1. INTRODUCTION

1.1 MOTIVATION:

Many marketplaces offer tools that allow prospective customers to communicate with sellers to ask them questions, but as the landscape of online reviews shows, most customers simply don't, while those that do frequently encounter language, timezone, availability or other barriers that prevent them from getting answers to their questions. This raises the fascinating question of whether a combination of curated filtering and can come to the rescue to restore the human curation and assistance of brick-and-mortar stores to the online world? Nearly every major online store today uses some form of analytics to group products together based on viewership patterns and to construct elaborate profiles of their users that they use to recommend products based on the customer's purchasing, searching and viewership histories.

Imagine a major online store that used this data to cluster its users into coarse groups like "Audiophiles with strong technical expertise in audio design and electrical engineering" or "Music lovers with history of purchasing high-end equipment but have minimal technology comfort level." Using these coarse automated clusterings, stores could sort through their products and use a second tier of clustering algorithms to collect products that best fit into each category, perhaps in some cases hand curating especially common or high-demand options. This would ensure that the music lover with little technical expertise doesn't purchase a bare-wire solder-it-yourself programmable filtering device, while ensuring that the audiophile with advanced electrical engineering experience sees the item up front in searches.

The amount and variety of multimedia data such as images, movies and music available on over social networks are increasing rapidly. However, the ability to analyze and exploit these unorganized multimedia data remains inadequate, even with state-of-the-art media processing techniques. Our finding is that the emerging automatic curation of images enabled by scene classification and object detection can offer a promising solution. One remarkable virtue of automatic curation is that while posting reviews on a website, when a user is attempting to post images in order to give context to his review, image validation is not generally found. A topic chosen for review might be totally unrelated to the image being posted. This application can help us avoid such a scenario by helping us ensure that our image is validated with respect to a given description in order to ensure image relevance.

Putting this all together we see that in the mad rush to put all the world's products online, we might have lost the curation and expert advice that the human touch offered, but through

better automated curation, we could restore at least a basic level of guidance that would go a long ways towards reducing customer uncertainty online.

1.2 PROBLEM DEFINITION:

In websites that contain a large amount of information, often, it is not possible to skim through all the reading material to figure out what one needs. Even if there is a search option available, searching through text offers little to the user when a majority of the user population is better serviced with visual tools that allow prospective customers to find the content that they are looking for. Images can be the best form of click bait for users and images curated based on user preferences service users by making their search experience convenient and personalized.

We can therefore use an automatic curation of images based on description primarily in order to validate the contents of images against their respective descriptions. I intend to identify the images that conform to a certain scene's verbal definition. It means that upon being given a verbal description, relevant images can be returned barring images that do not match the description.

1.3 OBJECTIVE: The main aim of this project is to ensure that the following objectives are met:

- Reducing time delay in curating content of a user based on the customer's purchasing, searching and viewership histories.
- Centralized data handling

2. LITERATURE SURVEY

2.1 INTRODUCTION:

Curation is the act of finding, contextualizing and organizing information.. Curators tend to have a unique and consistent point of view–providing a reliable context for the content that they discover and organize.

To understand how fast curation is growing on the web, just take a look at Pinterest. The two-year-old visual clipping and publishing platform has now surpassed 10 million users, making it one of the fastest-growing web service on the web ever. Pinterest both creates tools to organize the noisy web and at the same time, it creates more instances of information in a different context. So it's both part of the problem, and a solution. The trick is finding the Pinterest pinboards that you like, and tune out the rest. Increasingly, image curators are emerging as a critical filter that helps consumers of niche content find "signal" in noise.

One thing we can be sure of is that the web is going to keep growing fast and the solution to make sense of the massive volume is a new engaged partnership between humans and machines. There are a number of companies building interesting curation tools. Among them, a few are Curata, CurationSoft, Scoop.it, Google+, Storify.com, PearlTrees.com, MySyndicaat.com, Curated.by, Storyful,Evri, Paper.li, Pearltrees, and Magnify.net.

2.2 EXISTING SYSTEM:

In the existing system, while there are several methods of curation of images, there aren't methods that use Caffe for Scene Classification in addition to YOLO for object detection to make searching efficient. By combining the predictive capabilities of both Caffe and YOLO, search labels can be automatically generated.

2.3 PROPOSED SYSTEM:

The proposed system consists of a Web Application intended for demonstration of the project, which can be used to train the scene classification model using the Caffe by using YOLO as a final step of verification in determining the object to scene relevance. Upon training more and more images with both these models and by storing the predictions obtained by running images on both Caffe and YOLO, the number of search labels can be significantly increased for each image thereby making image curation more accurate. The application of such a method could change how images appear in any website. Search can be made possible based

not on the image file's name but rather using the image's contents and therefore, image curation could enable a user to view content based on his/her past history of preferences thereby enabling the user to find exactly what he/she is looking for with an interface that changes as per his/her preferences and needs. The user may also be recommended similar images based on what he/she is already viewing. Image validations based on content of the image and its related description is one more application.

2.3.1 SCENE CLASSIFICATION USING Caffe

A key problem in multimedia data analysis is discovery of effective representations for sensory inputs—images, soundwaves. While performance of conventional, handcrafted features has plateaued in recent years, new developments in deep compositional architectures have kept performance levels rising [8]. Deep models have outperformed hand-engineered feature representations in many domains, and made learning possible in domains where engineered features were lacking entirely.

We are particularly motivated by large-scale visual recognition, where a specific type of deep architecture has achieved a commanding lead on the state-of-the-art. These Convolutional Neural Networks, or CNNs, are discriminatively trained via back-propagation through layers of convolutional filters and other operations such as rectification and pooling. Following the early success of digit classification in the 90's, these models have recently surpassed all known methods for large-scale visual recognition, and have been adopted by industry heavyweights such as Google, Facebook, and Baidu for image understanding and search.

Caffe is an open framework containing models and worked examples for deep learning. Some of Caffe's features include but are not limited to having:

- over 3,000 citations, more than 150 contributors and over 15,000 stars
- more than 9,000 forks and 1 pull request per day on an average
- pure C++ / CUDA library for deep learning
- command line, Python and MATLAB interfaces
- Fast, well-tested code
- Tools, reference models, demos, and recipes
- Seamless switch between CPU and GPU

Though its focus has been vision, Caffe's applications have been branched out to sequences, reinforcement learning, speech + text etc.

Fig 2.3.1.1 Prototype, Train, Deploy

Caffe is characterized by the following:

- Nets, Layers, and Blobs: the anatomy of a Caffe model.
- Forward / Backward: the essential computations of layered compositional models.
- Loss: the task to be learned is defined by the loss.
- Solver: the solver coordinates model optimization.
- Layer Catalogue: the layer is the fundamental unit of modeling and computation – Caffe's catalogue includes layers for state-of-the-art models.
- Interfaces: command line, Python, and MATLAB Caffe.

2.3.1.1 Blobs, Layers and Nets: Anatomy of a Caffe Model

Deep networks are compositional models that are naturally represented as a collection of inter-connected layers that work on chunks of data. Caffe defines a net layer-by-layer in its own model schema. The network defines the entire model bottom-to-top from input data to loss. As data and derivatives flow through the network in the forward and backward passes, Caffe stores, communicates, and manipulates the information as blobs:

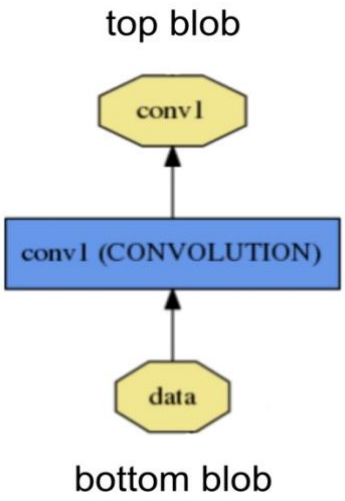


Fig 2.3.1.1.1 Layers

The **Blob** is the standard array and unified memory interface for the framework. A Blob is a wrapper over the actual data being processed and passed along by Caffe, and also under the hood provides synchronization capability between the CPU and the GPU. Mathematically, a blob is an N-dimensional array stored in a C-contiguous fashion.

The **Layer** comes next as the foundation of both model and computation. A layer takes input through bottom connections and makes output through top connections. Layers have two key responsibilities for the operation of the network as a whole: a forward pass that takes the inputs and produces the outputs, and a backward pass that takes the gradient with respect

to the output, and computes the gradients with respect to the parameters and to the inputs, which are in turn back-propagated to earlier layers. These passes are simply the composition of each layer's forward and backward.

The **Net** follows as the collection and connection of layers. The details of blob describe how information is stored and communicated in and across layers and nets. The net is defined as a set of layers and their connections in a plaintext modeling language. The net is a set of layers connected in a computation graph – a directed acyclic graph (DAG) to be exact.

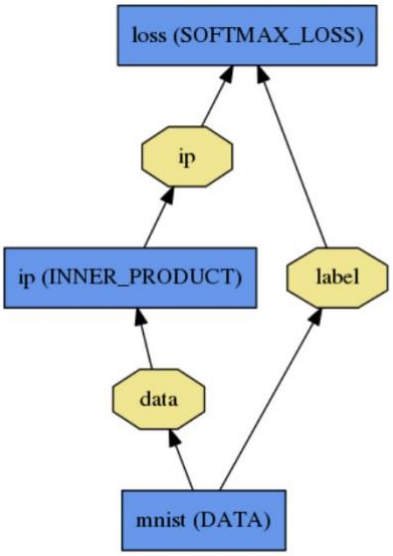


Fig 2.3.1.1.1 Network

2.3.1.2 Forward and Backward Pass

The forward pass computes the output given the input for inference. In forward Caffe composes the computation of each layer to compute the "function" represented by the model. This pass goes from bottom to top.

The backward pass computes the gradient given the loss for learning. In backward Caffe reverse-composes the gradient of each layer to compute the gradient of the whole model by automatic differentiation. This is back-propagation. This pass goes from top to bottom.

2.3.1.3 Loss

In Caffe, as in most of machine learning, learning is driven by a loss function (also known as an error, cost, or objective function). A loss function specifies the goal of learning by mapping parameter settings (i.e., the current network weights) to a scalar value specifying the "badness" of these parameter settings. Hence, the goal of learning is to find a setting of the weights that minimizes the loss function.

The loss in Caffe is computed by the Forward pass of the network. Each layer takes a set of input (bottom) blobs and produces a set of output (top) blobs. Some of these layers' outputs may be used in the loss function. A typical choice of loss function for one-versus-all classification tasks is the SoftmaxWithLoss function, used in a network definition as follows, for example:

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "pred"
```

bottom: "label"

top: "loss"

}

The final loss in Caffe, then, is computed by summing the total weighted loss over the network.

2.3.1.4 Solver

The solver orchestrates model optimization by coordinating the network's forward inference and backward gradients to form parameter updates that attempt to improve the loss. The responsibilities of learning are divided between the Solver for overseeing the optimization and generating parameter updates and the Net for yielding loss and gradients.

The solver

- scaffolds the optimization bookkeeping and creates the training network for learning and test network(s) for evaluation.
- iteratively optimizes by calling forward / backward and updating parameters
- (periodically) evaluates the test networks
- snapshots the model and solver state throughout the optimization

where each iteration

- calls network forward to compute the output and loss
- calls network backward to compute the gradients
- incorporates the gradients into parameter updates according to the solver method
- updates the solver state according to learning rate, history, and method
- to take the weights all the way from initialization to learned model.

2.3.1.5 Layer Catalogue

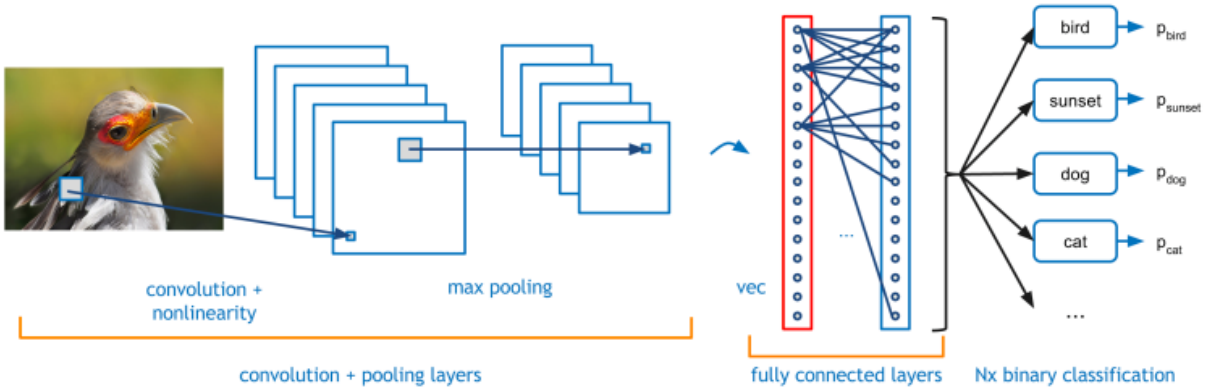


Fig 2.3.1.5.1 Layers in Caffe's neural network

Data Layers

- Image Data - read raw images.
- Database - read data from LEVELDB or LMDB.

Vision Layers

- Convolution Layer - convolves the input image with a set of learnable filters, each producing one feature map in the output image.
- Pooling Layer - max, average, or stochastic pooling.

Recurrent

- RNN
- Long-Short Term Memory (LSTM)

Common

- Inner Product - fully connected layer.
- Dropout
- Embed - for learning embeddings of one-hot encoded vector (takes index as input).

Normalization Layers

- Local Response Normalization (LRN) - performs a kind of "lateral inhibition" by normalizing over local input regions.
- Mean Variance Normalization (MVN) - performs contrast normalization / instance normalization.
- Batch Normalization - performs normalization over mini-batches.
- The bias and scale layers can be helpful in combination with normalization.

Activation

- ReLU / Rectified-Linear and Leaky-ReLU - ReLU and Leaky-ReLU rectification.
- PReLU - parametric ReLU.
- ELU - exponential linear rectification.
- Sigmoid
- TanH

Loss

- Multinomial Logistic Loss
- Infogain Loss - a generalization of MultinomialLogisticLossLayer.
- Softmax with Loss - computes the multinomial logistic loss of the softmax of its inputs. It's conceptually identical to a softmax layer followed by a multinomial logistic loss layer, but provides a more numerically stable gradient.

- Sum-of-Squares / Euclidean - computes the sum of squares of differences of its two inputs.
- Hinge / Margin - The hinge loss layer computes a one-vs-all hinge (L1) or squared hinge loss (L2).
- Sigmoid Cross-Entropy Loss - computes the cross-entropy (logistic) loss, often used for predicting targets interpreted as probabilities.
- Accuracy / Top-k layer - scores the output as an accuracy with respect to target – it is not actually a loss and has no backward step.

2.3.1.6 Interfaces

Caffe has command line, Python, and MATLAB interfaces for day-to-day usage, interfacing with research code, and rapid prototyping. While Caffe is a C++ library at heart and it exposes a modular interface for development, not every occasion calls for custom compilation. The cmdcaffe, pycaffe, and matcaffe interfaces are here for you.

2.3.1.7 Data: Inputs and Outputs

Data flows through Caffe as Blobs. Data layers load input and save output by converting to and from Blob to other formats. Common transformations like mean-subtraction and feature-scaling are done by data layer configuration. New input types are supported by developing a new data layer – the rest of the Net follows by the modularity of the Caffe layer catalogue.

Tops and Bottoms: A data layer makes top blobs to output data to the model. It does not have bottom blobs since it takes no input.

Data and Label: a data layer has at least one top canonically named data. For ground truth a second top can be defined that is canonically named label. Both tops simply produce blobs and there is nothing inherently special about these names. The (data, label) pairing is a convenience for classification models.

Transformations: data preprocessing is parametrized by transformation messages within the data layer definition.

Prefetching: for throughput data layers fetch the next batch of data and prepare it in the background while the Net computes the current batch.

Multiple Inputs: a Net can have multiple inputs of any number and type. Define as many data layers as needed giving each a unique name and top. Multiple inputs are useful for non-trivial ground truth: one data layer loads the actual data and the other data layer loads the ground truth in lock-step. In this arrangement both data and label can be any 4D array.

2.3.2 OBJECT DETECTION USING YOLO

Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections.

YOLO uses a totally different approach as a single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

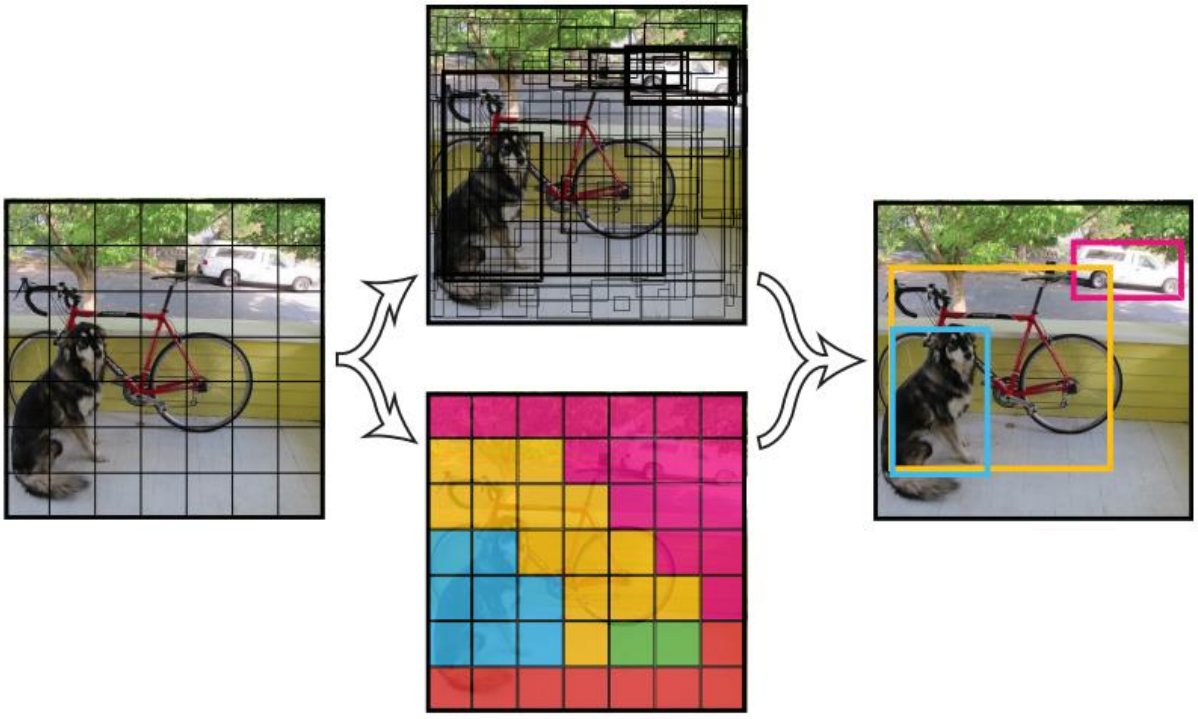


Fig 2.3.2.1 The Model

- Divide the image into a SxS grid.
- If the center of an object fall into a grid cell, it will be the responsible for the object.
- Each grid cell predict:
  - B bounding boxes;
  - B confidence scores as C=Pr(Obj)*IOU.
  - C cond. Class prob. as P=Pr(***Classi***|Object);
- Confidence Prediction is obtained as IOU of predicted box and any ground truth box.
- We obtain the class-specific confidence score as:

  Pr(***Classi***|Object)*Pr(Object)*IOU = Pr(***Classi***)*IOU

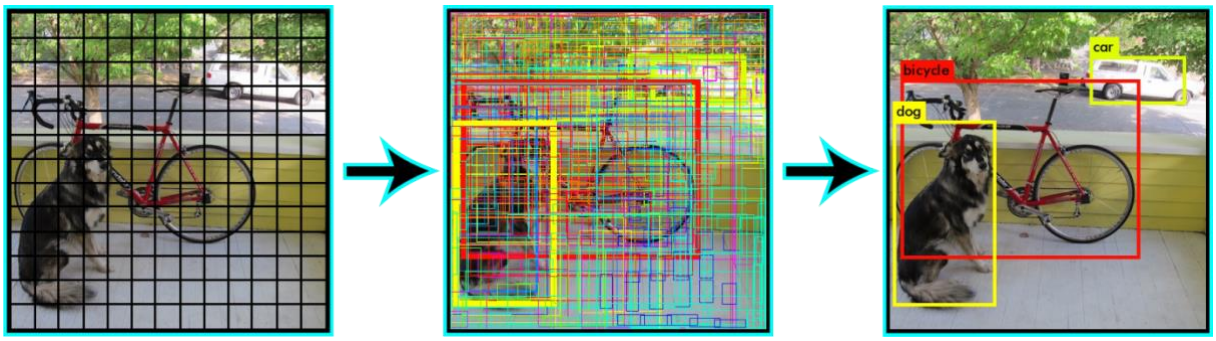


Fig 2.3.2.2 Making predictions

YOLO reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Therefore, you only look once (YOLO) at an image to predict what objects are present and where they are. YOLO is refreshingly simple: see Figure 1.

First, YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems.

Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method [14], mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.

Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected input.

All of our training and testing code is open source and available online at [removed for review]. A variety of pre-trained models are also available to download.

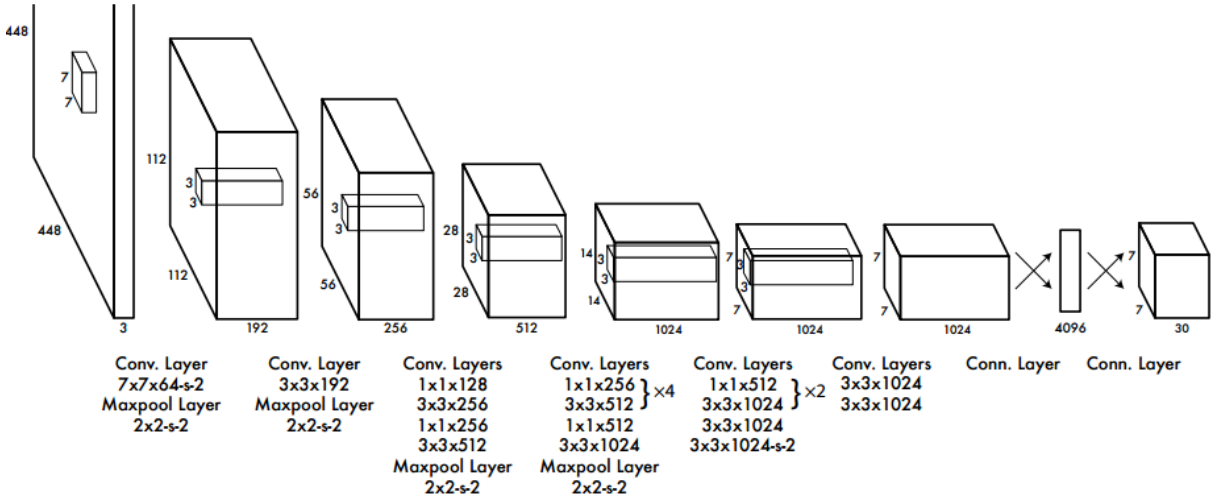


Fig 2.3.2.3 YOLO Architecture.

YOLO's detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 × 1 convolutional layers reduce the features space from preceding layers, it pre-trains the convolutional layers on the ImageNet classification task at half the resolution (224 × 224 input image) and then double the resolution for detection.

2.4 CONCLUSION:

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project. During system analysis, the feasibility study of the proposed system was carried out. This was to ensure that the proposed system makes automatic image curation a convenience for the users as well as the developers who would like to blend the software with their existing software applications. For feasibility analysis, some understanding of the major requirements of the system was essential.

Three key considerations were involved in the feasibility analysis:

Economic Feasibility: It is economically feasible especially for the users facing search inaccuracies as by curating images based on the user's search history and preferential search items, the user can now be directed towards the items that he is most likely interested in looking at. For example, if a user has stayed in Hotel Hyatt in Hyderabad and Delhi and he is now travelling to Dubai, he can be shown Hyatt hotels in Dubai for he is more likely to choose Hyatt if his history states that he has chosen Hyatt more than once before.

Technical Feasibility: The project is technically feasible as the software packages that are being used to build the application are capable of fulfilling the functional requirements specification of the project.

3. METHODOLOGY

3.1 SYSTEM REQUIREMENT SPECIFICATION

3.1.1 Minimum Hardware Requirements:-

1. **Processor :** Intel
2. **RAM :** 6 GB or above
3. **GPU:** NVIDIA GeForce GTX 980 or above (Optional)
   Better GPU's ensure faster training and even faster prediction results
4. **Hard Disk :** 100 GB atleast

3.1.2 Minimum Software Requirements:-

1. **Computer's Operating System :** Any Linux based Operating System
2. **Python**
3. **Tornado Server:** Tornado is a Python web framework and asynchronous networking library, originally developed at FriendFeed. By using non-blocking network I/O, Tornado can scale to tens of thousands of open connections, making it ideal for long polling, WebSockets, and other applications that require a long-lived connection to each user.
4. **Flask:** Flask is a microframework for Python based on Werkzeug and Jinja 2. The "micro" in microframework means Flask aims to keep the core simple but extensible. Flask supports extensions to add such functionality to your application as if it was implemented in Flask itself. Numerous extensions provide database integration, form validation, upload handling, various open authentication technologies, and more. Flask may be "micro", but it's ready for production use on a variety of needs.
5. **Caffe:** Caffe provides multimedia scientists and practitioners with a clean and modifiable framework for state-of-the-art deep learning algorithms and a collection of reference models. The framework is a BSD-licensed C++ library with Python and MATLAB bindings for training and deploying generalpurpose convolutional neural networks and other deep models efficiently on commodity architectures. Caffe fits industry and internet-scale media needs by CUDA GPU computation, processing over 40 million images a day on a single K40 or Titan GPU (≈ 2.5 ms per image). By separating model representation from actual implementation, Caffe allows

experimentation and seamless switching among platforms for ease of development and deployment from prototyping machines to cloud environments.

6. **Database:** **PostgreSQL** is a powerful, open source object-relational **database** system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness.

3.2 PRODUCT REQUIREMENT SPECIFICATIONS

3.2.1 Functional Requirements:

A **functional requirement** defines a function of a system or its component. A function is described as a set of inputs, the behavior, and outputs. Functional requirements drive the *application architecture* of a system, while non-functional requirements drive the *technical architecture* of a system. The hierarchy of functional requirements is: user/stakeholder request → feature → use case → business rule. The system consists of the following functional requirements.

User Interface:

- **Application System:** The application's functionality is divided into Health Status and Diagnostics respectively.

3.2.2 Non Functional Requirements:

Maintainability: The user will be able to reset all options and all stored user variables to default settings.

Reliability: Some of the attributes identified for the reliability are listed below:

- Diagnostics information is the current configuration of the terminal and new updated information is received each time the Diagnostics option is inflated.
- Health Status, as per requirement only requires periodic updates to the database and therefore the system is reliable in fulfilling the requirement of posting data to the server every few minutes and this data is posted to the database by the server. The database is accessible by the Android App.
- All data storage for user variables will be committed to the database at the time of entry.

3.3 INTRODUCTION TO DESIGN

3.3.1 SDLC Methodology

The Software Development Life Cycle for the project is represented by the Spiral Model. The Spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. It provides the potential for rapid development of incremental versions of the software. Using the spiral model, software is developed in a series of incremental releases. A spiral model is divided into a number of framework activities known as "Task regions", defined by the software engineering team. In Fig 4.1, the spiral model shows 4 Task regions, a combination of which produces intermediate prototypes.

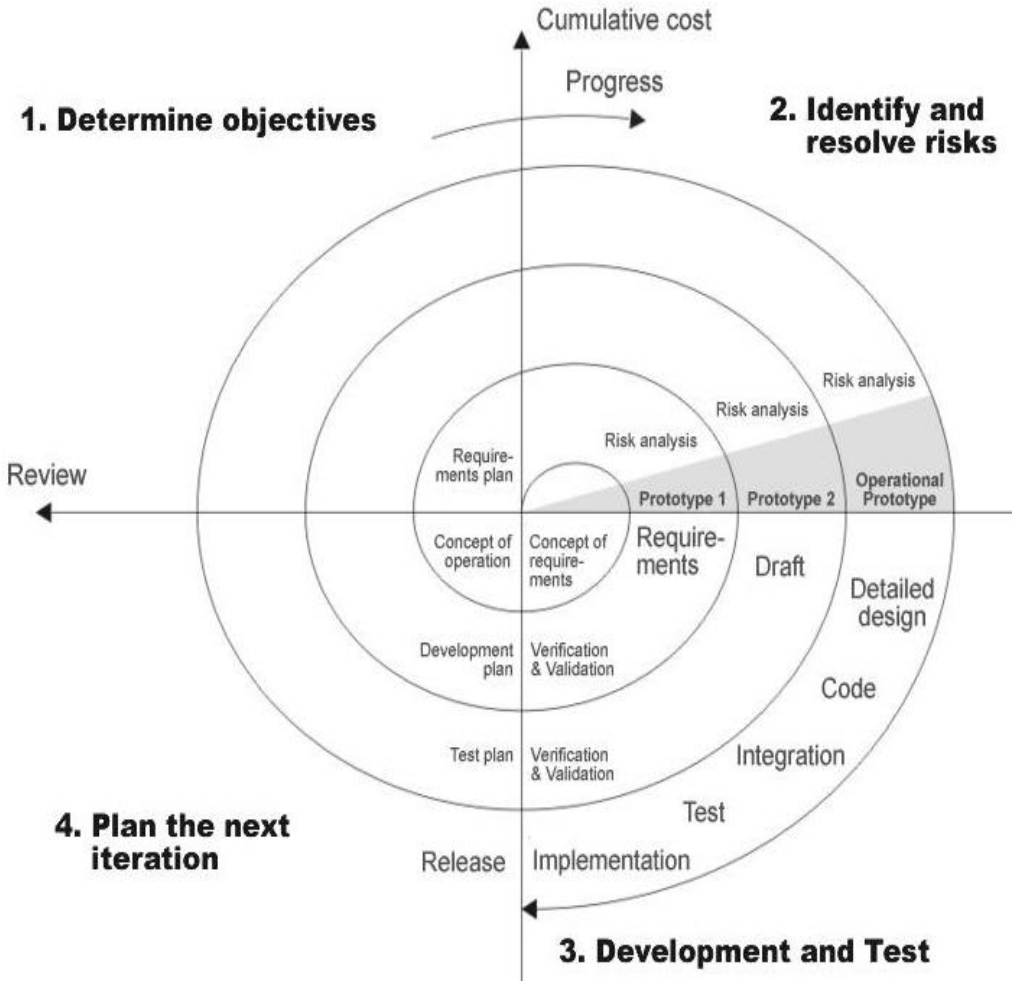


Fig 3.3.1.1 Spiral Model

3.4 UML Diagrams

Design is a meaningful engineering representation of something that is to be built. Software design is a process through which the requirements are translated into a representation of the software. Design is the place where quality is fostered in software engineering. Design is the perfect way to accurately translate a customer's requirement 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. This chapter discusses about the design part of the project. Here in this document the various UML diagrams that are used for the implementation of the project are discussed.

Design Principle

The Unified Modelling Language (UML) is a visual modelling language used to specify, visualize, construct and document a software intensive system. The embedded real-time software systems encountered in applications such as telecommunications, school systems, aerospace, and defence typically tends to be large and extremely complex. It is crucial in such systems that the software is designed with a sound architecture. A good architecture not only simplifies construction of the initial system, but also, readily accommodates changes forced by a steady stream of new requirements.

The UML represents a collection of best engineering practices that have proven successful in the modelling 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. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

The primary goals in the design of the UML are: Provide users with a ready-to-use, expressive visual modelling language so they can develop and exchange meaningful models. Provide extensibility and specialization mechanisms to extend the core concepts. Be independent of particular programming languages and development processes. Provide a formal basis for understanding the modelling language. Encourage the growth of the OO tools market.

3.4.1 Use Case Diagrams:

Use Case: Use cases focuses on a specific scenario, and describes the steps that are necessary to bring it to a successful completion. Each step in a use case represents an interaction with people or entities outside the computer system and the system itself. A use case thus lists a sequence of actions that yields a result that is of value to an actor. An essential aspect of the use case is that it must describe a scenario that completes to a point that is of some value to one of the actors.

Use case describes the behaviour of a system. It is used to structure things in a model. It contains multiple scenarios, each of which describes a sequence of actions that is clear enough for outsiders to understand. A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. While a use case itself might drill into a lot of detail about every possibility, a use-case diagram can help provide a higher-level view of the system. It has been said before that "Use case diagrams are the blueprints for your system". They provide the simplified and graphical representation of what the system must actually do. Due to their simplistic nature, use case diagrams can be a good communication tool for stakeholders. The drawings attempt to mimic the real world and provide a view for the stakeholder to understand how the system is going to be designed.

Actor: An actor represents a coherent set of roles that users of a system play when interacting with the use cases of the system. An actor participates in use cases to accomplish an overall purpose. An actor can represent the role of a human, a device, or any other systems.

Associations. Associations between actors and use cases are indicated in use case diagrams by solid lines. An association exists whenever an actor is involved with an interaction described by a use case. Associations are modelled as lines connecting use cases and actors to one another, with an optional arrowhead on one end of the line. The arrowhead is often used to indicating the direction of the initial invocation of the relationship or to indicate the primary actor within the use case. The arrowheads are typically confused with data flow and as a result I avoid their use.

Use Case Diagram for storing image predictions in the database

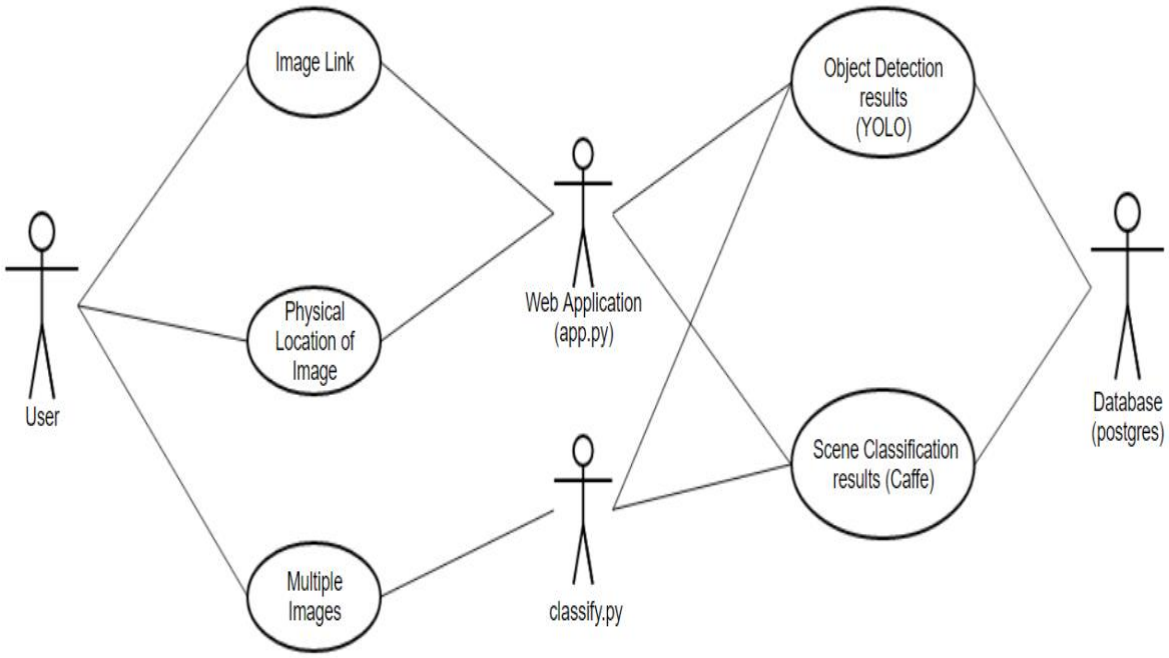


Fig 3.4.1.1 Use Case Diagram for storing image predictions in the database

In the Use Case described in the Fig 4.2.2.1, explains how a user stores object detection results and scene classification results in the database.

User can give input of the image in three ways:

- **Image Link:** User can give input of the image in the form of URL.
- **Physical Location:** User can give input of the image in the form of directory path.
- **Multiple Images:** User can give multiple images as an input by specifying folder name to classify.py.

Web application (app.py) takes the input and gives it to YOLO model and Caffe model to generate object detection results and scene classification results respectively. Once we get the predictions of the image, we store them in database along with path/URL of the image. Path/URL is used to retrieve images from the database at later point.

Classify.py takes multiple images from the folder and stores the predictions of the images without any user interface. It will help us to store predictions of many images in less time

Use Case diagram for retrieving images from the database based on description

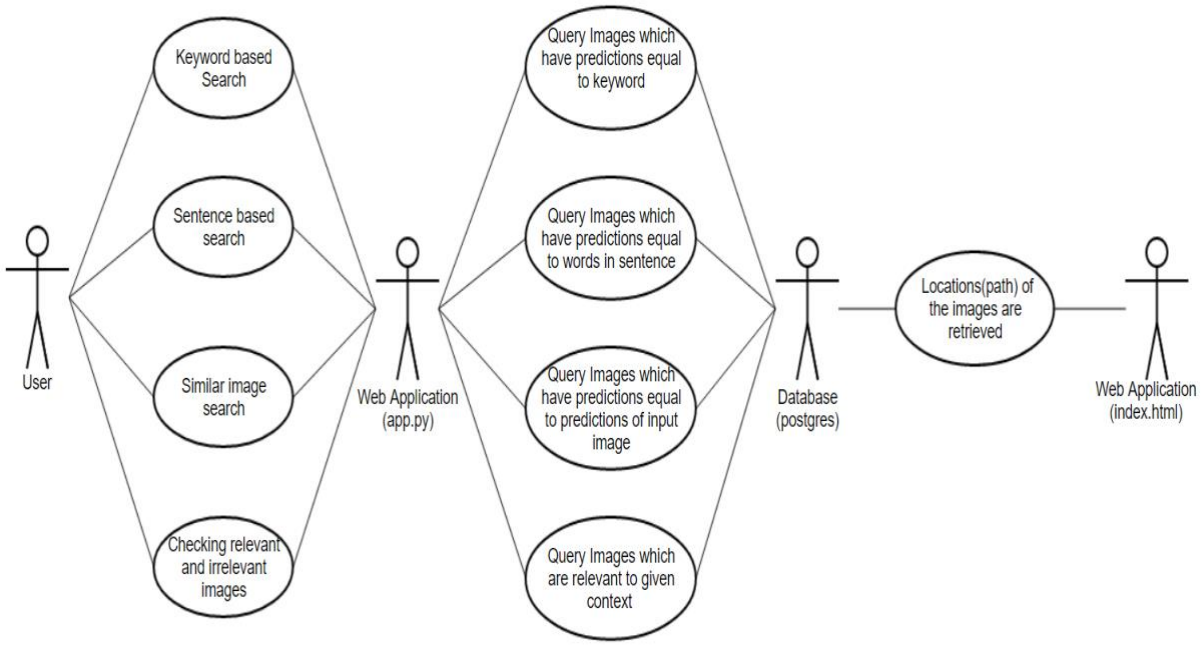


Fig: 3.4.1.2 Use Case diagram for retrieving images from the database based on description

In the Use Case described in the Fig 4.2.2.2, explains different ways to search for Images in the database.

They are three ways to search for the images in the database:

- **Keyword based search:** User will search for the images by specifying a single keyword. Based on the keyword, python application will search for the predictions of the images in the database. Set of images are retrieved from the database whose predictions matches with the keyword.
- **Sentence based search:** User will search for the images by giving a sentence with group of words. Python application will divide the sentence in to set of meaningful words. Set of images are retrieved from the database whose predictions matches with these words.
- **Similar Image Search:** User will input the image for searching set of similar images. Python application matches the predictions of the input images with the predictions in the database and retrieves similar images.
- **Check relevant or irrelevant images:** Python application will check whether an image is relevant or not based on the specified context.

3.4.2 Class Diagram

A Class diagram shows classes and the relationships among them. The Class diagram represents the relationship between classes throughout the lifetime of the system. A class is a template for objects. An object of a class should have a State, Behaviour and Identity. A class definition will therefore include the operations that are allowed on the objects of the class and the possible states for the objects of the class.

Classes are drawn as boxes, which contain the class name and, when appropriate, the names of attributes and methods in additional compartments.

After all the requirements have been gathered, the next step was to get a conceptual picture of the objects. The diagram shows the most important objects that had to be modelled in the system so that the system can carry out the tasks specified by the requirements.

Class diagrams depict:

Class: A Class is a description for a set of objects that shares the same attributes, and has similar operations, relationships, behaviours and semantics.

Generalization: Generalization is a relationship between a general element and a more specific kind of that element. It means that the more specific element can be used whenever the general element appears. This relation is also known as specialization or inheritance link.

Realization: Realization is the relationship between a specialization and its implementation. It is an indication of the inheritance of behaviour without the inheritance of structure.

Association: Association is represented by drawing a line between classes. Associations represent structural relationships between classes and can be named to facilitate model understanding. If two classes are associated, you can navigate from an object of one class to an object of the class.

Aggregation: Aggregation is a special kind of association in which one class represents as the larger class that consists of a smaller class. It has the meaning of “has-a” relationship.

Class Diagram to store predictions of the image in the database

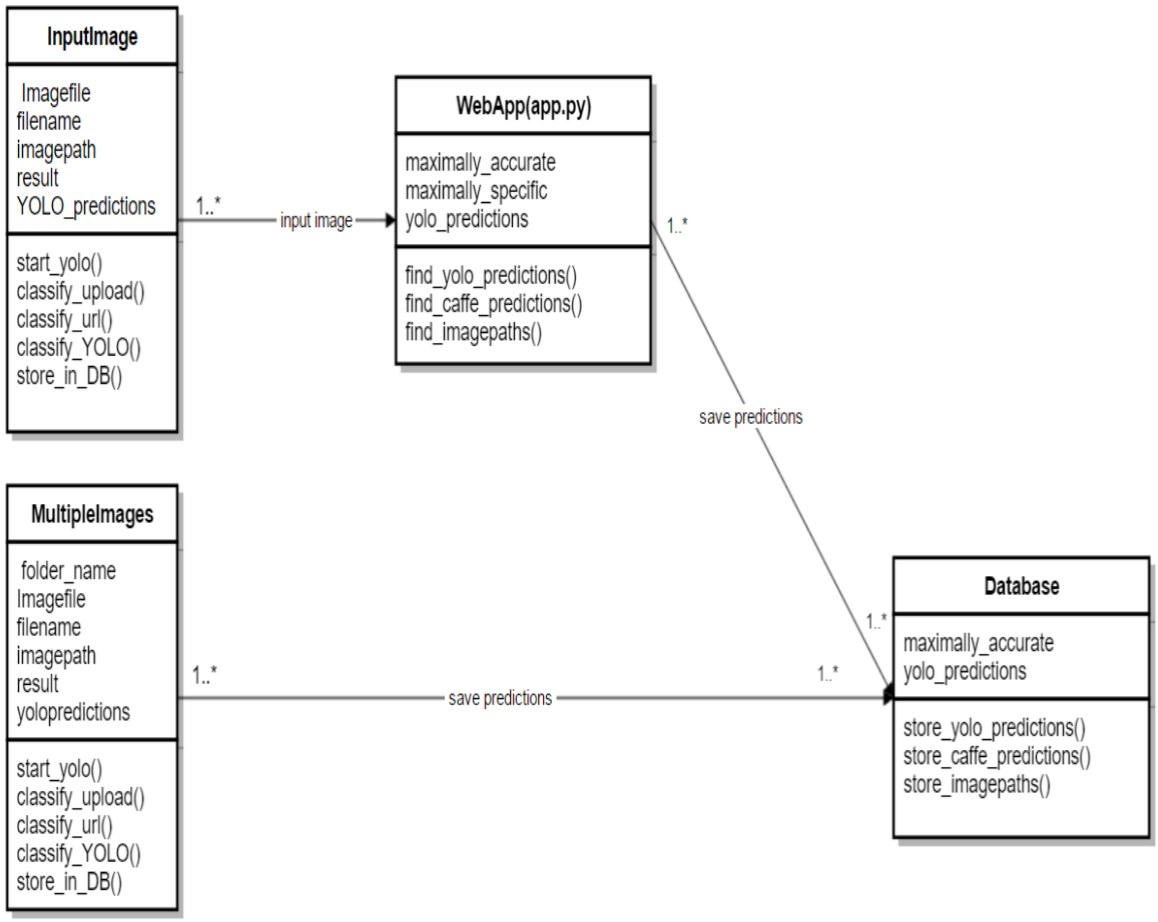


Fig 3.4.2.1 Class Diagram to store predictions of the image in the database

InputImage class and MultipleImages class

- **Imagefile:** Image whose predictions are to be stored is specified in Imagefile.
- **Imagepath:** Path of the image whose predictions are to be stored is specified in imagepath.
- **Result:** Caffe Predictions are stored in this variable.
- **YOLO_predictions:** YOLO predictions are stored.
- **Folder_name:** It will have folder name which contains multiple images.
- **Start_yolo ():** This method will process the image and gives the YOLO predictions.
- **Classify_upload ():** This method takes the physical location of the image as an input and classify the image to get scene classification results.
- **Classify_url ():** This method takes the URL of the image as an input and classify the image to get scene classification results.
- **Store_in_DB ():** Store all these results in the database.

WebApp Class:

- **Maximally accurate:** This will have top five maximally accurate results.
- **Maximally specific:** This will have top five maximally specific results.
- **Yolo_predictions:** This will have top five object detections results.
- **Find_yolo_predictions ():** It will find the object detection results of an image.
- **Find_caffe_predictions ():** It will find scene classification results of an image.
- **Find_imagepaths ():** It will find the path of the image to store them in the databases.

Database Class:

- **Maximally accurate:** This will have top five maximally accurate results.
- **Maximally specific:** This will have top five maximally specific results.
- **Store_yolo_predictions ():** It will store YOLO predictions in the database.
- **Store_caffe_predictions ():** It will store Caffe predictions in the database.
- **Store_imagepaths ():** It will store the paths of the images along with their predictions.

Class Diagram to display the images based on search

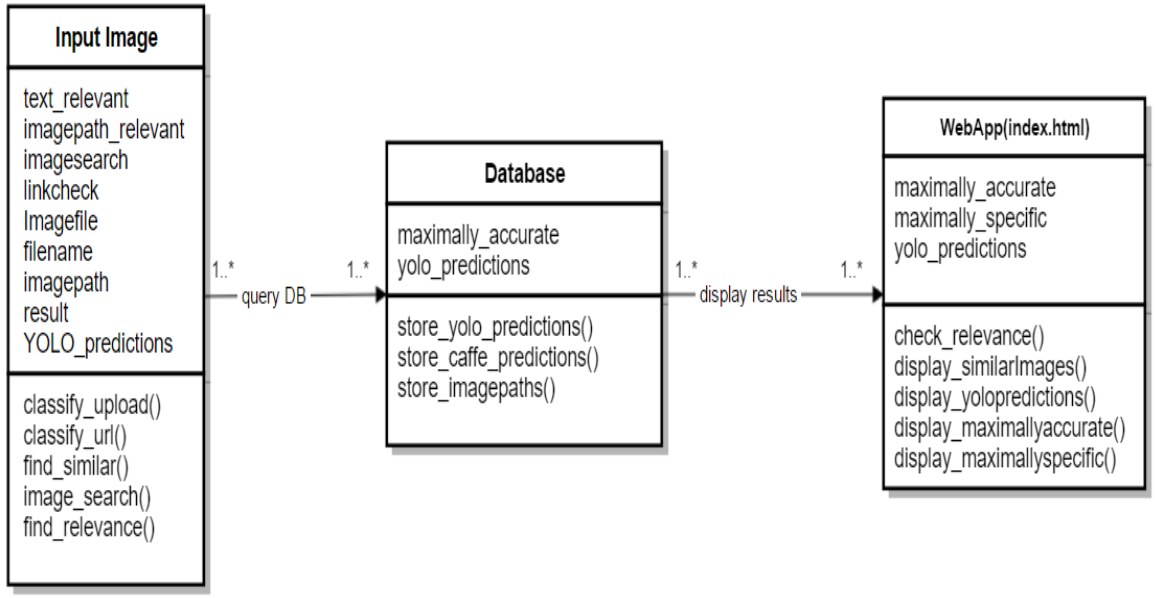


Fig 3.4.2.2 Class Diagram to display the images based on search

Input_Image Class:

- **Text_relevant:** It will have the description of the image to check whether an image is relevant to the description or not.

- **Imagepath_relevant:** It will have path of the image to check relevance.
- **Imagefile:** Image whose predictions are to be stored is specified in Imagefile.
- **Imagepath:** Path of the image whose predictions are to be stored is specified in imagepath.
- **Result:** Caffe Predictions are stored in this variable.
- **YOLO_predictions:** YOLO predictions are stored.
- **Folder_name:** It will have folder name which contains multiple images.
- **Start_yolo ():** This method will process the image and gives the YOLO predictions.
- **Classify_upload ():** This method takes the physical location of the image as an input and classify the image to get scene classification results.
- **Classify_url ():** This method takes the URL of the image as an input and classify the image to get scene classification results.
- **Store_in_DB ():** Store all these results in the database.

Database Class:

- **Maximally accurate:** This will have top five maximally accurate results.
- **Maximally specific:** This will have top five maximally specific results.
- **Store_yolo_predictions ():** This method will store YOLO predictions in the database.
- **Store_caffe_predictions ():** This method will store Caffe predictions in the database.
- **Store_imagepaths ():** This method will store the paths of the images along with their predictions.

WebApp Class:

Display_similarImages (): This method will display similar images based on the search.

Display_yolopredictions (): This method will display top five object detection results.

Display_maximallyaccurate (): This will display top five maximally accurate results.

Display_maximallyspecific (): This will display top five maximally specific results.

3.4.3 Activity Diagram:

The activity diagram focuses on representing activities or chunks of processing which may or may not correspond to the methods of classes. An activity is a state with an internal action and one or more outgoing transitions which automatically follow the termination of the internal activity.

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows). Activity diagrams show the overall flow of control.

Activity Diagram to store predictions in the database

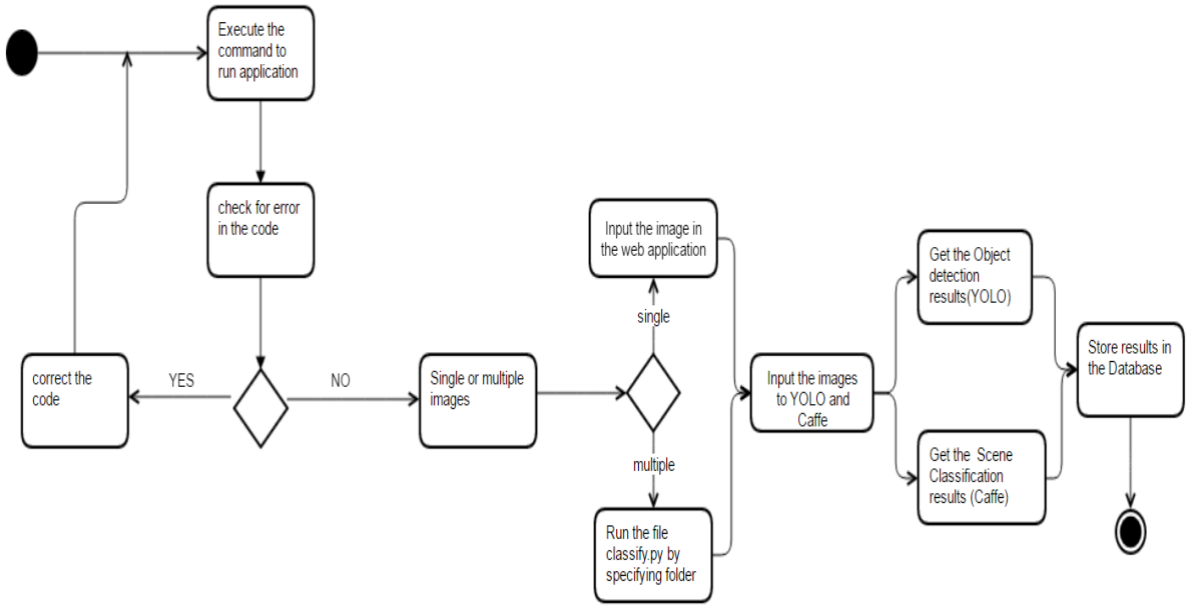


Fig 3.4.3.1 Activity Diagram to store predictions in the database

Stepwise explanation of Fig 3.4.3.1

- Execute the command to start the application.
- If there are no errors in the code, we can proceed to run the application by giving images as input.
- Python application will process it and gives object detection results and scene classification results
- These results are to be stored in the database.

Activity Diagram to search images in the database based on description

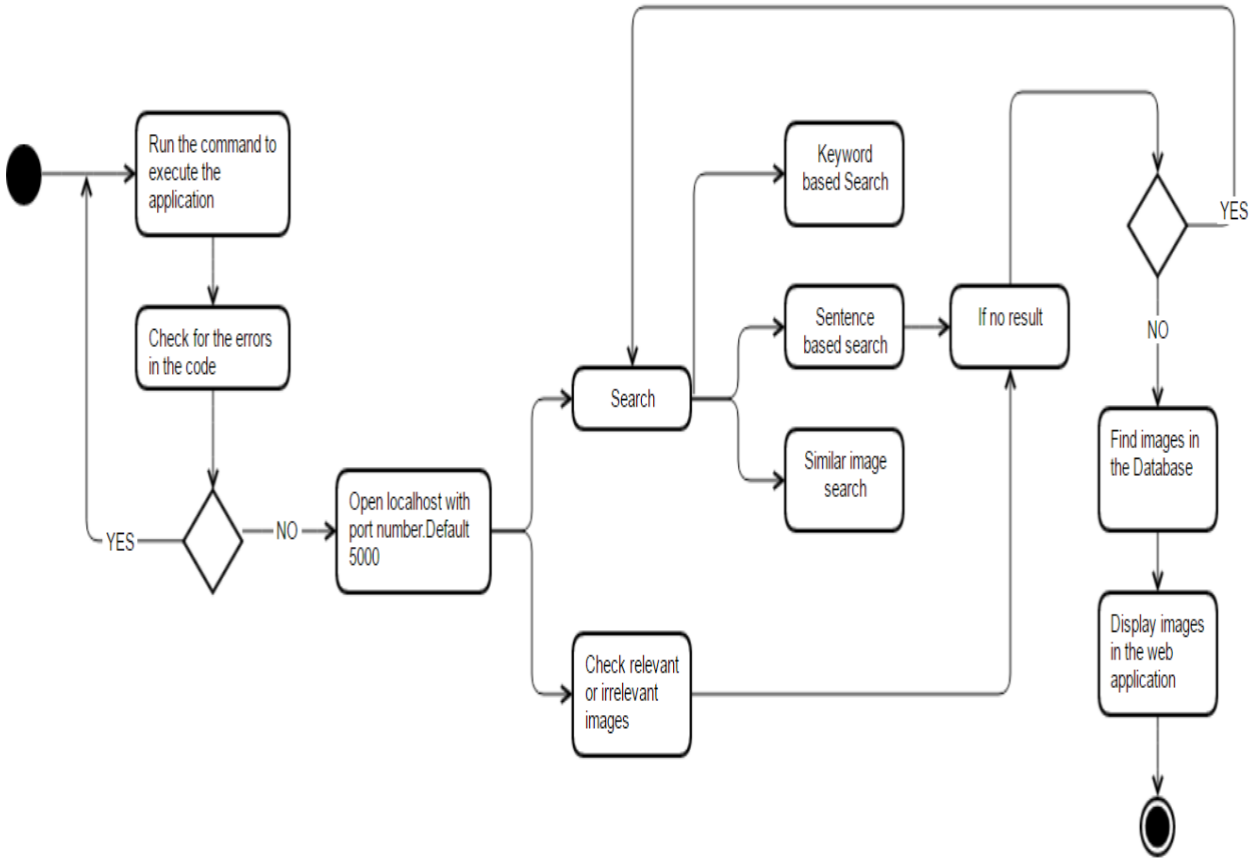


Fig 3.4.3.2 Activity Diagram to search images in the database based on description

Stepwise explanation of Fig 3.4.3.2

- Execute the command to start the application.
- If there are no errors in the code, we can proceed to search for the images in database by giving some description
- Search can be done by giving single keyword, sentence with multiple words and in the image. It will match the description with the predictions that are stored in the database and gives the result.
- It can also check whether an image is relevant or not based on the given context.
- If we have no results found we can proceed to search images with other description.
- Display the images in the web application.

3.4.4 Sequence Diagrams:

A Sequence diagram describes the dynamics of a certain scenario as well as the communication patterns among the objects associated with that scenario. The rectangles around the lifeline are known as *Activation Bars*, which indicate that the object has control while executing a method. In the most common form, the sequence diagram illustrates the behavior of a single method.

The next step in the conceptual design phase was to draw some basic sequence diagrams. The sequence diagrams were mainly used to solidify the knowledge of the system's objects that have already been presented in the class diagram.

Sequence Diagram: This diagram is simple and visually logical, so it is easy to see the sequence of the flow of control. It also clearly shows concurrent processes and activations in a design.

Object: Object can be viewed as an entity at a particular point in time with a specific value and as a holder of identity that has different values over time. Associations among objects are not shown. When you place an object tag in the design area, a lifeline is automatically drawn and attached to that object tag.

Actor: An actor represents a coherent set of roles that users of a system play when interacting with the use cases of the system. An actor participates in use cases to accomplish an overall purpose. An actor can represent the role of a human, a device, or any other systems.

Message: A message is a sending of a signal from one sender object to other receiver object(s). It can also be the call of an operation on receiver object by caller object. The arrow can be labelled with the name of the message (operation or signal) and its argument values

Duration Message: A message that indicates an action will cause transition from one state to another state.

Self-Message: A message that indicates an action will perform at a particular state and stay there.

Create Message: A message that indicates an action that will perform between two states.

Sequence Diagram for storing predictions in the database

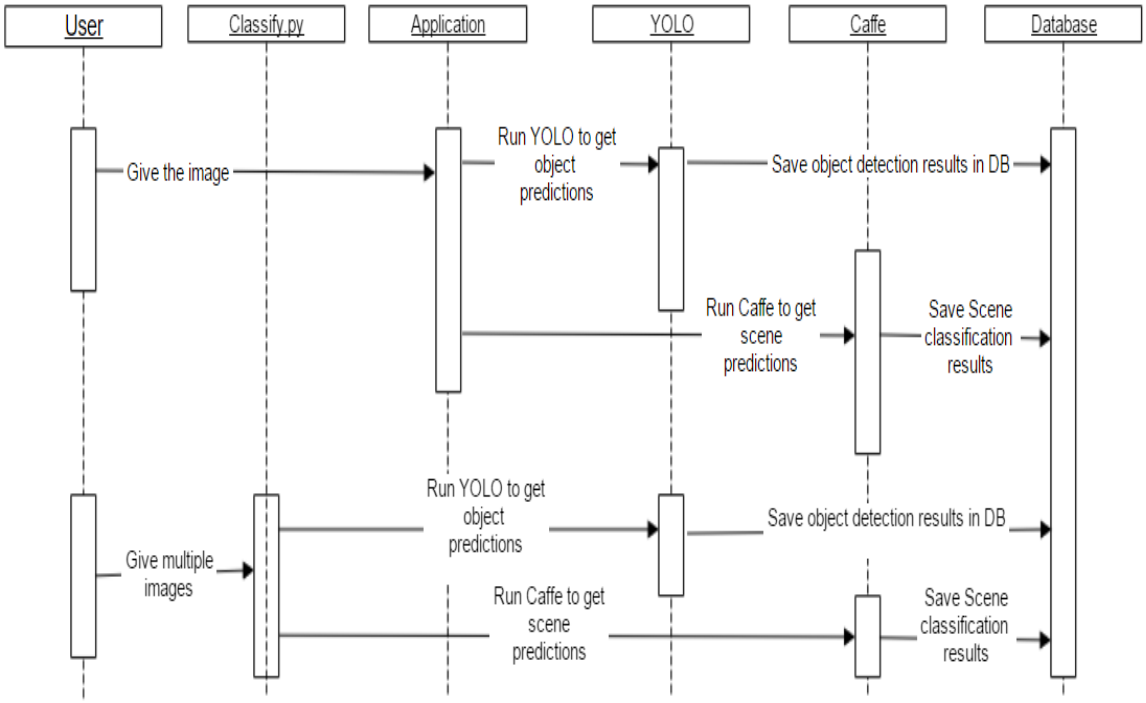


Fig 3.4.4.1 Sequence Diagram for storing predictions in the database

Stepwise explanation of Fig 3.4.4.1

1. User object gives the single image to the application.
2. Application object will run YOLO to store object detection results in the database.
3. Application object will run Caffe to store scene classification results in the database.
4. User object gives multiple images to the classify.py.
5. Classify.py will generate object detection results and scene classification results and stores them in the database

Sequence Diagram for searching images based on description

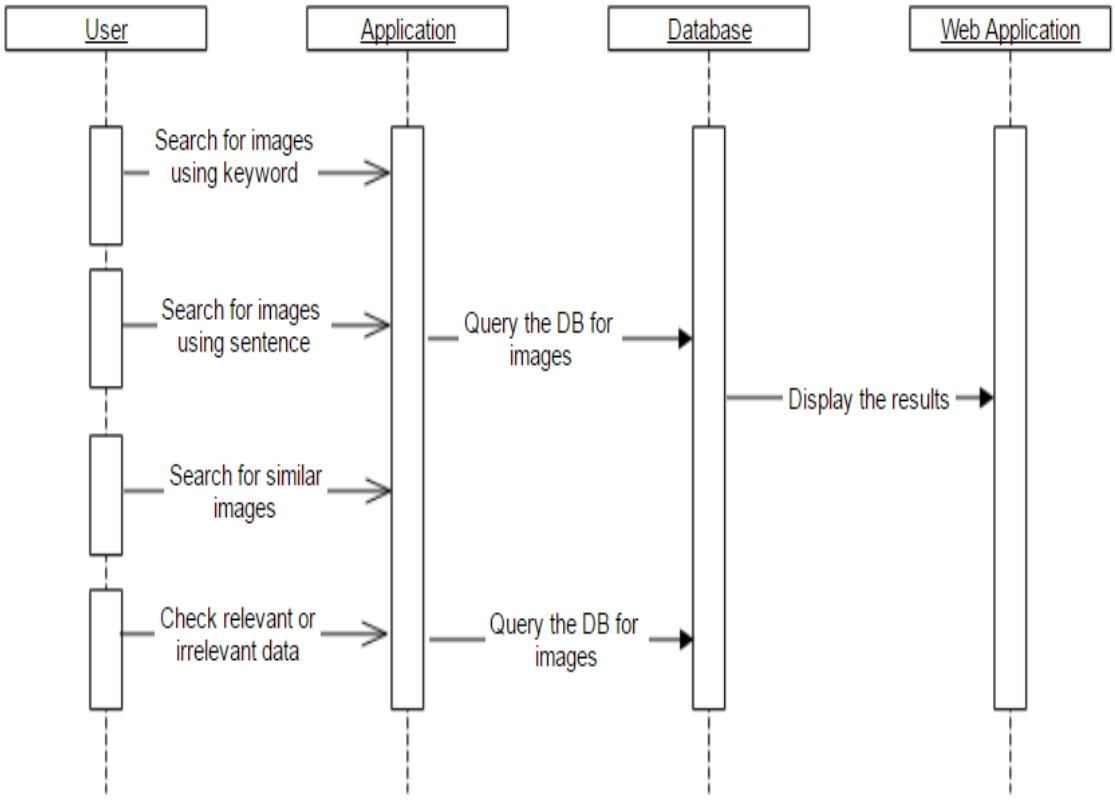


Fig 3.4.4.2 Sequence Diagram for searching images based on description

Stepwise explanation of Fig 3.4.4.2

1. User can search for images by giving single keyword. This keyword is compared with the predictions in the database. Images are retrieved whose images have predictions equal to keyword.
2. User can search images by giving description with multiple words. These words are compared with the predictions in the database. Images are retrieved whose images have predictions equal to words.
3. User can search for similar images by giving input as image. Input image predictions are calculated and compared with the predictions in the database.
4. User can also check whether a particular image is relevant to the context given or not.

3.5 MODEL DESIGN AND ORGANIZATION

The modules in this project are organized into the following:

3.5.1 IMAGE SEARCH USING A KEYWORD

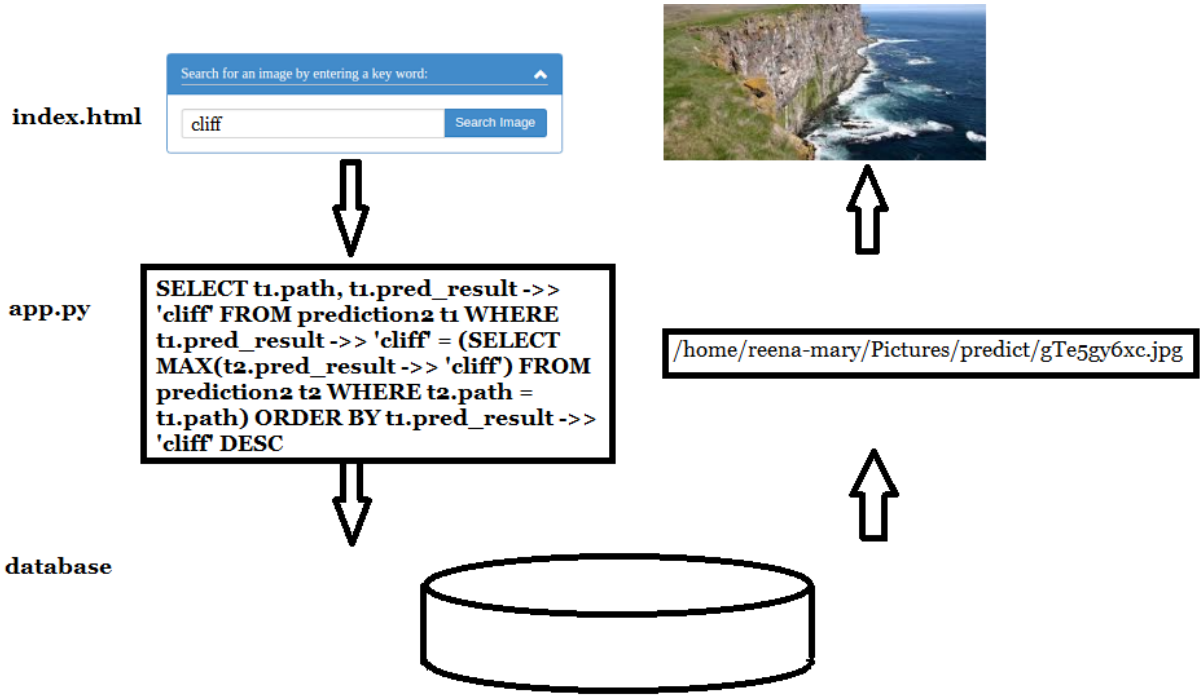


Fig 3.5.1.1 Image Search using a Keyword

Image search is possible upon entering a keyword whose image you need to find. Upon searching for a word "cliff", the word is passed to app.py and a query is formed to perform the required search for the given word. The word is now searched in the database and is ranked based on the confidence scores and the image corresponding to the prediction is now fetched and displayed to the user. Therefore, image search is done based on the pre-saved predictions for images and not based on the name of the image.

3.5.2 IMAGE SEARCH USING A SENTENTIAL DESCRIPTION

When someone issues a search request using a description, a lot of things happen. First, we have to translate natural language into a search query that the computer systems actually understand. In Python, we can use the **nltk** package for this purpose. Then we have to figure out the set of photos that have terms that match the query. Then we have to rank those photos according to relevance, based on the scores of the terms that match. Then we build up a web page to display those results and send that to the user.

Extraction of nouns:

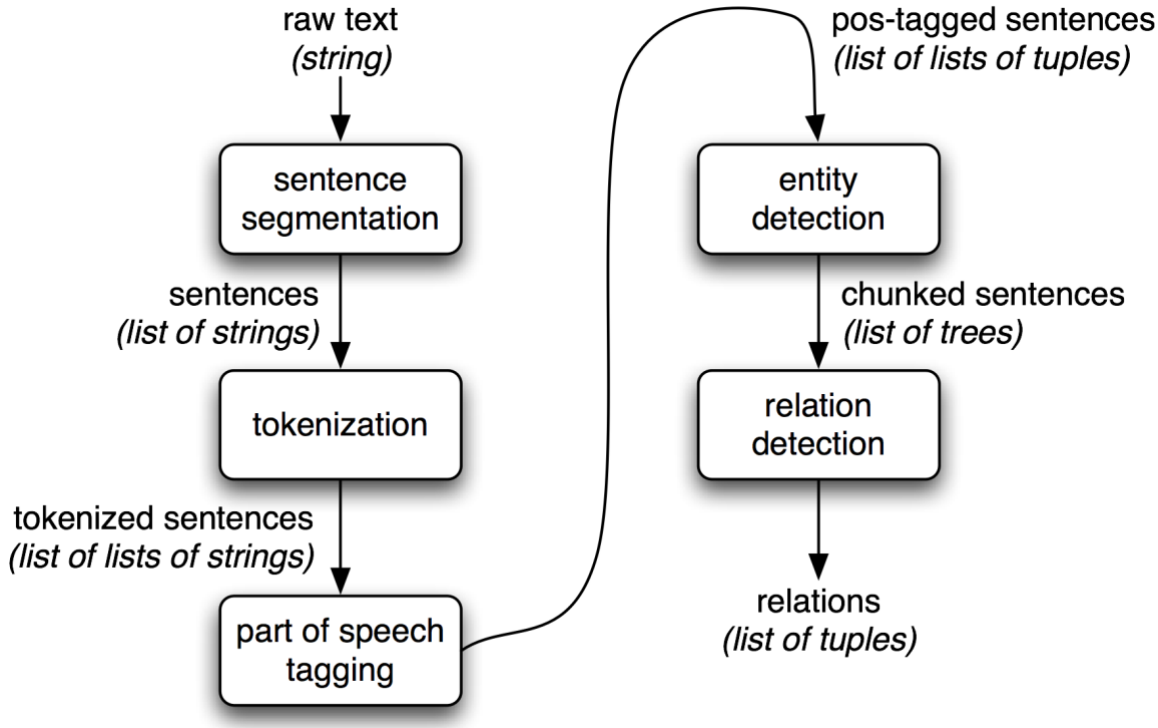


Fig 3.5.2.1 Noun Extraction using NLTK package

Stemming Algorithm:

In linguistic morphology and information retrieval, **stemming** is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation. Stemming programs are commonly referred to as stemming algorithms or stemmers.

The Porter **stemming algorithm** (or 'Porter **stemmer**') is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalization process that is usually done when setting up Information Retrieval systems.

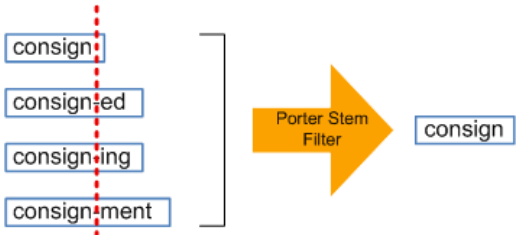


Fig 3.5.2.2 Porter Stemmer

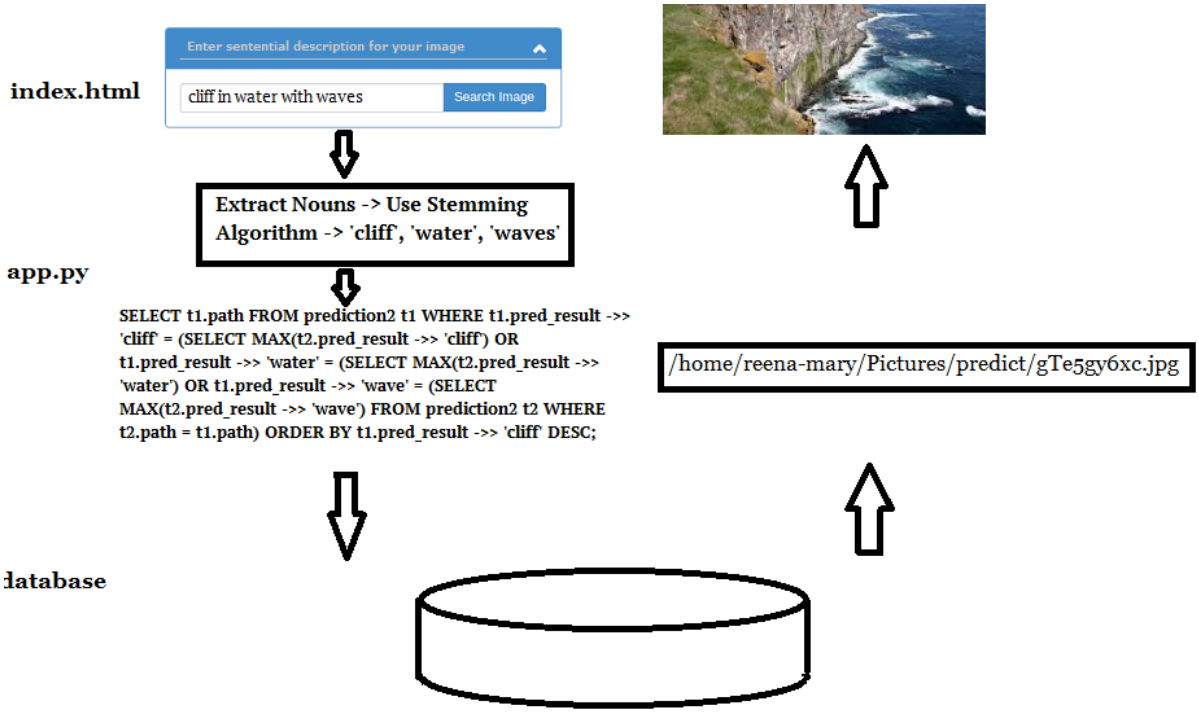


Fig 3.5.2.3 Image Search using Sentential Description

3.5.3 SIMILAR IMAGE SEARCH

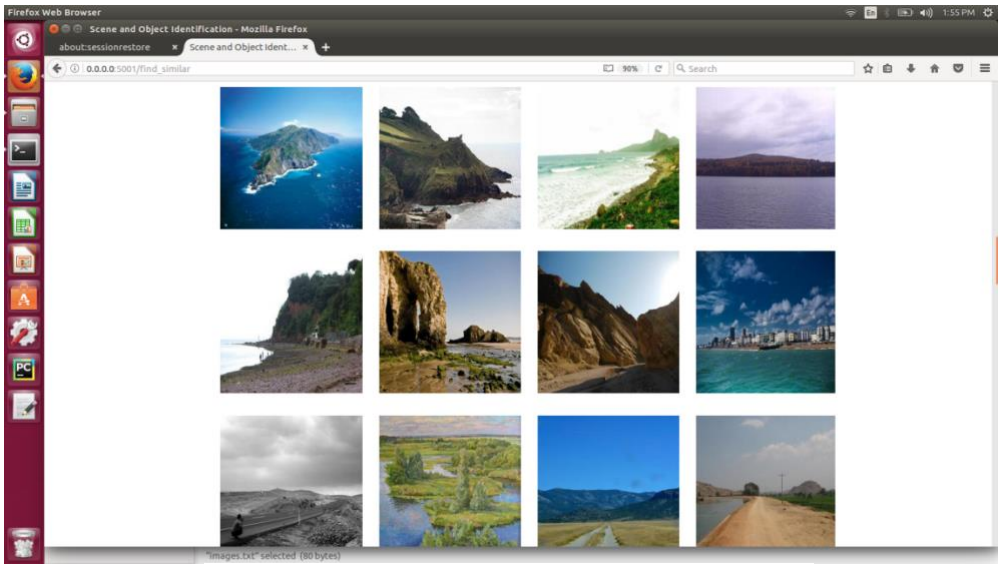


Fig 3.5.3.1 Similar Image Search Results

3.5.5 SCENE CLASSIFICATION USING CAFFE

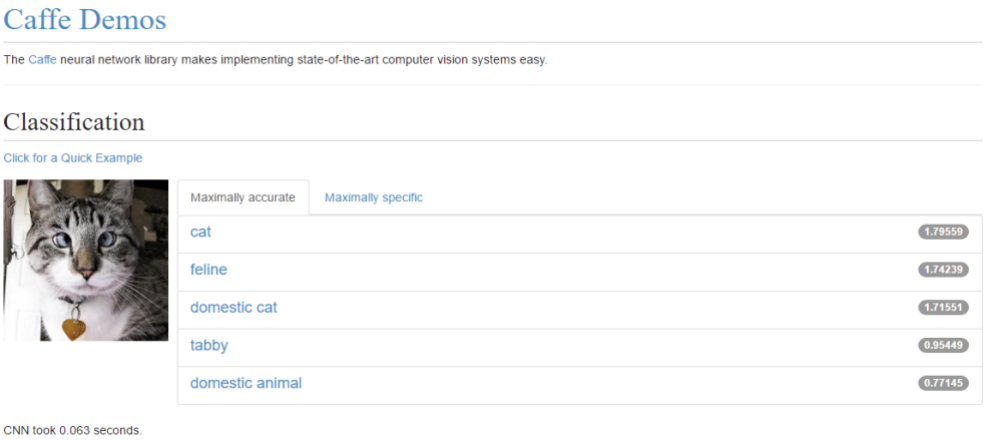
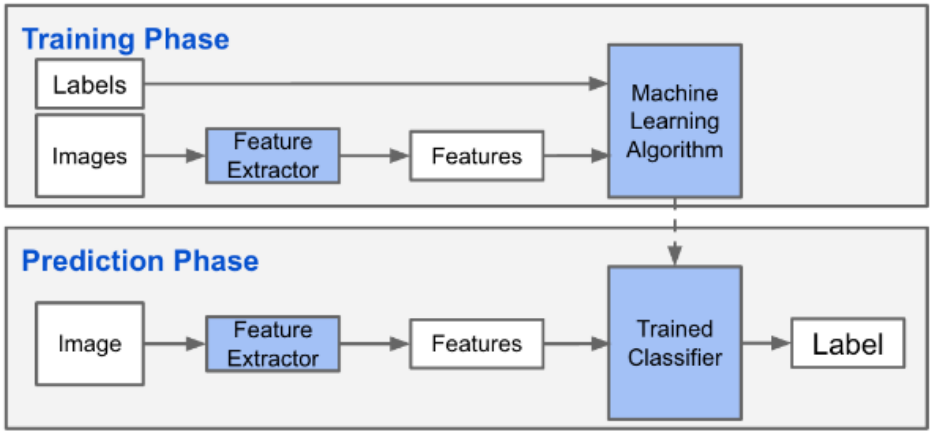


Fig 3.5.5.1 Scene Classification

Classification using a machine learning algorithm has 2 phases- In **training** phase, we train a machine learning algorithm using a dataset comprised of the images and their corresponding labels. In **prediction** phase, we utilize the trained model to predict labels of unseen images.



Machine Learning Phases

Fig 3.5.5.2 Machine Learning Phases

The training phase for an image classification problem has 2 main steps:

1. **Feature Extraction:** In this phase, we utilize domain knowledge to extract new features that will be used by the machine learning algorithm. HoG and SIFT are examples of features used in image classification.

2. **Model Training:** In this phase, we utilize a clean dataset composed of the images' features and the corresponding labels to train the machine learning model. In the predicition phase, we apply the same feature extraction process to the new images and we pass the features to the trained machine learning algorithm to predict the label.

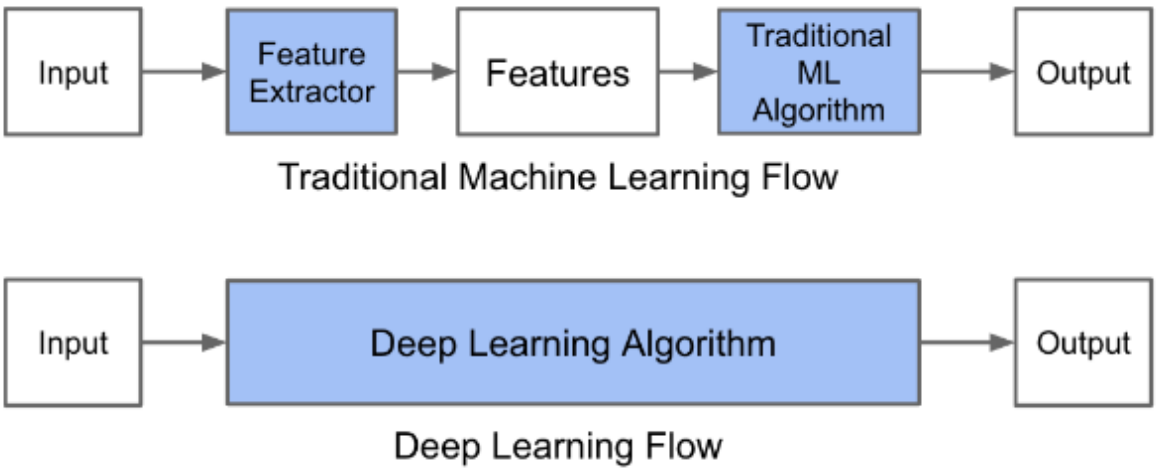


Fig 3.5.5.3 Machine Learning Versus Deep Learning

The main difference between traditional machine learning and deep learning algorithms is in the feature engineering. In traditional machine learning algorithms, we need to hand-craft the features. By contrast, in deep learning algorithms feature engineering is done automatically by the algorithm. Feature engineering is difficult, time-consuming and requires domain expertise. The promise of deep learning is more accurate machine learning algorithms compared to traditional machine learning with less or no feature engineering.

ImageNet predictions are made at the leaf nodes, but the organization of the project allows leaf nodes to be united via more general parent nodes, with 'entity' at the very top. To give "maximally accurate" results, we "back off" from maximally specific predictions to maintain a high accuracy. The bet_file that is loaded in the demo provides the graph structure and names of all relevant ImageNet nodes as well as measures of information gain between them.

Scene classification is made possible because of several files that include:

- deploy.prototxt
- bvlc_reference_caffenet.caffemodel- model on which data is trained
- ilsvrc_2012_mean.npy- mean computed on all images

  Generate the mean image of training data. We will substract the mean image from each input image to ensure every feature pixel has zero mean.
- synset_words.txt- labels file
- imagenet.bet.pickle- file containing bets

3.5.6 OBJECT DETECTION USING YOLO

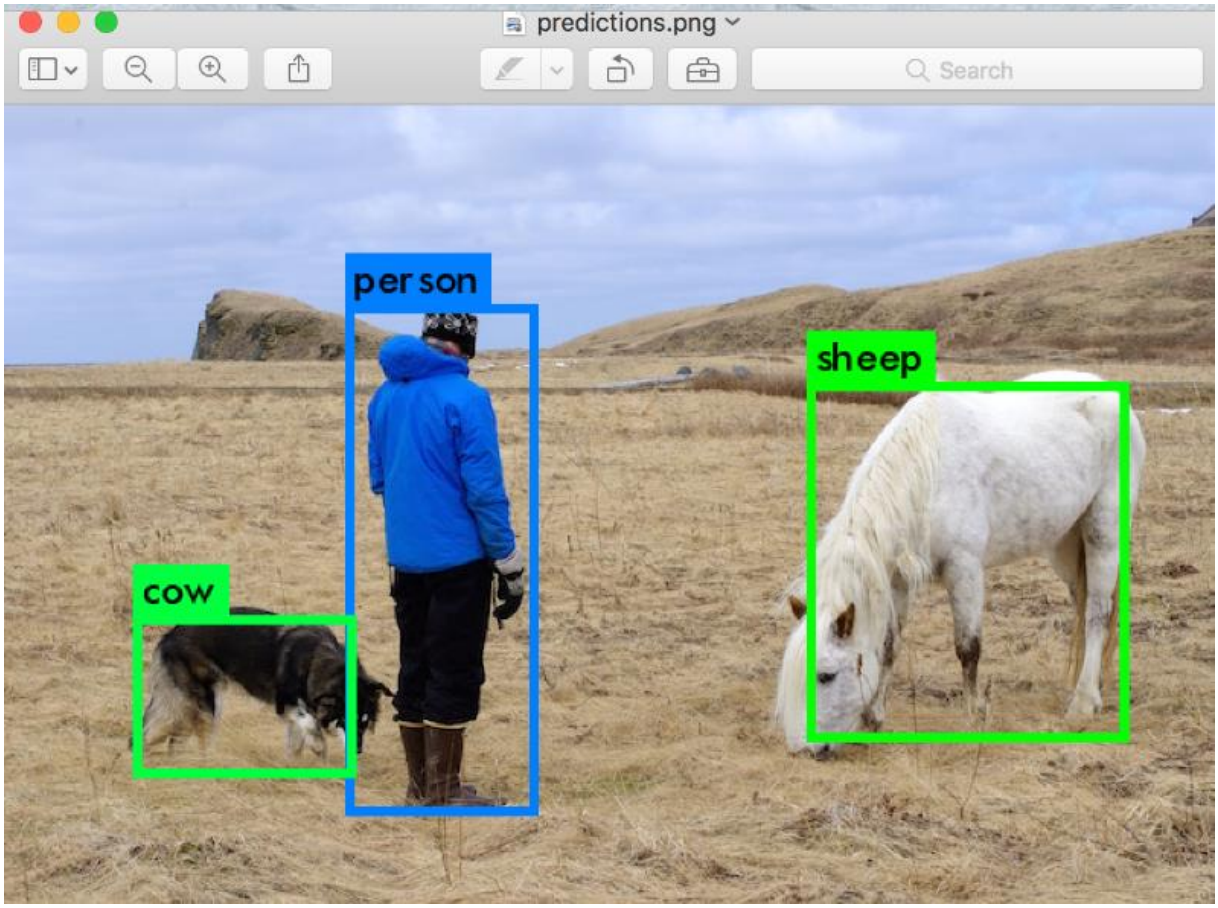


Fig 3.5.6. Object Detection using YOLO

4. IMPLEMENTATION

4.1 INTRODUCTION

The most crucial phase of any project is the implementation. This includes all those activities that take place to convert from the old system to the new system. It involves setting up of the system for use by the concerned end user. A successful implementation involves a high level of interaction between the analyst, programmers and the end user. The most common method of implementation is the phased approach, which involves installation of the system concurrently with the existing system. This has its advantage in that the normal activity carried out, as part of the existing system is anyway hampered. The end users are provided with sufficient documentation and adequate training in the form of demonstration/presentation in order to familiarize with the system.

4.1.1 What is Scene and Object?

Our project mainly focuses on classifying scene, detecting objects and as such it's imperative that we clarify what we mean by a scene. To distinguish scene from 'object' or 'texture', we consider the absolute distance between observer and the fixated zone as the discriminating factor. So, an 'object' is something that subtends about 1 to 2 meters around the observer; but in case of a scene, the distance between the observer and the fixated point is usually more than 5 meters. To put it in common terms, object is something that is at hand distance whereas scene is mainly a place in which one can move.

4.1.2 Why use Caffe?

Scene recognition: Recognition of scene means providing information about the semantic category and the function of the environment**.** It is one of the greatest tasks of computer vision. Much of the recent progress in high-level vision has been made by designing robust image features for recognition tasks such as object and scene recognition. Nearly all of these features are based on some kind of low-level image

With the help of more and more sophisticated statistical models, these features have achieved good successes in high-level recognition tasks. Here, Caffe model is used to recognize the scenes of the images. Scene classification results are generated by the Caffe. These results are stored in database to fetch the images later as per description. In this project, we give images to Caffe model. It gives search labels for each and every image. These search labels are stored in the database.

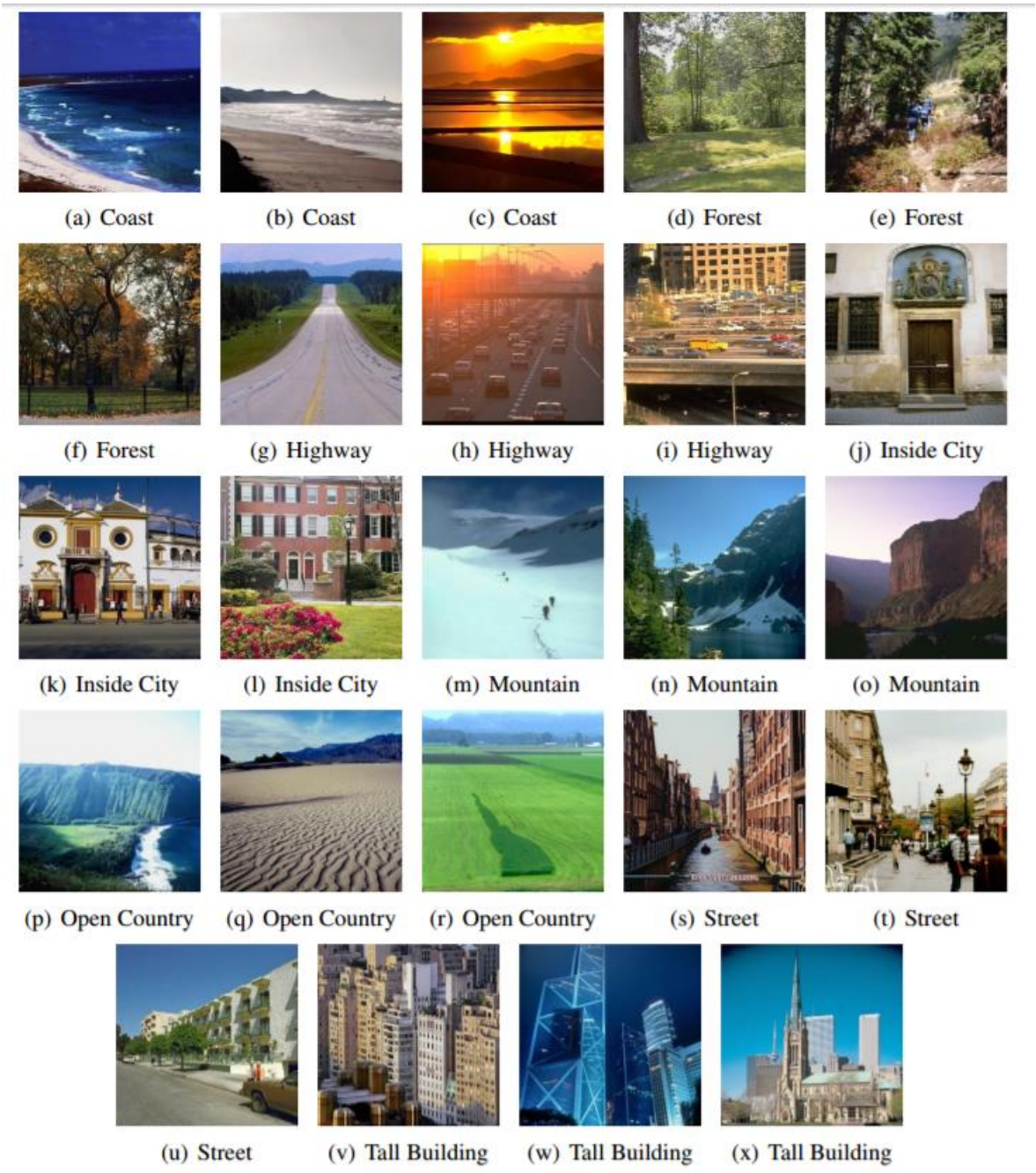


Fig 4.1.2.1 Images with Scene labels

To make machine learn about the scenes, we have to first give the images with labels and train it. In our project we used pre-trained models. Given an image as an input we wish to classify as one of the following

1. Coast 2. Forest 3. Highway 4. Inside City 5. Mountain 6. Open Country 7. Street 8. Tall Buildings. We have to train the model with following images.

4.1.3 Why use YOLO (You Only Look Once)?

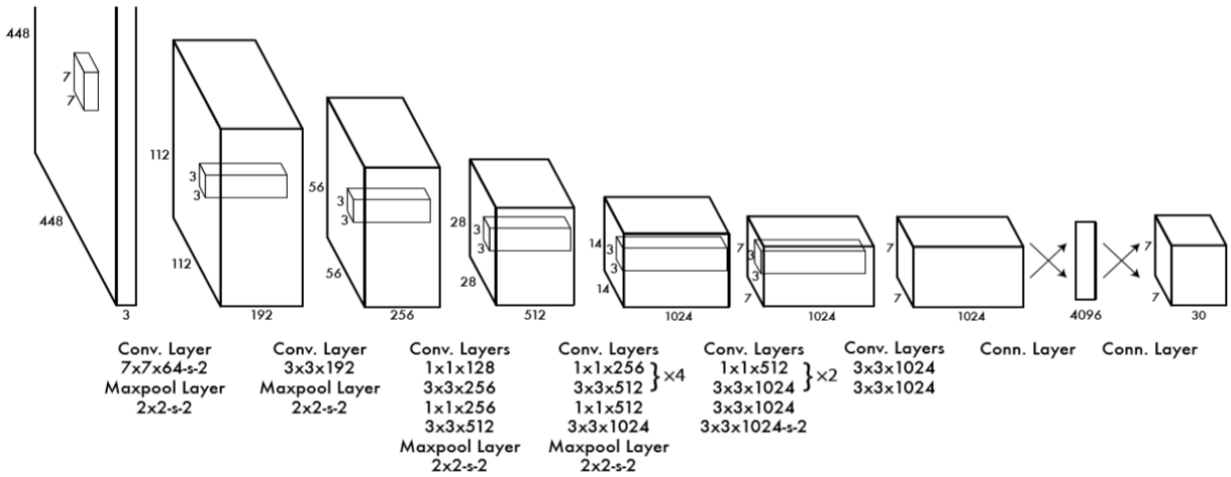


Fig 4.1.3.1 YOLO Architecture

Object Detection: It is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. YOLO is the fastest and accurate real time object detector. Object detection will help us to recognize the things in an image. It uses the concept of Convolutional Neural Network (CNN). It has several advantages over classifier based systems. It looks image only for one time and creates single neural network for object detection.

Other Object detectors:

DPM: It uses sliding window approach where the classifier is run at evenly spaced lcations over the entire image.

R-CNN: It uses region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. Both of these method are looking at the image thousands of times to perform detection. But, YOLO looks only for one time to detect the objects in the image. It can achieve it with ne neural network.

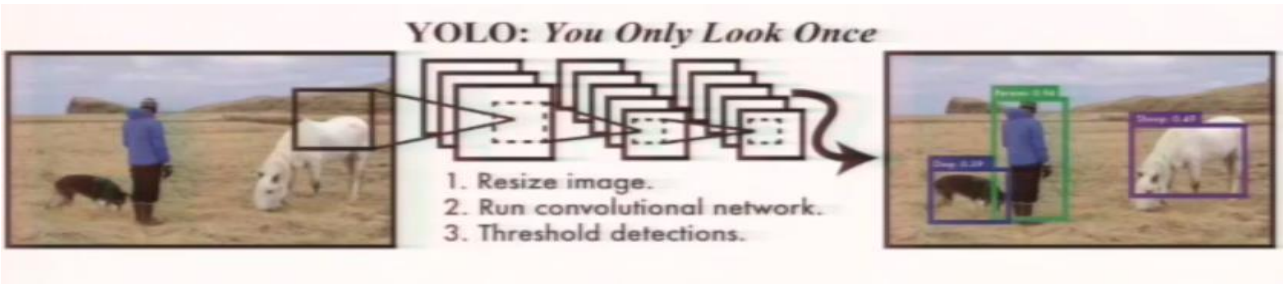


Fig 4.1.3.1 YOLO Single Neural Network

Working of YOLO:

- This system divides the input image into S×S grid. Where each cell will have certain number of bounding boxes
- Each bounding box consists of 5 predictions: x, y, w, h, and confidence.
- It will predict two different things.
- A)Bounding boxes with confidence value:
  1. Confidence reflects how confident the model is that the box contains an object. PR (object).
  2. When we visualize all the predictions together, we see the image with many bounding boxes.
- Class Probabilities:
  1. We only know where the objects are in the image, but we don't know what they are.
  2. Each grid cell will predict the class probabilities. If the grid cell predicts an object (car), this is not saying that grid cell is a car, if it contains some object then it might be a car.
  3. We are going to multiply conditional class probabilities and the individual box confidence predictions.

  $PR(Class(I)|Object) * PR(Object) * IOU(truth\ prediction) = PR(Class(I)) * IOU(truth\ prediction)$

Comparison with other models:

Table 4.1.3.1 Comparison of YOLO's speed with other models

Model	FPS	Speed
DPM v5	**0.07**	**14s / img**
R-CNN	**0.05**	**20s/ img**
FASR R-CNN	**0.5**	**2s/ img**
FASTER R-CNN	**7**	**140ms /img**
YOLO	**45**	**22ms/ img**

Example:

Showing how objects are detected in YOLO

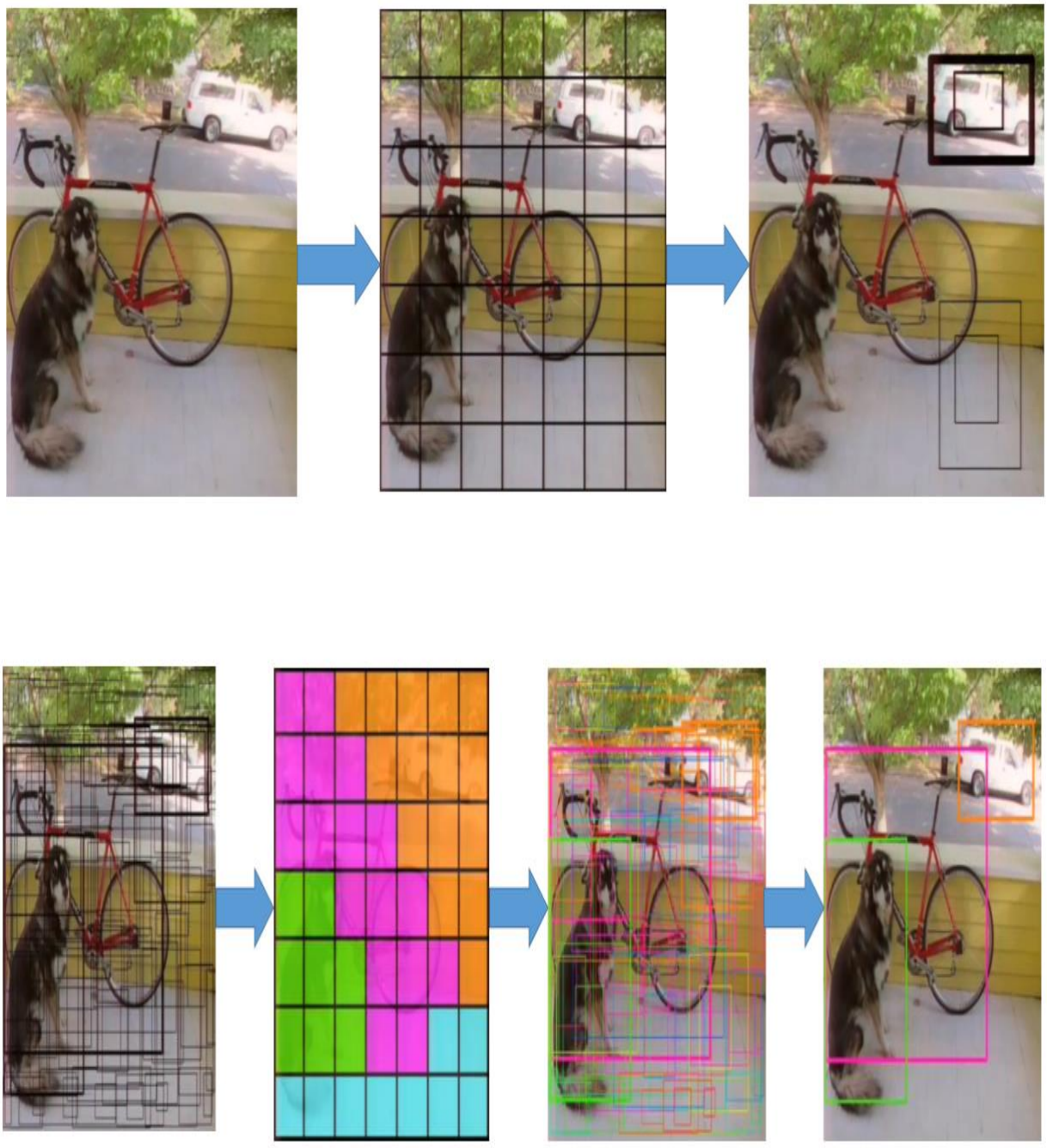


Fig 4.1.3.2 Object detections using YOLO

4.1.4 Why use Caffe and YOLO together?

YOLO is refreshingly simple. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection. First, YOLO is extremely fast. Since it frame detection as a regression problem it don't need a complex pipeline. Simply run neural network on a new image at test 1 time to predict detections. Base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means it can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems. Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method, mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN. Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected input.

Caffe is a deep learning framework developed by the Berkeley Vision and Learning Center (BVLC). It is written in C++ and has Python and Matlab bindings.There are 4 steps in training a CNN using Caffe:

- Data preparation: In this step, we clean the images and store them in a format that can be used by Caffe. We will write a Python script that will handle both image pre-processing and storage.
- Model definition: In this step, we choose a CNN architecture and we define its parameters in a configuration file with extension .prototxt.
- Solver definition: The solver is responsible for model optimization. We define the solver parameters in a configuration file with extension .prototxt.
- Model training: We train the model by executing one Caffe command from the terminal. After training the model, we will get the trained model in a file with extension .caffemodel.

Example: Using Caffe and YOLO together

Caffe is used for scene classification, it will gives scene classification results. YOLO is used for object detections, it gives object detection results. These result are stored as search labels for a particular image. It can only detect either scene or objects if we classify individually.

By combining both the models, when user gives an input image to the python application both YOLO and Caffe predictions are stored as search labels for that image. It enables us to detect an image with description.

4.2 Building the System

4.2.1System Architecture

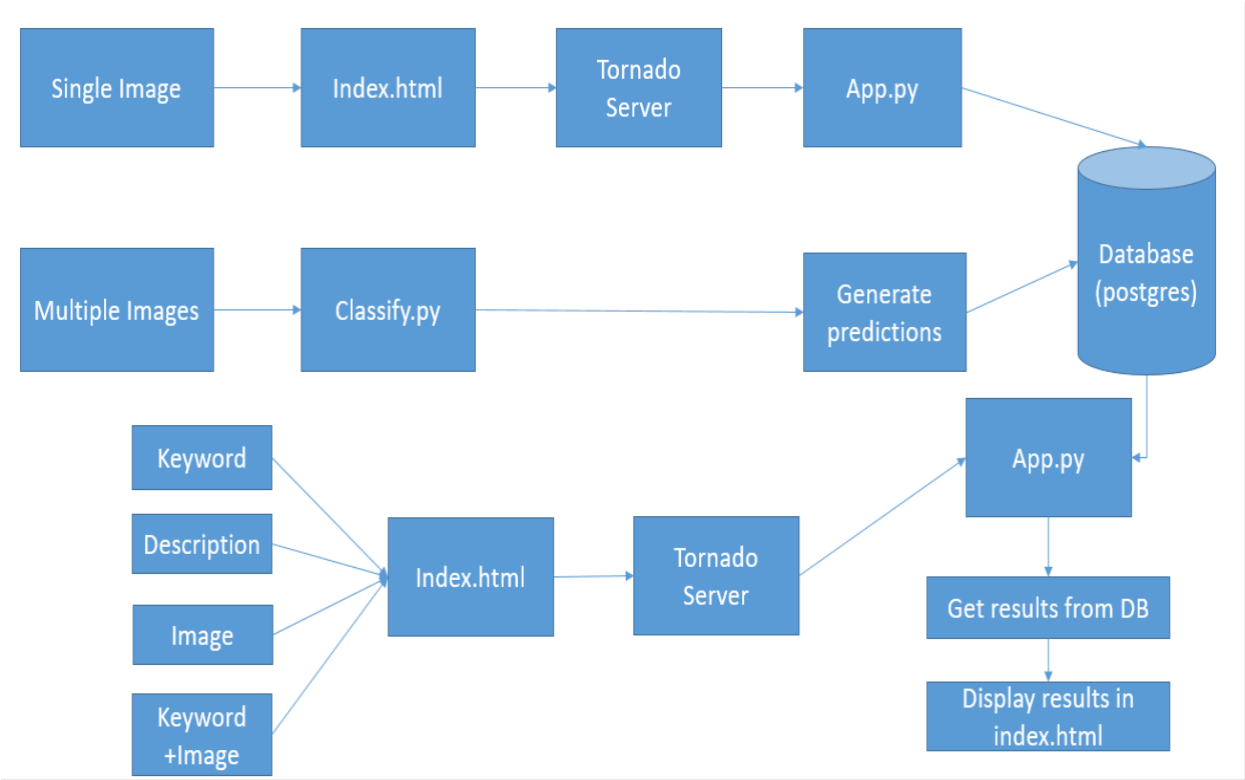


Fig 4.2.1.1 System Architecture

4.2.2 Training to improve accuracy

By training more objects with more images we get even more labels, we can teach our model how to recognize objects it hasn't learnt about yet and increasing its intelligence. Thereby, we can detect more objects in an image. Therefore, search labels stored in database for an image can be increased. So, searching capability is made more efficient and accurate results can be obtained.

Caffe has command line, Python, and MATLAB interfaces for day-to-day usage, interfacing with research code, and rapid prototyping. While Caffe is a C++ library at heart and it exposes a modular interface for development, not every occasion calls for custom compilation. The cmdcaffe, pycaffe, and matcaffe interfaces are here for you.

The command line interface – cmdcaffe – is the caffe tool for model training, scoring, and diagnostics. Run caffe without any arguments for help. This tool and others are found in caffe/build/tools. (The following example calls require completing the LeNet / MNIST example first.)

Training: caffe train learns models from scratch, resumes learning from saved snapshots, and fine-tunes models to new data and tasks:

- All training requires a solver configuration through the -solver solver.prototxt argument.
- Resuming requires the -snapshot model_iter_1000.solverstate argument to load the solver snapshot.
- Fine-tuning requires the -weights model.caffemodel argument for the model initialization.

4.3 Building the Database

PostgreSQL is a powerful, open source object-relational **database** system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness.

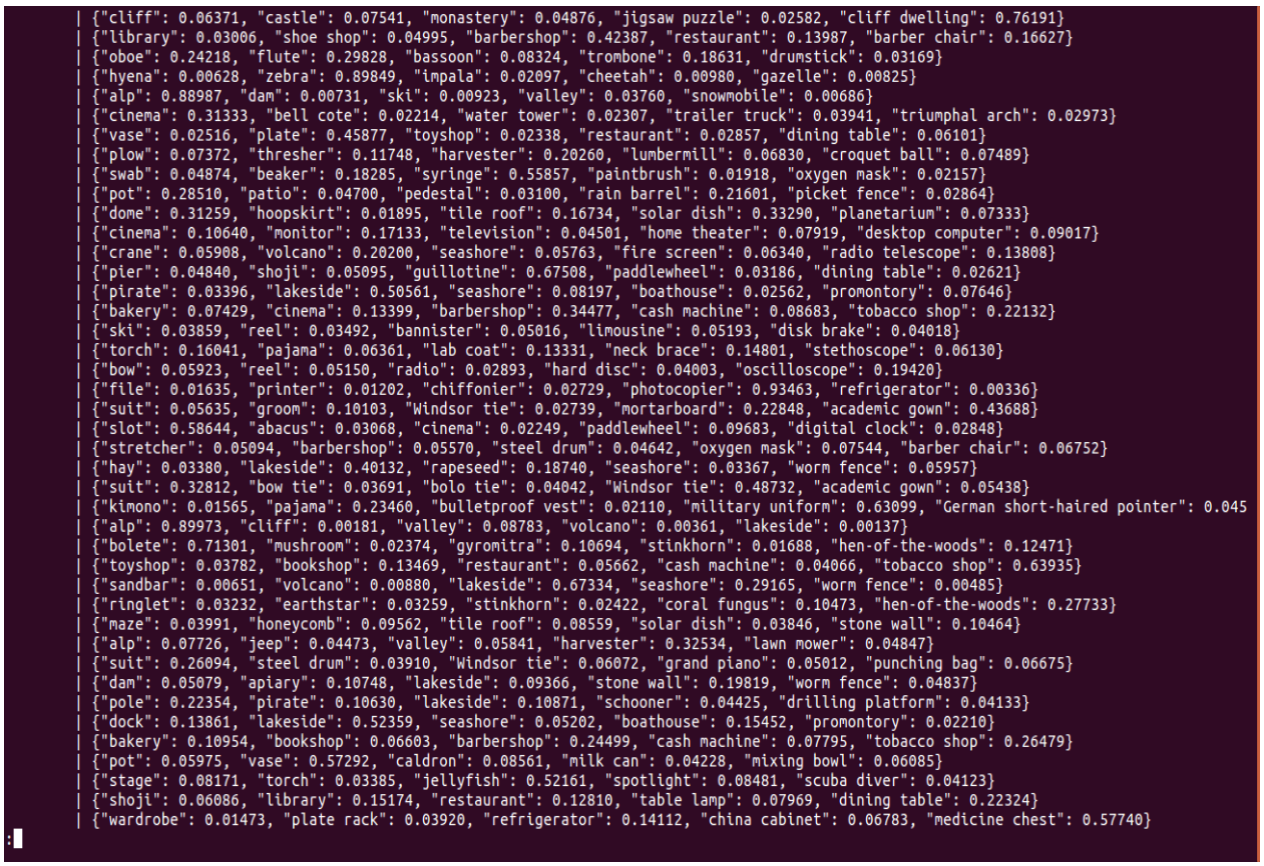


Fig 4.3.1 Screen shot of predictions stored in the database

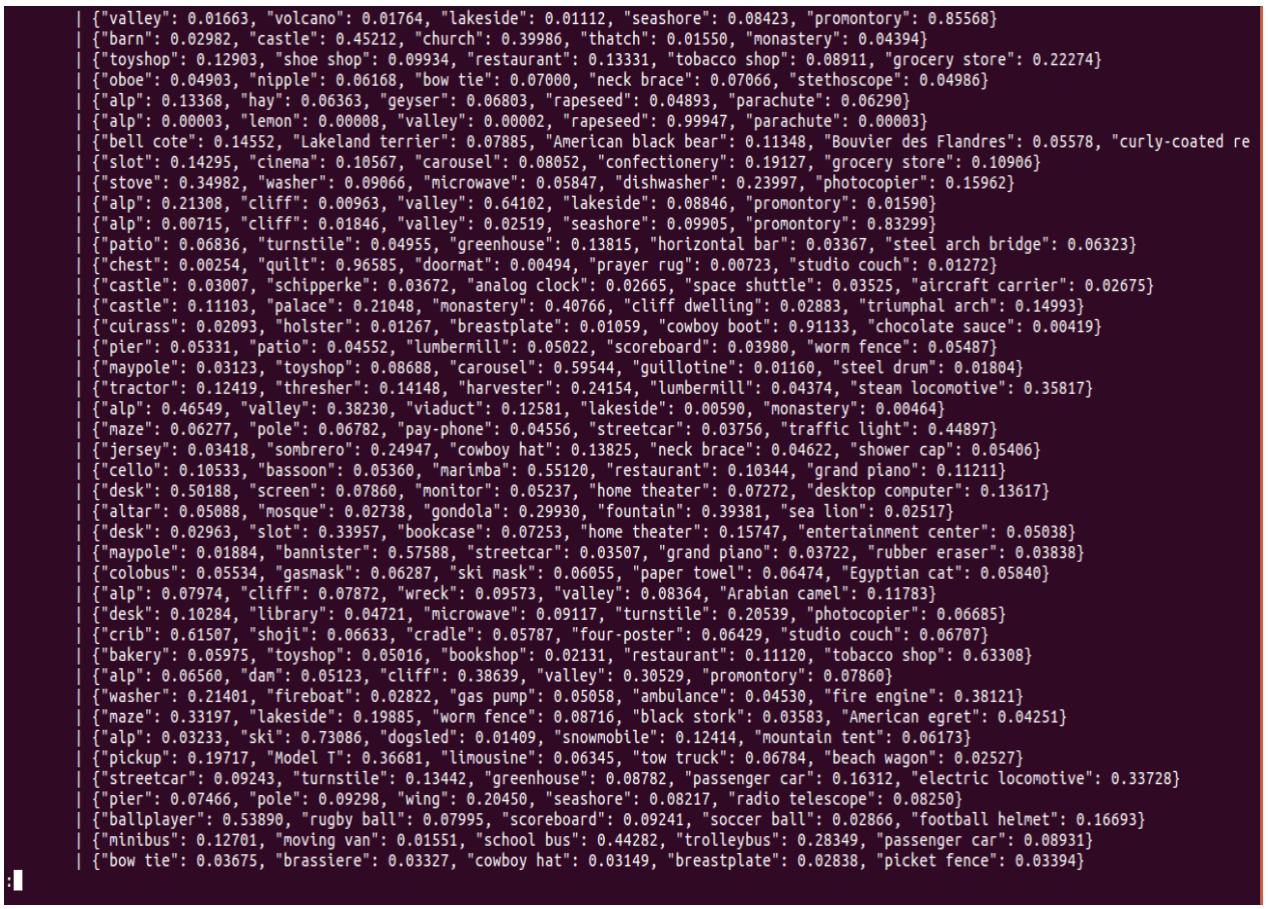


Fig 4.3.2 Screen shot of predictions stored in the database

4.4 OUTPUT SCREENS:

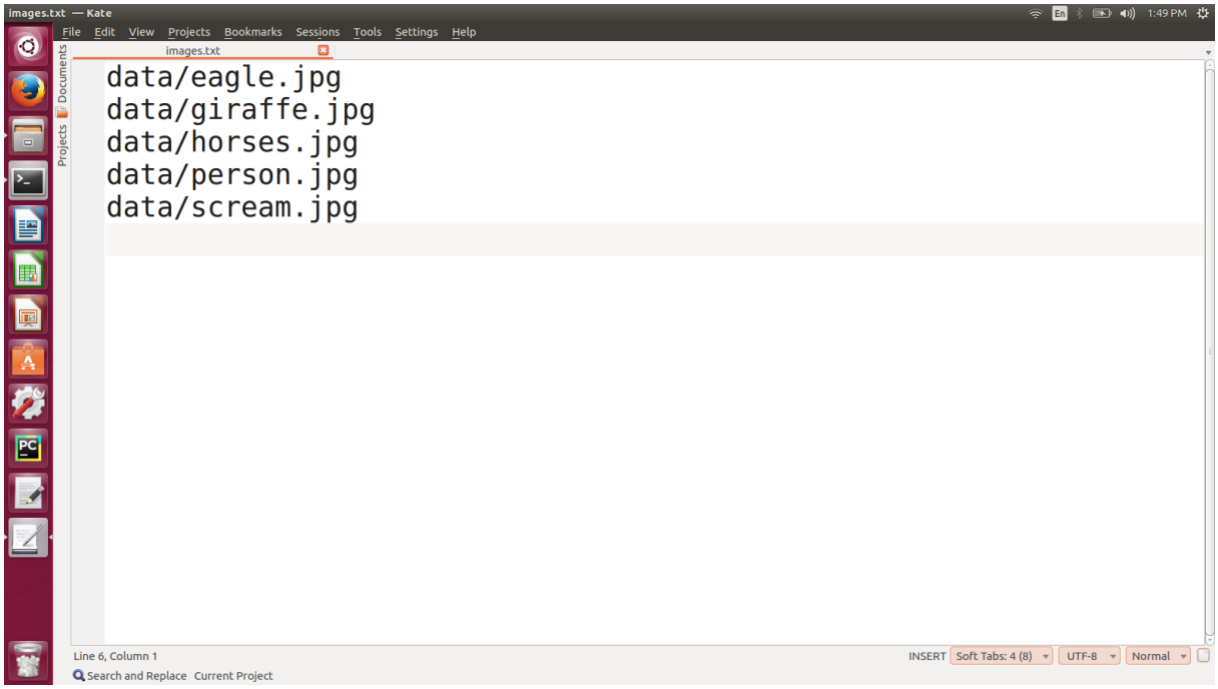


Fig 4.4.1 File- "images.txt" containing list of input images to be given to YOLO

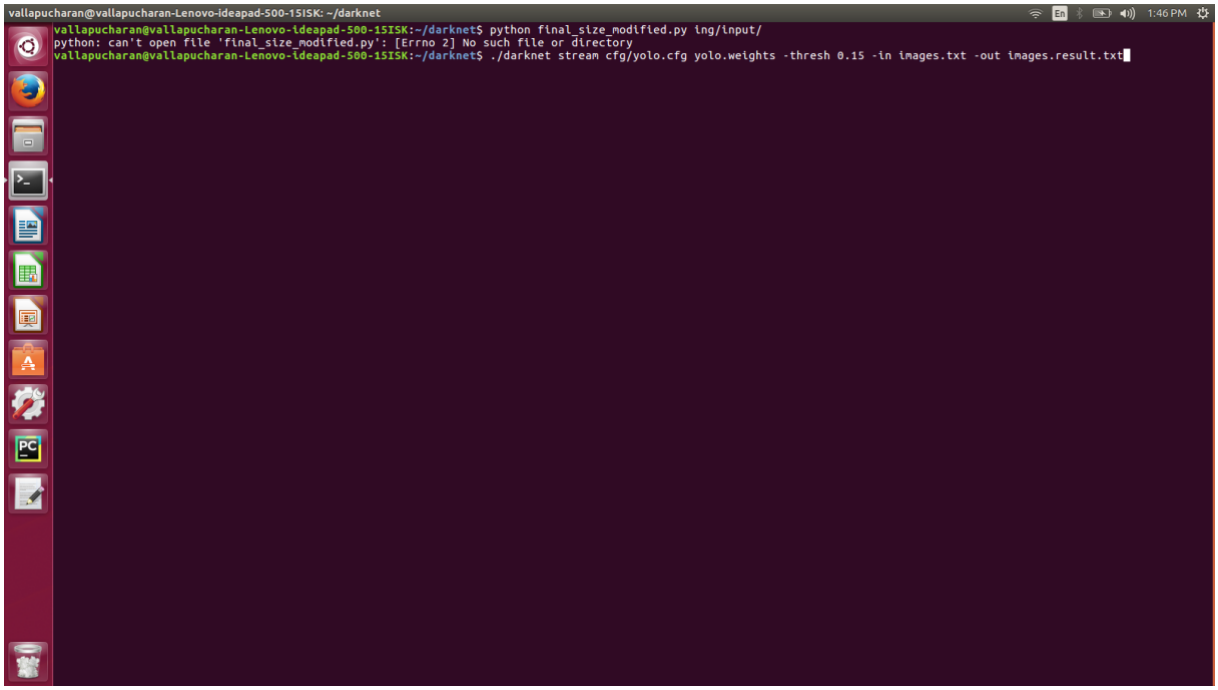


Fig 4.4.2 Running YOLO with images.txt as input and images.result.txt as output

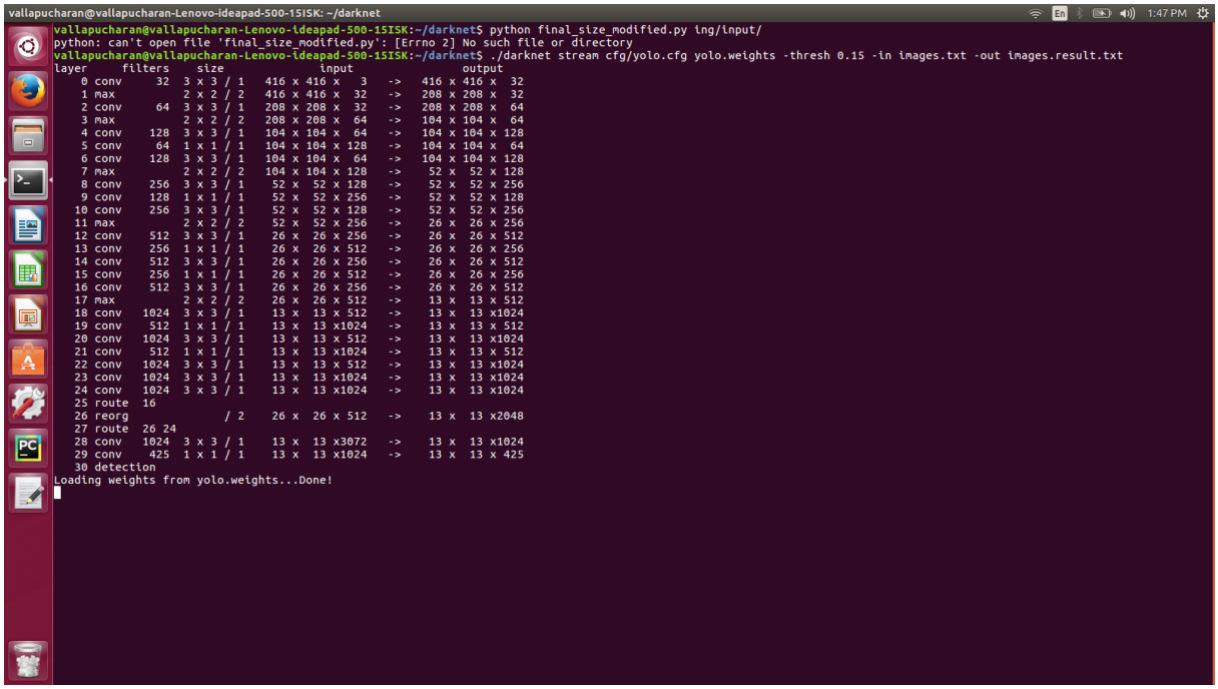


Fig 4.4.3 YOLO running the image through different layers of Network

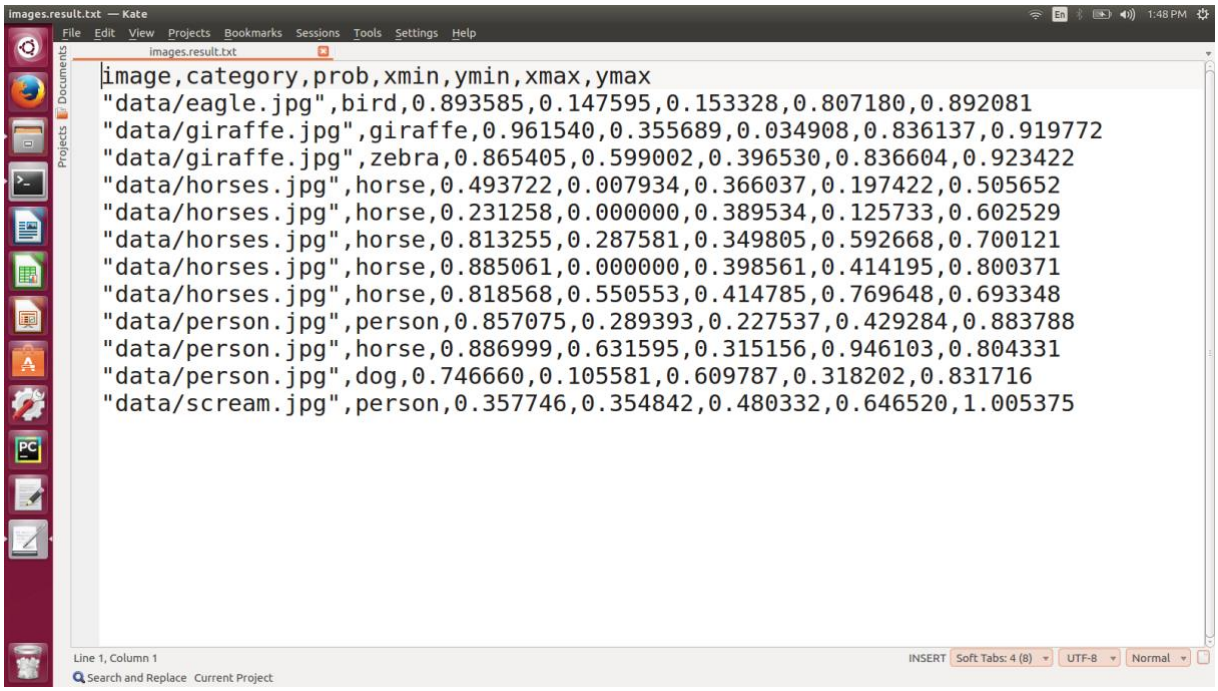


Fig 4.4.4 Saving values from detector.c into a csv file

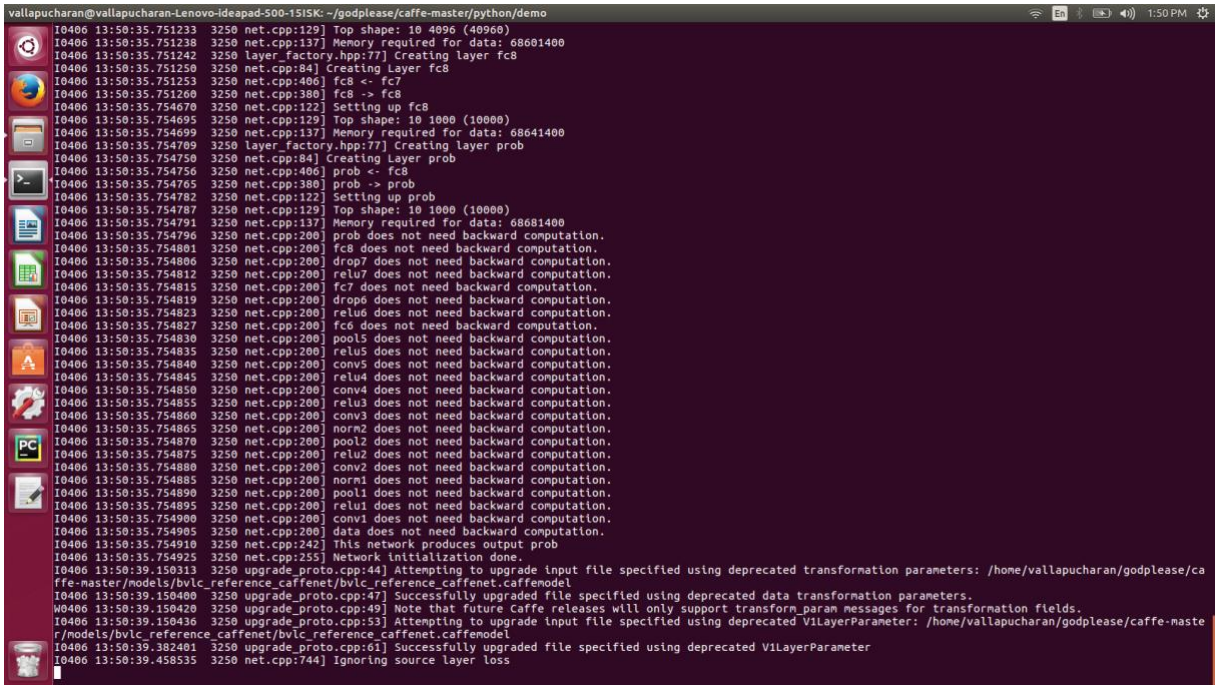


Fig 4.4.5 Caffe running initializations before Tornado server starts running

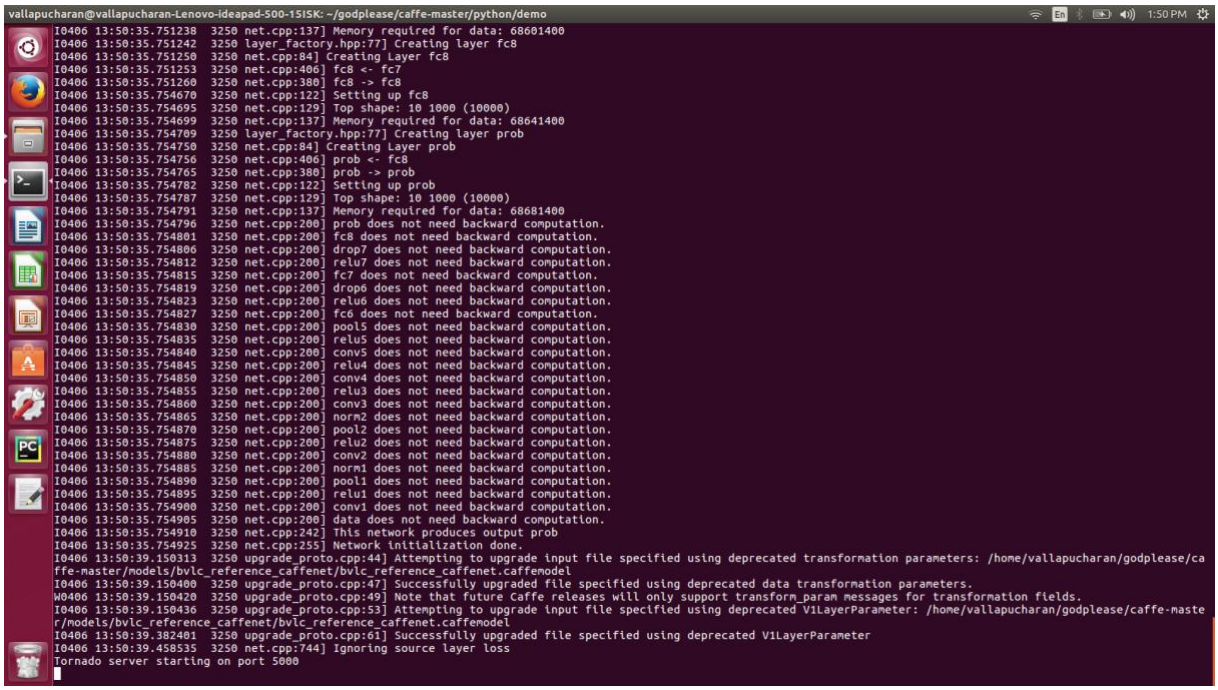


Fig 4.4.6 Tornado server starting on port 5000

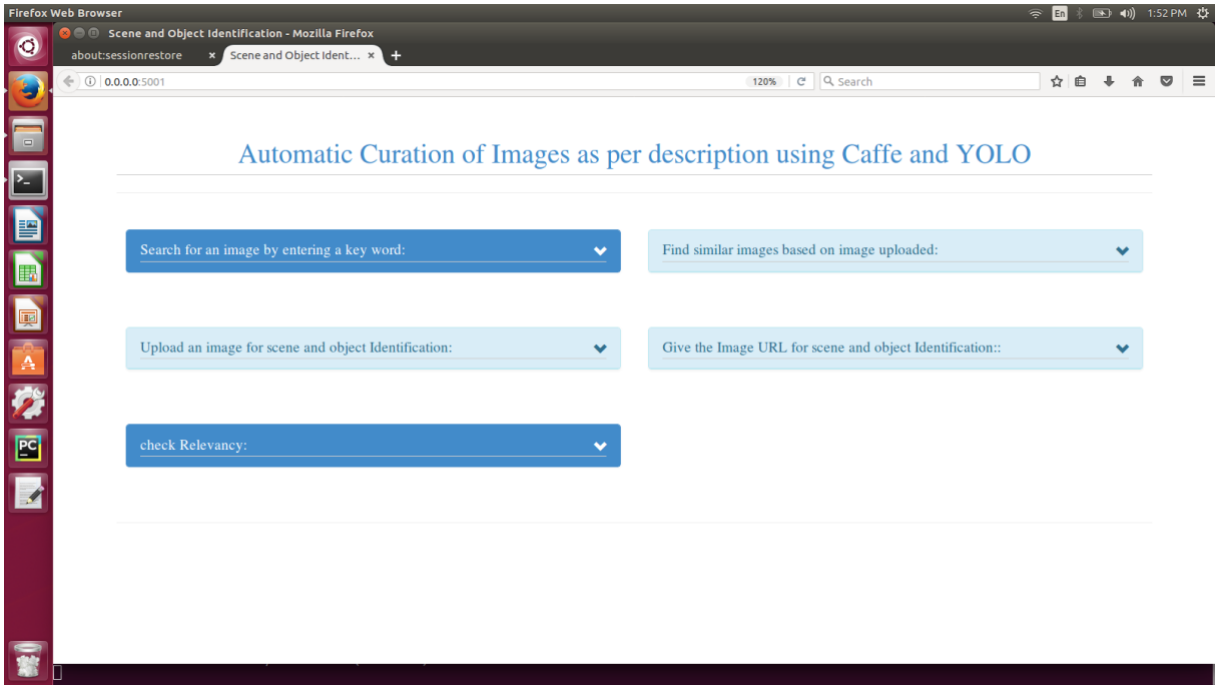


Fig 4.4.7 Querying screen for curating content (Collapsed Panels)

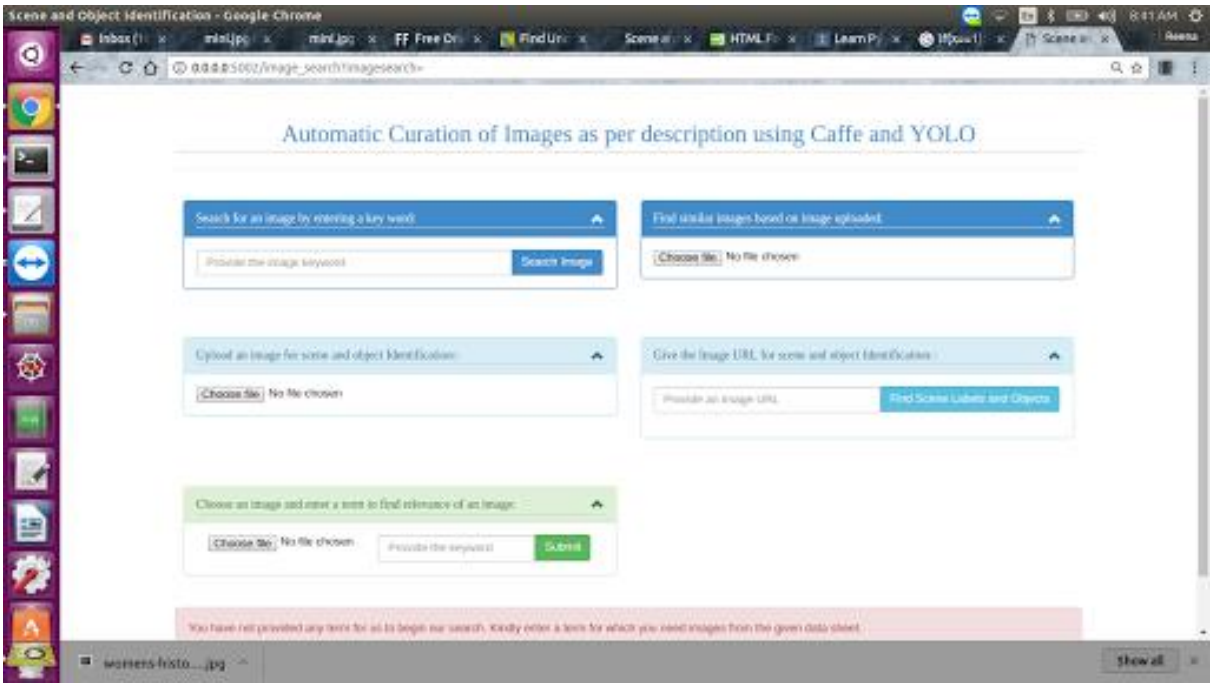


Fig 4.4.8 Querying screen for curating content (Un-collapsed Panels)

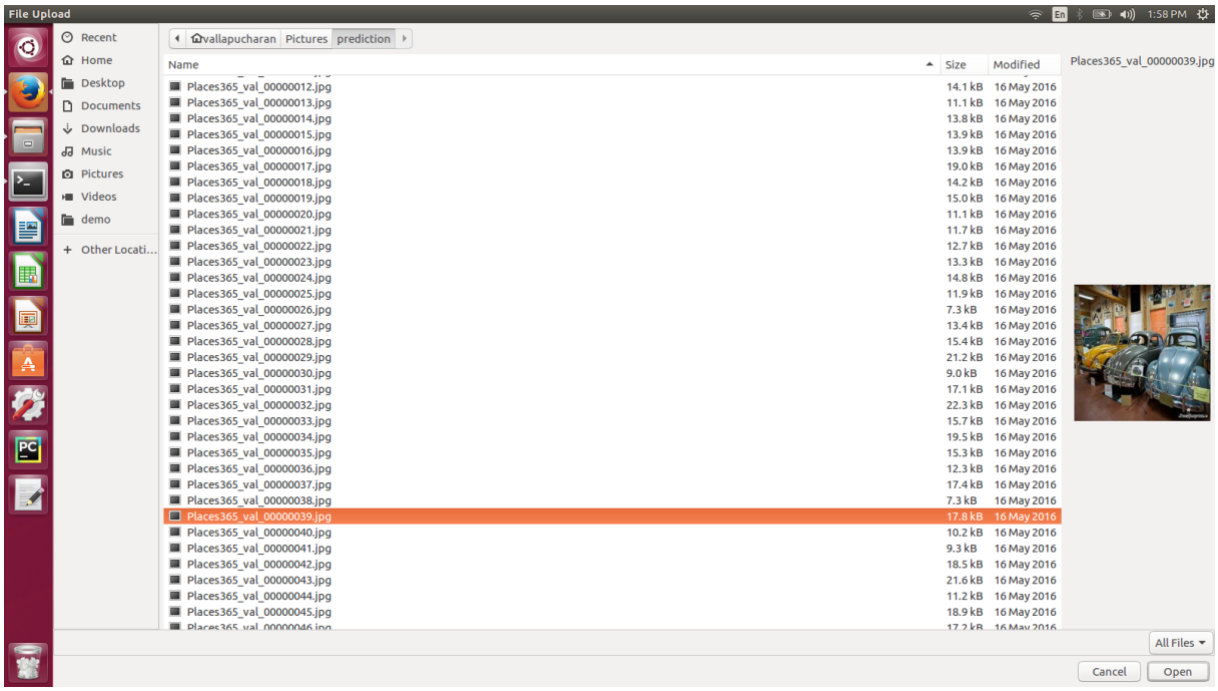


Fig 4.4.9 Uploading an image to find scene and objects

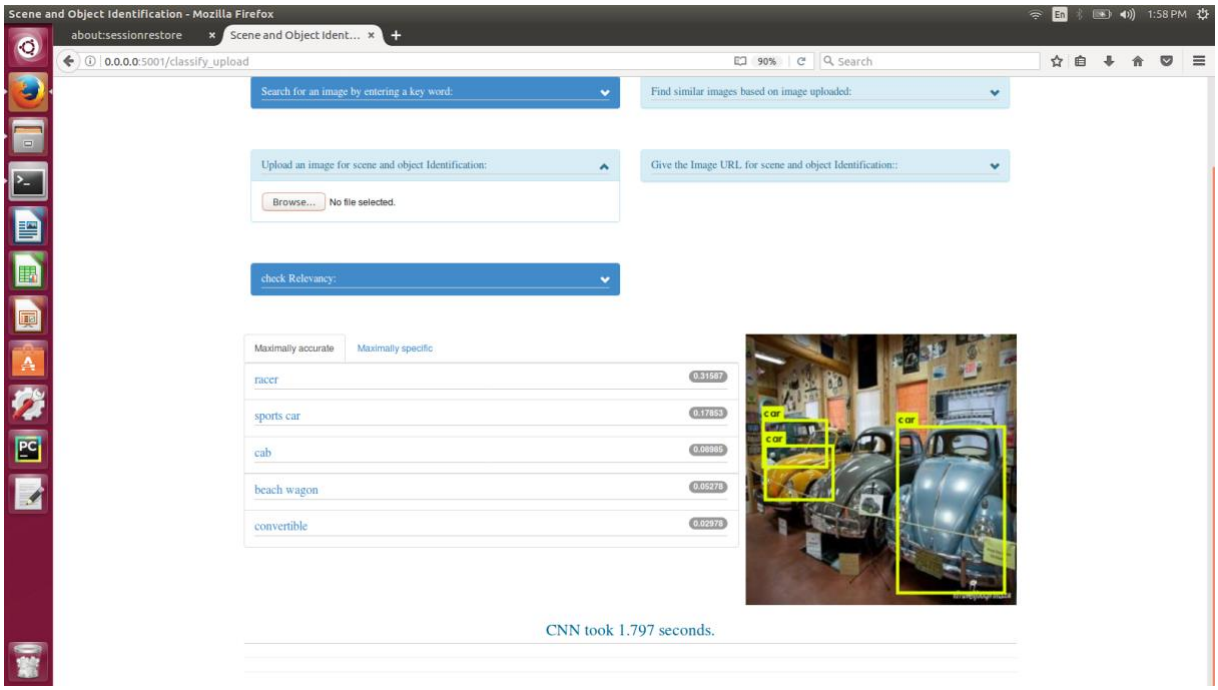


Fig 4.4.10 Detected Scene and Objects in an image

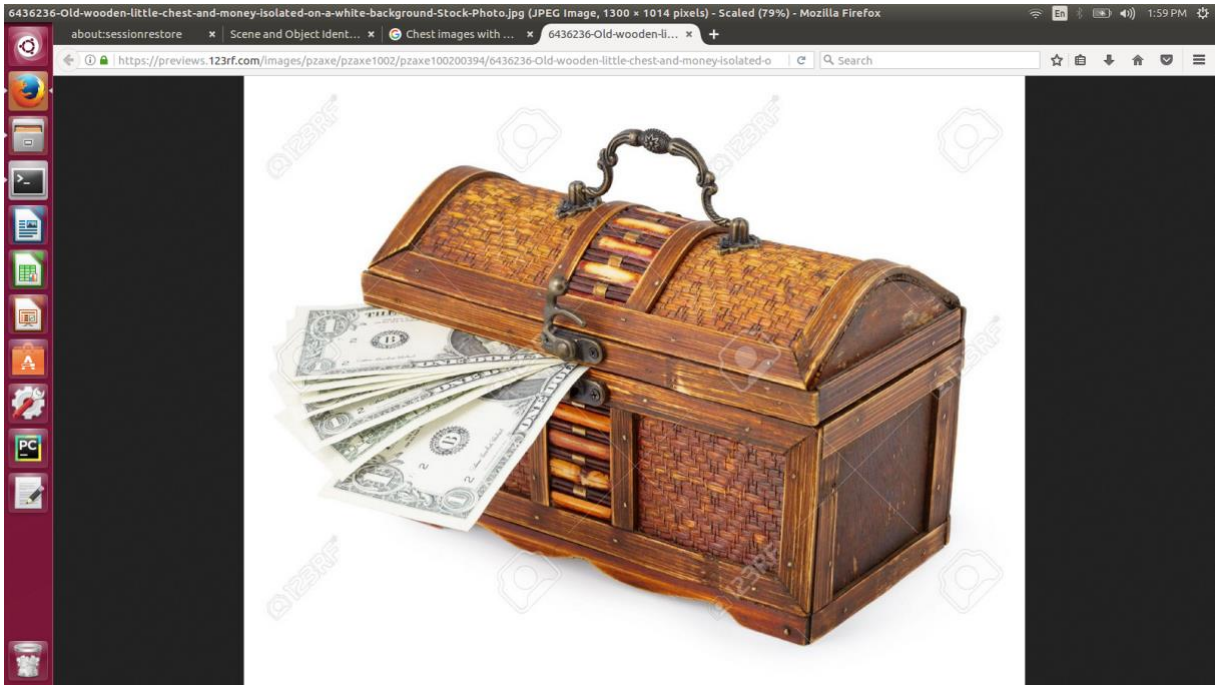


Fig 4.4.11 Image at a path

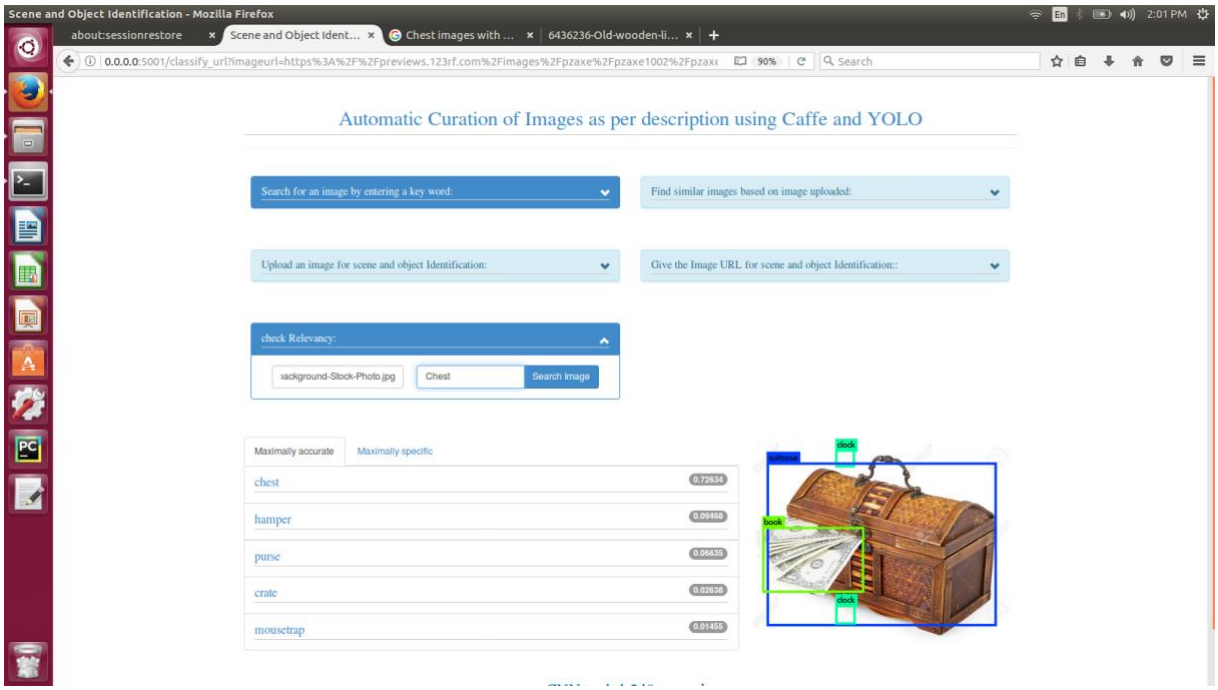


Fig 4.4.12 Entering image path and description to check relevance of image

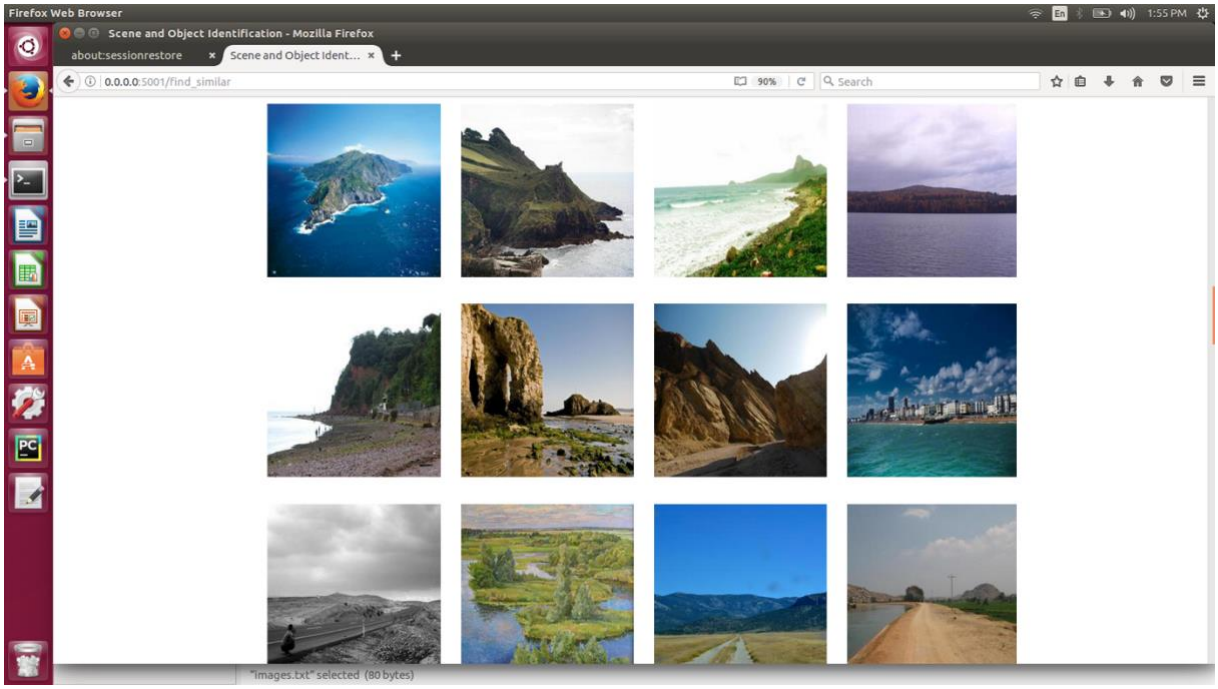


Fig 4.4.13 Finding similar images for a given image of mountain

5. RESULT ANALYSIS AND FUTURE WORK

5.1 Introduction to Testing

Software Testing is the process used to help identify the correctness, completeness, security, and quality of developed computer software. Testing is a process of technical investigation, performed on behalf of stakeholders, that is intended to reveal quality related information about the product with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding errors. Quality is not an absolute; it is value to some person. With that in mind, testing can never completely establish the correctness of arbitrary computer software; testing furnishes a criticism or comparison that compares the state and behaviour of the product against a specification. An important point is that software testing should be distinguished from the separate discipline of Software Quality Assurance (SQA), which encompasses all business process areas, not just testing.

There are many approaches to software testing, but effective testing of complex products is essentially a process of investigation, not merely a matter of creating and following routine procedure. One definition of testing is "the process of questioning a product in order to evaluate it", where the "questions" are operations the tester attempts to execute with the product, and the product answers with its behaviour in reaction to the probing of the tester[citation needed]. Although most of the intellectual processes of testing are nearly identical to that of review or inspection, the word testing is connoted to mean the dynamic analysis of the product—putting the product through its paces. Some of the common quality attributes include capability, reliability, efficiency, portability, maintainability, compatibility and usability. A good test is sometimes described as one which reveals an error; however, more recent thinking suggests that a good test is one which reveals information of interest to someone who matters within the project community. In general, software engineers distinguish software faults from software failures. In case of a failure, the software does not do what the user expects. A fault is a programming error that may or may not actually manifest as a failure. A fault can also be described as an error in the correctness of the semantic of a computer program. A fault will become a failure if the exact computation conditions are met, one of them being that the faulty portion of computer software executes on the CPU. A fault can also turn into a failure when the software is ported to a different hardware platform or a different compiler, or when the software gets extended.

A common practice of software testing is that it is performed by an independent group of testers after the functionality is developed but before it is shipped to the customer. This

practice often results in the testing phase being used as project buffer to compensate for project delays. Another practice is to start software testing at the same moment the project starts and it is a continuous process until the project finishes. Another common practice is for test suites to be developed during technical support escalation procedures. Such tests are then maintained in regression testing suites to ensure that future updates to the software don't repeat any of the known mistakes.

5.2 TEST STRATEGY AND APPROACH

Testing starts with known conditions, uses predefined procedures, and has predictable outcomes; only whether or not the program passes the test is unpredictable. Just like the program, tests should also be planned and designed. So, both tests and systems can be modular. A module is a discrete, well-defined, small component of a system. The smaller, the component, the easier it is to understand; but every component has interfaces with other components and all interfaces can me sources of confusion. Testing should also be organized into modular components. Small, independent test cases have the virtue of easy repeatability. If an error is found by testing, only the small test, not a large component that consists of a sequence of independent tests, need to be rerun to confirm that a test design bug has been fixed. Similarly, if the test has a bug, only that test needs to be changed and not the whole test plan.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Test if the entries of forms are of the correct format or not.
- Test the forms to check if they show an error for invalid entries or not.
- Test if all links direct the user to the correct page or not.
- Test if edits are updated in the database or not.
- Test if each button/menu item is performing its function correctly or not.

5.2.1 Testing Strategies Quality Assurance

The aim of this step is to maintain or to ensure the quality of the system developed. The quality assurance goals in the system life cycle involves

1. Quality Factors Specification: - This was done to determine the factors that lead to high quality of a system. Correctness- The extent to which a program meets System specification.

Reliability – The degree to which a program meets system specification.

Efficiency - The amount of computer resources required by the entire program to perform a function.

Usability – The effort required learning and operating the system.

Maintainability – The ease with which the program errors are located and corrected.

Testability – The effort required to test a program to ensure its correct performance.

Portability – The ease of transporting a program from one hardware configuration to another.

Accuracy - The required precision in input editing, computation and output

Error Tolerance – Error detection and correction versus error avoidance.

Expandability - Ease of adding or expanding existing databases.

Access Controls and Audit – Control of access to the system and the extent to which that access can be audited.

Communication – How useful the input and output of the system are.

2. Software Requirements Specification: - This was done to generate the required documents that provide the technical specification for the design and development of the software.

3. Software Design Specification: - This was done in order to provide the functions and features described in the previous stage.

4. Software Testing and Implementation: - This was done to provide necessary software adjustment for the system to continue to comply with the original specifications.

Quality Assurance is the review of software and related documentation for correctness, accuracy, maintainability, reliability, and expendability. This also includes assurances that the system meets the specifications and requirements for its intended use performance.

5.3 DESIGN OF TEST CASES

Test Case 1

Table 5.3.1 Classify image using an image path

S. No	Steps Involved	Inputs	Expected Result	Actual Result	Test Cases Pass/ Fail
1.	Execute the python code to start application .	Enter the port number in the command Ex :5000	Browser should be opened loading the UI of the application.	Browser is opened with UI of the application	pass
2.	Enter the correct path of the image.	Path of the image should be entered in the field. Ex: Upload an image for scene and object Identification: c:/pic.png/	YOLO and Caffe predictions should be displayed.	YOLO and Caffe predictions are displayed.	pass
3.	Enter the wrong path of the image.	Path of the image should be entered in the field. Ex: Upload an image for scene and object Identification: c:/pic.png/	Should raise an exception saying 'Cannot open uploaded image.'	'Cannot open uploaded image.' Exception is displayed.	pass
4.	Check for maximally specific results.	Click on maximally specific tab to view the results.	Top five maximally specific results are to be displayed.	Top five maximally specific results are displayed.	pass
5.	Check for maximally accurate results	Click on maximally accurate tab to view the results.	Top five maximally accurate results are to be displayed.	Top five maximally accurate results are displayed.	pass
6.	Check for YOLO predictions and image with object detection.	View right side of the screen to view YOLO predictions and image with object detections.	YOLO predictions and image with object detections are to be displayed.	YOLO predictions and image with object detections are displayed.	pass

Test Case 2

Table 5.3.2 Classify image using image URL

S. No	Steps Involved	Inputs	Expected Result	Actual Result	Test Cases Pass/ Fail
1.	Execute the python code to start application.	Enter the port number in the command Ex :5000	Browser should be opened loading the UI of the application.	Browser is opened with UI of the application	pass
2.	Enter the correct URL of the image.	URL of the image should be entered in the field. Ex: Give the Image URL for scene and object Identification: www.google.com/picture.png	YOLO and Caffe predictions should be displayed.	YOLO and Caffe predictions are displayed.	pass
3.	Enter the wrong URL of the image.	URL of the image should be entered in the field. Ex: Give the Image URL for scene and object Identification: www.google.com/picture.png	Should raise an exception saying 'Kindly enter the correct URL of an image. '	'Kindly enter the correct URL of an image. ' Exception is displayed.	pass
4.	Check for maximally specific results.	Click on maximally specific tab to view the results.	Top five maximally specific results are to be displayed.	Top five maximally specific results are displayed.	pass
5.	Check for maximally accurate results	Click on maximally accurate tab to view the results.	Top five maximally accurate results are to be displayed.	Top five maximally accurate results are displayed.	pass
6.	Check for YOLO predictions and image with object detection.	View right side of the screen to view YOLO predictions and image with object detections.	YOLO predictions and image with object detections are to be displayed.	YOLO predictions and image with object detections are displayed.	pass

Test Case 3

Table 5.3.3 Search images using a keyword

S. No	Steps Involved	Inputs	Expected Result	Actual Result	Test Cases Pass/ Fail
1.	Execute the python code to start application.	Enter the port number in the command Ex :5000	Browser should be opened loading the UI of the application.	Browser is opened with UI of the application	pass
2.	Enter the correct Keyword for searching the images	Keyword should be entered. Ex: Search for an image by entering a key word: Car	Keyword related images should be displayed. Ex Car images.	Keyword related images are displayed. Ex Car images.	pass
3.	Enter the wrong Keyword for searching the images	Keyword should be entered. Ex: Search for an image by entering a key word: XXX	Should raise an exception saying ‘ Sorry, we have found no images for your search term ’	‘Sorry, we have found no images for your search term’ Exception is displayed.	pass
4.	Enter multiple key words	Keyword should be entered. Ex: Search for an image by entering a key word: car bike	Should raise an exception saying ‘ Enter single Keyword’	‘Enter single Keyword ‘Exception is displayed.	pass
5.	Enter keyword which is not present in the database.	Keyword should be entered. Ex: Search for an image by entering a key word: Car	Should raise an exception saying ‘ Sorry, we have found no images for your search term ’	‘Sorry, we have found no images for your search term’ Exception is displayed.	pass
6.	Check for YOLO and Caffe predictions and image with object detection.	View right side of the screen to view YOLO predictions and image with object detections.	YOLO and Caffe predictions and image with object detections are to be displayed.	YOLO and Caffe predictions and image with object detections are displayed.	pass

Test Case 4

Table 5.3.4 Search images using a sentential description

S. No	Steps Involved	Inputs	Expected Result	Actual Result	Test Cases Pass/ Fail
1.	Execute the python code to start application.	Enter the port number in the command Ex :5000	Browser should be opened loading the UI of the application.	Browser is opened with UI of the application	pass
2.	Enter the correct sentence for searching the images	Sentence should be entered. Ex: Search for an image by entering a description: Ram with gun	Sentence related images should be displayed. Ex- Ram with Gun images.	Sentence related images are displayed. Ex- Ram with Gun images.	pass
3.	Enter the wrong Sentence with wrong words for searching the images.	Sentence should be entered. Ex: Search for an image by entering a description: ram with gun	Should raise an exception saying ‘ Sorry, we have found no images for your search term ’	‘Sorry, we have found no images for your search term’ Exception is displayed.	pass
4.	Description with multiple keywords should be divided into individual words.	Sentence should be entered. Ex: Search for an image by entering a description: ram with gun	Words should be divided and images are to be displayed accordingly.	Words should be divided and images are displayed accordingly.	pass
5.	Enter description with words which are not present in the database.	Sentence should be entered. Ex: Search for an image by entering a description: ram with gun	Should raise an exception saying ‘ Sorry, we have found no images for your search term ’	‘Sorry, we have found no images for your search term’ Exception is displayed.	pass
6.	Check for YOLO and Caffe predictions and image with object detection.	View right side of the screen to view YOLO predictions and image with object detections.	YOLO and Caffe predictions and image with object detections are to be displayed.	YOLO and Caffe predictions and image with object detections are displayed.	pass

Test Case 5

Table 5.3.5 Search similar images

S. No	Steps Involved	Inputs	Expected Result	Actual Result	Test Cases Pass/ Fail
1.	Execute the python code to start application.	Enter the port number in the command Ex :5000	Browser should be opened loading the UI of the application.	Browser is opened with UI of the application	pass
2.	Enter the path of the image in the field.	Image path should be entered	Images should be displayed which are similar to the given input image.	Images are displayed which are similar to the given input image.	pass
3.	Enter the wrong path of the image.	Path of the image should be entered in the field. Ex: Upload an image for scene and object Identification: c:/pic.png/	Should raise an exception saying 'Cannot open uploaded image.'	'Cannot open uploaded image.' Exception is displayed.	pass
4.	Enter the Image with predictions which are present in the Database	Path of the image should be entered in the field. Ex: Upload an image for scene and object Identification: c:/pic.png/	Given image predictions are to be compared with database predictions and generate similar images	Similar images are displayed	pass
5.	Enter the Image with predictions which are not present in the Database	Path of the image should be entered in the field. Ex: Upload an image for scene and object Identification: c:/pic.png/	As predictions of the images are not present in the database, no images should be displayed.	No images are displayed	pass
6.	Check for YOLO and Caffe predictions and image with object detection.	View right side of the screen to view YOLO predictions and image with object detections.	YOLO and Caffe predictions and image with object detections are to be displayed.	YOLO and Caffe predictions and image with object detections are displayed.	pass

Test Case 6

Table 5.3.6 Check Image Relevance

S. No	Steps Involved	Inputs	Expected Result	Actual Result	Test Cases Pass/ Fail
1.	Execute the python code to start application.	Enter the port number in the command Ex :5000	Browser should be opened loading the UI of the application.	Browser is opened with UI of the application	pass
2.	Enter the correct keyword and path of the image	Imagepath and keyword are to be entered to check relevance.	As Keyword matches with the image, It should print it as a relevant image.	'It is relevant image' message is displayed	pass
3.	Enter the Mismatch keyword and path of the image	Mismatch Imagepath and keyword are to be entered to check relevance.	As Keyword does not match with the image, It should print it as irrelevant image.	'It is irrelevant image' message is displayed	pass
4.	Enter the keyword and Image with predictions which are present in the Database	Keywords and path of the image should be entered in the fields. Ex: Upload an image for scene and object Identification: c:/pic.png/	Given image predictions are to be compared with database predictions and Keyword and display message ' It is relevant image'	'It is relevant image' message is displayed	pass
5.	Enter the keyword and Image with predictions which are not present in the Database	Keywords and Path of the image should be entered in the fields. Ex: Upload an image for scene and object Identification: c:/pic.png/	Given image predictions are to be compared with database predictions and Keyword and display message 'It is irrelevant image'	'It is irrelevant image' message is displayed	pass

REFERENCES

Papers

[1] Dmitry Kislyuk1 , Yuchen Liu1,2 , David Liu1 , Eric Tzeng1,3 , Yushi Jing –'Human Curation and Convnets: Powering Item-to-Item Recommendations on Pinterest', 1 Visual Discovery & Recommendations, Pinterest, Inc. 2 University of California, Los Angeles 3 University of California, Berkeley

[2] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, Trevor Darrell – 'Caffe: Convolutional Architecture for Fast Feature Embedding', SUBMITTED to ACM MULTIMEDIA 2014 OPEN SOURCE SOFTWARE COMPETITION UC Berkeley EECS, Berkeley, CA 94702

[3] Katsuhiko Ishiguro, Akisato Kimura, and Koh Takeuchi – 'Towards Automatic Image Understanding and Mining via Social Curation', 2012 IEEE 12th International Conference on Data Mining, NTT Communication Science Laboratories NTT Corporation, Kyoto, Japan

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi – 'You Only Look Once: Unified, Real-Time Object Detection',

Web References

[www1] https://en.wikipedia.org/wiki/Functional_requirement
[www2] https://www.forbes.com/sites/kalevleetaru/2016/09/19/how-automated-curation-and-chat-bots-could-restore-the-human-touch-to-online-commerce/#156b35b54e7a
[www3]https://docs.google.com/presentation/d/1UeKXVgRvvxg9OUdh_UiC5G71UMscNPlvArsWER41PsU/edit#slide=id.gc2fcdcce7_216_56
[www4]https://www.fastcompany.com/1834177/content-curators-are-new-superheros-web

Text Book References

[1] Object Oriented Design & Patterns – Cay Horstmann, Second Edition

[2] M. J. Christensen, R.H. Thayer, The Project Manager's Guide to Software Engineering's Best Practices, IEEE Computer Society, 2001

[3] R.S. Pressman, Software Engineering – A Practitioner's Approach, Fifth Edition., The McGraw-Hill, 2001

[4] Software Testing techniques – Bariz Beizer, Dreamtech, Second Editio

APPENDIX

Glossary

1. **YOLO:** You Only Look once, it is state of art real time object detection system.

2. **CNN or Convnets:** Convolutional Neural Network, A simple CNN is a sequence of layers and every layer of a CNN transforms one volume of activations to another through a differentiable function

3. **CAFFE:** Convolutional Architecture for Fast Feature Embedding**,** Caffe is a deep learning framework developed by the Berkeley Vision and Learning Center (BVLC). It is written in C++ and has Python and Matlab bindings.

4. **Clustering :** The process of organizing objects into groups whose members are similar in some way

5. **Confidence:** It reflects how confident the model is that the box contains an object.

6. **GPU:** Graphics Processing Unit

7. **Tornado server:** It is a scalable, non-blocking web **server** and web application framework written in Python. It was developed for use by Friend Feed; the company was acquired by Facebook in 2009 and **Tornado** was open-sourced soon after.

8. **Flask:** A lightweight **Python** web framework based on Werkzeug and Jinja 2.