not even close to MNIST raster images, a lot of pre-processing was done to make a real image to look like a raster image.

4.1 Accuracy score

Our model stopped training at the 2nd epoch as it reached 98.21% training accuracy and 98.51% validation accuracy with 5% training loss and 4% validation loss. The progression of accuracy and loss are represented in Figure 5.

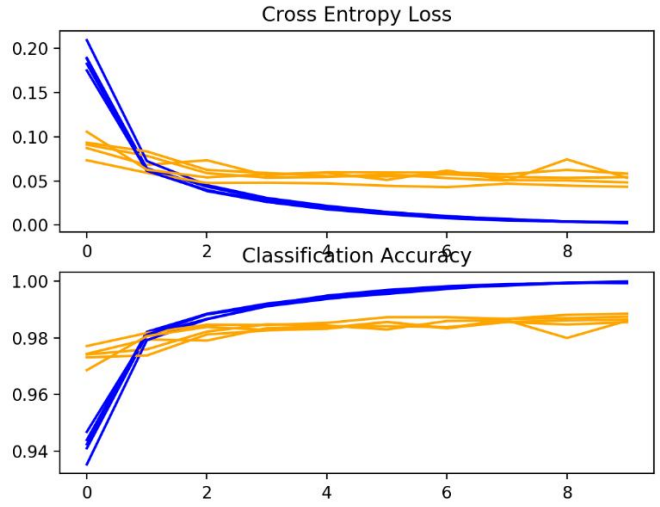


Figure 5: Loss and Accuracy Learning Curves

From the above curve, we can observe that the loss and accuracy are cooperatively changed at every fold during k-fold cross-validation. Before two folds, efficiency almost reached 98%, and that's why the number of iterations stopped at the 2nd epoch. Stability inaccuracy score can be observed from 2nd iteration.

4.2 Prediction

Our model is able to recognize computer-generated digits as well as handwritten digits. Computer-generated digit prediction is more accurate compared to real-world digit prediction, which can be observed in Figure 6.

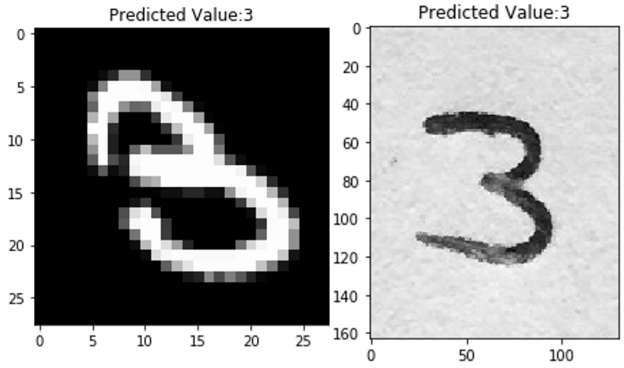


Figure 6: Raster vs. Real image prediction

5. CONCLUSION

The performance of CNN for handwritten recognition performed significantly. The proposed method obtained 98% accuracy and is able to identify real-world images as well; the loss percentage in both training and evaluation is less than 0.1, which is negligible. The only challenging part is the noise present in the real-world image, which needs to look after. The learning rate of the model is much dependent on the number of dense neurons and the cross-validation measure

REFERENCES

1. Y. Yin, W. Zhang, S. Hong, J. Yang, J. Xiong, and G. Gui, "**Deep Learning-Aided OCR Techniques for Chinese Uppercase Characters in the Application of Internet of Things**," in IEEE Access, vol. 7, pp. 47043-47049, 2019. https://doi.org/10.1109/ACCESS.2019.2909401
2. N. B. Venkateswarlu, "**New raster, adaptive document binarization technique,**" in Electronics Letters, vol. 30, no. 25, pp. 2114-2115, 8 Dec. 1994. https://doi.org/10.1049/el:19941439
3. S. Aly and A. Mohamed, "**Unknown-Length Handwritten Numeral String Recognition Using Cascade of PCA-SVMNet Classifiers**," in IEEE Access, vol. 7, pp. 52024-52034, 2019.
4. Seong-Whan Lee and Sang-Yup Kim, "**Integrated segmentation and recognition of handwritten numerals with cascade neural network**," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 29, no. 2, pp. 285-290, May 1999. https://doi.org/10.1109/5326.760572
5. Bailing Zhang, Minyue Fu, Hong Yan, and M. A. Jabri, "**Handwritten digit recognition by adaptive-subspace self-organizing map (ASSOM)**," in IEEE Transactions on Neural Networks, vol. 10, no. 4, pp. 939-945, July 1999.
6. L. Deng, "**The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web],**" in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, Nov. 2012. https://doi.org/10.1109/MSP.2012.2211477
7. S.Sudhakar, V.Vijayakumar, C.SathiyaKumar, V.Priya, LogeshRavi, V.Subramaniyaswamy, **Unmanned Aerial Vehicle (UAV) based Forest Fire Detection and monitoring for reducing false alarms in forest-fires**, Elsevier- Computer Communications 149 (2020) 1–16, https://doi.org/10.1016/j.comcom.2019.10.007
8. M. M. A. Ghosh and A. Y. Maghari, "**A Comparative Study on Handwriting Digit Recognition Using Neural Networks,**" 2017 International Conference on Promising Electronic Technologies (ICPET), Deir El-Balah, 2017, pp. 77-81.
9. A. K. Jain and D. Zongker, "**Representation and recognition of handwritten digits using deformable**

templates," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 12, pp. 1386-1390, Dec. 1997.

10. S.Sudhakar, V.Vijayakumar, C.SathiyaKumar, V.Priya, LogeshRavi, V.Subramaniyaswamy, **Unmanned Aerial Vehicle (UAV) based Forest Fire Detection and monitoring for reducing false alarms in forest-fires**, *Elsevier- Computer Communications* 149 (2020) 1–16, https://doi.org/10.1016/j.comcom.2019.10.007
11. R. Alhajj and A. Elnagar, "**Multiagents to separating handwritten connected digits,**" in IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, vol. 35, no. 5, pp. 593-602, Sept. 2005. https://doi.org/10.1109/TSMCA.2005.843389
12. H. I. Avi-Itzhak, T. A. Diep and H. Garland, "**High accuracy optical character recognition using neural networks with centroid dithering,**" in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 17, no. 2, pp. 218-224, Feb. 1995. https://doi.org/10.1109/34.368165
13. Naimin Zhang, Wei Wu, and Gaofeng Zheng, "**Convergence of gradient method with momentum for two-Layer feedforward neural networks,**" in IEEE Transactions on Neural Networks, vol. 17, no. 2, pp. 522-525, March 2006.
14. R.Vasanthi, R.Jayavadivel, K.Prasadh, J.Vellingiri, G.A kilarasu, S.Sudhakar, P.M.Balasubramaniam, **A novel user interaction middleware component system for ubiquitous soft computing environment by using the fuzzy agent computing system**, Journal of Ambient Intelligence and Humanized Computing (2020), Springer, doi.org/10.1007/s12652-020-01893-4.
15. Jagadeesh Gopal, Vellingiri J, Gitanjali J, Arivuselvan K, Sudhakar S, **An Improved Trusted On-Demand Multicast Routing with QoS for Wireless Networks**, International Journal of Advanced Trends in Computer Science and Engineering, Vol.9,No.1, January – February 2020, P.P.:261-265, https://doi.org/10.30534/ijatcse/2020/39912020.
16. Satheesh N, Sudha D, Suganthi D, Sudhakar S, Dhanaraj S, Sriram V.P., Priya V, "**Certain improvements to Location aided packet marking and DDoS attacks in internet**," Journal of Engineering Science and Technology Vol. 15, No. 1 (2020), pp: 94 - 107, School of Engineering, Taylor's University.
17. Sathiya Kumar C, Priya V, Sriram V P, Sankar Ganesh K, Murugan G, Devi Mani, Sudhakar S, "**An Efficient Algorithm for Quantum Key distribution with Secure Communication**, Journal of Engineering Science and Technology Vol. 15, No. 1 (2020), pp:77-93, School of Engineering, Taylor's University.
18. S.Sudhakar, N.Satheesh, S.Balu, Amireddy Srinish Reddy, G.Murugan,2019,"**Optimizing Joins in a Map-Reduce for Data Storage and Retrieval Performance Analysis of Query Processing in HDFS for Big Data,"** International Journal of Advanced Trends in Computer Science and Engineering, (IJATCSE), Vol.8 No.5,pp:2062-2067, DOI:10.30534/ijatcse/2019/33852019.
19. K.B., Prakash. **"Information extraction in current Indian web documents."** International Journal of and Technology(UAE), 2018: 68-71. https://doi.org/10.14419/ijet.v7i2.8.10332
20. Kolla B.P., Dorairangaswamy M.A., Rajaraman A. **"A neuron model for documents containing multilingual Indian texts."** 2010 International Conference on Computer and Communication Technology, ICCCT-2010,2010: 451-454. https://doi.org/10.1109/ICCCT.2010.5640489
21. Kolla B.P., Raman A.R. **"Data Engineered Content Extraction Studies for Indian Web Pages."** Advances in Intelligent Systems and Computing, 2019: 505-512. https://doi.org/10.1007/978-981-10-8055-5_45
22. Prakash K.B., Dorai Rangaswamy M.A. **"Content extraction studies using neural network and attributegeneration."** Indian Journal of Science and Technology, 2016: 1-10.
23. Prakash K.B., Dorai Rangaswamy M.A., Raman A.R. **"Text studies towards multi-lingual content mining for web communication."** Proceedings of the 2nd International Conference on Trendz in Information Sciences and Computing, TISC-2010, 2010: 28-31. https://doi.org/10.1109/TISC.2010.5714601
24. Prakash K.B., Rangaswamy M.A.D. **"Content extraction of biological datasets using soft computing techniques.",** Journal of Medical Imaging and Health Informatics, 2016: 932-936. https://doi.org/10.1166/jmihi.2016.1931

Handwritten Digit Recognition using Machine Learning in Python

S.V.S.S. Lakshmi
Assistant Professor, Department of Computer Science and Engineering, Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam, Andhra Pradesh, India.
lakshmi.cse@anits.edu.in

K. Sri Pragnya
Student, Department of Computer Science and Engineering, Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam, Andhra Pradesh, India.
prajju.17.cse@anits.edu.in

K. Rama Krishna
Student, Department of Computer Science and Engineering, Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam, Andhra Pradesh, India.
ramakrishnakoppala.17.cse@anits.edu.in

B. Sri Sahithi
Student, Department of Computer Science and Engineering, Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam, Andhra Pradesh, India.
sri.sahithi.17.cse@anits.edu.in

M. Lokesh
Student, Department of Computer Science and Engineering, Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam, Andhra Pradesh, India.
mlokesh.l18.cse@anits.edu.in

***Abstract*- Handwritten Digit Recognition is one of the essentially significant issues in pattern recognition applications. The main purpose of this project is to build an automatic handwritten digit recognition method for the recognition of handwritten digit strings. This paper proposes a simple convolution neural network approach to handwritten digit recognition. Convolutional Neural Network model is implemented using MNIST dataset. This dataset consists 60,000 small square 28×28pixel grayscale images of handwritten single digits between 0 and 9. The applications of digit recognition include postal mail sorting, check processing, form data entry, etc. The core of the issue exists in the capacity to foster a proficient calculation that can perceive manually written digits and which is put together by clients by the method of a scanner, tablet, and other computerized gadgets.**

Keywords – Pattern Recognition, Digit Recognition, Convolutional Neural Networks, MNIST dataset.

I. INTRODUCTION

Machine Learning and Deep learning assume a significant part in computer technology and artificial intelligence. With the utilization of deep learning and machine learning, human effort is often reduced in recognizing, learning, predictions and lots of more areas. This article presents perceiving the transcribed digits (0 to 9) from the acclaimed MNIST dataset, using a convolution neural network model on basis of performance, accuracy, time and efficiency. To make machines more intelligent, developers are diving into machine learning and deep learning techniques. A human learns to perform a task by practicing and repeating it again and again so that it memorizes how to perform the tasks. Then the neurons in his brain automatically trigger and that they can quickly perform the task they learnt. Deep learning is also very similar to this. It uses different types of neural network architectures for various sorts of problems. For example – visual perception, image and sound classification, object detection, image segmentation, etc. Handwritten digit recognition is that the ability of computers to acknowledge human handwritten digits. It is a difficult task for the machine because handwritten digits aren't always perfect and may be written in different styles. Handwritten digit recognition is the solution for the current issue which utilizes the picture of a digit and perceives the digits present within in the image. Digit Recognition System: Digit recognition system is that the working of a machine to teach itself or understanding and perceiving the digits from different sources like messages, bank cheques, papers, images, etc. and in various real-world scenarios for online handwriting recognition on computers, recognize number plates of vehicles, processing bank cheques, numeric entries in forms filled manually, etc.

II. Related Work

An early notable attempt in the area of character recognition research is by Grimsdale in 1959. The origin of a great deal of research work in the early sixties was based on an approach known as analysis-by-synthesis method suggested by Eden in 1968. The great importance of Eden's work was that he formally proved that all handwritten characters are formed by a finite number of schematic features, a point that was implicitly included in previous works.This notion was later used in all methods in syntactic (structural) approaches of character recognition.

1. **K. Gaurav, Bhatia P. K. ,** his paper deals with the various pre-processing techniques involved in the character recognition with different kind of images ranges from a simple handwritten form based documents and documents containing colored and complex background and varied intensities.

   In this, different preprocessing techniques like skew detection and correction, image enhancement techniques of contrast stretching, binarization, noise removal techniques, normalization and segmentation, morphological processing techniques are discussed.

1. **Sandhya Arora** , used four feature extraction techniques namely, intersection, shadow feature, chain code histogram and straight line fitting features.
   Shadow features are computed globally for character image while intersection features, chain code histogram features and line fitting features are computed by dividing the character image into different segments.
   On experimentation with a dataset of 4900 samples the overall recognition rate observed was 92.80% for Devanagari characters.

2. **Brakensiek, J. Rottland, A. Kosmala, J. Rigoll**, in their paper a system for off-line cursive handwriting recognition is described which is based on Hidden Markov Models (HMM) using discrete and hybrid modelling techniques.Handwriting recognition experiments using a discrete and two different hybrid approaches, which consist of a discrete and semi-continuous structures, are compared.

III. Methodolgy

We used MNIST as a primary dataset to train the model, and it consists of 70,000 handwritten raster images from 250 different sources out of which 60,000 are used for training, and the rest are used for training validation. Our proposed method mainly separated into stages, Pre-Processing, Model Construction, Training & Validation, Model Evaluation & Prediction.

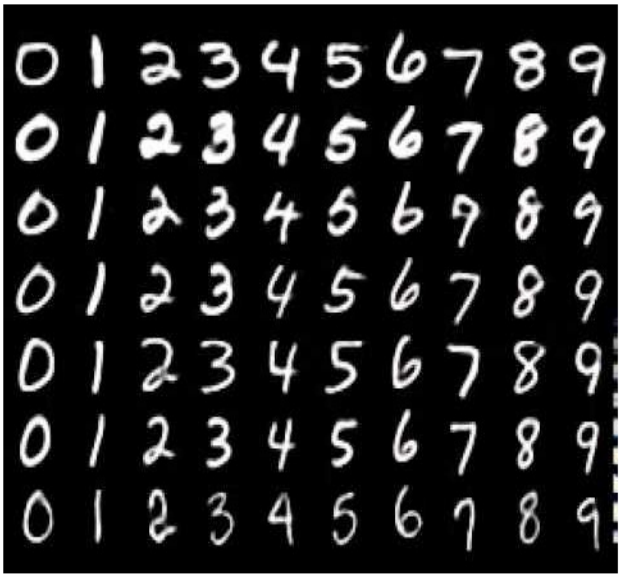


Figure1 Sample MNIST dataset

2.1 Pre-Processing:

Data preprocessing plays a key role in any recognition process. Data preprocessing is a data mining procedure that is used to change the raw data to a useful and efficient format. To shape the input images in a structure appropriate for segmentation, pre-processing is used. The image data cannot be fed directly into the model so we need to perform some operations and process the data to make it ready for our neural network. The dimension of the training data is (60000,28,28).

CNN model will require one more dimension so we reshape the matrix to shape (60000,28,28,1).

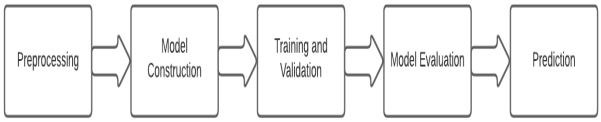


Figure 2 Methodology Flow Chart

2.2 Model Construction:

After preprocessing the images and labels are ready to be fed into our model. A CNN model was built using the Keras library. A CNN model usually consists of convolutional and pooling layers. It works better for the data that are represented as grid structures, this is often the reason why CNN works well for image classification problems. The dropout layer is used to deactivate some of the neurons and while training, it reduces the overfitting of the model. Adadelta optimizer is used to compile the model.

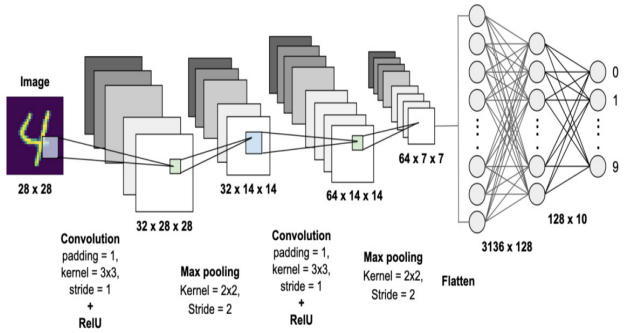


Figure 3 Model Architecture

2.3 Training & Validation:

After building the model, we compiled a model with adam optimizer and particular cross-entropy loss function which are standard in making a convolution neural network. Once the model is successfully constructed, then we can train the model with training data for the required number of iterations, but as the number of iteration increases, there is a chance for overfitting. Therefore we limit the training up to 98% accuracy, as we are using real-world data for prediction, test data was used to validate the model.

2.4 Model Evaluation:

Model Evaluation is an important part of CNN model development. It is used to find the efficient model that represents our data and how well the constructed model will work in the future. The MNIST dataset consists of 10,000 images which will be used to evaluate how well our model works. The testing data was not included in the training of the data therefore, it is new for our model. The MNIST dataset is well balanced so we can get good accuracy.

2.5 Prediction:

The Tkinter library in the Python standard library is used for this task. A function predict_digit() is defined that takes the image as input and then trained model is used to predict the digit. Then an App class is defined which is responsible for building the GUI for our project. The canvas created provides a window where we can draw by capturing the mouse event and with a button, we invoke the predict_digit() function and display the results.

IV. RESULTS

4.1 Accuracy Score:

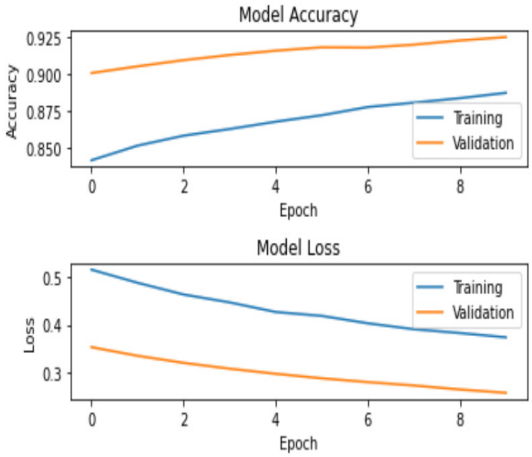


Figure 4 Accuracy and Loss Curve

From the above curve we can say that accuracy during training and validation has increased with increase in number of epochs and loss has been subsequently decreases during training and validation.

4.2 Prediction:

The output digit will be displayed along with its accuracy on the window created. Our model gives a good accuracy score with almost 90% prediction rate.

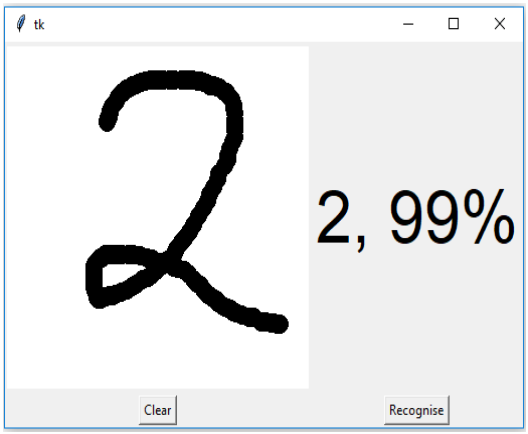


Figure 4 Prediction output of 2

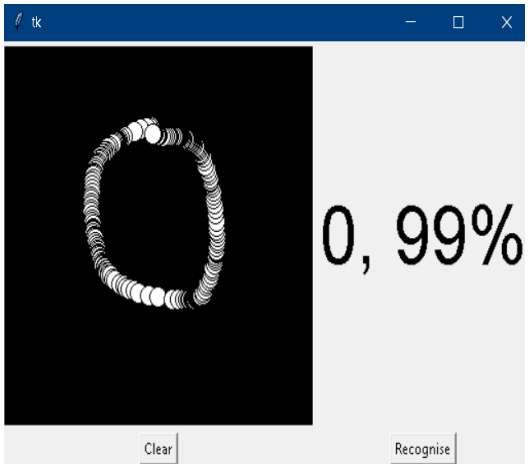


Figure 5 Prediction output of 0

V.CONCLUSION

The performance of CNN for handwritten recognition performed significantly. The proposed method obtained around 98% accuracy and is able to identify real-world images as well; the loss percentage obtained in both training and evaluation is almost negligible. The only difficult part is the noise present in the input canvas image, which needs to be taken care of. The learning rate of the model is much dependent on the number of dense neurons and the cross-validation measure.

REFERENCES

[1] https://www.geeksforgeeks.org/python-tkinter-tutorial/

[2] https://medium.com/the-andela-way/applying-machine-learning-to-recognize handwritten-characters-babcd4b8d705

[3] https://nanonets.com/blog/handwritten-character-recognition/

[4] https://www.tensorflow.org/learn

[5] R. Alhajj and A. Elnagar, "Multiagents to separating handwritten connected digits," in IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, vol. 35, no. 5, pp. 593-602, Sept. 2005. https://doi.org/10.1109/TSMCA.2005.843389

[6] https://towardsdatascience.com/useful-plots-to-diagnose-your-neural-network-521907fa2f45

[7] https://towardsdatascience.com/the-best-machine-learning-algorithm-for-handwritten-digits-recognition-2c6089ad8f09

[8] https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python

[9] https://www.analyticsvidhya.com/blog/2021/01/building-a-cnn-model-with-95-accuracy/

[10] https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/

[11] https://www.kdnuggets.com/2018/11/top-python-deep-learning-libraries.html

[12] https://analyticsindiamag.com/7-types-classification-algorithms/

[13] https://towardsdatascience.com/image-recognition-with-machine-learning-on-python-image-processing-3abe6b158e9a