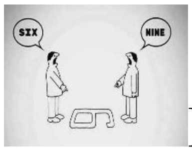


Digit Recognition System

------------------------------- Index --

INTRODUCTION

Artificial Intelligence

The birth of artificial intelligence conversations is marked by Alan Turing's pioneering work published in 1950. Turing, often referred to in the article as "the father of computer science," Turing asks, "Can machines think?" From now on, he gives a test in which a human interviewer tries to distinguish between computer and human textual answers. This test is now known as the "Turing test." Although this test has been the subject of much research since its announcement, it remains an important part of the history of artificial intelligence and remains an ongoing concept within its philosophy as it utilizes ideas from linguistics.

Human approach:

- Systems that think like humans
- Systems that act like humans

Ideal approach:

- Systems that think rationally
- Systems that act rationally

Alan Turing's definition would have fallen under the category of "systems that act like humans."

In its simplest form, artificial intelligence is the realm of combining computing and powerful data sets to solve problems. It also includes sub-domains of machine learning and deep learning that are often mentioned in the context of artificial intelligence. These disciplines consist of artificial intelligence algorithms that seek to create expert systems that make predictions or classifications based on input data.

Types of artificial intelligence

1. Weak AI

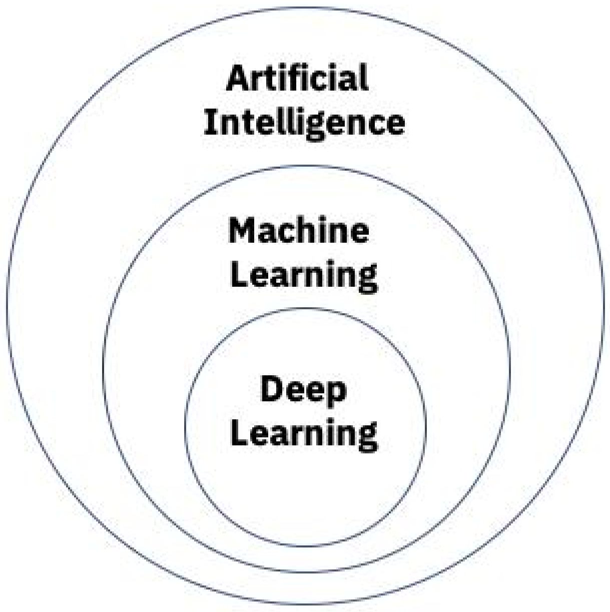
2. Strong AI

Weak AI

Weak AI, also known as Narrow AI or Artificial Narrow Intelligence (ANI), is an artificial intelligence that is trained and focused on a specific task. Weak AI moves much of the artificial intelligence that surrounds us today. Because this kind of artificial intelligence is weak, "narrow" may be a more accurate description. It enables some very powerful applications such as Apple Siri, Amazon Alexa, IBM Watson and standalone vehicles.

Strong AI

Strong AI is made up of Artificial General Intelligence (AGI) and Artificial Super Intelligence (ASI). Artificial general intelligence (AGI), or general AI, is a theoretical form of AI where a machine would have an intelligence equaled to humans; it would have a self-aware consciousness that has the ability to solve problems, learn, and plan for the future. While strong AI is still entirely theoretical with no practical examples in use today, that doesn't mean AI researchers aren't also exploring its development. In the meantime, the best examples of ASI might be from science fiction, such as HAL, the superhuman, rogue computer assistant in *2001: A Space Odyssey.*



Sub Fields of AI

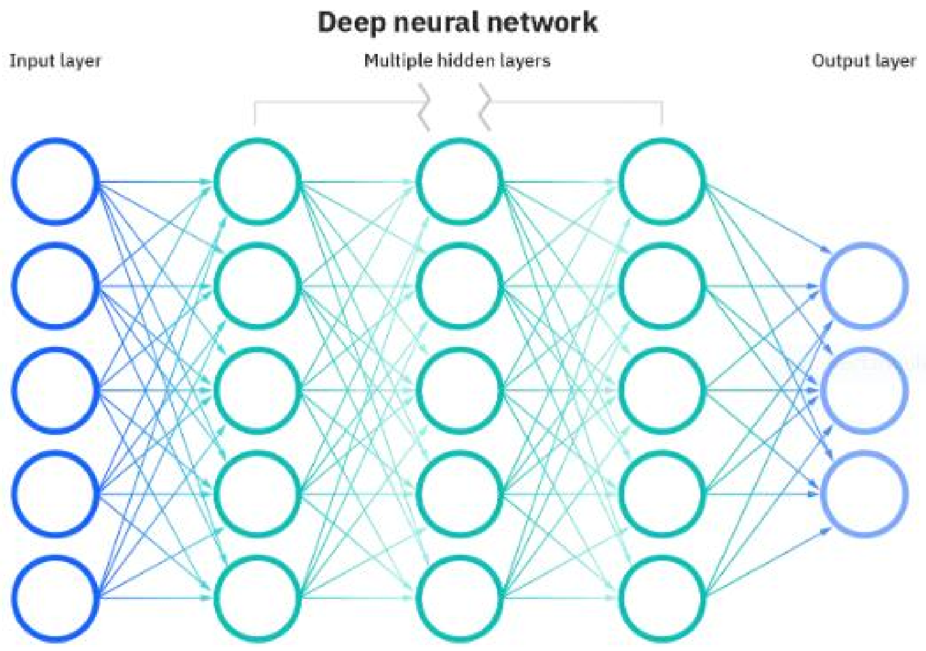
- Machine Learning
- Deep Learning

Machine Learning

Machine learning is an important component of the growing field of data science. Through the use of statistical methods, algorithms are trained to make classifications or predictions, uncovering key insights within data mining projects. These insights subsequently drive decision making within applications and businesses, ideally impacting key growth metrics. As big data continues to expand and grow, the market demand for data scientists will increase, requiring them to assist in the identification of the most relevant business questions and subsequently the data to answer them.

Deep Learning

Since deep learning and machine learning tend to be used interchangeably, it's worth noting the nuances between the two. As mentioned above, both deep learning and machine learning are sub-fields of artificial intelligence, and deep learning is actually a sub-field of machine learning. Deep learning is actually comprised of neural networks. "Deep" in deep learning refers to a neural network comprised of more than three layers—which would be inclusive of the inputs and the output—can be considered a deep learning algorithm. This is generally represented using the following diagram:



Machine Learning VS Deep Learning

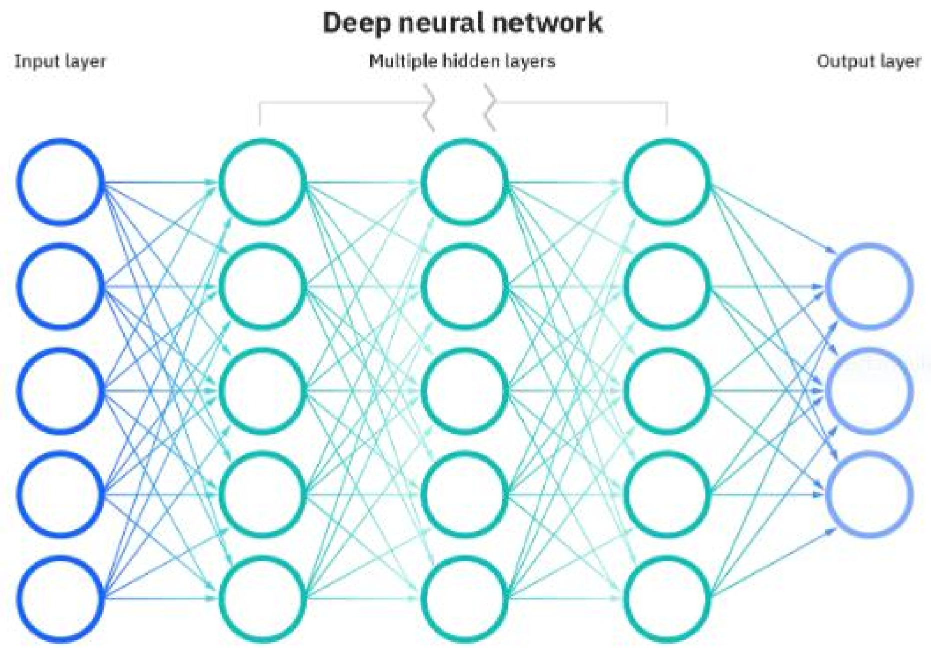
The way in which deep learning and machine learning differ is in how each algorithm learns. Deep learning automates much of the feature extraction piece of the process, eliminating some of the manual human intervention required and enabling the use of larger data sets. Classical, or "non-deep", machine learning is more dependent on human intervention to learn. Human experts determine the set of features to understand the differences between data inputs, usually requiring more structured data to learn.

Neural networks

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm.



LITRATURE REVIEW

Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN or ConvNet) is a special type of multi-layer neural network inspired by the mechanism of the optical system of living creatures. Hubel and Wiesel discovered that animal visual cortex cells detect light in the small receptive field. Motivated by this work, in 1980, Kunihiko Fukushima introduced neocognitron which is a multi-layered neural network capable of recognizing visual pattern hierarchically through learning. This network is considered as the theoretical inspiration for CNN. In 1990 LeCun et al. introduced the practical model of CNN and developed LeNet-5. Training by backpropagation algorithm helped LeNet-5 recognizing visual patterns from raw pixels directly without using any separate feature engineering mechanism. Also fewer connections and parameters of CNN than conventional feed forward neural networks with similar network size, made model training easier.

Convolutional neural networks (ConvNets or CNNs) are more often utilized for classification and computer vision tasks. Prior to CNNs, manual, time-consuming feature extraction methods were used to identify objects in images. However, convolutional neural networks now provide a more scalable approach to image classification and object recognition tasks, leveraging principles from linear algebra, specifically matrix multiplication, to identify patterns within an image. That said, they can be computationally demanding, requiring graphical processing units (GPUs) to train models.

Architecture of CNN:

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

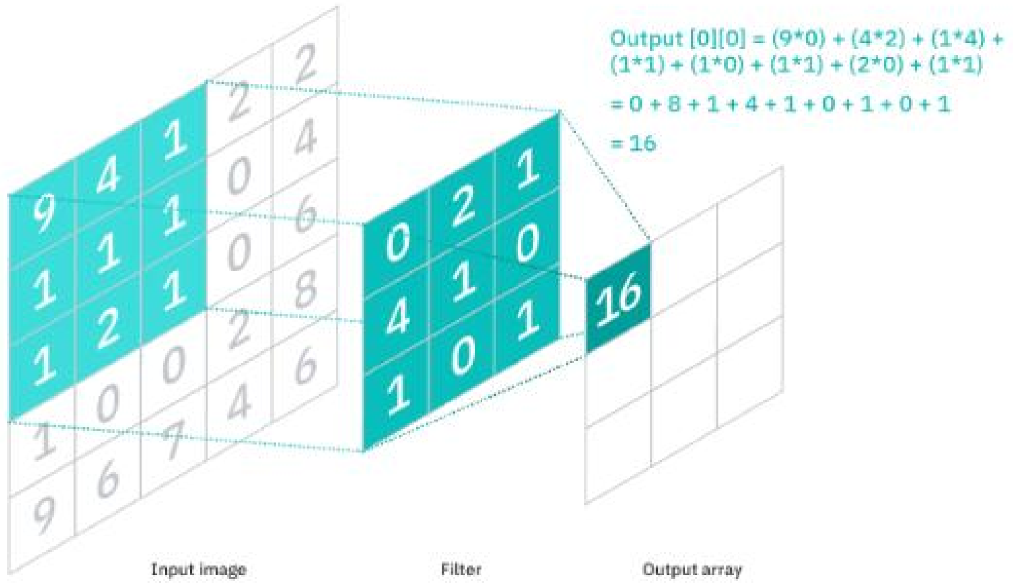
- Convolutional layer
- Pooling layer
- Fully-connected (FC) layer

The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

Convolutional Layer

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let’s assume that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature.


Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16
Input image
Filter
Output array

As you can see in the image above, each output value in the feature map does not have to connect to each pixel value in the input image. It only needs to connect to the receptive field, where the filter is being applied. Since the output array does not need to map directly to each input value, convolutional (and pooling) layers are commonly referred to as “partially connected” layers. However, this characteristic can also be described as local connectivity.

Note that the weights in the feature detector remain fixed as it moves across the image, which is also known as parameter sharing. Some parameters, like the weight values, adjust during training through the process of backpropagation and gradient descent. However, there are three hyper parameters which affect the volume size of the output that need to be set before the training of the neural network begins. These include:

1. N**umber of filters**

Number of filters affect the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.

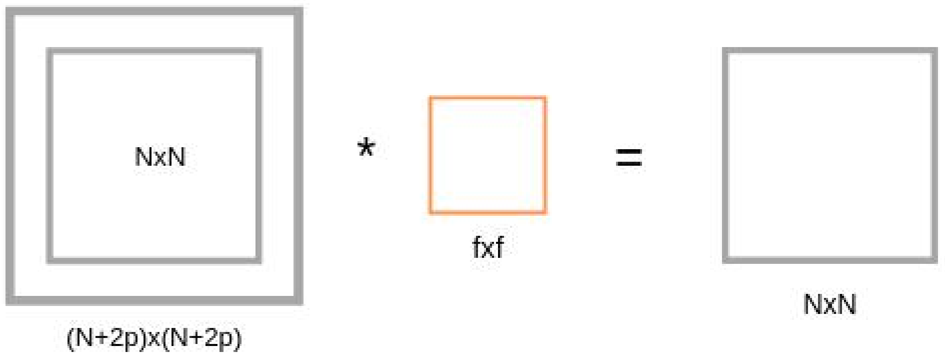
2. **Stride**

This is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.

3. **Zero-padding**

It is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, producing a larger or equally sized output. There are three types of padding:

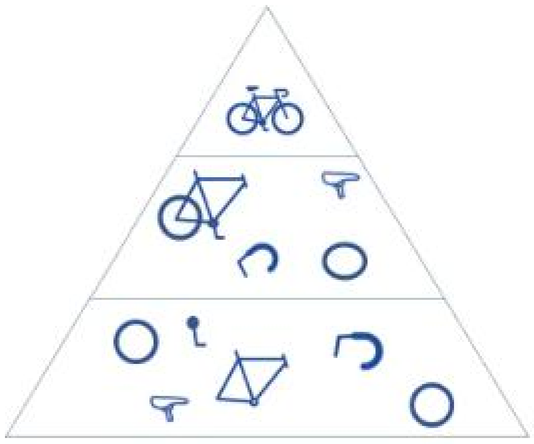
- **Valid padding:**
  - This is also known as no padding. In this case, the last convolution is dropped if dimensions do not align.
- **Same padding:**
  - This padding ensures that the output layer has the same size as the input layer
- **Full padding:**
  - This type of padding increases the size of the output by adding zeros to the border of the input.



In the above figure if we apply filter F x F in (N+2p) x (N+2p) input matrix with padding, then we will get output matrix dimension (N+2p-F+1) x (N+2p-F+1). As we know that after applying padding we will get the same dimension as original input dimension (N x N).

After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, introducing nonlinearity to the model.

As we mentioned earlier, another convolution layer can follow the initial convolution layer. When this happens, the structure of the CNN can become hierarchical as the later layers can see the pixels within the receptive fields of prior layers. As an example, let's assume that we're trying to determine if an image contains a bicycle. You can think of the bicycle as a sum of parts. It is comprised of a frame, handlebars, wheels, pedals, et cetera. Each individual part of the bicycle makes up a lower-level pattern in the neural net, and the combination of its parts represents a higher-level pattern, creating a feature hierarchy within the CNN.



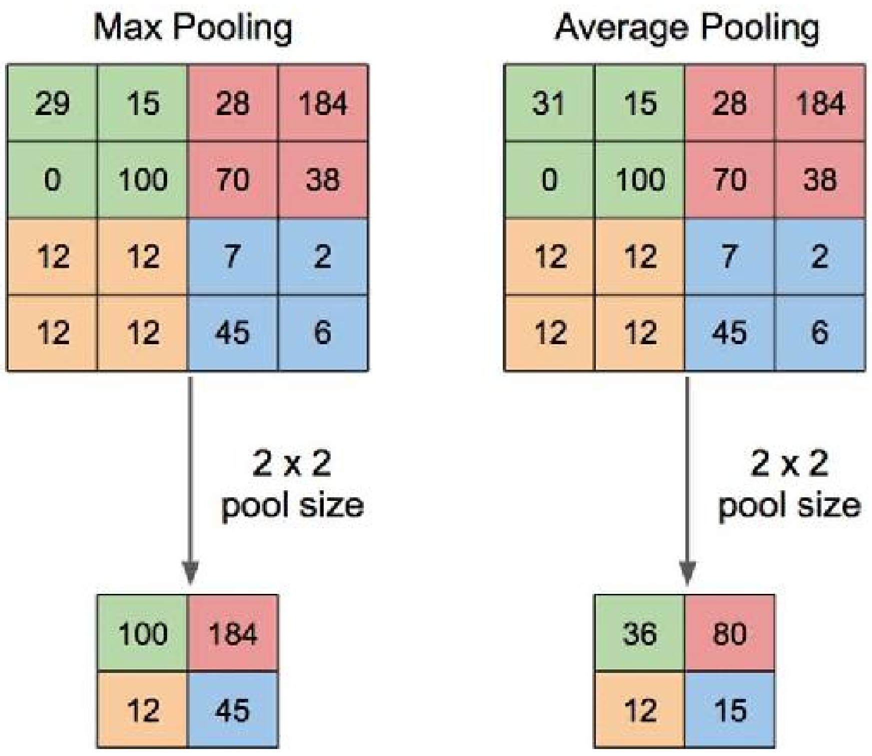
Ultimately, the convolutional layer converts the image into numerical values, allowing the neural network to interpret and extract relevant patterns.

Pooling Layer

Pooling layers, also known as downsampling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling:

- **Max pooling:** As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.
- **Average pooling:** As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

While a lot of information is lost in the pooling layer, it also has a number of benefits to the CNN. They help to reduce complexity, improve efficiency, and limit risk of overfitting.



Fully-Connected Layer

The name of the full-connected layer aptly describes itself. As mentioned earlier, the pixel values of the input image are not directly connected to the output layer in partially connected layers. However, in the fully-connected layer, each node in the output layer connects directly to a node in the previous layer.

This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

Convolutional neural networks and computer vision

Convolutional neural networks power image recognition and computer vision tasks. Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs, and based on those inputs, it can take action. This ability to provide recommendations distinguishes it from image recognition tasks. Some common applications of this computer vision today can be seen in:

- **Marketing:** Social media platforms provide suggestions on who might be in photograph that has been posted on a profile, making it easier to tag friends in photo albums.
- **Healthcare:** Computer vision has been incorporated into radiology technology, enabling doctors to better identify cancerous tumors in healthy anatomy.
- **Retail:** Visual search has been incorporated into some e-commerce platforms, allowing brands to recommend items that would complement an existing wardrobe.
- **Automotive**: While the age of driverless cars hasn't quite emerged, the underlying technology has started to make its way into automobiles, improving driver and passenger safety through features like lane line detection.

1. Decoding Facial Recognition

Facial recognition is broken down by a convolutional neural network into the following major components -

- ➢ Identifying every face in the picture
- ➢ Focusing on each face despite external factors, such as light, angle, pose, etc.
- ➢ Identifying unique features
- ➢ Comparing all the collected data with already existing data in the database to match a face with a name.

2. Analyzing Documents

Convolutional neural networks can also be used for document analysis. This is not just useful for handwriting analysis, but also has a major stake in recognizers. For a machine to be able to scan an individual's writing, and then compare that to the wide database it has, it must execute almost a million commands a minute. It is said with the use of CNNs and newer models and algorithms, the error rate has been brought down to a minimum of 0.4% at a character level, and though it's complete testing is yet to be widely seen.

3. Historic and Environmental Collections

CNNs are also used for more complex purposes such as natural history collections. These collections act as key players in documenting major parts of history such as biodiversity, evolution, habitat loss, biological invasion, and climate change.

4. Understanding Climate

CNNs can be used to play a major role in the fight against climate change, especially in understanding the reasons why we see such drastic changes and how we could experiment in curbing the effect. It is said that the data in such natural history collections can also provide greater social and scientific insights, but this would require skilled human resources such as researchers who can physically visit these types of repositories. There is a need for more manpower to carry out deeper experiments in this field.

5. Grey Areas

Introduction of the grey area into CNNs is posed to provide a much more realistic picture of the real world. Currently, CNNs largely function exactly like a machine, seeing a true and false value for every question. However, as humans, we understand that the real world plays out in a thousand shades of grey. Allowing the machine to understand and process fuzzier logic will help it understand the grey area us humans live in and strive to work against. This will help CNNs get a more holistic view of what human sees.

6. Advertising

CNNs have already brought in a world of difference to advertising with the introduction of programmatic buying and data-driven personalized advertising.

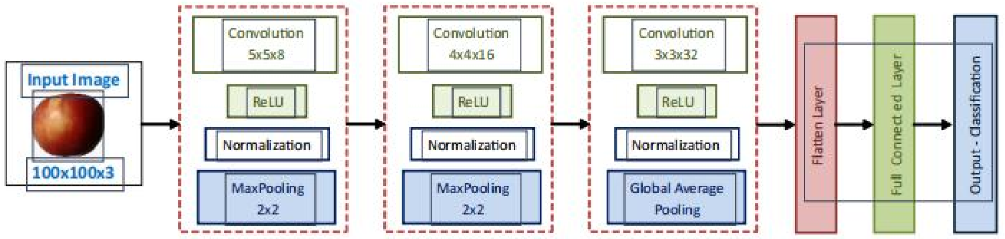
Different Solutions by using CNN

MLP-BP is simple and capable of resolving nonlinearly separable problems. Number of neurons is connected in layers to build a multilayer perceptron network where each of perceptron is used to identify small linearly separable sections of the inputs and is used to get the final output in the network. This technique is used for predict the single or multiple mobile terminals which are moving in the cellular network. This analytical prediction result is used for location update information in the cellular network. It will be good research work in the direction of Fuzzy and Neuro-Fuzzy which do not have any well-defined pattern.

For classification of Fruits, we used the dataset Fruit-360, which contains 82,213
Images (100 × 100 pixels) of fruits and vegetables of 120 classes, already subdivided
Into training and test sets. We selected six categories (three kind of apples and pears)

from the dataset-Apple Golden 1, Apple Pink Lady, Apple Red 1, Pear Red, Pear Williams and Pear Monster.

Once the categories have been selected to carry out the classification study, the next Step is to design the CNN architecture. Figure 3 depicts the CNN architecture adopted in this case. We designed a CNN with quite simple architecture comprised of the input layer, three convolutional layers, a flattening layer, one fully connected layer, and the output layer.



We train our model by 10 epochs and using the Stochastic Gradient Descent (SGD) optimizer with momentum. Firstly, all filters are randomly initialized from a normal distribution and all biases are equal zero.

We perform 10-fold cross-validation on the database aiming to obtain unbiased results in our experiments. Besides, we separate 10% of the training data for validation purposes. Thus, our training set was comprised of 2779 images and 307 images in the validation set. Figure 4 shows the average loss and accuracy curves during the model training process. The *x*-axis represents the training epoch number. It should be noted that from the 5th-epoch, the loss value is close to 0, and the accuracy result is close to 1 for both training and validation. These training and validation results seem to be perfect due to the low variability of the used dataset. After the training process, we evaluate our model on 1034 testing images. In Figure 4 we show the activation maps obtained by the 3rd convolution layer for an Apple Pink Lady and a Pear Williams. It is noticeable the differences between both fruits based on their activation maps, which are later used by the fully connected layers to make the class predictions. Figure 4 depicts the confusion matrix for classification results on the testing image set, where it can be noted the misclassifications between Pear Red, Apple Red 1, and Apple Pink Lady classes. The final average accuracy was 95.45% with an average F1-score of 0.96.

RESULT AND METHODOLOGY

Architecture Of Network:

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

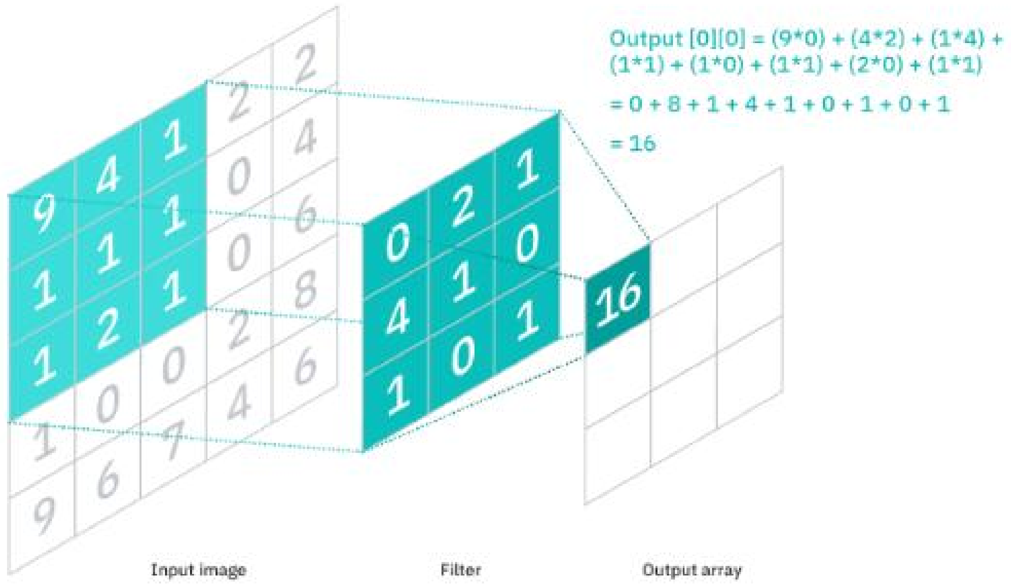
- Convolutional layer
- Pooling layer
- Fully-connected (FC) layer

The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

Convolutional Layer

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let's assume that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature.



As you can see in the image above, each output value in the feature map does not have to connect to each pixel value in the input image. It only needs to connect to the receptive field, where the filter is being applied. Since the output array does not need to map directly to each input value, convolutional (and pooling) layers are commonly referred to as "partially connected" layers. However, this characteristic can also be described as local connectivity.

Note that the weights in the feature detector remain fixed as it moves across the image, which is also known as parameter sharing. Some parameters, like the weight values, adjust during training through the process of backpropagation and gradient descent. However, there are three hyper parameters which affect the volume size of the output that need to be set before the training of the neural network begins. These include:

1. N**umber of filters**

Number of filters affect the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.

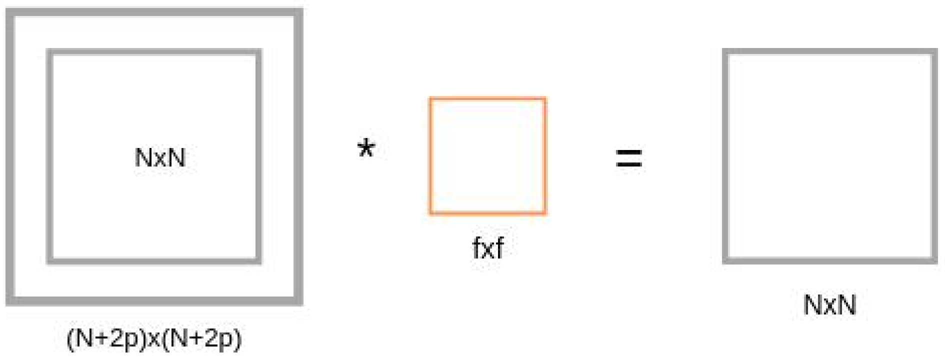
2. **Stride**

This is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.

3. **Zero-padding**

It is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, producing a larger or equally sized output. There are three types of padding:

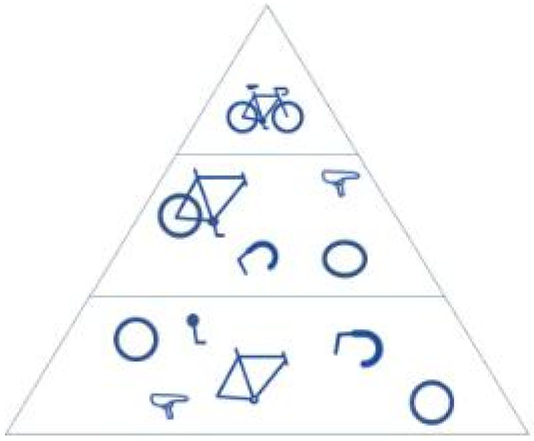
- **Valid padding:**
  - This is also known as no padding. In this case, the last convolution is dropped if dimensions do not align.
- **Same padding:**
  - This padding ensures that the output layer has the same size as the input layer
- **Full padding:**
  - This type of padding increases the size of the output by adding zeros to the border of the input.



In the above figure if we apply filter F x F in (N+2p) x (N+2p) input matrix with padding, then we will get output matrix dimension (N+2p-F+1) x (N+2p-F+1). As we know that after applying padding we will get the same dimension as original input dimension (N x N).

After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, introducing nonlinearity to the model.

As we mentioned earlier, another convolution layer can follow the initial convolution layer. When this happens, the structure of the CNN can become hierarchical as the later layers can see the pixels within the receptive fields of prior layers. As an example, let's assume that we're trying to determine if an image contains a bicycle. You can think of the bicycle as a sum of parts. It is comprised of a frame, handlebars, wheels, pedals, et cetera. Each individual part of the bicycle makes up a lower-level pattern in the neural net, and the combination of its parts represents a higher-level pattern, creating a feature hierarchy within the CNN.



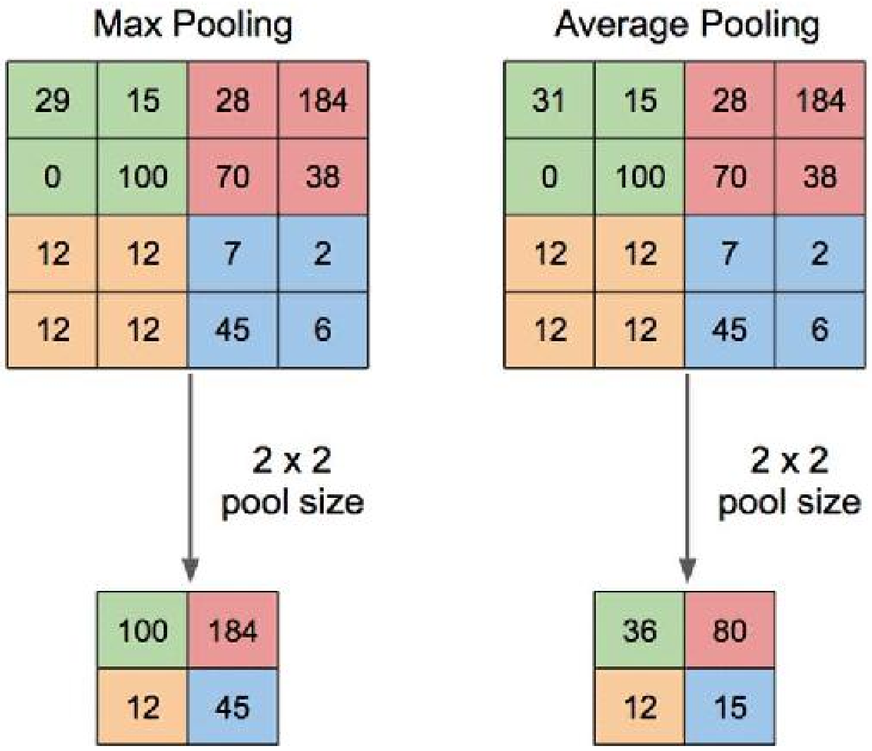
Ultimately, the convolutional layer converts the image into numerical values, allowing the neural network to interpret and extract relevant patterns.

Pooling Layer

Pooling layers, also known as downsampling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling:

- **Max pooling:** As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.
- **Average pooling:** As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

While a lot of information is lost in the pooling layer, it also has a number of benefits to the CNN. They help to reduce complexity, improve efficiency, and limit risk of overfitting.



Fully-Connected Layer

The name of the full-connected layer aptly describes itself. As mentioned earlier, the pixel values of the input image are not directly connected to the output layer in partially connected layers. However, in the fully-connected layer, each node in the output layer connects directly to a node in the previous layer.

This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

Mathematical Representation:

x **w1** **y1** **Applying Relu** **y2** **y3**

[28x28] convolute [9x9x256] → [256x256x256] →[256x256x256] → pooling [128x 256x256] → **Filter** y4 **Reshape w5** **y4** **v5** **Relu** **y5**

[25600x1] → [1280x25600] → [25600x1] →[1280x1] → [1280x1] →

w0 **y5**

[128x1280] x [1280x1]

v **Softmax** **y**

→[128x1] → [128x1]

Find Error (e): e= [design output] d - y [Actual output]

e= [128x1]-[128x1]

Apply Delta Rule

e5=delta transpose [w0]

[1280x1]= [128x1280]*[128x1280]

detla5= (y5>0) *e5;

e3=reshape (e4, y3);

e4=transpose [w5]*delta5;

w3

e2= e3kron [1 1] * [256x256x256]

[1 1]

delta2 delta-x

delta2= (y2>0)*e2 x [28 28] convolution [256x256x256]→[9x9x256]

dw1=dw1+delta-x;

[1280x25600]= [9x9x256] + [9x9x256]

dw5= dw5+delta5*transpose [y4]

[1280x25600]+ [1280x1]*[25600x1]

dw0=dw0+delta*transpose [y5]

[128x1280]= [128x1280] + [128x1]*[1280x1]

dw1=dw1/bsize

[1280x1]

dw5=dw5/bsize

[1280x1]

dw0=dw0/bsize

[1280x1]

<u>Description</u>

In the start of the expression, we take an image of 28x28 size. Then we declare three variable to store the values that are going to be generate after the program execution those are w0, w1 and w5. Then we convolute the image to store value in w1. Then we apply the Relu Function on the matrix of y2 of size [nxnxn].

Then we start pooling of the matrix y3 by making the first input layer half to make the training more efficient. We uses amnist data set for training of our input. Then we increase the number of filters from 20 to 256. After that we reshape the w5 values. Then we apply the Relu function on y5. We apply max-pooling on y and then we start finding the error (e) that is design output (d) – actual output(y). Then we apply the delta rule on each of the variable which we have created dynamically

Before this we are using 20 filters of our program to identify the image in the given system. Only grey scale image are allowed to check. We generate random numbers and store them in the dynamic variables we defined at the beginning of our **Amnist.test** file. When the program runs in matlab and generate the random values we store those values in excel files and recall them to run our program for fixed values that we have store in our runtime created variables. The accuracy of the program on random values is 94% and on fixed values it tends up to 95.7%

Now we increase our number of filters up to 256 the same is the procedure of the limitations. We adjust the code and make changes to the dimension values because after increasing the number of filters there is dimension mismatch error arise due to less order size of the matrix.

As we have increase the number of filters, the output creates random variables at the accuracy of the output to 93% and then we save the randomly generated values in the dynamic variables. After executing the program on fixed values the accuracy increase to