

FOREST FIRE DETECTION AND RECOGNITION

at

**Sathyabama Institute of Science and Technology
(Deemed to be University)**

Submitted in partial fulfillment of the requirements for the award of Bachelor
of Engineering Degree in Computer Science and Engineering

By

**TAMMALAPUDI NAGESWARA SATYA PRASAD
REG. NO: 38110590**



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

**SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119, TAMILNADU**

MAY 2022



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

Accredited with Grade “A” by NAAC
(Established under Section 3 of UGC Act, 1956)
JEPPIAAR NAGAR, RAJIV GANDHI SALAI
CHENNAI– 600119
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 **Tammalapudi Nageswara Satya Prasad (38110590)** who carried out the project entitled “**FOREST FIRE DETECTION AND RECOGNITION**” under my supervision from November 2021 to March 2022.

Internal Guide

Dr.Malini Deepika

Head of the Department

Dr. S. Vigneshwari M.E., Ph.D., and Dr. L. Lakshmanan M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **Tammalapudi Nageswara Satya Prasad** hereby declare that the project report entitled **FOREST FIRE DETECTION AND RECOGNITION** done by me under the guidance of **Dr. Malini Deepika** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering.

DATE:10-04-2022

PLACE:

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. Vigneshwari, 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.Malini Deepika** for his 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

Forest fire is becoming one of the most significant natural disasters at the expense of ecology and economy. In this article, we develop an effective SqueezeNet based asymmetric encoder-decoder U-shape architecture, Attention U-Net and SqueezeNet (ATT Squeeze U-Net), mainly functions as an extractor and a discriminator of forest fire. This model takes attention mechanism to highlight useful features and suppress irrelevant contents by embedding Attention Gate (AG) units in the skip connection of U-shape structure. In this way, salient features are emphasized so that the proposed method could be competent at forest fire segmentation tasks with a small number of parameters. Specifically, we first replace classical convolution layer by a depthwise one and engage a Channel Shuffle operation as a feature communicator in the Fire module of classical SqueezeNet. Then, this modified SqueezeNet is employed as a substitution of the encoder of Attention U-Net and a corresponding DeFire module designed is combined into the decoder as well. Finally, to classify true fire, we take use of a fragment of the encoder in ATT Squeeze U-Net. The experimental results of modified SqueezeNet integrated Attention U-Net show that a competitive accuracy at 0.93 and an average prediction time at 0.89 second per image are achieved for reliable real-time forest fire detection.

	TABLE OF CONTENTS	
CHAPTER No.	TITLE	PAGE No
	ABSTRACT	i
	LIST OF FIGURES	iv
	LIST OF TABLES	v
	LIST OF ABBREVIATIONS	vi
1	INTRODUCTION	1
	1.1 intrudction	1
	1.2 fucture scope	2
2.	AIM AND SCOPE OF THE PRESENT INVESTIGATION	3
	2.1 literature survey	3
³	METHODOLOGY	5
		4

3. EXPERIMENTAL OR MATERIALS AND METHODS, ALGORITHMS USED

7

	3.1 existing system	7
	3.2 proposed system	8
	3.3 software and hardware requiriments	8
	3.3.1 hardware requirements	8
	3.3.2 software requirements	9
	3.4 Algorithm Used	9

	3.5 Programming Languages	9
	3.6 Introduction to Deep Learning	10
	3.7 CNN	12
	3.8 MODULES	14
	3.8.1 Module E xplanation	14
	3.8.2 Dataset Creation	18

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
	1.1	
	1.2	
	2.1	
	3.1	

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO.
1.1		
1.2		
2.1		
3.1		

LIST OF ABBREVIATIONS

ABBREVIATION

EXPANSION

CHAPTER 1

INTRODUCTION

1.1 Introduction

Early detection and identification of forest fire can avoid damaging disaster. Fire detection methods such as satellite-based detection, optical sensing, wireless sensing and remote sensing gain notable improvements to forest fire alarm. In this study, we focus on monitoring fire detection driven by computer vision. Computer vision mechanisms for fire detection could be mainly classified into two categories, traditional image processing method and deep Convolutional Neural Network (CNN) method. Existing conventional detection algorithms mainly operate based on visual properties of fire, such as color, spectral, texture, motion and geometric features. Despite the low cost and simplicity, traditional methods strongly rely on appropriate feature description of fire. Some natural phenomena, such as sunset and fog would cause false alarm and missing report to these approaches occasionally. To solve these problems, a more advanced fire detection scheme proposing the use of CNN technology instead of feature description has attracted more and more attention. Meanwhile, recent development of GPU allows the use of CNN-based methods for fire detection. Common disadvantage of them seems to be that large datasets are required to learn the best features. As a result of this, the model would over-fitted under huge training dataset, whereas applying a small number of dataset for learning would be insufficient. Recently, some lightweight compression networks [1] that could achieve real-time processing have been introduced when reasonable mistakes are allowed. In this article, we propose an efficient neural network architecture for forest fire detection and recognition based on Attention U-Net and SqueezeNet (ATT Squeeze U-Net). The proposed framework consists of two stages, a segmentation module extracting the shape of forest fire, and a classification module identifying whether the detected fire area is true or not. We remove the encoder of conventional Attention U-Net [2] and place our modified SqueezeNet with new designed Fire modules at the contraction path. Besides, some implementations are carried out on traditional Fire module [3] in SqueezeNet. We replace the original 3 × 3 convolution layer with a Depthwise Convolution (DWConv) kernel, and a Channel Shuffle operation is added to have feature communication enhanced. We design corresponding DeFire modules for ATT Squeeze U-Net model and embed them into the decoder for better up-sampling. We then develop a new classification framework for fire identification by reusing a part of the encoder of ATT Squeeze U-Net. A discussion of how many output feature maps of the encoder layers are chosen is raised to reach a most effective selection for subsequent fire recognition. We evaluated the ATT Squeeze U-Net on some publicly available datasets, and the results demonstrate that the proposed framework can produce better fire area extraction results than some existing

algorithms.

1.2 Future Scope

It can be summarizing that the future scope of the project circles around maintaining information regarding: It helps us to determine the efficient and more accurate results for recognition of handwritten digits. Various challenges are identified which may provide more lively interest to the researchers. The main challenge is to identify the diverse human writing styles, different angles, different shapes and size of digits, pure input quality, low accuracy rate in recognition etc. Hence, a lot of research work is done to solve these problems. Handwritten Digit Recognition is the capacity of a computer to interpret the manually written digits from various sources like Messages, Bank cheques, Papers and pictures, Postal mail sorting, Form data entry, Digits entered in any forms etc.

CHAPTER 2

LITERATURE SURVEY

2.1 Literature Survey

Fire attack in forest makes major degradation in the forest environment and ecosystems. Forest fire detection in the early stage can prevent major causes due to fire attack. A novel digital image processing based on multi-feature best decision-based forest fire detection (MF-BD-FFD) is proposed in this work. To increase the sensitivity of detection, color and texture feature with hybrid decision making algorithms such as artificial neural networks (ANN), Support Vector machine (SVM), k- nearest classifier (KNN) is used and optimized output will be selected. By using proposed method, the accuracy of the system is increased with a factor of 5% when compared to the conventional technique.

The Convolutional Neural Network (CNN) has been used in many fields and has achieved remarkable results, such as image classification, face detection, and speech recognition. Compared to GPU (graphics processing unit) and ASIC, a FPGA (field programmable gate array)-based CNN accelerator has great advantages due to its low power consumption and reconfigurable property. However, FPGA's extremely limited resources and CNN's huge amount of parameters and computational complexity pose great challenges to the design. Based on the ZYNQ heterogeneous platform and the coordination of resource and bandwidth issues with the roofline model, the CNN accelerator we designed can accelerate both standard convolution and depthwise separable convolution with a high hardware resource rate. The accelerator can handle network layers of different scales through parameter configuration and maximizes bandwidth and achieves full pipelined by using a data stream interface and ping-pong on-chip cache. The experimental results show that the accelerator designed in this paper can achieve 17.11GOPS for 32bit floating point when it can also accelerate depthwise separable convolution, which has obvious advantages compared with other designs.

We present a novel and practical deep fully convolutional neural network architecture for semantic pixel-wise segmentation termed SegNet. This core trainable segmentation engine consists of an encoder network, a corresponding decoder network followed by a pixel-wise classification layer. The architecture of the encoder network is topologically identical to the 13 convolutional layers in the VGG16 network [1]. The role of the decoder network is to map the low resolution encoder feature maps to full input resolution feature maps for

pixel-wise classification. The novelty of SegNet lies in the manner in which the decoder upsamples its lower resolution input feature map(s). Specifically, the decoder uses pooling indices computed in the max-pooling step of the corresponding encoder to perform non-linear upsampling. This eliminates the need for learning to upsample. The upsampled maps are sparse and are then convolved with trainable filters to produce dense feature maps. We compare our proposed architecture with the widely adopted FCN [2] and also with the well known DeepLab-LargeFOV [3], DeconvNet [4] architectures. This comparison reveals the memory versus accuracy trade-off involved in achieving good segmentation performance. SegNet was primarily motivated by scene understanding applications. Hence, it is designed to be efficient both in terms of memory and

computational time during inference. It is also significantly smaller in the number of trainable parameters than other competing architectures and can be trained end-to-end using stochastic gradient descent. We also performed a controlled benchmark of SegNet and other architectures on both road scenes and SUN RGB-D indoor scene segmentation tasks. These quantitative assessments show that SegNet provides good performance with competitive inference time and most efficient inference memory-wise as compared to other architectures. We also provide a Caffe implementation of SegNet and a web demo.

CHAPTER 3

METHODOLOGY

3.1 Existing system

To detect fire, researchers have presented both traditional and learned representation based fire detection methods. In literature, the traditional methods use either color or motion characteristics for fire detection. For instance, used color features for fire detection by exploring different color models including HSI, YUV, YCbCr, RGB, and YUC. The major issue with these methods is their high rate of false alarms. Several attempts have been made to solve this issue by combining the color information with motion and analyses of fire's shape and other characteristics

3.2 Proposed system

We propose an efficient CNN based system for fire detection in videos captured in uncertain surveillance scenarios. Our approach uses light-weight deep neural networks with no dense fully connected layers, making it computationally inexpensive. Once detect a fire the information will pass through the firebase. Firebase is a one type of cloud database. Then the firebase to send a notification in android smartphone

3.3 Software And Hardware Requirements

3.3.1 Hardware Requirements

- Hard Disk - 500Gb and Above
- Ram - 4Gb and Above
- Processor - i3 and Above
- Webcam1

3.3.2 Software Requirements

- Operating System - Windows 7,8,10(64 bit)
- Software - Python 3.7
- Tools - Anaconda(Jupyter Notebook IDE)
- Database - Firebase Database

3.4 Algorithm Used

CNN-Convolution Neural Network

3.5 Programming Languages

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- System scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- Python 2.0 was released in 2000, and the 2.x versions were the prevalent releases until December 2008. At that time, the development team made the decision to release version 3.0, which contained a few relatively small but significant changes that were not backward compatible with the 2.x versions. Python 2 and 3 are very similar, and some features of Python 3 have been backported to Python 2. But in general, they remain not quite compatible.
- Both Python 2 and 3 have continued to be maintained and developed, with periodic release updates for both. As of this writing, the most recent versions available are 2.7.15 and 3.6.5. However, an official End Of Life date of January 1, 2020 has been established for Python 2, after which time it will no longer be maintained.
- Python is still maintained by a core development team at the Institute, and Guido is still in charge, having been given the title of BDFL (Benevolent Dictator For Life) by the Python community. The name Python, by the way, derives not from the snake, but from the British comedy troupe Monty Python's Flying Circus, of which Guido was, and presumably still is, a fan. It is common to find references to Monty Python sketches and movies scattered throughout the Python documentation.
- It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

- Python was designed to for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Python is Interpreted

- Many languages are compiled, meaning the source code you create needs to be translated into machine code, the language of your computer's processor, before it can be run. Programs written in an interpreted language are passed straight to an interpreter that runs them directly.
- This makes for a quicker development cycle because you just type in your code and run it, without the intermediate compilation step.
- One potential downside to interpreted languages is execution speed. Programs that are compiled into the native language of the computer processor tend to run more quickly than interpreted programs. For some applications that are particularly computationally intensive, like graphics processing or intense number crunching, this can be limiting.
- In practice, however, for most programs, the difference in execution speed is measured in milliseconds, or seconds at most, and not appreciably noticeable to a human user. The expediency of coding in an interpreted language is typically worth it for most applications.
- For all its syntactical simplicity, Python supports most constructs that would be expected in a very high-level language, including complex dynamic data types, structured and functional programming, and object-oriented programming.
- Additionally, a very extensive library of classes and functions is available that provides capability well beyond what is built into the language, such as database manipulation or GUI programming.
- Python accomplishes what many programming languages don't: the language itself is simply designed, but it is very versatile in terms of what you can accomplish with it.

3.6 Introduction to 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.

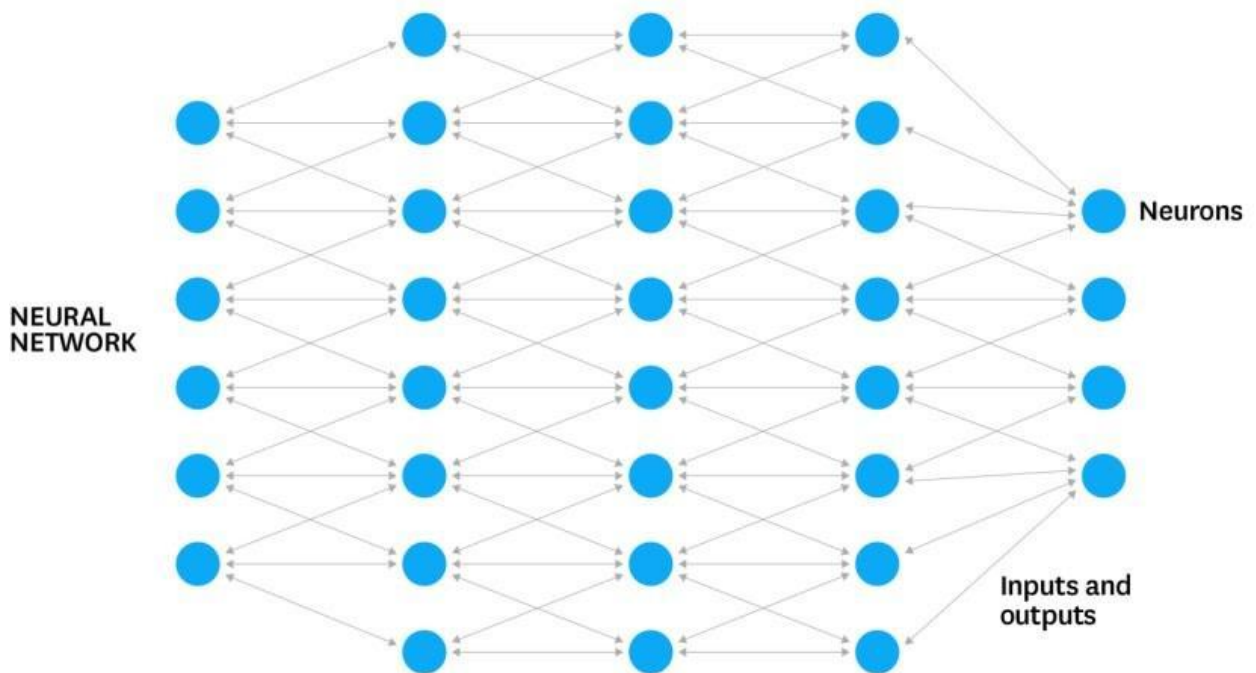
Machine Learning is an approach or subset of Artificial Intelligence that is based on the idea that machines can be given access to data along with the ability to learn from it. Deep Learning takes Machine Learning to the next level.

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).



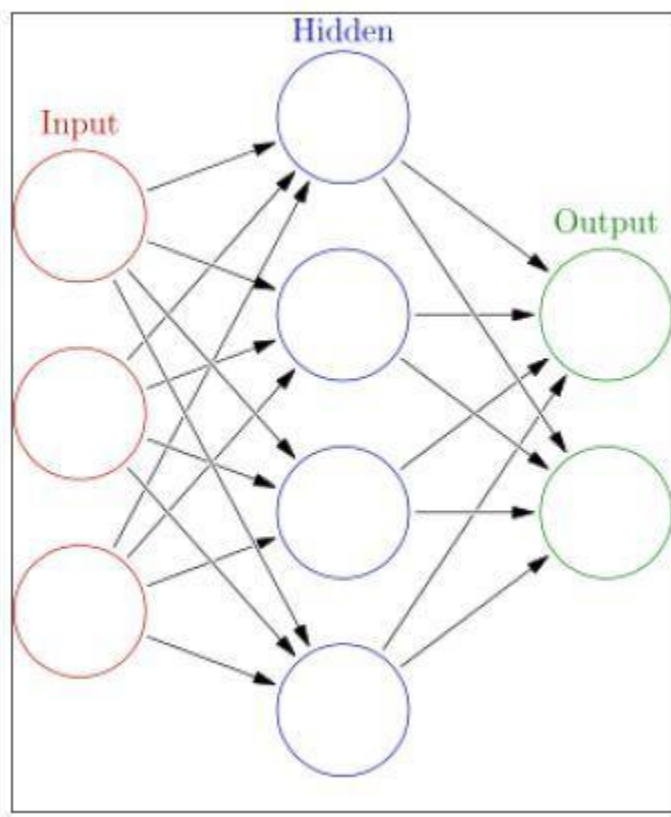
ANNs are processing devices (algorithms or actual hardware) that are modeled on the neuronal structure of the mammalian cerebral cortex but on a much smaller scale. It is a

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 tumors 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.

3.7CNN

Introduction

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.

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 49 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 26 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 60 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 69 36 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 57 05 54
01 70 54 71 83 51 54 69 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 to 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.

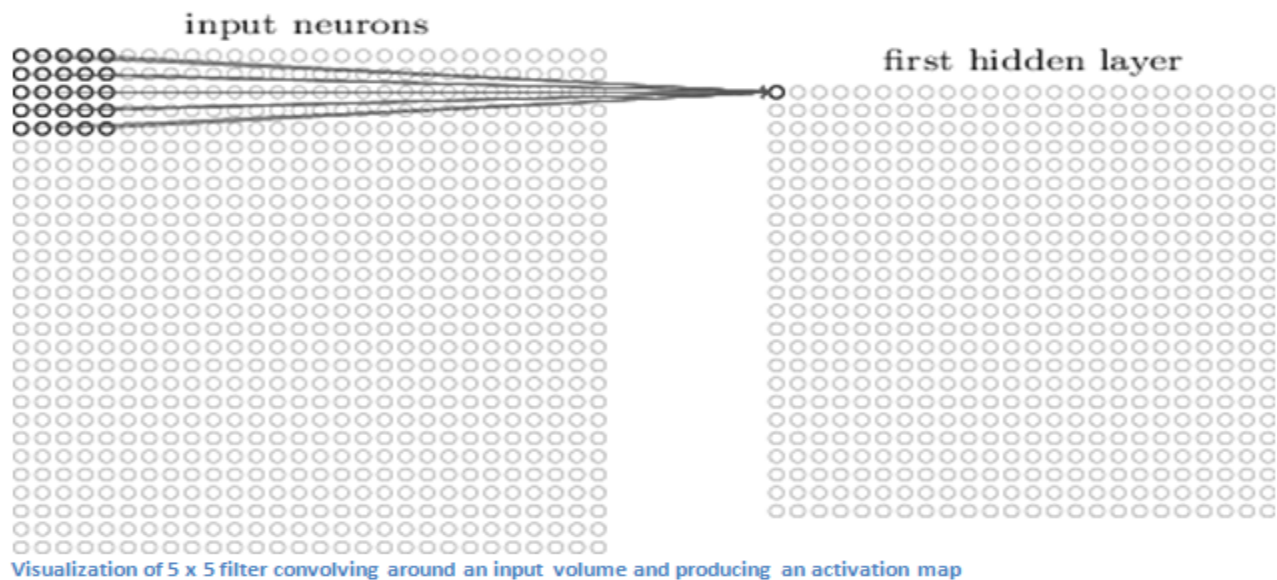
Structure

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 x 28 array.



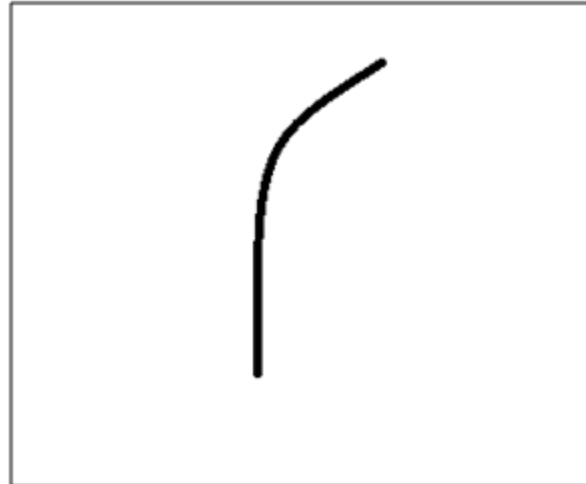
Let's say now we use two 5 x 5 x 3 filters instead of one. Then our output volume would be 28 x 28 x 2. By using more filters, we are able to preserve the spatial dimensions better. Mathematically, this is what's going on in a convolutional layer.

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 x 7 x 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

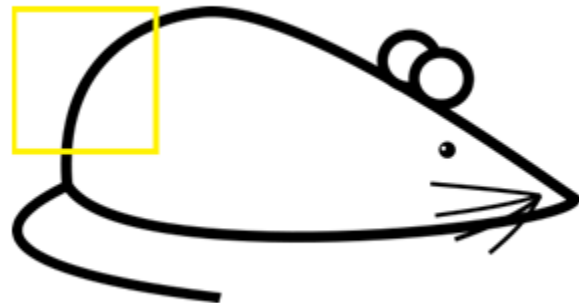


Visualization of a curve detector filter

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.



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

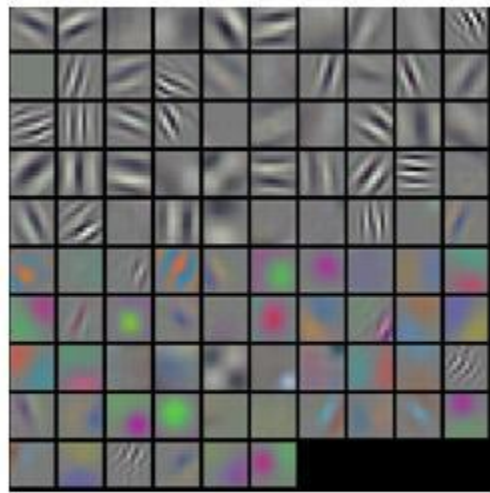
Multiplication and Summation = 0

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.



Visualizations of filters

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.

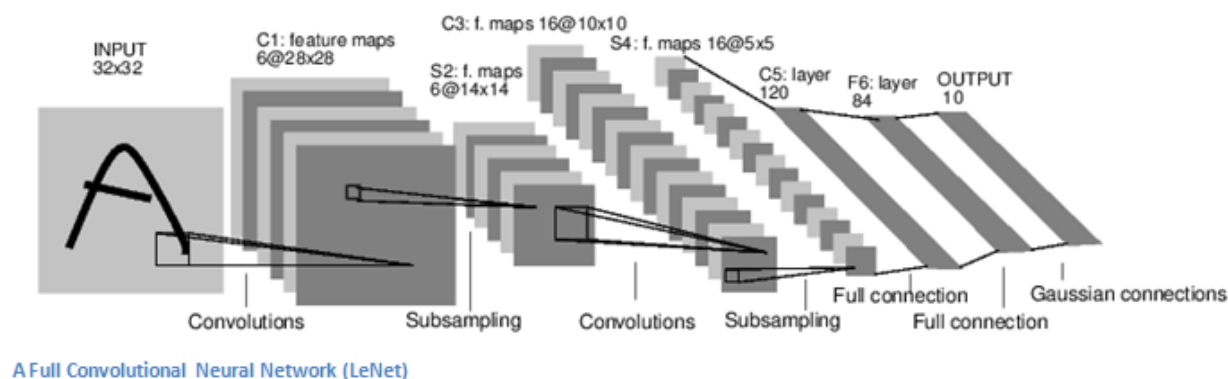
Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool -> Fully Connected

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 $[0.1 \ 0.1 \ 0.75 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0.05]$, 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 softmax 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)

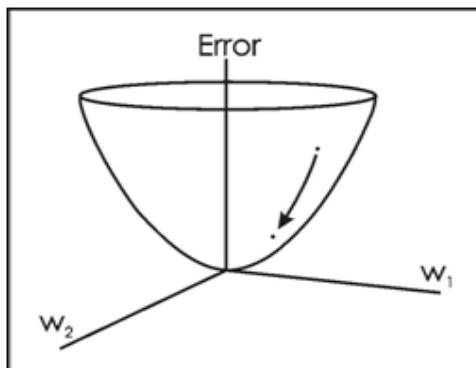
Now, this is the one aspect of neural networks that I purposely haven't mentioned yet and it is probably the most important part. There may be a lot of questions you had while reading. How do the filters in the first conv layer know to look for edges and curves? How does the fully connected layer know 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**.

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.

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.



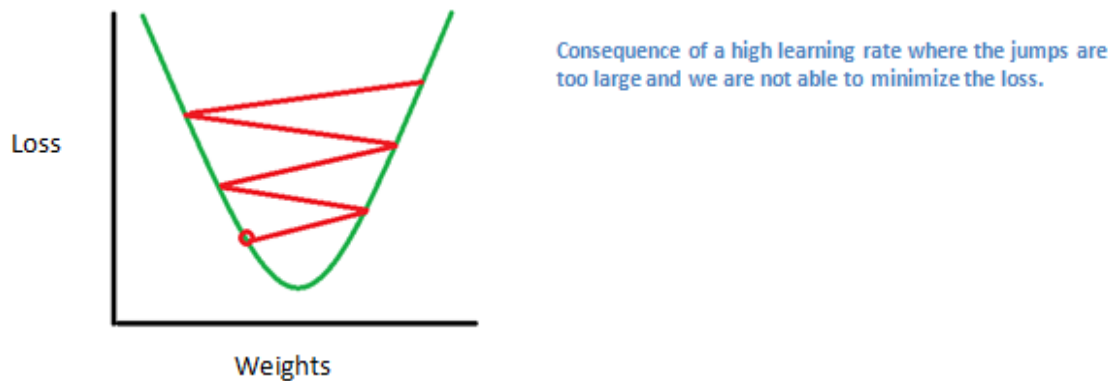
One way of visualizing this idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves 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.



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, and more were also not discussed (yet).

3.8MODULES

Dataset Creation

Frame by frame detection

Convolutional Neural Network

Detection

3.8.1 MODULE EXPLANATION:

3.8.2 Dataset Creation:

To Collect images of fire and non fire images and train both the fire and non fire images separately. Now there created a trained dataset with well determined fire images which compares with the video sequence.

Frame by frame detection

The captured images are splitted into frame by frame and the frame is again splitted into pixels with the help of cnn algorithm from the surveillance

Convolutional Neural Network

Any neural network is divided into three layers. First layer being the input layer, the second as the

hidden layer and the third layer is the output layer. Each layer consists of several nodes and this number is generally in the power of 2 to maintain the symmetry of the entire model. The nodes of one layer are connected to the nodes of another layer by means of an edge. And this edge is assigned with a weight which signifies the importance of that node in the outcome of the network. At every node, the outcome is calculated by the summation of the product of the input nodes and the weights assigned to them. And then an activation function is assigned to the summation and the outcome of the node is calculated. Activation functions include Relu, Sigmoid, etc. and based on the outcome, the appropriate activation function is chosen. This process is carried on for every node in the network and the final outcome is assigned.

When we are dealing with images, the data is generally huge and if it is fed as it is, the model becomes clumsy and the training will take a lot of time as well as the memory requirements will be huge. So, we will make use of a Convolutional neural network. The pre-processed data is fed into the input layer of the Convolutional neural network. At the time of reading the pre-processed data, a filter is applied which helps in decreasing the dimensions of the input data.

Detection:

We give the input video and the program analysis the video. If any fire detected means suddenly the value sent the firebase. The firebase send a notification to the end user.

CHAPTER 4

RESULTS AND DISCUSSION

CNN has been applied on the MNIST dataset in order to observe the variation of accuracies for handwritten digits. The accuracies are obtained using TensorFlow in python. Training and validation accuracy for 5 different epochs were observed by taking the batch size 100. For 5 hidden layers, with the batch size of 100 and 5 epochs, the minimum training accuracy was found to be 93.43% and the minimum validation accuracy was found to be 97.89%, both at epoch 1. For the same setup, the maximum training was found to be 99.77% at epoch 5 and the maximum validation accuracy was found to be 99.12% at epoch 5. During testing, the model performed with the accuracy of 99.16% over the 10000 test images.

CHAPTER 5

CONCLUSION

5.1 Conclusion

The performance of CNN for handwritten digit recognition performed significantly. The proposed method obtained 98% accuracy and is able to identify real-world images as well; the loss percentage in both training and evaluation is less than 0.1, which is negligible. The only challenging part is the noise present in the real-world image, which needs to look after. The learning rate of the model is much dependent on the number of dense neurons and the cross-validation measure. The task for handwritten digit recognition has been troublesome due to various variations in writing styles. Therefore, we have tried to create a base for future researches in the area so that the researchers can overcome the existing problems. Handwritten character recognition is one of the practically important issues in pattern recognition applications.

Timberland and steppe fires kill individuals as well as harm the climate. Throughout the most recent twenty years, huge endeavors have been made to foster tactile guides to help FMS. There are three primary strategies used to identify woods and steppe fires: the utilization of satellite information, infrared/smoke scanners, and the utilization of nearby sensors (eg meteorological). In this article, we present an insightful framework for understanding woodlands utilizing underlying data. The proposed framework utilizes imaging methods and clever innovation. Computerized readings from a distant sensor were utilized to distinguish the fire. The clever framework shown utilizes shading evolving space, anisotropic conveyance division, and outspread basal utilitarian neural organizations. The consequences of the review featured the presentation of astute frameworks in the quest for woodland and steppe fires utilizing pertinent data.

REFERENCES:

- [1] M. Maier, M. Chowdhury, B. P. Rimal, and D. P. Van, "The tactile internet: Vision, recent progress, and open challenges," *IEEE Commun. Mag.*, vol. 54, no. 5, pp. 138–145, May 2016.
- [2] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "5G-enabled tactile internet," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 4, pp. 460–473, Mar. 2016.
- [3] K. Muhammad, R. Hamza, J. Ahmad, J. Lloret, H. H. G.Wang, and S.W. Baik, "Secure surveillance framework for IoT systems using probabilistic image encryption," *IEEE Trans. Ind. Inform.*, vol. 14, no. 8, pp. 3679–3689, Aug. 2018.
- [4] N. Pogkas, G. E. Karastergios, C. P. Antonopoulos, S. Koubias, and G. Papadopoulos, "Architecture design and implementation of an ad-hoc network for disaster relief operations," *IEEE Trans. Ind. Inform.*, vol. 3, no. 1, pp. 63–72, Feb. 2007.
- [5] A. Filonenko, D. C. Hern´andez, and K.-H. Jo, "Fast smoke detection for video surveillance using CUDA," *IEEE Trans. Ind. Inform.*, vol. 14, no. 4, pp. 725–733, Feb. 2018.
- [6] U. L. Puvvadi, K. Di Benedetto, A. Patil, K.-D. Kang, and Y. Park, "Costeffective security support in real-time video surveillance," *IEEE Trans. Ind. Inform.*, vol. 11, no. 6, pp. 1457–1465, Dec. 2015. \
- [7] S. Mumtaz, A. Bo, A. Al-Dulaimi, and K.-F. Tsang, "5G and beyond mobile technologies and applications for industrial IoT (IIoT)," *IEEE Trans. Ind. Inform.*, vol. 14, no. 6, pp. 2588–2591, Jun. 2018.
- [8] D. Guha-Sapir and P. Hoyois, "Estimating populations affected by disasters: A review of methodological issues and research gaps," *Centre Res. Epidemiol. Disast., Inst. Health Soc., Univ. Catholique de Louvain, Brussels*, 54 Belgium, 2015.
- [9] P. Foggia, A. Saggese, and M. Vento, "Real-time fire detection for videosurveillance applications using a combination of experts based on color, shape, and motion," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 9, pp. 1545–1556, Sep. 2015.
- [10] B. U. Toreyin, Y. Dedeoglu, U. Gudukbay, and A. E. Cetin, "Computer vision basedmethod for real-time fire and flame detection," *Pattern Recognit. Lett.*, vol. 27, pp. 49–58, 2006.
- [11] D. Han and B. Lee, "Development of early tunnel fire detection algorithm using the image processing,"

in Proc. Int. Symp. Vis. Comput., 2006, pp. 39–48.

[12] T.-H. Chen, P.-H. Wu, and Y.-C. Chiou, “An early fire-detection method based on image processing,” in Proc. Int. Conf. Image Process., 2004, pp. 1707–1710.

[13] G. Marbach, M. Loepfe, and T. Brupbacher, “An image processing technique for fire detection in video images,” *Fire Saf. J.*, vol. 41, pp. 285–289, 2006.

[14] T. Celik and H. Demirel, “Fire detection in video sequences using a generic color model,” *Fire Saf. J.*, vol. 44, pp. 147–158, 2009.

[15] A. Rafiee, R. Dianat, M. Jamshidi, R. Tavakoli, and S. Abbaspour, “Fire and smoke detection using wavelet analysis and disorder characteristics,” in Proc. 3rd Int. Conf. Comput. Res. Develop., 2011, pp. 262–265.

APPENDICES

A)SOURCE CODE:

#INSTALL TENSORFLOW

```
!pip install tensorflow==2.0
```

#IMPORT TENSORFLOW

```
import tensorflow as tf
```

#LOADING "MNIST DATA SET"

```
mnist=tf.keras.datasets.mnist
```

#AFTER LOADING THE MNIST DATA,DIVIDE INTO TRAIN AND TEST DATASETS

```
(x_train,y_train),(x_test,y_test)=mnist.load_data()
```

Output

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz  
11493376/11490434 [=====] - 0s 0us/step
```

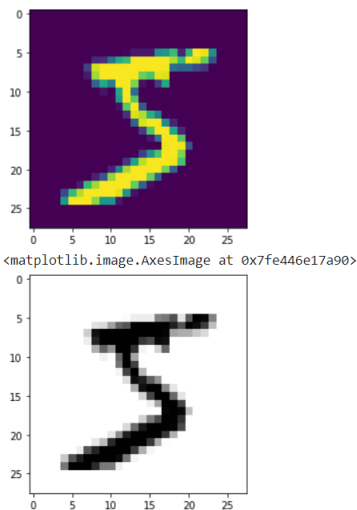
#INSTALL MATPLOTLIB

```
!pip install matplotlib
```

#CHECKING THE GRAPH, HOW DATA LOOKS LIKE

```
import matplotlib.pyplot as plt  
plt.imshow(x_train[0])  
plt.show()#in order to execute the graph  
#however we dont know whethe is color image or binary images  
#so inorder to plot it change the configuration  
plt.imshow(x_train[0],cmap=plt.cm.binary)
```

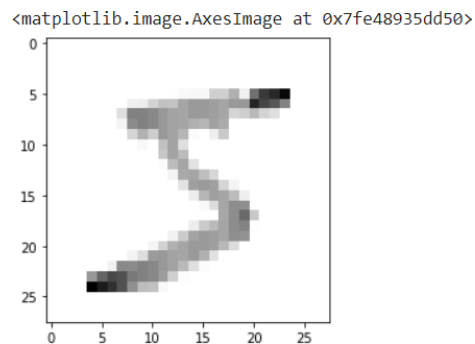
Output



#NORMALIZING THE DATA

```
x_train=tf.keras.utils.normalize(x_train,axis=1)
x_test=tf.keras.utils.normalize(x_test,axis=1)
plt.imshow(x_train[0],cmap=plt.cm.binary)
```

Output



#INSTALL NUMPY

```
!pip install numpy
```

#RESIZING IMAGE TO MAKE IT SUITABLE FOR APPLY CNN

```
import numpy as np
IMG_SIZE=28
x_trainr=np.array(x_train).reshape(-1,IMG_SIZE,IMG_SIZE,1)
#increasing one dimension for kernel operation
x_testr=np.array(x_test).reshape(-1,IMG_SIZE,IMG_SIZE,1)
#increasing one dimension for kernel operation
print("Training samples dimesion",x_trainr.shape)
print("Testing samples dimension",x_testr.shape)
```

Output

```
Training samples dimension (60000, 28, 28, 1)
Testing samples dimension (10000, 28, 28, 1)
```

#CREATING A DEEP NEURAL NETWORK

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense , Dropout , Activation , Flatten , Conv2D , MaxPooling2D

# Creating a neural network now
model= Sequential()
model.add(Conv2D(64, (3,3), input_shape= x_trainr.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
# 2nd Convolution Layer 26-3+1= 24*24
model.add(Conv2D(64, (3,3))) # 2nd Convolution Layer
model.add(Activation("relu")) # activation funtion
model.add(MaxPooling2D(pool_size=(2, 2))) #MAXpooling
# 3rd Convolution Layer
model.add(Conv2D(64, (3,3))) # 24*24
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
# Fully Connected Layer # 1 20x20= 400
model.add (Flatten())
model.add (Dense(64))
model.add(Activation("relu"))
# Fully Connected Layer # 2
model.add (Dense(32))
model.add(Activation("relu"))
model.add(Dense(10))
model.add(Activation( 'softmax'))
```

#MODEL SUMMARY

```
model.summary()
```

Output

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 64)	640
activation (Activation)	(None, 26, 26, 64)	0
max_pooling2d (MaxPooling2D)	(None, 13, 13, 64)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	36928
activation_1 (Activation)	(None, 11, 11, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36928
activation_2 (Activation)	(None, 3, 3, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 1, 1, 64)	0
flatten (Flatten)	(None, 64)	0
dense (Dense)	(None, 64)	4160
activation_3 (Activation)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2080
activation_4 (Activation)	(None, 32)	0
dense_2 (Dense)	(None, 10)	330
activation_5 (Activation)	(None, 10)	0
Total params: 81,066		
Trainable params: 81,066		
Non-trainable params: 0		

#COMPILING MODEL

```
model.compile(loss="sparse_categorical_crossentropy",optimizer="adam",metrics=['accuracy'])
```

#TRAINING MODEL

```
model.fit(x_train,y_train,epochs=5,validation_split=0.3)
```

Output

```
Epoch 1/5
1313/1313 [=====] - 67s 50ms/step - loss: 0.3322 - accuracy: 0.8940 - val_loss: 0.1176 - val_accuracy: 0.9647
Epoch 2/5
1313/1313 [=====] - 66s 50ms/step - loss: 0.1059 - accuracy: 0.9680 - val_loss: 0.1000 - val_accuracy: 0.9696
Epoch 3/5
1313/1313 [=====] - 66s 50ms/step - loss: 0.0764 - accuracy: 0.9763 - val_loss: 0.0706 - val_accuracy: 0.9780
Epoch 4/5
1313/1313 [=====] - 66s 50ms/step - loss: 0.0584 - accuracy: 0.9822 - val_loss: 0.0615 - val_accuracy: 0.9813
Epoch 5/5
1313/1313 [=====] - 66s 50ms/step - loss: 0.0470 - accuracy: 0.9853 - val_loss: 0.0658 - val_accuracy: 0.9803
<tensorflow.python.keras.callbacks.History at 0x7fe441c05ed0>
```

#EVALUATING ON TESTING DATA SET MNIST

```
test_loss,test_acc=model.evaluate(x_testr,y_test)
print("Test loss on 10,000 test samples",test_loss)
print("validation accuracy on 10,000 test samples",test_acc)
```

Output

```
313/313 [=====] - 4s 13ms/step - loss: 0.0579 - accuracy: 0.9838
Test loss on 10,000 test samples 0.05786069482564926
validation accuracy on 10,000 test samples 0.9837999939918518
```

#PREDICTIONS

```
predictions=model.predict([x_testr])
```

```
print(predictions)
```

Output

```
[[1.8952541e-08 2.0202178e-06 1.7596547e-05 ... 9.9997365e-01
 1.8573429e-07 3.4380107e-06]
 [5.2573041e-06 4.0824478e-05 9.9994671e-01 ... 6.6016555e-06
 2.0302537e-07 3.7041996e-08]
 [6.4439315e-10 1.0000000e+00 1.0974468e-09 ... 3.8937877e-08
 6.5729480e-09 5.0442797e-09]
 ...
 [4.9119153e-10 1.2636329e-07 1.2462887e-09 ... 1.3397519e-07
 1.3367534e-06 2.2533300e-06]
 [6.3650325e-05 5.4306719e-09 6.6939890e-08 ... 3.1545380e-06
 3.5937030e-03 7.1285802e-05]
 [6.0746851e-07 5.1018116e-08 5.8588762e-06 ... 1.7740145e-08
 1.0736002e-07 1.5893772e-07]]
```

#INSTALL OPEN CV-PYTHON

```
!pip install opencv-python
```

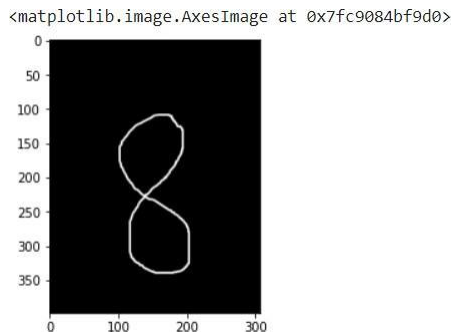
#IMPORT CV2

```
import cv2
```

#LOADING IMAGE

```
img = cv2.imread("/content/drive/MyDrive/digits/eight.png")
```

#SHOW IMAGE



#RESIZING IMAGE

```
img.shape
```

```
(345, 391, 3)
```

```
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
gray.shape
```

```
(345, 391)
```

```
resized=cv2.resize(gray,(28,28),interpolation=cv2.INTER_AREA)
```

```
resized.shape
```

```
(28, 28)
```

```
newimg=tf.keras.utils.normalize(resized,axis=1)#0 to 1 scaling
```

```
newimg=np.array(newimg).reshape(-1,IMG_SIZE,IMG_SIZE,1)
```

```
newimg.shape
```

```
(1, 28, 28, 1)
```

#RECOGNITION AND OUTPUT GENERATION

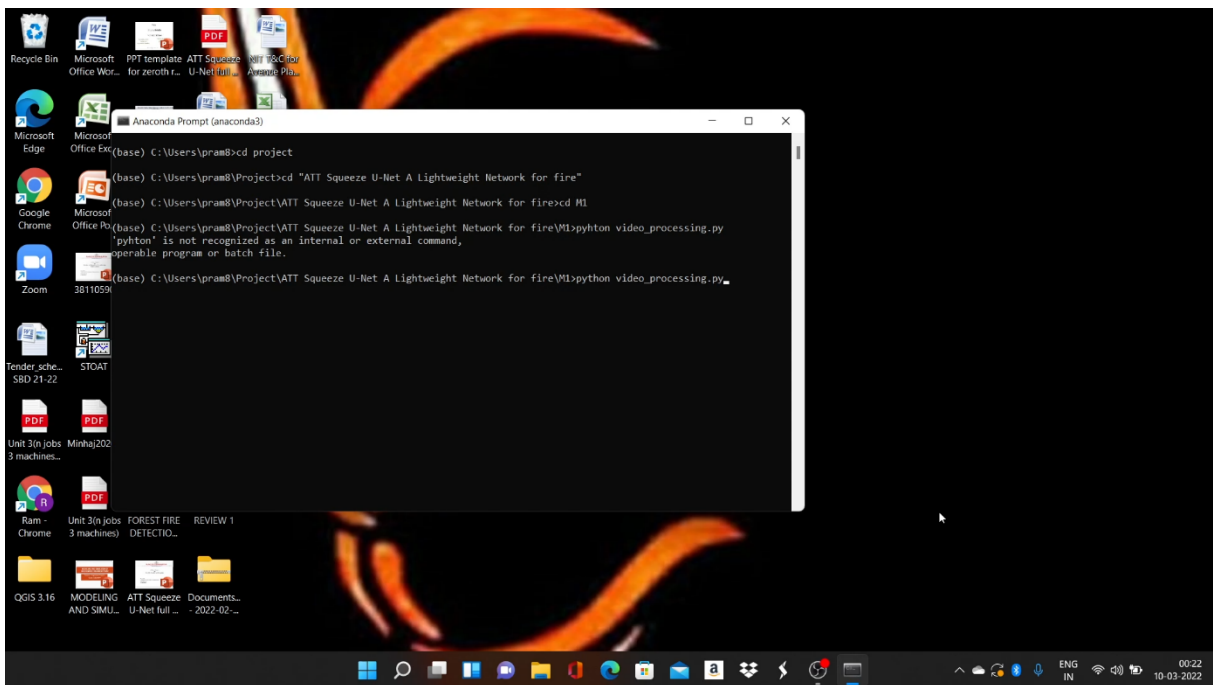
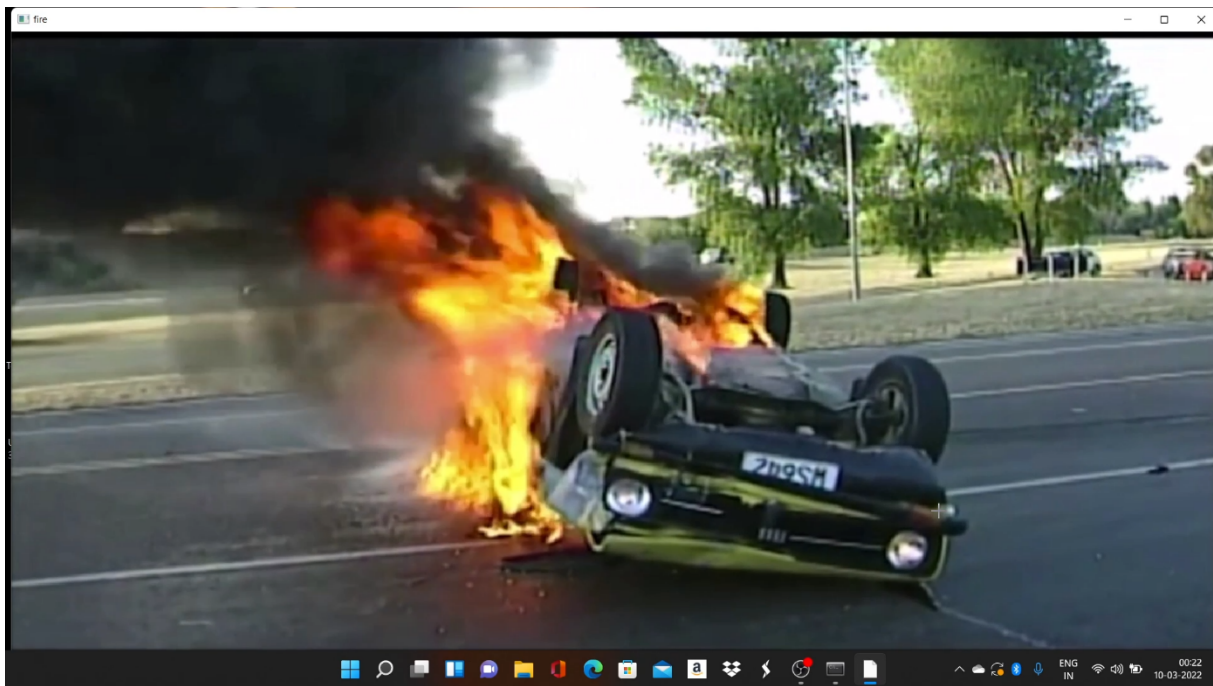
```
predictions=model.predict(newimg)
```

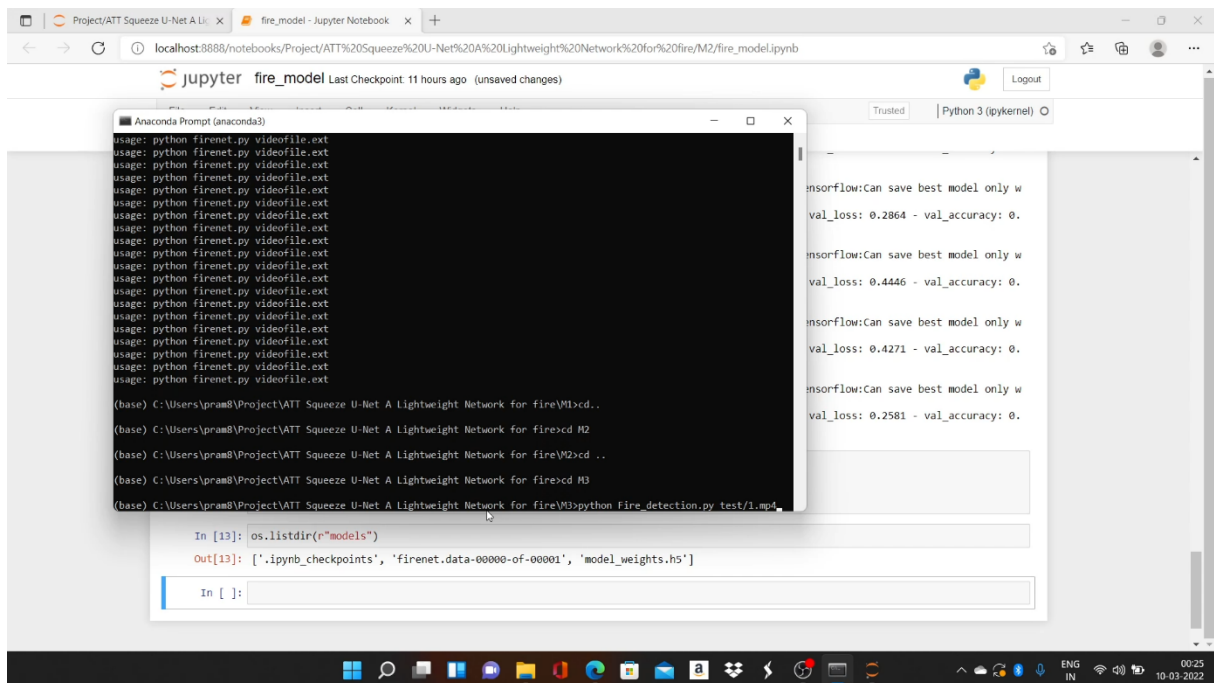
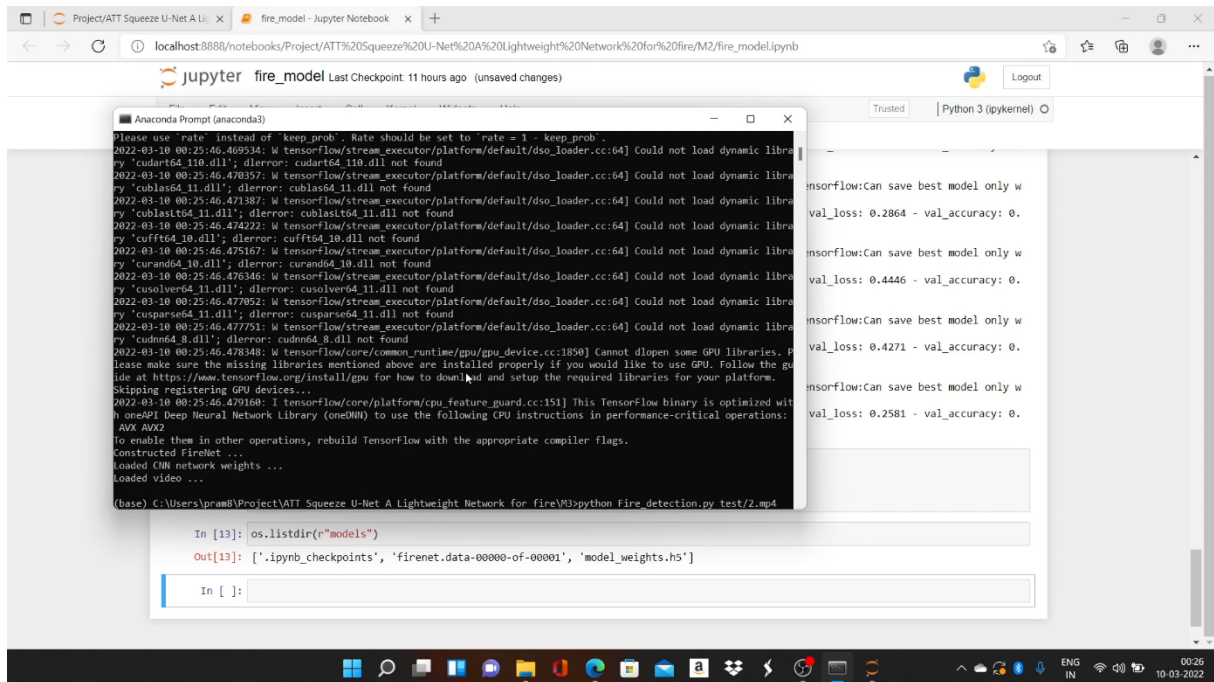
```
print(np.argmax(predictions))
```

Output

8

B)SCREENSHOTS:





Project/ATT Squeeze U-Net A Li x fire_model - Jupyter Notebook x +

localhost:8888/notebooks/Project/ATT%20Squeeze%20U-Net%20A%20Lightweight%20Network%20for%20fire/M2/fire_model.ipynb

jupyter fire_model Last Checkpoint 11 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Kernel starting, please wait... Saving every 120 sec. Trusted Python 3 (ipykernel)

Run

Import libraries

```
In [ ]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
```

```
In [ ]: from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import itertools
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix
import itertools
```

Store every image in an array

```
In [ ]: import os
data_dir = (r'.\fire_dataset')
categories = ['non_fire_images', 'fire_images']
for i in categories:
    path = os.path.join(data_dir, i)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path, img))
```

Loading [MathJax]/jax/outputHTML-CSS/fonts/STIX-Web/fontdata.js

00:22 10-03-2022

Project/ATT Squeeze U-Net A Li x +

localhost:8888/tree/Project/ATT%20Squeeze%20U-Net%20A%20Lightweight%20Network%20for%20fire/M2

jupyter Quit Logout

Files Running Clusters

Select items to perform actions on them. Upload New

	Name	Last Modified	File size
	..	seconds ago	
	fire_dataset	11 hours ago	
	models	11 hours ago	
	fire_model.ipynb	4 minutes ago	8.86 kB

00:22 10-03-2022