## PART A

(PART A : TO BE REFFERED BY STUDENTS)

## EXPERIMENT NO. 1

* 1. **AIM: -** Handwritten Digit Recognition System using Machine Learning in Python

## Prerequisite

* + - Different programming language (Python or Java), Understanding of Machine Learning Algorithms, Machine Learning Algorithms

## Outcome

After successful completion of this experiment students will be able to understand working of Convolutional Neural Networks (CNN) and apply this algorithm wherever required

## Theory

Convolutional Neural Networks (CNN) are complex feed forward neural networks. CNNs are used for image classification and recognition because of its high accuracy There are three types of layers in a convolutional neural network:

1. Convolutional layer
2. Pooling layer
3. Fully connected layer .

Each of these layers has different parameters that can be optimized and performs a different task on the input data.

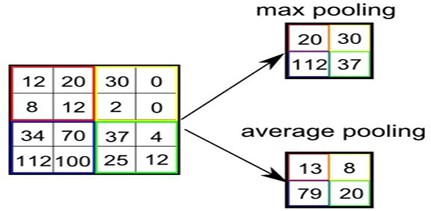
## What is Pooling Layer?

Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. There are two types of Pooling i. Average Pooling. ii. Max Pooling Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.

## What is Convolutional Layer?

Convolutional layers are the major building blocks used in convolutional neural networks. A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image. A convolutional layer contains a set of filters whose parameters need to be learned. The height and weight of the filters are smal ler

than those of the input volume. Each filter is convolved with the input volume to compute an activation map made of neurons.



## What is Fully Connected Layer?

A fully connected layer that takes the output of convolution/pooling and predicts the best lab the image.

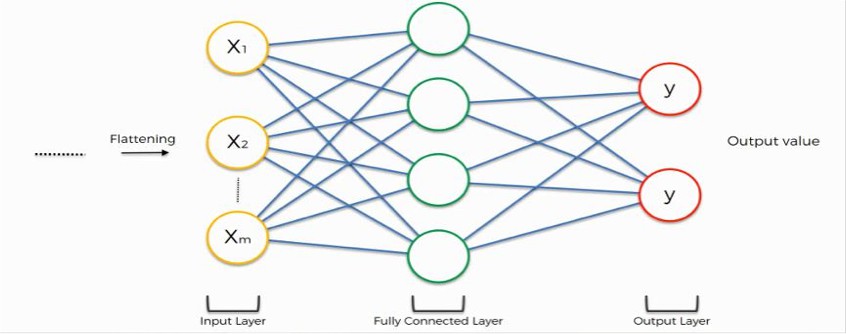
We have three layers in the full connection step

1. Input layer
2. Fullyconnected layer
3. Output layer Input Layer .

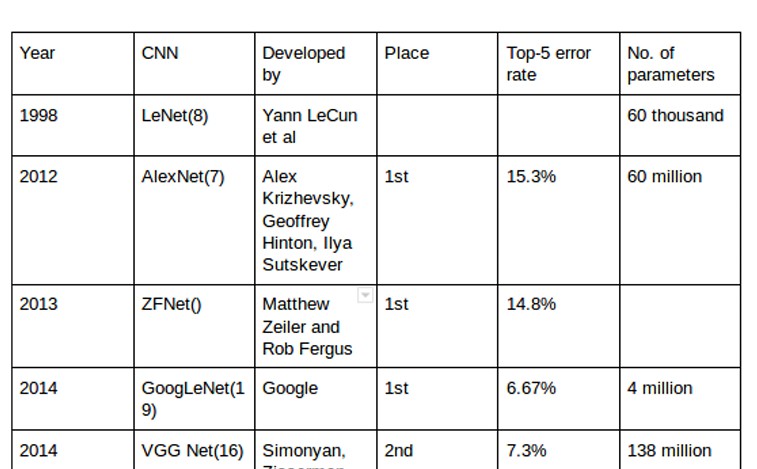
It takes the output of the previous layers, “flattens” them and turns them into a single vector that can be an input for the next stage.

Fully Connected Layer It takes the inputs from the feature analysis and applies weights to predict the correct label.

Output Layer It gives the final probabilities for each label.



## ReLU Layer

ReLU is an activation function. Rectified Linear Unit (ReLU) transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent var iable. The main aim is to remove all the negative values from the convolution. All the positive values remain the same but all the negative values get changed to zero

**Machine Learning with Tensor Flow**

Machine learning is a comple is an open sou x discipline but with help of Tensor force machine learning frame work. It gives us Flow it is quite easy to do. Tensor Flow ease the process of acquiring data, training models, serving predictions, and refining futu re results.

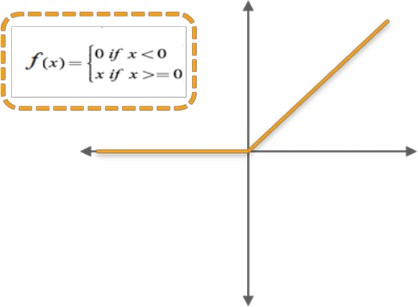
**Installation of Machine Learning Library**

1. Keras
2. Tensorflow

You can install these packages from command prompt by using below commands

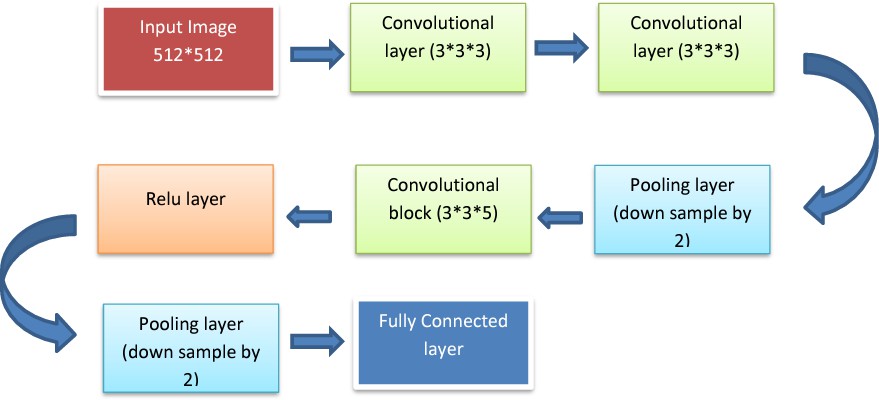
1. Python Python m pip install keras
2. pip install tensorflow

Hint: if Tensor Flow is not installed or give some error so first install virtual environment by this command python –m pip install –user virtualenv then install Tensor Flow. Lab



**CNN Architectures :** CNN architecture is formed by the s has its own number tack of distinct layers eg : conv layer pooling of layers and convolutional block. In the year 1994, and LENET5 convolutional neural networks [2] relu.. Each architecture is one of the very first with the passage of time different architectures are introduced with reduced error rate. Below is the comparison of different CNN architectures. Each architecture has its own no of layers and paramenters. [1]

Task: You are given a basic CNN architecture implement this architecture in python with the help of tensor flow library. In this architecture you have 3 convolutional layers,2 pooling layers and 1 relu layer and at the end there is fully connected layer.



**Dimensions:** Suppose we have input image of size 512\* 512 then after first convolution block in which we will apply 3 no of filters of size 3\*3 and with padding .We get 512\*512\*3 dimensions again after second layer of convolutional block we will get same dimension 512 \*512\*3 then after third layer we will able to get less no of values because this layer will be able to down sample input values by 2 so we will have 256\*256\*3 dimensions ,again convolutional block in which we apply 5 filters of size 3\*3 dimensions would be 256\*256\*5 then relu layer would have no effect on dimensions again pooling layer down samples the input values by 2 so dimensions would be 128\*128\*5 and when it comes to fully connected layer it flattens the 2D matrix into 1D finally we have 81,920\*1 no of input values.

## Some Useful Commands:

Several utilities/classes need to be imported first:

from keras.models import Sequential

from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D An instance of Sequential class is then initiated (our model):

my\_model = Sequential()

Now, the layers can be added as we please:

my\_model.add(Conv2D(64, kernel\_size = 3, activation = “relu”, input\_shape=(len,width,depth)))

my\_model.add(Conv2D (32, kernel\_size = 3, activation = “relu”))

my\_model.add(MaxPooling2D(pool\_size=(2,2))) my\_model.add(Flatten()) my\_model.add(Dense(3, activation = “softmax))

For compilation, training and making new predictions:

my\_model.compile(loss = “categorical\_crossentropy”, optimizer = optimizers.Adam(lr = 1e-4), metrics = [‘acc’])

my\_model.fit(training\_data, training\_labels, epochs = 10, batch\_size = 32) my\_predictions = my\_model.predict(test\_data)

**A5. Task**

**Given the MNIST data set your goal is to correctly identify digits from a dataset of tens of thousands of handwritten images. Perform Handwriting detection using**

1. **Computer vision fundamentals including simple neural networks.**
2. **Classification methods such as SVM and K-nearest neighbors**

**Link:** [**http://yann.lecun.com/exdb/mnist/**](http://yann.lecun.com/exdb/mnist/) **Or**

**Link:** [**https://www.kaggle.com/competitions/digit-recognizer**](http://www.kaggle.com/competitions/digit-recognizer)

## Note: Assume necessary Details. Use Exploratory Data Analysis and show details.

**You can use any technique for pre-processing if required.**

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

|  |  |
| --- | --- |
| Roll No. C027 | Name: Vishesh Giyanani |
| Class : Sem X | Batch : B1 |
| Date of Experiment: 12/12/2023 | Date of Submission |
| Grade : |  |

# Documentation written by student:

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

A graph of a number of digits

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screen shot of a computer program

Description automatically generated

A black and white image of a number

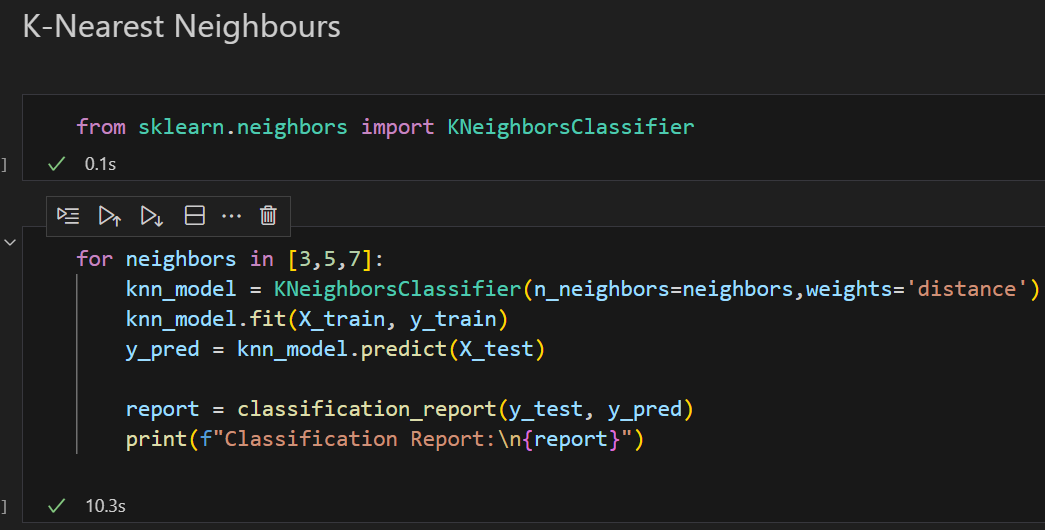
Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated



A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

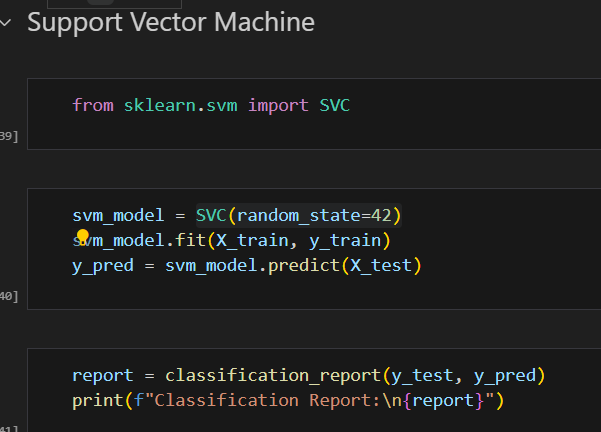
Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated



A screenshot of a computer screen

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

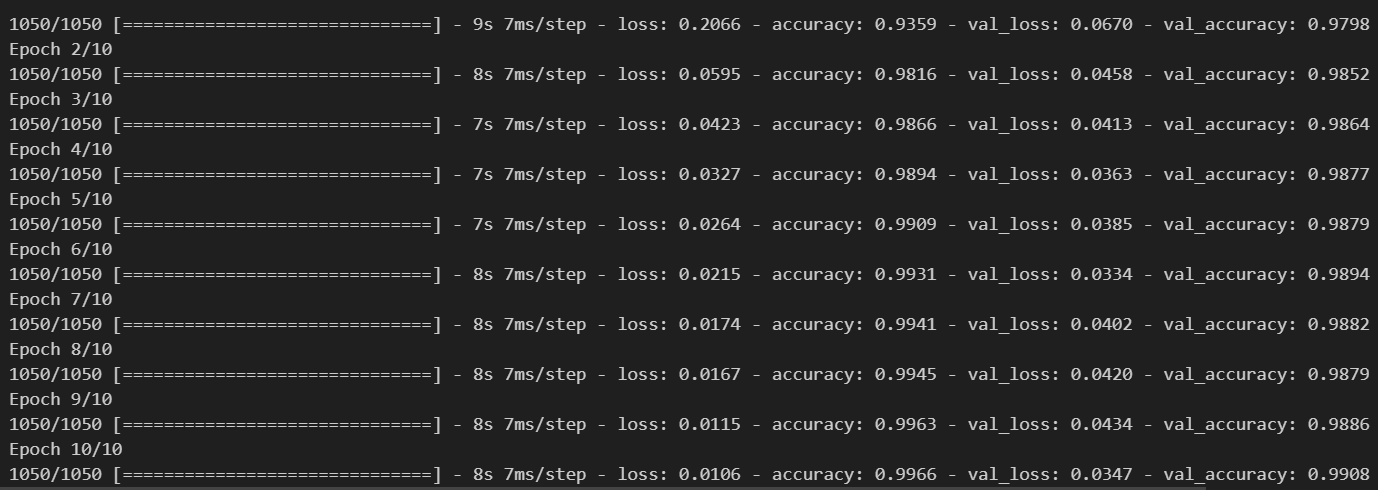
Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated



# Observations and learning:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy | Precision | Recall | F1-score |
| Logistic Regression 1 | 92.0% | Macro avg: 92.0%  Weighted avg: 92.0% | Macro avg: 92.0%  Weighted avg: 92.0% | Macro avg: 92.0%  Weighted avg: 92.0% |
| Logistic Regression 2 | 92.0% | Macro avg: 92.0%  Weighted avg: 92.0% | Macro avg: 92.0%  Weighted avg: 92.0% | Macro avg: 92.0%  Weighted avg: 92.0% |
| K- Nearest Neighbours 1 | 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% |
| K- Nearest Neighbours 2 | 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% |
| K- Nearest Neighbours 3 | 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% | Macro avg: 96.0%  Weighted avg: 96.0% | Macro avg: 96.0%  Weighted avg: 96.0% |
| Decision Tree 1 | 85.0% | Macro avg: 85.0%  Weighted avg: 85.0% | Macro avg: 85.0%  Weighted avg: 85.0% | Macro avg: 85.0%  Weighted avg: 85.0% |
| Decision Tree 2 | 86.0% | Macro avg: 86.0%  Weighted avg: 86.0% | Macro avg: 86.0%  Weighted avg: 86.0% | Macro avg: 86.0%  Weighted avg: 86.0% |
| Support Vector Machines 1 | 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% |
| Support Vector Machines 2 | 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% | Macro avg: 97.0%  Weighted avg: 97.0% |

**Logistic Regression**

**Logistic Regression 1**

**logistic\_model = LogisticRegression(max\_iter=1000, random\_state=42)**

* **Max iter:** In logistic regression for the digits dataset, setting max\_iter to 1000 was justified as it ensured sufficient iterations for the optimization algorithm to converge, resulting in a stable training process and achieving a satisfactory accuracy of 92% without triggering convergence warnings.This accuracy is acceptable as logistic regression is a simpler model
* **Random State**: Setting random\_state=42 ensures reproducibility, making the results consistent across multiple runs

**Logistic Regression 2**

**logistic\_model = LogisticRegression(**

**max\_iter=1000,**

**random\_state=42,**

**C=1.0,**

**multi\_class='multinomial',**

**penalty='l2')**

* **C:**A value of `C=1.0` was chosen to balance regularization and fitting the training data. It helps prevent overfitting by controlling the influence of each feature on the model, promoting a stable logistic regression with improved generalization to unseen data.
* **Multi-Class:** With `multi\_class='multinomial'`, logistic regression uses the multinomial logistic loss, suitable for multi-class classification tasks like digit recognition. This strategy ensures a cohesive modeling approach, enhancing the algorithm's ability to capture the complexities inherent in recognizing multiple digit classes.
* **Penalty:** Setting `penalty='l2'` utilizes L2 regularization, encouraging smaller weights for features and helping prevent overfitting. This regularization term is suitable for the digit recognition task on a dataset with high dimensionality (4200x784), promoting a more robust logistic regression model.

# K–Nearest Neighbours

# K–Nearest Neighbours 1

# KneighboursClassfier(n\_neighbours=3,weights=’distance’)

# K–Nearest Neighbours 2

# KneighboursClassfier(n\_neighbours=5,weights=’distance’)

# K–Nearest Neighbours 3

# KneighboursClassfier(n\_neighbours=7,weights=’distance’)

# 'p' and 'metric' (Default: p=2, metric='minkowski'): With `p=2` and `metric='minkowski'`, the KNN algorithm uses Euclidean distance as the default distance metric. For pixel recognition of digits, Euclidean distance is suitable as it considers the geometric distance between points in the high-dimensional space formed by pixel values. This helps capture the similarity between digit images based on their pixel values.

# algorithm' (Default: algorithm='auto'): Setting `algorithm='auto'` allows the KNN algorithm to automatically choose the most efficient algorithm (either 'ball\_tree', 'kd\_tree', or 'brute') based on the training dataset's size and dimensionality (784 features). This flexibility ensures optimal performance for pixel recognition on a dataset with varying characteristics.

* **number of neighbours(3,5,7)**: With a high-dimensional dataset like this (784 features representing pixels), a moderate number of neighbors, such as 3, 5, or 7, helps balance the influence of individual features and prevents overfitting.For digit recognition, neighboring digits in pixel space should have similar pixel patterns. A small number of neighbors ensures that the model considers local patterns while avoiding the noise that might be present with too many neighbors
* **weights='distance’:** it was observed that setting the weights parameter to distance rather than the default uniform gives better results by 1% for accuracy,f1-score,precision and recall for 5 and 7 neigbours. Assigning higher weights to closer neighbors allows the model to focus on the local pixel patterns, emphasizing the relevance of nearby points in capturing the structure of digit images.

# Decision Tree

# Decision Tree 1

# DecisionTree(random\_state=42)

# Decision trees may not perform well on this dataset due to its high dimensionality (784 features), treating pixels independently and struggling with spatial relationships. The model's tendency to overfit, limited expressiveness for complex pixel patterns, and sensitivity to small variations in pixel values contribute to lower accuracy compared to other models like KNN, SVM, and neural networks, which better capture spatial configurations and leverage feature relationships for digit recognition.

# Decision Tree 2

# DecisionTree(random\_state=42,max\_depth=20,min\_samples\_split=5,min\_samples\_leaf)

# max\_depth=20: limiting the maximum depth of the decision tree to 20 prevents the model from creating an overly complex tree that captures noise in the training data. It helps generalize better to unseen data by avoiding deep, intricate structures that may not be representative of digit patterns.

# min\_samples\_split=5: setting a minimum number of samples required to split a node to 5 ensures that the tree doesn't split too often, potentially capturing noise. This constraint promotes more robust and meaningful splits in the decision tree, contributing to improved generalization and accuracy.

# min\_samples\_leaf=2: specifying a minimum number of samples required to be in a leaf node (2 in this case) helps prevent the model from creating small leaf nodes that capture noise. It encourages the tree to form more generalized leaf nodes, making predictions based on a broader set of examples and enhancing overall model performance.

# Support Vector Machines

# Support Vector Machines 1

# SVC(random\_state=42)

# It was observed that SVM outperforms logistic regression and decision trees for this dataset. This could be due to its ability to find optimal hyperplane margins, especially in high-dimensional spaces like the digits\_data dataset. SVM is effective when the data has complex decision boundaries, allowing it to better capture the intricate patterns present in handwritten digits.

# Support Vector Machines 2

# SVC(random\_state=42,kernel=’rfb’,C=1.0,gamma=’scale’)

# kernel='rbf'`:the radial basis function (RBF) kernel is versatile and effective for capturing non-linear relationships in data. It allows SVM to model complex decision boundaries, potentially improving performance on datasets with intricate patterns.

# C=’1.0’: setting C to 1.0 balances the trade-off between achieving a smooth decision boundary and classifying training points correctly. It prevents overfitting and enhances generalization.

# gamma='scale': choosing 'scale' for gamma is suitable when dealing with datasets of varying scales. It adapts the influence of a single training point, contributing to better performance across different datasets.

# Convolutional Neural Networks

# Convolutional Neural Networks 1:

# model = keras.models.Sequential()

# model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)))

# model.add(keras.layers.MaxPooling2D((2, 2)))

# model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))

# model.add(keras.layers.MaxPooling2D((2, 2)))

# model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))

# model.add(keras.layers.Flatten())

# model.add(keras.layers.Dense(64, activation='relu'))

# model.add(keras.layers.Dense(10, activation='softmax'))

# Convolutional Neural Networks (CNNs) excel in digit recognition tasks due to their ability to automatically learn hierarchical features from pixel patterns. The inherent spatial awareness of CNNs, coupled with shared weight parameters, allows them to capture intricate relationships within digit images. This adaptability, combined with the ability to extract meaningful features through convolution and pooling layers, empowers CNNs to surpass traditional machine learning models like SVM, decision trees, logistic regression, and KNN in recognizing complex digit patterns, resulting in superior accuracy, as demonstrated by the 99.08% accuracy achieved.

# Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)): The first convolutional layer with 32 filters and a 3x3 kernel applies a ReLU activation, capturing local patterns in the 28x28 pixel images. The input shape corresponds to grayscale images (1 channel).

# MaxPooling2D((2, 2):Max-pooling with a 2x2 window reduces spatial dimensions, preserving essential features while enhancing computational efficiency. This helps the model focus on significant patterns.

# Conv2D(64, (3, 3), activation='relu'): The second convolutional layer with 64 filters and a 3x3 kernel further refines learned features, enabling the model to recognize more complex patterns in the digit images.

# MaxPooling2D((2, 2)): Another max-pooling layer downsamples the spatial dimensions, emphasizing the most salient features learned by the previous convolutional layers.

# Conv2D(64, (3, 3), activation='relu'):The third convolutional layer with 64 filters and a 3x3 kernel continues feature extraction, capturing higher-level abstractions in digit representations.

* **Flatten():** Flattening the output from convolutional layers transforms the spatial hierarchies into a one-dimensional vector, preparing the data for dense layers to capture global relationships.

# Dense(64, activation='relu'): A dense layer with 64 units and ReLU activation processes the flattened features, capturing global relationships and facilitating more complex digit representations.

# Dense(10, activation='softmax'): The final dense layer with 10 units (corresponding to digit classes) and softmax activation produces probabilities for each class. This layer enables the model to make predictions across multiple classes, facilitating digit recognition.

# Conclusion:

# In summary, various machine learning models were assessed on the digits\_data dataset for precise digit recognition based on pixel values. Logistic Regression, with carefully tuned hyperparameters, achieved a commendable 92% accuracy. K-Nearest Neighbours, utilizing Euclidean distance and 'distance' weighting, demonstrated robust performance with 97% accuracy. Decision Trees struggled with high dimensionality but showed improvement with controlled parameters, reaching 85-86% accuracy. Support Vector Machines excelled at 97%, emphasizing optimal hyperplane margins. Convolutional Neural Networks outperformed all, securing an impressive 99.08% accuracy by adeptly capturing intricate pixel patterns, highlighting the superiority of deep learning in digit recognition.

# Question of Curiosity

1. Write a Python program to find out when given an array of size N, the task is to partition the given array into two subsets such that the average of all the elements in both subsets is equal. If no such partition exists print -1. Otherwise, print the partitions. If multiple solutions exist, print the solution where the length of the first subset is minimum. If there is still a tie then print the partitions where the first subset is lexicographically smallest.

def find\_partition(arr):

    total\_sum = sum(arr)

    n = len(arr)

    left\_sum = 0

    left\_count = 0

    left\_partition = []

    for i, num in enumerate(arr):

        left\_sum += num

        left\_count += 1

        right\_sum = total\_sum - left\_sum

        right\_count = n - left\_count

        left\_avg = left\_sum / left\_count

        right\_avg = right\_sum / right\_count if right\_count != 0 else 0

        if left\_avg == right\_avg:

            return arr[:left\_count], arr[left\_count:]

    return -1

arr = [1, 2, 3, 4, 5, 6]

result = find\_partition(arr)

if result == -1:

    print("No such partition exists.")

else:

    subset1, subset2 = result

    print("Partition 1:", subset1)

    print("Partition 2:", subset