**INTELLIGENT BREAST CANCER DIAGNOSIS THROUGH DEEP LEARNING SEGMENTATION**

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **SL.NO** | **TITLE** | **PAGE.NO** |
| **01** | **ABSTRACT** |  |
| **02** | **EXISTING SYSTEM**  2.1 DRAWBACKS |  |
| **03** | **INTRODUCTION**  3.1 DATA SCIENCE  3.2 ARTIFICIAL INTELLIGENCE |  |
| **04** | **MACHINE LEARNING** |  |
| **05** | **PREPARING DATASET** |  |
| **06** | **PROPOSED SYSTEM**  6.1 MERITS |  |
| **07** | **LITERATURE SURVEY** |  |
| **08** | **SYSTEM STUDY**  8.1 OBJECTIVES  8.2 PROJECT GOAL  8.3 SCOPE OF THE PROJECT |  |
| **09** | **FEASIBILITY STUDY** |  |
| **10** | **LIST OF MODULES** |  |
| **11** | **PROJECT REQUIREMENTS**  11.1 FUNCTIONAL REQUIREMENTS  11.2 NON-FUNCTIONAL REQUIREMENTS |  |
| **12** | **ENVIRONMENT REQUIREMENT** |  |
| **13** | **SOFTWARE DESCRIPTION**  13.1 ANACONDA NAVIGATOR  13.2 JUPYTER NOTEBOOK |  |
| **14** | **PYTHON** |  |
| **15** | **SYSTEM ARCHITECTURE** |  |
| **16** | **WORKFLOW DIAGRAM** |  |
| **17** | **USECASE DIAGRAM** |  |
| **18** | **CLASS DIAGRAM** |  |
| **19** | **ACTIVITY DIAGRAM** |  |
| **20** | **SEQUENCE DIAGRAM** |  |
| **21** | **ER – DIAGRAM** |  |
| **22** | **MODULE DESCRIPTION**  22.1 MODULE DIAGRAM  22.2 MODULE GIVEN INPUT EXPECTED OUTPUT |  |
| **23** | **DEPLOYMENT** |  |
| **24** | **HTML** |  |
| **25** | **CSS** |  |
| **26** | **CODING** |  |
| **27** | **CONCLUSION** |  |
| **28** | **FUTURE WORK** |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **SL.NO** | **TITLE** | **PAGE.NO** |
| **01** | **SYSTEM ARCHITECTURE** |  |
| **02** | **WORKFLOW DIAGRAM** |  |
| **03** | **USECASE DIAGRAM** |  |
| **04** | **CLASS DIAGRAM** |  |
| **05** | **ACTIVITY DIAGRAM** |  |
| **06** | **SEQUENCE DIAGRAM** |  |
| **07** | **ER – DIAGRAM** |  |
| **08** | **MODULE DIAGRAM** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NOTATION**  **NAME** | **NOTATION** | **DESCRIPTION** |
| 1. | Class | *Class Name*  *-attribute*  *-attribute*  *+operation*  *+operation*  *+operation*  *+public*  *-private*  *# protected* | Represents a collection of similar entities grouped together. |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 2. | Association | nAME  Class B  Class A    Class B  Class A | Associations represents static relationships between classes. Roles represents the way the two classes see each other. |
| 3. | Actor | Class A  Class A  Class B  Class B | It aggregates several classes into a single classes. |
| 4. | Aggregation | Interaction between the system and external environment |

|  |  |  |  |
| --- | --- | --- | --- |
| 5. | Relation  (uses) | uses | Used for additional process communication. |
| 6. | Relation  (extends) | EXTENDS | Extends relationship is used when one use case is similar to another use case but does a bit more. |
| 7. | Communication |  | Communication between various use cases. |
| 8. | State | State | State of the process. |
| 9. | Initial State |  | Initial state of the object |
| 10. | Final state |  | Final state of the object |
| 11. | Control flow |  | Represents various control flow between the states. |
| 12. | Decision box |  | Represents decision making process from a constraint |
| 13. | Usecase |  | Interact ion between the system and external environment. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14. | Component |  | | Represents physical modules which is a collection of components. |
| 15. | Node | |  | Represents physical modules which are a collection of components. |
| 16. | Data Process/State | |  | A circle in DFD represents a state or process which has been triggered due to some event or action. |
| 17. | External entity | |  | Represents external entities such as keyboard,sensors,etc. |
| 18. | Transition | |  | Represents communication that occurs between processes. |
| 19. | Object Lifeline | |  | Represents the vertical dimensions that the object communications. |
| 20. | Message | | Message | Represents the message exchanged. |

**INTELLIGENT BREAST CANCER DIAGNOSIS THROUGH DEEP LEARNING SEGMENTATION**

1. **ABSTRACT:**

Breast cancer remains one of the most prevalent and deadly forms of cancer worldwide, necessitating innovative approaches for early detection and treatment. The advent of deep learning has revolutionized medical imaging, particularly in the field of breast cancer diagnosis. This research explores the application of deep learning techniques in breast cancer image processing and segmentation. Utilizing convolutional neural networks (CNNs), the study aims to enhance the accuracy and efficiency of identifying malignant tumors in ultrasound images and other breast imaging modalities. By leveraging large datasets and advanced image processing algorithms, deep learning models are trained to differentiate between healthy and cancerous tissues with high precision. This approach not only improves diagnostic capabilities but also aids in the segmentation of tumors, which is crucial for treatment planning and monitoring. The integration of deep learning in breast cancer detection holds significant promise for reducing false positives and negatives, ultimately contributing to better patient outcomes and personalized care.

**Keywords**: Breast cancer, Classification, Segmentation, Artificial intelligence (AI), Machine learning, Deep learning, Texture analysis, Shape analysis

**2. Existing System:**

The incidence of breast cancer is increasing rapidly around the world. Accurate classification of the breast cancer subtype from hematoxylin and eosin images is the key to improve the precision of treatment. However, the high consistency of disease subtypes and uneven distribution of cancer cells seriously affect the performance of multi-classification methods. Furthermore, it is difficult to apply existing classification methods to multiple datasets. In this article, we propose a collaborative transfer network (CTransNet) for multi-classification of breast cancer histopathological images. CTransNet consists of a transfer learning backbone branch, a residual collaborative branch, and a feature fusion module. The transfer learning branch adopts the pre-trained DenseNet structure to extract image features from ImageNet. The residual branch extracts target features from pathological images in a collaborative manner. The feature fusion strategy of optimizing these two branches is used to train and fine-tune CTransNet. Experiments show that CTransNet achieves 98.29% classification accuracy on the public BreaKHis breast cancer dataset, exceeding the performance of state-of-the-art methods. Visual analysis is carried out under the guidance of oncologists. Based on the training parameters of the BreaKHis dataset, CTransNet achieves superior performance on other two public breast cancer datasets (breast-cancer-grade-ICT and ICIAR2018\_BACH\_Challenge), indicating that CTransNet has good generalization performance.

**DISADVANTAGES:**

**High Consistency of Disease Subtypes**: The similarity between different subtypes of breast cancer can make it challenging to distinguish between them accurately.

**Uneven Distribution of Cancer Cells**: The irregular distribution of cancer cells in histopathological images can adversely impact the effectiveness of multi-classification methods.

**Difficulty in Applying Existing Methods to Multiple Datasets**: Existing classification methods often struggle to perform consistently across various datasets, limiting their generalizability.

**3. INTRODUCTION**

Intelligent Breast Cancer Diagnosis through Deep Learning Segmentation using LeNet and U-Net Architecture\*\* represents a significant advancement in medical imaging and artificial intelligence. Breast cancer is one of the most prevalent cancers among women globally, and early, accurate diagnosis is crucial for effective treatment and improved patient outcomes. Traditional diagnostic methods, such as ultrasound images, can be limited by human interpretation, variability, and the quality of imaging. Leveraging deep learning, particularly Convolutional Neural Networks (CNNs), offers the ability to automate and enhance the diagnostic process. In this context, LeNet and U-Net architectures are pivotal. LeNet, a pioneering CNN model, excels in image classification, while U-Net, a more advanced network, is tailored for image segmentation, making it ideal for identifying and isolating cancerous regions in medical images. By combining these architectures, the system can both detect (classify) and precisely locate (segment) tumor areas within breast tissue. This integration of deep learning techniques aims to assist radiologists, reduce false positives/negatives, and provide a faster, more accurate diagnostic tool for breast cancer screening and treatment planning. As a result, such intelligent diagnostic systems could revolutionize the healthcare industry, offering more reliable and consistent results in cancer detection and improving survival rates through early intervention.

**3.1 Data Science:**

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains.

The term "data science" has been traced back to 1974, when Peter Naur proposed it as an alternative name for computer science. In 1996, the International Federation of Classification Societies became the first conference to specifically feature data science as a topic. However, the definition was still in flux.

The term “data science” was first coined in 2008 by D.J. Patil, and Jeff Hammer bacher, the pioneer leads of data and analytics efforts at LinkedIn and Facebook. In less than a decade, it has become one of the hottest and most trending professions in the market.

Data science is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data.

Data science can be defined as a blend of mathematics, business acumen, tools, algorithms and machine learning techniques, all of which help us in finding out the hidden insights or patterns from raw data which can be of major use in the formation of big business decisions.

**Data Scientist:**

Data scientists examine which questions need answering and where to find the related data. They have business acumen and analytical skills as well as the ability to mine, clean, and present data. Businesses use data scientists to source, manage, and analyze large amounts of unstructured data.

**Required Skills for a Data Scientist:**

* **Programming**: Python, SQL, Scala, Java, R, MATLAB.
* **Machine Learning**: Natural Language Processing, Classification, Clustering.
* **Data Visualization**: Tableau, SAS, D3.js, Python, Java, R libraries.
* **Big data platforms**: MongoDB, Oracle, Microsoft Azure, Cloudera.

**3.2 ARTIFICIAL INTELLIGENCE**:

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving.

Artificial intelligence (AI) is [intelligence](https://en.wikipedia.org/wiki/Intelligence) demonstrated by [machines](https://en.wikipedia.org/wiki/Machine), as opposed to the natural intelligence [displayed by humans](https://en.wikipedia.org/wiki/Human_intelligence) or [animals](https://en.wikipedia.org/wiki/Animal_cognition). Leading AI textbooks define the field as the study of “[intelligent agents](https://en.wikipedia.org/wiki/Intelligent_agent)” any system that perceives its environment and takes actions that maximize its chance of achieving its goals.

Some popular accounts use the term “artificial intelligence” to describe machines that mimic “cognitive” functions that humans associate with the [human mind](https://en.wikipedia.org/wiki/Human_mind), such as “learning” and “problem solving”, however this definition is rejected by major AI researchers.

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processing, speech recognition and machine vision.

AI applications include advanced web search engines, recommendation systems (used by Youtube, Amazon and Netflix), Understanding human speech (such as Siri or Alexa), self-driving cars (e.g. Tesla), and competing at the highest level in strategic game systems (such as chess and Go), As machines become increasingly capable, tasks considered to require “intelligence” are often removed from the definition of AI, a phenomenon known as the AI effect. For instance, optical character recognition is frequently excluded from things considered to be AI, having become a routine technology.

Artificial intelligence was founded as an academic discipline in 1956, and in the years since has experienced several waves of optimism, followed by disappointment and the loss of funding (known as an “AI winter”), followed by new approaches, success and renewed funding.

AI research has tried and discarded many different approaches during its lifetime, including simulating the Breast, modeling human problem solving, formal logic, large databases of knowledge and imitating animal behavior. In the first decades of the 21st century, highly mathematical statistical machine learning has dominated the field, and this technique has proved highly successful, helping to solve many challenging problems throughout industry and academia.

The various sub-fields of AI research are centered around particular goals and the use of particular tools. The traditional goals of AI research include reasoning, knowledge representation, planning, learning, natural language processing, perception and the ability to move and manipulate objects. General intelligence (the ability to solve an arbitrary problem) is among the field’s long-term goals.

To solve these problems, AI researchers use versions of search and mathematical optimization, formal logic, artificial neural networks, and methods based on statistics, probability and economics. AI also draws upon computer science, psychology, linguistics, philosophy, and many other fields.

The field was founded on the assumption that human intelligence “can be so precisely described that a machine can be made to simulate it”. This raises philosophical arguments about the mind and the ethics of creating artificial beings endowed with human-like intelligence.

These issues have been explored by myth, fiction and philosophy since antiquity. Science fiction and futurology have also suggested that, with its enormous potential and power, AI may become an existential risk to humanity.

As the hype around AI has accelerated, vendors have been scrambling to promote how their products and services use AI. Often what they refer to as AI is simply one component of AI, such as machine learning.

AI requires a foundation of specialized hardware and software for writing and training machine learning algorithms. No one programming language is synonymous with AI, but a few, including Python, R and Java, are popular.

In general, AI systems work by ingesting large amounts of labeled training data, analyzing the data for correlations and patterns, and using these patterns to make predictions about future states.

In this way, a chatbot that is fed examples of text chats can learn to produce life like exchanges with people, or an image recognition tool can learn to identify and describe objects in images by reviewing millions of examples.

AI programming focuses on three cognitive skills: learning, reasoning and self-correction.

**Learning processes.** This aspect of AI programming focuses on acquiring data and creating rules for how to turn the data into actionable information. The rules, which are called algorithms, provide computing devices with step-by-step instructions for how to complete a specific task.

**Reasoning processes.** This aspect of AI programming focuses on choosing the right algorithm to reach a desired outcome.

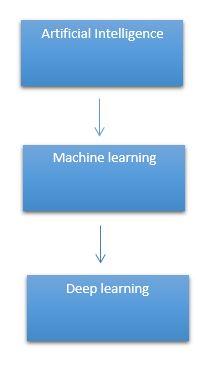
**Self-correction processes.** This aspect of AI programming is designed to continually fine-tune algorithms and ensure they provide the most accurate results possible.

AI is important because it can give enterprises insights into their operations that they may not have been aware of previously and because, in some cases, AI can perform tasks better than humans. Particularly when it comes to repetitive, detail-oriented tasks like analyzing large numbers of legal documents to ensure relevant fields are filled in properly, AI tools often complete jobs quickly and with relatively few errors.

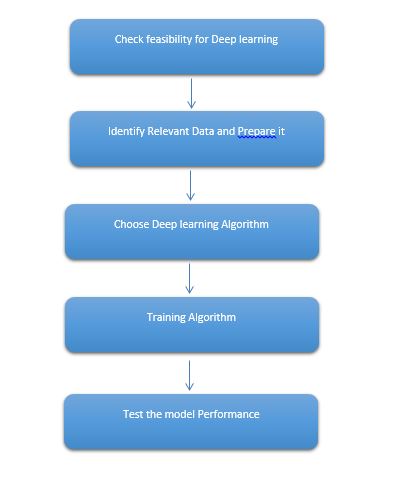
Artificial neural networks and deep learning artificial intelligence technologies are quickly evolving, primarily because AI processes large amounts of data much faster and makes predictions more accurately than humanly possible.

**4. DEEP LEARNING**

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human disease so deep learning is also a kind of mimic of human disease. It’s on hype nowadays because earlier we did not have that much processing power and a lot of data. A formal definition of deep learning is- neurons Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. In disease approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousands of their neighbors. The question here is how it recreates these neurons in a computer. So, it creates an artificial structure called an artificial neural net where we have nodes or neurons. It has some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.



# It need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). It needs to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Choose the Deep Learning Algorithm appropriately. Algorithm should be used while training the dataset. Final testing should be done on the dataset



Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological disease. Specifically, neural networks tend to be static and symbolic, while the biological disease of most living organisms is dynamic (plastic) and analogue.

The adjective "deep" in deep learning refers to the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, but that a network with a non-polynomial activation function with one hidden layer of unbounded width can. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions. In deep learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

Deep learning is a class of [machine learning](https://en.wikipedia.org/wiki/Machine_learning) [algorithms](https://en.wikipedia.org/wiki/Algorithm) that uses multiple layers to progressively extract higher-level features from the raw input. For example, in [image processing](https://en.wikipedia.org/wiki/Image_processing), lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

**Interpretations:**

Deep neural networks are generally interpreted in terms of the universal approximation theorem or probabilistic inference.

The classic universal approximation theorem concerns the capacity of feed-forward neural networks with a single hidden layer of finite size to approximate continuous functions. In 1989, the first proof was published by George Cybenko for sigmoid activation functions and was generalised to feed-forward multi-layer architectures in 1991 by Kurt Hornik. Recent work also showed that universal approximation also holds for non-bounded activation functions such as the rectified linear unit.

The universal approximation theorem for deep neural networks concerns the capacity of networks with bounded width but the depth is allowed to grow proved that if the width of a deep neural network with ReLU activation is strictly larger than the input dimension, then the network can approximate any Lebesgue integrable function; If the width is smaller or equal to the input dimension, then deep neural network is not a universal approximator.

The probabilistic interpretation derives from the field of machine learning. It features inference, as well as the optimization concepts of training and testing, related to fitting and generalization, respectively. More specifically, the probabilistic interpretation considers the activation nonlinearity as a cumulative distribution function. The probabilistic interpretation led to the introduction of dropout as regularizer in neural networks. The probabilistic interpretation was introduced by researchers including Hopfield, Widrow and Narendra and popularized in surveys such as the one by Bishop.

### Deep learning revolution:

### In 2012, a team led by George E. Dahl won the "Merck Molecular Activity Challenge" using multi-task deep neural networks to predict the biomolecular target of one drug. In 2014, Hochreiter's group used deep learning to detect off-target and toxic effects of environmental chemicals in nutrients, household products and drugs and won the "Tox21 Data Challenge" of NIH, FDA and NCATS.

Significant additional impacts in image or object recognition were felt from 2011 to 2012. Although CNNs trained by back-propagation had been around for decades, and GPU implementations of NNs for years, including CNNs, fast implementations of CNNs on GPUs were needed to progress on computer vision. In 2011, this approach achieved for the first time superhuman performance in a visual pattern recognition contest. Also in 2011, it won the ICDAR Chinese handwriting contest, and in May 2012, it won the ISBI image segmentation contest. Until 2011, CNNs did not play a major role at computer vision conferences, but in June 2012, a paper by Ciresan et al. at the leading conference CVPR showed how max-pooling CNNs on GPU can dramatically improve many vision benchmark records.

In October 2012, a similar system by Krizhevsky et al. won the large-scale ImageNet competition by a significant margin over shallow machine learning methods. In November 2012, Ciresan et al.'s system also won the ICPR contest on analysis of large medical images for cancer detection, and in the following year also the MICCAI Grand Challenge on the same topic. In 2013 and 2014, the error rate on the ImageNet task using deep learning was further reduced, following a similar trend in large-scale speech recognition.

Image classification was then extended to the more challenging task of generating descriptions (captions) for images, often as a combination of CNNs and LSTMs.

Some researchers state that the October 2012 ImageNet victory anchored the start of a "deep learning revolution" that has transformed the AI industry.

In March 2019, Yoshua Bengio, Geoffrey Hinton and Yann LeCun were awarded the Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

**5. MACHINE LEARNING:**

It seems like your request is quite brief. "ML" typically stands for "Machine Learning." Machine learning is a subset of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to perform a specific task without being explicitly programmed for that task. If you have a specific question or topic related to machine learning that you'd like more information about, please provide more details, and I'll do my best to assist you!

Top of Form

**6. PROPOSED SYSTEM:**

The proposed system leverages advanced deep-learning techniques for the segmentation and detection of breast cancer from medical imaging data, specifically ultrasound images. This system integrates convolutional neural networks (CNNs) with sophisticated image processing algorithms to enhance the accuracy and efficiency of detecting malignant regions. The workflow begins with the acquisition of high-resolution ultrasound images, followed by preprocessing steps such as noise reduction, contrast enhancement, and normalization. These preprocessed images are then fed into a CNN-based segmentation model, which has been trained on a large dataset of annotated breast cancer images. The model identifies and segments potential cancerous regions, highlighting areas of concern for further analysis. Post-segmentation, a classification module evaluates the segmented regions to distinguish between benign and malignant tissues, providing a probability score for each region. The system's performance is continuously improved through iterative training and validation on diverse datasets, ensuring robustness and generalizability. Additionally, the integration of explainable AI techniques allows for the visualization of model decisions, enhancing transparency and aiding radiologists in their diagnostic process. The proposed system aims to significantly reduce the time and effort required for breast cancer detection, while increasing diagnostic accuracy and early detection rates, ultimately improving patient outcomes.

**ADVANTAGES:**

**Increased Diagnostic Accuracy**: By integrating CNNs and sophisticated image processing algorithms, the system significantly improves the accuracy of detecting malignant regions in ultrasound images, ensuring more reliable diagnoses.

**Enhanced Early Detection Rates**: The advanced segmentation and classification capabilities enable earlier identification of potential cancerous regions, leading to timely interventions and better patient outcomes.

**Reduction in Diagnostic Time and Effort**: The automated workflow streamlines the detection process, reducing the time and effort required by radiologists to analyze ultrasound images, thus increasing efficiency.

**Minimized Human Error**: The system's consistent and objective analysis minimizes the likelihood of human error, ensuring more precise and consistent diagnostic results.

**Explainable AI for Improved Trust and Decision-Making**: The integration of explainable AI techniques allows for the visualization of model decisions, enhancing transparency and helping radiologists understand and validate the system's outputs, which facilitates better clinical decision-making.

**7. PREPARING THE DATASET:**

This dataset contains approximately 1000 train and 180 test image records of features extracted, which were then classified into 3 classes.

1. BENIGN
2. MALIGNANT
3. NORMAL

**8. LITERATURE SURVEY**

**General**

A literature review is a body of text that aims to review the critical points of current knowledge on and/or methodological approaches to a particular topic. It is secondary sources and discuss published information in a particular subject area and sometimes information in a particular subject area within a certain time period.

Its ultimate goal is to bring the reader up to date with current literature on a topic and forms the basis for another goal, such as future research that may be needed in the area and precedes a research proposal and may be just a simple summary of sources. Usually, it has an organizational pattern and combines both summary and synthesis.

A summary is a recap of important information about the source, but a synthesis is a re-organization, reshuffling of information. It might give a new interpretation of old material or combine new with old interpretations or it might trace the intellectual progression of the field, including major debates. Depending on the situation, the literature review may evaluate the sources and advise the reader on the most pertinent or relevant of them. Loan default trends have been long studied from a socio-economic stand point.

Most economics surveys believe in empirical modeling of these complex systems in order to be able to predict the loan default rate for a particular individual. The use of machine learning for such tasks is a trend which it is observing now. Some of the survey’s to understand the past and present perspective of loan approval or not.

**Review of Literature Survey**

**Title** : Patient Graph Deep Learning to Predict Breast Cancer Molecular Subtype

**Author**: Isaac Furtney, Ray Bradley, and Mansur R. Kabuka

**Year** : 2023

Breast cancer is a heterogeneous disease consisting of a diverse set of genomic mutations and clinical characteristics. The molecular subtypes of breast cancer are closely tied to prognosis and therapeutic treatment options. We investigate using deep graph learning on a collection of patient factors from multiple diagnostic disciplines to better represent breast cancer patient information and predict molecular subtype. Our method models breast cancer patient data into a multi-relational directed graph with extracted feature embeddings to directly represent patient information and diagnostic test results. We develop a radiographic image feature extraction pipeline to produce vector representation of breast cancer tumors in DCE-MRI and an autoencoder-based genomic variant embedding method to map variant assay results to a low-dimensional latent space. We leverage related-domain transfer learning to train and evaluate a Relational Graph Convolutional Network to predict the probabilities of molecular subtypes for individual breast cancer patient graphs. Our work found that utilizing information from multiple multimodal diagnostic disciplines improved the model’s prediction results and produced more distinct learned feature representations for breast cancer patients. This research demonstrates the capabilities of graph neural networks and deep learning feature representation to perform multimodal data fusion and representation in the breast cancer domain.

**Title** : Proposal of SVM Utility Kernel for Breast Cancer Survival Estimation

**Author** : Nikhilanand Arya , Archana Mathur , Snehanshu Saha, and Sriparna Saha

**Year** : 2023

The advancement of medical research in the field of cancer prognosis and diagnosis using various modalities has put oncologists under tremendous stress. The complexity and heterogeneity involved in multiple modalities and their significantly varied clinical outcomes make it difficult to analyze the disease and provide the correct treatment. Breast cancer is the major concern among all cancers worldwide, specifically for females. To help oncologists and cancer patients, research for breast cancer survival estimation has been proposed. It ranges from complex deep neural networks to simple and interpretable architectures. We propose a utility kernel for a support vector machine (SVM) in this article. It is a simple yet powerful function, which performs better than other popular machine learning algorithms and deep neural networks in the task of breast cancer survival prediction using the TCGA-BRCA dataset. This study validates the proposed utility kernel using four different modalities (gene expression, copy number variation, clinical, and histopathological tissue images) and their multi-modal combinations. The SVM based on our utility kernel empirically proves its efficacy by achieving the highest value on various performance measures, whereas advanced deep neural networks fail to train on small and highly imbalanced breast cancer data

**Title** : Recurrence Network Analysis of Histopathological Images for the Detection of Invasive Ductal Carcinoma in Breast Cancer

**Author**: Cheng-Bang Chen, Yujie Wang, Xuanya Fu, and Hui Yang

The histopathological image analysis is one of the most crucial diagnostic procedures to identify Invasive ductal carcinoma (IDC) in breast cancers. However, this diagnosis process is currently time-consuming and heavily dependent on human expertise. Prior research has shown that different degrees of tumors present various microstructures in the histopathological images. However, very little has been done to utilize spatial recurrence features of microstructures for identifying IDC. This paper presents a novel recurrence analysis methodology for automatic image-guided IDC detection. We first utilize wavelet decomposition to delineate the subtle information in the images. Then, we model the patches with a weighted recurrence network approach to characterize the recurrence patterns of the histopathological images. Finally, we develop automated IDC detection models leveraging machine learning methods with spatial recurrence features extracted. The developed recurrence analysis models successfully characterize the complex microstructures of histopathological images and achieve the IDC detection performances of at least AUC = 0.96. This research developed a spatial recurrence analysis methodology to effectively identify IDC regions in histopathological images for BC. It shows a high potential to assist physicians in the decision-making process. The proposed methodology can further be applicable to image processing for other medical or biological applications.

**Title** : Transfer Learning Based Lightweight Ensemble Model for Imbalanced Breast Cancer Classification

**Author**: Shankey Garg and Pradeep Singh

**Year** : 2020

Automated classification of breast cancer can often save lives, as manual detection is usually time-consuming & expensive. Since the last decade, deep learning techniques have been most widely used for the automatic classification of breast cancer using histopathology images. This paper has performed the binary and multi-class classification of breast cancer using a transfer learningbased ensemble model. To analyze the correctness and reliability of the proposed model, we have used an imbalance IDC dataset, an imbalance BreakHis dataset in the binary class scenario, and a balanced BACH dataset for the multi-class classification. A lightweight shallow CNN model with batch normalization technology to accelerate convergence is aggregated with lightweight MobileNetV2 to improve learning and adaptability. The aggregation output is fed into a multilayer perceptron to complete the final classification task. The experimental study on all three datasets was performed and compared with the recent works. We have fine-tuned three different pre-trained models (ResNet50, InceptionV4, and MobilNetV2) and compared it with the proposed lightweight ensemble model in terms of execution time, number of parameters, model size, etc. In both the evaluation phases, it is seen that our model outperforms in all three datasets

**Title** : VGGIN-Net: Deep Transfer Network for Imbalanced Breast Cancer Dataset

**Author**: Manisha Saini and Seba Susan

**Year** : 2023

In this paper, we have presented a novel deep neural network architecture involving transfer learning approach, formed by freezing and concatenating all the layers till block4 pool layer of VGG16 pre-trained model (at the lower level) with the layers of a randomly initialized naı¨ve Inception block module (at the higher level). Further, we have added the batch normalization, flatten, dropout and dense layers in the proposed architecture. Our transfer network, called VGGIN-Net, facilitates the transfer of domain knowledge from the larger ImageNet object dataset to the smaller imbalanced breast cancer dataset. To improve the performance of the proposed model, regularization was used in the form of dropout and data augmentation. A detailed block-wise fine tuning has been conducted on the proposed deep transfer network for images of different magnification factors. The results of extensive experiments indicate a significant improvement of classification performance after the application of fine-tuning. The proposed deep learning architecture with transfer learning and fine-tuning yields the highest accuracies in comparison to other state-of-the-art approaches for the classification of BreakHis breast cancer dataset. The articulated architecture is designed in a way that it can be effectively transfer learned on other breast cancer datasets

**9. SYSTEM STUDY**

**9.1 project goal:**

The aim of the breast cancer classification and segmentation using artificial intelligence techniques is to develop and implement advanced machine learning and computer vision models that can accurately and efficiently assist in the diagnosis and characterization of breast cancer from medical imaging data. This includes the identification of malignant and benign tumors, tumor segmentation for precise localization, and the provision of valuable insights to medical professionals for early detection and treatment planning.

#### **9.2 Objectives:**

1. Improved Diagnosis: Develop AI models capable of accurately classifying breast tumors as benign or malignant, reducing the chances of misdiagnosis and ensuring timely medical intervention.

2. Tumor Localization: Implement segmentation techniques to precisely delineate the boundaries of tumors within breast images, aiding in surgical planning and treatment strategies.

3. Early Detection: Enable early detection of breast cancer by automating the analysis of screening ultrasound images, potentially increasing survival rates through timely interventions.

4. Reduced Workload: Assist healthcare professionals by automating time-consuming tasks, allowing them to focus more on patient care and less on image analysis.

5. Enhanced Accuracy: Improve the overall accuracy of breast cancer diagnosis by leveraging AI's ability to analyze vast datasets and patterns that may be challenging for human experts to discern.

6. Scalability: Develop scalable solutions that can be applied to a wide range of medical imaging data and integrated into healthcare systems for broader clinical use.

**9.3 Scope:**

The scope of breast cancer classification and segmentation using artificial intelligence encompasses various aspects of medical image analysis and diagnosis. This includes, but is not limited to, ultrasound images, ultrasound, MRI, and histopathological image analysis. The application of AI in breast cancer diagnosis has the potential to revolutionize the healthcare industry by offering more accurate, efficient, and accessible diagnostic tools. Additionally, it can support medical professionals in making informed decisions, particularly in cases of breast cancer, which is one of the most common and critical forms of cancer affecting women worldwide. The scope extends to both research and practical implementation, with a focus on improving patient outcomes and reducing the global burden of breast cancer.

**10. OUTLINE OF THE PROJECT**

**10.1 Overview of the system:**

* Define a problem
* Gathering image data set
* Evaluating algorithms
* Detecting results

The steps involved in Building the data model is depicted below.

**Data collection** (Splitting Training set & Test) set)

**Pre Processing** (Sequential)

**Building classification Model**

**Prediction (**Breast cancer**)**

Fig: data flow diagram for CNN model

**11. PROJECT REQUIREMENTS**

**General:**

Requirements are the basic constrains that are required to develop a system. Requirements are collected while designing the system. The following are the requirements that are to be discussed.

1. Functional requirements

2. Non-Functional requirements

3. Environment requirements

A. Hardware requirements

B. software requirements

**11. 1 Functional requirements:**

The software requirements specification is a technical specification of requirements for the software product. It is the first step in the requirements analysis process. It lists requirements of a particular software system. The following details to follow the special libraries like tensorflow, keras, matplotlib.

**11.2 Non-Functional Requirements:**

Process of functional steps,

1. Problem define
2. Preparing data
3. Evaluating algorithm
4. Improving results
5. Prediction the result

**Environment Requirements:**

**Framework :** Keras.

**Software Requirements:**

* Operating System : Windows / Linux
* Simulation Tool : Anaconda with Jupyter Notebook
* Language : Python

**Hardware requirements:**

* Processor : Intel i3
* Hard disk : minimum 400 GB
* RAM : minimum 4 GB

12. **FEASIBILITY STUDY**

**Splitting the dataset:**

The data use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. It has the test dataset (or subset) in order to test our models and it will do this using Tensor flow library in Python using the Keras method.

**Construction of a Detecting Model:**

## Deep learning needs data gathering have lot of past image data’s. Training and testing this model working and predicting correctly.

Data Gathering

CNN Algorithm

Train model

Test model

Prediction

Steps of dataflow diagram

**12.1 Data Flow Diagram:**

Normal and mask images

Test dataset

Preprocessing

Breast cancer segmentation

U-NET Architecture

Training dataset

Fig: Process of dataflow diagram

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model. Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top down approach to Systems Design. Symbols and Notations Used in DFDs Using any convention’s DFD rules or guidelines, the symbols depict the four components of data flow diagrams.

External entity: an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.

Process: any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules.

Data store: files or repositories that hold information for later use, such as a database table or a membership form.

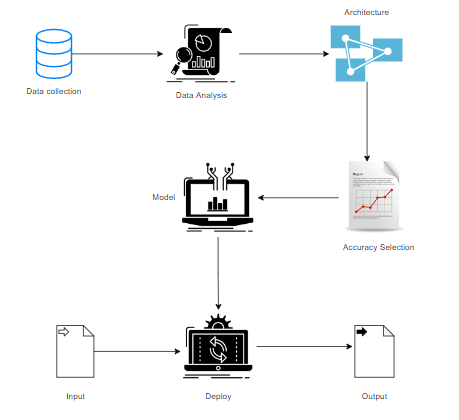
Data flow: the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like “Billing details.”

DFD levels and layers A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what you are trying to accomplish. DFD Level 0 is also called a Context Diagram. It’s a basic overview of the whole system or process being analyzed or modeled. It’s designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts & developers

**13. DESIGN ARCHITECTURE**

**General**

Design is meaningful engineering representation of something that is to be built. Software design is a process design is the perfect way to accurately translate requirements in to a finished software product. Design creates a representation or model, provides detail about software data structure, architecture, interfaces and components that are necessary to implement a system.



**14. Work flow diagram:**

Data Collection

Pre-processing

Training Dataset

Testing Dataset

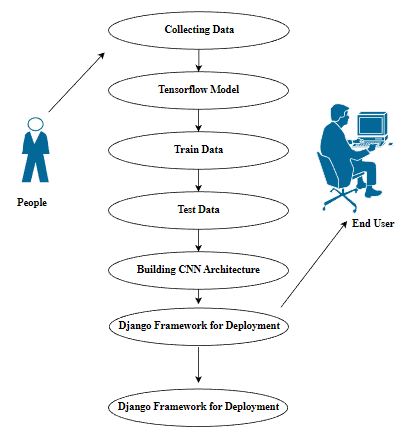
U-NET Architecture

Model

Segmentation

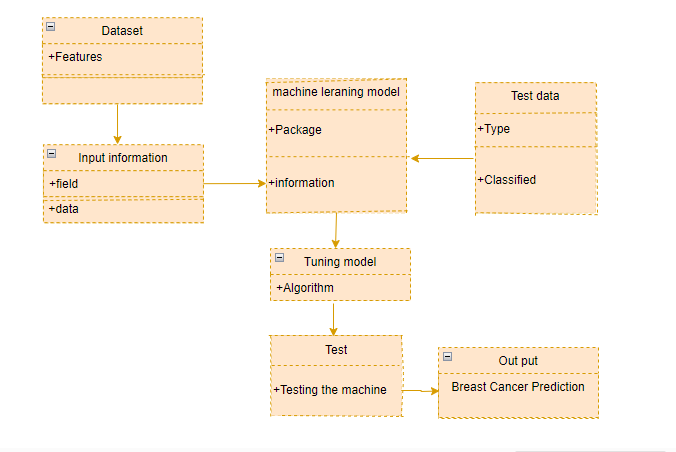
Workflow Diagram

**15. USECASE DIAGRAM:**



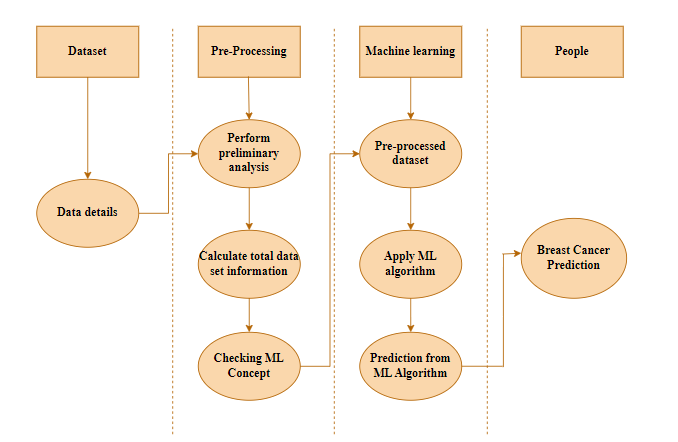
Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

**16. CLASS DIAGRAM:**



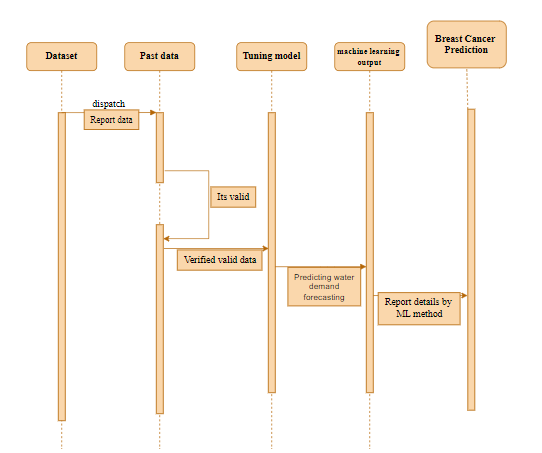
Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system. Each element and their relationships should be identified in advance Responsibility (attributes and methods) of each class should be clearly identified for each class minimum number of properties should be specified and because, unnecessary properties will make the diagram complicated. Use notes whenever required to describe some aspect of the diagram and at the end of the drawing it should be understandable to the developer/coder. Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

17. **ACTIVITY DIAGRAM:**



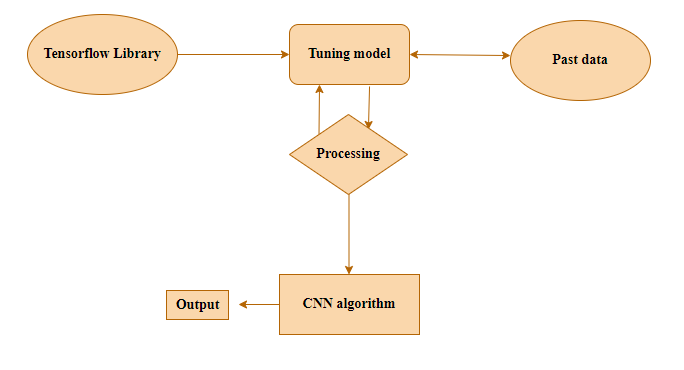
Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

**18. SEQUENCE DIAGRAM:**



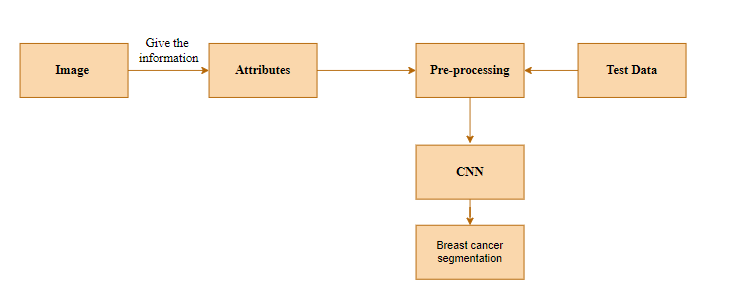
Sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modelling, which focuses on identifying the behaviour within your system. Other dynamic modelling techniques include [activity diagramming](http://agilemodeling.com/artifacts/activityDiagram.htm), [communication diagramming](http://agilemodeling.com/artifacts/communicationDiagram.htm), [timing diagramming](http://agilemodeling.com/artifacts/timingDiagram.htm), and [interaction overview diagramming](http://agilemodeling.com/artifacts/interactionOverviewDiagram.htm). Sequence diagrams, along with [class diagrams](http://agilemodeling.com/artifacts/classDiagram.htm) and [physical data models](http://agiledata.org/essays/dataModeling101.html) are in my opinion the most important design-level models for modern business application development.

**19. ER DIAGRAM:**

****

An entity relationship diagram (ERD), also known as an entity relationship model, is a graphical representation of an information system that depicts the relationships among people, objects, places, concepts or events within that system. An ERD is a data modeling technique that can help define business processes and be used as the foundation for a relational database. Entity relationship diagrams provide a visual starting point for database design that can also be used to help determine information system requirements throughout an organization. After a relational database is rolled out, an ERD can still serve as a referral point, should any debugging or business process re-engineering be needed later.

**20. COLLABORATION DIAGRAM:**



A collaboration diagram show the objects and relationships involved in an interaction, and the sequence of messages exchanged among the objects during the interaction.

The collaboration diagram can be a decomposition of a class, class diagram, or part of a class diagram.it can be the decomposition of a use case, use case diagram, or part of a use case diagram.

The collaboration diagram shows messages being sent between classes and object (instances). A diagram is created for each system operation that relates to the current development cycle (iteration).

**21. SOFTWARE DESCRIPTION**

Anaconda is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source) distribution of the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) and [R](https://en.wikipedia.org/wiki/R_(programming_language)) programming languages for [scientific computing](https://en.wikipedia.org/wiki/Scientific_computing) ([data science](https://en.wikipedia.org/wiki/Data_science), [machine learning](https://en.wikipedia.org/wiki/Machine_learning) applications, large-scale data processing, [predictive analytics](https://en.wikipedia.org/wiki/Predictive_analytics), etc.), that aims to simplify [package management](https://en.wikipedia.org/wiki/Package_management) and deployment. Package versions are managed by the [package management system](https://en.wikipedia.org/wiki/Package_manager) “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. So, Anaconda distribution comes with more than 1,400 packages as well as the [Conda](https://en.wikipedia.org/wiki/Conda_(package_manager)) package and virtual environment manager called Anaconda Navigator and 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](https://en.wikipedia.org/wiki/Pip_(package_manager)) 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](https://en.wikipedia.org/wiki/Python_Package_Index) 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.

**21.1 ANACONDA NAVIGATOR:**

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository.

Anaconda. Now, if you are primarily doing data science work, Anaconda is also a great option. Anaconda is created by Continuum Analytics, and it is a Python distribution that comes preinstalled with lots of useful python libraries for data science.

Anaconda is a 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.

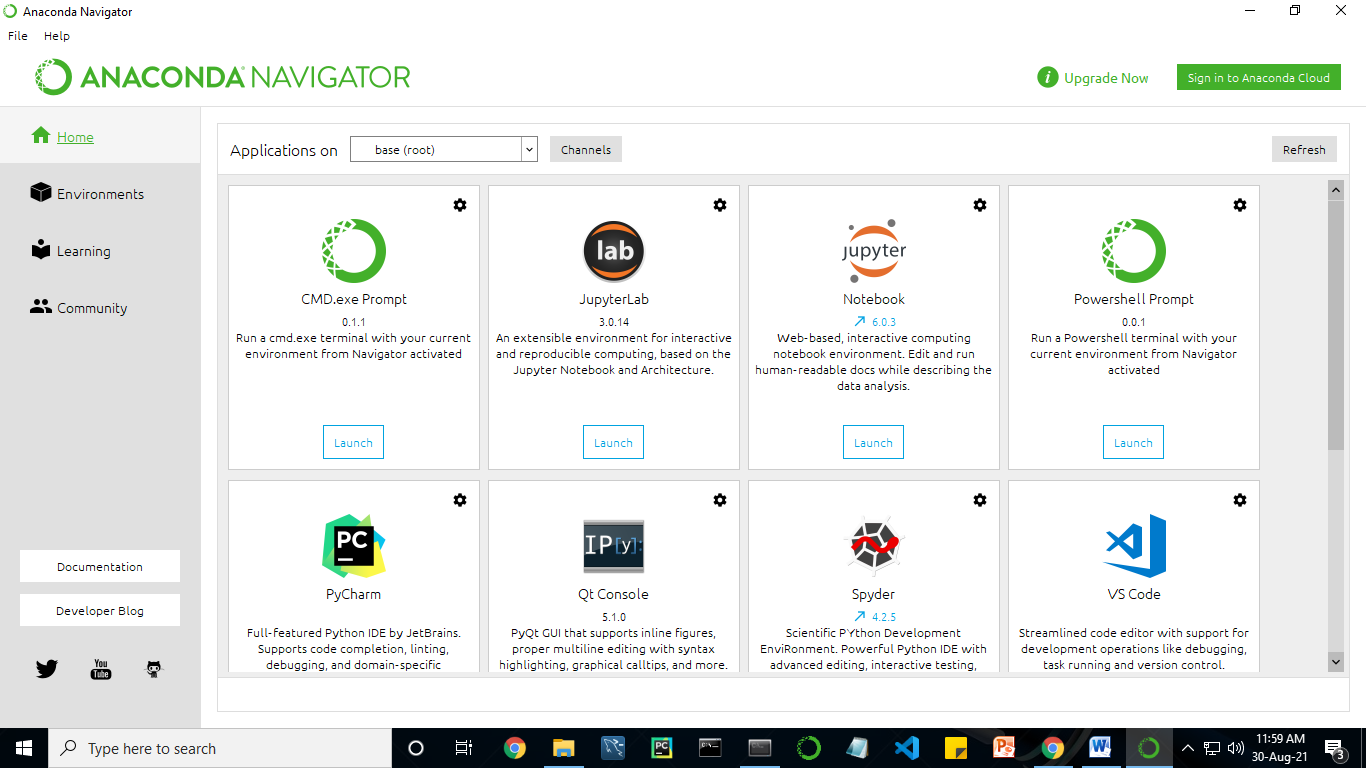
In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

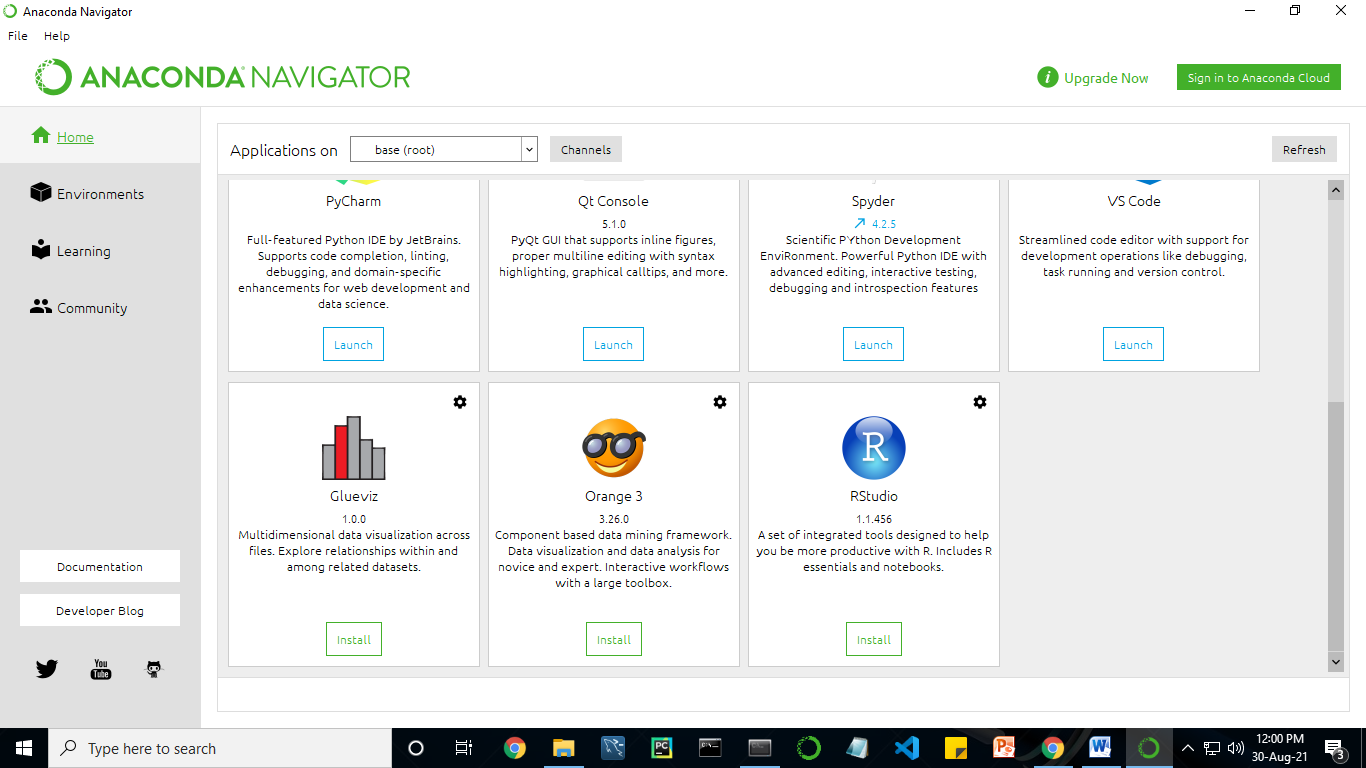
The command-line program conda is both a package manager and an environment manager. This helps data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages, and update them – all inside Navigator.

The following applications are available by default in Navigator:

* [JupyterLab](https://jupyterlab.readthedocs.io/en/stable/)
* [Jupyter Notebook](https://jupyter.readthedocs.io/en/latest/)
* [Spyder](https://www.spyder-ide.org/)
* [VSCode](https://code.visualstudio.com/docs)
* [Glueviz](http://glueviz.org/en/stable/)
* [Orange 3 App](http://orange.biolab.si/docs/)
* [RStudio](http://docs.rstudio.com/)
* Anaconda Prompt (Windows only)
* Anaconda PowerShell (Windows only)





Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution.

Navigator allows you to launch common Python programs and easily 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.

Anaconda comes with many built-in packages that you can easily find with conda list on your anaconda prompt. As it has lots of packages (many of which are rarely used), it requires lots of space and time as well. If you have enough space, time and do not want to burden yourself to install small utilities like JSON, YAML, you better go for Anaconda.

**21.2 JUPYTER NOTEBOOK:**

This website acts as “meta” documentation for the Jupyter ecosystem. It has a collection of resources to navigate the tools and communities in this ecosystem, and to help you get started.

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Perez.

Notebook documents are documents produced by the [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app), which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc…). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents which can be run to perform data analysis.

## Installation: The easiest way to install the Jupyter Notebook App is installing a scientific python distribution which also includes scientific python packages. The most common distribution is called **Anaconda**

# Running the Jupyter Notebook

## Launching Jupyter Notebook App: The [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app) can be launched by clicking on the Jupyter Notebook icon installed by Anaconda in the start menu (Windows) or by typing in a terminal (cmd on Windows): “jupyter notebook”

## This will launch a new browser window (or a new tab) showing the [Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#dashboard), a sort of control panel that allows (among other things) to select which notebook to open.

## When started, the [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app) can access only files within its start-up folder (including any sub-folder). No configuration is necessary if you place your notebooks in your home folder or subfolders. Otherwise, you need to choose a [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app) start-up folder which will contain all the notebooks.

## Save notebooks: Modifications to the notebooks are automatically saved every few minutes. To avoid modifying the original notebook, make a copy of the notebook document (menu file -> make a copy…) and save the modifications on the copy.

## Executing a notebook: Download the notebook you want to execute and put it in your notebook folder (or a sub-folder of it).

* Launch the jupyter notebook app
* In the [Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#dashboard) navigate to find the notebook: clicking on its name will open it in a new browser tab.
* Click on the menu Help -> User Interface Tour for an overview of the [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app) user interface.
* You can run the notebook document step-by-step (one cell a time) by pressing shift + enter.
* You can run the whole notebook in a single step by clicking on the menu Cell -> Run All.
* To restart the [kernel](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#kernel) (i.e. the computational engine), click on the menu Kernel -> Restart. This can be useful to start over a computation from scratch (e.g. variables are deleted, open files are closed, etc…).

[**Purpose**](https://www.google.com/search?q=project+jupyter+purpose&sa=X&ved=2ahUKEwin49vtmdjyAhXx4zgGHXSOCuwQ6BMoADAkegQINxAC&cshid=1630307847256010)**:** To support [interactive](https://www.google.com/search?q=interactive&stick=H4sIAAAAAAAAAONgVuLUz9U3MM0uyYpfxMqdmVeSWpSYXJJZlgoApkTFPhsAAAA&sa=X&ved=2ahUKEwin49vtmdjyAhXx4zgGHXSOCuwQmxMoATAkegQINxAD&cshid=1630307847256010) data science and scientific computing across all programming languages.

**File Extension:** An IPYNB file is a notebook document created by Jupyter Notebook, an interactive computational environment that helps scientists manipulate and analyze data using Python.

**JUPYTER Notebook App:** The Jupyter Notebook Appis a server-client application that allows editing and running [notebook documents](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document) via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet.

In addition to displaying/editing/running notebook documents, the Jupyter Notebook App has a “Dashboard” ([Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#dashboard)), a “control panel” showing local files and allowing to open notebook documents or shutting down their [kernels](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#kernel).

## [**kernel**](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#id7)**:** A notebook kernel is a “computational engine” that executes the code contained in a [Notebook document](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document). The ipython kernel*,* referenced in this guide, executes python code. Kernels for many other languages exist ([official kernels](http://jupyter.readthedocs.org/en/latest/#kernels)).

When you open a [Notebook document](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document), the associated kernel is automatically launched. When the notebook is executed (either cell-by-cell or with menu Cell -> Run All), the kernel performs the computation and produces the results. Depending on the type of computations, the kernel may consume significant CPU and RAM. Note that the RAM is not released until the kernel is shut-down

## [Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#id8): The Notebook Dashboard is the component which is shown first when you launch [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app). The Notebook Dashboard is mainly used to open [notebook documents](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document), and to manage the running [kernels](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#kernel) (visualize and shutdown).

The Notebook Dashboard has other features similar to a file manager, namely navigating folders and renaming/deleting files

**Working Process:**

* Download and install anaconda and get the most useful package for machine learning in Python.
* Load a dataset and understand its structure using statistical summaries and data visualization.
* Machine learning models, pick the best and build confidence that the accuracy is reliable.

Python is a popular and powerful interpreted language. Unlike R, Python is a complete language and platform that you can use for both research and development and developing production systems. There are also a lot of modules and libraries to choose from, providing multiple ways to do each task. It can feel overwhelming.

The best way to get started using Python for machine learning is to complete a project.

* It will force you to install and start the Python interpreter (at the very least).
* It will give you a bird’s eye view of how to step through a small project.
* It will give you confidence, maybe to go on to your own small projects.

When you are applying machine learning to your own datasets, you are working on a project. A machine learning project may not be linear, but it has a number of well-known steps:

* Define Problem.
* Prepare Data.
* Evaluate Algorithms.
* Improve Results.
* Present Results.

The best way to really come to terms with a new platform or tool is to work through a machine learning project end-to-end and cover the key steps. Namely, from loading data, summarizing data, evaluating algorithms and making some predictions.

Here is an overview of what we are going to cover:

1. Installing the Python anaconda platform.
2. Loading the dataset.
3. Summarizing the dataset.
4. Visualizing the dataset.
5. Evaluating some algorithms.
6. Making some predictions.

**21.3 Visual Studio Code:**

Visual Studio Code (VS Code) is a versatile and lightweight source code editor that provides a wide range of features for developers. It supports various programming languages and extensions, making it a popular choice for Python development and beyond.

**VS Code Features:**

Intelligent Coding Assistance: VS Code offers intelligent code completion, error checking, and quick fixes to enhance your coding experience. It supports Python and a variety of other languages, providing a smart and configurable editor.

Code Editing: The smart code editor in VS Code includes support for Python, JavaScript, TypeScript, HTML, CSS, and more. It features language-aware code completion, error detection, and on-the-fly code fixes to boost your productivity.

Smart Code Navigation: Use the built-in search functionality to quickly navigate to classes, files, symbols, and IDE actions. Switching between declarations, references, and implementations is just a click away.

Refactoring Tools: VS Code supports fast and safe refactorings, including renaming, extracting methods, introducing variables, and more. Language-specific refactorings help you make project-wide code changes efficiently.

Built-in Developer Tools: VS Code comes with integrated tools like a debugger, test runner, profiler, terminal, and version control system (VCS) support for Git and others. It also integrates seamlessly with remote development tools, Docker, and more.

Debugging, Testing, and Profiling: Benefit from the graphical debugger with Python support, run tests with coding assistance, and use the integrated test runner. Profiler integration helps you analyze and optimize your code.

VCS, Deployment, and Remote Development : Manage version control with a unified UI supporting Git, SVN, and others. Run and debug applications on remote machines, and configure automatic deployment to remote hosts or virtual machines.

Database Tools: VS Code allows you to access and edit databases such as MySQL, PostgreSQL, SQL Server, and others directly from the IDE. It provides support for editing SQL code, running queries, and managing database schemas.

Web Development: Besides Python, VS Code provides excellent support for various web development frameworks, JavaScript, TypeScript, HTML, CSS, AngularJS, and Node.js.

Python Web Frameworks: VS Code offers support for popular web development frameworks like Django and Flask, providing features such as code completion, navigation, and debugging.

JavaScript & HTML: VS Code provides first-class support for JavaScript, TypeScript, HTML, and CSS. It includes a JavaScript debugger integrated with server run configurations.

Live Editing: Live Editing Preview allows you to open a page in the editor and see code changes instantly in the browser. Auto-saving ensures your changes are reflected in real-time.

Scientific Tools: VS Code integrates with Jupyter Notebooks, provides an interactive Python console, and supports scientific packages such as Matplotlib and NumPy.

Interactive Python Console: Run a REPL Python console in VS Code with features like on-the-fly syntax checking, braces and quotes matching, and code completion.

Scientific Stack Support:VS Code has built-in support for scientific libraries like Pandas, NumPy, and Matplotlib, offering advanced code intelligence and visualization tools.

Conda Integration: Manage dependencies with separate Conda environments per project, ensuring isolation and easy environment selection.

Customizable and Cross-platform: VS Code is a customizable and cross-platform IDE, supporting Windows, Mac OS, and Linux. Enjoy a tailored workspace with customizable color schemes and keybindings.

Plugins: VS Code has a rich extension ecosystem with a variety of plugins, offering additional VCS support, integrations with tools and frameworks, and editor enhancements.

Cross-platform IDE: Run VS Code on Windows, Mac OS, or Linux, maintaining the same environment and functionality across multiple machines.

**22. PYTHON:**

**Introduction:**

**Python** is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language) [high-level](https://en.wikipedia.org/wiki/High-level_programming_language) [general-purpose programming language](https://en.wikipedia.org/wiki/General-purpose_programming_language). Its design philosophy emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability) with its use of [significant indentation](https://en.wikipedia.org/wiki/Off-side_rule). Its [language constructs](https://en.wikipedia.org/wiki/Language_construct) as well as its [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) approach aim to help [programmers](https://en.wikipedia.org/wiki/Programmers) write clear, logical code for small and large-scale projects.

Python is [dynamically-typed](https://en.wikipedia.org/wiki/Type_system#DYNAMIC) and [garbage-collected](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigms), including [structured](https://en.wikipedia.org/wiki/Structured_programming) (particularly, [procedural](https://en.wikipedia.org/wiki/Procedural_programming)), object-oriented and [functional programming](https://en.wikipedia.org/wiki/Functional_programming). It is often described as a "batteries included" language due to its comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

[Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) began working on Python in the late 1980s, as a successor to the [ABC programming language](https://en.wikipedia.org/wiki/ABC_(programming_language)), and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as [list comprehensions](https://en.wikipedia.org/wiki/List_comprehension) and a garbage collection system using [reference counting](https://en.wikipedia.org/wiki/Reference_counting). Python 3.0 was released in 2008 and was a major revision of the language that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility). Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages

**History:**

Python was conceived in the late 1980s by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) at [Centrum Wiskunde & Informatica](https://en.wikipedia.org/wiki/Centrum_Wiskunde_%26_Informatica) (CWI) in the [Netherlands](https://en.wikipedia.org/wiki/Netherlands) as a successor to [ABC programming language](https://en.wikipedia.org/wiki/ABC_(programming_language)), which was inspired by [SETL](https://en.wikipedia.org/wiki/SETL),  capable of [exception handling](https://en.wikipedia.org/wiki/Exception_handling) and interfacing with the [Amoeba](https://en.wikipedia.org/wiki/Amoeba_(operating_system)) operating system. Its implementation began in December 1989.  Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's [Benevolent Dictator For Life](https://en.wikipedia.org/wiki/Benevolent_Dictator_For_Life), a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a 5-member "Steering Council" to lead the project.  As of 2021, the current members of this council are Barry Warsaw, Brett Cannon, Carol Willing, Thomas Wouters, and Pablo Galindo Salgado.

Python 2.0 was released on 16 October 2000, with many major new features, including a [cycle-detecting](https://en.wikipedia.org/wiki/Cycle_detection) [garbage collector](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) and support for [Unicode](https://en.wikipedia.org/wiki/Unicode).

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility). Many of its major features were [backported](https://en.wikipedia.org/wiki/Backporting) to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2 to 3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's [end-of-life](https://en.wikipedia.org/wiki/End-of-life_(product)) 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. No more security patches or other improvements will be released for it. With Python 2's [end-of-life](https://en.wikipedia.org/wiki/End-of-life_(product)), only Python 3.6.x  and later are supported.

Python 3.9.2 and 3.8.8 were expeditedas all versions of Python (including 2.7) had security issues, leading to possible [remote code execution](https://en.wikipedia.org/wiki/Remote_code_execution) and [web cache poisoning](https://en.wikipedia.org/wiki/Cache_poisoning).

**Design Philosophy & Feature**

Python is a [multi-paradigm programming language](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language). [Object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) and [structured programming](https://en.wikipedia.org/wiki/Structured_programming) are fully supported, and many of its features support functional programming and [aspect-oriented programming](https://en.wikipedia.org/wiki/Aspect-oriented_programming) (including by [meta-programming](https://en.wikipedia.org/wiki/Metaprogramming) and [meta-objects](https://en.wikipedia.org/wiki/Metaobject) (magic methods)). Many other paradigms are supported via extensions, including [design by contract](https://en.wikipedia.org/wiki/Design_by_contract) and [logic programming](https://en.wikipedia.org/wiki/Logic_programming).

Python uses [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_typing) and a combination of [reference counting](https://en.wikipedia.org/wiki/Reference_counting) and a cycle-detecting garbage collector for [memory management](https://en.wikipedia.org/wiki/Memory_management). It also features dynamic [name resolution](https://en.wikipedia.org/wiki/Name_resolution_(programming_languages)) ([late binding](https://en.wikipedia.org/wiki/Late_binding)), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the [Lisp](https://en.wikipedia.org/wiki/Lisp_(programming_language)) tradition. It has filter, map and reduce functions;  [list comprehensions](https://en.wikipedia.org/wiki/List_comprehension), [dictionaries](https://en.wikipedia.org/wiki/Associative_array), sets, and [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)) expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)) and [Standard ML](https://en.wikipedia.org/wiki/Standard_ML).

The language's core philosophy is summarized in the document The [Zen of Python](https://en.wikipedia.org/wiki/Zen_of_Python) (PEP 20), which includes [aphorisms](https://en.wikipedia.org/wiki/Aphorism) such as:

* 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](https://en.wikipedia.org/wiki/Extensibility) (with modules). 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](https://en.wikipedia.org/wiki/ABC_(programming_language)), which espoused the opposite approach.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to [Perl](https://en.wikipedia.org/wiki/Perl)'s "[there is more than one way to do it](https://en.wikipedia.org/wiki/There_is_more_than_one_way_to_do_it)" motto, Python embraces a "there should be one— and preferably only one —obvious way to do it" design philosophy. [Alex Martelli](https://en.wikipedia.org/wiki/Alex_Martelli), a [Fellow](https://en.wikipedia.org/wiki/Fellow) at the [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation) and Python book author, writes that "To describe something as 'clever' is not considered a compliment in the Python culture."

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

Python's developers aim to keep the language fun to use. This is reflected in its name a tribute to the British comedy group [Monty Python](https://en.wikipedia.org/wiki/Monty_Python) and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (a reference to a [Monty Python sketch](https://en.wikipedia.org/wiki/Spam_(Monty_Python))) instead of the standard [foo and bar](https://en.wikipedia.org/wiki/Foobar).

A common [neologism](https://en.wikipedia.org/wiki/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 Pythonistas

**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](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) to delimit blocks, and semicolons after statements are allowed but are rarely, if ever, used. It has fewer syntactic exceptions and special cases than [C](https://en.wikipedia.org/wiki/C_(programming_language)) or [Pascal](https://en.wikipedia.org/wiki/Pascal_(programming_language)).

**Indentation :**

Main article: [Python syntax and semantics & Indentation](https://en.wikipedia.org/wiki/Python_syntax_and_semantics#Indentation)

Python uses [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation, rather than [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) or keywords, to delimit [blocks](https://en.wikipedia.org/wiki/Block_(programming)). 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 sometimes termed the [off-side rule](https://en.wikipedia.org/wiki/Off-side_rule), which some other languages share, but in most languages indentation does not have any semantic meaning. The recommended indent size is four spaces.

**Statements and control flow :**

Python's [statements](https://en.wikipedia.org/wiki/Statement_(computer_science)) include:

* The [assignment](https://en.wikipedia.org/wiki/Assignment_(computer_science)) statement, using a single equals sign =.
* The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).
* The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
* The while statement, which executes a block of code as long as its condition is true.
* The Try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.
* The raise statement, used to raise a specified exception or re-raise a caught exception.
* The class statement, which executes a block of code and attaches its local namespace to a [class](https://en.wikipedia.org/wiki/Class_(computer_science)), for use in object-oriented programming.
* The def statement, which defines a [function](https://en.wikipedia.org/wiki/Function_(computing)) or [method](https://en.wikipedia.org/wiki/Method_(computing)).
* The with statement, which encloses a code block within a context manager (for example, acquiring a [lock](https://en.wikipedia.org/wiki/Lock_(computer_science)) before the block of code is run and releasing the lock afterwards, or opening a [file](https://en.wikipedia.org/wiki/Computer_file) and then closing it), allowing [resource-acquisition-is-initialization](https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization) (RAII) - like behavior and replaces a common try/finally idiom.
* The break statement, exits from a loop.
* The continue statement, skips this iteration and continues with the next item.
* The del statement, removes a variable, which means the reference from the name to the value is deleted and trying to use that variable will cause an error. A deleted variable can be reassigned.
* The pass statement, which serves as a [NOP](https://en.wikipedia.org/wiki/NOP_(code)). It is syntactically needed to create an empty code block.
* The assert statement, used during debugging to check for conditions that should apply.
* The yield statement, which returns a value from a [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)#Python) function and yield is also an operator. This form is used to implement [co-routines](https://en.wikipedia.org/wiki/Coroutine).
* The return statement, used to return a value from a function.
* The import statement, which is used to import modules whose functions or variables can be used in the current program.

The assignment statement (=) operates by binding a name as a [reference](https://en.wikipedia.org/wiki/Pointer_(computer_programming)) to a separate, dynamically-allocated [object](https://en.wikipedia.org/wiki/Object_(computer_science)). Variables may be subsequently rebound at any time to any object. In Python, a variable name is a generic reference holder and does not have a fixed [data type](https://en.wikipedia.org/wiki/Type_system) associated with it. However, at a given time, a variable will refer to some object, which will have a type. This is referred to as [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_type) and is contrasted with [statically-typed](https://en.wikipedia.org/wiki/Statically-typed) programming languages, where each variable may only contain values of a certain type.

Python does not support [tail call](https://en.wikipedia.org/wiki/Tail_call) optimization or [first-class continuations](https://en.wikipedia.org/wiki/First-class_continuations), and, according to Guido van Rossum, it never will.[[80]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-55-80)[[81]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-56-81) However, better support for [co-routine](https://en.wikipedia.org/wiki/Coroutine)-like functionality is provided, by extending Python's [generators](https://en.wikipedia.org/wiki/Generator_(computer_programming)). Before 2.5, generators were [lazy](https://en.wikipedia.org/wiki/Lazy_evaluation) [iterators](https://en.wikipedia.org/wiki/Iterator); information was passed uni-directionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.

**Expressions** :

Some Python [expressions](https://en.wikipedia.org/wiki/Expression_(computer_science)) are similar to those found in languages such as C and [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), while some are not:

* Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division (or integer division) // and floating-point/division. Python also uses the \*\* operator for exponentiation.
* From Python 3.5, the new @ infix operator was introduced. It is intended to be used by libraries such as [NumPy](https://en.wikipedia.org/wiki/NumPy) for [matrix multiplication](https://en.wikipedia.org/wiki/Matrix_multiplication).
* From Python 3.8, the syntax :=, called the 'walrus operator' was introduced. It assigns values to variables as part of a larger expression.
* In Python, == compares by value, versus Java, which compares numerics by value and objects by reference. (Value comparisons in Java on objects can be performed with the equals() method.) Python's is operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example A<=B<=C.
* Python uses the words and, or, not for or its boolean operators rather than the symbolic &&, ||, ! used in Java and C.
* Python has a type of expression termed a [list comprehension](https://en.wikipedia.org/wiki/List_comprehension#Python) as well as a more general expression termed a [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)) expression.
* [Anonymous functions](https://en.wikipedia.org/wiki/Anonymous_function) are implemented using [lambda expressions](https://en.wikipedia.org/wiki/Lambda_(programming)); however, these are limited in that the body can only be one expression.
* Conditional expressions in Python are written as x if c else y (different in order of operands from the c ? x : y operator common to many other languages).
* Python makes a distinction between [lists](https://en.wikipedia.org/wiki/List_(computer_science)) and [tuples](https://en.wikipedia.org/wiki/Tuple). Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be [immutable](https://en.wikipedia.org/wiki/Immutable) in Python). Tuples are written as (1, 2, 3), are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable t initially equal to (1, 2, 3), executing t = t + (4, 5) first evaluates t + (4, 5), which yields (1, 2, 3, 4, 5), which is then assigned back to t, thereby effectively "modifying the contents" of t, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.
* Python features sequence unpacking wherein multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in an identical manner to that forming tuple literals and, as a whole, are put on the left-hand side of the equal sign in an assignment statement. The statement expects an iterable object on the right-hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through and will iterate through it, assigning each of the produced values to the corresponding expression on the left.
* Python has a "string format" operator %. This functions analogously ton printf format strings in C, e.g. “spam=%s eggs=%d” % (“blah”,2) evaluates to “spam=blah eggs=2”. In Python 3 and 2.6+, this was supplemented by the format() method of the str class, e.g. “spam={0} eggs={1}”.format(“blah”,2). Python 3.6 added "f-strings": blah = “blah”; eggs = 2; f‘spam={blah} eggs={eggs}’
* Strings in Python can be [concatenated](https://en.wikipedia.org/wiki/Concatenation), by "adding" them (same operator as for adding integers and floats). E.g. “spam” + “eggs” returns “spameggs”. Even if your strings contain numbers, they are still added as strings rather than integers. E.g. “2” + “2” returns “2”.

Python has various kinds of [string literals](https://en.wikipedia.org/wiki/String_literal):

* + Strings delimited by single or double quote marks. Unlike in [Unix shells](https://en.wikipedia.org/wiki/Unix_shell), [Perl](https://en.wikipedia.org/wiki/Perl) and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an [escape character](https://en.wikipedia.org/wiki/Escape_character). [String interpolation](https://en.wikipedia.org/wiki/String_interpolation) became available in Python 3.6 as "formatted string literals".
  + Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like [here documents](https://en.wikipedia.org/wiki/Here_document) in shells, Perl and [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)).
  + [Raw string](https://en.wikipedia.org/wiki/Raw_string) varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as [regular expressions](https://en.wikipedia.org/wiki/Regular_expression) and [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows)-style paths. Compare "@-quoting" in [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)).
* Python has [array index](https://en.wikipedia.org/wiki/Array_index) and [array slicing](https://en.wikipedia.org/wiki/Array_slicing) expressions on lists, denoted as a[Key], a[start:stop] or a[start:stop:step]. Indexes are [zero-based](https://en.wikipedia.org/wiki/Zero-based_numbering), and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example a[:] returns a copy of the entire list. Each element of a slice is a [shallow copy](https://en.wikipedia.org/wiki/Shallow_copy).

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp), [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language)), or [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)). This leads to duplicating some functionality. For example:

* [List comprehensions](https://en.wikipedia.org/wiki/List_comprehensions) vs. for-loops
* [Conditional](https://en.wikipedia.org/wiki/Conditional_(programming)) expressions vs. if blocks
* The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements.

Statements cannot be a part of an expression, so list and other comprehensions or [lambda expressions](https://en.wikipedia.org/wiki/Lambda_(programming)), all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as a=1 cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c==1) {…} is syntactically valid (but probably unintended) C code but if c=1: … causes a syntax error in Python.

**Methods** :

[Methods](https://en.wikipedia.org/wiki/Method_(programming)) on objects are [functions](https://en.wikipedia.org/wiki/Function_(programming)) attached to the object's class; the syntax instance.method(argument) is, for normal methods and functions, [syntactic sugar](https://en.wikipedia.org/wiki/Syntactic_sugar) for Class.method(instance, argument). Python methods have an explicit self parameter access [instance data](https://en.wikipedia.org/wiki/Instance_data), in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., [C++](https://en.wikipedia.org/wiki/C%2B%2B), Java, [Objective-C](https://en.wikipedia.org/wiki/Objective-C), or [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language))). Apart from this Python also provides methods, sometimes called d-under methods due to their names beginning and ending with double-underscores, to extend the functionality of custom class to support native functions such as print, length, comparison, support for arithmetic operations, type conversion, and many more.

### Typing :

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically-typed, Python is strongly-typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using [classes](https://en.wikipedia.org/wiki/Class_(computer_science)), which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, SpamClass() or EggsClass()), and the classes are instances of the metaclass type (itself an instance of itself), allowing meta-programming and reflection.

Before version 3.0, Python had two kinds of classes: old-style and new-style.The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from object and are instances of type). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long-term plan is to support gradual typing and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named mypy supports compile-time type checking.

**23. METHODOLOGY**

Preprocessing and Training the model (CNN): The dataset is preprocessed such as Image reshaping, resizing and conversion to an array form. Similar processing is also done on the test image. A dataset consisting of about 3 different benign, malignant and normal is obtained, out of which any image can be used as a test image for the software.

CNN Weights

Breast cancer segmentation

CNN train

Raw image

Build a sequential model

The train dataset is used to train the model (CNN) so that it can identify the test image and the disease it has CNN has different layers that are Dense, Dropout, Activation, Flatten, Convolution2D, and MaxPooling2D. After the model is trained successfully, the software can identify the Breast, Pneumonia, tuberculosis Classification image contained in the dataset. After successful training and preprocessing, comparison of the test image and trained model takes place to predict the Breast, Pneumonia, tuberculosis.

**CNN Model steps:**

**Conv2d:**

The 2D convolution is a fairly simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel “slides” over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

The kernel repeats this process for every location it slides over, converting a 2D matrix of features into yet another 2D matrix of features. The output features are essentially, the weighted sums (with the weights being the values of the kernel itself) of the input features located roughly in the same location of the output pixel on the input layer.

Whether or not an input feature falls within this “roughly same location”, gets determined directly by whether it’s in the area of the kernel that produced the output or not. This means the size of the kernel directly determines how many (or few) input features get combined in the production of a new output feature.

This is all in pretty stark contrast to a fully connected layer. In the above example, we have 5×5=25 input features, and 3×3=9 output features. If this were a standard fully connected layer, you’d have a weight matrix of 25×9 = 225 parameters, with every output feature being the weighted sum of every single input feature. Convolutions allow us to do this transformation with only 9 parameters, with each output feature, instead of “looking at” every input feature, only getting to “look” at input features coming from roughly the same location. Do take note of this, as it’ll be critical to our later discussion.

**MaxPooling2D layer**

Down samples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by pool\_size) for each channel of the input. The window is shifted by strides along each dimension.

The resulting output, when using the "valid" padding option, has a spatial shape (number of rows or columns) of: output\_shape = math.floor((input\_shape - pool\_size) / strides) + 1 (when input\_shape >= pool\_size)

The resulting output shape when using the "same" padding option is: output\_shape = math.floor((input\_shape - 1) / strides) + 1

**Arguments**

• pool\_size: integer or tuple of 2 integers, window size over which to take the maximum. (2, 2) will take the max value over a 2x2 pooling window. If only one integer is specified, the same window length will be used for both dimensions.

• strides: Integer, tuple of 2 integers, or None. Strides values. Specifies how far the pooling window moves for each pooling step. If None, it will default to pool\_size.

• padding: One of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input.

• data\_format: A string, one of channels\_last (default) or channels\_first. The ordering of the dimensions in the inputs. channels\_last corresponds to inputs with shape (batch, height, width, channels) while channels\_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image\_data\_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels\_last".

**Input shape**

• If data\_format='channels\_last': 4D tensor with shape (batch\_size, rows, cols, channels).

• If data\_format='channels\_first': 4D tensor with shape (batch\_size, channels, rows, cols).

**Output shape**

• If data\_format='channels\_last': 4D tensor with shape (batch\_size, pooled\_rows, pooled\_cols, channels).

• If data\_format='channels\_first': 4D tensor with shape (batch\_size, channels, pooled\_rows, pooled\_cols).

**Flatten layer**

It is used to flatten the dimensions of the image obtained after convolving it. Dense: It is used to make this a fully connected model and is the hidden layer. Dropout: It is used to avoid over fitting on the dataset and dense is the output layer contains only one neuron which decide to which category image belongs.

Flatten is used to flatten the input. For example, if flatten is applied to layer having input shape as (batch\_size, 2,2), then the output shape of the layer will be (batch\_size, 4)

Flatten has one argument as follows

keras.layers.Flatten(data\_format = None)

data\_format is an optional argument and it is used to preserve weight ordering when switching from one data format to another data format. It accepts either channels\_last or channels\_first as value. channels\_last is the default one and it identifies the input shape as (batch\_size, ..., channels) whereas channels\_first identifies the input shape as (batch\_size, channels, ...)

**Dense layer**

Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use\_bias is True). These are all attributes of Dense.

Note: If the input to the layer has a rank greater than 2, then Dense computes the dot product between the inputs and the kernel along the last axis of the inputs and axis 0 of the kernel (using tf.tensordot). For example, if input has dimensions (batch\_size, d0, d1), then we create a kernel with shape (d1, units), and the kernel operates along axis 2 of the input, on every sub-tensor of shape (1, 1, d1) (there are batch\_size \* d0 such sub-tensors). The output in this case will have shape (batch\_size, d0, units).

Besides, layer attributes cannot be modified after the layer has been called once (except the trainable attribute). When a popular kwarg input\_shape is passed, then keras will create an input layer to insert before the current layer. This can be treated equivalent to explicitly defining an InputLayer.

**Arguments**

• units: Positive integer, dimensionality of the output space.

• activation: Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).

• use\_bias: Boolean, whether the layer uses a bias vector.

• kernel\_initializer: Initializer for the kernel weights matrix.

• bias\_initializer: Initializer for the bias vector.

• kernel\_regularizer: Regularizer function applied to the kernel weights matrix.

• bias\_regularizer: Regularizer function applied to the bias vector.

• activity\_regularizer: Regularizer function applied to the output of the layer (its "activation").

• kernel\_constraint: Constraint function applied to the kernel weights matrix.

• bias\_constraint: Constraint function applied to the bias vector.

**Input shape**

N-D tensor with shape: (batch\_size, ..., input\_dim). The most common situation would be a 2D input with shape (batch\_size, input\_dim).

**Output shape**

N-D tensor with shape: (batch\_size, ..., units). For instance, for a 2D input with shape (batch\_size, input\_dim), the output would have shape (batch\_size, units).

# **Dropout layer**

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

Note that the Dropout layer only applies when training is set to True such that no values are dropped during inference. When using model.fit, training will be appropriately set to True automatically, and in other contexts, you can set the kwarg explicitly to True when calling the layer.

(This is in contrast to setting trainable=False for a Dropout layer. trainable does not affect the layer's behavior, as Dropout does not have any variables/weights that can be frozen during training.)

**Arguments**

* **rate**: Float between 0 and 1. Fraction of the input units to drop.
* **noise\_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape (batch\_size, timesteps, features) and you want the dropout mask to be the same for all timesteps, you can use noise\_shape=(batch\_size, 1, features).

**seed**: A Python integer to use as random seed.

**Image Data Generator:**

It is that rescales the image, applies shear in some range, zooms the image and does horizontal flipping with the image. This Image Data Generator includes all possible orientation of the image.

**Training Process:**

train\_datagen.flow\_from\_directory is the function that is used to prepare data from the train\_dataset directory Target\_size specifies the target size of the image. Test\_datagen.flow\_from\_directory is used to prepare test data for the model and all is similar as above. fit\_generator is used to fit the data into the model made above, other factors used are steps\_per\_epochs tells us about the number of times the model will execute for the training data.

**Epochs:**

It tells us the number of times model will be trained in forward and backward pass.

**Validation process:**

Validation\_data is used to feed the validation/test data into the model. Validation\_steps denotes the number of validation/test samples.

### 24. Artificial Neural Networks

Artificial Neural Networks contain artificial neurons which are called **units**. These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. Commonly, Artificial Neural Network has an input layer, an output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyze or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer. Finally, the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided.

In the majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit. As the data transfers from one unit to another, the neural network learns more and more about the data which eventually results in an output from the output layer.

**25. ARCHITECTURE OF CNN**

**CONVOLUTIONAL NEURAL NETWORK:**

A Convolutional neural network (CNN) is one type of Artificial Neural Network. A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are used mainly for image processing, classification, segmentation and also for other auto correlated data.

**Models API:**

**There are three ways to create Keras models:**

• The Sequential model, which is very straightforward (a simple list of layers), but is limited to single-input, single-output stacks of layers (as the name gives away).

• The Functional API, which is an easy-to-use, fully-featured API that supports arbitrary model architectures. For most people and most use cases, this is what you should be using. This is the Keras "industry strength" model.

• Model subclassing, where you implement everything from scratch on your own. Use this if you have complex, out-of-the-box research use cases.

**Types of Keras Models**

**Models in keras are available in two types:**

• Keras Sequential Model

• Keras Functional API

**1. Sequential Model in Keras**

It allows us to create models layer by layer in sequential order. But it does not allow us to create models that have multiple inputs or outputs.

It is best for simple stack of layers which have 1 input tensor and 1 output tensor.

This model is not suited when any of the layer in the stack has multiple inputs or outputs. Even if we want non-linear topology, it is not suited.

**2. Functional API in Keras**

It provides more flexibility to define a model and add layers in keras. Functional API allows us to create models that have multiple input or output.

It also allows us to share these layers. In other words. we can make graphs of layers using Keras functional API.

As functional API is a data structure, it is easy to save it as a single file that helps in recreating the exact model without having the original code. Also its easy to model the graph here and access its nodes as well.

**Model Subclassing in Keras**

Sequential model does not allow you much flexibility to create your models. Functional API also only has a little of customization available for you. But you may create your own fully-customizable models in Keras. This is done by subclassing the Model class and implementing a call method.

Input() is used to instantiate a Keras tensor.

A Keras tensor is a symbolic tensor-like object, which we augment with certain attributes that allow us to build a Keras model just by knowing the inputs and outputs of the model.

For instance, if a, b and c are Keras tensors, it becomes possible to do: model = Model(input=[a, b], output=c)

# **kernels:**

Each convolutional layer contains a series of filters known as convolutional kernels. The filter is a matrix of integers that are used on a subset of the input pixel values, the same size as the kernel. Each pixel is multiplied by the corresponding value in the kernel, then the result is summed up for a single value for simplicity representing a grid cell, like a pixel, in the output channel/feature map. These are linear transformations, each convolution is a type of affine function.

In computer vision the input is often a 3 channel RGB image. For simplicity, if we take a greyscale image that has one channel (a two dimensional matrix) and a 3x3 convolutional kernel (a two dimensional matrix). The kernel strides over the input matrix of numbers moving horizontally column by column, sliding/scanning over the first rows in the matrix containing the images pixel values. Then the kernel strides down vertically to subsequent rows. Note, the filter may stride over one or several pixels at a time, this is detailed further below.

In other non-vision applications, a one dimensional convolution may slide vertically over an input matrix.

**Padding:**

To handle the edge pixels there are several approaches:

* Losing the edge pixels
* Padding with zero value pixels
* Reflection padding

Reflection padding is by far the best approach, where the number of pixels needed for the convolutional kernel to process the edge pixels are added onto the outside copying the pixels from the edge of the image. For a 3x3 kernel, one pixel needs to be added around the outside, for a 7x7 kernel then three pixels would be reflected around the outside. The pixels added around each side is the dimension, halved and rounded down.

Traditionally in many research papers, the edge pixels are just ignored, which loses a small proportion of the data and this gets increasing worse if there are many deep convolutional layers. For this reason, I could not find existing diagrams to easily convey some of the points here without being misleading and confusing stride 1 convolutions with stride 2 convolutions.

With padding, the output from a input of width w and height h would be width w and height h (the same as the input with a single input channel), assuming the kernel takes a stride of one pixel at a time.

# **Strides:**

It is common to use a stride two convolution rather than a stride one convolution, where the convolutional kernel strides over 2 pixels at a time, for example our 3x3 kernel would start at position (1,1), then stride to (1,3), then to 1, 5) and so on, halving the size of the output channel/feature map, compared to the convolutional kernel taking strides of one.

With padding, the output from an input of width w, height h and depth 3 would be the ceiling of width w/2, height h/2 and depth 1, as the kernel outputs a single summed output from each stride.

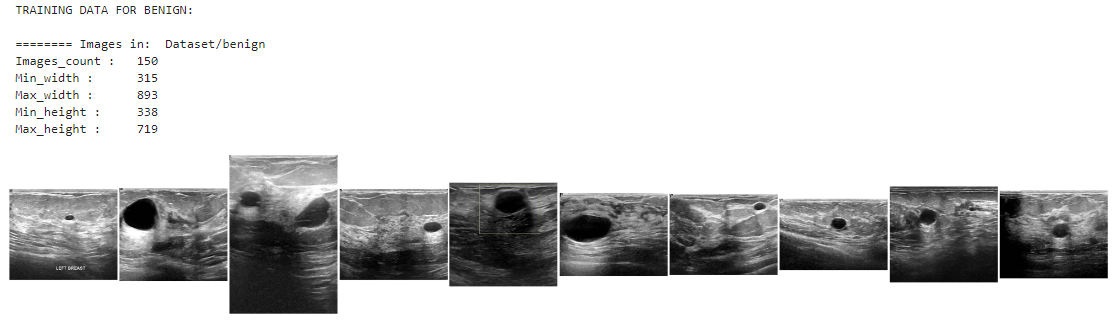
# **Rectified Linear Unit (ReLU):**

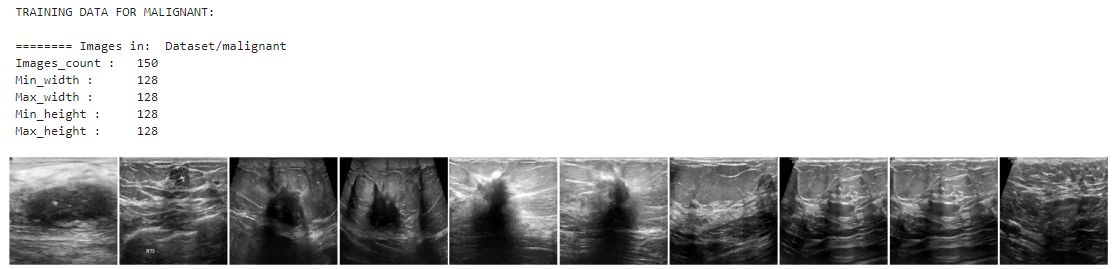
A Rectified Linear Unit is used as a non-linear activation function. A ReLU says if the value is less than zero, round it up to zero.

# **Normalisation:**

Normalisation is the process of subtracting the mean and dividing by the standard deviation. It transforms the range of the data to be between -1 and 1 making the data use the same scale, sometimes called Min-Max scaling.

It is common to normalize the input features, standardising the data by removing the mean and scaling to unit variance. It is often important the input features are centred around zero and have variance in the same order. With some data, such as images the data is scaled so that it’s range is between 0 and 1, most simply dividing the pixel values by 255.





**Batch normalisation:**

Batch normalisation has the benefits of helping to make a network output more stable predictions, reduce overfitting through regularisation and speeds up training by an order of magnitude.

Batch normalisation is the process of carrying normalisation within the scope activation layer of the current batch, subtracting the mean of the batch’s activations and dividing by the standard deviation of the batches activations.

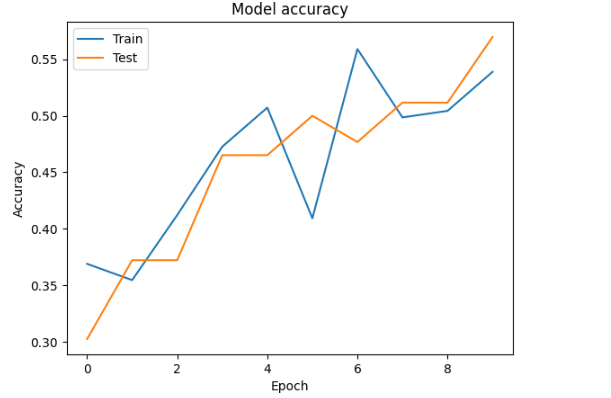
This is necessary as even after normalizing the input as some activations can be higher, which can cause the subsequent layers to act abnormally and makes the network less stable.

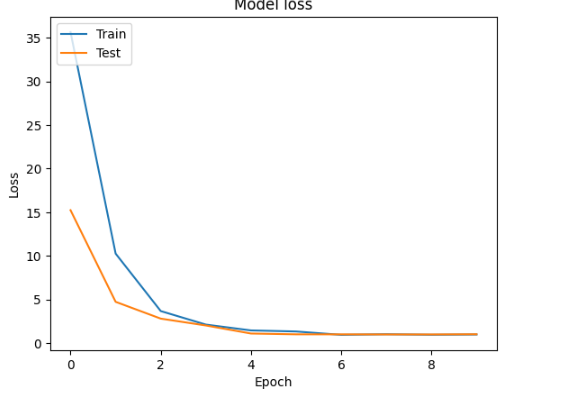
Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

Importantly, batch normalization works differently during training and during inference.

**During training** (i.e. when using fit() or when calling the layer/model with the argument training=True), the layer normalizes its output using the mean and standard deviation of the current batch of inputs. That is to say, for each channel being normalized, the layer returns gamma \* (batch - mean(batch)) / sqrt(var(batch) + epsilon) + beta, where:

* epsilon is small constant (configurable as part of the constructor arguments)
* gamma is a learned scaling factor (initialized as 1), which can be disabled by passing scale=False to the constructor.
* beta is a learned offset factor (initialized as 0), which can be disabled by passing center=False to the constructor.





**During inference** (i.e. when using evaluate() or predict() or when calling the layer/model with the argument training=False (which is the default), the layer normalizes its output using a moving average of the mean and standard deviation of the batches it has seen during training. That is to say, it returns gamma \* (batch - self.moving\_mean) / sqrt(self.moving\_var + epsilon) + beta.

self.moving\_mean and self.moving\_var are non-trainable variables that are updated each time the layer in called in training mode, as such:

* moving\_mean = moving\_mean \* momentum + mean(batch) \* (1 - momentum)
* moving\_var = moving\_var \* momentum + var(batch) \* (1 - momentum)

As such, the layer will only normalize its inputs during inference after having been trained on data that has similar statistics as the inference data.

**Arguments**

* **axis**: Integer, the axis that should be normalized (typically the features axis). For instance, after a Conv2D layer with data\_format="channels\_first", set axis=1 in BatchNormalization.
* **momentum**: Momentum for the moving average.
* **epsilon**: Small float added to variance to avoid dividing by zero.
* **center**: If True, add offset of beta to normalized tensor. If False, beta is ignored.
* **scale**: If True, multiply by gamma. If False, gamma is not used. When the next layer is linear (also e.g. nn.relu), this can be disabled since the scaling will be done by the next layer.
* **beta\_initializer**: Initializer for the beta weight.
* **gamma\_initializer**: Initializer for the gamma weight.
* **moving\_mean\_initializer**: Initializer for the moving mean.
* **moving\_variance\_initializer**: Initializer for the moving variance.
* **beta\_regularizer**: Optional regularizer for the beta weight.
* **gamma\_regularizer**: Optional regularizer for the gamma weight.
* **beta\_constraint**: Optional constraint for the beta weight.
* **gamma\_constraint**: Optional constraint for the gamma weight.

**Call arguments**

* **inputs**: Input tensor (of any rank).
* **training**: Python boolean indicating whether the layer should behave in training mode or in inference mode.

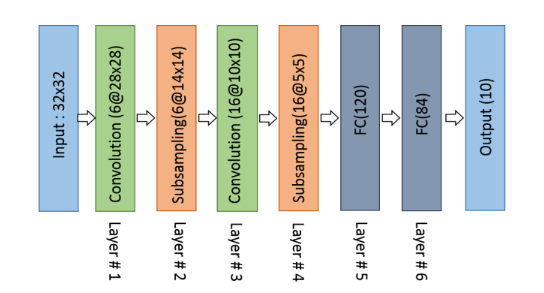
**26. TYPES OF CNN:**

* ALEXNET
* LENET

**26.1 AlexNET:**

AlexNet architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer. 2. Each convolutional layer consists of convolutional filters and a nonlinear activation function ReLU. 3. The pooling layers are used to perform max pooling. AlexNet contained eight layers; the first five were convolutional layers, some of them followed by max-pooling layers, and the last three were fully connected layers. It used the non-saturating ReLU activation function, which showed improved training performance over tanh and sigmoid.

**Architecture of AlexNet:**



Architecture of AlexNet

**Convolutional layers:**

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

**Pooling layers:**

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

**Dense or Fully connected layers:**

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

**Convolutional layers:**

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

**Pooling layers:**

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

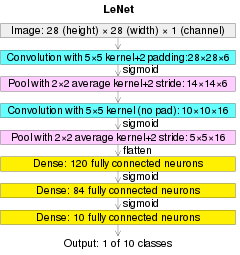
**Dense or Fully connected layers:**

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

**26.2 LENET**

As a representative of the early convolutional neural network, LeNet possesses the basic units of convolutional neural network, such as convolutional layer, pooling layer and full connection layer, laying a foundation for the future development of convolutional neural network. As shown in the figure (input image data with 32\*32 pixels) LeNet-5 consists of seven layers. In addition to input, every other layer can [train parameters](https://en.wikipedia.org/wiki/Train_parameter). In the figure, Cx represents convolution layer, Sx represents sub-sampling layer, Fx represents complete connection layer, and x represents layer index.

**Architecture of LeNet:**



Architecture of LeNet

**Convolutional layers:**

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

**Pooling layers:**

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

**Dense or Fully connected layers:**

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

**Convolutional layers:**

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

**Pooling layers:**

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

**Dense or Fully connected layers:**

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

27. **LIST OF MODULES**

1. Data Analysis

2. Manual Architecture

3. LeNet Architecture

4. U-Net Architecture

5. Deployment

**28. MODULE DESCRIPTION**

**28.1 IMPORT THE GIVEN IMAGE FROM DATASET:**

## We have to import our data set using keras preprocessing image data generator function also we create size, rescale, range, zoom range, horizontal flip. Then we import our image dataset from folder through the data generator function. Here we set train, test, and validation also we set target size, batch size and class-mode from this function we have to train using our own created network by adding layers of CNN.

**28.2 TO TRAIN THE MODULE BY GIVEN**

**1. Data Analysis**

Data analysis is the process of cleaning, changing, and processing raw data, and extracting actionable, relevant information that helps businesses make informed decisions. The procedure helps reduce the risks inherent in decision-making by providing useful insights. The data analysis process, or alternately, data analysis steps, involves gathering all the information, processing it, exploring the data, and using it to find patterns and other insights.

In data analysis we analyse the data that how the image data is available. We analyse how many data are available and we check whether the normal data is available corresponding to the mask data.

**Manual Architecture :**

Creating a manual architecture for image segmentation typically involves designing a neural network or algorithm that can process an input image and produce pixel-level segmentation masks or regions of interest. Here's a simplified manual architecture for image segmentation:

1. Input Image: The architecture takes an input image as its primary input. This can be a grayscale or color image depending on your application.

2. Preprocessing: Preprocess the input image to enhance features and reduce noise. Common preprocessing steps include resizing, normalization, and data augmentation.

3. Convolutional Neural Network (CNN): Use a convolutional neural network as the backbone of your segmentation architecture. CNNs are highly effective at capturing local patterns and spatial features in images.

- You can use a pre-trained CNN architecture like VGG, ResNet, or U-Net as a starting point, or design a custom architecture.

4. Encoder-Decoder Architecture: Many segmentation architectures follow an encoder-decoder structure:

- Encoder: The encoder part of the network extracts features from the input image through a series of convolutional layers. These layers reduce spatial dimensions while increasing the number of feature maps.

- Decoder: The decoder part of the network upsamples the feature maps to the original image size while reducing the number of channels. This process helps generate the segmentation mask.

5. Skip Connections: To improve segmentation accuracy, consider adding skip connections that concatenate feature maps from the encoder to the corresponding layers in the decoder. This allows the network to capture both high-level and low-level features.

6. Convolutional Transpose (Deconvolution) Layers: Use convolutional transpose layers (sometimes called deconvolution layers) to upsample feature maps in the decoder. These layers expand the spatial resolution of the feature maps.

7. Activation Function: Apply an activation function, typically a softmax or sigmoid, to the final layer of the decoder to produce the segmentation mask. For binary segmentation, sigmoid is often used; for multi-class segmentation, softmax is common.

8. Loss Function: Define an appropriate loss function to measure the difference between the predicted segmentation mask and the ground truth mask. Common loss functions for segmentation include binary cross-entropy and categorical cross-entropy.

9. Optimization Algorithm: Use an optimization algorithm like stochastic gradient descent (SGD), Adam, or RMSprop to update the network's weights and minimize the loss function.

10. Training Data: Train the network using a dataset of annotated images. The dataset should include input images and corresponding segmentation masks.

11. Post-processing: Apply post-processing techniques if needed, such as morphological operations (erosion, dilation) to refine the segmentation mask.

12. Inference: During inference, feed an unseen image through the trained network to obtain the segmentation mask.

13. Evaluation: Evaluate the segmentation accuracy using metrics like Intersection over Union (IoU), Dice coefficient, or pixel accuracy.

This manual architecture provides a high-level overview of the components and steps involved in image segmentation. Depending on your specific task and dataset, you may need to customize and fine-tune the architecture for optimal results. Additionally, modern architectures may incorporate more advanced techniques, such as attention mechanisms or conditional random fields, to further enhance segmentation performance.

**Le-Net Archietecture:**

LeNet, short for "LeNet-5," is a classic convolutional neural network (CNN) architecture developed by Yann LeCun in the early 1990s. While LeNet is renowned for its role in image classification tasks, it can be adapted for image segmentation, including applications like breast cancer segmentation. Here's a high-level overview of how LeNet can be modified for this purpose:

1. Input Image: The input to the network is a breast cancer image, typically in grayscale or color, depending on your dataset and requirements.

2. Preprocessing: Preprocess the input images as needed. Common preprocessing steps include resizing to a consistent input size, normalization, and data augmentation.

3. LeNet Architecture: LeNet consists of a series of convolutional and pooling layers followed by fully connected layers. For breast cancer segmentation, you will need to modify the architecture to produce pixel-wise segmentation instead of classification.

4. Convolutional Layers: Retain the convolutional layers from the original LeNet architecture. These layers are responsible for learning hierarchical features from the input image.

5. Pooling Layers: Keep the max-pooling layers from the original LeNet. Pooling helps reduce the spatial dimensions of feature maps.

6. Encoder-Decoder Modification: Modify the fully connected layers of the original LeNet into an encoder-decoder architecture for segmentation. Remove the final fully connected layers that were used for classification.

7. Decoder: Design a decoder portion that mirrors the encoder. It consists of transposed convolutional (also known as deconvolutional) layers that upsample the feature maps back to the original image size.

8. Activation Function: Apply an appropriate activation function, such as sigmoid (for binary segmentation) or softmax (for multi-class segmentation), to the output layer of the decoder to obtain pixel-wise segmentation masks.

9. Loss Function: Define a suitable loss function for segmentation tasks, such as binary cross-entropy or categorical cross-entropy, depending on the nature of your data.

10. Optimization Algorithm: Utilize an optimization algorithm, like stochastic gradient descent (SGD), Adam, or RMSprop, to train the network by minimizing the defined loss function.

11. Training Data: Train the network using a labeled dataset of breast cancer images and their corresponding pixel-wise segmentation masks.

12. Post-processing (Optional): Depending on the quality of the segmentation masks, you may apply post-processing techniques such as morphological operations (erosion, dilation) or connected component analysis to refine the segmentations.

13. Evaluation: Evaluate the segmentation performance using standard metrics like Intersection over Union (IoU), Dice coefficient, or pixel accuracy.

It's essential to note that while LeNet can serve as a starting point for segmentation tasks, modern architectures, such as U-Net or FCN (Fully Convolutional Network), are generally more suitable for segmentation due to their specialized design for pixel-wise predictions. These architectures often yield better results and are more commonly used in contemporary computer vision tasks, including medical image segmentation.

**U-Net Architecture:**

The U-Net architecture is a popular deep learning architecture for image segmentation tasks, including breast cancer segmentation. It was originally developed for biomedical image segmentation and has since found applications in various medical image analysis tasks. The name "U-Net" is derived from the U-shaped architecture of the network.

Here's an overview of the U-Net architecture for breast cancer segmentation:

**Encoder-Decoder Structure:**

U-Net follows an encoder-decoder structure. It consists of two main parts: the encoder and the decoder.

The encoder captures features from the input image at multiple scales by using convolutional and pooling layers. It gradually reduces the spatial dimensions while increasing the number of feature maps.

The decoder then takes these features and upsamples them to the original image size while reducing the number of feature maps. This helps generate a detailed segmentation mask.

**Skip Connections:**

One of the key innovations of U-Net is the use of skip connections that connect corresponding layers between the encoder and decoder.

These skip connections allow the network to capture both high-level and low-level features, which is crucial for accurate segmentation.

Skip connections concatenate feature maps from the encoder to the corresponding layers in the decoder.

**Contracting and Expansive Paths:**

The encoder path is often referred to as the contracting path because it reduces spatial dimensions.

The decoder path is called the expansive path because it increases the spatial dimensions.

Skip connections connect the contracting and expansive paths, facilitating the flow of information between them.

**Final Layer:**

The final layer of the U-Net architecture typically consists of a convolutional layer with a softmax activation function for multi-class segmentation or a sigmoid activation function for binary segmentation.

The output of this layer is the segmentation mask, where each pixel is classified into the desired classes (e.g., tumor or background).

**Loss Function:**

Common loss functions for U-Net-based segmentation tasks include binary cross-entropy loss for binary segmentation or categorical cross-entropy loss for multi-class segmentation.

The loss function measures the difference between the predicted segmentation mask and the ground truth mask.

**Training Data:**

To train a U-Net model for breast cancer segmentation, you need a dataset of annotated breast images. The dataset should include input ultrasound images or other relevant images and corresponding pixel-level segmentation masks indicating the regions of interest (e.g., tumors).

**Inference:**

During inference, you feed an unseen breast image through the trained U-Net model to obtain the segmentation mask, which highlights the areas of interest, such as tumors or lesions.

**Post-processing:**

Post-processing steps, such as morphological operations (e.g., erosion, dilation) or connected component analysis, can be applied to refine the segmentation mask and remove any artifacts.

U-Net has been widely adopted in medical image segmentation due to its ability to capture fine details and its effectiveness in handling limited training data. Researchers and practitioners often customize U-Net architectures by adjusting the number of layers, filter sizes, and skip connections to suit the specific requirements of their breast cancer segmentation tasks. Additionally, data augmentation techniques are commonly used to increase the diversity of training data and improve model generalization.

**29. DEPLOY**

**Deploying the model in Django Framework and predicting output**

In this module the trained deep learning model is converted into hierarchical data format file (.h5 file) which is then deployed in our django framework for providing better user interface and predicting the output.

**29.1 Django**

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Django helps you write software that is:

**Complete**

Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, follows consistent design principles, and has extensive and [up-to-date documentation](https://docs.djangoproject.com/en/stable/).

**Versatile**

Django can be (and has been) used to build almost any type of website — from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format (including HTML, RSS feeds, JSON, XML, etc). The site you are currently reading is built with Django!  
  
Internally, while it provides choices for almost any functionality you might want (e.g. several popular databases, templating engines, etc.), it can also be extended to use other components if needed.

**Secure**

Django helps developers avoid many common security mistakes by providing a framework that has been engineered to "do the right things" to protect the website automatically. For example, Django provides a secure way to manage user accounts and passwords, avoiding common mistakes like putting session information in cookies where it is vulnerable (instead cookies just contain a key, and the actual data is stored in the database) or directly storing passwords rather than a password hash.  
  
A password hash is a fixed-length value created by sending the password through a [cryptographic hash function](https://en.wikipedia.org/wiki/Cryptographic_hash_function). Django can check if an entered password is correct by running it through the hash function and comparing the output to the stored hash value. However due to the "one-way" nature of the function, even if a stored hash value is compromised it is hard for an attacker to work out the original password.  
  
Django enables protection against many vulnerabilities by default, including SQL injection, cross-site scripting, cross-site request forgery and clickjacking (see [Website security](https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Website_security) for more details of such attacks).

**Scalable**

Django uses a component-based “[shared-nothing](https://en.wikipedia.org/wiki/Shared_nothing_architecture)” architecture (each part of the architecture is independent of the others, and can hence be replaced or changed if needed). Having a clear separation between the different parts means that it can scale for increased traffic by adding hardware at any level: caching servers, database servers, or application servers. Some of the busiest sites have successfully scaled Django to meet their demands (e.g. Instagram and Disqus, to name just two).

**Maintainable**

Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In particular, it makes use of the Don't Repeat Yourself (DRY) principle so there is no unnecessary duplication, reducing the amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a lower level, groups related code into modules (along the lines of the [Model View Controller (MVC)](https://developer.mozilla.org/en-US/docs/Glossary/MVC) pattern).

**Portable**

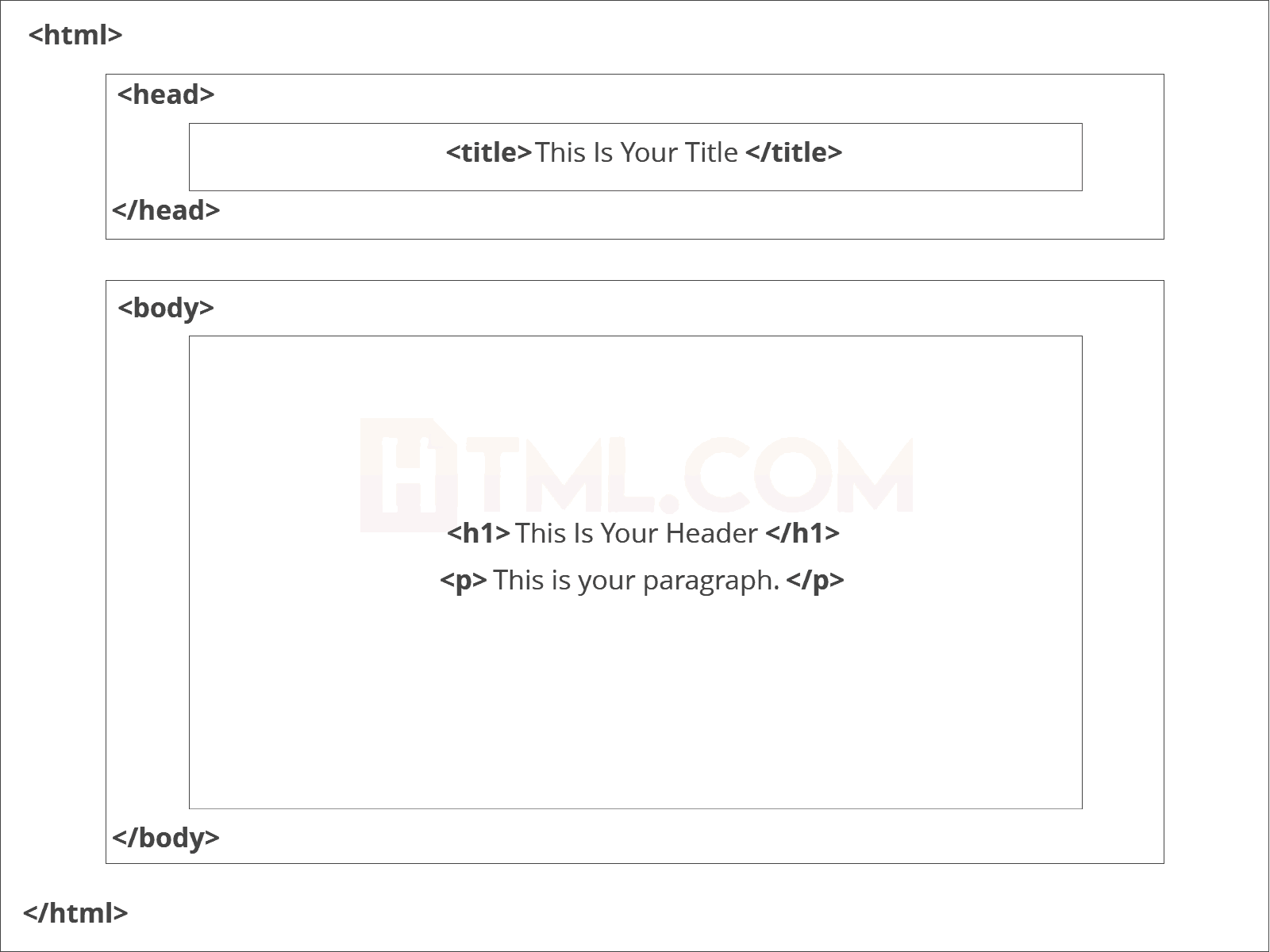
Django is written in Python, which runs on many platforms. That means that you are not tied to any particular server platform, and can run your applications on many flavours of Linux, Windows, and Mac OS X. Furthermore, Django is well-supported by many web hosting providers, who often provide specific infrastructure and documentation for hosting Django sites.

**29.2 HTML**

**HTML** stands for HyperText Markup Language. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A markup language is used to define the text document within tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

#### **Basic Construction of an HTML Page**

These tags should be placed underneath each other **at the top of every HTML page** that you create.



[<!DOCTYPE html>](https://html.com/tags/doctype/) — This tag **specifies the language** you will write on the page. In this case, the language is HTML 5.

[<html>](https://html.com/tags/html/) — This tag signals that from here on we are going to write in HTML code.

[<head>](https://html.com/tags/head/) — This is where all the **metadata for the page** goes — stuff mostly meant for search engines and other computer programs.

[<body>](https://html.com/tags/body/) — This is where the **content of the page** goes.

#### **Further Tags**

Inside the <head> tag, there is one tag that is always included: <title>, but there are others that are just as important:

[**<title>**](https://html.com/tags/title/)

This is where we **insert the page name** as it will appear at the top of the browser window or tab.

[**<meta>**](https://html.com/tags/meta/)

This is where information about the document is stored: character encoding, name (page context), description.

**HeadTag**  
<head>

<title>My First Webpage</title>

<meta charset="UTF-8">

<meta name="description" content="This field contains information about your page. It is usually around two sentences long.">.

<meta name="author" content="Conor Sheils">

</header>

### Adding Content

Next, we will make [<body>](https://html.com/tags/body/) tag.

The HTML <body> is where we add the content which is designed for viewing by human eyes.

This includes **text, images, tables, forms**and everything else that we see on the internet each day.

#### How to Add HTML Headings To Your Web Page

In HTML, [headings](https://html.com/tags/heading/) are written in the following elements:

* <h1>
  + <h2>
    - <h3>
      * <h4>
        + <h5>

<h6>

As you might have guessed <h1> and <h2> should be used for the most important titles, while the remaining tags should be used for sub-headings and less important text.

**Search engine bots use this order** when deciphering which information is most important on a page.

##### Creating Your Heading

Let’s try it out. On a new line in the HTML editor, type:

<h1>Welcome to My Page</h1>

And hit save. We will save this file as “index.html” in a new folder called “my webpage.”

***The Moment of Truth****: Click the newly saved file and your first ever web page should open in your default browser. It may not be pretty it’s yours… all yours. \*Evil laugh\**

**Add Text In HTML**

Adding text to our HTML page is simple using an element opened with the tag [<p>](https://html.com/tags/p/) which **creates a new paragraph**. We place all of our regular text inside the element <p>.

| **Element** | **Meaning** | **Purpose** |
| --- | --- | --- |
| **<b>** | Bold | Highlight important information |
| **<strong>** | Strong | Similarly to bold, to highlight key text |
| **<i>** | Italic | To denote text |
| **<em>** | Emphasised Text | Usually used as image captions |
| **<mark>** | Marked Text | Highlight the background of the text |
| **<small>** | Small Text | To shrink the text |
| **<strike>** | Striked Out Text | To place a horizontal line across the text |
| **<u>** | Underlined Text | Used for links or text highlights |
| **<ins>** | Inserted Text | Displayed with an underline to show an inserted text |
| **<sup>** | Superscript Text | Another typographical presentation style |
|  |  |  |

When we write text in HTML, we also have a number of other elements we can use to **control the text or make it appear in a certain way**.

#### **Add Links In HTML**

As you may have noticed, the internet is made up of lots of [links](https://html.com/anchors-links/).

Almost everything you click on while surfing the web is a link **takes you to another page** within the website you are visiting or to an external site.

Links are included in an attribute opened by the [**<a>**](https://html.com/tags/a/) tag. This element is the first that we’ve met which uses an attribute and so it **looks different to previously mentioned tags**.

<a href="[http://www.google.com](https://www.google.com/)">Google</a>

**Image Tag**

In today’s modern digital world, [images](https://html.com/blog/100-legal-sources-free-stock-images/) are everything. The [**<img>**](https://html.com/tags/img/) tag has everything you need to display images on your site. Much like the <a> anchor element, <img> also contains an attribute.

The attribute *features information* for your computer regarding the **source**, **height**, **width** and **alt text** of the image

<img src="yourimage.jpg" alt="Describe the image" height="X" width="X">

**29.3 CSS**

CSS stands for Cascading Style Sheets. It is the language for describing the presentation of Web pages, including colours, layout, and fonts, thus making our web pages presentable to the users.

CSS is designed to make style sheets for the web. It is independent of HTML and can be used with any XML-based markup language. Now let’s try to break the acronym:

* Cascading: Falling of Styles
* Style: Adding designs/Styling our HTML tags
* Sheets: Writing our style in different documents

## **CSS Syntax**

Selector {

Property 1 : value;

Property 2 : value;

Property 3 : value;

}

For example:

h1

{

Color: red;

Text-align: center;

}

#unique

{

color: green;

}

* Selector: selects the element you want to target
* Always remains the same whether we apply internal or external styling
* There are few basic selectors like tags, id’s, and classes
* All forms this key-value pair
* Keys: properties(attributes) like color, font-size, background, width, height,etc
* Value: values associated with these properties

## **CSS Comment**

* Comments don’t render on the browser
* Helps to understand our code better and makes it readable.
* Helps to debug our code
* Two ways to  comment:
  + Single line

## **CSS How-To**

* There are 3 ways to write CSS in our HTML file.
  + Inline CSS
  + Internal CSS
  + External CSS
* Priority order

Inline > Internal > External

**Inline CSS**

* Before CSS this was the only way to apply styles
* Not an efficient way to write as it has a lot of redundancy
* Self-contained
* Uniquely applied on each element
* The idea of separation of concerns was lost
* Example:

<h3 style=” color:red”> Have a great day </h3>

<p style =” color: green”> I did this , I did that </p>

**Internal CSS**

* With the help of style tag, we can apply styles within the HTML file
* Redundancy is removed
* But the idea of separation of concerns still lost
* Uniquely applied on a single document
* Example:

< style>

h1{

color:red;

}

</style>

<h3> Have a great day </h3>

**External CSS**

* With the help of <link> tag in the head tag, we can apply styles
* Reference is added
* File saved with .css extension
* Redundancy is removed
* The idea of separation of concerns is maintained
* Uniquely applied to each document
* Example:

<head>

<link rel=”stylesheet” type=”text/css” href=”name of the Css file”>

</head>

h1{

color:red; //.css file

}

## **CSS Selectors**

* The selector is used to target elements and apply CSS
* Three simple selectors
  + Element Selector
  + Id Selector
  + Class Selector
* Priority of Selectors

## **CSS Colors**

* There are different colouring schemes in CSS
* **RGB**-This starts with RGB and takes 3 parameter
* **HEX**-Hex code starts with # and comprises of 6 numbers which are further divided into 3 sets
* **RGBA**-This starts with RGB and takes 4 parameter

## **CSS Background**

* There are different ways by which CSS can have an effect on HTML elements
* Few of them are as follows:
  + Color – used to set the color of the background
  + Repeat – used to determine if the image has to repeat or not and if it is repeating then how it should do that
  + Image – used to set an image as the background
  + Position – used to determine the position of the image
  + Attachment – It basically helps in controlling the mechanism of scrolling

## **CSS BoxModel**

* Every element in CSS can be represented using the BOX model
* It allows us to add a border and define space between the content
* It helps the developer to develop and manipulate the elements
* It consists of 4 edges
  + Content edge – It comprises of the actual content
  + Padding edge – It lies in between content and border edge
  + Border edge – Padding is followed by the border edge

Margin edge – It is an outside border and controls the margin of the element.

30. **CODING:**

**Module 1:**

**Classification:**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Convolution2D

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Activation

import tensorflow

import tensorflow as tf

print(tf.\_\_version\_\_)

import os

import glob

from PIL import Image

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Convolution2D

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Activation

import os

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from os import listdir

import glob

from PIL import Image

import cv2

from random import randrange

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping

from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, MaxPool2D, Conv2DTranspose, Concatenate, Input

from tensorflow.keras.models import Model

import warnings

warnings.filterwarnings('ignore')

ori\_images = 'Dataset/Train/Images'

mask\_images = 'Dataset/Train/Masks'

def load\_data(dataset\_path, split=0.2):

images = [f for f in glob.glob(ori\_images + "\*\*/\*.png", recursive=True)]

masks = [f for f in glob.glob(mask\_images + "\*\*/\*.png", recursive=True)]

test\_size = int(len(images) \* split)

train\_x, test\_x = train\_test\_split(images, test\_size=test\_size, random\_state=42)

train\_y, test\_y = train\_test\_split(masks, test\_size=test\_size, random\_state=42)

return (train\_x, train\_y), (test\_x, test\_y)

def read\_image(path):

path = path.decode()

x = cv2.imread(path, cv2.IMREAD\_COLOR)

x = cv2.resize(x, (128, 128))

x = x/255.0

x = x.astype(np.float32)

return x

def read\_mask(path):

path = path.decode()

x = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

x = cv2.resize(x, (128, 128))

x = x/255.0

x = x.astype(np.float32)

x = np.expand\_dims(x, axis=-1)

return x

def tf\_parse(x, y):

def \_parse(x, y):

x = read\_image(x)

y = read\_mask(y)

return x, y

x, y = tf.numpy\_function(\_parse, [x, y], [tf.float32, tf.float32])

x.set\_shape([128, 128, 3])

y.set\_shape([128, 128, 1])

return x, y

def tf\_dataset(X, Y, batch):

dataset = tf.data.Dataset.from\_tensor\_slices((X, Y))

dataset = dataset.map(tf\_parse)

dataset = dataset.batch(batch)

dataset = dataset.prefetch(10)

return dataset

dataset\_path = "Dataset"

(train\_x, train\_y), (test\_x, test\_y) = load\_data(dataset\_path)

print(f"Train: {len(train\_x)} - {len(train\_y)}")

print(f"Test: {len(test\_x)} - {len(test\_y)}")

batch = 4

train\_dataset = tf\_dataset(train\_x, train\_y, batch)

valid\_dataset = tf\_dataset(test\_x, test\_y, batch)

def conv\_block(inputs, num\_filters):

x = Conv2D(num\_filters, 3, padding="same")(inputs)

x = BatchNormalization()(x)

x = Activation("relu")(x)

x = Conv2D(num\_filters, 3, padding="same")(x)

x = BatchNormalization()(x)

x = Activation("relu")(x)

return x

def encoder\_block(inputs, num\_filters):

x = conv\_block(inputs, num\_filters)

p = MaxPool2D((2, 2))(x)

return x, p

def decoder\_block(inputs, skip\_features, num\_filters):

x = Conv2DTranspose(num\_filters, (2, 2), strides=2, padding="same")(inputs)

x = Concatenate()([x, skip\_features])

x = conv\_block(x, num\_filters)

return x

def build\_unet(input\_shape):

inputs = Input(input\_shape)

""" Encoder """

s1, p1 = encoder\_block(inputs, 64)

s2, p2 = encoder\_block(p1, 128)

s3, p3 = encoder\_block(p2, 256)

s4, p4 = encoder\_block(p3, 512)

""" Bridge """

b1 = conv\_block(p4, 1024)

""" Decoder """

d1 = decoder\_block(b1, s4, 512)

d2 = decoder\_block(d1, s3, 256)

d3 = decoder\_block(d2, s2, 128)

d4 = decoder\_block(d3, s1, 64)

""" Outputs """

outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(d4)

""" Model """

model = Model(inputs, outputs)

return model

model = build\_unet((128, 128, 3))

model.compile(loss="binary\_crossentropy", optimizer=Adam(), metrics=['accuracy'])

model.summary()

model\_path = 'MALIGNANT.h5'

batch\_size = 32

num\_epoch = 40

model.fit(

train\_dataset.repeat(),

epochs=num\_epoch,

validation\_data=valid\_dataset.repeat(),

steps\_per\_epoch=len(train\_x)//batch\_size,

validation\_steps=len(test\_x)//batch\_size,

)

def graph():

#Plot training & validation accuracy values

plt.plot(model.history.history['accuracy'])

plt.plot(model.history.history['loss'])

plt.title('Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

graph()

model.save("MALIGNANT.h5")

**Module 2:**

import os

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from os import listdir

import glob

from PIL import Image

import cv2

from random import randrange

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping

from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, MaxPool2D, Conv2DTranspose, Concatenate, Input

from tensorflow.keras.models import Model

import warnings

warnings.filterwarnings('ignore')

ori\_images = 'Dataset/Train/Images'

mask\_images = 'Dataset/Train/Masks'

def load\_data(dataset\_path, split=0.2):

images = [f for f in glob.glob(ori\_images + "\*\*/\*.png", recursive=True)]

masks = [f for f in glob.glob(mask\_images + "\*\*/\*.png", recursive=True)]

test\_size = int(len(images) \* split)

train\_x, test\_x = train\_test\_split(images, test\_size=test\_size, random\_state=42)

train\_y, test\_y = train\_test\_split(masks, test\_size=test\_size, random\_state=42)

return (train\_x, train\_y), (test\_x, test\_y)

def read\_image(path):

path = path.decode()

x = cv2.imread(path, cv2.IMREAD\_COLOR)

x = cv2.resize(x, (128, 128))

x = x/255.0

x = x.astype(np.float32)

return x

def read\_mask(path):

path = path.decode()

x = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

x = cv2.resize(x, (128, 128))

x = x/255.0

x = x.astype(np.float32)

x = np.expand\_dims(x, axis=-1)

return x

def tf\_parse(x, y):

def \_parse(x, y):

x = read\_image(x)

y = read\_mask(y)

return x, y

x, y = tf.numpy\_function(\_parse, [x, y], [tf.float32, tf.float32])

x.set\_shape([128, 128, 3])

y.set\_shape([128, 128, 1])

return x, y

def tf\_dataset(X, Y, batch):

dataset = tf.data.Dataset.from\_tensor\_slices((X, Y))

dataset = dataset.map(tf\_parse)

dataset = dataset.batch(batch)

dataset = dataset.prefetch(10)

return dataset

dataset\_path = "Dataset"

(train\_x, train\_y), (test\_x, test\_y) = load\_data(dataset\_path)

print(f"Train: {len(train\_x)} - {len(train\_y)}")

print(f"Test: {len(test\_x)} - {len(test\_y)}")

batch = 4

train\_dataset = tf\_dataset(train\_x, train\_y, batch)

valid\_dataset = tf\_dataset(test\_x, test\_y, batch)

def conv\_block(inputs, num\_filters):

x = Conv2D(num\_filters, 3, padding="same")(inputs)

x = BatchNormalization()(x)

x = Activation("relu")(x)

x = Conv2D(num\_filters, 3, padding="same")(x)

x = BatchNormalization()(x)

x = Activation("relu")(x)

return x

def encoder\_block(inputs, num\_filters):

x = conv\_block(inputs, num\_filters)

p = MaxPool2D((2, 2))(x)

return x, p

def decoder\_block(inputs, skip\_features, num\_filters):

x = Conv2DTranspose(num\_filters, (2, 2), strides=2, padding="same")(inputs)

x = Concatenate()([x, skip\_features])

x = conv\_block(x, num\_filters)

return x

def build\_unet(input\_shape):

inputs = Input(input\_shape)

""" Encoder """

s1, p1 = encoder\_block(inputs, 64)

s2, p2 = encoder\_block(p1, 128)

s3, p3 = encoder\_block(p2, 256)

s4, p4 = encoder\_block(p3, 512)

""" Bridge """

b1 = conv\_block(p4, 1024)

""" Decoder """

d1 = decoder\_block(b1, s4, 512)

d2 = decoder\_block(d1, s3, 256)

d3 = decoder\_block(d2, s2, 128)

d4 = decoder\_block(d3, s1, 64)

""" Outputs """

outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(d4)

""" Model """

model = Model(inputs, outputs)

return model

model = build\_unet((128, 128, 3))

model.compile(loss="binary\_crossentropy", optimizer=Adam(), metrics=['accuracy'])

model.summary()

model\_path = 'BENIGN.h5'

batch\_size = 32

num\_epoch = 40

model.fit(

train\_dataset.repeat(),

epochs=num\_epoch,

validation\_data=valid\_dataset.repeat(),

steps\_per\_epoch=len(train\_x)//batch\_size,

validation\_steps=len(test\_x)//batch\_size,

)

def graph():

#Plot training & validation accuracy values

plt.plot(model.history.history['accuracy'])

plt.plot(model.history.history['loss'])

plt.title('Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

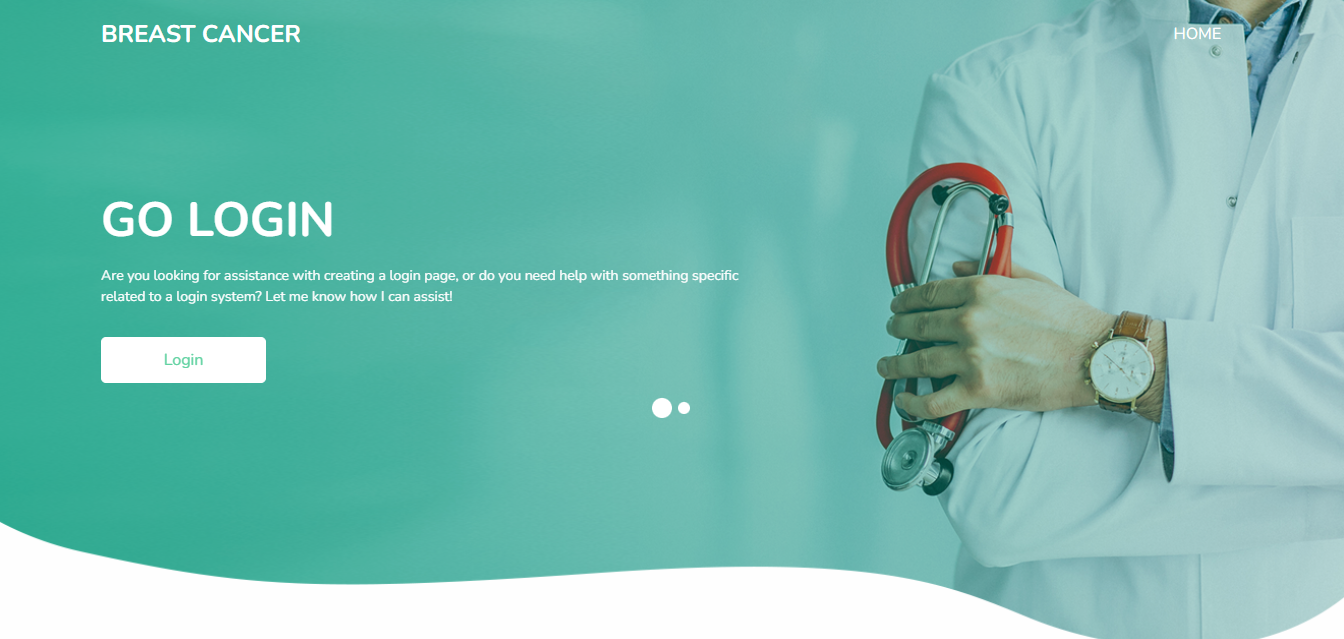
plt.legend(['Train', 'Test'], loc='upper left')

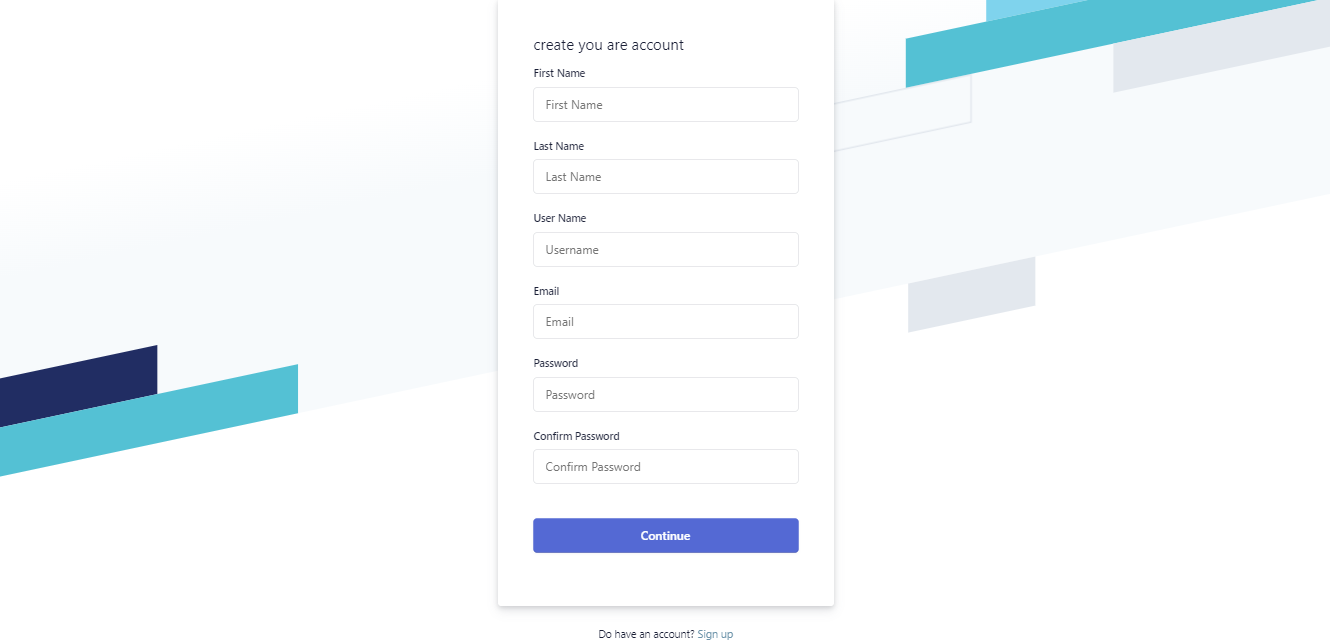
plt.show()

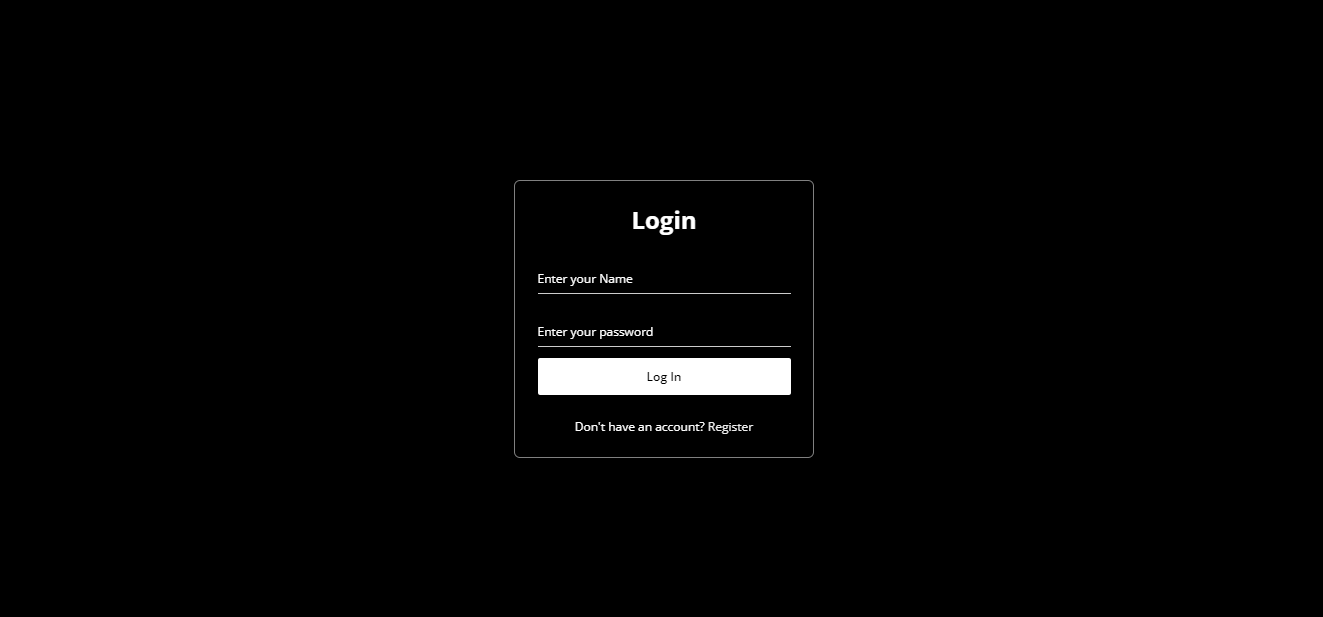
graph()

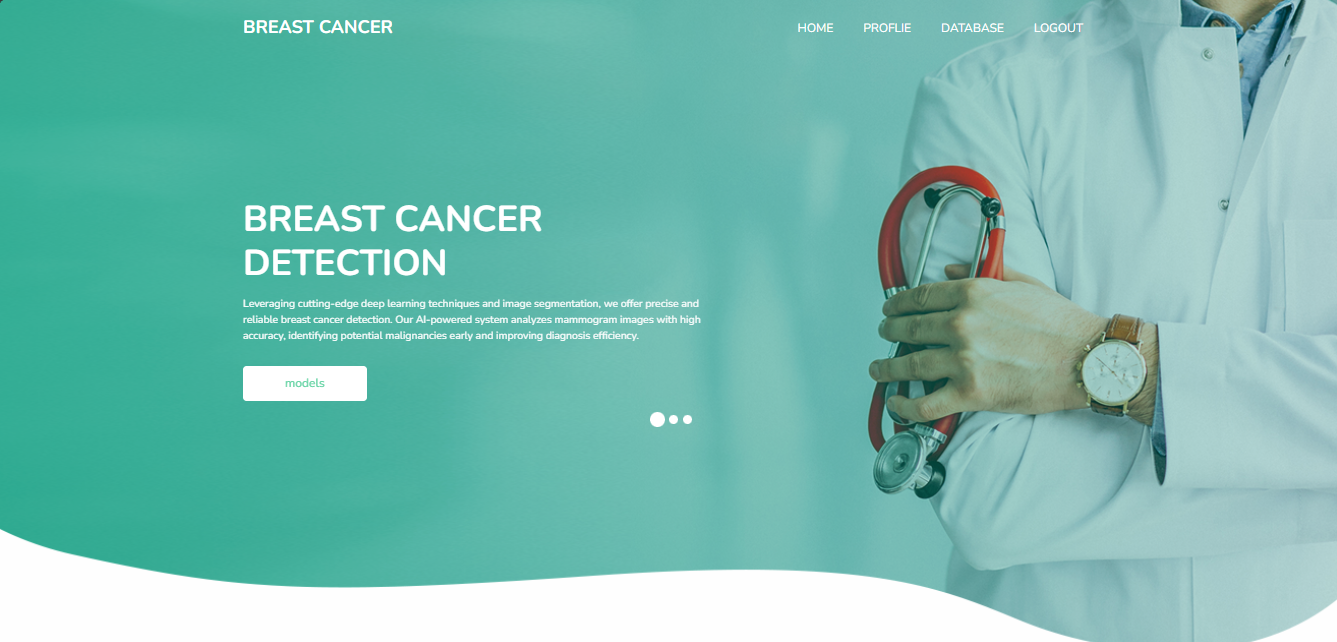
model.save("BENIGN.h5")

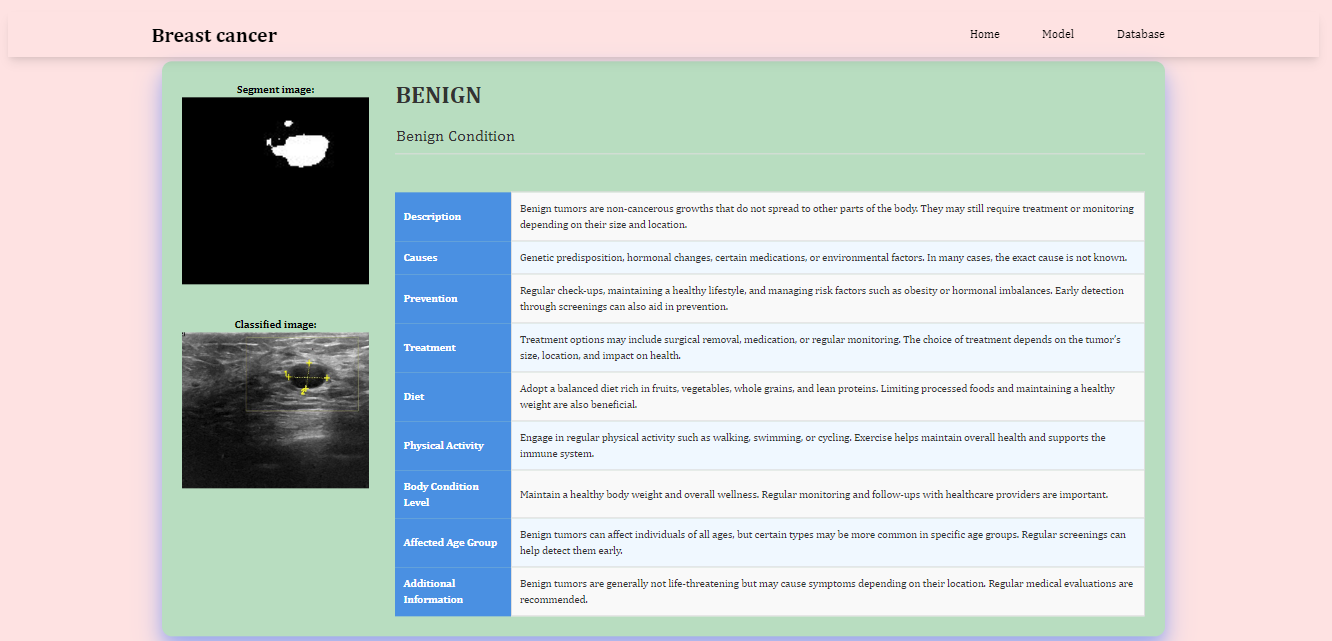
**Output Screenshot:**

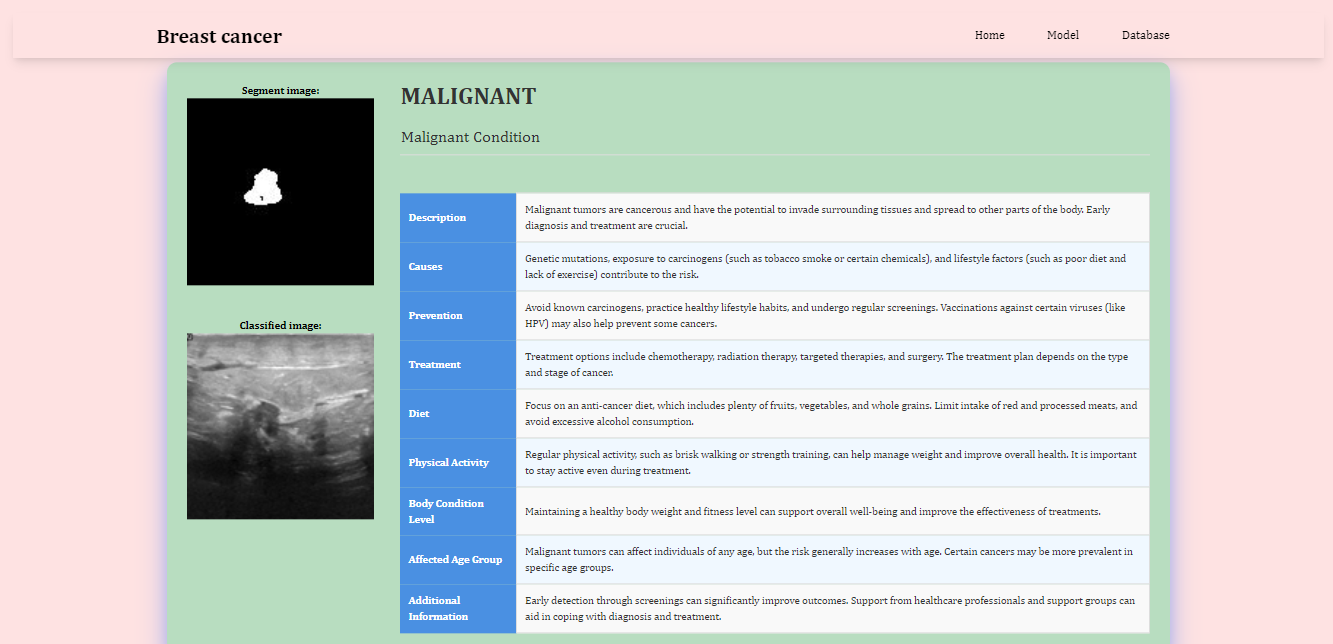
****

****

****



****

****

**31. Conclusion:**

In conclusion, the Breast Cancer Classification and Segmentation project utilizing artificial intelligence techniques represent a significant stride towards more accurate and efficient diagnostic methodologies. The integration of AI has demonstrated promising outcomes in discerning malignant from benign tumors, offering valuable support to healthcare practitioners. The potential for early detection and precise segmentation showcased in this project underscores the pivotal role AI can play in improving breast cancer diagnosis. As this field progresses, ongoing research should focus on refining algorithms, addressing interpretability challenges, and ensuring seamless integration into clinical workflows. Ultimately, the continued collaboration between data scientists, medical professionals, and technology experts holds the key to realizing the full potential of artificial intelligence in breast cancer diagnosis and contributing to advancements in healthcare.

32. **FUTURE WORK:**

Future work in breast cancer classification and segmentation should emphasize the incorporation of explainable AI techniques to enhance transparency and interpretability of model decisions, ensuring trust and acceptance in clinical settings.Exploring personalized medicine strategies by tailoring AI models to individual patient profiles and genetic variations can improve the accuracy of breast cancer diagnosis and treatment recommendations, paving the way for more targeted and effective healthcare interventions.