A

MINI PROJECT REPORT

on

Static Gesture Recognition(Numbers)

Submitted in partial fulfilment for the completion of

BE-IV Semester

In

INFORMATION TECHNOLOGY

By

Dixitha Kasturi(160116737003)

Under the guidance of

Ms. B.Swathi Sowmya
Assistant Professor,
Dept. of IT,CBIT.

DEPARTMENT OF INFORMATION TECHNOLOGY
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)
(Affiliated to Osmania University; Accredited by NBA(AICTE) and NAAC(UGC), ISO Certified 9001:2015)
GANDIPET,HYDERABAD – 500 075
Website: www.cbit.ac.in

2018-2019

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the project work titled "**Static Gesture Recognition(numbers)**" is submitted to **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY,** in partial fulfillment of the requirements for the award of the completion of 6th semester of B.E. in Information Technology, during the academic year 2018-2019, is a record of original work done by **DIXITHA KASTURI (160116737003)** during the period of study in Dept. of IT, CBIT, HYDERABAD, under our supervision and guidance.

Project Guide
Ms .B.Swathi Sowmya
Asst. Professor, Dept. of IT,
CBIT, Hyderabad.

Head of the Department
Dr.Suresh Pabboju
Professor, Dept. of IT,
CBIT, Hyderabad.

DECLARATION

This is to certify that the work reported in the present report titled "**Static Gesture Recognition(numbers)**" is a record of work done by us in the Department of Information Technology, Chaitanya Bharathi Institute of Technology, Hyderabad.

No part of the report is copied from books / journals / internet and wherever the portion is taken, the same has been duly referred. The reported results are based on the project work done entirely by us and not copied from any other source.

Dixitha Kasturi (160116737003)

ACKNOWLEDGEMENT

We take this opportunity to remember and acknowledge the cooperation, good will and support both moral and technical extended by several individuals out of which this project evolved. We shall always cherish my associate on with them.

We have immense pleasure in expressing my thanks and deep sense of gratitude to my project guide **Mrs. B. SwathiSowmya,** Assistant Professor, for the guidance and help throughout the development of this project work by providing us with required information.

We express our profound gratitude to **Dr. Suresh Pabboju, Head of Department, Department of Information Technology** for his support and encouragement in completing our project. We would like to thank for his encouragement and valuable guidance in bringing to this dissertation.

We're also thankful to **Dr.P.Ravinder Reddy, Principal** of our **CBIT**, for his continuous help and support during the project development.

A lot thanks to other faculty members of the department who gave their valuable suggestions at different stages of our project.

We are very much thankful to my parents who helped me with utmost friendliness and warmth always. They kept our spirit flying high and persistently encouraged us to undertake and complete this project.

Dixitha Kasturi(160116737003)

ABSTRACT

Machine Learning is training computers to have humanly behavior . Gestures a non-verbal form of communication. The goal of gesture recognition is to create a system which can identify specific human gestures and use them to convey information or for device control. The basis of static hand gesture recognition is image classification. Dataset of Creative Senz3D hand acquisitions will be used for static gesture recognition. The dataset contains 10 different gestures(numbers) accounting to a total of 2000+ images. Based on the data set, images will be classified and the system will be trained such that for a given image the gesture should be identified. Static gesture recognition will be the basis for dynamic or live gesture recognition which will further be used for Sign Language translation, which is a yet to be developed.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1. OVERVIEW

Static hand gesture recognition for sign language digits, which identifies the digits represents by that particular gesture. From 0 to 9 gestures will be identified.

1.2 AIM OF THE PROJECT

The aim of the project is to make an image classifier that can differentiate/identify sign-language digits accurately.

1.3 ORGANIZATION REPORT

The organization of the report is as follows:

Chapter 1 deals with the Introduction of the project and gives the details about the project in an abstract view.

Chapter 2 deals with the insights of Machine learning, Deep learning, Neural networks ,convolutional neural networks.

Chapter 3 deals with the Software Requirements Specifications which is a specification of the project software and hardware requirements.

Chapter 4 Description of the model and functions

Chapter 5 deals with the Testing of the project and screenshots of the project

Chapter 6 explains the Conclusion and further scope of the project.

2. INFORMATION

2.1 What is Machine Learning?

The ability of human to differentiate ,classify ,learn and identify can be incorporated into a machine. A machine can be trained so as to behave like a human. It is a study of algorithms that can be used in training the machine to perform tasks without explicit instructions. Machine learning is a category of algorithm that allows software applications to become more accurate in predicting outcomes without being explicitly programmed ,building algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available.

It can be classified into supervised and unsupervised learning. Under supervised learning ,the system is trained using previous or available labeled data. On the other hand unsupervised learning is done without classified data ,studies how systems can infer a function to describe a hidden structure from unlabeled data.

2.2 Deep Learning and Neural Networks

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. Deep learning is a collection of algorithms used in machine learning, used to model high-level abstractions in data through the use of model architectures, which are composed of multiple nonlinear transformations.

Deep learning is a specific approach used for building and training neural networks, which are considered highly promising decision-making nodes. An algorithm is considered to be deep if the input data is passed through a series of nonlinearities or nonlinear transformations before it becomes output. In contrast, most modern machine learning algorithms are considered "shallow" because the input can only go only a few levels of subroutine calling.

Deep learning removes the manual identification of features in data and, instead, relies on whatever training process it has in order to discover the useful patterns in the input examples. This makes training the neural network easier and faster, and it can yield a better result that advances the field of artificial intelligence.

2.3 Convolutional Neural Networks

Convolutional neural networks are deep artificial neural networks that are used primarily to classify images (e.g. name what they see), cluster them by similarity (photo search), and perform object recognition within scenes. They are algorithms that can identify faces, individuals, street signs, tumors, platypuses and many other aspects of visual data. One of the main applications is in image recognition or classification. Convolutional neural networks ingest and process images as tensors, and tensors are matrices of numbers with additional dimensions. A tensor encompasses the dimensions beyond that 2-D plane. You can easily picture a three-dimensional tensor, with the array of numbers arranged in a cube. Here's a 2 x 3 x 2 tensor presented flatly (picture the bottom element of each 2-element array extending along the z-axis to intuitively grasp why it's called a 3-dimensional array):

$$\begin{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 3 \\ 5 \end{pmatrix} & \begin{pmatrix} 4 \\ 7 \end{pmatrix} \\ \begin{pmatrix} 3 \\ 4 \end{pmatrix} & \begin{pmatrix} 4 \\ 6 \end{pmatrix} & \begin{pmatrix} 5 \\ 8 \end{pmatrix} \end{pmatrix}$$

Fig.2.3.1 2D representation of image

In code, the tensor above would appear like this: [[[2,3],[3,5],[4,7]],[[3,4],[4,6],[5,8]]]. And here's a visual:

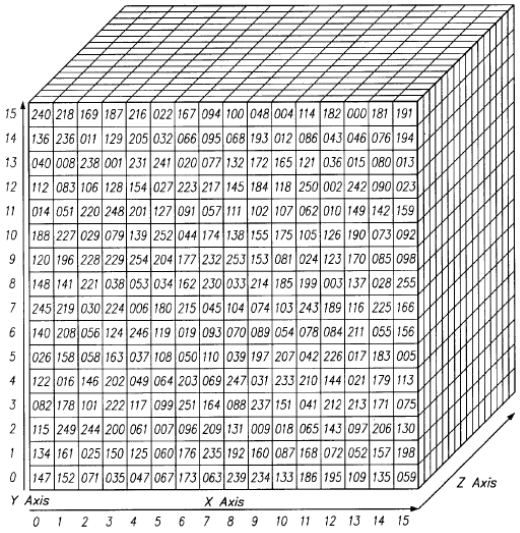


Fig.2.3.2 Complete Tensor 3D representation

In other words, tensors are formed by arrays nested within arrays, and that nesting can go on infinitely, accounting for an arbitrary number of dimensions far greater than what we can visualize spatially. A 4-D tensor would simply replace each of these scalars with an array nested one level deeper. Convolutional networks deal in 4-D tensors The width and height of an image are easily understood. The depth is necessary because of how colors are encoded. Red-Green-Blue (RGB) encoding, for example, produces an image three layers deep. Each layer is called a "channel", and through convolution it produces a stack of feature maps So instead of thinking of images as two-dimensional areas, in convolutional nets they are treated as four-dimensional volumes

From the Latin *convolvere*, "to convolve" means to roll together. For mathematical purposes, a convolution is the integral measuring how much two functions overlap as one passes over the other

The next thing to understand about convolutional nets is that they are passing *many* filters over a single image, each one picking up a different signal. At a fairly early layer, you could imagine them as passing a horizontal line filter, a vertical line filter, and a diagonal line filter to create a map of the edges in the image.

Convolutional networks take those filters, slices of the image's feature space, and map them one by one; that is, they create a map of each place that feature occurs. By learning different portions of a feature space, convolutional nets allow for easily scalable and robust feature engineering.

(Note that convolutional nets analyse images differently than RBMs. While RBMs learn to reconstruct and identify the features of each image as a whole, convolutional nets learn images in pieces that we call feature maps.)

So convolutional networks perform a sort of search. Picture a small magnifying glass sliding left to right across a larger image, and recommencing at the left once it reaches the end of one pass (like typewriters do). That moving window is capable recognizing only one thing, say, a short vertical line. Three dark pixels stacked atop one another. It moves that vertical-line-recognizing filter over the actual pixels of the image, looking for matches.

Each time a match is found, it is mapped onto a feature space particular to that visual element. In that space, the location of each vertical line match is recorded, a bit like birdwatchers leave pins in a map to mark where they last saw a great blue heron. A convolutional net runs many, many searches over a single image – horizontal lines, diagonal ones, as many as there are visual elements to be sought.

Convolutional nets perform more operations on input than just convolutions themselves.

After a convolutional layer, input is passed through a nonlinear transform such as *tanh* or *rectified linear* unit, which will squash input values into a range between -1 and 1.

2.4 Working Convolutional Neural Networks

The first thing to know about convolutional networks is that they don't perceive images like humans do. Therefore, you are going to have to think in a different way about what an image means as it is fed to and processed by a convolutional network.

Convolutional networks perceive images as volumes; i.e. three-dimensional objects, rather than flat canvases to be measured only by width and height. That's because digital colour images have a red-blue-green (RGB) encoding, mixing those three colours to produce the colour spectrum humans perceive. A convolutional network ingests such images as three separate strata of colour stacked one on top of the other.

So a convolutional network receives a normal colour image as a rectangular box whose width and height are measured by the number of pixels along those dimensions, and whose depth is three layers deep, one for each letter in RGB. Those depth layers are referred to as *channels*.

The next layer in a convolutional network has three names: max pooling, downsampling and subsampling. The activation maps are fed into a downsampling layer, and like convolutions, this method is applied one patch at a time. In this case, max pooling simply takes the largest value from one patch of an image, places it in a new matrix next to the max values from other patches, and discards the rest of the information contained in the activation maps.

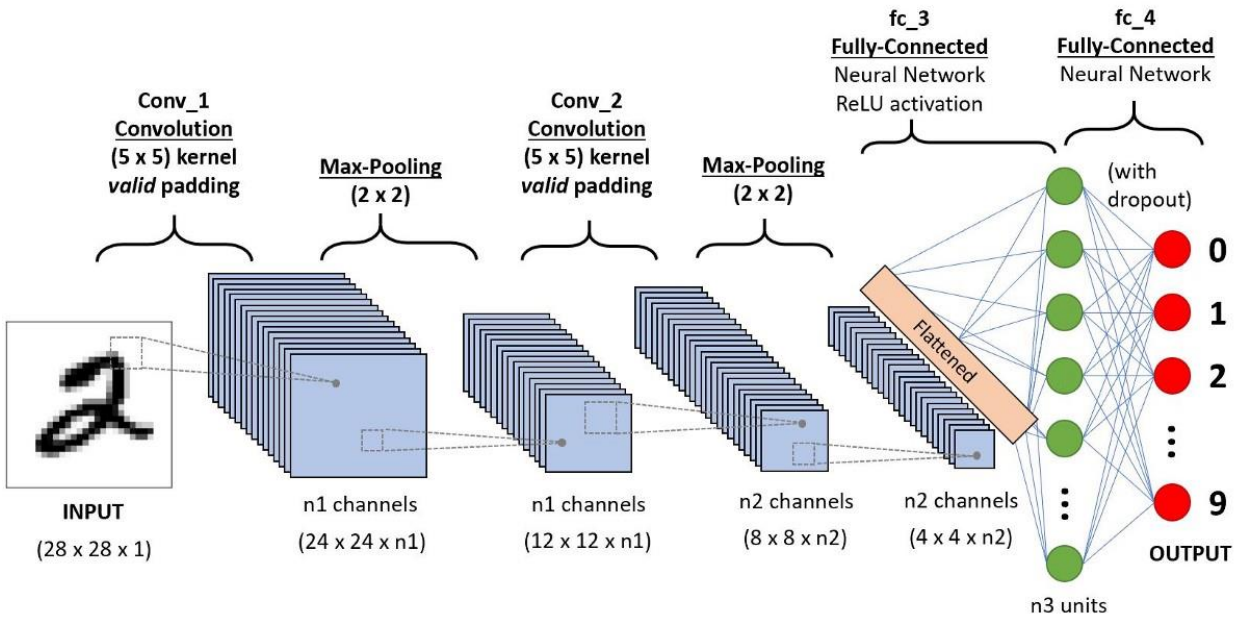


Fig.2.4.1 : Full representation of cnn layers

A **cnn** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

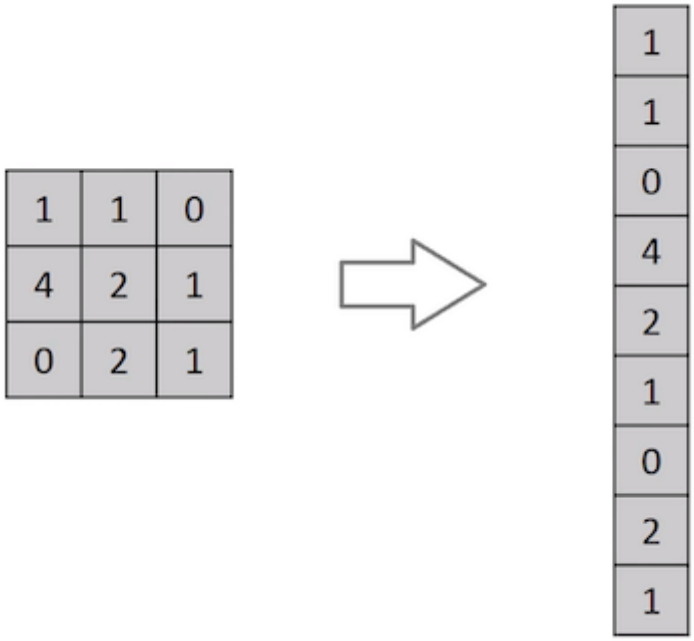


Fig.2.4.2 Flattening of a 3x3 image matrix into a 9x1 vector

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? Uh.. not really.

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.

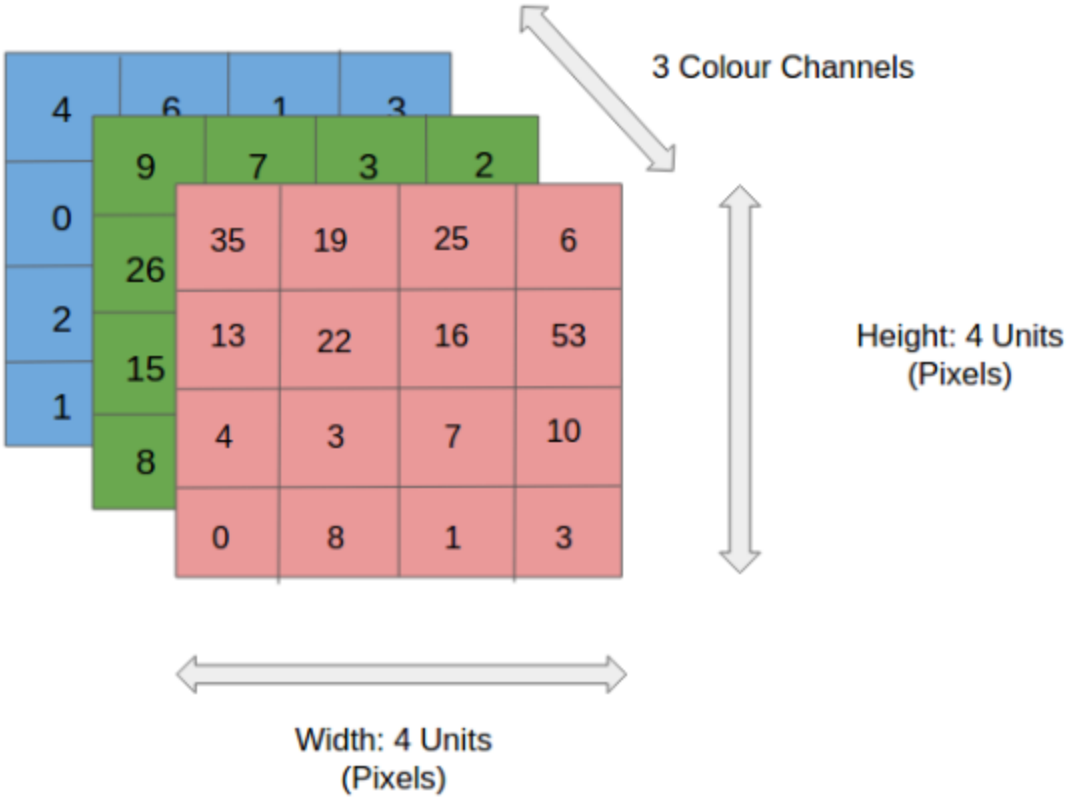


Fig.2.4.3 4x4x3 RGB Image

In the figure, we have an RGB image which has been separated by its three color planes—Red, Green, and Blue. There are a number of such color spaces in which images exist—Grayscale, RGB, HSV, CMYK, etc.

You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

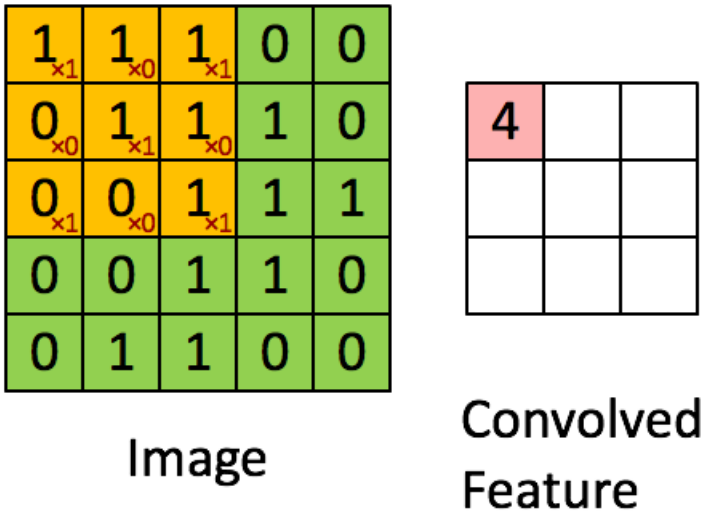


Fig.2.4.4 Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)

In the above demonstration, the green section resembles our 5x5x1 input image, I. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K**,** represented in the color yellow. We have selected K as a 3x3x1 matrix**.**The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided**)**, every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering.

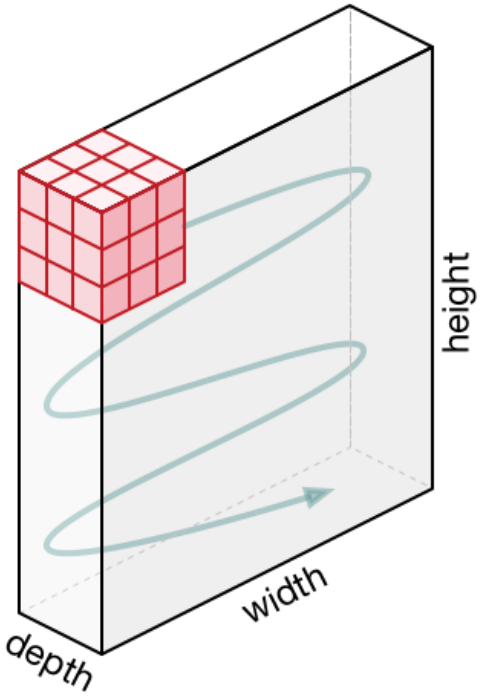


Fig.2.4.5 Movement of the Kernel

The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

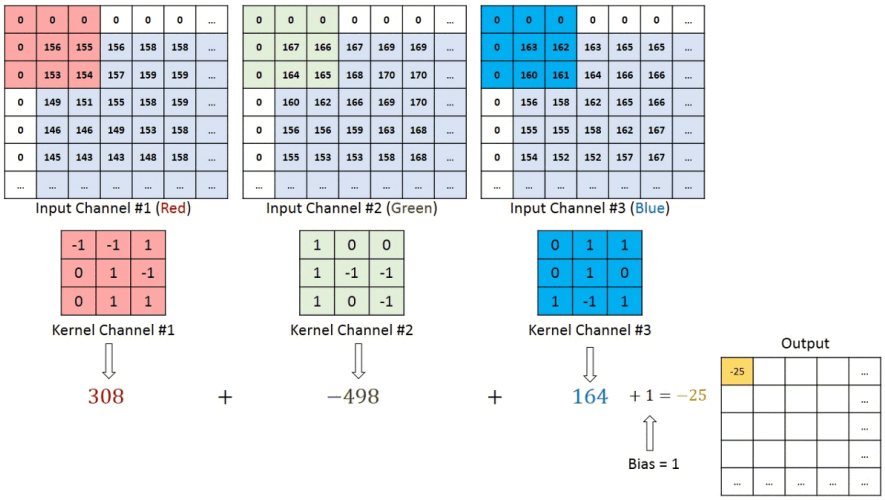


Fig.2.4.6 Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel

In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

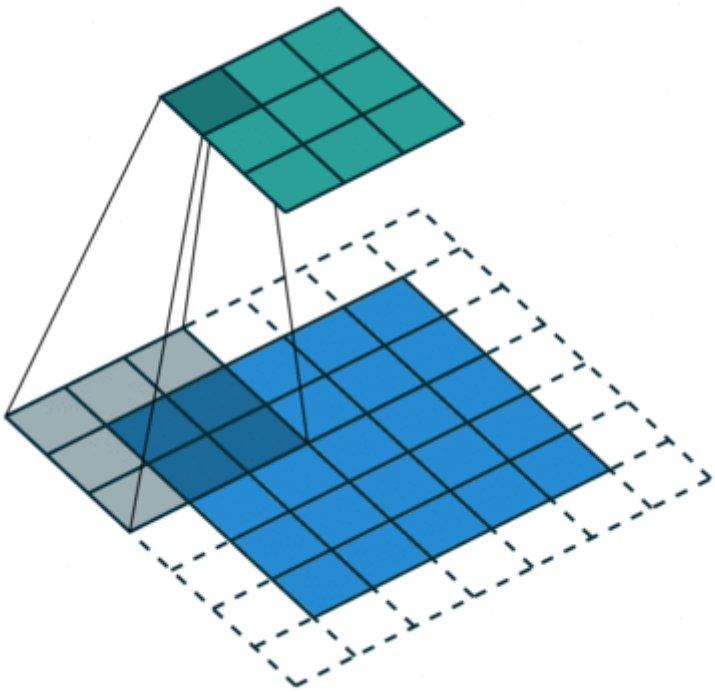


Fig.2.4.7 Convolution Operation with Stride Length = 2

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

There are two types of results to the operation—one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either

increased or remains the same. This is done by applying **Valid Padding** in case of the former, or **Same Padding** in the case of the latter.

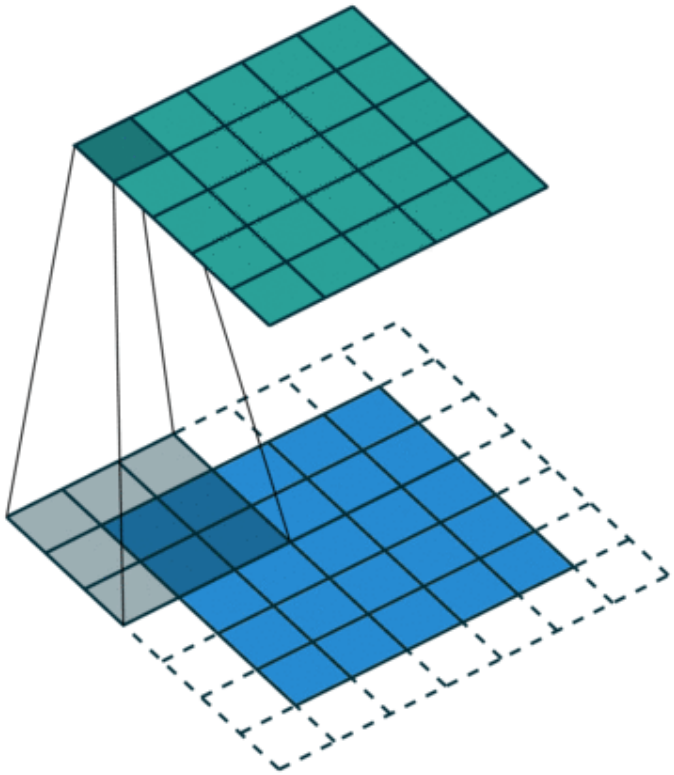


Fig.2.4.8 5x5x1 image is padded with 0s to create a 6x6x1 image

When we augment the 5x5x1 image into a 6x6x1 image and then apply the 3x3x1 kernel over it, we find that the convolved matrix turns out to be of dimensions 5x5x1. Hence the name—**Same Padding**.

On the other hand, if we perform the same operation without padding, we are presented with a matrix which has dimensions of the Kernel (3x3x1) itself—**Valid Padding**.

The following repository houses many such GIFs which would help you get a better understanding of how Padding and Stride Length work together to achieve results relevant to our needs.

3. SOFTWARE REQUIREMENT SPECIFICATION

3.1 INTRODUCTION

The requirements specification is a technical specification of requirements for the software products. It is the first step in the requirements analysis process it lists the requirements of a particular software system including functional, performance and security requirements. The requirements also provide usage scenarios from a user, an operational and an administrative perspective. The purpose of software requirements specification is to provide a detailed overview of the software project, its parameters and goals. This describes the project target audience and its user interface, hardware and software requirements. It defines how the client, team and audience see the project and its functionality.

3.2 SOFTWARE AND HARDWARE REQUIREMENTS

Operating System	Windows XP
Programming Languages	Python
Integrated Development Environment	GoogleColabs
Processor	Intel CORE i7,8th gen.
RAM	1 GB or more
Disk Space	1GB or more

4. IMPLEMENTATION

4.1 INTRODUCTION

The success of the software product is determined only when it is successfully implemented according to the requirements. The analysis and the design of the proposed system provide a perfect platform to implement the idea using the specified technology in the desired environment. The implementation of our system is made user friendly.Any software project is designed in modules and the project is said to be successfully implemented when each of the module is executed individually to obtain the expected result and, when all the modules are integrated and run together without any errors.

4.2 DATA

The dataset used in the project is Sign-Language-Digits dataset which contains 2000 images showing digits from 0 to 9 using hand gestures. This dataset is split into 33 % testing and 67% training. The images are in numpy array format i.e., their pixel values gradiated to grayscale images. The dimensions of the image are 64X64. With 10 labels assigned i.e., 0 to 9 respectively. User images are taken and modified according to the size used for testing purpose.

4.3 GOOGLE COLABS

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. With Colaboratory one can write and execute code, save and share the analyses, and access powerful computing resources, all for free from browser.

4.4 LIBRARIES

- Keras : An open-source neural-network library written in Python.
  - load_model : to reinstantiate and save model designed
  - Sequential : type of cnn created
  - Layers : Dense, Dropout, Flatten, Activation,Conv2D, MaxPooling2D
- Numpy : To deal with numpy array data

- PIL : To deal with images ,resizing and fitting of images
- Sklearn : model_selection,train_test_split to split dataset into train and test.

4.2 MODEL

The model chosen for this project is sequential model, which has 3 layers of cnn for training. With 20 epochs and the image sizes being sent as 64X64

1) 2D convolution layer (e.g. spatial convolution over images).This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not none, it is applied to the outputs as well.When using this layer as the first layer in a model, provide the keyword argument input_shape(tuple of integers, does not include the batch axis), **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

kernel_size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions. A 3X3 convolution window is used.

2) Max pooling operation for temporal data. Size of 2X2 is used in this model for first two layers.

pool_size: Integer, size of the max pooling windows.

3) Activation applies an activation function to an output. Activation use is "relu" rectifier networks

activation: name of activation function to use

4) Dropout applies Dropout to the input. Dropout applied in the model for first two layers in 0.25 , 0.4 and 0.3 for dense layer.

rate: float between 0 and 1. Fraction of the input units to drop.

5) Compile function is used to compile the model designed , Arguments are as follows

optimizer: String (name of optimizer) or optimizer instance. Adadelta optimizer is used as it is more efficient than others.

loss: String (name of objective function) or objective function. If the model has multiple outputs, you can use a different loss on each output by passing a dictionary or a list of losses. The loss value that will be minimized by the model will then be the sum of all individual losses categorical_crossentropy is used in this.

metrics: List of metrics to be evaluated by the model during training and testing.

6) Fit trains the model for a given number of epochs (iterations on a dataset).

x: Numpy array of training data (if the model has a single input), or list of Numpy arrays (if the model has multiple inputs). If input layers in the model are named, you can also pass a dictionary mapping input names to Numpy arrays. x can be none (default) if feeding from framework-native tensors

y: Numpy array of target (label) data (if the model has a single output), or list of Numpy arrays (if the model has multiple outputs). If output layers in the model are named, you can also pass a dictionary mapping output names to Numpy arrays. y can be none (default) if feeding from framework-native

batch_size: Integer or none. Number of samples per gradient update. If unspecified, batch_size will default to 32. 128 is used for this model.

epochs: Integer. Number of epochs to train the model. An epoch is an iteration over the entire x and y data provided. Note that in conjunction with initial_epoch, epochs is to be understood as "final epoch". The model is not trained for a number of iterations given by epochs , but merely until the epoch of index epochs is reached. 20 epochs accounting to an accuracy of 91% is used in this model.

verbose: Integer. 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch.

7) Evaluate function is used to find the accuracy of the designed model.

5. TESTING AND RESULTS

5.1 INTRODUCTION

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive.

A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned and conducted systematically. The underlying motivation of program testing is to affirm software quality with methods that can economically and effectively apply to both strategic to both large and small-scale systems.

5.2 TESTING OBJECTIVES

The main objective of performance testing is designed to test whether the app's display is as expected and whether the app is functioning properly or not.

As the test results are gathered and evaluated they begin to give a qualitative indication of the reliability of the app. If proper output is not obtained, the overall quality of the app is questioned. If, on the other hand, all the results which are not successful, are encountered, and are easily modifiable, then the following conclusion can be made: The tests are inadequate as the requirements mentioned are not compatible. The testing includes:

- Checking whether the information is displayed or not.
- Checking whether all the links between each button in the app works or is misdirected.
- Verifying if all the pictures are displayed and none of the files are corrupted.

5.3 SCREENS

```
batch_size = 128
epoch = 20
input_size = (64,64,1)
number_classes = 10

cnn = Sequential()
```

```
cnn.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_size))
cnn.add(MaxPooling2D((2, 2)))
cnn.add(Dropout(0.25))

cnn.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Dropout(0.25))

cnn.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
cnn.add(Dropout(0.4))

cnn.add(Flatten())

cnn.add(Dense(128, activation='relu'))
cnn.add(Dense(128, activation='relu'))
cnn.add(Dropout(0.3))
cnn.add(Dense(10, activation='softmax'))
```

```
cnn.compile(loss = keras.losses.categorical_crossentropy,
            optimizer = keras.optimizers.Adadelta(),
            metrics = ["accuracy"])
cnn.fit(x_train,y_train,
        batch_size = batch_size,
        epochs = epoch,
        verbose = 1,
        validation_data = (x_test,y_test))
```

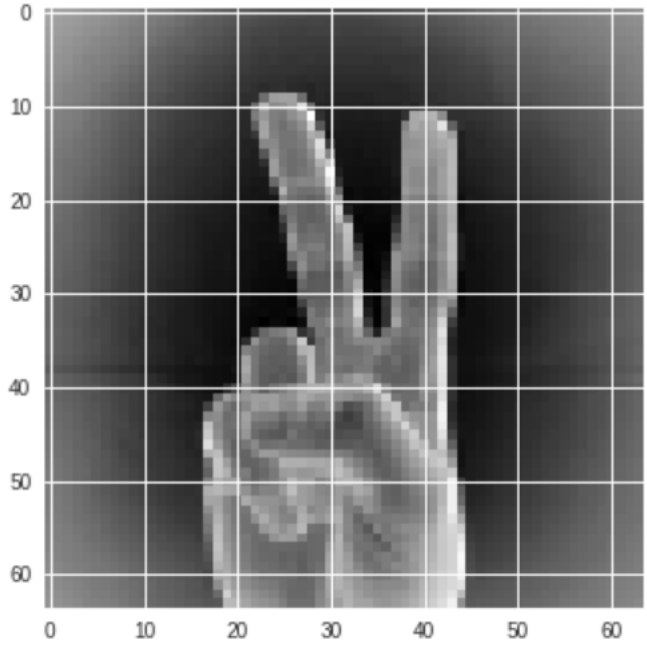
Fig.5.3.1 Layers and Model

To identify the test dataset and run the model :

```
classes = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

i = 101
plt.imshow(x_test[i].reshape(64,64), cmap = "gray")
y_test[i]
print(x_test[i].shape)
```

```
(64, 64, 1)
```




```
test1 = x_test[i].reshape(-1,64,64,1)
pre1 = cnn.predict(test1 ,batch_size = 1)
print("Test Cases--")
print("Prediction: ",np.round(pre1,0))
print("Number: ",classes[np.argmax(pre1)])
```

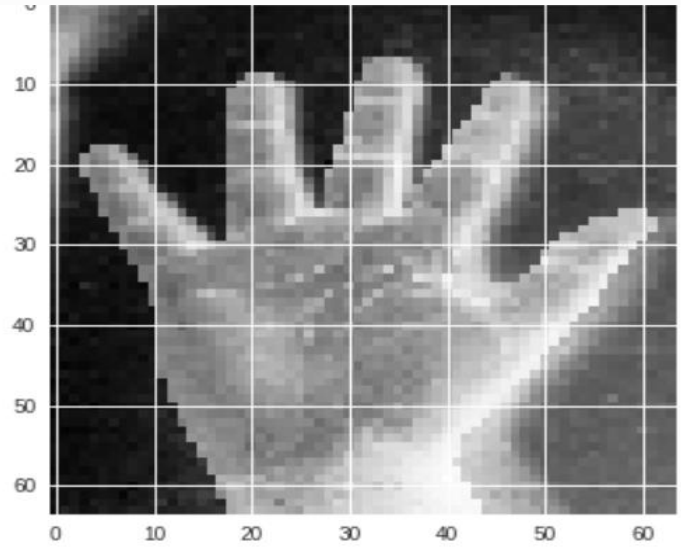
```
Test Cases--
Prediction:  [[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
Number:  2
```

Fig.5.3.2 Test set image and prediction

For user Input:

```
from PIL import Image
test="/content/drive/My Drive/testing/"
test_dir="/content/drive/My Drive/testing/hands.jpg"
def invert(pixel):
    return -pixel + 256
sample_img=Image.open(test_dir)
sample_img=sample_img.resize((64,64))
sample_img=sample_img.convert(mode='L')
plt.imshow(sample_img)
sample_img = sample_img.point(invert)
sample_img=np.array(sample_img)/255
sample_img=sample_img.reshape((1,64,64))
print(sample_img)
```

```
[[[0.55294118 0.55294118 0.56862745 ... 0.31372549 0.30980392 0.30196078]
  [0.54901961 0.54117647 0.56078431 ... 0.31764706 0.3254902  0.31372549]
  [0.55686275 0.56078431 0.57647059 ... 0.31764706 0.31764706 0.31372549]
  ...
  [0.29019608 0.29411765 0.30980392 ... 0.47843137 0.48235294 0.49411765]
  [0.29803922 0.29411765 0.30980392 ... 0.49411765 0.48235294 0.49019608]
  [0.28627451 0.34117647 0.32156863 ... 0.47058824 0.49803922 0.49803922]]]
```




```
test = sample_img.reshape(-1,64,64,1)
pre = cnn.predict(test, batch_size = 1)

print("User example--")
print("Prediction: ", np.round(pre, 0))
print("Number: ",classes[np.argmax(pre)])
```

```
User example--
Prediction:  [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
Number:  5
```

Fig.5.3.3 User image representation and prediction

6. CONCLUSION AND FUTURE SCOPE

The static method of classifying images can be used as the basis of video clustering. Dynamic hand gesture or sign language translation can be done by using video clustering coupled with object detection. The language translation can be done using text to speech convertor.

Bibliography
Online References:

1. Youtube : Various channels

2. https://keras.io/

3. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53