Industrial Major Project Report

On

Flooded Area Detector

Submitted in partial fulfillment of the

Requirements for the award of the degree of Bachelor of Technology

In

Computer Science and Engineering

By

Siddi Sujith	**15241A05T1**
Nallana Harish	**16245A0546**
Shashi Preetham G	**15241A05P9**
Naveen Naik	**16245A0552**

Under the Guidance of

Dr. Ch. Mallikarjuna Rao

Prof, Dept. of CSE.

Department of Computer Science and Engineering

GOKARAJU RANGARAJU

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous under JNTUH, Hyderabad)

Bachupally, Kukatpally, Hyderabad-500090.

Department of Computer Science and Engineering

GOKARAJU RANGARAJU

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous under JNTUH, Hyderabad)

Bachupally, Kukatpally, Hyderabad-500090.

CERTIFICATE

This is to certify that the Industrial major project entitled **"Flooded Area Detector"** is submitted by **Siddi Sujith (15241A05T1), Nallana Harish (16245A0546), Shashi Preetham G (15241A05P9), Naveen Naik (16245A0552)** in partial fulfillment of the requirement for the award of the degree in **BACHELOR OF TECHNOLOGY** in Computer Science and Engineering during the academic year 2018-2019.

Internal Guide
Dr. Ch. Mallikarjuna Rao

Head of the Department
Dr. K. Madhavi

External Examiner

ACKNOWLEDGEMENT

There are many people who helped us directly and indirectly to complete our project successfully. We would like to take this opportunity to thank one and all.

First of all we would like to express our deep gratitude towards our internal guide **Dr. Ch. Mallikarjuna Rao,** Department of CSE for his support in the completion of my dissertation. We wish to express our sincere thanks to **Dr. K. Madhavi, HOD, Department of CSE** and also to our principal **Dr. J. Praveen** for providing the facilities to complete the dissertation.

We would like to thank all our faculty and friends for their help and constructive criticism during the project period. Finally we are very much indebted to our parents for their moral support and encouragement to achieve our goals.

Siddi Sujith (15241A05T1)
Nallana Harish (16245A0546)
Shashi Preetham G (15241A05P9)
Naveen Naik (16245A0552)

DECLARATION

I hereby declare that the industrial major project entitled **"Flooded Area Detector"** is the work done during the period from 10-12-2018 to 05-04-2019 and is submitted in the partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering from Gokaraju Rangaraju Institute of Engineering and Technology (Autonomous under Jawaharlal Nehru Technology University, Hyderabad).

The results embodied in this project have not been submitted to any other University or Institution for the award of any degree or diploma.

Siddi Sujith (15241A05T1)
Nallana Harish (16245A0546)
Shashi Preetham G (15241A05P9)
Naveen Naik (16245A0552)

ABSTRACT

DESCRIPTION:

With the evolution in machine learning, many problems are being solved using various deep learning algorithms . One such problem is accurately identifying and classifying the flooded areas and non-flooded areas using less computational power. With deep learning, we can classify the images effectively using the aerial images collected. The aerial images can be collected easily with the help of a drone. This work mainly involves usage of Convolutional Neural Network (CNN) which is a feed-forward network. This network uses supervised learning technique and multi-layer perceptron which is a deep learning algorithm. This algorithm mimics the functions of human brain and thus enables the network to learn using the inputs given. During this learning process, the errors and the performance are measured using the losses and metrics.

CONTENTS Page No

1.INTRODUCTION

1.1 DESCRIPTION:

This project is based on Data Science. We will be using the datasets of images of various flooded areas and non-flooded areas. According to the images involved in the dataset, the model is created. This project helps in detecting the flooded areas by image classification . The classification is done by using training sets and test sets of the images data. The image classifier will classify the images based on the pixels of the images. The model can be trained using the trained set and tested using the test set. Thus, when there is a flood in a place, drones can be used to take images of the city or state and the real-time images can be sent to the computer by which detection of flooded areas can be made.

By this, flooded areas can be accurately identified and necessary relief can be sent to those areas by various organizations on time .This project involves usage of tools such that the tool is capable of data analysis and visualization, computer with a high end graphic card and good amount of RAM.

1.2 OVERVIEW:

The flooded area detection mainly focuses on the classification of images of floods to those of non-flooded images. This can be done using convolutional neural networks which is a feed-forward artificial neural network. This uses supervised learning with back-propagation as its technique. When the images of both the flooded and non-flooded such as roads, different types of terrains are given to the model, it transforms the images into matrices of pixels and the computation is done on those pixels. By this, accuracy can be achieved because of the coverage of all the pixels from the images and thus successfully classify the images. This model can be helpful when an organization can send a drone for surveillance and the captured images can then be classified in real-time and thus be notified of the floods and appropriate action can be taken without any time delay.

1.3 HOW OUR SYSTEM WORKS?

The main goal of this project is to provide service to various organizations which tend to work for disaster management. Using various machine learning libraries such as keras and tensorflow as its backend, the deep learning model created will help the organizations to effectively provide help to those particular areas on time. The accuracy of the model is around 80% which is a good accuracy percentage and thus is effective to solve the problems occurred during flood times. The images will be collected from drones which the organizations will be using and continuous classification and prediction of flooded areas can be done so that timely help can be provided without any manual intervention for the surveillance of flooded areas.

2.SYSTEM ANALYSIS

2.1 EXISTING SYSTEM:

- Previously , when floods would occur, manual work had to be done to detect and spot flooded areas.
- There were no sophisticated tools for timely surveillance of flood prone areas.
- Some of the models are present which classify based only on the satellite images of the satellite due to which the accuracy is less and the satellite organizations do not provide permissions to everyone to access them. It also requires high computational power.

2.2 PROPOSED SYSTEM:

- This system provides real-time classification of the images and the classification is done not on the spectral data but using the real-images captured by the drones.
- Thus, the accuracy in classifying and predicting the flooded-areas images will be more and it can be easily used by various service organizations.
- The computational power used is also very less compared to the satellites computational power usage. A single computer with a better GPU and CPU can perform the classification faster and effectively.

2.3 SOFTWARE/HARDWARE REQUIREMENTS:

Software:

- Anaconda IDE
- Tensorflow and keras libraries(Machine learning).
- Numpy and pandas libraries(Python).
- Images data set for training and testing.

Hardware:

- A computer with a minimum configuration of Intel i5 processor along with Nvidia 960 MX GPU is preferred for better processing.
- A GPU with CUDA support for parallel processing.
- Minimum 4GB of RAM.
- Minimum 4GB of graphic RAM.

2.4 SYSTEM ARCHITECTURE:

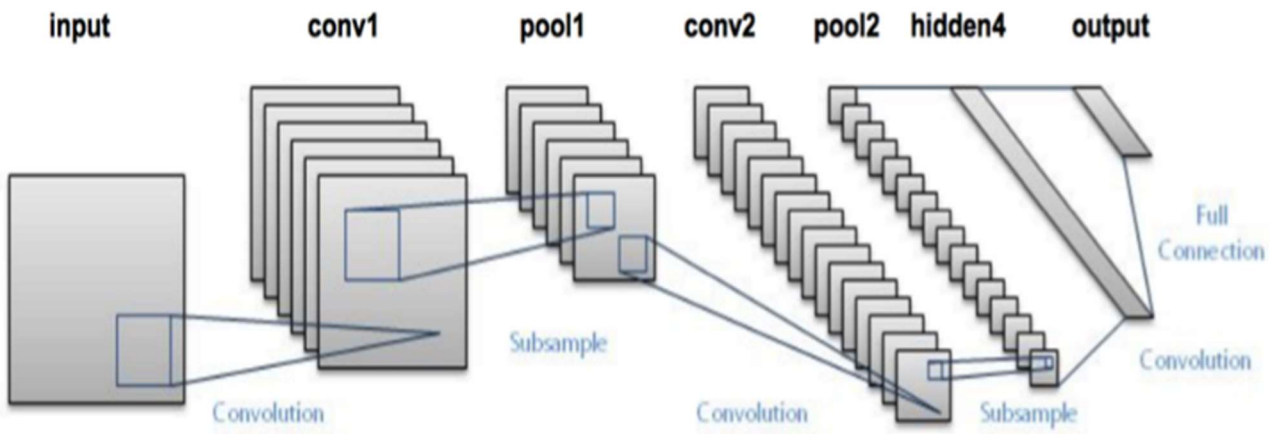


Fig 2.1 System Architecture.

2.5 MODULES AND ITS DESCRIPTION:

Tensorflow:

Created by the Google Brain team, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++.

How TensorFlow works:

TensorFlow allows developers to create *dataflow graphs*—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or *tensor*.

TensorFlow provides all of this for the programmer by way of the Python language. Python is easy to learn and work with, and provides convenient ways to express how high-level abstractions can be coupled together. Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications.

The actual math operations, however, are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries. Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together.

TensorFlow applications can be run on most any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.

TensorFlow benefits:

The single biggest benefit TensorFlow provides for machine learning development is *abstraction.* Instead of dealing with the nitty-gritty details of implementing algorithms, or figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application. TensorFlow takes care of the details behind the scenes.

TensorFlow offers additional conveniences for developers who need to debug and gain introspection into TensorFlow apps. The eager execution mode lets you evaluate and modify each graph operation separately and transparently, instead of constructing the entire graph as a single opaque object and evaluating it all at once. The TensorBoard visualization suite lets you inspect and profile the way graphs run by way of an interactive, web-based dashboard.

And of course TensorFlow gains many advantages from the backing of an A-list commercial outfit in Google. Google has not only fueled the rapid pace of development behind the project, but created many significant offerings around TensorFlow that make it easier to deploy and easier to use: the above-mentioned TPU silicon for accelerated performance in Google's cloud; an online hub for sharing models created with the framework; in-browser and mobile-friendly incarnations of the framework; and much more.

One caveat: Some details of TensorFlow's implementation make it hard to obtain totally deterministic model-training results for some training jobs. Sometimes a model trained on one system will vary slightly from a model trained on another, even when they are fed the exact same data. The reasons for this are slippery—e.g., how random numbers are seeded and where, or certain non-deterministic behaviors when using GPUs). That said, it is possible to work around those issues, and TensorFlow's team is considering more controls to affect determinism in a workflow.

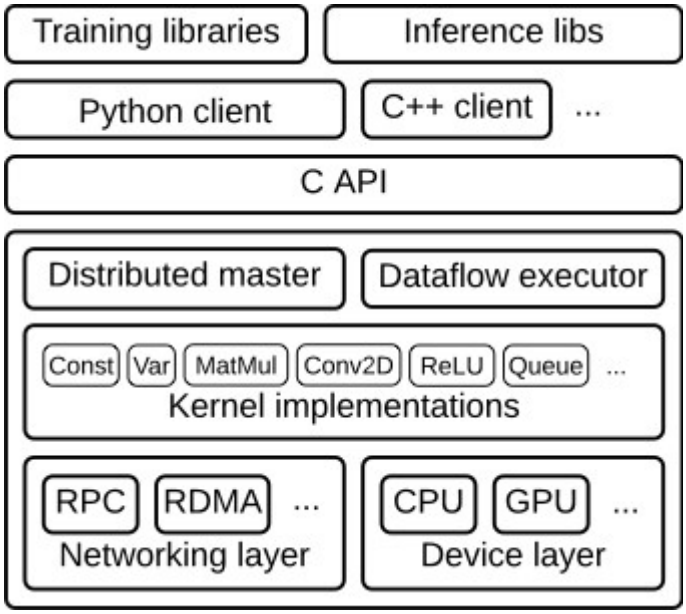


Fig 2.2 Tensorflow architecture.

Keras:

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Guiding principles:

- **User friendliness.** Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
- **Modularity.** A model is understood as a sequence or a graph of standalone, fully configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions and regularization schemes are all standalone modules that you can combine to create new models.
- **Easy extensibility.** New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.
- **Work with Python**. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

Convolutional Neural Networks:

- A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layer, pooling layer, flattening layer, fully connected layer and loss layer.
- The process of convolution in the convolutional neural networks is by cross-correlation.

Cross Correlation:

Cross-correlation is a measure of similarity between two series or features measured as a function.

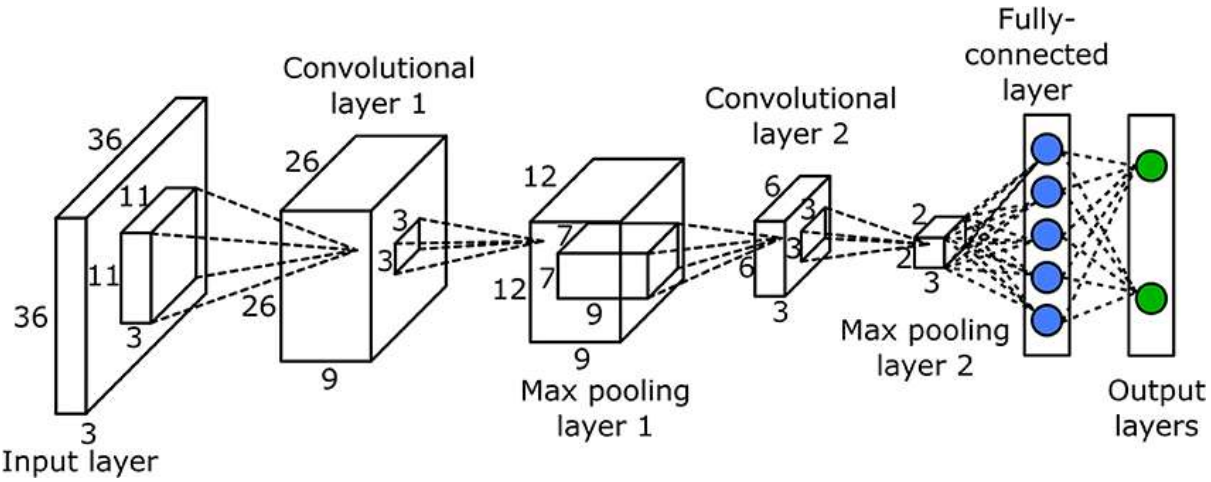


Fig 2.3 Convolutional Neural Network architecture.

Convolution:

- Convolutional layers apply a convolution operation to the input, passing the result to the next layer.
- Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features as well as classify data, it would require many resources and more energy would be required for processing. For instance, a fully connected layer for an image of size 100 x 100 has 10000 weights for *each* neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of parameters, allowing the network to be deeper with fewer parameters.

Pooling:

- In this layer, the outputs of neuron clusters at one layer are combined and passed onto a single neuron in the next layer.

Types of Pooling:

- Maximum pooling
- Average pooling

Receptive Field:

In a fully connected layer, each neuron receives input from *every* element of the previous layer. In a convolutional layer, neurons receive input from only a restricted subarea of the previous layer. So, in a fully connected layer, the receptive field is the entire previous layer. In a convolutional layer, the receptive area is smaller than the entire previous layer.

Weights:

- Each neuron in a neural network computes an output value by applying some function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is specified by a vector of weights. Learning in a neural network progresses by making incremental adjustments to the weights. The vector of weights is called a *filter*. A distinguishing feature of CNNs is that many neurons share the same filter. This reduces memory footprint because a single vector of weights is used across all receptive fields sharing that filter, rather than each receptive field having its own vector of weights.

Activation:

- Activation functions are used to bring non-linear properties to the network.
- Generally, without an activation function, the output would be a linear function. The drawback with linear function is that it is not curvilinear and thus reduces accuracy of the network. The linear function has less power to learn from the network.
- Whereas, activation functions introduce non-linear properties to the output of each layer and increase the learning power of the layers in the network.

- Different activations functions are present such as the sigmoid function, ReLu function and Softmax function.

os-miscellaneous operating system interface:

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see open(), if you want to manipulate paths, see the os.path module, and if you want to read all the lines in all the files on the command line see the fileinput module. For creating temporary files and directories see the tempfile module, and for high-level file and directory handling see the shutil module.

Notes on the availability of these functions:

- The design of all built-in operating system dependent modules of Python is such that as long as the same functionality is available, it uses the same interface; for example, the function os.stat(path) returns stat information about *path* in the same format (which happens to have originated with the POSIX interface).
- Extensions peculiar to a particular operating system are also available through the os module, but using them is of course a threat to portability.
- All functions accepting path or file names accept both bytes and string objects, and result in an object of the same type, if a path or file name is returned.

Shutil-high level file operations:

The shutil module offers a number of high-level operations on files and collections of files. In particular, functions are provided which support file copying and removal.

Numpy:

Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

Traits:

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

Pandas:

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term " panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

Matplotlib:

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There

is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

Matplotlib was originally written by John D. Hunter, has an active development community, and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012, and further joined by Thomas Caswell.

As of 23 June 2017, matplotlib 2.0.x supports Python versions 2.7 through 3.6. Matplotlib 1.2 is the first version of matplotlib to support Python 3.x. Matplotlib 1.4 is the last version of Matplotlib to support Python 2.6.

Matplotlib has pledged to not support Python 2 past 2020 by signing the Python 3 Statement.

Stochastic Gradient Descent:

Stochastic gradient descent (often shortened to SGD), also known as incremental gradient descent, is an iterative method for optimizing a differentiable objective function, a stochastic approximation of gradient descent optimization. A 2018 article implicitly credits Herbert Robbins and Sutton Monro for developing SGD in their 1951 article titled "A Stochastic Approximation Method"; see Stochastic approximation for more information. It is called stochastic because samples are selected randomly (or shuffled) instead of as a single group (as in standard gradient descent) or in the order they appear in the training set.

Extensions and variants:

Many improvements on the basic stochastic gradient descent algorithm have been proposed and used. In particular, in machine learning, the need to set a learning rate (step size) has been recognized as problematic. Setting this parameter too high can cause the algorithm to diverge; setting it too low makes it slow to converge. A conceptually simple extension of stochastic gradient descent makes the learning rate a decreasing function η_t of the iteration number t, giving a learning rate schedule, so that the first iterations cause large changes in the parameters, while the later ones do only fine-tuning. Such schedules have been known since the work of MacQueen on k-means clustering.

Implicit Updates:

As mentioned earlier, classical stochastic gradient descent is generally sensitive to learning rate η. Fast convergence requires large learning rates but this may induce numerical instability. The problem can be largely solved by considering *implicit updates* whereby the stochastic gradient is evaluated at the next iterate rather than the current one:

Momentum:

Further proposals include the momentum method, which appeared in Rumelhart, Hinton and Williams' seminal paper on backpropagation learning. Stochastic gradient descent with momentum remembers the update Δw at each iteration, and determines the next update as a linear combination of the gradient and the previous update:

Averaging:

Averaged stochastic gradient descent, invented independently by Ruppert and Polyak in the late 1980s, is ordinary stochastic gradient descent that records an average of its parameter vector over time. That is, the update is the same as for ordinary stochastic gradient descent, but the algorithm also keeps track of updates.

AdaGrad:

AdaGrad (for adaptive gradient algorithm) is a modified stochastic gradient descent with per-parameter learning rate, first published in 2011. Informally, this increases the learning rate for more sparse parameters and decreases the learning rate for fewer sparse ones. This strategy often improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse parameters are more informative. Examples of such applications include natural language processing and image recognition. It still has a base learning rate η, but this is multiplied with the elements of a vector $\{G_{j,j}\}$ which is the diagonal of the outer product matrix.

RMSProp:

RMSProp (for Root Mean Square Propagation) is also a method in which the learning rate is adapted for each of the parameters. The idea is to divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight. So, first the running average is calculated in terms of means square.

Adam:

Adam (short for Adaptive Moment Estimation) is an update to the RMSProp optimizer. In this optimization algorithm, running averages of both the gradients and the second moments of the gradients are used.

Natural Gradient Descent and kSGD:

Kalman-based Stochastic Gradient Descent (kSGD) is an online and offline algorithm for learning parameters from statistical problems from quasi-likelihood models, which include linear models, non-linear models, generalized linear models, and neural networks with squared error loss as special cases. For online learning problems, kSGD is a special case of the Kalman Filter for linear regression problems, a special case of the Extended Kalman Filter for non-linear regression problems, and can be viewed as an incremental Gauss-Newton method. Moreover, because of kSGD's relationship to the Kalman Filter and natural gradient descent's relationship to the Kalman Filter, kSGD is a rigorous improvement over the popular natural gradient descent method.

The benefits of kSGD, in comparison to other methods, are (1) it is not sensitive to the condition number of the problem , (2) it has a robust choice of hyperparameters, and (3) it has a stopping condition. The drawbacks of kSGD is that the algorithm requires storing a dense covariance matrix between iterations, and requires a matrix-vector product at each iteration.

Backpropagation:

Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. Backpropagation is shorthand for "the backward propagation of errors," since an error is computed at the output and distributed backwards throughout the network’s layers. It is commonly used to train deep neural networks.

Backpropagation is a generalization of the delta rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer. It is closely related to the Gauss–Newton algorithm and is part of continuing research in neural backpropagation.

Backpropagation is a special case of a more general technique called automatic differentiation. In the context of learning, backpropagation is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function.

Loss Function:

The loss function is a function that maps values of one or more variables onto a real number intuitively representing some "cost" associated with those values. For backpropagation, the loss function calculates the difference between the network output and its expected output, after a training example has propagated through the network.

Images Dataset:

2.6 UML DIAGRAMS:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

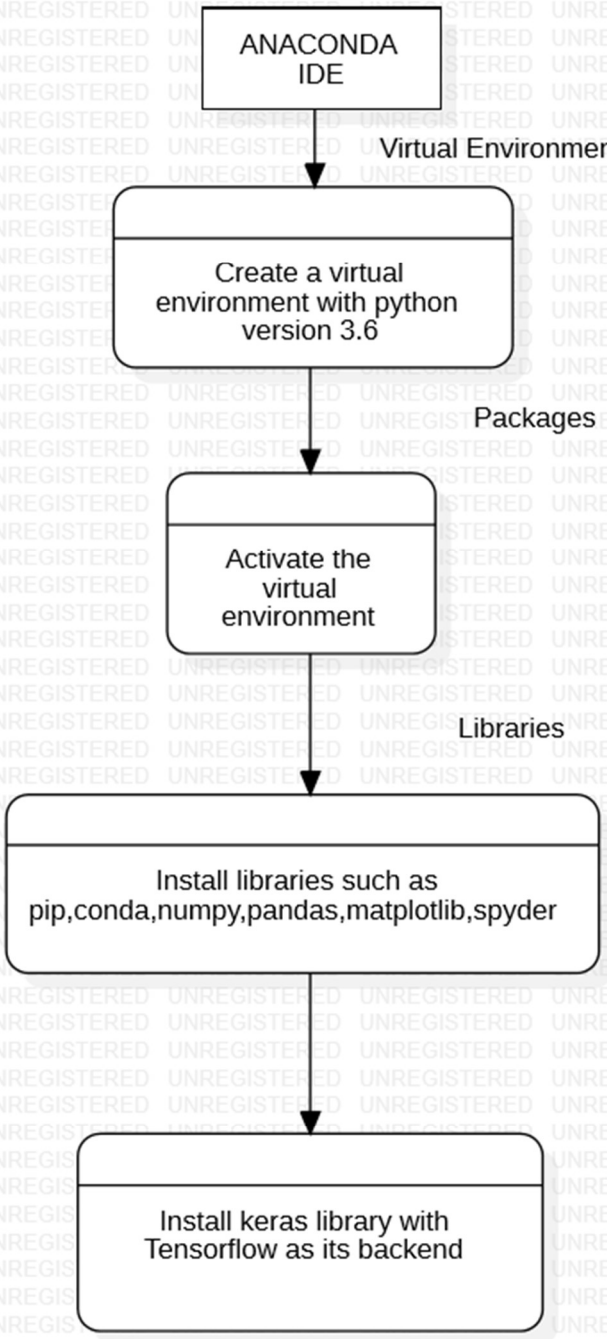
It is important to distinguish between the UML model and the set of diagrams of a system. A diagram is a partial graphic representation of a system's model. The set of diagrams need not completely cover the model and deleting a diagram does not change the model. The model may also contain documentation that drives the model elements and diagrams (such as written use cases).

UML diagrams represent two different views of a system model:

- Static (or *structural*) view: emphasizes the static structure of the system using objects, attributes, operations and relationships. It includes class diagrams and composite structure diagrams.
- Dynamic (or *behavioral*) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

UML models can be exchanged among UML tools by using the XML Metadata Interchange (XMI) format.

In UML, one of the key tools for behavior modelling is the use-case model, caused by OOSE. Use cases are a way of specifying required usages of a system. Typically, they are used to capture the requirements of a system, that is, what a system is supposed to do.


ANACONDA IDE
Virtual Environment
Create a virtual environment with python version 3.6
Packages
Activate the virtual environment
Libraries
Install libraries such as pip,conda,numpy,pandas,matplotlib,spyder
Install keras library with Tensorflow as its backend

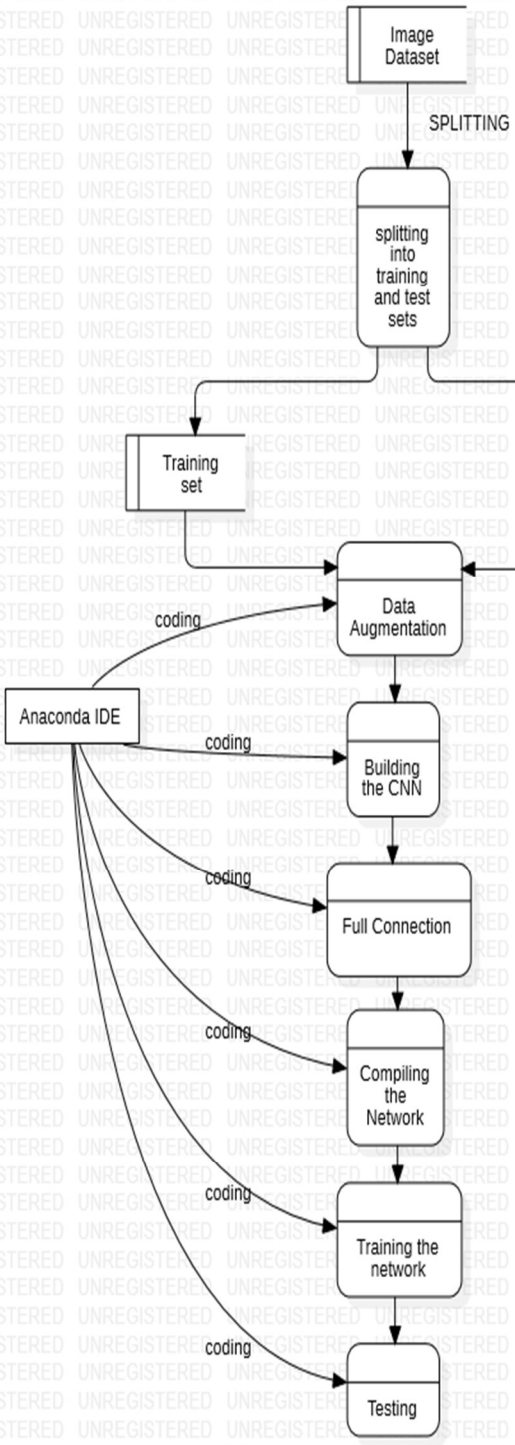


Image Dataset
SPLITTING
splitting into training and test sets
Training set
Test set
Data Augmentation
coding
Anaconda IDE
coding
Building the CNN
coding
Full Connection
coding
Compiling the Network
coding
Training the network
coding
Testing

3.IMPLEMENTATION

3.1 SYSTEM DEVELOPMENT ENVIRONMENT:

This system is developed using Arduino Integrated Development Environment.

Anaconda IDE:

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system *conda*. The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS.

Anaconda distribution comes with more than 1,400 packages as well as the Conda package and virtual environment manager, called Anaconda Navigator , so it eliminates the need to learn to install each library independently.

The open source packages can be individually installed from the Anaconda repository with the `conda install` command or using the `pip install` command that is installed with Anaconda. Pip packages provide many of the features of conda packages and in most cases they can work together.

Custom packages can be made using the `conda build` command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories.

The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda .

Anaconda Navigator:

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator :

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glueviz
- Orange
- Rstudio
- Visual Studio Code

Conda:

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects. The Conda package and environment manager is included in all versions of Anaconda, Miniconda, and Anaconda Repository.

Anaconda Cloud:

Anaconda Cloud is a package management service by Anaconda where you can find, access, store and share public and private notebooks, environments, and conda and PyPI packages. Cloud hosts useful Python packages, notebooks and environments for a wide variety of applications. You do not need to log in or to have a Cloud account, to search for public packages, download and install them.

You can build new packages using the Anaconda Client command line interface (CLI), then manually or automatically upload the packages to Cloud.

Anaconda Installation:

1. Download the Anaconda installer.
2. Optional: Verify data integrity with MD5.
3. Double click the installer to launch.

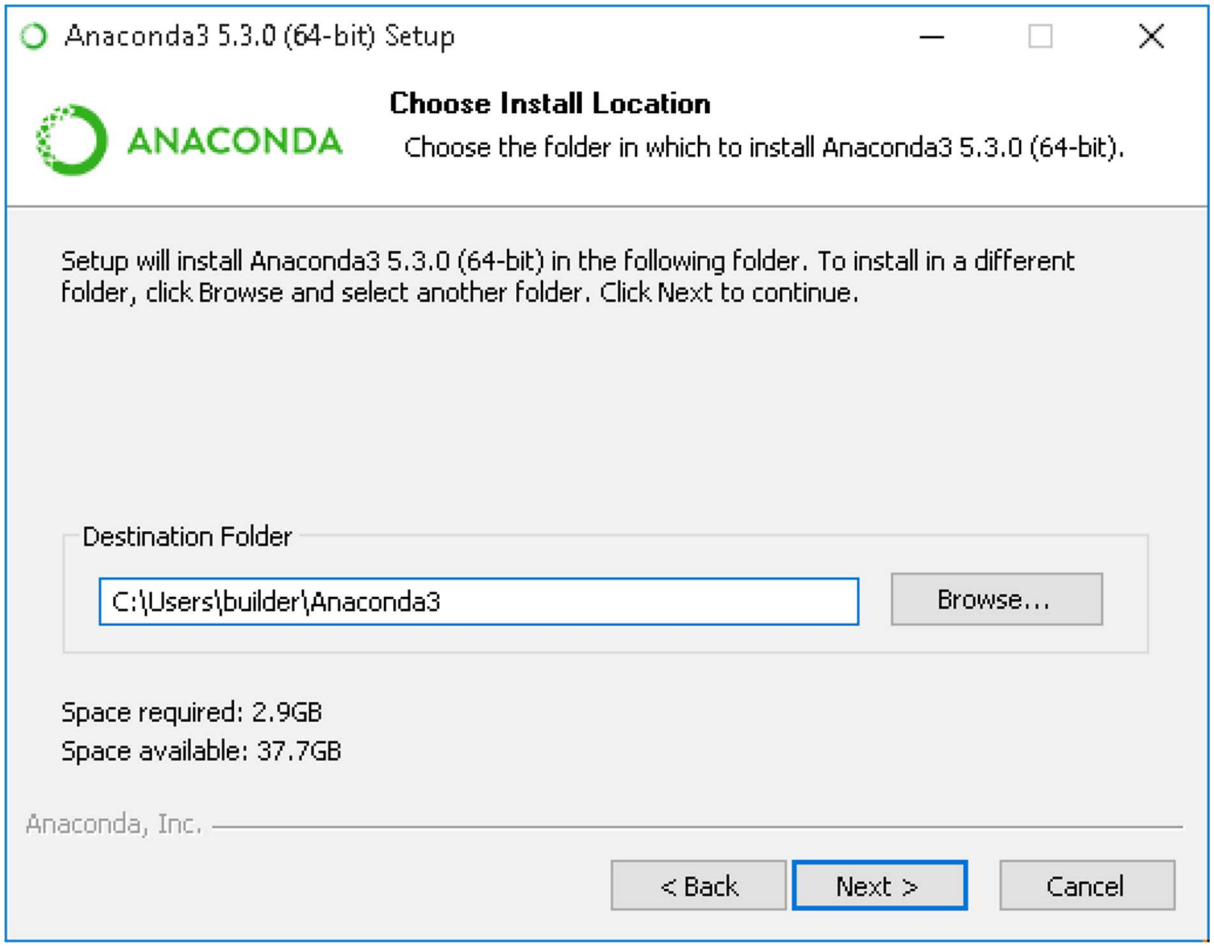
NOTE: To prevent permission errors, do not launch the installer from the Favorites folder.

NOTE: If you encounter issues during installation, temporarily disable your anti-virus software during install, then re-enable it after the installation concludes. If you installed for all users, uninstall Anaconda and re-install it for your user only and try again.

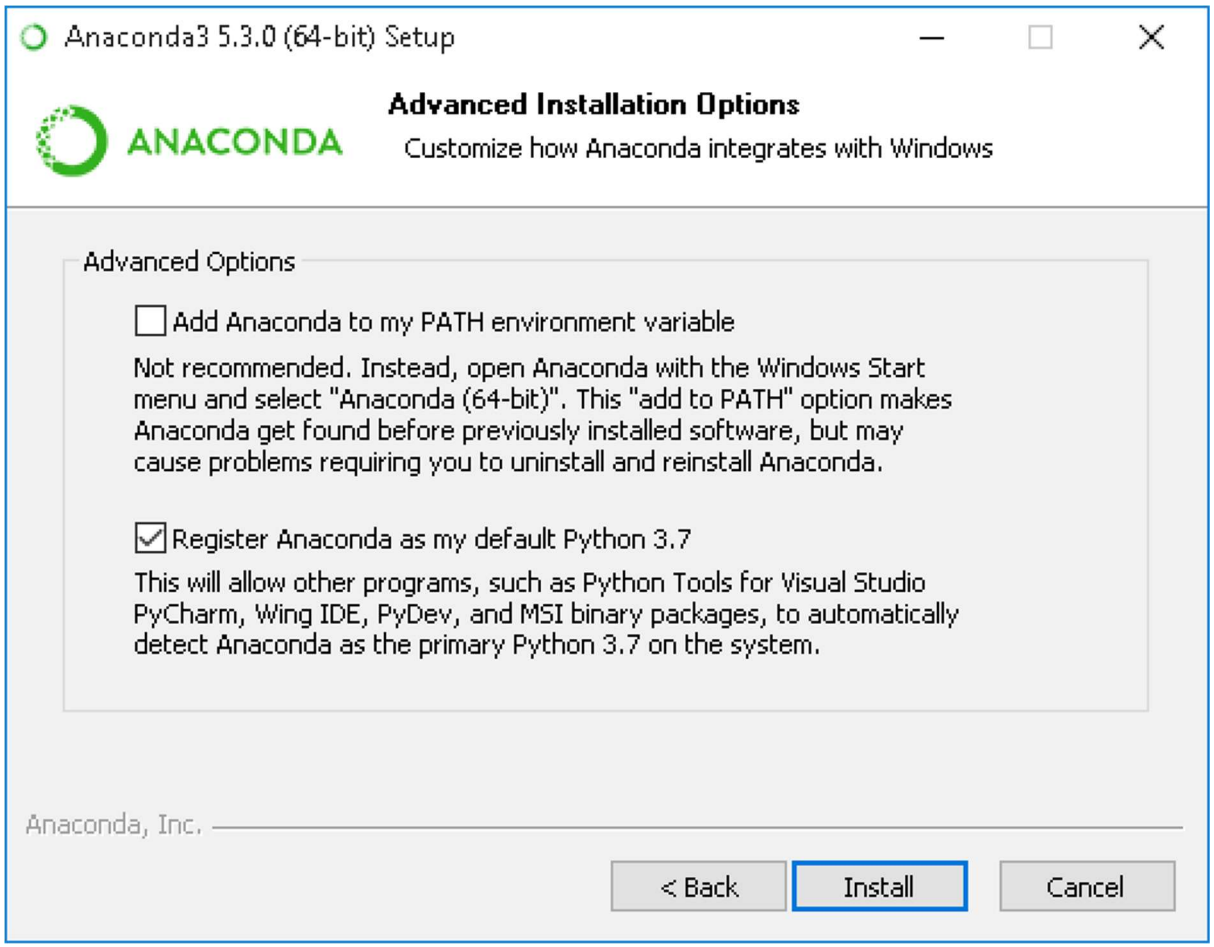
4. Click Next.
5. Read the licensing terms and click “I Agree”.
6. Select an install for “Just Me” unless you’re installing for all users (which requires Windows Administrator privileges) and click Next.
7. Select a destination folder to install Anaconda and click the Next button.

   NOTE: Install Anaconda to a directory path that does not contain spaces or unicode characters.

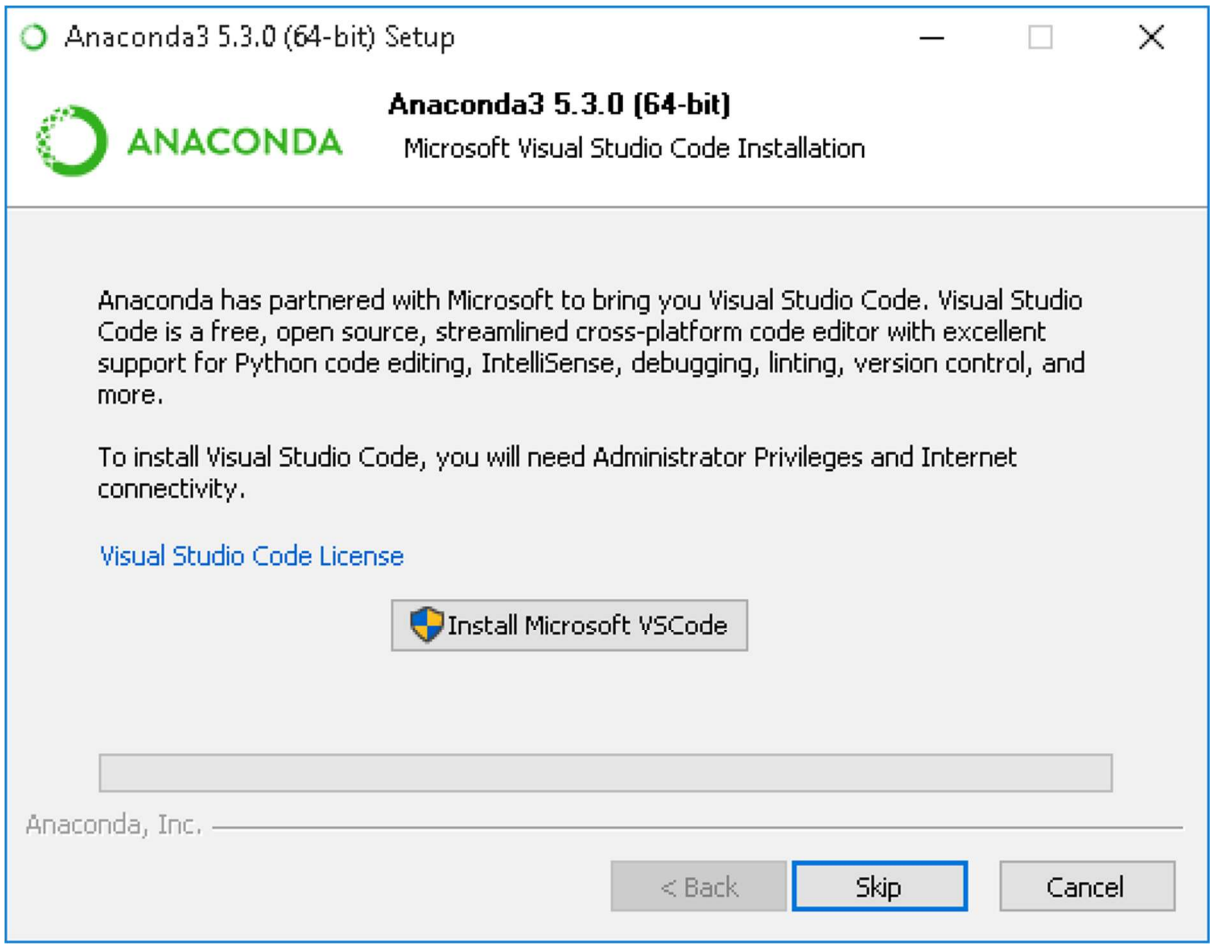
   NOTE: Do not install as Administrator unless admin privileges are required.



8. Choose whether to add Anaconda to your PATH environment variable. We recommend not adding Anaconda to the PATH environment variable, since this can interfere with other software. Instead, use Anaconda software by opening Anaconda Navigator or the Anaconda Prompt from the Start Menu.



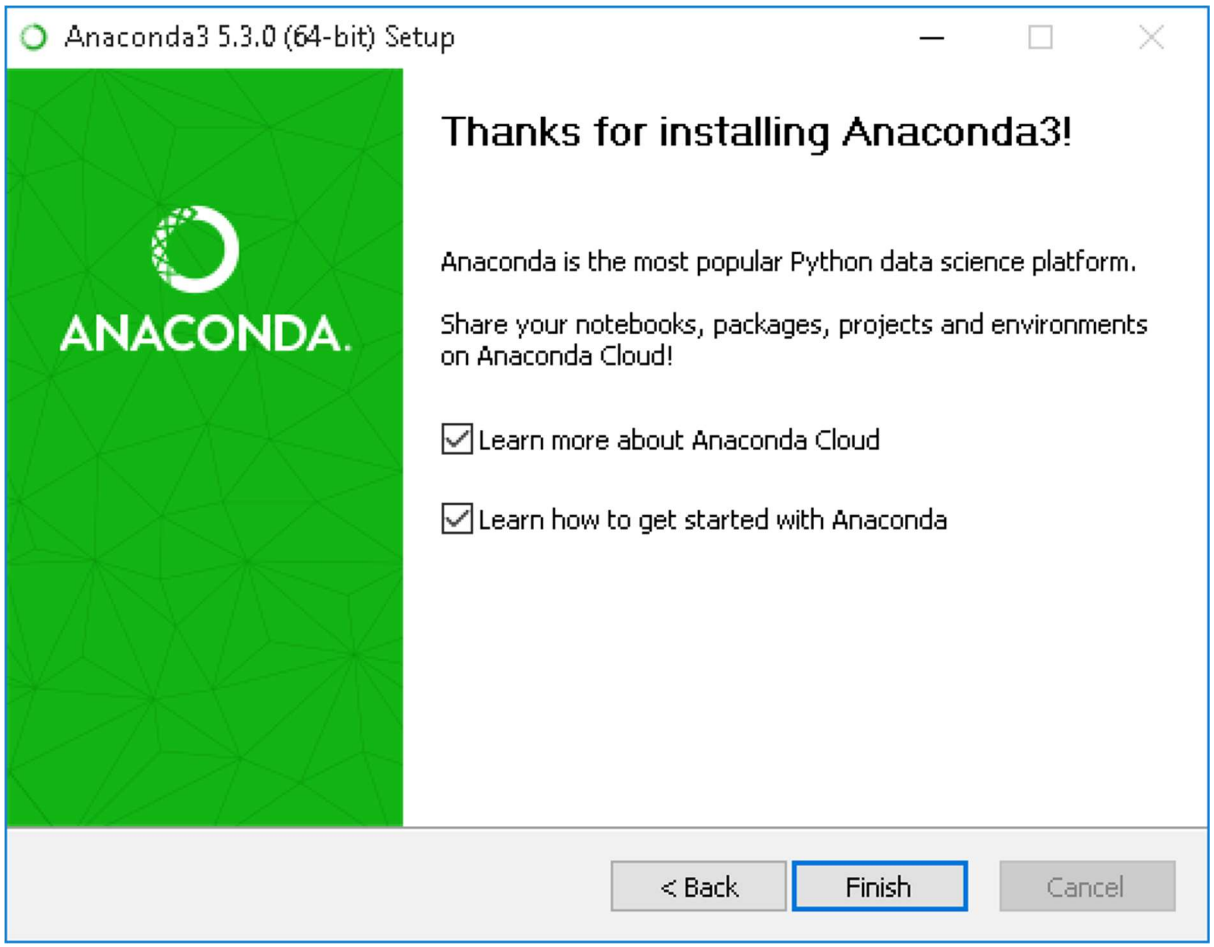
9. Choose whether to register Anaconda as your default Python. Unless you plan on installing and running multiple versions of Anaconda, or multiple versions of Python, accept the default and leave this box checked.
10. Click the Install button. If you want to watch the packages Anaconda is installing, click Show Details.
11. Click the Next button.
12. Optional: To install VS Code, click the Install Microsoft VS Code button. After the install completes click the Next button.



Or to install Anaconda without VS Code, click the Skip button.

NOTE: Installing VS Code with the Anaconda installer requires an internet connection. Offline users may be able to find an offline VS Code installer from Microsoft.

13. After a successful installation you will see the “Thanks for installing Anaconda” dialog box:



14. If you wish to read more about Anaconda Cloud and how to get started with Anaconda, check the boxes “Learn more about Anaconda Cloud” and “Learn how to get started with Anaconda”. Click the Finish button.
15. After your install is complete, verify it by opening Anaconda Navigator, a program that is included with Anaconda: from your Windows Start menu, select the shortcut Anaconda Navigator. If Navigator opens, you have successfully installed Anaconda. If not, check that you completed each step above, then see our Help page.

System requirements:

- License: Free use and redistribution under the terms of the End User License Agreement.
- Operating system: Windows 7 or newer, 64-bit macOS 10.10+, or Linux, including Ubuntu, RedHat, CentOS 6+, and others.
- System architecture: Windows- 64-bit x86, 32-bit x86; MacOS- 64-bit x86; Linux- 64-bit x86, 64-bit Power8/Power9.

Spyder:

Spyder, the Scientific Python Development Environment, includes the following key features:

- **Editor:**
  - Customizable syntax highlighting themes
  - Debugging breakpoints and conditional breakpoints (through ipdb integration)
  - Interactive execution: Run line/selection, run cell, run file, re-run and debug
  - Run configuration settings:
    - Working directory selection
    - Command line options
    - Run in Current/dedicated/external console choice
    - Automatically clear variables or enter debugging
  - Outline Explorer: Navigate cells, functions, classes, blocks, and more
  - Real-time code introspection features (powered by rope and jedi):
    - On-demand (Tab) and "dot" automatic code completion
    - Automatic popup calltips showing function signatures
    - Go-to-definition for any symbol: Functions, classes, attributes, etc. (Ctrl-Click or Ctrl-G by default)
  - Occurrence highlighting: Select or double-click any word to show all other instances throughout the current document
  - On-the-fly automatic formatters (optional):
    - Automatic insertion of closing quotes parentheses, braces and brackets
    - Automatic indentation after 'else', 'elif', 'finally', etc.
    - Smart auto-indentation based on code structure
    - Automatic insertion of colons after for, if, def, etc.
    - Automatically fix mixed indentation, EOL characters and trailing spaces
  - Real-time code analysis: * Errors/warnings/problems (powered by pyflakes) * PEP 8 and code style (powered by pycodestyle) * Code annotation parsing and "Todo lists" (TODO, FIXME, XXX, etc.)

- **IPython Console:**
    - Any number of individual consoles, each executed in a separate, isolated processes
    - Each console uses the full IPython kernel as a backend with a light GUI frontend
    - Supports all of the powerful IPython magic commands and functionality
    - Automatic code completion and calltips, and automatic link to Help
    - Range of code run options and interactivity
    - Debugging integration with enhanced ipdb debugger and the Editor
    - Inline display of Matplotlib graphics (optional)
    - The User Module Reloader, automatically re-importing modified source files
- **Variable Explorer:**
    - Can list all global variables, functions, classes, and other objects, or filter them by several criteria
    - GUI-based editors for numerous data types (numeric, strings, collections, NumPy arrays, Pandas DataFrames, dates/times, images, and more)
    - Import/export data or an entire session from/to many formats (text, csv, NumPy files, MATLAB files)
    - Interactive data visualization options (plot, histogram, image…) using Matplotlib
- **Help:**
    - Provides documentation or source code for any Python object (class, function, module…)
    - Can be triggered manually, on demand (Ctrl-I by default) or automatically on typing a left parenthesis after a function name (optional)
    - Real-time rendering and rich HTML display of the popular numpydoc docstring format (powered by Sphinx)
- Static Code Analysis: Detects an array of style issues, bad practices, potential bugs, and other problems with your code (powered by pylint)
- Profiler: Measures the performance impact of every function in a file to identify bottlenecks and aid optimization
- Projects: Allows for easy saving and restoring of settings, sessions and setup for working on multiple development efforts simultaneously
- File Explorer: Integrated filesystem viewing supporting many common operations

- Find in Files: Find string occurrences in a file, directory, or entire project with full support for powerful regular expressions and excluded locations
- Online Help: Search and browse rich HTML documentation on installed Python modules, packages, functions, classes, builtins and more, including your own
- History Log: Chronologically records every command entered into any Spyder console with timestamps, syntax highlighting and de-duplication
- Internal Console: Provides access to viewing, exploring and controlling Spyder's own operation
- **Preferences:**
    - Fully customizable keyboard shortcuts editor
    - Selection of a custom Python interpreter to use for consoles
    - Choice of 10 built-in syntax coloring theme, or create your own
    - Toggle automatic editor and console introspection, analysis and helper features
    - Options to use a variety of graphics backends and display preferences
    - Much more…
- **General Features:**
    - MATLAB-like PYTHONPATH management dialog
    - User environment variables viewer/editor (Windows-only)
    - Handy links to useful resources and documentation (Python, Matplotlib, NumPy, Scipy, Qt, IPython, etc.)
    - Interactive tour, tutorial and shortcut cheat sheet for new users

Beyond its many built-in features, Spyder's abilities can be extended even further via its plugin system and API. Spyder can also be used as a PyQt5 extension library, allowing you to build upon its functionality and embed its components, such as the interactive console, in your own software.

3.2 TECHNOLOGIES:

Python Programming Language:

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018.Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural. It also has a comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

History:

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL), capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum's long influence on Python is reflected in the title given to him by the Python community: *Benevolent Dictator For Life* (BDFL) – a post from which he gave himself permanent vacation on July 12, 2018.[34]

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backportedto Python 2.6.x and 2.7.x version series. Releases of Python 3 include the `2to3` utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. In January

2017, Google announced work on a Python 2.7 to Go transcompiler to improve performance under concurrent workloads.

Features and Philosophy:

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming[42] and metaobjects (magic methods)).[43] Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has `filter()`, `map()`, and `reduce()` functions; list comprehensions, dictionaries, sets and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document *The Zen of Python* (*PEP 20*), which includes aphorisms such as:[49]

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.[31]

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax (such as that of Perl) in favor of a simpler, less-cluttered grammar. As Alex Martelli put it: "To describe something as 'clever' is *not* considered a compliment in the Python culture." Python's

philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it".[49]

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity.[51]When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as *Pythonists*, *Pythonistas*, and *Pythoneers*.

Syntax and Semantics:

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation:

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the off-side rule.

Development:

Python's development is conducted largely through the *Python Enhancement Proposal* (PEP) process, the primary mechanism for proposing major new features, collecting community input on issues and documenting Python design decisions. Outstanding PEPs are reviewed and commented on by the Python community and Guido Van Rossum, Python's Benevolent Dictator For Life.

Enhancement of the language corresponds with development of the CPython reference implementation. The mailing list python-dev is the primary forum for the language's development. Specific issues are discussed in the Roundup bug tracker maintained at python.org. Development originally took place on a self-hosted source-code repository running Mercurial, until Python moved to GitHub in January 2017.

CPython's public releases come in three types, distinguished by which part of the version number is incremented:

- Backward-incompatible versions, where code is expected to break and need to be manually ported. The first part of the version number is incremented. These releases happen infrequently—for example, version 3.0 was released 8 years after 2.0.
- Major or "feature" releases, about every 18 months, are largely compatible but introduce new features. The second part of the version number is incremented. Each major version is supported by bugfixes for several years after its release.
- Bugfix releases, which introduce no new features, occur about every 3 months and are made when a sufficient number of bugs have been fixed upstream since the last release. Security vulnerabilities are also patched in these releases. The third and final part of the version number is incremented.

Many alpha, beta, and release-candidates are also released as previews and for testing before final releases. Although there is a rough schedule for each release, they are often delayed if the code is not ready. Python's development team monitors the state of the code by running the large unit test suite during development, and using the BuildBot continuous integration system.[116]

The community of Python developers has also contributed over 86,000[117] software modules (as of 20 August 2016) to the Python Package Index (PyPI), the official repository of third-party Python libraries.

The major academic conference on Python is PyCon. There are also special Python mentoring programmes, such as Pyladies.

Uses:

Since 2003, Python has consistently ranked in the top ten most popular programming languages in the TIOBE Programming Community Index where, as of December 2018, it is the third most popular language (behind Java, and C). It was selected Programming Language of the Year in 2007, 2010, and 2018.

An empirical study found that scripting languages, such as Python, are more productive than conventional languages, such as C and Java, for programming problems involving string manipulation and search in a dictionary, and determined that memory consumption was often "better than Java and not much worse than C or C++".

Large organizations that use Python include Wikipedia, Google, Yahoo!, CERN, NASA, Facebook, Amazon, Instagram, Spotify a nd some smaller entities like ILM and ITA. The social news networking site Reddit is written entirely in Python.

Python can serve as a scripting language for web applications, e.g., via mod_wsgi for the Apache web server. With Web Server Gateway Interface, a standard API has evolved to facilitate these applications. Web frameworks like Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle and Zope support developers in the design and maintenance of complex applications. Pyjs and IronPython can be used to develop the client-side of Ajax-based applications. SQLAlchemy can be used as data mapper to a relational database. Twisted is a framework to program communications between computers, and is used (for example) by Dropbox.

Libraries such as NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing, with specialized libraries such as Biopython and Astropy providing domain-specific functionality. SageMath is a mathematical software with a notebook interface programmable in Python: its library covers many aspects of mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus.

Python has been successfully embedded in many software products as a scripting language, including in finite element method software such as Abaqus, 3D parametric modeler like FreeCAD, 3D animation packages such as 3ds Max, Blender, Cinema

4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, the visual effects compositor Nuke, 2D imaging programs like GIMP, Inkscape, Scribus and Paint Shop Pro, and musical notation programs like scorewriter and capella. GNU Debugger uses Python as a pretty printer to show complex structures such as C++ containers. Esri promotes Python as the best choice for writing scripts in ArcGIS. It has also been used in several video games, and has been adopted as first of the three available programming languages in Google App Engine, the other two being Java and Go. Python is also used in algorithmic trading and quantitative finance. Python can also be implemented in APIs of online brokerages that run on other languages by using wrappers.

Python is commonly used in artificial intelligence projects with the help of libraries like TensorFlow, Keras and Scikit-learn. As a scripting language with modular architecture, simple syntax and rich text processing tools, Python is often used for natural language processing.

Many operating systems include Python as a standard component. It ships with most Linux distributions, AmigaOS 4, FreeBSD, NetBSD, OpenBSD and macOS, and can be used from the command line (terminal). Many Linux distributions use installers written in Python: Ubuntu uses the Ubiquity installer, while Red Hat Linux and Fedora use the Anaconda installer. Gentoo Linux uses Python in its package management system, Portage.

Python is used extensively in the information security industry, including in exploit development.

Most of the Sugar software for the One Laptop per Child XO, now developed at Sugar Labs, is written in Python. The Raspberry Pi single-board computer project has adopted Python as its main user-programming language.

LibreOffice includes Python, and intends to replace Java with Python. Its Python Scripting Provider is a core feature[153] since Version 4.0 from 7 February 2013.

Artificial Neural Networks:

Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

In common ANN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called 'edges'. Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. Artificial neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

History:

Warren McCulloch and Walter Pitts (1943) created a computational model for neural networks based on mathematics and algorithms called threshold logic. This model paved the way for neural network research to split into two approaches. One approach focused on biological processes in the brain while the other focused on the application of neural networks to artificial intelligence. This work led to work on nerve networks and their link to finite automata

Hebbian learning:

In the late 1940s, D. O. Hebb created a learning hypothesis based on the mechanism of neural plasticity that became known as Hebbian learning. Hebbian learning is unsupervised learning. This evolved into models for long term potentiation. Researchers started applying these ideas to computational models in 1948 with Turing's B-type machines. Farley and Clark (1954) first used computational machines, then called "calculators", to simulate a Hebbian network. Other neural network computational machines were created by Rochester, Holland, Habit and Duda (1956). Rosenblatt (1958) created the perceptron, an algorithm for pattern recognition. With mathematical notation, Rosenblatt described circuitry not in the basic perceptron, such as the exclusive-or circuit that could not be processed by neural networks at the time. In 1959, a biological model proposed by Nobel laureates Hubel and Wiesel was based on their discovery of two types of cells in the primary visual cortex: simple cells and complex cells. The first functional networks with many layers were published by Ivakhnenko and Lapa in 1965, becoming the Group Method of Data Handling.

Neural network research stagnated after machine learning research by Minsky and Papert (1969), who discovered two key issues with the computational machines that processed neural networks. The first was that basic perceptrons were incapable of processing the exclusive-or circuit. The second was that computers didn't have enough processing power to effectively handle the work required by large neural networks. Neural network research slowed until computers achieved far greater processing power. Much of artificial intelligence had focused on high-level (symbolic) models that are processed by using algorithms, characterized for example by expert systems with knowledge embodied in *if-then* rules, until in the late 1980s research expanded to low-level (sub-symbolic) machine learning, characterized by knowledge embodied in the parameters of a cognitive model.

Backpropagation:

A key trigger for renewed interest in neural networks and learning was Werbos's (1975) backpropagation algorithm that effectively solved the exclusive-or problem by making the training of multi-layer networks feasible and efficient. Backpropagation distributed the error term back up through the layers, by modifying the weights at each node.

In the mid-1980s, parallel distributed processing became popular under the name connectionism. Rumelhart and McClelland (1986) described the use of connectionism to simulate neural processes.

Support vector machines and other, much simpler methods such as linear classifiers gradually overtook neural networks in machine learning popularity. However, using neural networks transformed some domains, such as the prediction of protein structures.

In 1992, max-pooling was introduced to help with least shift invariance and tolerance to deformation to aid in 3D object recognition. In 2010, Backpropagation training through max-pooling was accelerated by GPUs and shown to perform better than other pooling variants.

The vanishing gradient problem affects many-layered feedforward networks that used backpropagation and also recurrent neural networks (RNNs). As errors propagate from layer to layer, they shrink exponentially with the number of layers, impeding the tuning of neuron weights that is based on those errors, particularly affecting deep networks.

To overcome this problem, Schmidhuber adopted a multi-level hierarchy of networks (1992) pre-trained one level at a time by unsupervised learning and fine-tuned by backpropagation. Behnke (2003) relied only on the sign of the gradient (Rprop) on problems such as image reconstruction and face localization.

Hinton et al. (2006) proposed learning a high-level representation using successive layers of binary or real-valued latent variables with a restricted Boltzmann machine to model each layer. Once sufficiently many layers have been learned, the deep architecture may be used as a generative model by reproducing the data when sampling down the model (an "ancestral pass") from the top level feature activations. In 2012, Ng and Dean created a network that learned to recognize higher-level concepts, such as cats, only from watching unlabeled images taken from YouTube videos.

Earlier challenges in training deep neural networks were successfully addressed with methods such as unsupervised pre-training, while available computing power increased through the use

of GPUs and distributed computing. Neural networks were deployed on a large scale, particularly in image and visual recognition problems. This became known as "deep learning"

Hardware-based designs:

Computational devices were created in CMOS, for both biophysical simulation and neuromorphic computing. Nanodevices for very large scale principal components analyses and convolution may create a new class of neural computing because they are fundamentally analog rather than digital (even though the first implementations may use digital devices). Ciresan and colleagues (2010) in Schmidhuber's group showed that despite the vanishing gradient problem, GPUs make back-propagation feasible for many-layered feedforward neural networks.

Contests:

Between 2009 and 2012, recurrent neural networks and deep feedforward neural networks developed in Schmidhuber's research group won eight international competitions in pattern recognition and machine learning. For example, the bi-directional and multi-dimensional long short-term memory (LSTM) of Graves et al. won three competitions in connected handwriting recognition at the 2009 International Conference on Document Analysis and Recognition (ICDAR), without any prior knowledge about the three languages to be learned.

Ciresan and colleagues won pattern recognition contests, including the IJCNN 2011 Traffic Sign Recognition Competition, the ISBI 2012 Segmentation of Neuronal Structures in Electron Microscopy Stacks challenge and others. Their neural networks were the first pattern recognizers to achieve human-competitive or even superhuman performance on benchmarks such as traffic sign recognition (IJCNN 2012), or the MNIST handwritten digits problem.

Researchers demonstrated (2010) that deep neural networks interfaced to a hidden Markov model with context-dependent states that define the neural network output layer can drastically reduce errors in large-vocabulary speech recognition tasks such as voice search.

GPU-based implementations of this approach won many pattern recognition contests, including the IJCNN 2011 Traffic Sign Recognition Competition, the ISBI 2012 Segmentation of neuronal structures in EM stacks challenge, the ImageNet Competition and others.

Deep, highly nonlinear neural architectures similar to the neocognitron and the "standard architecture of vision", inspired by simple and complex cells, were pre-trained by

unsupervised methods by Hinton. A team from his lab won a 2012 contest sponsored by Merck to design software to help find molecules that might identify new drugs.

Convolutional networks:

As of 2011, the state of the art in deep learning feedforward networks alternated between convolutional layers and max-pooling layers, topped by several fully or sparsely connected layers followed by a final classification layer. Learning is usually done without unsupervised pre-training. In the convolutional layer, there are filters that are convolved with the input. Each filter is equivalent to a weights vector that has to be trained.

Such supervised deep learning methods were the first to achieve human-competitive performance on certain tasks.

Artificial neural networks were able to guarantee shift invariance to deal with small and large natural objects in large cluttered scenes, only when invariance extended beyond shift, to all ANN-learned concepts, such as location, type (object class label), scale, lighting and others. This was realized in Developmental Networks (DNs) whose embodiments are Where-What Networks, WWN-1 (2008) through WWN-7 (2013) .

Machine Learning:

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in a wide variety of applications, such as email filtering, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task. Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

Overview of Machine Learning:

The name machine learning was coined in 1959 by Arthur Samuel. Tom M. Mitchell provided a widely quoted, more formal definition of the algorithms studied in the machine learning field: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." This definition of the tasks in which machine learning is concerned offers a fundamentally operational definition rather than defining the field in cognitive terms. This follows Alan Turing's proposal in his paper "Computing Machinery and Intelligence", in which the question "Can machines think?" is replaced with the question "Can machines do what we (as thinking entities) can do?". In Turing's proposal the various characteristics that could be possessed by a thinking machine and the various implications in constructing one are exposed.

Machine learning tasks are classified into several broad categories. In supervised learning, the algorithm builds a mathematical model from a set of data that contains both the inputs and the desired outputs. For example, if the task were determining whether an image contained a certain object, the training data for a supervised learning algorithm would include images with and without that object (the input), and each image would have a label (the output) designating whether it contained the object. In special cases, the input may be only partially available, or restricted to special feedback. Semi-supervised learning algorithms develop mathematical models from incomplete training data, where a portion of the sample input doesn't have labels.

Classification algorithms and regression algorithms are types of supervised learning. Classification algorithms are used when the outputs are restricted to a limited set of values. For a classification algorithm that filters emails, the input would be an incoming email, and the output would be the name of the folder in which to file the email. For an algorithm that identifies spam emails, the output would be the prediction of either "spam" or "not spam", represented by the Boolean values true and false. Regression algorithms are named for their continuous outputs, meaning they may have any value within a range. Examples of a continuous value are the temperature, length, or price of an object.

In unsupervised learning, the algorithm builds a mathematical model from a set of data which contains only inputs and no desired output labels. Unsupervised learning algorithms are used to find structure in the data, like grouping or clustering of data points. Unsupervised learning can discover patterns in the data, and can group the inputs into categories, as in feature

learning. Dimensionality reduction is the process of reducing the number of "features", or inputs, in a set of data.

Active learning algorithms access the desired outputs (training labels) for a limited set of inputs based on a budget, and optimize the choice of inputs for which it will acquire training labels. When used interactively, these can be presented to a human user for labeling. Reinforcement learning algorithms are given feedback in the form of positive or negative reinforcement in a dynamic environment, and are used in autonomous vehicles or in learning to play a game against a human opponent. Other specialized algorithms in machine learning include topic modeling, where the computer program is given a set of natural language documents and finds other documents that cover similar topics. Machine learning algorithms can be used to find the unobservable probability density function in density estimation problems. Meta learning algorithms learn their own inductive bias based on previous experience. In developmental robotics, robot learning algorithms generate their own sequences of learning experiences, also known as a curriculum, to cumulatively acquire new skills through self-guided exploration and social interaction with humans. These robots use guidance mechanisms such as active learning, maturation, motor synergies, and imitation.

History and Relationships to other fields:

Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. As a scientific endeavour, machine learning grew out of the quest for artificial intelligence. Already in the early days of AI as an academic discipline, some researchers were interested in having machines learn from data. They attempted to approach the problem with various symbolic methods, as well as what were then termed "neural networks"; these were mostly perceptrons and other models that were later found to be reinventions of the generalized linear models of statistics Probabilistic reasoning was also employed, especially in automated medical diagnosis.

However, an increasing emphasis on the logical, knowledge-based approach caused a rift between AI and machine learning. Probabilistic systems were plagued by theoretical and practical problems of data acquisition and representation. By 1980, expert systems had come to dominate AI, and statistics was out of favor. Work on symbolic/knowledge-based learning did continue within AI, leading to inductive logic programming, but the more statistical line of research was now outside the field of AI proper, in pattern recognition and information

retrieval. Neural networks research had been abandoned by AI and computer science around the same time. This line, too, was continued outside the AI/CS field, as "connectionism", by researchers from other disciplines including Hopfield, Rumelhart and Hinton. Their main success came in the mid-1980s with the reinvention of backpropagation.

Machine learning, reorganized as a separate field, started to flourish in the 1990s. The field changed its goal from achieving artificial intelligence to tackling solvable problems of a practical nature. It shifted focus away from the symbolic approaches it had inherited from AI, and toward methods and models borrowed from statistics and probability theory. It also benefited from the increasing availability of digitized information, and the ability to distribute it via the Internet.

Relation to data mining:

Machine learning and data mining often employ the same methods and overlap significantly, but while machine learning focuses on prediction, based on known properties learned from the training data, data mining focuses on the discovery of (previously) unknown properties in the data (this is the analysis step of knowledge discovery in databases). Data mining uses many machine learning methods, but with different goals; on the other hand, machine learning also employs data mining methods as "unsupervised learning" or as a preprocessing step to improve learner accuracy. Much of the confusion between these two research communities (which do often have separate conferences and separate journals, ECML PKDD being a major exception) comes from the basic assumptions they work with: in machine learning, performance is usually evaluated with respect to the ability to reproduce known knowledge, while in knowledge discovery and data mining (KDD) the key task is the discovery of previously unknown knowledge. Evaluated with respect to known knowledge, an uninformed (unsupervised) method will easily be outperformed by other supervised methods, while in a typical KDD task, supervised methods cannot be used due to the unavailability of training data.

Relation to optimization:

Machine learning also has intimate ties to optimization: many learning problems are formulated as minimization of some loss function on a training set of examples. Loss functions express the discrepancy between the predictions of the model being trained and the actual problem instances (for example, in classification, one wants to assign a label to instances, and models are trained to correctly predict the pre-assigned labels of a set of examples). The difference between the two fields arises from the goal of generalization: while optimization

algorithms can minimize the loss on a training set, machine learning is concerned with minimizing the loss on unseen samples.

Relation to statistics:

Machine learning and statistics are closely related fields. According to Michael I. Jordan, the ideas of machine learning, from methodological principles to theoretical tools, have had a long pre-history in statistics. He also suggested the term data science as a placeholder to call the overall field.

Leo Breiman distinguished two statistical modelling paradigms: data model and algorithmic model, wherein "algorithmic model" means more or less the machine learning algorithms like Random forest.

Some statisticians have adopted methods from machine learning, leading to a combined field that they call statistical learning.

Types of learning algorithms:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

Supervised and semi-supervised learning:

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data, and consists of a set of training examples. Each training example has one or more inputs and a desired output, also known as a supervisory signal. In the case of semi-supervised learning algorithms, some of the training examples are missing the desired output. In the mathematical model, each training example is represented by an array or vector, and the training data by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs. An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task.

Supervised learning algorithms include classification and regression Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs may have any numerical value within a range. Similarity learning is an area of supervised machine learning closely related to regression and classification, but the

goal is to learn from examples using a similarity function that measures how similar or related two objects are. It has applications in ranking, recommendation systems, visual identity tracking, face verification, and speaker verification.

Unsupervised learning:

Unsupervised learning algorithms take a set of data that contains only inputs, and find structure in the data, like grouping or clustering of data points. The algorithms therefore learn from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning algorithms identify commonalities in the data and react based on the presence or absence of such commonalities in each new piece of data. A central application of unsupervised learning is in the field of density estimation in statistics though unsupervised learning encompasses other domains involving summarizing and explaining data features.

Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to one or more predesignated criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some similarity metric and evaluated, for example, by internal compactness, or the similarity between members of the same cluster, and separation, the difference between clusters. Other methods are based on estimated density and graph connectivity.

Reinforcement learning:

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Due to its generality, the field is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms. In machine learning, the environment is typically represented as a Markov Decision Process (MDP). Many reinforcement learning algorithms use dynamic programming techniques Reinforcement learning algorithms do not assume knowledge of an exact mathematical model of the MDP, and are used when exact models are infeasible Reinforcement learning algorithms are used in autonomous vehicles or in learning to play a game against a human opponent.

Processes and techniques:

Various processes, techniques and methods can be applied to one or more types of machine learning algorithms to enhance their performance.

Several learning algorithms aim at discovering better representations of the inputs provided during training. Classic examples include principal components analysis and cluster analysis. Feature learning algorithms, also called representation learning algorithms, often attempt to preserve the information in their input but also transform it in a way that makes it useful, often as a pre-processing step before performing classification or predictions. This technique allows reconstruction of the inputs coming from the unknown data-generating distribution, while not being necessarily faithful to configurations that are implausible under that distribution. This replaces manual feature engineering, and allows a machine to both learn the features and use them to perform a specific task.

Feature learning can be either supervised or unsupervised. In supervised feature learning, features are learned using labeled input data. Examples include artificial neural networks, multilayer perceptrons, and supervised dictionary learning. In unsupervised feature learning, features are learned with unlabeled input data. Examples include dictionary learning, independent component analysis, autoencoders, matrix factorization and various forms of clustering.

Manifold learning algorithms attempt to do so under the constraint that the learned representation is low-dimensional. Sparse coding algorithms attempt to do so under the constraint that the learned representation is sparse, meaning that the mathematical model has many zeros. Multilinear subspace learning algorithms aim to learn low-dimensional representations directly from tensor representations for multidimensional data, without reshaping them into higher-dimensional vectors. Deep learning algorithms discover multiple levels of representation, or a hierarchy of features, with higher-level, more abstract features defined in terms of (or generating) lower-level features. It has been argued that an intelligent machine is one that learns a representation that disentangles the underlying factors of variation that explain the observed data

Feature learning is motivated by the fact that machine learning tasks such as classification often require input that is mathematically and computationally convenient to process. However, real-world data such as images, video, and sensory data has not yielded to attempts to

algorithmically define specific features. An alternative is to discover such features or representations through examination, without relying on explicit algorithms.

Sparse dictionary learning:

Sparse dictionary learning is a feature learning method where a training example is represented as a linear combination of basis functions, and is assumed to be a sparse matrix. The method is strongly NP-hard and difficult to solve approximately A popular heuristic method for sparse dictionary learning is the K-SVD algorithm. Sparse dictionary learning has been applied in several contexts. In classification, the problem is to determine to which classes a previously unseen training example belongs. For a dictionary where each class has already been built, a new training example is associated with the class that is best sparsely represented by the corresponding dictionary. Sparse dictionary learning has also been applied in image de-noising. The key idea is that a clean image patch can be sparsely represented by an image dictionary, but the noise cannot.

Outlier Detection:

In data mining, anomaly detection, also known as outlier detection, is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data. Typically, the anomalous items represent an issue such as bank fraud, a structural defect, medical problems or errors in a text. Anomalies are referred to as outliers, novelties, noise, deviations and exceptions.

In particular, in the context of abuse and network intrusion detection, the interesting objects are often not rare objects, but unexpected bursts in activity. This pattern does not adhere to the common statistical definition of an outlier as a rare object, and many outlier detection methods (in particular, unsupervised algorithms) will fail on such data, unless it has been aggregated appropriately. Instead, a cluster analysis algorithm may be able to detect the micro-clusters formed by these patterns.

Three broad categories of anomaly detection techniques exist. Unsupervised anomaly detection techniques detect anomalies in an unlabeled test data set under the assumption that the majority of the instances in the data set are normal, by looking for instances that seem to fit least to the remainder of the data set. Supervised anomaly detection techniques require a data set that has been labeled as "normal" and "abnormal" and involves training a classifier (the key difference to many other statistical classification problems is the inherent unbalanced nature of outlier detection). Semi-supervised anomaly detection techniques construct a model representing

normal behavior from a given normal training data set, and then test the likelihood of a test instance to be generated by the model.

Decision trees:

Decision tree learning uses a decision tree as a predictive model to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modeling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data, but the resulting classification tree can be an input for decision making.

Association rules:

Association rule learning is a rule-based machine learning method for discovering relationships between variables in large databases. It is intended to identify strong rules discovered in databases using some measure of "interestingness" This rule-based approach generates new rules as it analyzes more data. The ultimate goal, assuming the set of data is large enough, is to help a machine mimic the human brain’s feature extraction and abstract association capabilities for data that has not been categorized.

Rule-based machine learning is a general term for any machine learning method that identifies, learns, or evolves "rules" to store, manipulate or apply knowledge. The defining characteristic of a rule-based machine learning algorithm is the identification and utilization of a set of relational rules that collectively represent the knowledge captured by the system. This is in contrast to other machine learning algorithms that commonly identify a singular model that can be universally applied to any instance in order to make a prediction. Rule-based machine learning approaches include learning classifier systems, association rule learning, and artificial immune systems.

Based on the concept of strong rules, Rakesh Agrawal, Tomasz Imieliński and Arun Swami introduced association rules for discovering regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets. For example, the

rule found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat. Such information can be used as the basis for decisions about marketing activities such as promotional pricing or product placements. In addition to market basket analysis, association rules are employed today in application areas including Web usage mining, intrusion detection, continuous production, and bioinformatics. In contrast with sequence mining, association rule learning typically does not consider the order of items either within a transaction or across transactions.

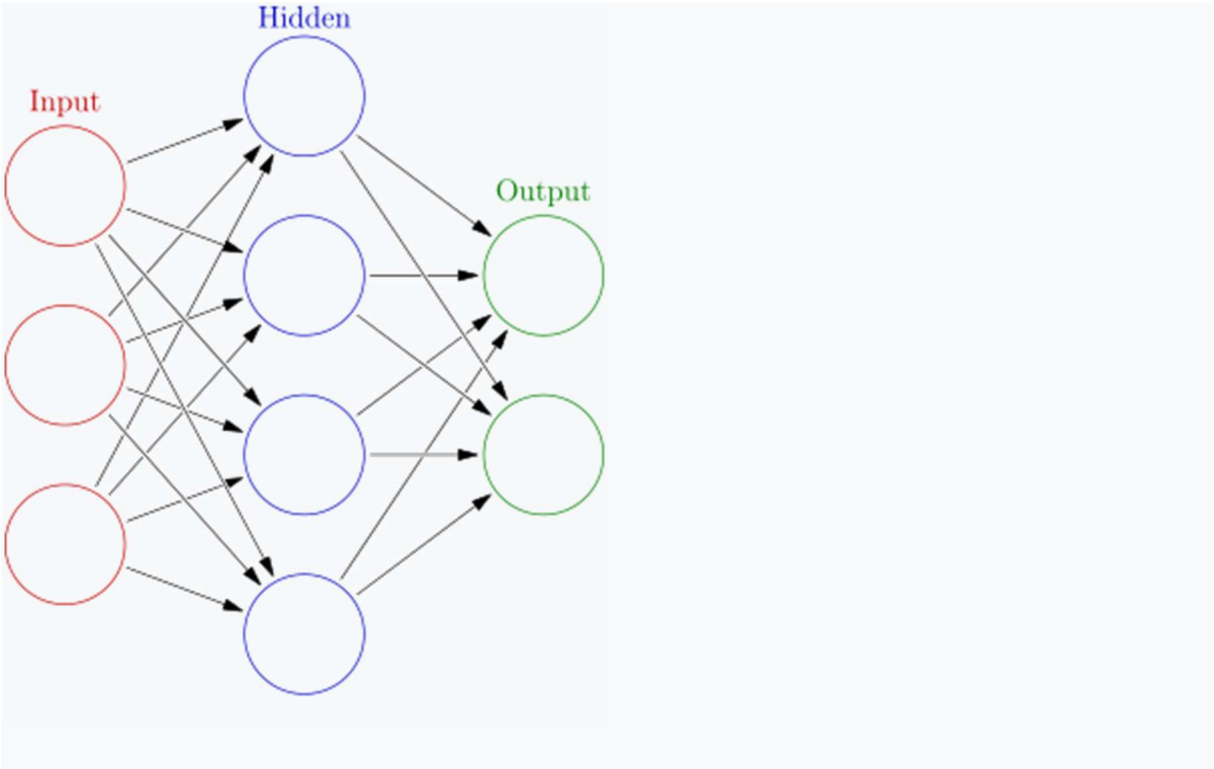
Learning classifier systems (LCS) are a family of rule-based machine learning algorithms that combine a discovery component, typically a genetic algorithm, with a learning component, performing either supervised learning, reinforcement learning, or unsupervised learning. They seek to identify a set of context-dependent rules that collectively store and apply knowledge in a piecewise manner in order to make predictions

Inductive logic programming (ILP) is an approach to rule-learning using logic programming as a uniform representation for input examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program that entails all positive and no negative examples. Inductive programming is a related field that considers any kind of programming languages for representing hypotheses (and not only logic programming), such as functional programs.

Inductive logic programming is particularly useful in bioinformatics and natural language processing. Gordon Plotkin and Ehud Shapiro laid the initial theoretical foundation for inductive machine learning in a logical setting Shapiro built their first implementation (Model Inference System) in 1981: a Prolog program that inductively inferred logic programs from positive and negative examples. The term inductive here refers to philosophical induction, suggesting a theory to explain observed facts, rather than mathematical induction, proving a property for all members of a well-ordered set.

Models:

Artificial neural networks:



An ANN is a model based on a collection of connected units or nodes called "artificial neurons", which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit information, a "signal", from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it. In common ANN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between nput layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. Artificial neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

Deep learning consists of multiple hidden layers in an artificial neural network. This approach tries to model the way the human brain processes light and sound into vision and hearing. Some successful applications of deep learning are computer vision and speech recognition

Support vector machines:

Support vector machines (SVMs), also known as support vector networks, are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other An SVM training algorithm is a non-probabilistic, binary, linear classifier, although methods such as Platt scaling exist to use SVM in a probabilistic classification setting. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independence with a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning. Bayesian networks that model sequences of variables, like speech signals or protein sequences, are called dynamic Bayesian networks. Generalizations of Bayesian networks that can represent and solve decision problems under uncertainty are called influence diagrams.

Genetic algorithms:

A genetic algorithm (GA) is a search algorithm and heuristic technique that mimics the process of natural selection, using methods such as mutation and crossover to generate new genotypes in the hope of finding good solutions to a given problem. In machine learning, genetic algorithms were used in the 1980s and 1990s Conversely, machine learning techniques have been used to improve the performance of genetic and evolutionary algorithms

Ethics:

Machine learning poses a host of ethical questions. Systems which are trained on datasets collected with biases may exhibit these biases upon use (algorithmic bias), thus digitizing cultural prejudices For example, using job hiring data from a firm with racist hiring policies may lead to a machine learning system duplicating the bias by scoring job applicants against similarity to previous successful applicants Responsible collection of data and documentation of algorithmic rules used by a system thus is a critical part of machine learning.

Because language contains biases, machines trained on language corpora will necessarily also learn bias.

Other forms of ethical challenges, not related to personal biases, are more seen in health care. There are concerns among health care professionals that these systems might not be designed in the public's interest, but as income generating machines. This is especially true in the United States where there is a perpetual ethical dilemma of improving health care, but also increasing profits. For example, the algorithms could be designed to provide patients with unnecessary tests or medication in which the algorithm's proprietary owners hold stakes in. There is huge potential for machine learning in health care to provide professionals a great tool to diagnose, medicate, and even plan recovery paths for patients, but this will not happen until the personal biases mentioned previously, and these "greed" biases are addressed.

3.3 PLATFORM:

A computer with a CPU and GPU with CUDA support is the platform for the project.

CPU:

A **central processing unit** (**CPU**), also called a **central processor** or **main processor**, is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logic, controlling, and input/output (I/O) operations specified by the instructions. The computer industry has used the term "central processing unit" at least since the early 1960s. Traditionally, the term "CPU" refers to a processor, more specifically to its processing unit and control unit (CU), distinguishing these core elements of a computer from external components such as main memory and I/O circuitry.

The form, design, and implementation of CPUs have changed over the course of their history, but their fundamental operation remains almost unchanged. Principal components of a CPU include the arithmetic logic unit (ALU) that performs arithmetic and logic operations, processor registers that supply operands to the ALU and store the results of ALU operations and a control unit that orchestrates the fetching (from memory) and execution of instructions by directing the coordinated operations of the ALU, registers and other components.

Most modern CPUs are microprocessors, meaning they are contained on a single integrated circuit (IC) chip. An IC that contains a CPU may also contain memory, peripheral interfaces, and other components of a computer; such integrated devices are variously called microcontrollers or systems on a chip (SoC). Some computers employ a multi-core processor, which is a single chip containing two or more CPUs called "cores"; in that context, one can speak of such single chips as "sockets".

Array processors or vector processors have multiple processors that operate in parallel, with no unit considered central. There also exists the concept of virtual CPUswhich are an abstraction of dynamical aggregated computational resources.

Operation:

The fundamental operation of most CPUs, regardless of the physical form they take, is to execute a sequence of stored instructions that is called a program. The instructions to be executed are kept in some kind of computer memory. Nearly all CPUs follow the fetch, decode and execute steps in their operation, which are collectively known as the instruction cycle.

After the execution of an instruction, the entire process repeats, with the next instruction cycle normally fetching the next-in-sequence instruction because of the incremented value in the program counter. If a jump instruction was executed, the program counter will be modified to contain the address of the instruction that was jumped to and program execution continues normally. In more complex CPUs, multiple instructions can be fetched, decoded and executed simultaneously. This section describes what is generally referred to as the "classic RISC pipeline", which is quite common among the simple CPUs used in many electronic devices (often called microcontroller). It largely ignores the important role of CPU cache, and therefore the access stage of the pipeline.

Some instructions manipulate the program counter rather than producing result data directly; such instructions are generally called "jumps" and facilitate program behavior like loops, conditional program execution (through the use of a conditional jump), and existence of functions.[c] In some processors, some other instructions change the state of bits in a "flags" register. These flags can be used to influence how a program behaves, since they often indicate the outcome of various operations. For example, in such processors a "compare" instruction evaluates two values and sets or clears bits in the flags register to indicate which one is greater

or whether they are equal; one of these flags could then be used by a later jump instruction to determine program flow.

Fetch:

The first step, fetch, involves retrieving an instruction (which is represented by a number or sequence of numbers) from program memory. The instruction's location (address) in program memory is determined by a program counter (PC), which stores a number that identifies the address of the next instruction to be fetched. After an instruction is fetched, the PC is incremented by the length of the instruction so that it will contain the address of the next instruction in the sequence.[d] Often, the instruction to be fetched must be retrieved from relatively slow memory, causing the CPU to stall while waiting for the instruction to be returned. This issue is largely addressed in modern processors by caches and pipeline architectures (see below).

Decode:

The instruction that the CPU fetches from memory determines what the CPU will do. In the decode step, performed by the circuitry known as the *instruction decoder*, the instruction is converted into signals that control other parts of the CPU.

The way in which the instruction is interpreted is defined by the CPU's instruction set architecture (ISA). Often, one group of bits (that is, a "field") within the instruction, called the opcode, indicates which operation is to be performed, while the remaining fields usually provide supplemental information required for the operation, such as the operands. Those operands may be specified as a constant value (called an immediate value), or as the location of a value that may be a processor register or a memory address, as determined by some addressing mode.

In some CPU designs the instruction decoder is implemented as a hardwired, unchangeable circuit. In others, a microprogram is used to translate instructions into sets of CPU configuration signals that are applied sequentially over multiple clock pulses. In some cases the memory that stores the microprogram is rewritable, making it possible to change the way in which the CPU decodes instructions.

Execute:

After the fetch and decode steps, the execute step is performed. Depending on the CPU architecture, this may consist of a single action or a sequence of actions. During each action, various parts of the CPU are electrically connected so they can perform all or part of the desired operation and then the action is completed, typically in response to a clock pulse. Very often the results are written to an internal CPU register for quick access by subsequent instructions. In other cases results may be written to slower, but less expensive and higher capacity main memory.

For example, if an addition instruction is to be executed, the arithmetic logic unit (ALU) inputs are connected to a pair of operand sources (numbers to be summed), the ALU is configured to perform an addition operation so that the sum of its operand inputs will appear at its output, and the ALU output is connected to storage (e.g., a register or memory) that will receive the sum. When the clock pulse occurs, the sum will be transferred to storage and, if the resulting sum is too large (i.e., it is larger than the ALU's output word size), an arithmetic overflow flag will be set.

Hardwired into a CPU's circuitry is a set of basic operations it can perform, called an instruction set. Such operations may involve, for example, adding or subtracting two numbers, comparing two numbers, or jumping to a different part of a program. Each basic operation is represented by a particular combination of bits, known as the machine language opcode; while executing instructions in a machine language program, the CPU decides which operation to perform by "decoding" the opcode. A complete machine language instruction consists of an opcode and, in many cases, additional bits that specify arguments for the operation (for example, the numbers to be summed in the case of an addition operation). Going up the complexity scale, a machine language program is a collection of machine language instructions that the CPU executes.

The actual mathematical operation for each instruction is performed by a combinational logic circuit within the CPU's processor known as the arithmetic logic unit or ALU. In general, a CPU executes an instruction by fetching it from memory, using its ALU to perform an operation, and then storing the result to memory. Beside the instructions for integer mathematics and logic operations, various other machine instructions exist, such as those for loading data from memory and storing it back, branching operations, and mathematical operations on floating-point numbers performed by the CPU's floating-point unit (FPU).

Control unit:

The control unit of the CPU contains circuitry that uses electrical signals to direct the entire computer system to carry out stored program instructions. The control unit does not execute program instructions; rather, it directs other parts of the system to do so. The control unit communicates with both the ALU and memory.

Arithmetic logic unit*:*

The arithmetic logic unit (ALU) is a digital circuit within the processor that performs integer arithmetic and bitwise logic operations. The inputs to the ALU are the data words to be operated on (called operands), status information from previous operations, and a code from the control unit indicating which operation to perform. Depending on the instruction being executed, the operands may come from internal CPU registers or external memory, or they may be constants generated by the ALU itself.

When all input signals have settled and propagated through the ALU circuitry, the result of the performed operation appears at the ALU's outputs. The result consists of both a data word, which may be stored in a register or memory, and status information that is typically stored in a special, internal CPU register reserved for this purpose.

Memory management unit (MMU) :

Most high-end microprocessors (in desktop, laptop, server computers) have a memory management unit, translating logical addresses into physical RAM addresses, providing memory protection and pagingabilities, useful for virtual memory. Simpler processors, especially microcontrollers, usually don't include an MMU.

Clock rate :

Most CPUs are synchronous circuits, which means they employ a clock signal to pace their sequential operations. The clock signal is produced by an external oscillator circuit that generates a consistent number of pulses each second in the form of a periodic square wave. The frequency of the clock pulses determines the rate at which a CPU executes instructions and, consequently, the faster the clock, the more instructions the CPU will execute each second.

To ensure proper operation of the CPU, the clock period is longer than the maximum time needed for all signals to propagate (move) through the CPU. In setting the clock period to a value well above the worst-case propagation delay, it is possible to design the entire CPU and the way it moves data around the "edges" of the rising and falling clock signal. This has the

advantage of simplifying the CPU significantly, both from a design perspective and a component-count perspective. However, it also carries the disadvantage that the entire CPU must wait on its slowest elements, even though some portions of it are much faster. This limitation has largely been compensated for by various methods of increasing CPU parallelism.

However, architectural improvements alone do not solve all of the drawbacks of globally synchronous CPUs. For example, a clock signal is subject to the delays of any other electrical signal. Higher clock rates in increasingly complex CPUs make it more difficult to keep the clock signal in phase (synchronized) throughout the entire unit. This has led many modern CPUs to require multiple identical clock signals to be provided to avoid delaying a single signal significantly enough to cause the CPU to malfunction. Another major issue, as clock rates increase dramatically, is the amount of heat that is dissipated by the CPU. The constantly changing clock causes many components to switch regardless of whether they are being used at that time. In general, a component that is switching uses more energy than an element in a static state. Therefore, as clock rate increases, so does energy consumption, causing the CPU to require more heat dissipation in the form of CPU cooling solutions.

One method of dealing with the switching of unneeded components is called clock gating, which involves turning off the clock signal to unneeded components (effectively disabling them). However, this is often regarded as difficult to implement and therefore does not see common usage outside of very low-power designs. One notable recent CPU design that uses extensive clock gating is the IBM PowerPC-based Xenon used in the Xbox 360; that way, power requirements of the Xbox 360 are greatly reduced.[60] Another method of addressing some of the problems with a global clock signal is the removal of the clock signal altogether. While removing the global clock signal makes the design process considerably more complex in many ways, asynchronous (or clockless) designs carry marked advantages in power consumption and heat dissipation in comparison with similar synchronous designs. While somewhat uncommon, entire asynchronous CPUs have been built without using a global clock signal. Two notable examples of this are the ARM compliant AMULET and the MIPS R3000 compatible MiniMIPS.

Rather than totally removing the clock signal, some CPU designs allow certain portions of the device to be asynchronous, such as using asynchronous ALUs in conjunction with superscalar pipelining to achieve some arithmetic performance gains. While it is not altogether clear whether totally asynchronous designs can perform at a comparable or better level than their synchronous counterparts, it is evident that they do at least excel in simpler math operations.

This, combined with their excellent power consumption and heat dissipation properties, makes them very suitable for embedded computers.

GPU:

A **graphics processing unit** (**GPU**) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles. Modern GPUs are very efficient at manipulating computer graphics and image processing. Their highly parallel structure makes them more efficient than general-purpose CPUsfor algorithms that process large blocks of data in parallel. In a personal computer, a GPU can be present on a video card or embedded on the motherboard. In certain CPUs, they are embedded on the CPU die.

The term GPU has been used from at least the 1980s. It was popularized by Nvidia in 1999, who marketed the GeForce 256 as "the world's first GPU". It was presented as a "single-chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines". Rival ATI Technologies coined the term "**visual processing unit**" or **VPU** with the release of the Radeon 9700 in 2002.

History:

1970s:

Arcade system boards have been using specialized graphics chips since the 1970s. In early video game hardware, the RAM for frame buffers was expensive, so video chips composited data together as the display was being scanned out on the monitor

Fujitsu's MB14241 video shifter was used to accelerate the drawing of sprite graphics for various 1970s arcade games from Taito and Midway, such as *Gun Fight* (1975), *Sea Wolf* (1976) and *Space Invaders*(1978). The Namco Galaxian arcade system in 1979 used specialized graphics hardware supporting RGB color, multi-colored sprites and tilemap backgrounds. The Galaxian hardware was widely used during the golden age of arcade video games, by game companies such as Namco, Centuri, Gremlin, Irem, Konami, Midway, Nichibutsu, Sega and Taito.

In the home market, the Atari 2600 in 1977 used a video shifter called the Television Interface Adaptor. The Atari 8-bit computers (1979) had ANTIC, a video processor which interpreted instructions describing a "display list"—the way the scan lines map to specific bitmapped or character modes and where the memory is stored (so there did not need to be a contiguous frame buffer). 6502 machine codesubroutines could be triggered on scan lines by setting a bit on a display list instruction. ANTIC also supported smooth vertical and horizontal scrolling independent of the CPU

1980:

The NEC μPD7220 was one of the first implementations of a graphics display controller as a single Large Scale Integration (LSI) integrated circuit chip, enabling the design of low-cost, high-performance video graphics cards such as those from Number Nine Visual Technology. It became one of the best known of what were known as graphics processing units in the 1980s.

The Williams Electronics arcade games *Robotron 2084*, *Joust*, *Sinistar*, and *Bubbles*, all released in 1982, contain custom blitter chips for operating on 16-color bitmaps.

In 1985, the Commodore Amiga featured a custom graphics chip, with a blitter unit accelerating bitmap manipulation, line draw, and area fill functions. Also included is a coprocessor (commonly referred to as "The Copper") with its own primitive instruction set, capable of manipulating graphics hardware registers in sync with the video beam (e.g. for per-scanline palette switches, sprite multiplexing, and hardware windowing), or driving the blitter.

In 1986, Texas Instruments released the TMS34010, the first microprocessor with on-chip graphics capabilities. It could run general-purpose code, but it had a very graphics-oriented instruction set. In 1990-1992, this chip would become the basis of the Texas Instruments Graphics Architecture ("TIGA") Windows accelerator cards.

In 1987, the IBM 8514 graphics system was released as one of[*vague*] the first video cards for IBM PC compatibles to implement fixed-function 2D primitives in electronic hardware. The same year, Sharpreleased the X68000, which used a custom graphics chipse that was powerful for a home computer at the time, with a 65,536 color palette and hardware support for sprites, scrolling and multiple playfields, eventually serving as a development machine for Capcom's CP System arcade board. Fujitsu later competed with the FM Towns computer, released in 1989 with support for a full 16,777,216 color palette.[21]

In 1988, the first dedicated polygonal 3D graphics boards were introduced in arcades with the Namco System 21 and Taito Air System

1990:

In 1991, S3 Graphics introduced the *S3 86C911*, which its designers named after the Porsche 911 as an indication of the performance increase it promised. The 86C911 spawned a host of imitators: by 1995, all major PC graphics chip makers had added 2D acceleration support to their chips. By this time, fixed-function *Windows accelerators* had surpassed expensive general-purpose graphics coprocessors in Windows performance, and these coprocessors faded away from the PC market.

Throughout the 1990s, 2D GUI acceleration continued to evolve. As manufacturing capabilities improved, so did the level of integration of graphics chips. Additional application programming interfaces (APIs) arrived for a variety of tasks, such as Microsoft's WinG graphics library for Windows 3.x, and their later DirectDraw interface for hardware acceleration of 2D games within Windows 95 and later.

In the early- and mid-1990s, real-time 3D graphics were becoming increasingly common in arcade, computer and console games, which led to an increasing public demand for hardware-accelerated 3D graphics. Early examples of mass-market 3D graphics hardware can be found in arcade system boards such as the Sega Model 1, Namco System 22, and Sega Model 2, and the fifth-generation video game consoles such as the Saturn, PlayStation and Nintendo 64. Arcade systems such as the Sega Model 2 and Namco Magic Edge Hornet Simulator in 1993 were capable of hardware T&L (transform, clipping, and lighting) years before appearing in consumer graphics cards. Some systems used DSPs to accelerate transformations. Fujitsu, which worked on the Sega Model 2 arcade system began working on integrating T&L into a single LSI solution for use in home computers in 1995; the Fujitsu Pinolite, the first 3D geometry processor for personal computers, released in 1997. The first hardware T&L GPU on home video game consoles was the Nintendo 64's Reality Coprocessor, released in 1996. In 1997, Mitsubishi released the 3Dpro/2MP, a fully featured GPU capable of transformation and lighting, for workstations and Windows NT desktops; ATi utilized it for their FireGL 4000 graphics card, released in 1997

In the PC world, notable failed first tries for low-cost 3D graphics chips were the S3 *ViRGE*, ATI *Rage*, and Matrox *Mystique*. These chips were essentially previous-generation 2D accelerators with 3D features bolted on. Many were even pin-compatible with the earlier-generation chips for ease of implementation and minimal cost. Initially, performance 3D graphics were possible only with discrete boards dedicated to accelerating 3D functions

(and lacking 2D GUI acceleration entirely) such as the PowerVR and the 3dfx *Voodoo*. However, as manufacturing technology continued to progress, video, 2D GUI acceleration and 3D functionality were all integrated into one chip. Rendition's *Verite* chipsets were among the first to do this well enough to be worthy of note. In 1997, Rendition went a step further by collaborating with Hercules and Fujitsu on a "Thriller Conspiracy" project which combined a Fujitsu FXG-1 Pinolite geometry processor with a Vérité V2200 core to create a graphics card with a full T&L engine years before Nvidia's GeForce 256. This card, designed to reduce the load placed upon the system's CPU, never made it to market

OpenGL appeared in the early '90s as a professional graphics API, but originally suffered from performance issues which allowed the Glide API to step in and become a dominant force on the PC in the late '90s However, these issues were quickly overcome and the Glide API fell by the wayside. Software implementations of OpenGL were common during this time, although the influence of OpenGL eventually led to widespread hardware support. Over time, a parity emerged between features offered in hardware and those offered in OpenGL. DirectX became popular among Windows game developers during the late 90s. Unlike OpenGL, Microsoft insisted on providing strict one-to-one support of hardware. The approach made DirectX less popular as a standalone graphics API initially, since many GPUs provided their own specific features, which existing OpenGL applications were already able to benefit from, leaving DirectX often one generation behind.

Over time, Microsoft began to work more closely with hardware developers, and started to target the releases of DirectX to coincide with those of the supporting graphics hardware. Direct3D 5.0 was the first version of the burgeoning API to gain widespread adoption in the gaming market, and it competed directly with many more-hardware-specific, often proprietary graphics libraries, while OpenGL maintained a strong following. Direct3D 7.0 introduced support for hardware-accelerated transform and lighting (T&L) for Direct3D, while OpenGL had this capability already exposed from its inception. 3D accelerator cards moved beyond being just simple rasterizers to add another significant hardware stage to the 3D rendering pipeline. The Nvidia *GeForce 256* (also known as NV10) was the first consumer-level card released on the market with hardware-accelerated T&L, while professional 3D cards already had this capability. Hardware transform and lighting, both already existing features of OpenGL, came to consumer-level hardware in the '90s and set the precedent for later pixel shader and vertex shader units which were far more flexible and programmable.

2000 to 2010:

Nvidia was first to produce a chip capable of programmable shading, the *GeForce 3* (code named NV20). Each pixel could now be processed by a short "program" that could include additional image textures as inputs, and each geometric vertex could likewise be processed by a short program before it was projected onto the screen. Used in the Xbox console, it competed with the PlayStation 2 (which used a custom vector DSP for hardware accelerated vertex processing; commonly referred to VU0/VU1). The earliest incarnations of shader execution engines used in Xbox were not general purpose and could not execute arbitrary pixel code. Vertices and pixels were processed by different units which had their own resources with pixel shaders having much tighter constraints (being as they are executed at much higher frequencies than with vertices). Pixel shading engines were actually more akin to a highly customizable function block and didn't really "run" a program. Many of these disparities between vertex and pixel shading wouldn't be addressed until much later with the Unified Shader Model.

By October 2002, with the introduction of the ATI *Radeon 9700* (also known as R300), the world's first Direct3D 9.0 accelerator, pixel and vertex shaders could implement looping and lengthy floating pointmath, and were quickly becoming as flexible as CPUs, yet orders of magnitude faster for image-array operations. Pixel shading is often used for bump mapping, which adds texture, to make an object look shiny, dull, rough, or even round or extruded

With the introduction of the GeForce 8 series, which was produced by Nvidia, and then new generic stream processing unit GPUs became a more generalized computing device. Today, parallel GPUs have begun making computational inroads against the CPU, and a subfield of research, dubbed GPU Computing or GPGPU for *General Purpose Computing on GPU*, has found its way into fields as diverse as machine learning,[38] oil exploration, scientific image processing, linear algebra, statistics, 3D reconstruction and even stock options pricing determination. GPGPU at the time was the precursor to what we now call compute shaders (e.g. CUDA, OpenCL, DirectCompute) and actually abused the hardware to a degree by treating the data passed to algorithms as texture maps and executing algorithms by drawing a triangle or quad with an appropriate pixel shader. This obviously entails some overheads since we involve units like the Scan Converter where they aren't really needed (nor do we even care about the triangles, except to invoke the pixel shader). Over the years, the energy consumption of GPUs has increased and to manage it, several techniques have been proposed.

Nvidia's CUDA platform, first introduced in 2007, was the earliest widely adopted programming model for GPU computing. More recently OpenCL has become broadly supported. OpenCL is an open standard defined by the Khronos Group which allows for the development of code for both GPUs and CPUs with an emphasis on portability OpenCL solutions are supported by Intel, AMD, Nvidia, and ARM, and according to a recent report by Evan's Data, OpenCL is the GPGPU development platform most widely used by developers in both the US and Asia Pacific.[*citation needed*]

2010 to present:

In 2010, Nvidia began a partnership with Audi to power their cars' dashboards. These Tegra GPUs were powering the cars' dashboard, offering increased functionality to cars' navigation and entertainment systems Advancements in GPU technology in cars has helped push self-driving technology. AMD's Radeon HD 6000 Series cards were released in 2010 and in 2011, AMD released their 6000M Series discrete GPUs to be used in mobile devices. The Kepler line of graphics cards by Nvidia came out in 2012 and were used in the Nvidia's 600 and 700 series cards. A new feature in this new GPU microarchitecture included GPU boost, a technology adjusts the clock-speed of a video card to increase or decrease it according to its power draw. The Kepler microarchitecture was manufactured on the 28 nm process.

The PS4 and Xbox One were released in 2013, they both use GPUs based on AMD's Radeon HD 7850 and 7790. Nvidia's Kepler line of GPUs was followed by the Maxwell line, manufactured on the same process. 28 nm chips by Nvidia were manufactured by TSMC, the Taiwan Semiconductor Manufacturing Company, that was manufacturing using the 28 nm process at the time. Compared to the 40 nm technology from the past, this new manufacturing process allowed a 20 percent boost in performance while drawing less power headsets like the Oculus Rift and the HTC Vive have very high system requirements. VR headset manufacturers recommended the GTX 970 and the R9 290X or better at the time of their release. Pascal is the next generation of consumer graphics cards by Nvidia released in 2016. The GeForce 10 series of cards are under this generation of graphics cards. They are made using the 16 nm manufacturing process which improves upon previous microarchitectures. Nvidia has released one non-consumer card under the new Volta architecture, the Titan V. Changes from the Titan XP, Pascal's high-end card, include an increase in the number of CUDA cores, the addition of tensor cores, and high-bandwidth memory. Tensor cores are cores specially designed for deep learning, while high-bandwidth memory is on-die, stacked, lower-clocked memory that offers an extremely wide

memory bus that is useful for the Titan V's intended purpose. To emphasize that the Titan V is not a gaming card, Nvidia removed the "Geforce GTX" suffix it adds to consumer gaming cards. A new generation, the RTX Turing GPUs were unveiled on August 20, 2018 that add ray-tracing cores to GPUs, improving their performance on lighting effects. Polaris 11 and Polaris 10 GPUs from AMD are fabricated a 14-nanometer process. Their release results in a substantial increase in the performance per watt of AMD video cards. AMD has also released for its high-end market the Vega GPUs, which also feature high-bandwidth memory like the Titan V.

Computational functions:

Modern GPUs use most of their transistors to do calculations related to 3D computer graphics. In addition to the 3D hardware, today's GPUs include basic 2D acceleration and framebuffer capabilities (usually with a VGA compatibility mode). Newer cards like AMD/ATI HD5000-HD7000 even lack 2D acceleration; it has to be emulated by 3D hardware. GPUs were initially used to accelerate the memory-intensive work of texture mapping and rendering polygons, later adding units to accelerate geometric calculations such as the rotation and translation of vertices into different coordinate systems. Recent developments in GPUs include support for programmable shaders which can manipulate vertices and textures with many of the same operations supported by CPUs, oversampling and interpolation techniques to reduce aliasing, and very high-precision color spaces. Because most of these computations involve matrix and vector operations, engineers and scientists have increasingly studied the use of GPUs for non-graphical calculations; they are especially suited to other embarrassingly parallel problems.

With the emergence of deep learning, the importance of GPUs has increased. In research done by Indigo, it was found that while training deep learning neural networks, GPUs can be 250 times faster than CPUs. The explosive growth of Deep Learning in recent years has been attributed to the emergence of general purpose GPUs There has been some level of competition in this area with ASICs, most prominently the Tensor Processing Unit (TPU) made by Google. However, these can require changes to existing code and GPUs are still very popular.

GPU accelerated video decoding :

Most GPUs made since 1995 support the YUV color space and hardware overlays, important for digital video playback, and many GPUs made since 2000 also support MPEG primitives such as motion compensation and iDCT. This process of hardware accelerated video decoding, where portions of the video decoding process and video post-processing are offloaded to the GPU hardware, is commonly referred to as "GPU accelerated video decoding", "GPU assisted video decoding", "GPU hardware accelerated video decoding" or "GPU hardware assisted video decoding".

More recent graphics cards even decode high-definition video on the card, offloading the central processing unit. The most common APIs for GPU accelerated video decoding are DxVA for Microsoft Windows operating system and VDPAU, VAAPI, XvMC, and XvBA for Linux-based and UNIX-like operating systems. All except XvMC are capable of decoding videos encoded with MPEG-1, MPEG-2, MPEG-4 ASP (MPEG-4 Part 2), MPEG-4 AVC (H.264 / DivX 6), VC-1, WMV3/WMV9, Xvid / OpenDivX (DivX 4), and DivX 5 codecs, while XvMC is only capable of decoding MPEG-1 and MPEG-2.

The video decoding processes that can be accelerated by today's modern GPU hardware are:

- Motion compensation (mocomp)
- Inverse discrete cosine transform (iDCT)
  - Inverse telecine 3:2 and 2:2 pull-down correction
- Inverse modified discrete cosine transform (iMDCT)
- In-loop deblocking filter
- Intra-frame prediction
- Inverse quantization (IQ)
- Variable-length decoding (VLD), more commonly known as slice-level acceleration
- Spatial-temporal deinterlacing and automatic interlace/progressive source detection
- Bitstream processing (Context-adaptive variable-length coding/Context-adaptive binary arithmetic coding) and perfect pixel positioning.

3.4 CODE SNIPPETS:

```
#DATA AUGMENTATION
from keras.preprocessing.image import ImageDataGenerator  #importing ImageDataGenerator class from keras library

train_data=ImageDataGenerator(rescale=1./255,
                              rotation_range=15,
                              brightness_range=(0.5,0.8),
                              horizontal_flip=True)  #Creating an object of ImageDataGenerator class with rescaling

train_set=train_data.flow_from_directory('E:\\major project\\major project images\\data\\training',  #Directory where training images are present
                                         target_size=(128,128),                                       #size after resizing
                                         color_mode='rgb',                                            #color mode of the image
                                         class_mode='binary',                                         #number of classes is 2 so binary
                                         batch_size=30,
                                         #save_to_dir='E:\\major project\\major project images\\augmented_data\\trained_data',
                                         #save_prefix='augtrain',
                                         #save_format='png',
                                         interpolation='nearest')

#for i in range(600):
#   train_set._get_batches_of_transformed_samples([i])

test_data=ImageDataGenerator(rescale=1./255)

test_set=test_data.flow_from_directory('E:\\major project\\major project images\\data\\test',
                                       target_size=(128,128),
                                       color_mode='rgb',
                                       class_mode='binary',
                                       batch_size=20,
                                       shuffle=True,
                                       #save_to_dir='E:\\major project\\major project images\\augmented_data\\tested_data',
                                       #save_prefix='augtest',
                                       interpolation='nearest')
```

Fig 3.1 Initializing objects and including libraries.

```

#CREATION OF DATA MODEL
from keras.models import Sequential
from keras.layers import Conv2D #importing sequential class from keras library
from keras.layers import MaxPooling2D #importing MaxPooling2D from keras
from keras.layers import Flatten #importing flatten
#from keras.layers import Dropout
from keras.layers import Dense
from keras.optimizers import Adam

model=Sequential()             #Creating an object of Sequential class
model.add(Conv2D(16,(3,3),activation='tanh',input_shape=(128,128,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
#model.add(Dropout(0.5))
model.add(Conv2D(16,(3,3),activation='tanh'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(16,(3,3),activation='tanh'))
model.add(MaxPooling2D(pool_size=(2,2)))
#model.add(Dropout(0.5))
model.add(Conv2D(8,(3,3),activation='tanh'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(8,(3,3),activation='tanh' ))
model.add(MaxPooling2D(pool_size=(2,2)))
#model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(8,activation='tanh'))
model.add(Dense(4,activation='tanh'))
model.add(Dense(1,activation='sigmoid'))
summary=model.summary()
```

Fig 3.2 Model creation and importing layers.

```
from contextlib import redirect_stdout

with open('C:\\Users\\sid\\Desktop\\1.txt', 'w') as f:
    with redirect_stdout(f):
        model.summary()
#CONFIGURING THE MODEL
model.compile(optimizer=Adam(),loss='binary_crossentropy',metrics=['accuracy'])
#Training the model
#model.fit_generator(train_set,steps_per_epoch=40,epochs=30,validation_data=test_set,validation_steps=30)
import matplotlib.pyplot as plt

x=model.fit_generator(train_set,steps_per_epoch=40,epochs=1,validation_data=test_set,validation_steps=30,verbose=1)
from keras.utils import plot_model
plot_model(model, to_file='C:\\Users\\sid\\Desktop\\model.png',show_shapes=True)
#q=model.layers[2].get_weights()
#print(q)
# Plot training & validation accuracy values
plt.plot(x.history['acc'])
plt.plot(x.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
fig=plt.gcf()
plt.show()
fig.savefig("C:\\Users\\sid\\Desktop\\1.png")

# Plot training & validation loss values
plt.plot(x.history['loss'])
plt.plot(x.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
fig1=plt.gcf()
plt.show()
fig1.savefig("C:\\Users\\sid\\Desktop\\2.png")
```

Fig 3.3 Training and fitting the model, plotting metrics.

```
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
#import numpy as np
import os
import shutil
import pandas as pd
#from IPython.display import display, Image
from IPython.display import Image,display
model=load_model("E:\\major project\\Model\\model.hdf5")
predict_data=ImageDataGenerator(rescale=1./255)
predict_set=predict_data.flow_from_directory(r"E:\major project\major project images\data\images",target_size=(128,128),class_mode=None,shuffle=False)
#predict_set.reset()
predictprob=model.predict_generator(predict_set,steps=len(predict_set),verbose=1)
print(predictprob)
path, dirs, files = next(os.walk(r"E:\major project\major project images\data\images\predict"))
image_count = len(files)
print(image_count)
predict=[]
labels=[]
```

Fig 3.4 Importing the model and libraries.

```
for i in range(image_count):
    if predictprob[i] > 0.5:
        prediction = "non-flooded"
    else:
        prediction = "flooded"
    predict.append(prediction)
#labels=labels.append(["Image"+str(i) for i in range(image_count)])
df=pd.DataFrame(predict,columns=['PREDICTIONS'])
df
writer=pd.ExcelWriter(r"E:\major project\predicted data\predictions.xlsx")
df.to_excel(writer)
writer.save()
# Function to rename multiple files
from shutil import move
name=r'E:\major project\major project images\data\rename\predict'
name1=r'E:\major project\major project images\data\images\predict'
```

Fig 3.5 .Predicting the probabilities.

```
j = 0
for file in os.listdir(name):
    full_name=os.path.join(name,file)
    os.remove(full_name)
for file1 in os.listdir(name1):
    full_name1=os.path.join(name1,file1)
    shutil.copy(full_name1,name)
for filename in os.listdir(name):
    dst ='\\'+ predict[j] + str(j)+ ".jpg"
    src =name+'\\'+ filename
    dst =name+ dst
    move(src,dst)
    j += 1
for files in os.listdir(name):
    img=Image(name+'\\'+files)
    display(img)
    print(files)
```

Fig 3.6 Predicting the images against probability and saving them in folders.

4.SCREENSOTS

```
In [1]: runfile('E:/major project/project code.py', wdir='E:/major project')
Using TensorFlow backend.
Found 600 images belonging to 2 classes.
Found 400 images belonging to 2 classes.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 126, 126, 16)      448
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 61, 61, 16)        2320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 16)        2320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 8)         1160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 8)           0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 8)           584
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 8)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 264
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 36
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 5
=================================================================
Total params: 7,137
Trainable params: 7,137
Non-trainable params: 0
_________________________________________________________________
Epoch 1/1
40/40 [==============================] - 25s 631ms/step - loss: 0.6753 - acc: 0.5925 - val_loss: 0.6188 - val_acc: 0.7367
```

```
Found 11 images belonging to 1 classes.
1/1 [==============================] - 0s 475ms/step
[[0.11315729]
 [0.9414911 ]
 [0.45975667]
 [0.9866117 ]
 [0.98831266]
 [0.02580504]
 [0.98668075]
 [0.01328536]
 [0.01580959]
 [0.01378887]
 [0.9760593 ]]
11
```

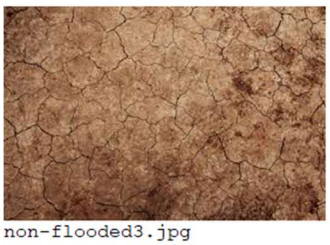
flooded0.jpg

flooded2.jpg

flooded9.jpg

non-flooded1.jpg

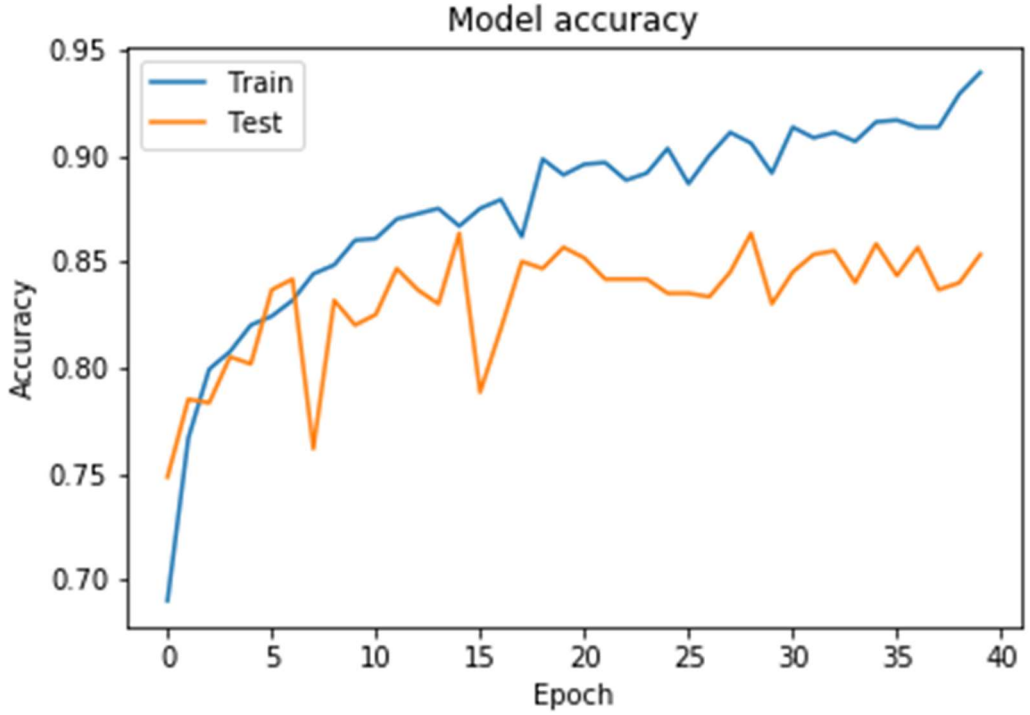
non-flooded10.jpg

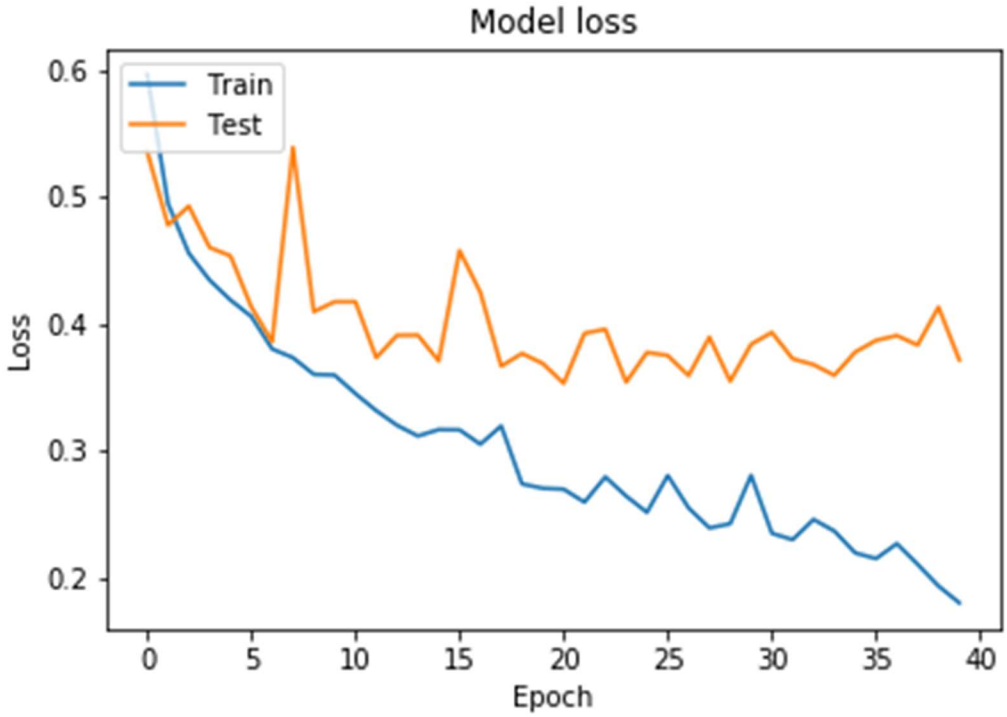


non-flooded3.jpg

non-flooded4.jpg

non-flooded6.jpg


Model accuracy
Train
Test
Accuracy
Epoch
0.95
0.90
0.85
0.80
0.75
0.70
0
5
10
15
20
25
30
35
40


Model loss
Train
Test
Loss
Epoch
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25
30
35
40

5.CONCLUSION

The entire project has been developed and deployed as per the requirements stated, it is found to be bug free as per the tested standards that are implemented. Any specification untraced errors will be concentrated in the coming versions, which are planned to be developed. This is a novel architecture for classifying and predicting the flooded and non-flooded areas. The captured images from drone are collected and processed through the deep neural network model created and this helps the machine to learn itself to classify the images accordingly with a good accuracy percentage of approximately 85%.

6.BENEFITS

- It uses deep neural networks. SO, the machine can learn itself without the need of algorithms.
- Better accuracy compared to any other image classification algorithms.
- Less Computational power required.
- Drones which are cheaper and easy to use allow for easy capturing of images and thus images can be easily collected.
- Due to continuous surveillance, this can help flood prone areas constantly monitoring and classifying.
- It can save man power and can do work continuously in an automated way.

7.BIBLIOGRAPHY

1. www.keras.io which gives the keras documentation, information on the layers and generators and various functions.

2.https://www.tensorflow.org which is used for backend reference.

3. https://www.wikipedia.org which is used for Neural Networks reference.

4.https://www.anaconda.com is used to provide documentation and libraries for anaconda IDE.