

DEEP ENSEMBLE MACHINE FOR VIDEO CLASSIFICATION

Submitted in partial fulfillment of the requirements
for the award of

Bachelor of Engineering degree in Computer Science and Engineering

by

KOSURI RECHITHA (Reg. No. 37110367)

JOGI HARIKA (Reg. No. 37110294)



**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC JEPPIAAR NAGAR, RAJIV

GANDHI SALAI, CHENNAI – 600 119

MARCH 2021



SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**



Accredited with “A” grade by NAAC
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work of **KOSURI RECHITHA (Reg. No. 37110367)** and **JOGI HARIKA (Reg. No. 37110294)** who carried out the project entitled “**DEEP ENSEMBLE MACHINE FOR VIDEO CLASSIFICATION**” under my supervision from **November 2020** to **March 2021**.

Internal Guide

Dr. N. Srinivasan, M.E., Ph.D.,

Head of the Department

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I **KOSURI RECHITHA** hereby declare that the Project Report entitled “**DEEP ENSEMBLE MACHINE FOR VIDEO CLASSIFICATION**” is done by me under the guidance of **Dr. N. Srinivasan, M.E., Ph.D.**, Department of Computer Science and Engineering at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean**, School of Computing, **Dr. S. Vigneswari, M.E., Ph.D., and Dr. L. Lakshmanan, M.E., Ph.D., Heads of the Department** of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. N. Srinivasan, M.E., Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Video classification has been extensively researched in computer vision due to its wide spread applications. However, it remains an outstanding task because of the great challenges in effective spatial-temporal feature extraction and efficient classification with high- dimensional video representations. To address these challenges, in this paper, we propose an end-to-end learning framework called deep ensemble machine (DEM) for video classification. Specifically, to establish effective spatial-temporal features, we propose using two deep convolutional neural networks(CNNs), i.e., vision and graphics group and C3-D to extract heterogeneous spatial and temporal features for complementary representations. To achieve efficient classification, we propose ensemble learning based on random projections aiming to transform high-dimensional features into a set of lower dimensional compact features in subspaces; an ensemble of classifiers is trained on the subspaces and combined with a weighting layer during the backpropagation. To further enhance the performance, we introduce rectified linear encoding (RLE) inspired from error-correcting output coding to encode the initial outputs of classifiers, followed by a soft max layer to produce the final classification results. DEM combines the strengths of deep CNNs and ensemble learning, which establishes a new end-to-end learning architecture for more accurate and efficient video classification. We show the great effectiveness of DEM by extensive experiments on four data sets for diverse video classification tasks including action recognition and dynamic scene classification. Results have shown that DEM achieves high performance on all tasks with an improvement of up to 13% on CIFAR10 data set over the baseline model...

TABLE OF CONTENTS

	ABSTRACT	v
	LIST OF FIGURES	viii
CHAPTER No.	TITLE	PAGE No.
1.	INTRODUCTION	1
1.1	AIM	1
1.2	SYNOPSIS	1
2.	SYSTEM ANALYSIS	2
2.1	Existing system	2
2.2	Proposed system	2
3.	REQUIREMENT SPECIFICATIONS	3
3.1	Introduction	3
3.2	Hardware and Software Specification	5
3.2.1	Hardware Requirements	5
3.2.2	Software Requirements	5
3.3	Technologies Used	5
4.	DESIGN AND IMPLEMENTATION CONSTRAINTS	21
4.5.1	Constraints to Analysis	21
4.5.2	Constraints in Design	21
4.5.3	Constraints in Implementation	21
4.6	NON-FUNCTIONAL REQUIREMENTS	22
4.6.1	Performance Requirements	22
4.6.2	Safety Requirements	22
5.	SEQUENCE DIAGRAM	23

6.	MODULES	28
6.1	Module Explanation	28
7.	CODING AND TESTING	29
7.1	Coding	29
7.2	Coding Standards	29
7.3	Test Procedure	31
7.4	Test Data and Output	32
	SOURCE CODE	39
	SNAP SHOTS	43
	REFERENCES	44

LIST OF FIGURES

5.1	Sequence Diagram	23
5.2	Use Case Diagram	25
5.3	Activity Diagram	26
5.4	Collaboration Diagram	27

CHAPTER 1

INTRODUCTION

Aim:

This paper aim to classify the different types of videos using deep learning framework with convolution neural network (CNN)

Synopsis:

Video classification has been extensively researched in computer vision due to its wide spread use in many important applications such as human action recognition and dynamic scene classification. It is highly desired to have an end-to-end learning framework that can establish effective video representations while simultaneously conducting efficient video classification. The convolution 3-D (C3-D) and VGG (vision and graphics group) are first deployed to extract temporal and spatial features from the input videos cooperatively, which establishes comprehensive and informative representations of videos. VGG and C3-D are chosen due to their strong capability of extracting complementary spatial and temporal features for comprehensive video representation.

CNNs by randomly throwing away certain activations, and then combines

these CNNs into one network. Such methods incorporate the ensemble theory into deep learning

by training many subnetworks. To leverage the strength of ensemble learning, we propose

incorporating random projections-based ensemble learning into the deep learning framework,

which enables effectively handling high-dimensional video representations for efficient and more

accurate video classification.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

The convolution 3-D (C3-D) and VGG (vision and graphics group) are first deployed to extract temporal and spatial features from the input videos cooperatively, which establishes comprehensive and informative representations of videos. VGG and C3-D are chosen due to their strong capability of extracting complementary spatial and temporal features for comprehensive video representations. The resultant high-dimensional representations are further reduced by random projections into a set of lower dimensional subspaces, on which an ensemble of efficient classifiers is trained with these lower dimensional features. The introduced RLE layer is further deployed to encode the initial outputs of classifiers, which is followed by a weighting layer jointly learned in the end-to-end framework to combine classification results.

PROPOSED SYSTEM

In this proposed system, we propose the convolution neural network method for action recognition in video. The input video will be captured by using the webcam. The input video is converted into number of frames. Then the CNN (Convolution Neural Network) algorithm is used in order to detect the particular part of the frame. Then the maximum weight values are taken from the feature extraction frames by using the Convolution neural network. Finally the action will be detected in the videos and then the label (action name) is identified

CHAPTER 3

REQUIREMENT SPECIFICATIONS

3.1 INTRODUCTION

VIDEO classification has been extensively researched in computer vision due to its wide spread use in many important applications such as human action recognition [1]–[5] and dynamic scene classification [6]–[8], Manuscript received July 30, 2017; revised January 30, 2018 and May 22, 2018; accepted May 31, 2018. Date of publication July 3, 2018; date of current version January 21, 2019. This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFB1200100, in part by the National Science Fund for Distinguished Young Scholars under Grant 61425014, in part by the National Natural Science Foundation of China under Grant 91538204, and in part by the Foundation for Innovative Research Groups of the National Natural Science Foundation of China under Grant 61521091. (*Corresponding author: Xianbin Cao.*) J. Zheng, X. Cao, and X. Zhen are with the School of Electronic and Information Engineering, Beihang University, Beijing 100191, China, and with the Key Laboratory of Advanced Technologies for Near Space Information Systems, Ministry of Industry and Information Technology of China, and also with the Beijing Advanced Innovation Center for Big Data-Based Precision Medicine, Beijing, China (e-mail: xbcao@buaa.edu.cn zhenxt@buaa.edu.cn) B. Zhang and X. Su are with the School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China. Color versions of one or more of the figures in this paper are available and so on. However, video sequences present great variability due to huge scale changes, viewpoint variation, and camera motion, which pose great

challenges for both video representations and classification. With the great success of deep video domain for a large variety of classification tasks. It is highly desired to have an end-to-end learning framework that can establish effective video representations while simultaneously conducting efficient video classification. Video representations have played a crucially important role in video analysis, which directly affect the ultimate performance of video classification. Both the spatial and temporal information should be captured and encoded for comprehensive and informative representation of video sequences. The two-stream CNNs [3], [9]–[11] that extract spatial and temporal features have shown better performance than the one-stream counterpart. Deep feature level fusion schemes have great capability of video representations for improving the performance of video classification. For instances, Karpathy *et al.* [12] proposed various fusion schemes to process the features; Sun *et al.* [13] used restricted Boltzmann machine [14] to perform the inference from complementary high-level features extracted from two ConvNet groups with a two-level average pooling hierarchy; Ng *et al.* [11] designed an end-to-end framework using the long short-term memory (LSTM) to process the features extracted from frames by CNNs. Those feature-level fusion methods concatenate the features simply for the following processing, which do not investigate the ensemble essence from the classification point of view. When it comes to video classification tasks, one has to consider not only the appearance information in the spatial domain but also the complex temporal evolution of the scenes. However, due to the fusion of spatial and temporal features, it would unavoidably induce high-dimensional video representations, which raise great challenge for efficient and accurate video classification. Ensemble learning has shown great capability of handling high-dimensional features to conduct accurate and efficient classification tasks. By working with deep learning models, ensemble learning can largely improve the performance [15]. For instance, both

vision and graphics group (VGG) [16] and AlexNet [17] use different initializations to retrain the networks and average the predictions. Dropout [18] trains multiple

CNNs by randomly throwing away certain activations, and then combines these CNNs into one network. Such methods incorporate the ensemble theory into deep learning by training many subnetworks. To leverage the strength of ensemble learning, we propose incorporating random projections-based ensemble learning into the deep learning framework, which enables effectively handling high-dimensional video representations for efficient and more accurate video classification.

3.2 SPECIFICATION HARDWARE AND SOFTWARE

3.1.1 HARDWARE REQUIREMENTS

- Hard Disk : 500GB and Above
- RAM : 4GB and Above
- Processor : I3 and Above

Webcam

3.2.1 SOFTWARE REQUIREMENTS



Operating System : Windows 7 , 8, 10 (64 bit)

Software : Python and Anaconda



Tools : Jupiter Note Book and Spyder



3.2 TECHNOLOGIES USED

CNN

Introduction Deep Learning

Deep Learning is a specialized form of Machine Learning that uses supervised, unsupervised, or semi-supervised learning to learn from data representations.

It is similar to the structure and function of the human nervous system, where a complex network of interconnected computation units works in a coordinated fashion to process complex information.

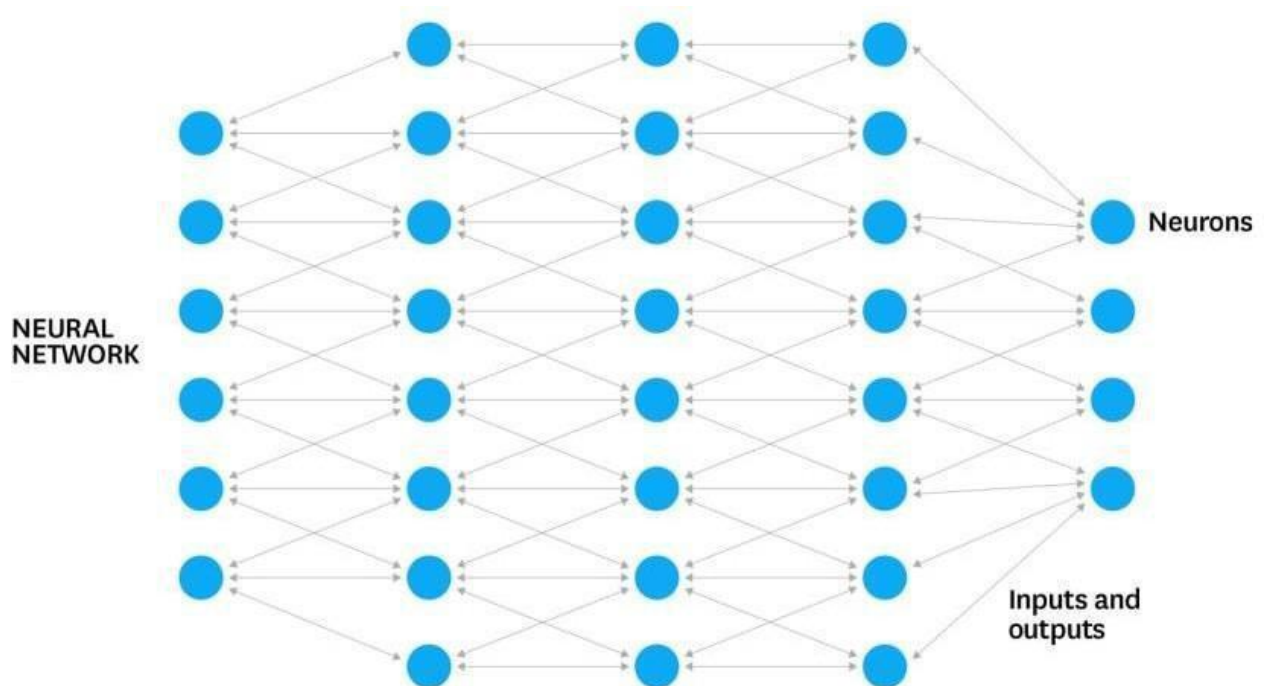
Machine Learning is an approach or subset of Artificial Intelligence that is based on the idea that machines can be given access to data along with the ability to learn from it. Deep Learning takes Machine Learning to the next level.

There are many aspects of Deep Learning as listed below

1. Multiple levels of hierarchical representations
2. Multi-layered neural networks
3. Training of large neural networks
4. Multiple non-linear transformations
5. Pattern recognition
6. Feature extraction
7. High-level data abstractions model

Human Brain vs. Artificial Neural Networks

The computational models in Deep Learning are loosely inspired by the human brain. The multiple layers of training are called Artificial Neural Networks (ANN).



ANNs are processing devices (algorithms or actual hardware) that are modelled on the neuronal structure of the mammalian cerebral cortex but on a

muchsmaller scale. It is a computing system made up of a number of simple, highly

interconnected processing elements which process information through their dynamic state response to external inputs.

Artificial Neural Networks Process

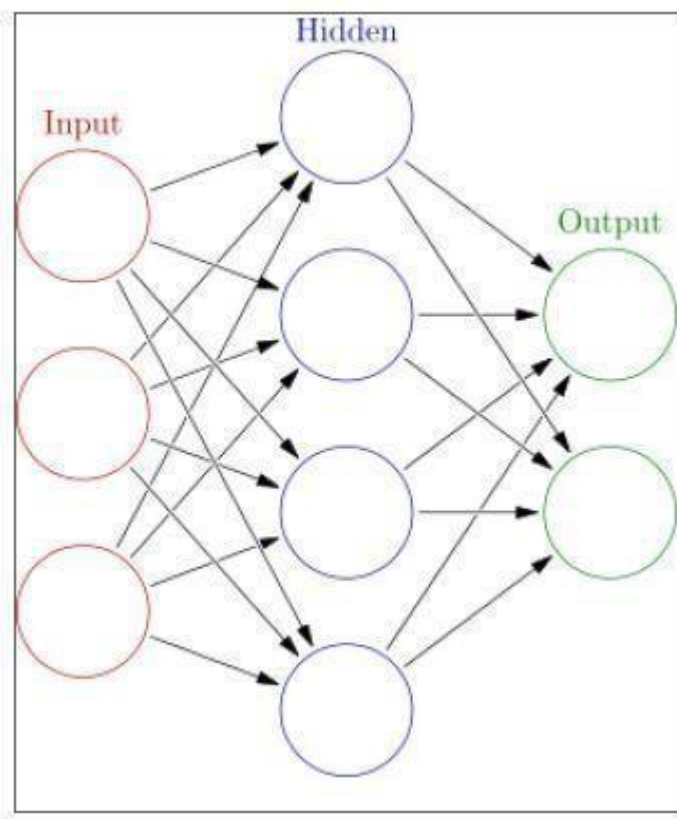
Artificial Neural Networks consist of the following four main parts:

Neurons

Nodes

Input

Output



Neuron

Artificial Neural Networks contain layers of neurons. A neuron is a computational unit that calculates a piece of information based on weighted input parameters. Inputs accepted by the neuron are separately weighted.

Inputs are summed and passed through a non-linear function to produce output. Each layer of neurons detects some additional information, such as edges of things in a picture or tumours in a human body. Multiple layers of neurons can be used to detect additional information about input parameters.

Nodes

Artificial Neural Network is an interconnected group of nodes akin to the vast network of layers of neurons in a brain. Each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another.

Inputs

Inputs are passed into the first layer. Individual neurons receive the inputs, with each of them receiving a specific value. After this, an output is produced based on these values.

Outputs

The outputs from the first layer are then passed into the second layer to be processed. This continues until the final output is produced. The assumption is that the correct output is predefined.

Each time data is passed through the network, the end result is compared with the correct one, and tweaks are made to their values until the network creates the correct final output each time.

Introduction CNN:

Convolutional neural networks (CNN) sounds like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year's ImageNet competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15%, an astounding improvement at the time. Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure.

Before we get into backpropagation, we must first take a step back and talk about what a neural network needs in order to work. At the moment we all were born, our minds were fresh. We didn't know what a cat or dog or bird was. In a similar sort of way, before the CNN starts, the weights or filter values are randomized. The filters don't know to look for edges and curves. The filters in the higher layers don't know to look for paws and beaks. As we grew older however, our parents and teachers showed us different pictures and images and gave us a corresponding label. This idea of being given an image and a label is the training process that CNNs go through. Before getting too into it, let's just say that we have a training set that has thousands of images of dogs, cats, and birds and each of the images has a label of what animal that picture is. Back to backprop.

However, the classic, and arguably most popular, use case of these networks is for image processing. Within image processing, let's take a look at how to use these CNNs for image classification.

The Problem Space

Image classification is the task of taking an input image and outputting a class (a cat, dog, etc.) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults. Without even thinking twice, we're able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we are able to immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly recognize patterns, generalize from prior knowledge, and adapt to different image environments are ones that we do not share with our fellow machines.



What We See

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 24 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 40 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 49 34 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 97 05 54
01 70 54 71 83 51 54 49 16 92 33 48 61 43 52 01 89 19 67 48
```

What Computers See

Inputs and Outputs

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values). Just to drive home the point, let's say we have a color image in JPG form and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255 which describe the pixel intensity at that point. These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer. The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class (.80 for cat, .15 for dog, .05 for bird, etc).

What We Want the Computer to do

Now that we know the problem as well as the inputs and outputs, let's think about how to approach this. What we want the computer to do is to be able to differentiate between all the images it's given and figure out the unique features that make a dog a dog or that make a cat a cat. This is the process that goes on in our minds subconsciously as well. When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or 4 legs. In a similar way, the computer is able perform image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts through a series of convolutional layers. This is a general overview of what a CNN does. Let's get into the specifics.

Biological Connection

But first, a little background. When you first heard of the term convolutional neural networks, you may have thought of something related to neuroscience or biology, and you would

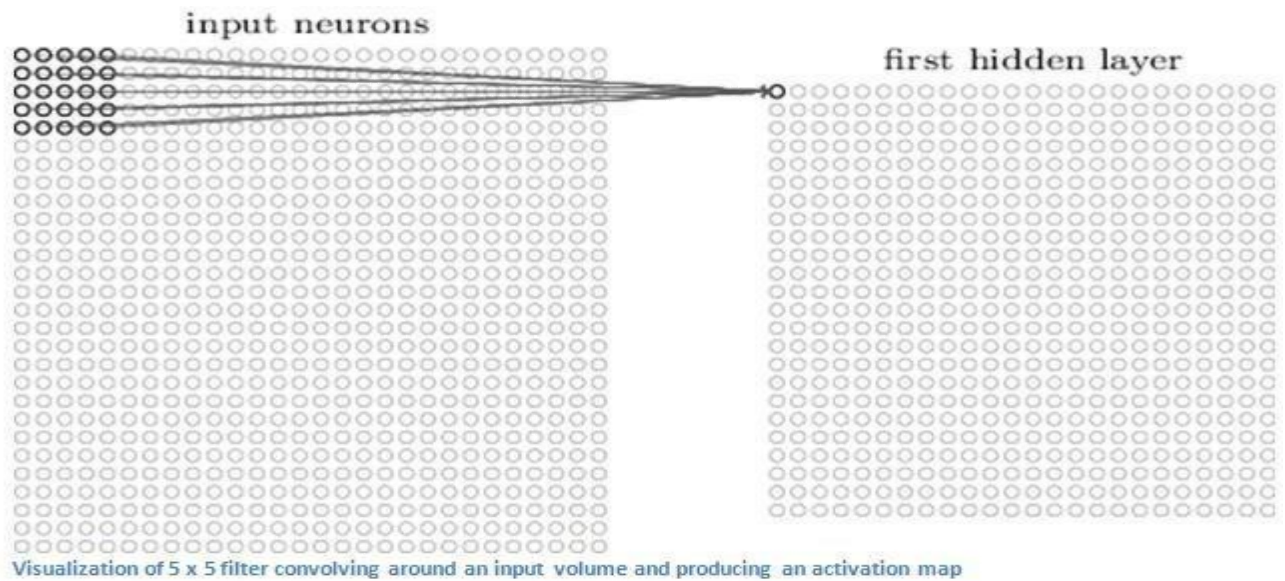
be right. Sort of. CNNs do take a biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel and Wiesel in 1962 (Video) where they showed that some individual neuronal cells in the brain responded (or fired) only in the presence of edges of a certain orientation. For example, some neurons fired when exposed to vertical edges and some when shown horizontal or diagonal edges. Hubel and Wiesel found out that all of these neurons were organized in a columnar architecture and that together, they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks (the neuronal cells in the visual cortex looking for specific characteristics) is one that machines use as well, and is the basis behind CNNs.

Structure

Back to the specifics. A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (down sampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image. Now, the hard part understands what each of these layers do. So let's get into the most important one.

First Layer – Math Part

The first layer in a CNN is always a Convolutional Layer. First thing to make sure you remember is what the input to this conv (I'll be using that abbreviation a lot) layer is. Like we mentioned before, the input is a $32 \times 32 \times 3$ array of pixel values. Now, the best way to explain a conv layer is to imagine a flashlight that is shining over the top left of the image. Let's say that the light this flashlight shines covers a 5×5 area. And now, let's imagine this flashlight sliding across all the areas of the input image. In machine learning terms, this flashlight is called a filter (or sometimes referred to as a neuron or a kernel) and the region that it is shining over is called the receptive field. Now this filter is also an array of numbers (the numbers are called weights or parameters). A very important note is that the depth of this filter has to be the same as the depth of the input (this makes sure that the math works out), so the dimension of this filter is $5 \times 5 \times 3$. Now, let's take the first position the filter is in for example. It would be the top left corner. As the filter is sliding, or convolving, around the input image, it is multiplying the values in the filter with the original pixel values of the image (aka computing element wise multiplications). These multiplications are all summed up (mathematically speaking, this would be 75 multiplications in total). So now you have a single number. Remember, this number is just representative of when the filter is at the top left of the image. Now, we repeat this process for every location on the input volume. (Next step would be moving the filter to the right by 1 unit, then right again by 1, and so on). Every unique location on the input volume produces a number. After sliding the filter over all the locations, you will find out that what you're left with is a $28 \times 28 \times 1$ array of numbers, which we call an activation map or feature map. The reason you get a 28×28 array is that there are 784 different locations that a 5×5 filter can fit on a 32×32 input image. These 784 numbers are mapped to a 28×28 array.



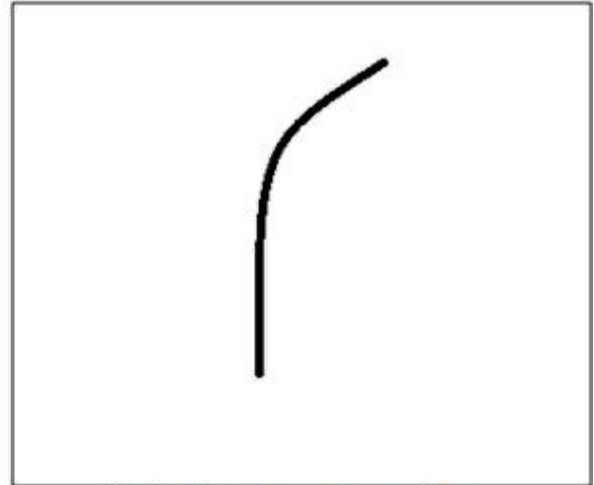
Let's say now we use two $5 \times 5 \times 3$ filters instead of one. Then our output volume would be $28 \times 28 \times 2$. By using more filters, we are able to preserve the spatial dimensions better. Mathematically, this is what's going on in a convolutional layer.

First Layer – High Level Perspective

However, let's talk about what this convolution is actually doing from a high level. Each of these filters can be thought of as **feature identifiers**. When I say features, I'm talking about things like straight edges, simple colors, and curves. Think about the simplest characteristics that all images have in common with each other. Let's say our first filter is $7 \times 7 \times 3$ and is going to be a curve detector. (In this section, let's ignore the fact that the filter is 3 units deep and only consider the top depth slice of the filter and the image, for simplicity.) As a curve detector, the filter will have a pixel structure in which there will be higher numerical values along the area that is a shape of a curve (Remember, these filters that we're talking about as just numbers!).

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

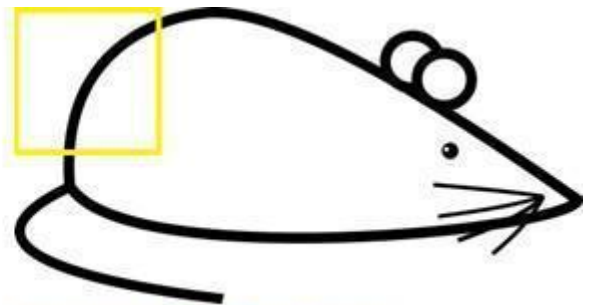


Visualization of a curve detector filter

Now, let's go back to visualizing this mathematically. When we have this filter at the top left corner of the input volume, it is computing multiplications between the filter and pixel values at that region. Now let's take an example of an image that we want to classify, and let's put our filter at the top left corner.



Original image



Visualization of the filter on the image

Remember, what we have to do is multiply the values in the filter with the original pixel values of the image.



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = $(50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600$ (A large number!)

Basically, in the input image, if there is a shape that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value! Now let's see what happens when we move our filter. The more training data that you can give to a network, the more training iterations you can make, the more weight updates you can make, and the better tuned to the network is when it goes to production. Facebook (and Instagram) can use all the photos of the billion users it currently has, Pinterest can use information of the 50 billion pins that are on its site, Google can use search data, and Amazon can use data from the millions of products that are bought every day. And now you know the magic behind how they use it.



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

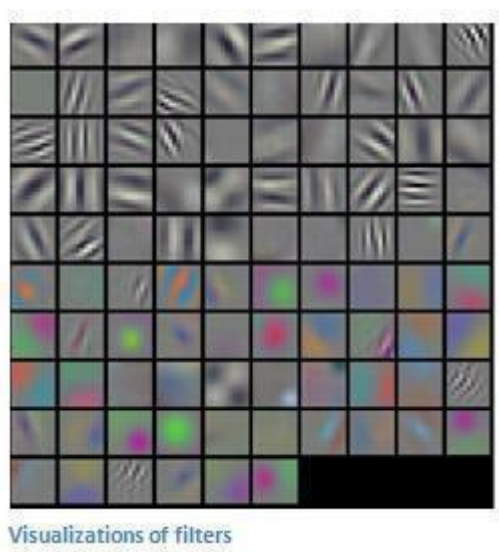
The value is much lower! This is because there wasn't anything in the image section that responded to the curve detector filter. Remember, the output of this conv layer is an activation map. So, in the simple case of a one filter convolution (and if that filter is a curve detector), the activation map will show the areas in which there are mostly likely to be curves in the picture. In this example, the top left value of our 26 x 26 x 1 activation map (26 because of the 7x7 filter instead of 5x5) will be 6600. This high value means that it is likely that there is some sort of curve in the input volume that caused the filter to activate. The top right value in our activation map will be 0 because there wasn't anything in the input volume that caused the filter to activate

(or more simply said, there wasn't a curve in that region of the original image). Remember, this is just for

one filter. This is just a filter that is going to detect lines that curve outward and to the right. We can have other filters for lines that curve to the left or for straight edges. The more filters, the greater the depth of the activation map, and the more information we have about the input volume.

Disclaimer:

The filter I described in this section was simplistic for the main purpose of describing the math that goes on during a convolution. In the picture below, you'll see some examples of actual visualizations of the filters of the first conv layer of a trained network. Nonetheless, the main argument remains the same. The filters on the first layer convolve around the input image and “activate” (or compute high values) when the specific feature it is looking for is in the input volume.



Going Deeper Through the Network

Now in traditional convolutional neural network architecture, there are other layers that are interspersed between these conv layers. I'd strongly encourage those interested to read up on them and understand their function and effects, but in a general sense, they provide nonlinearities and preservation of dimension that help to improve the robustness of the network and control over fitting. A classic CNN architecture would look like this.

Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool -> Fully Connected
--

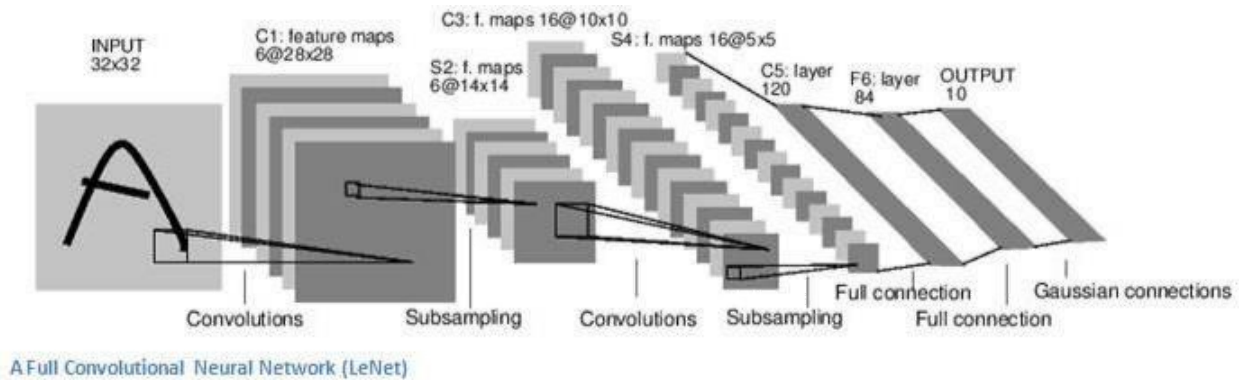
The last layer, however, is an important one and one that we will go into later on. Let's just take a step back and review what we've learned so far. We talked about what the filters in the

first conv layer are designed to detect. They detect low level features such as edges and curves.
As one

would imagine, in order predicting whether an image is a type of object, we need the network to be able to recognize higher level features such as hands or paws or ears. So let's think about what the output of the network is after the first conv layer. It would be a $28 \times 28 \times 3$ volume (assuming we use three $5 \times 5 \times 3$ filters). When we go through another conv layer, the output of the first conv layer becomes the input of the 2nd conv layer. Now, this is a little bit harder to visualize. When we were talking about the first layer, the input was just the original image. However, when we're talking about the 2nd conv layer, the input is the activation map(s) that result from the first layer. So each layer of the input is basically describing the locations in the original image for where certain low level features appear. Now when you apply a set of filters on top of that (pass it through the 2nd conv layer), the output will be activations that represent higher level features. Types of these features could be semicircles (combination of a curve and straight edge) or squares (combination of several straight edges). As you go through the network and go through more conv layers, you get activation maps that represent more and more complex features. By the end of the network, you may have some filters that activate when there is handwriting in the image, filters that activate when they see pink objects, etc. If you want more information about visualizing filters in ConvNets, Matt Zeiler and Rob Fergus had an excellent [research paper](#) discussing the topic. Jason Yosinski also has a [video](#) on YouTube that provides a great visual representation. Another interesting thing to note is that as you go deeper into the network, the filters begin to have a larger and larger receptive field, which means that they are able to consider information from a larger area of the original input volume (another way of putting it is that they are more responsive to a larger region of pixel space).

Fully Connected Layer

Now that we can detect these high level features, the icing on the cake is attaching a **fully connected layer** to the end of the network. This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from. For example, if you wanted a digit classification program, N would be 10 since there are 10 digits. Each number in this N dimensional vector represents the probability of a certain class. For example, if the resulting vector for a digit classification program is $[0.1 \ 0.1 \ 0.75 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0.05]$, then this represents a 10% probability that the image is a 1, a 10% probability that the image is a 2, a 75% probability that the image is a 3, and a 5% probability that the image is a 9 (Side note: There are other ways that you can represent the output, but I am just showing the soft max approach). The way this fully connected layer works is that it looks at the output of the previous layer (which as we remember should represent the activation maps of high level features) and determines which features most correlate to a particular class. For example, if the program is predicting that some image is a dog, it will have high values in the activation maps that represent high level features like a paw or 4 legs, etc. Similarly, if the program is predicting that some image is a bird, it will have high values in the activation maps that represent high level features like wings or a beak, etc. Basically, a FC layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.



Training (AKA: What Makes this Stuff Work)

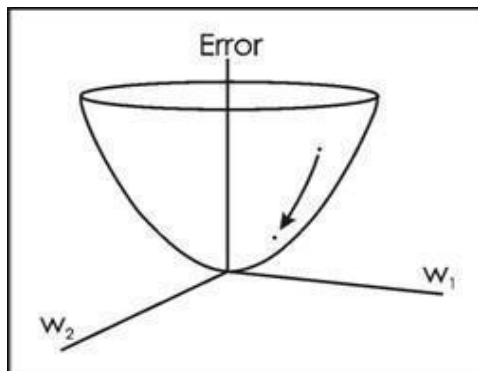
Now, this is the one aspect of neural networks that I purposely haven't mentioned yet and it is probably the most important part. There may be a lot of questions you had while reading. How do the filters in the first conv layer know to look for edges and curves? How does the fully connected layer know what activation maps to look at? How do the filters in each layer know what values to have? The way the computer is able to adjust its filter values (or weights) is through a training process called **backpropagation**.

So backpropagation can be separated into 4 distinct sections, the forward pass, the loss function, the backward pass, and the weight update. During the **forward pass**, you take a training image which as we remember is a 32 x 32 x 3 array of numbers and pass it through the whole network. On our first training example, since all of the weights or filter values were randomly initialized, the output will probably be something like [.1 .1 .1 .1 .1 .1 .1 .1 .1 .1], basically an output that doesn't give preference to any number in particular. The network, with its current weights, isn't able to look for those low level features or thus isn't able to make any reasonable conclusion about what the classification might be. This goes to the **loss function** part of backpropagation. Remember that what we are using right now is training data. This data has both an image and a label. Let's say for example that the first training image inputted was a 3. The label for the image would be [0 0 0 1 0 0 0 0 0]. A loss function can be defined in many different ways but a common one is MSE (mean squared error), which is $\frac{1}{2}$ times (actual - predicted) squared.

$$E_{total} = \sum \frac{1}{2} (target - output)^2$$

Let's say the variable L is equal to that value. As you can imagine, the loss will be extremely high for the first couple of training images. Now, let's just think about this intuitively. We want to get to a point where the predicted label (output of the ConvNet) is the same as the training label (This means that our network got its prediction right). In order to get there, we want to minimize the amount of loss we have. Visualizing this as just an optimization problem in calculus, we want to find out which inputs (weights in our case) most directly contributed to the

loss (or error) of the network.



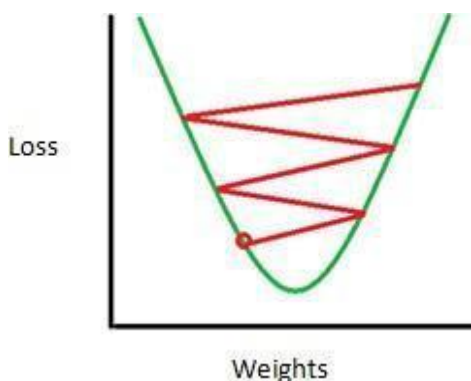
One way of visualizing this idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves trying to adjust the weights so that the loss decreases. In visual terms, we want to get to the lowest point in our bowl shaped object. To do this, we have to take a derivative of the loss (visual terms: calculate the slope in every direction) with respect to the weights.

This is the mathematical equivalent of a dL/dW where W are the weights at a particular layer. Now, what we want to do is perform a **backward pass** through the network, which is determining which weights contributed most to the loss and finding ways to adjust them so that the loss decreases. Once we compute this derivative, we then go to the last step which is the **weight update**. This is where we take all the weights of the filters and update them so that they change in the opposite direction of the gradient.

$$w = w_i - \eta \frac{dL}{dW}$$

w = Weight
 w_i = Initial Weight
 η = Learning Rate

The **learning rate** is a parameter that is chosen by the programmer. A high learning rate means that bigger steps are taken in the weight updates and thus, it may take less time for the model to converge on an optimal set of weights. However, a learning rate that is too high could result in jumps that are too large and not precise enough to reach the optimal point.



Consequence of a high learning rate where the jumps are too large and we are not able to minimize the loss.

The process of forward pass, loss function, backward pass, and parameter update is one training iteration. The program will repeat this process for a fixed number of iterations for each set of training images (commonly called a batch). Once you finish the parameter update on the last training example, hopefully the network should be trained well enough so that the weights of the layers are tuned correctly.

Testing

Finally, to see whether or not our CNN works, we have a different set of images and labels (can't double dip between training and test!) and pass the images through the CNN. We compare the outputs to the ground truth and see if our network works!

How Companies Use CNNs

Data, data, data. The companies that have lots of this magic 4 letter word are the ones that have an inherent advantage over the rest of the competition. The more training data that you can give to a network, the more training iterations you can make, the more weight updates you can make, and the better tuned to the network is when it goes to production. Facebook (and Instagram) can use all the photos of the billion users it currently has, Pinterest can use information of the 50 billion pins that are on its site, Google can use search data, and Amazon can use data from the millions of products that are bought every day. And now you know the magic behind how they use it.

Disclaimer

While this post should be a good start to understanding CNNs, it is by no means a comprehensive overview. Things not discussed in this post include the nonlinear and pooling layers as well as hyperparameters of the network such as filter sizes, stride, and padding. Topics like network architecture, batch normalization, vanishing gradients, dropout, initialization techniques, non-convex optimization, biases, and choices of loss functions, data augmentation, regularization methods, computational considerations, modifications of backpropagation. The more training data that you can give to a network, the more training iterations you can make, the more weight updates you can make, and the better tuned to the network is when it goes to production. Facebook (and Instagram) can use all the photos of the billion users it currently has, Pinterest can use information of the 50 billion pins that are on its site, Google can use search data, and Amazon can use data from the millions of products that are bought every day. And now you know the magic behind how they use it.

CHAPTER 4

4.5 Design and Implementation Constraints

4.5.1 Constraints in Analysis

Constraints as Informal Text

Constraints as Operational Restrictions

Constraints Integrated in Existing Model Concepts

Constraints as a Separate Concept

Constraints Implied by the Model Structure

4.5.2 Constraints in Design

Determination of the Involved Classes

Determination of the Involved Objects

Determination of the Involved Actions

Determination of the Require Clauses

Global actions and Constraint Realization

4.5.3 Constraints in Implementation

A hierarchical structuring of relations may result in more classes and a more complicated structure to implement. Therefore it is advisable to transform the hierarchical relation structure to a simpler structure such as a classical flat one. It is rather straightforward to transform the developed hierarchical model into a bipartite, flat model, consisting of classes on the one hand and flat relations on the other. Flat relations are preferred at the design level for reasons of simplicity and implementation ease. There is no identity or functionality associated with a flat relation. A flat relation corresponds with the relation concept of entity-relationship modeling and many object oriented methods.

4.6 Other Nonfunctional Requirements

4.6.1 Performance Requirements

The application at this side controls and communicates with the following three main general components.

embedded browser in charge of the navigation and accessing to the web service;

Server Tier: The server side contains the main parts of the functionality of the proposed architecture. The components at this tier are the following.

Web Server, Security Module, Server-Side Capturing Engine, Preprocessing Engine, Database System, Verification Engine, Output Module.

4.6.2 Safety Requirements

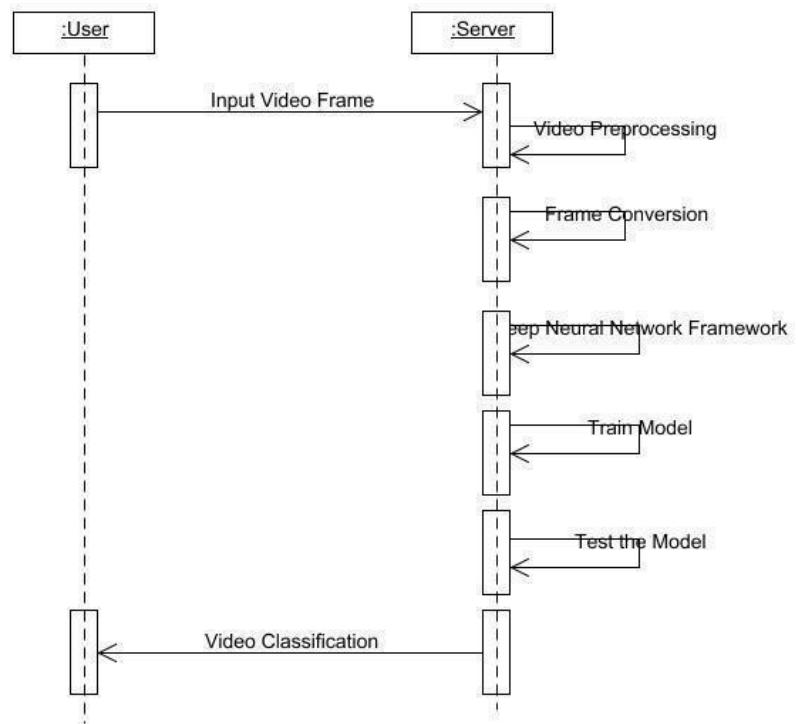
1. The software may be safety-critical. If so, there are issues associated with its integrity level
2. The software may not be safety-critical although it forms part of a safety-critical system. For example, software may simply log transactions.
3. If a system must be of a high integrity level and if the software is shown to be of that integrity level, then the hardware must be at least of the same integrity level.
4. There is little point in producing 'perfect' code in some language if hardware and system software (in widest sense) are not reliable.
5. If a computer system is to run software of a high integrity level then that system should not at the same time accommodate software of a lower integrity level.

6. Systems with different requirements for safety levels must be separated.
7. Otherwise, the highest level of integrity required must be applied to all systems

CHAPTER 5

5.1 Sequence Diagram:

A Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.



5.2 Use Case Diagram:

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed and was created by the Object Management Group. UML includes a set of graphic notation techniques to create visual models of software intensive systems. This language is used to specify, visualize, modify, construct and document the artifacts of an object oriented software intensive system under development.

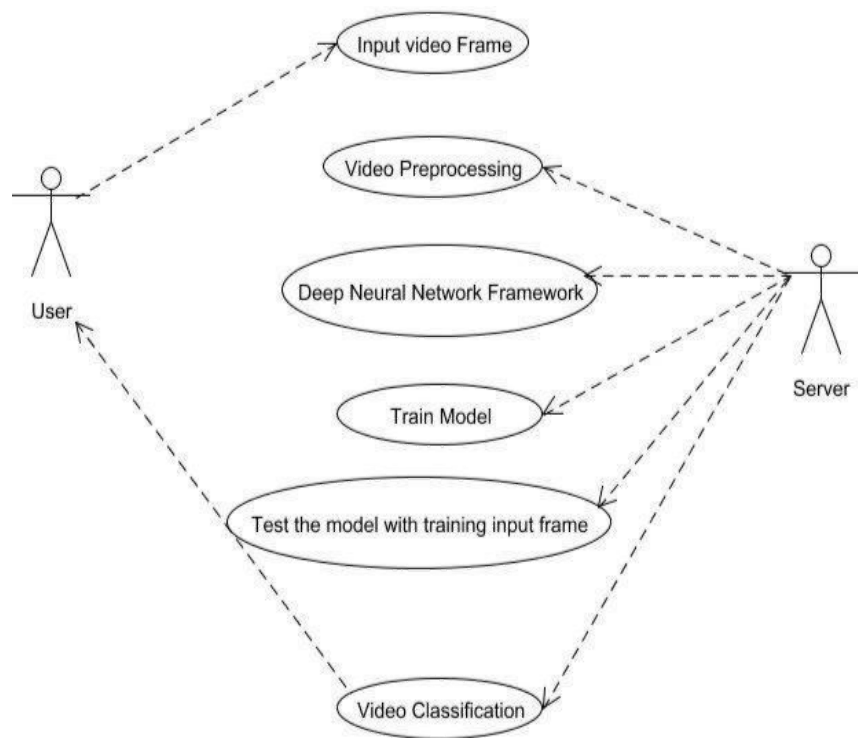
5.2.1 USECASE DIAGRAM

A Use case Diagram is used to present a graphical overview of the functionality provided by a system in terms of actors, their goals and any dependencies between those use cases.

Use case diagram consists of two parts:

Use case: A use case describes a sequence of actions that provided something of measurable value to an actor and is drawn as a horizontal ellipse.

Actor: An actor is a person, organization or external system that plays a role in one or more interaction with the system.



5.3 Activity Diagram:

Activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.

The most important shape types:

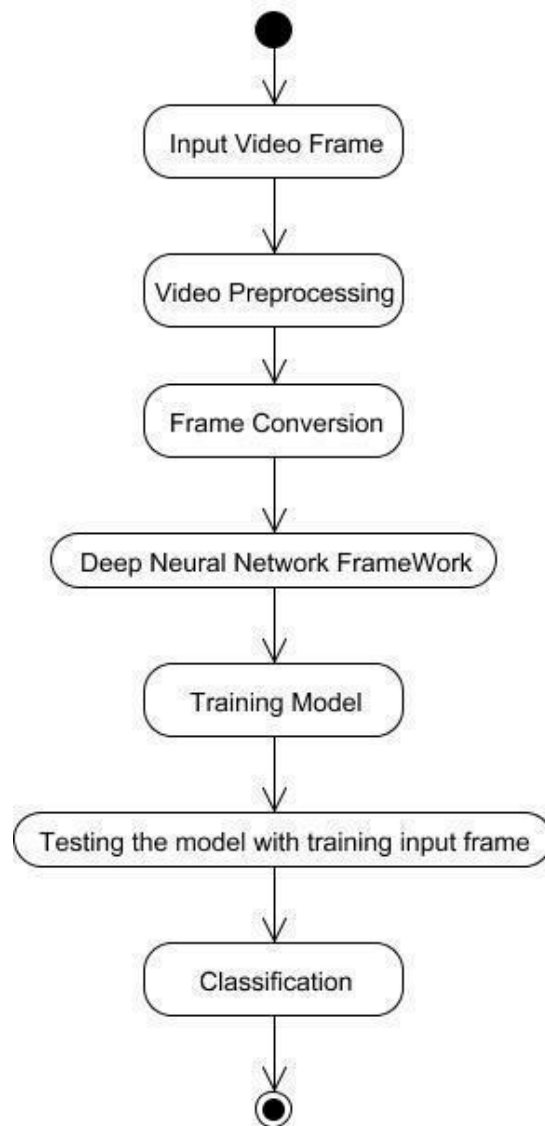
Rounded rectangles represent activities.

Diamonds represent decisions.

Bars represent the start or end of concurrent activities.

A black circle represents the start of the workflow.

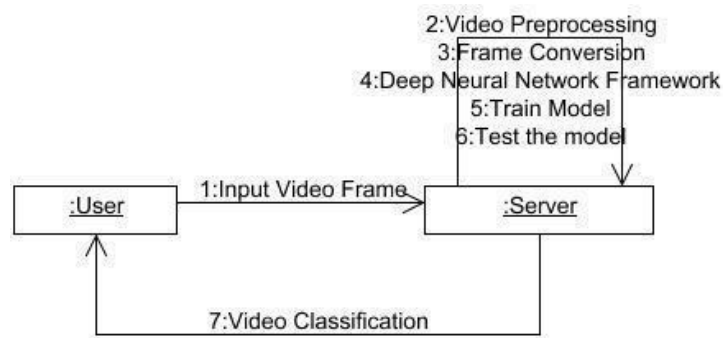
An encircled circle represents the end of the workflow.



5.4 Collaboration Diagram:

UML Collaboration Diagrams illustrate the relationship and interaction between software objects. They require use cases, system operation contracts and domain model to already exist.

The collaboration diagram illustrates messages being sent between classes and objects.



CHAPTER 6

6.1 MODULES

Video Streaming

Applying deep learning algorithm

Classification

6.2 MODULE EXPLANATION:

6.2.1 Module I: Video streaming

The input video will be captured by using the webcam. The input video is converted into number of frames. Video sequences present great variability due to huge scale changes, viewpoint variation and camera motion which pose great challenges for both video representations and classification.

Module II: Applying deep learning algorithm

The convolution neural networks (CNN) have been studied in the video domain for a large variety of classification tasks. The captured video is feed into the convolution neural network and it is highly desired to have an end-to-end learning framework that can establish effective video representations while simultaneously conducting efficient video classification

Module III: Classification

Then the maximum weight values are taken from the feature extraction frames by using the Convolution neural network. According to video labels the entire network is trained for action recognition. Finally the action will be detected in the videos and then label (action name) is id

CHAPTER 7

CODING AND TESTING

7.1 CODING

Once the design aspect of the system is finalized, the system enters into the coding and testing phase. The coding phase brings the actual system into action by converting the design of the system into the code in a given programming language. Therefore, a good coding style has to be taken whenever changes are required, it easily screws into the system.

7.2 CODING STANDARDS

Coding standards are guidelines to programming that focus on the physical structure and appearance of the program. They make the code easier to read, understand, and maintain. This phase of the system actually implements the blueprint developed during the design phase. The coding specification should be in such a way that any programmer must be able to understand the code and can bring about changes whenever felt necessary. Some of the standards needed to achieve the above-mentioned objectives are as follows:

Program should be simple, clear, and easy to understand.

Naming conventions

Value conventions

Script and comment procedure

Message box format

Exception and error handling

7.1.1 NAMING CONVENTIONS

Naming conventions of classes, data member, member functions, procedures etc., should be **self-descriptive**. One should even get the meaning and scope of the variable by its name. The conventions are adopted for **easy understanding** of the intended message by the user. So it is customary to follow the conventions. These conventions are as follows:

Class names

Class names are problem domain equivalence and begin with capital letter and have mixed

cases.

Member Function and Data Member name

Member function and data member name begins with a lowercase letter with each subsequent letters of the new words in uppercase and the rest of letters in lowercase.

7.1.2 VALUE CONVENTIONS

Value conventions ensure values for variable at any point of time. This involves the following:

Proper default values for the variables.

Proper validation of values in the field.

Proper documentation of flag values.

7.1.3 SCRIPT WRITING AND COMMENTING STANDARD

Script writing is an art in which indentation is utmost important. Conditional and looping statements are to be properly aligned to facilitate easy understanding. Comments

are included to minimize the number of surprises that could occur when going through the code.

7.1.4 MESSAGE BOX FORMAT

When something has to be prompted to the user, he must be able to understand it properly. To achieve this, a specific format has been adopted in displaying messages to the user. They are as follows:

X – User has performed illegal operation.

! – Information to the user.

1. TEST PROCEDURE

SYSTEM TESTING

Testing is performed to identify errors. It is used for quality assurance. Testing is

an integral part of the entire development and maintenance process. The goal of the testing during phase is to verify that the specification has been accurately and completely incorporated into the design, as well as to ensure the correctness of the design itself. For example the design must not have any logic faults in the design is detected before coding commences, otherwise the cost of fixing the faults will be considerably higher as reflected. Detection of design faults can be achieved by means of inspection as well as walkthrough.

Testing is one of the important steps in the software development phase. Testing checks for the errors, as a whole of the project testing involves the following test cases:

Static analysis is used to investigate the structural properties of the Source code.

Dynamic testing is used to investigate the behavior of the source code by executing the

program on the test data.

2. TEST DATA AND OUTPUT

7.1.5 UNIT TESTING

Unit testing is conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of the software design (i.e.), the module. The white-box testing techniques were heavily employed for unit testing.

7.1.6 FUNCTIONAL TESTS

Functional test cases involved exercising the code with nominal input values for which the expected results are known, as well as boundary values and special values, such as logically related inputs, files of identical elements, and empty files.

Three types of tests in Functional test:

Performance Test

Stress Test

Structure Test

7.1.7 PERFORMANCE TEST

It determines the amount of execution time spent in various parts of the unit, program throughput, and response time and device utilization by the program unit.

7.1.8 STRESS TEST

Stress Test is those test designed to intentionally break the unit. A Great deal can be learned about the strength and limitations of a program by examining the manner in which a programmer in which a program unit breaks.

7.1.9 STRUCTURED TEST

Structure Tests are concerned with exercising the internal logic of a program and traversing particular execution paths. The way in which White-Box test strategy was employed to ensure that the test cases could Guarantee that all independent paths within a module have been have been exercised at least once.

Exercise all logical decisions on their true or false sides.

Execute all loops at their boundaries and within their operational bounds.

Exercise internal data structures to assure their validity.

Checking attributes for their correctness.

Handling end of file condition, I/O errors, buffer problems and textual errors in output information

7.1.10 INTEGRATION TESTING

Integration testing is a systematic technique for construction the program structure while at the same time conducting tests to uncover errors associated with interfacing. i.e., integration testing is the complete testing of the set of modules which makes up the product. The objective is to take untested modules and build a program structure tester should identify critical modules. Critical modules should be tested as early as possible. One approach is to wait until all the units have passed testing, and then combine them and then tested. This approach is evolved from unstructured testing of small programs. Another strategy is to construct the product in

increments of tested units. A small set of modules are integrated together and tested, to which another module is added and tested in combination. And so on. The advantages of this approach are that, interface dispenses can be easily found and corrected.

The major error that was faced during the project is linking error. When all the modules are combined the link is not set properly with all support files. Then we checked out for interconnection and the links. Errors are localized to the new module and its intercommunications. The product development can be staged, and modules integrated in as they complete unit testing. Testing is completed when the last module is integrated and tested.

3. TESTING TECHNIQUES / TESTING STRATEGIES

7.1.11 TESTING

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet –undiscovered error. A successful test is one that uncovers an as-yet- undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hang together. System testing requires a test consists of several key activities and steps for run program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise the program or the project is not said to be complete. Software testing is the

critical element of software quality assurance and represents the ultimate the review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that as a probability of finding an yet undiscovered error. A successful test is one that uncovers an yet undiscovered error. Any engineering product can be tested in one of the two ways:

7.1.12 SOFTWARE TESTING STRATEGIES:

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned in advance and conducted systematically. For this reason a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

Testing begins at the module level and works “outward” toward the integration of the entire computer based system.

Different testing techniques are appropriate at different points in time.

The developer of the software and an independent test group conducts testing.

Testing and Debugging are different activities but debugging must be accommodated in any testing strategy.

7.5.1.1 WHITE BOX TESTING

This testing is also called as Glass box testing. In this testing, by knowing the specific functions that a product has been design to perform test can be conducted that demonstrate each function is fully operational at the same time searching for errors in each

function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis

path testing is a white box testing.

Basis path testing:

Flow graph notation

Cyclometric complexity

Deriving test cases

Graph matrices Control

7.5.1.2 BLACK BOX TESTING

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black box test case design are:

Graph based testing methods

Equivalence partitioning

Boundary value analysis

Comparison testing

7.5.2.1 INTEGRATION TESTING:

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with. Individual

modules, which are highly prone to interface errors, should not be assumed to work instantly when

we put them together. The problem of course, is “putting them together”- interfacing. There may be the chances of data lost across on another’s sub functions, when combined may not produce the desired major function; individually acceptable impression may be magnified to unacceptable levels; global data structures can present problems.

7.5.2.2 PROGRAM TESTING:

The logical and syntax errors have been pointed out by program testing. A syntax error is an error in a program statement that in violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax error. These errors are shown through error messages generated by the computer. A logic error on the other hand deals with the incorrect data fields, out-off-range items and invalid combinations. Since the compiler s will not deduct logical error, the programmer must examine the output. Condition testing exercises the logical conditions contained in a module. The possible types of elements in a condition include a Boolean operator, Boolean variable, a pair of Boolean parentheses A relational operator or on arithmetic expression. Condition testing method focuses on testing each condition in the program the purpose of condition test is to deduct not only errors in the condition of a program but also other a errors in the program.

7.5.2.3 SECURITY TESTING:

Security testing attempts to verify the protection mechanisms built in to a system well, in fact, protect it from improper penetration. The system security must be tested for invulnerability from frontal attack must also be tested for invulnerability from rear attack. During security, the tester places the role of individual who desires to penetrate system.

7.5.2.4 VALIDATION TESTING

At the culmination of integration testing, software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software test-validation testing begins. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in manner that is reasonably expected by the customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirement. After validation test has been conducted, one of two conditions exists.

- * The function or performance characteristics confirm to specifications and are accepted.
- * A validation from specification is uncovered and a deficiency created.

Deviation or errors discovered at this step in this project is corrected prior to completion of the project with the help of the user by negotiating to establish a method for resolving deficiencies. Thus the proposed system under consideration has been tested by using validation testing and found to be working satisfactorily. Though there were deficiencies in the system they were not catastrophic

7.5.2.5 USER ACCEPTANCE TESTING

User acceptance of the system is key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system and user at the time of developing and making changes whenever required. This is done in regarding to the following points.

- Input screen design.
- Output screen design.

Source Code

```
outputs = net.forward()

label = CLASSES[np.argmax(outputs)]
lbl=str(CLASSES[np.argmax(outputs)])
print(str(CLASSES[np.argmax(outputs)]
))

#      loop over our
frames for frame in
frames:

    #      draw the predicted activity on the frame
    cv2.rectangle(frame, (0, 0), (300, 40), (0, 0, 0),
-1) cv2.putText(frame, label, (10, 25) #
USAGE

#      python video_classification.py --model resnet-34_kinetics.onnx
--classes action_recognition_kinetics.txt --input
example_activities.mp4

# import the necessary packages

import numpy as np

import argparse

import imutils

import sys

import cv2

import datetime

import time

from firebase import firebase
```

```
# construct the argument parser and parse the arguments

firebase =
firebase.FirebaseApplication('https://video-10646-default-rtdb.firebaseio.com', None)

ap = argparse.ArgumentParser()
```

```

ap.add_argument("-m", "--model", required=True,
                help="path to trained human activity recognition model")

ap.add_argument("-c", "--classes",
                required=True, help="path to class labels
                file")

ap.add_argument("-i", "--input", type=str,
                default="", help="optional path to video
                file")

args = vars(ap.parse_args())

# load the contents of the class labels file, then define the sample
# duration (i.e., # of frames for classification) and sample size
# (i.e., the spatial dimensions of the frame)

CLASSES = open(args["classes"]).read().strip().split("\n")
SAMPLE_DURATION = 16
SAMPLE_SIZE = 112

# load the human activity recognition model

print("[INFO] loading human activity recognition model...")
net = cv2.dnn.readNet(args["model"])

# grab a pointer to the input video stream

print("[INFO] accessing video stream...")
vs = cv2.VideoCapture(args["input"] if args["input"] else 0)

# loop until we explicitly break from it

```

```
while True:
```

```
    # initialize the batch of frames that will be passed through the
```

```
    # model
```

```
    frames = []
```

```

# loop over the number of required
sample frames for i in range(0,
SAMPLE_DURATION):

    # read a frame from the
    video stream (grabbed, frame) =
    vs.read()

    # if the frame was not grabbed then we've reached the end of

    # the video stream so exit the script
    if not grabbed:

        print("[INFO] no frame read from stream - exiting")
        sys.exit(0)

    # otherwise, the frame was read so resize it and add it to

    # our frames list
    frame = imutils.resize(frame, width=400)
    frames.append(frame)

# now that our frames array is filled we can construct
our blob blob = cv2.dnn.blobFromImages(frames, 1.0,

    (SAMPLE_SIZE, SAMPLE_SIZE), (114.7748, 107.7354,
    99.4750), swapRB=True, crop=True)

blob = np.transpose(blob, (1, 0, 2,
3)) blob = np.expand_dims(blob,
axis=0)

# pass the blob through the network to obtain our human activity

# recognition predictions

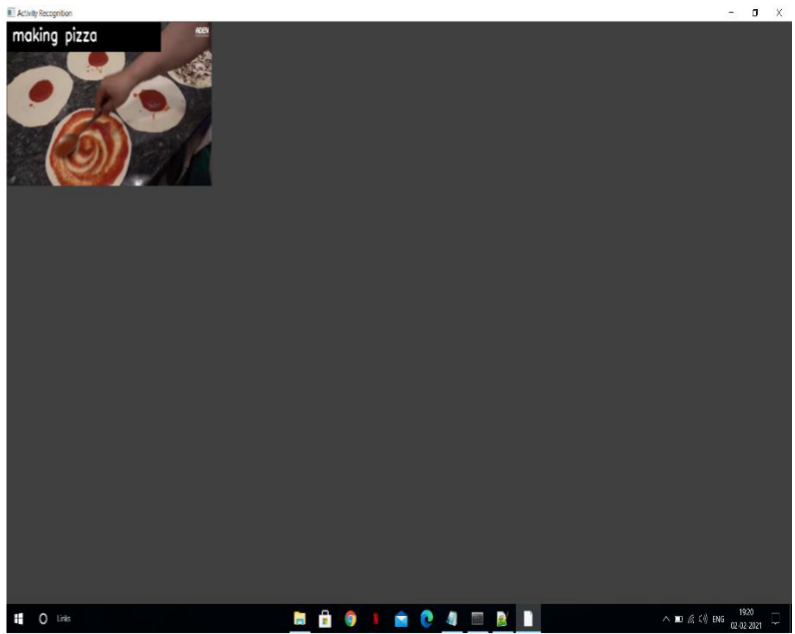
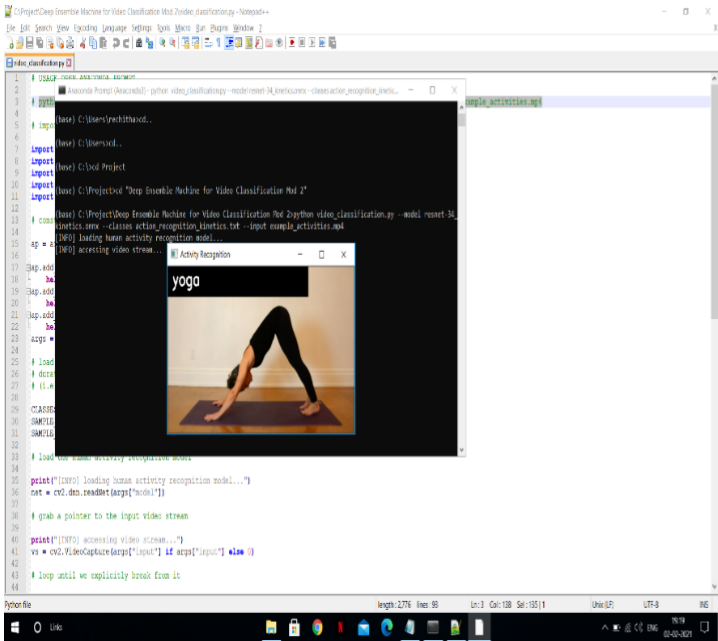
```

```
net.setInput(blob)
, cv2.FONT_HERSHEY_SIMPLEX,
    0.8, (255, 255, 255), 2)
datetime1=datetime.datetime.now()

Status=[str(CLASSES[np.argmax(outputs)])]
```

```
data1={"Status":[str(CLASSES[np.argmax(outputs)])]}\n\nprint(data1)\n\na=1\n\nif a==1:\n\n    firebase.put(" 'Location 1',data1)\n\n    #    display the frame to our screen\n    cv2.imshow("Activity Recognition",\n    frame) key = cv2.waitKey(1) & 0xFF\n\n\n    #    if the `q` key was pressed, break from\n    the loop if key == ord("q"):\n\n        break
```


Screenshots:



REFERENCES

- L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri, "Actions as space-time shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 12, pp. 2247–2253, Dec. 2007.
- [2] Z. Lin, Z. Jiang, and L. S. Davis, "Recognizing actions by shapemotion prototype trees," in *Proc. IEEE 12th Int. Conf. Comput. Vis.*, Sep./Oct. 2009, pp. 444–451.
- [3] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 568–576.
- [4] B. Zhang *et al.*, "One-two-one networks for compression artifacts reduction in remote sensing," *ISPRS J. Photogramm. Remote Sens.*, to be published, doi: [10.1016/j.isprsjprs.2018.01.003](https://doi.org/10.1016/j.isprsjprs.2018.01.003).
- [5] L. Shao, X. Zhen, D. Tao, and X. Li, "Spatio-temporal Laplacian pyramid coding for action recognition," *IEEE Trans. Cybern.*, vol. 44, no. 6, pp. 817–827, Jun. 2014.
- [6] C. Thériault, N. Thome, and M. Cord, "Dynamic scene classification: Learning motion descriptors with slow features analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 2603–2610.
- [7] A. Gangopadhyay, S. M. Tripathi, I. Jindal, and S. Raman, "Dynamic scene classification using convolutional neural networks," in *Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP)*, Dec. 2016, pp. 1255–1259.
- [8] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 4489–4497.

- [9] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional two-stream network fusion for video action recognition," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 1933–1941.
- [10] L. Wang *et al.*, "Temporal segment networks: Towards good practices for deep action recognition," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, 2016, pp. 20–36.
- [11] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 4694–4702.
- [12] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1725–1732.
- [13] Y. Sun, X. Wang, and X. Tang, "Hybrid deep learning for face verification," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2013, pp. 1489–1496.
- [14] G. Hinton, "A practical guide to training restricted Boltzmann machines," *Momentum*, vol. 9, no. 1, p. 926, 2012.
- [15] H. Song, J. J. Thiagarajan, P. Sattigeri, and A. Spanias, "Optimizing kernel machines using deep learning," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published, doi: [10.1109/TNNLS.2018.2804895](https://doi.org/10.1109/TNNLS.2018.2804895).
- [16] K. Simonyan and A. Zisserman. (2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: <https://arxiv.org/abs/1409.1556>

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

- [18] G. Hinton, O. Vinyals, and J. Dean. (2015). “Distilling the knowledge in a neural network.” [Online]. Available: <https://arxiv.org/abs/1503.02531>
- [19] M. Liu, D. Zhang, S. Chen, and H. Xue, “Joint binary classifier learning for ECOC-based multi-class classification,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 11, pp. 2335–2341, Nov. 2016.
- [20] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 807–814.
- [21] H. Liu, N. Shu, Q. Tang, and W. Zhang, “Computational model based on neural network of visual cortex for human action recognition,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 5, pp. 1427–1440, May 2018.
- [22] A. Iosifidis, A. Tefas, and I. Pitas, “View-invariant action recognition based on artificial neural networks,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 3, pp. 412–424, Mar. 2012.
- [23] L. Niu, X. Xu, L. Chen, L. Duan, and D. Xu, “Action and event recognition in videos by learning from heterogeneous Web sources,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 6, pp. 1290–1304, Jun. 2017.
- [24] X. Zhen, F. Zheng, L. Shao, X. Cao, and D. Xu, “Supervised local descriptor learning for human action recognition,” *IEEE Trans. Multimedia*, vol. 19, no. 9, pp. 2056–2065, Sep. 2017.
- [25] K. G. Derpanis, M. Lecce, K. Daniilidis, and R. P. Wildes, “Dynamic scene understanding: The role of orientation features in space and time in scene classification,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2012, pp. 1306–1313.
- [26] N. Shroff, P. Turaga, and R. Chellappa, “Moving vistas: Exploiting motion for describing

scenes,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2010, pp. 1911–1918.

- [27] X. Zhen and L. Shao, "Action recognition via spatio-temporal local features: A comprehensive study," *Image Vis. Comput.*, vol. 50, pp. 1–13, Jun. 2016.
- [28] C. Hua, Y. Makihara, and Y. Yagi, "Pedestrian detection by using a spatio-temporal histogram of oriented gradients," *IEICE Trans. Inf. Syst.*, vol. E96-D, no. 6, pp. 1376–1386, 2013.
- [29] P. Scovanner, S. Ali, and M. Shah, "A 3-dimensional SIFT descriptor and its application to action recognition," in *Proc. 15th ACM Int. Conf. Multimedia*, 2007, pp. 357–360.
- [30] B. Zhang, Y. Yang, C. Chen, L. Yang, J. Han, and L. Shao, "Action recognition using 3D histograms of texture and a multi-class boosting classifier," *IEEE Trans. Image Process.*, vol. 26, no. 10, pp. 4648–4660, Oct. 2017.
- [31] X. Wang, A. Farhadi, and A. Gupta, "Actions _ transformations," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 2658–2667.
- [32] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.
- [33] Y. Freund, "Boosting a weak learning algorithm by majority," *Inf. Comput.*, vol. 121, no. 2, pp. 256–285, Sep. 1995.
- [34] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, Aug. 1998.
- [35] R. E. Schapire, "The strength of weak learnability," *Mach. Learn.*, vol. 5, no. 2, pp. 197–227, 1990.
- [36] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[37] P. Bachman, O. Alsharif, and D. Precup, “Learning with pseudoensembles,” in *Proc. Adv.*

Neural Inf. Process. Syst., 2014, pp. 3365–3373.

- [38] T. Sercu, C. Puhersch, B. Kingsbury, and Y. LeCun, “Very deep multilingual convolutional neural networks for LVCSR,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2016, pp. 4955–4959.
- [39] A. Veit, M. J. Wilber, and S. Belongie, “Residual networks behave like ensembles of relatively shallow networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 550–558.
- [40] J. Qin *et al.*, “Zero-shot action recognition with error-correcting output codes,” in *Proc. CVPR*, Jul. 2017, pp. 1042–1051.
- [41] T. G. Dietterich and G. Bakiri, “Solving multiclass learning problems via error-correcting output codes,” *J. Artif. Intell. Res.*, vol. 2, no. 1, pp. 263–286, 1995.
- [42] O. Pujol, P. Radeva, and J. Vitria, “Discriminant ECOC: A heuristic method for application dependent design of error correcting output codes,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 6, pp. 1007–1012, Jun. 2006.
- [43] J. D. Zhou, X. D. Wang, H. J. Zhou, J. M. Zhang, and N. Jia, “Decoding design based on posterior probabilities in ternary error-correcting output codes,” *Pattern Recognit.*, vol. 45, no. 4, pp. 1802–1818, 2012.
- [44] T. Takenouchi and S. Ishii, “Ternary bradley-terry model-based decoding for multi-class classification,” in *Proc. IEEE Workshop Mach. Learn. Signal Process. (MLSP)*, Oct. 2008, pp. 121–126.
- [45] G. Zhong, K. Huang, and C.-L. Liu, “Joint learning of error-correcting output codes and dichotomizers from data,” *Neural Comput. Appl.*, vol. 21, no. 4, pp. 715–724, 2012.
- [46] K. Crammer and Y. Singer, “On the learnability and design of output codes for multiclass

problems,” *Mach. Learn.*, vol. 47, no. 2, pp. 201–233, 2002.

- [47] M. A. Bagheri, Q. Gao, and S. Escalera, "A genetic-based subspace analysis method for improving error-correcting output coding," *Pattern Recognit.*, vol. 46, no. 10, pp. 2830–2839, 2013.
- [48] C. Deng, Z. Chen, X. Liu, X. Gao, and D. Tao, "Triplet-based deep hashing network for cross-modal retrieval," *IEEE Trans. Image Process.*, vol. 27, no. 8, pp. 3893–3903, Aug. 2018.
- [49] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 769–790, Apr. 2018.
- [50] Z. Lin, G. Ding, J. Han, and J. Wang, "Cross-view retrieval via probability-based semantics-preserving hashing," *IEEE Trans. Cybern.*, vol. 47, no. 12, pp. 4342–4355, Dec. 2017.
- [51] Y. Guo, G. Ding, L. Liu, J. Han, and L. Shao, "Learning to hash with optimized anchor embedding for scalable retrieval," *IEEE Trans. Image Process.*, vol. 26, no. 3, pp. 1344–1354, Mar. 2017.
- [52] Z. Lin, G. Ding, J. Han, and L. Shao, "End-to-end feature-aware label space encoding for
multilabel classification with many classes," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2472–2487, Jun. 2018.
- [53] D. Achlioptas, "Database-friendly random projections: Johnson- Lindenstrauss with binary coins," *J. Comput. Syst. Sci.*, vol. 66, no. 4, pp. 671–687, 2003.
- [54] W. B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space," *Contemp. Math.*, vol. 26, nos. 189–206, p. 1, 1984.

- [55] T. G. Dietterich, “Ensemble methods in machine learning,” in *Proc. 1st Int. Workshop Multiple Classifier Syst.* Springer, 2000, pp. 1–15.
- [56] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” M.S.

thesis, Univ. Toronto, Toronto, ON, Canada, 2009.

[57] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc.*

IEEE Conf. Comput. Vis. Pattern Recognit.,

Jun. 2016, pp. 770–778.

[58] K. Soomro, A. R. Zamir, and M. Shah. (2012). “UCF101: A dataset of 101 human actions

classes from videos in the wild.” [Online]. Available: <https://arxiv.org/abs/1212.0402>

[59] H. Kuehne, H. Jhuang, R. Stiefelhagen, and T. Serre, “HMDB51: A large video database for

human motion recognition,” in *High Performance Computing in Science and Engineering*.

Berlin, Germany: Springer, 2013, pp. 571–582.

[60] B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva. (2016). “Places: An image database for deep scene understanding.” [Online]. Available: <https://arxiv.org/abs/1610.02055>

[61] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*,

Jun. 2009, pp. 248–255.

[62] X. Peng, L. Wang, X. Wang, and Y. Q. Qiao, “Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice,” *Comput. Vis. Image*

Understand., vol. 150, pp. 109–125, 2016.

design to derive test cases. Basis path testing is a white box testing.

Basis path testing:

Flow graph notation

Cyclometric complexity

Deriving test cases

Graph matrices Control

7.5.1.3 BLACK BOX TESTING

In this testing by knowing the

Graph based testing methods

Equivalence partitioning

Boundary value analysis

Comparison testing

7.1.13 SOFTWARE TESTING STRATEGIES:

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned in advance and conducted systematically. For this reason a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

Testing begins at the module level and works “outward” toward the integration of the entire computer based system.

Different testing techniques are appropriate at different points in time.

desired major function; individually acceptable impression may be magnified to unacceptable levels; global data structures can present problems.

7.5.2.6 PROGRAM TESTING:

The logical and syntax errors have been pointed out by program testing. A syntax error is an error in a program statement that in violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax error. These errors are shown through error messages generated by the computer. A logic error on the other hand deals with the incorrect data fields, out-off-range items and invalid combinations. Since the compiler s will not deduct logical error, the programmer must examine the output. Condition testing exercises the logical conditions contained in a module. The possible types of elements in a condition include a Boolean operator, Boolean variable, a pair of Boolean parentheses A relational operator or on arithmetic expression. Condition testing method focuses on testing each condition in the program the purpose of condition test is to deduct not only errors in the condition of a program but also other a errors in the program.

7.5.2.7 SECURITY TESTING:

Security testing attempts to verify the protection mechanisms built in to a system well, in fact, protect it from improper penetration. The system security must be tested for invulnerability from frontal attack must also be tested for invulnerability from rear attack. During security, the tester places the role of individual who desires to penetrate system.

7.5.2.8 VALIDATION TESTING

At the culmination of integration testing, software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software test-validation testing begins. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in manner that is reasonably expected by the customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirement. After validation test has been conducted, one of two conditions exists.

- * The function or performance characteristics confirm to specifications and are accepted.
- * A validation from specification is uncovered and a deficiency created.

Deviation or errors discovered at this step in this project is corrected prior to completion of the project with the help of the user by negotiating to establish a method for resolving deficiencies. Thus the proposed system under consideration has been tested by using validation testing and found to be working satisfactorily. Though there were deficiencies in the system they were not catastrophic

7.5.2.9 USER ACCEPTANCE TESTING

User acceptance of the system is key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system and user at the time of developing and making changes whenever required. This is done in regarding to the following points.

Input screen design.

Output screen design.

