**1. INTRODUCTION**

**1.1 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 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.

**1.1.1 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.

**1.2 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.

**2. UML DIAGRAMS**

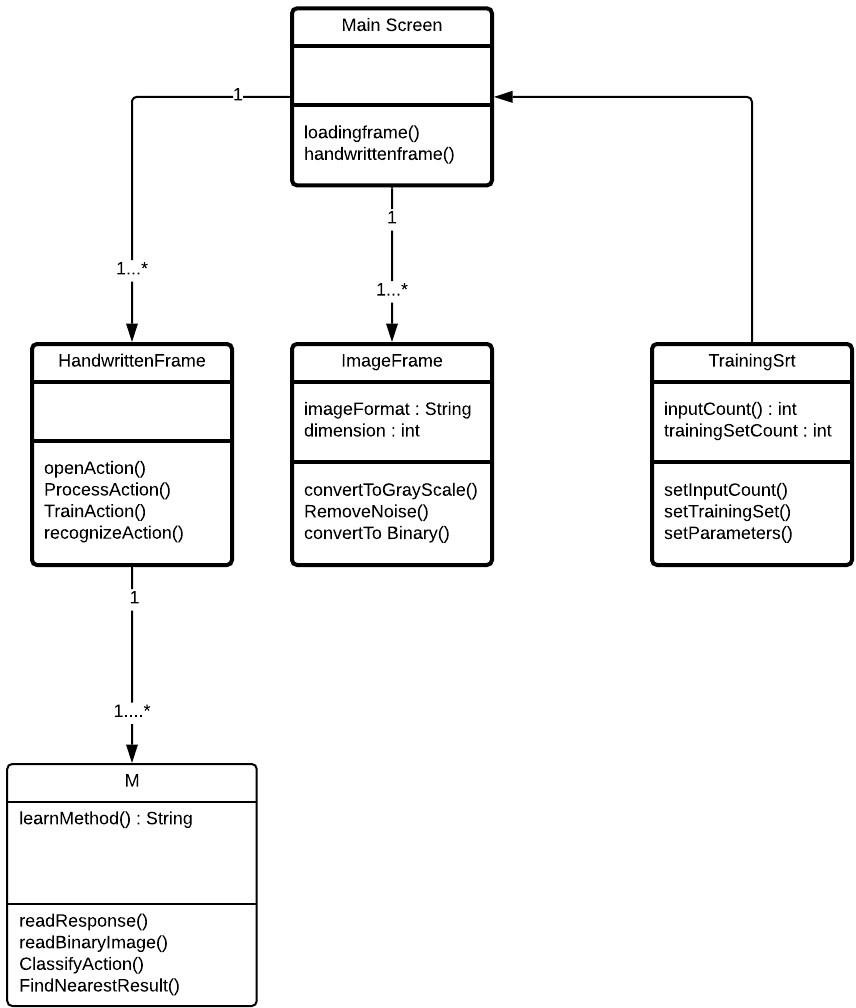
A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of **visually representing a system** along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system. UML is an acronym 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](https://tallyfy.com/business-process-modeling-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.

**2.1 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.



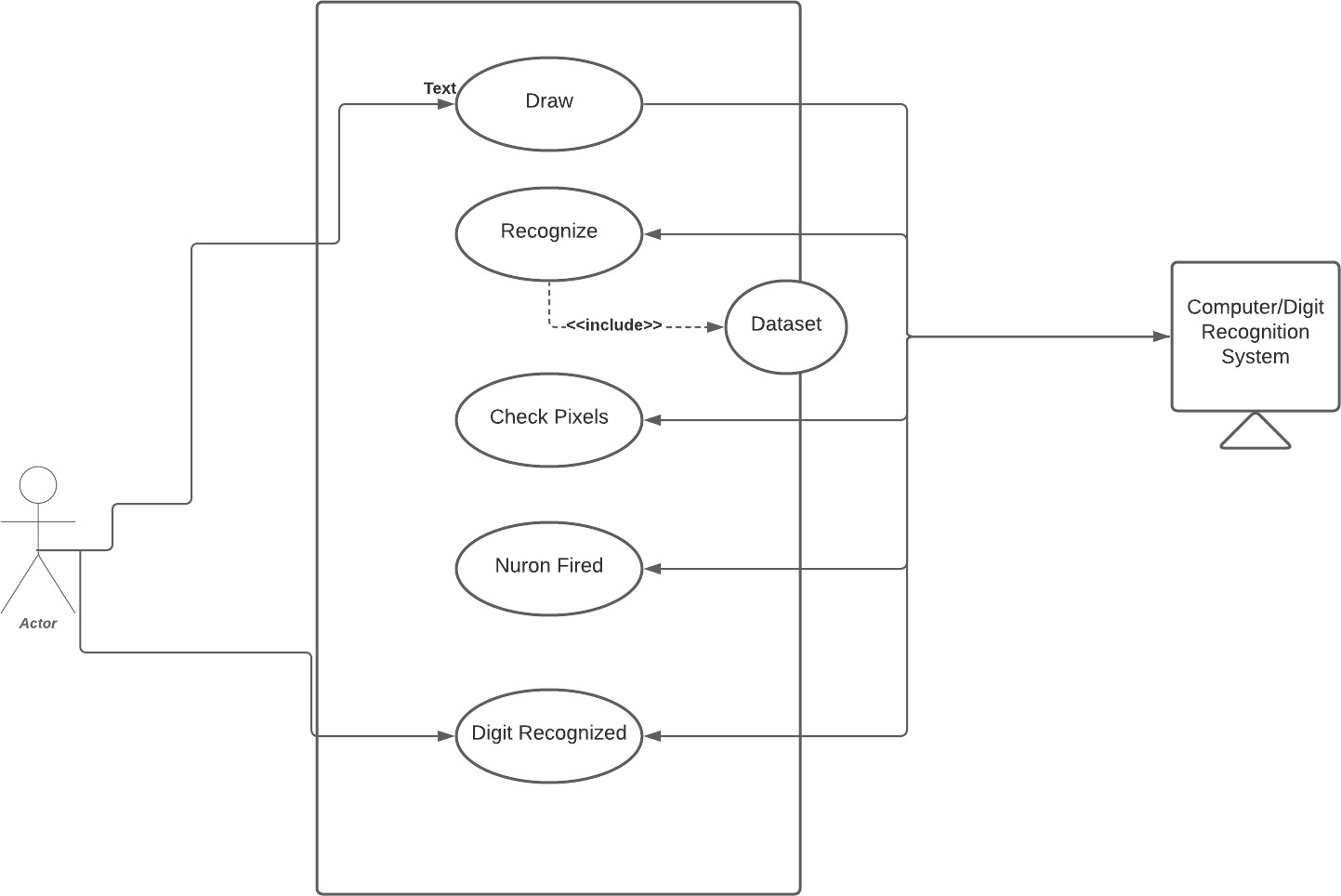
**Figure 1 Class Diagram**

**2.2 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**

**3. METHODOLOGY**

**3.1 Basic Steps in Constructing a Machine Learning M3.odel**

**3.1.1 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

**3.1.2 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

**3.1.3 Choose a Model:**

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

**3.1.4 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

**3.1.5 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.

**3.1.6 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.

**3.1.7 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

**3.2 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**

**Deep learning**

Deep learning is an [artificial intelligence (AI)](https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp) 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](https://www.investopedia.com/terms/m/machine-learning.asp) [learning](https://www.investopedia.com/terms/m/machine-learning.asp) 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.

**3.2.1 Import the Libraries:**

Libraries required are Keras, Tensor flow, Numpy

1. **Keras:**

Keras is a powerful and easy-to-use free open source Python library for developing and evaluating [**deep**](https://machinelearningmastery.com/what-is-deep-learning/)[**learning**](https://machinelearningmastery.com/what-is-deep-learning/)models**.**

It wraps the efficient numerical computation libraries [**Theano**](https://machinelearningmastery.com/introduction-python-deep-learning-library-theano/)and [**TensorFlow**](https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/)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.

1. **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**](https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/)**.** 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.

1. **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.

1. **Matplotlib**

Matplotlib is a popular Python library used for creating high-quality 2D plots and graphs. It provides a wide range of customizable features for visualizing data in various formats such as line plots, scatter plots, bar charts, histograms, and many more.

Matplotlib was originally created by John D. Hunter in 2003 as an alternative to MATLAB's plotting capabilities. Since then, it has become one of the most widely used data visualization libraries in the Python ecosystem, and is often used in conjunction with other data analysis libraries such as NumPy and Pandas.

Matplotlib provides a comprehensive set of functions for creating and customizing plots, as well as support for a wide range of output formats including PNG, PDF, and SVG. It can also be integrated with interactive tools such as Jupyter notebooks to create dynamic visualizations.

Overall, Matplotlib is a powerful and flexible tool for creating professional-grade plots and charts in Python, making it an essential component in the toolkit of many data scientists and analysts.

1. Bottom of Form

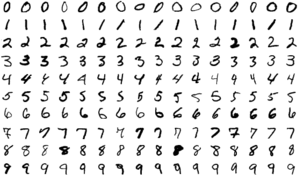
**3.3 Loading the Data Set**

**3.3.1 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

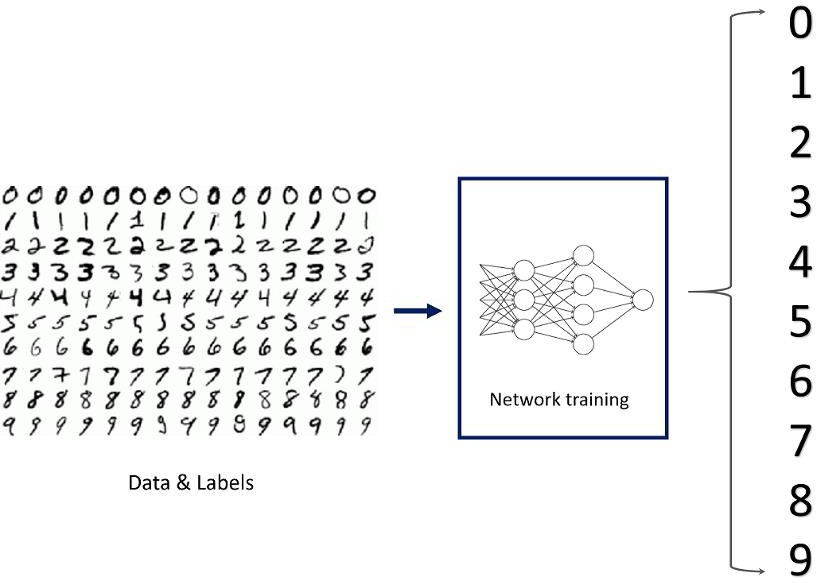
binary images of handwritten digits. The training set contains handwritten digits from 250 people, among them 50% training dataset was employees from the Census Bureau and the rest of it was from high school students. However, it is often attributed as the first datasets among

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 Example**

**3.4 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](https://serokell.io/blog/data-preprocessing#data-quality-assessment)
* [Data cleaning](https://serokell.io/blog/data-preprocessing#data-cleaning)
* [Data transformation](https://serokell.io/blog/data-preprocessing#data-transformation)
* [Data reduction](https://serokell.io/blog/data-preprocessing#data-reduction)

**3.4.1 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](https://www.experian.co.uk/business/glossary/data-cleansing/index) [the data](https://www.experian.co.uk/business/glossary/data-cleansing/index) 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.

**3.4.2 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.

**3.4.3 Data Transformation:**

In fact, by cleaning and smoothing the data, we have already performed data modification. However, by [data transformation](https://www.geeksforgeeks.org/data-transformation-in-data-mining/?ref=rp), 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.

**3.4.4 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:
   1. Wavelet Transform
   2. Principal Component Analysis
   3. Attribute Subset Selection
2. Numerosity Reduction:
   1. Parametric
   2. 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.

**3.5 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.

**3.6 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.

**3.6.1 Learning Algorithms:**

Machine learning algorithms could be broadly categorized 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”.

1. **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.

**3.** **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 outputs 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.

**3.6.2 Model that can be used for the project:**

1. **CONVOLUTION NEURAL NETWORK**

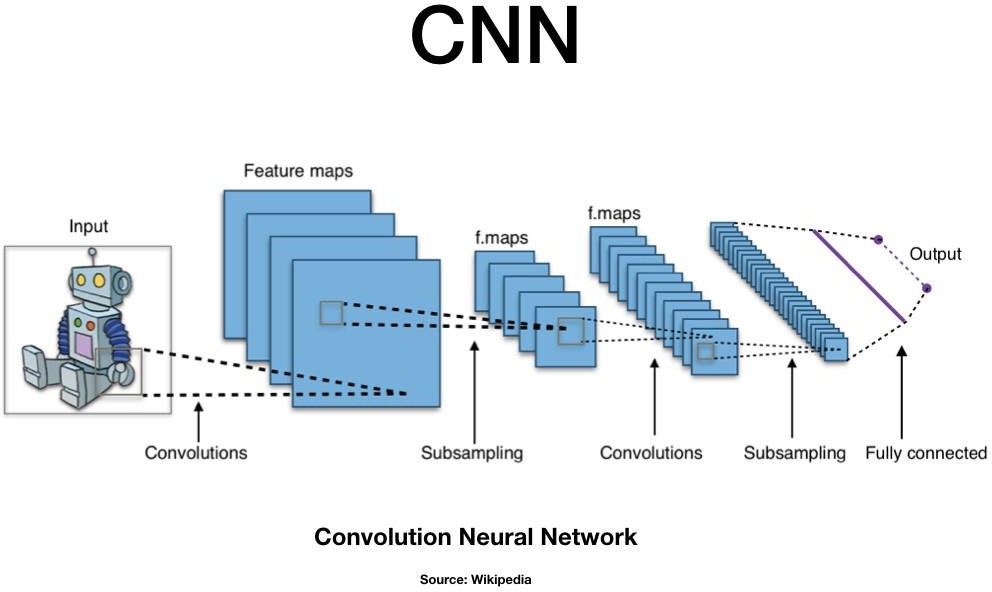
In simpler words, CNN is an artificial neural network that specializes in picking out or

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 6 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](https://en.wikipedia.org/wiki/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*”.*

1. **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.

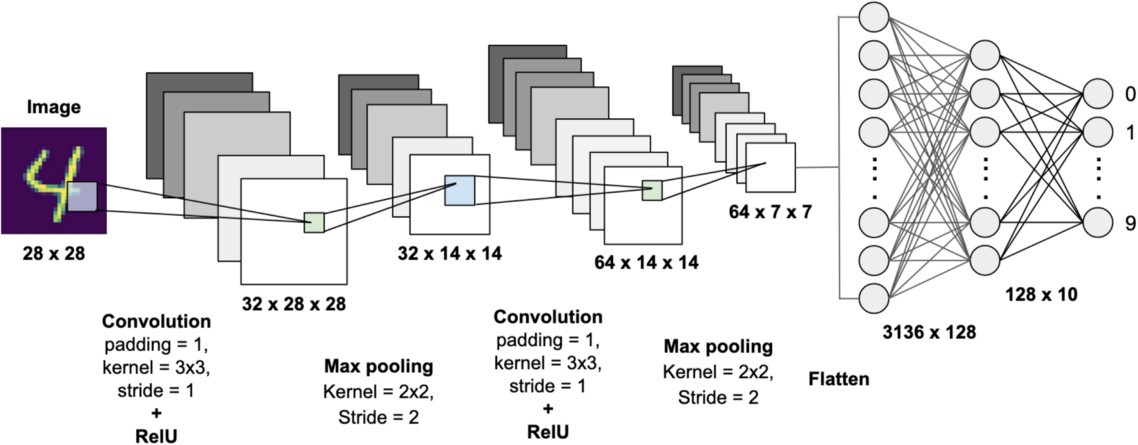
* Convolutional neural networks apply a filter to an input to create a feature map that summarizes the presence of detected features in the input.
* Filters can be handcrafted, such as line detectors, but the innovation of convolutional neural networks is to learn the filters during training in the context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

1. **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 behavior 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 7 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.

**3.7 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

**3.7.1 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](http://dpkingma.com/) from OpenAI and [Jimmy Ba](https://jimmylba.github.io/) from the University of Toronto in their 2015 [ICLR](http://www.iclr.cc/doku.php?id=iclr2015%3Amain) paper (poster) titled “[Adam: A Method for Stochastic Optimization](https://arxiv.org/abs/1412.6980)“.

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 Adma:

* + - * 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.

**3.8 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 preprocessing, 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](https://www.datarobot.com/wiki/training-validation-holdout/) and [Cross-](https://towardsdatascience.com/cross-validation-70289113a072) [validation](https://towardsdatascience.com/cross-validation-70289113a072). 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](https://elitedatascience.com/overfitting-in-machine-learning).

**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](https://machinelearningmastery.com/k-fold-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](https://medium.com/datadriveninvestor/k-fold-cross-validation-6b8518070833), 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.

**4. 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.

**4.1 Basic Architecture**

There are two main parts to a CNN architecture

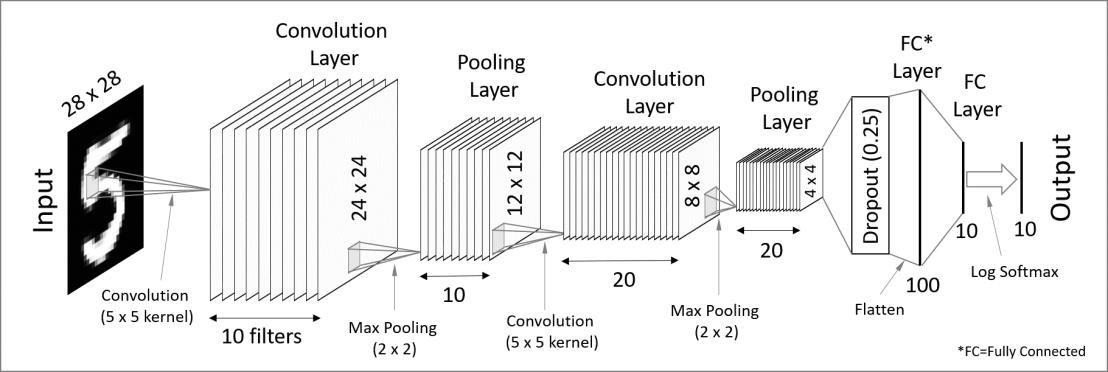
* A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction
* A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.

**4.1.1 CNN Layer:**

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.

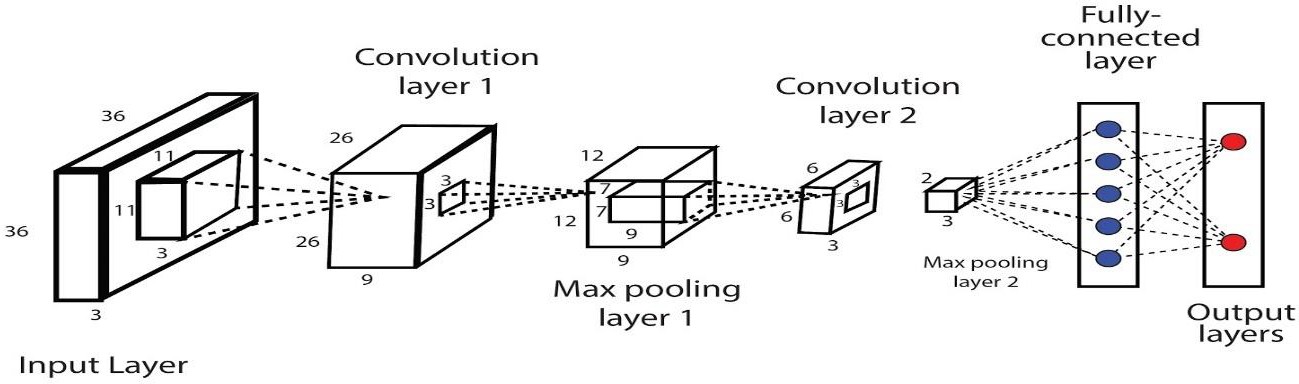
As we go deeper and deeper in the layers, the complexity is increased a lot. But it might be worth going as accuracy may increase but unfortunately, time consumption also increases.



**Figure 8 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).

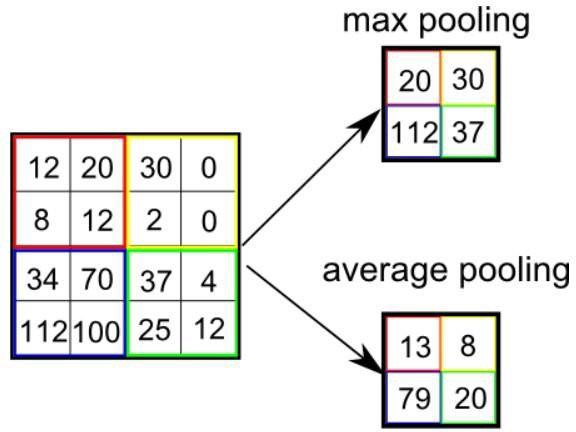


**Figure 9 Convolutional Layer**

**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.

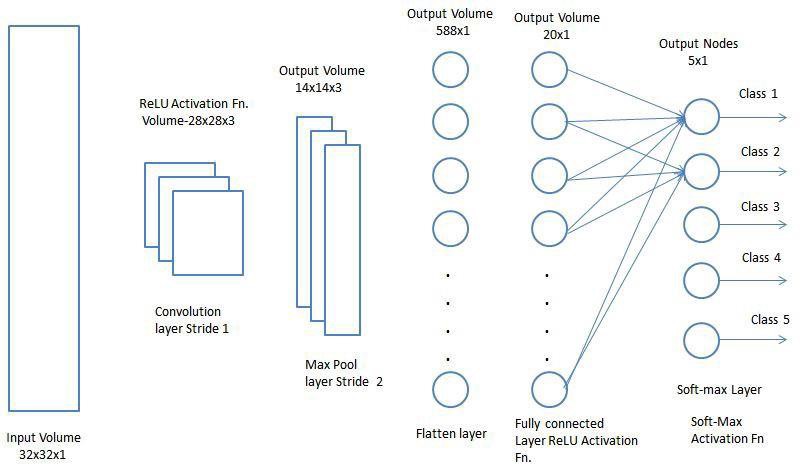
In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional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



**Figure 10 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.

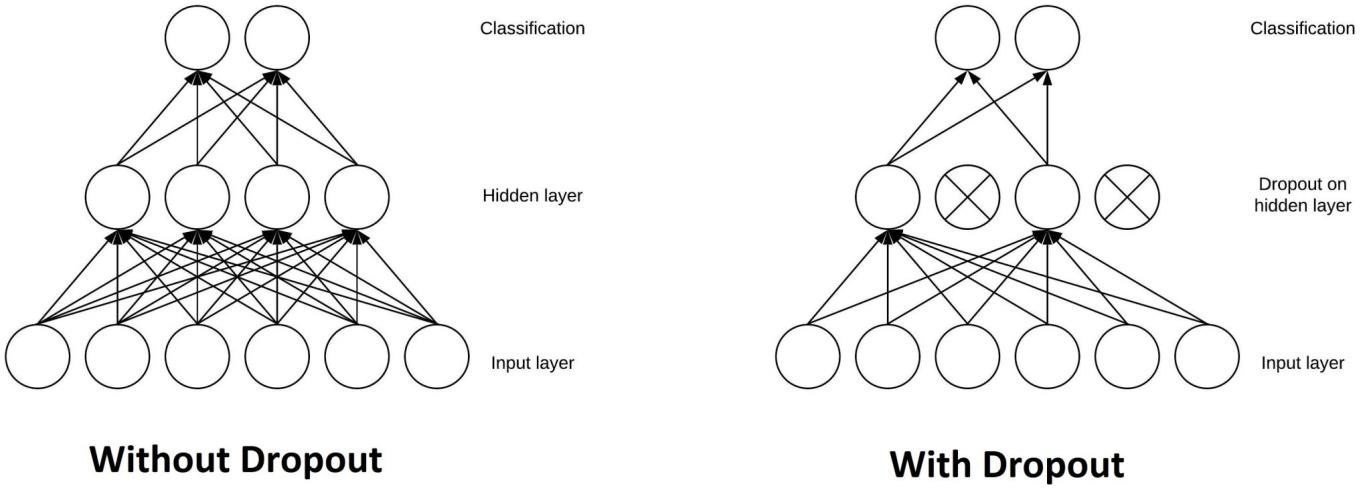


**Figure 11 Fully Connected Layer**

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 12 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. Active Functions**

An [activation function](https://en.wikipedia.org/wiki/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.

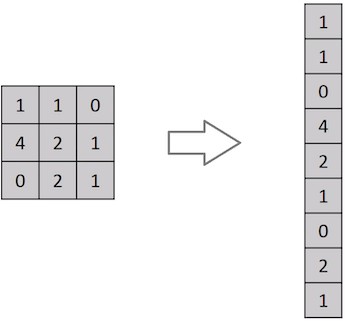
* A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction
* A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.

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](https://arxiv.org/pdf/1509.02634.pdf) to incorporate rich information into an image segmentation model. Researchers from UC Berkeley also built [fully convolutional networks](https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf) that improved upon state-of- the-art semantic segmentation.

**4.1.2 Why ConvNet 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 13 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](https://intellipaat.com/blog/tutorial/artificial-intelligence-tutorial/convolution-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.

**5. EXPERIMENTAL ANALYSIS AND RESULTS**

**5.1 System configuration**

**5.1.1 Software requirements:**

These are the software configurations used:

**Operating system:** windows 11

**IDE:** Google Collab

**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.

**Google Collab:**

Google Colab, short for Google Collaboratory, is a free online platform offered by Google that provides users with an interactive environment for working with Python. It allows users to write and run code in a Jupyter notebook-style interface, without requiring any installation or setup on their local computer.

With Google Colab, users can write and execute Python code, as well as import and manipulate data, create visualizations, and perform machine learning tasks using popular libraries like TensorFlow, PyTorch, and scikit-learn. Google Colab also provides access to GPUs and TPUs for accelerated computation, which is particularly useful for training deep learning models.

Google Colab notebooks are stored on Google Drive, which makes it easy to share them with collaborators and access them from any device with internet access. Additionally, Colab supports real-time collaboration, allowing multiple users to work on the same notebook simultaneously.

**5.1.2 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

**5.2 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](https://www.investopedia.com/terms/p/production_efficiency.asp) and to [forecast costs](https://www.investopedia.com/terms/f/forecasting.asp). 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.

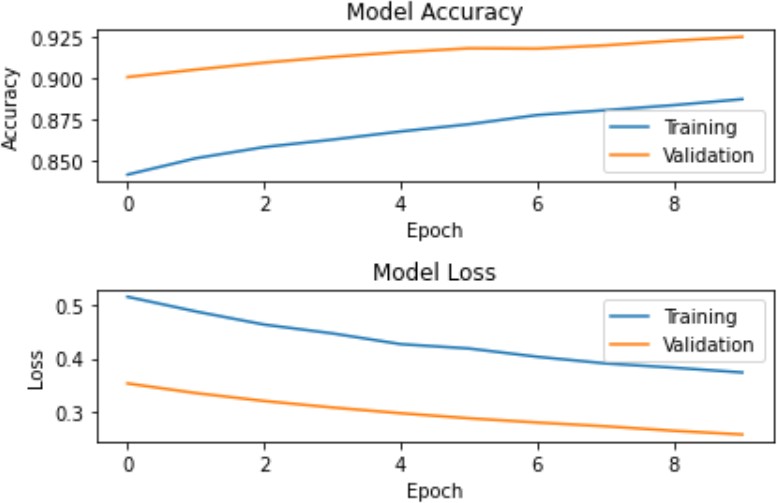
According to this, we’ll have a measure of time or progress in the x-axis and a measure of error or performance in the y-axis.

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](file://localhost/G:/machine-learning/wiki/metrics-in-machine-learning) in machine learning.



**Figure 14 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.

**5.2.1 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.

**5.2.2 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.

**5.3 Sample Code**

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

First, we are going to import all the modules such as NumPy, Matplotlib, TensorFlow 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.

import tensorflow. keras as tf

import matplotlib. pyplot as plt

import numpy as np

import tensorflow\_datasets as tfds

mnist = tf. datasets.mnist

(xtrain, ytrain), (xtest, ytest) = mnist. load\_data()

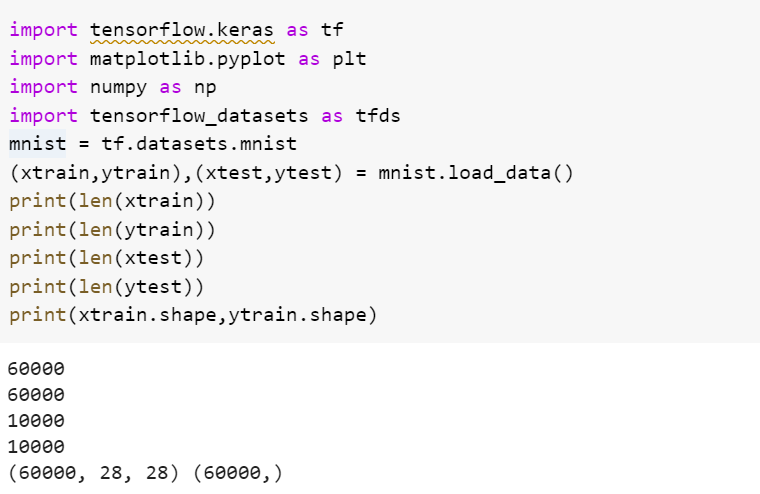
print(len(xtrain))

print(len(ytrain))

print(len(xtest))

print(len(ytest))

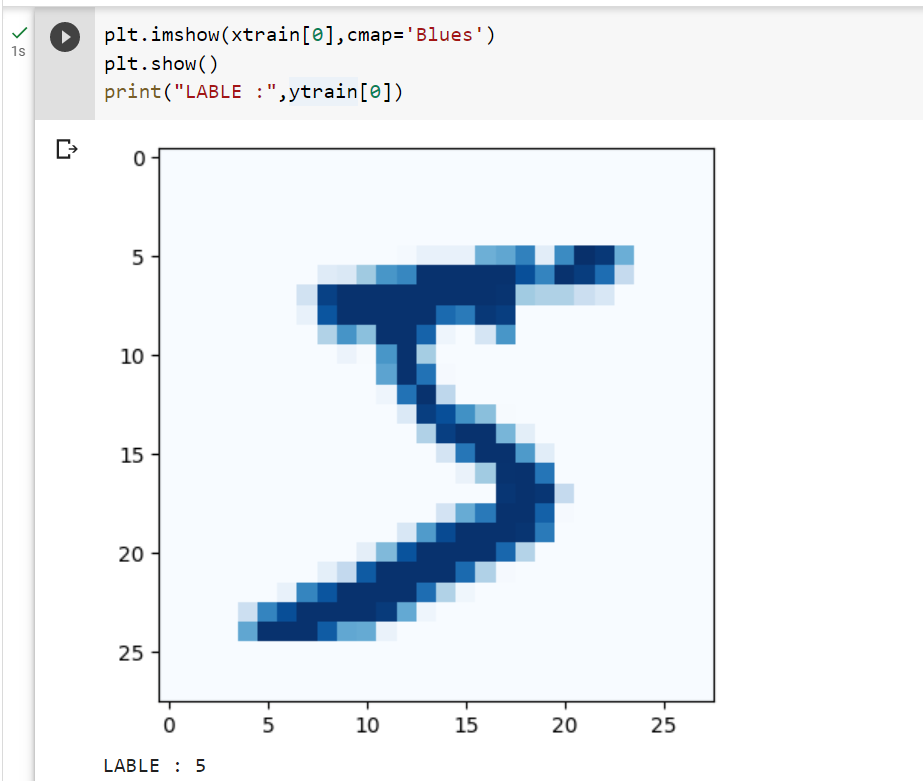
print(xtrain.shape,ytrain.shape)



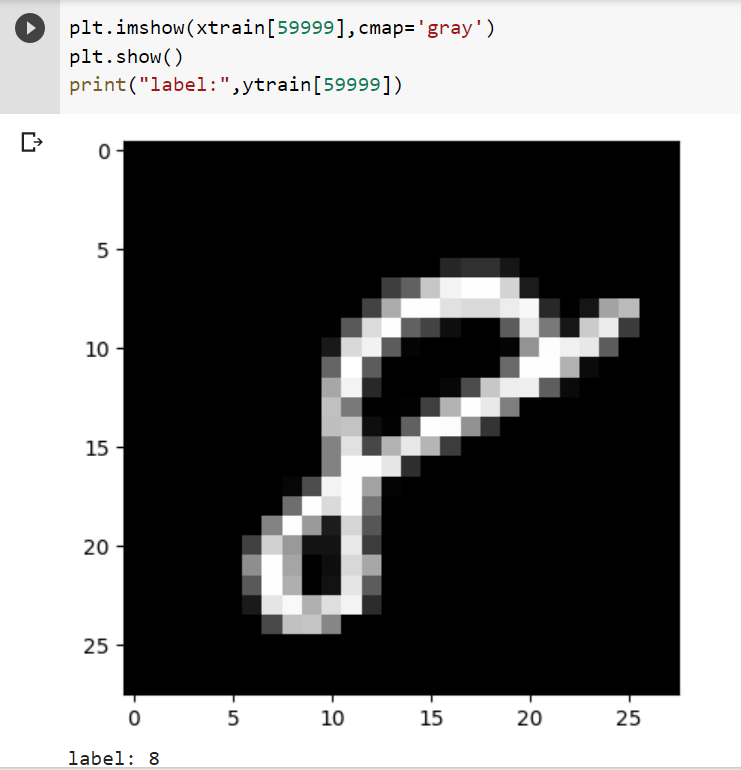
**Figure 15 Output Training and Test Data Shape**

**5.3.2 Testing the data set:**

Testing the MNIST data set by visualizing the digits and labels using matplotlib



**Figure 16 OUTPUT ‘DIGIT 5’**



**Figure 17 OUTPUT ‘8’**

**Visualize first 100 Samples:**

Visualizing 100 samples From MNIST DATA SET to test the Data

plt.figure(figsize=(20,20))

for i in range(100):

  plt.subplot(10,10,i+1)

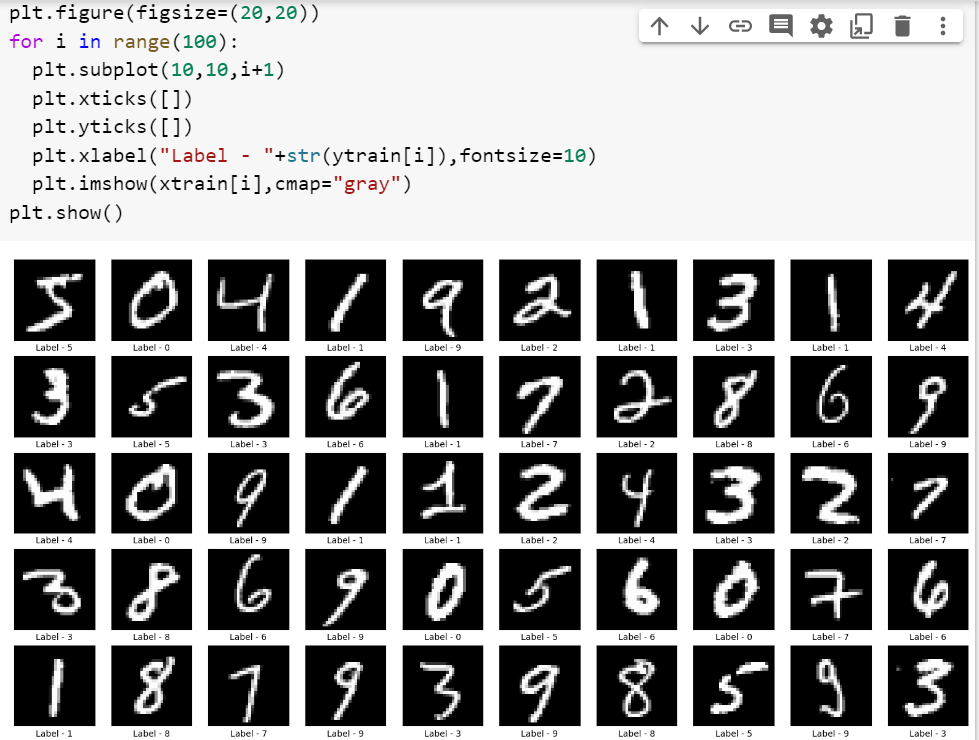
  plt.xticks([])

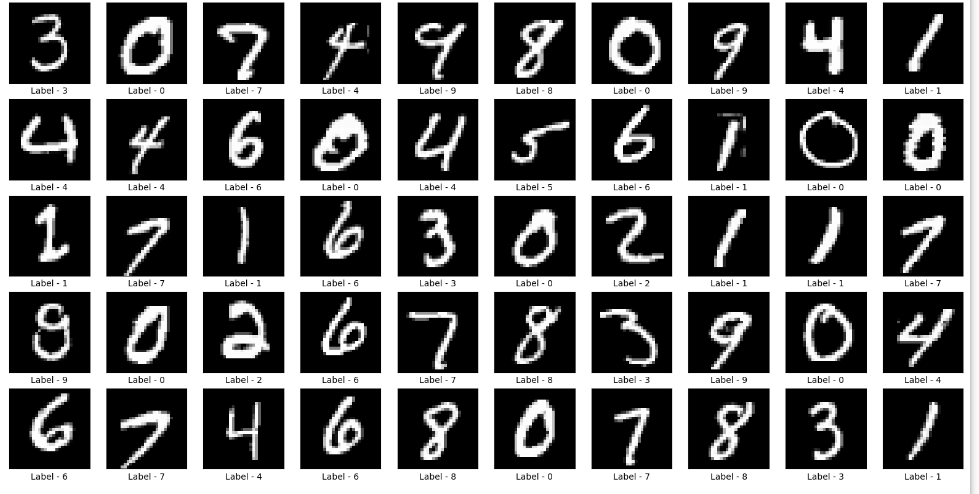
  plt.yticks([])

  plt.xlabel("Label - "+str(ytrain[i]),fontsize=15)

  plt.imshow(xtrain[i],cmap="gray")

plt.show()





**Figure 18 OUTPUT ‘100 Samples’**

**6.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**](https://machinelearningmastery.com/what-is-deep-learning/)[**larning**](https://machinelearningmastery.com/what-is-deep-learning/)models**.**

It wraps the efficient numerical computation libraries [**Theano**](https://machinelearningmastery.com/introduction-python-deep-learning-library-theano/)and [**TensorFlow**](https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/)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**](https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/)**.** 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.

**Machine Learning:**

Machine learning is a method of data analysis that automates analytical model building. It is a branch of [artificial intelligence](https://www.sas.com/en_in/insights/analytics/what-is-artificial-intelligence.html) 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)](https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp) 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](https://www.investopedia.com/terms/m/machine-learning.asp) [learning](https://www.investopedia.com/terms/m/machine-learning.asp) 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.

**Google Collab:**

Google Colab, short for Google Collaboratory, is a cloud-based platform for running Jupyter notebook-based machine learning and data analysis workflows. It is a free-to-use platform that provides users with a Jupyter notebook environment and free access to a Tesla K80 GPU for up to 12 hours at a time. This makes it a popular choice for students, researchers, and data scientists who want to experiment with deep learning and other computationally intensive tasks without having to invest in expensive hardware. With Google Colab, users can write, run, and share Python code, as well as leverage built-in libraries for machine learning, data analysis, and visualization.

**7.CONCLUSION AND FUTURE WORK**

**7.1 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 ANN (Artificial Neural Networks), and CNN (Convolutional Neural Networks) architectures are used to achieve high performance on the digit string recognition problem.

**7.2 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.

**References links:**

* https://colab.research.google.com/drive/1NRBEQn0enrXsmenLc5aWDprrU-yXSB7F?usp=share\_link
* <https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network>
* <https://www.javatpoint.com/machine-learning-vs-deep-learning>