**UML Diagrams**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful insists the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**Goals:**

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

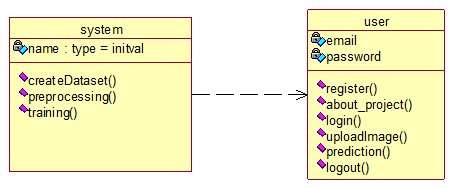
**Use Case Diagram:**

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



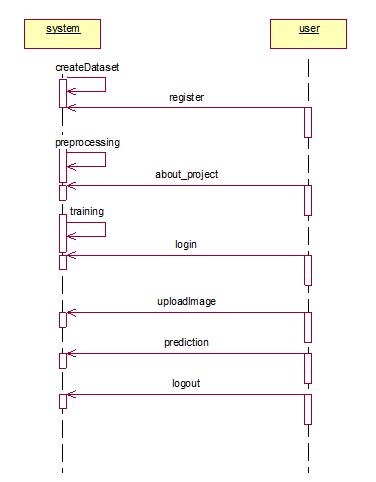
**Class Diagram:**

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.



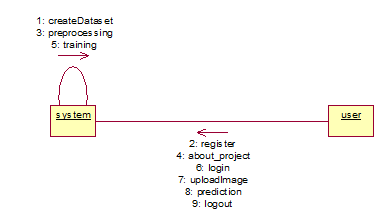
**Sequence Diagram:**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and time diagrams.



**Collaboration Diagram:**

In collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram. The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization whereas the collaboration diagram shows the object organization.



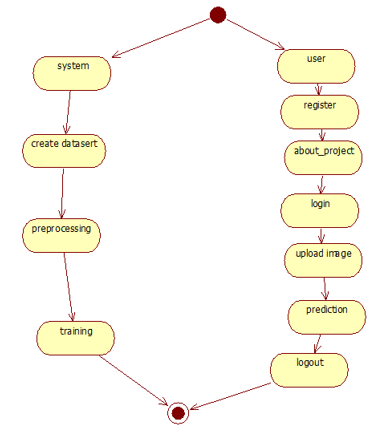
**Deployment diagram**

Deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware’s used to deploy the application.



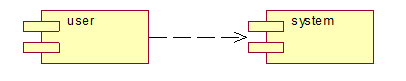
**Activity Diagram:**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



**Component diagram**:

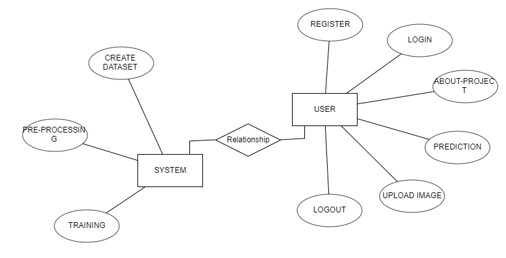
A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical **c**omponents in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.



**ER Diagram:**

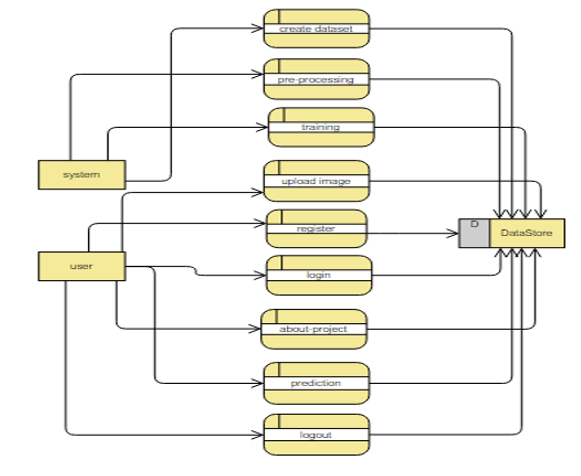
An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

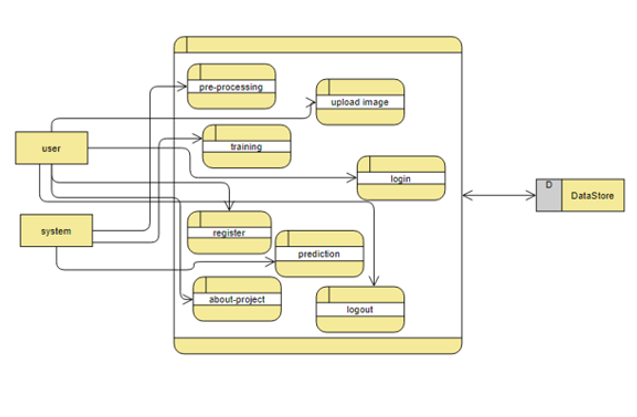
An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Let’s have a look at a simple ER diagram to understand this concept.

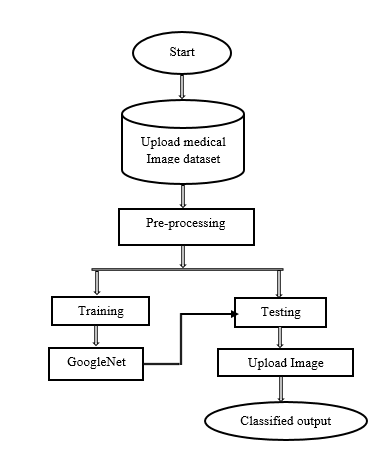


**DFD Diagram:**

A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or a combination of both. It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.







**Fig. Flow of Proposed method**

**Algorithm used in proposed method:**

**Convolution Neural Network (CNN)**

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analysing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on the shared-weight architecture of the convolution kernels that scan the hidden layers and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, Image segmentation, medical image analysis, natural language processing, brain-computer interfaces, and financial time series. The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolutional networks are a specialized type of neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

A convolutional neural network consists of an input layer, hidden layers and an output layer. In any feed-forward neural network, any middle layers are called hidden because their inputs and outputs are masked by the activation function and final convolution. In a convolutional neural network, the hidden layers include layers that perform convolutions. Typically this includes a layer that does multiplication or other dot product, and its activation function is commonly ReLU. This is followed by other convolution layers such as pooling layers, fully connected layers and normalization layers.

Convolutional layer

A convolutional layer within a neural network should have the following attributes:

• Convolution filters/kernels defined by a width and height (hyper-parameters).

• The number of input channels and output channels (hyper-parameter).

• The depth of the convolution kernel/filter (the input channels) must equal the number channels (depth) of the input feature map.

• The hyper parameters of the convolution operation, like padding size and stride.

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its receptive field.

**Pooling layers**

Convolutional networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. There are two common types of pooling: max and average. Max pooling uses the maximum value of each cluster of neurons at the prior layer, while average pooling instead uses the average value.

**Dense Layer**

The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models.

In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of back propagation. The output generated by the dense layer is an ‘m’ dimensional vector. Thus, dense layer is basically used for changing the dimensions of the vector. Dense layers also applies operations like rotation, scaling, translation on the vector.

**ReLU Layer**

ReLU is the abbreviation of rectified linear unit, which applies the non-saturating activation function. It effectively removes negative values from an activation map by setting them to zero. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Other functions are also used to increase nonlinearity, for example the saturating hyperbolic tangent and the sigmoid function. ReLU is often preferred to other functions because it trains the neural network several times faster without a significant penalty to generalization accuracy.

**Softmax**

The Softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the Softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities. If one of the inputs is small or negative, the Softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1.

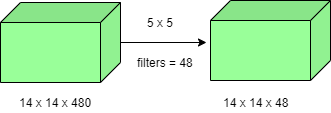
**GoogLeNet:**

Google Net (or Inception V1) was proposed by research at Google (with the collaboration of various universities) in 2014 in the research paper titled “Going Deeper with Convolutions”. This architecture was the winner at the ILSVRC 2014 image classification challenge. It has provided a significant decrease in error rate as compared to previous winners AlexNet (Winner of ILSVRC 2012) and ZF-Net (Winner of ILSVRC 2013) and significantly less error rate than VGG (2014 runner up). This architecture uses techniques such as 1×1 convolutions in the middle of the architecture and global average pooling.

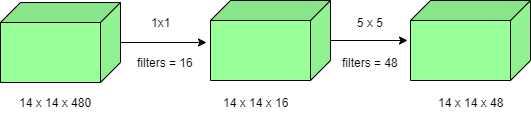
**Features of GoogleNet:**

The GoogLeNet architecture is very different from previous state-of-the-art architectures such as AlexNet and ZF-Net. It uses many different kinds of methods such as 1×1 convolution and global average pooling that enables it to create deeper architecture. In the architecture, we will discuss some of these methods:

* **1×1 convolution:** The inception architecture uses 1×1 convolution in its architecture. These convolutions used to decrease the number of parameters (weights and biases) of the architecture. By reducing the parameters we also increase the depth of the architecture. Let’s look at an example of a 1×1 convolution below:
  + For Example, If we want to perform 5×5 convolution having 48 filters without using 1×1 convolution as intermediate:



* Total Number of operations : (14 x 14 x 48) x (5 x 5 x 480) = 112.9 M
  + With 1×1 convolution :



* (14 x 14 x 16) x (1 x 1 x 480) + (14 x 14 x 48) x (5 x 5 x 16) = 1.5M + 3.8M = 5.3M which is much smaller than 112.9M.
* **Global Average Pooling:**

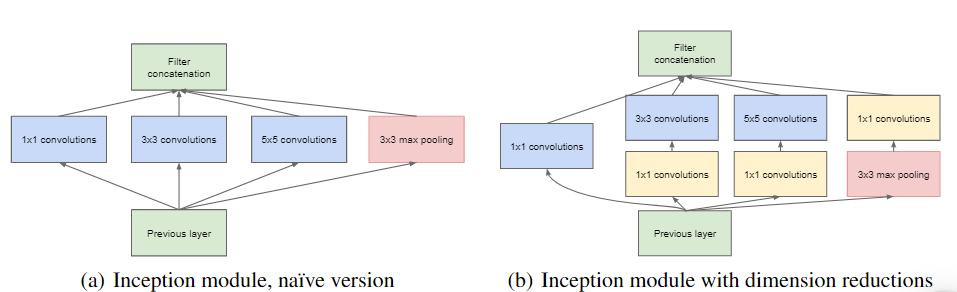
In the previous architecture such as AlexNet, the fully connected layers are used at the end of the network. These fully connected layers contain the majority of parameters of many architectures that causes an increase in computation cost.

In GoogLeNet architecture, there is a method called global average pooling is used at the end of the network. This layer takes a feature map of 7×7 and averages it to 1×1. This also decreases the number of trainable parameters to 0 and improves the top-1 accuracy by 0.6%

* **Inception Module:**

The inception module is different from previous architectures such as AlexNet, ZF-Net. In this architecture, there is a fixed convolution size for each layer.

In the Inception module 1×1, 3×3, 5×5 convolution and 3×3 max pooling performed in a parallel way at the input and the output of these are stacked together to generated final output. The idea behind that convolution filters of different sizes will handle objects at multiple scale better.

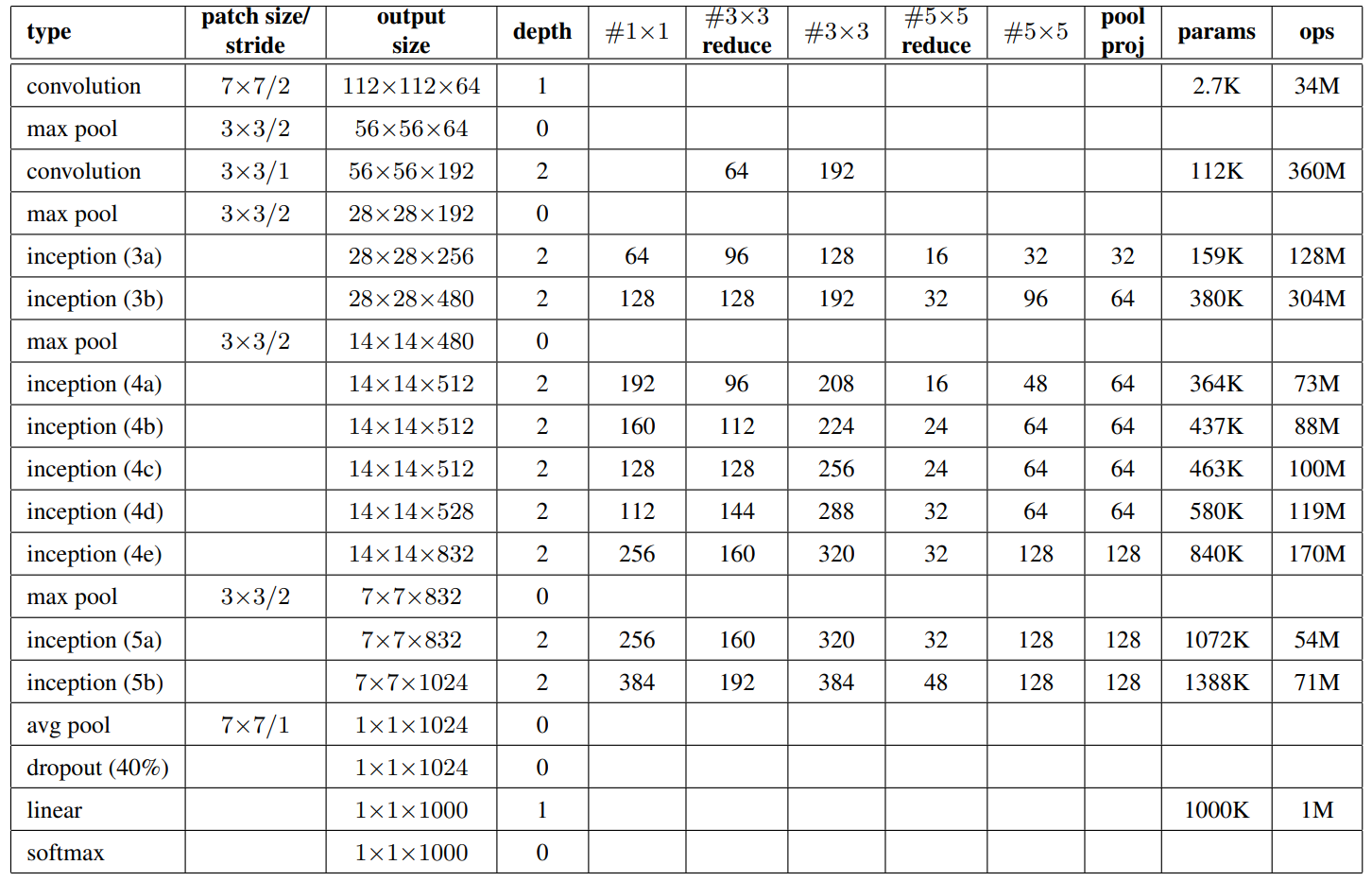


* **Auxiliary Classifier for Training:**

Inception architecture used some intermediate classifier branches in the middle of the architecture, these branches are used during training only. These branches consist of a 5×5 average pooling layer with a stride of 3, a 1×1 convolutions with 128 filters, two fully connected layers of 1024 outputs and 1000 outputs and a Softmax classification layer. The generated loss of these layers added to total loss with a weight of 0.3. These layers help in combating gradient vanishing problem and also provide regularization.

**Model Architecture:**

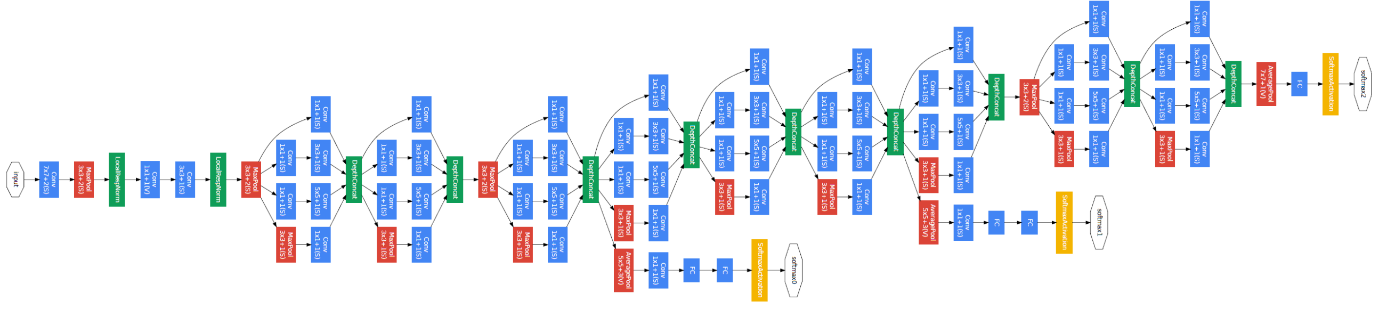
Below is Layer by Layer architectural details of GoogLeNet.



The overall architecture is 22 layers deep. The architecture was designed to keep computational efficiency in mind. The idea behind that the architecture can be run on individual devices even with low computational resources. The architecture also contains two auxiliary classifier layer connected to the output of Inception (4a) and Inception (4d) layers.

The architectural details of auxiliary classifiers as follows:

* An average pooling layer of filter size 5×5 and stride 3.
* A 1×1 convolution with 128 filters for dimension reduction and ReLU activation.
* A fully connected layer with 1025 outputs and ReLU activation
* Dropout Regularization with dropout ratio = 0.7
* A softmax classifier with 1000 classes output similar to the main softmax classsifier.



This architecture takes image of size 224 x 224 with RGB colour channels. All the convolutions inside this architecture uses Rectified Linear Units (ReLU) as their activation functions.