International Journal of Pure and Applied Mathematics
Volume 119 No. 14 2018, 49-57
ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
Special Issue

ijpam.eu

Design and Implementation of Asset Management Using Deep Learning

Pratibha S N [1], Renuka V Tali [2], Shilpa A P [3], Gunasheela T J [4]

[1,2,3,4]*Assistance Professor, ECE, K. S School of Engineering and Management,VTU, Bangalore, INDIA*
**Corresponding author and Email:*
[*1]*pratibha.nedode@gmail.com*
[*2]*renukavtali@gmail.com*
[*3]*shilpaap2012@gmail.com*
[*4]*gunasheela.t.j@kssem.edu.in*

Abstract

This paper deals with the field of the biological science point of view, computer vision aims to come up with computational models of the human visual system. From the engineering point of view, computer vision aims to build autonomous systems which could perform some of the tasks which the human visual system can perform (and even surpass it in many cases). The two goals are of course intimately related. Deep learning is growing rapidly and is surpassing traditional approaches for machine learning since 2012 by a factor of approximately 10%–20% in accuracy. This gives an introduction to Deep Learning and its application in Computer Vision. For computer vision tasks, special architecture of Deep Learning is used and that is called a Convolutional Neural Network. Deep learning (DL) is a subset of Machine learning (ML) in Artificial Intelligence (AI) that has networks which are capable of learning unsupervised from data that is unstructured or unlabelled. Also known as Deep Neural Learning or Deep Neural Network. Hence Deep Learning is more accurate compared to Machine Learning.

Keywords— *Deep Learning (DL), Machine learning (ML), Artificial Intelligence (AI), Neural Network (NN)*

1. Introduction

There are numerous indications that the field of **Artificial intelligence** (AI) is now well established: a specialized journal is entering its third year of publication; the third major biennial conference organized by an ad hoc international council will be held in August; several textbooks and paper collections have been published, and several Computer Science Departments have AI specialties; and some important disciplines whose roots are in AI, such as Pattern Recognition and Symbolic Algebraic Manipulation, have already spun off as independent areas [1], [3].

Machine learning (ML) is a subfield of **Artificial Intelligence** (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people. Two of the most widely adopted machine learning methods is **supervised learning** which trains algorithms based on example input and output data that is labelled by humans, and **unsupervised learning** which provides the algorithm with no labelled data in order to allow it to find structure within its input data [1] [2].

Deep learning (DL) is a form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts. Because the computer gathers knowledge from experience, there is no need for a human computer operator to formally specify all the knowledge that the computer needs. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones; a graph of these hierarchies would be many layers. Deep learning attempts to imitate how the human brain can process light and sound stimuli into vision and hearing [1] [2].

A deep learning architecture is inspired by biological neural networks and consists of multiple layers in an artificial neural network made up of hardware and GPUs. **Deep learning** (DL) is a particular subset of ML methodologies using artificial neural networks (ANN) slightly inspired by the structure of neurons located in the human brain informally, the word deep refers to the presence of many layers in the artificial neural network, but this meaning has changed over time. While 4 years ago, 10 layers were already sufficient to consider a network as deep, today it is more common to consider a network as deep when it has hundreds of layers [3].

Object detection as one of the important applications in the field of computer vision has been the focus of research, and convolution neural network has made great progress in object detection. Object detection is developing from the single object recognition to the multi-object recognition. The meaning of the first is just from an image to identify a single object, it can be said that it is a problem of classification, and the meaning of the later is not only can identify all the objects in an image, including the exact location of the objects. Deep learning has formed a mainstream object recognition algorithm based on R- CNN [1], and these algorithms are refreshing the higher accuracy in a number of famous datasets.

Many of the IEEE papers what I referred, the papers include title as **Object Detection Using Deep Learning**, and so the different kinds of objects or things are referred to as Asset. Hence our project title includes the word **Asset**.

2. Implementation

The requirement for implementing the Asset Management Using Deep Learning as follows

1. Hardware Requirements
2. Software Requirements

1. Hardware Requirements

A. Raspberry PI-3 model
B. Raspberry PI-2 5 MP camera

2. Software Requirements

A. Python
B. Tenserflow
C. OpenCV
D. NumPy
E. Matplotlib
F. COCO

2.1 Overview Architectural Diagram of Design and Implementation of Asset Management Using Deep Learning

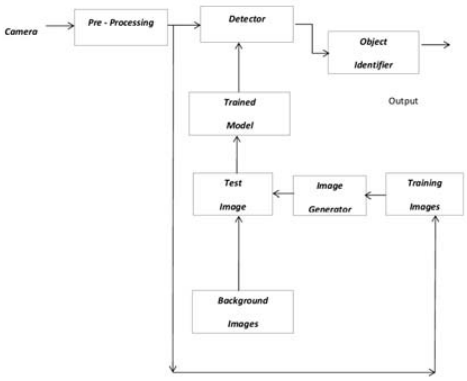

Fig. 1 Architectural Diagram of a Design and Implementation of Asset Management Using Deep Learning

Raspberry Pi, which itself is a minicomputer of a credit card size and is of a very low price. The system is programmed using Python programming language. Both real time face detection and face detection from specific images, i.e. Object Recognition, is carried out and the proposed system is tested across various standard face databases, with and without noise and blurring effects. Efficiency of the system is analyzed by calculating the Face detection rate for each of the database. The results reveal that the proposed system can be used for face detection even from poor quality images and shows excellent performance efficiency.

A. Python

Python is an interpreted high level programming language for general purpose programming. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object oriented, imperative, functional and procedural, and has a large and comprehensive standard library. It supports various kinds of Libraries such as TenserFlow, NumPy, Matpotlib etc. In which the mentioned libraries are useful for image detection, image classification.

B. TenserFlow

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

How TensorFlow Works?

At first, computation in TensorFlow may seem needlessly complicated. But there is a reason for it, because of how TensorFlow treats computation, developing more complicated algorithms is relatively easy.
Here we will introduce the general flow of TensorFlow algorithms.

- **Import or generate datasets** all of our machine-learning algorithms will depend on datasets, we will either generate data or use an outside source of datasets. Sometimes it is better to rely on generated data because we will just want to know the expected outcome.
- **Transform and normalize data** normally, input datasets do not come in the TensorFlow would expect so we need to transform TensorFlow them to the accepted shape. The data is usually not in the correct dimension or type that our algorithms expect. We will have to transform our data before we can use it. Most algorithms also expect normalized data and we will do this here as well.
- **Partition datasets into train, test, and validation sets** we generally want to test our algorithms on different sets that we have trained on. Also, many algorithms require hyper parameter tuning, so we set aside a validation set for determining the best set of hyper parameters. Set algorithm parameters our algorithms usually have a set of parameters that we hold constant throughout the procedure.
- **Initialize variables and place holders** TensorFlow depends on knowing what it can and cannot modify. TensorFlow will modify/adjust the variables and weight/bias during optimization to minimize a loss function. To accomplish this, we feed in data through placeholders. We need to initialize both of these variables and placeholders with size and type, so that TensorFlow knows what to expect. TensorFlow also needs to know the type of data to expect, we will use float32. TensorFlow also provides float64 and float16. Note that the more bytes used for precision results in slower algorithms, but the less we use results in less precision.
- **Define the model structure** after we have the data, and have initialized our variables and placeholders, we have to define the model. This is done by building a computational graph. TensorFlow chooses what operations and values must be the variables and placeholders to arrive at our model outcomes.
- **Declare the loss functions** after defining the model, we must be able to evaluate the output. This is where we declare the loss function. The loss function is very important as it tells us how far off our predictions are from the actual values. The different types of loss functions are explored in greater detail, in the Implementing Back Propagation.
- **Initialize and train the model** now that we have everything in place, we need to create an instance of our graph, feed in the data through the placeholders, and let TensorFlow change the variables to better predict our training data.
- **Evaluate the model** once we have built and trained the model, we should evaluate the model by looking at how well it does with new data through some specified criteria. We evaluate on

the train and test set and these evaluations will allow us to see if the model is under fit or over fit. We will address these in later recipes.

- **Tune hyper parameters** most of the time, we will want to go back and change some of the hyper paramters, based on the model performance. We then repeat the previous steps with different hyper parameters and evaluate the model on the validation set.
- **Deploy/predict new outcomes**: It is also important to know how to make predictions on new, unseen, data. We can do this with all of our models, once we have them trained.

In TensorFlow, we have to set up the data, variables, placeholders, and model before we tell the program to train and change the variables to improve the predictions. TensorFlow accomplishes this through the computational graphs. These computational graphs are a directed graphs with no recursion, which allows for computational parallelism. We create a loss function for TensorFlow to minimize. TensorFlow accomplishes this by modifying the variables in the computational graph. Tensorflow knows how to modify the variables because it keeps track of the computations in the model and automatically computes the gradients for every variable. Because of this, we can see how easy it can be to make changes and try different data sources.

TensorFlow supports the following

- TensorFlow lets you deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device in a very simple way. This way the things can be done very fast.
- TensorFlow lets you express your computation as a data flow graph.
- TensorFlow lets you visualize the graph using the in-built tensorboard. You can inspect and debug the graph very easily.
- TensorFlow gives the best performance with an ability to iterate quickly, train models faster and run more experiments.
- TensorFlow runs on nearly everything: GPUs and CPUs—including mobile and embedded platforms—and even tensor processing units (TPUs), which are specialized hardware to do the tensor math on.

C. OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras.

After processing of images it follows the different types of steps to for tracking they are,

Brightness constancy the pixel intensities of an object in an image do not change between consecutive images.

Temporal regularity the between-frame time is short enough to consider the motion change between images using differentials (used to derive the central equation below).

Spatial consistency Neighbouring pixels have similar motion. In many cases these assumptions break down, but for small motions and short time steps between images it is a good model. For individual points in the image, this equation is under-determined and cannot be solved (one equation with two unknowns in v). By enforcing some spatial consistency, it is possible to obtain solutions though. In the Lucas-Kanade algorithm below we will see how that assumption is used.

OpenCV contains several optical flow implementations, which uses block matching, which uses (both of these currently only in the old cv module), the pyramidal Lucas-Kanade algorithm and finally based on the last one is considered one of the best methods for obtaining dense flow fields. Let's look at an example of using this to find motion vectors in video.

After tracking of objects displaying images and results are performed

2.2 Displaying images and results

OpenCV for image processing and how to show results with OpenCV plotting and window management. It reads an image from file and creates an integral image representation. After reading the image and converting to grey-scale the function creates an image where the value at each pixel is the sum of the intensities above and to the left. This is a very useful trick for quickly evaluating features. Integral images are used in OpenCV's Cascade Classifier which is based on a framework. Before saving the resulting image, we normalize the values to 0 . . . 255 by dividing with the largest value [1] [4].

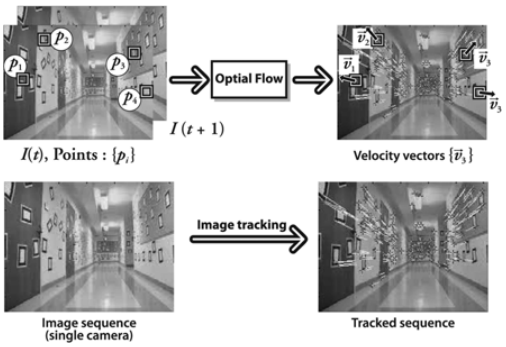

Fig. 2 Image of Optical flow vectors.

D. NumPy

NumPy It is an open source extension module for Python, which provides fast precompiled functions for mathematical and numerical routines. Furthermore, NumPy enriches the programming language Python with powerful data structures for efficient computation of multi-dimensional arrays and matrices. The implementation is even aiming at huge matrices and arrays. Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays.
NumPy is a package popularly used for scientific computing with Python. NumPy contains a number of useful concepts such as array objects (for representing vectors, matrices, images and much more) and linear algebra functions. The array object let's do important operations such as matrix multiplication, transposition, solving equation systems, vector multiplication, and normalization, which

are needed to do things like aligning images, warping images, modelling variations, classifying images, grouping images, and so on.

Using NumPY the various operations can be performed such as

- Array image representation
- Grey-level transforms
- Averaging images

E. Matplotlib

Matplotlib is probably the single most used Python package for 2D-graphics. It provides both a very quick way to visualize data from Python and publication-quality figures in many formats. We are going to explore matplotlib in interactive mode covering most common cases. Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits.

When working with mathematics and plotting graphs or drawing points, lines and curves on images, Matplotlib is a good graphics library with much more powerful features than the plotting available in PIL. Matplotlib produces high quality figures like many of the illustrations. Matplotlib's PyLab interface is the set of functions that allow the user to create plots.

2.3 Plotting images, points and lines

Although it is possible to create bar plots, pie charts, scatter plots, etc., Most importantly, we want to be able to show things like interest points, correspondences and detected objects using points and lines. Note that PyLab uses a coordinate origin at the top left corner as is common for images. The axes are useful for debugging, but if you want a prettier plot, add, this will give a plot.

Fig. 3 Examples of Plotting with Matplotlib. An image with points and a line

2.4 Image contours and histograms

Image contours and image histograms are Visualizing image iso-contours (or iso-contours of other 2D functions) can be very useful. This needs grey-scale images, because the contours need to be taken on a single value for every coordinate [x, y].

Table 1 Basic colour formatting commands for plotting with PyLab

color	
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

As before, the PIL method does conversion to grey-scale. An image histogram is a plot showing the distribution of pixel values. A number of bins are specified for the span of values and each bin gets a count of how many pixels have values in the bin's range. The visualization of the (grey-level) image histogram is done using the function.

The second argument specifies the number of bins to use. Note that the image needs to be flattened first, because takes a one-dimensional array as input. The method converts any array to a one-dimensional array with values taken row-wise.

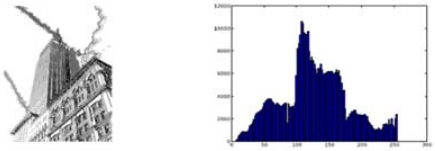

Fig. 4 Examples of visualizing image contours and plotting image histograms

F. Coco

COCO is a large-scale object detection, segmentation, and captioning *dataset*. *COCO* has several features they are,

- Object segmentation,
- Recognition in context,
- Super pixel stuff segmentation,
- 330K *images* (>200K labelled),
- 1.5 million object instances,
- 80 object categories,
- 91 stuff categories,
- Captions per *image.*

Common Objects in Context (COCO) is a database that aims to enable future research for object detection, instance segmentation, image captioning, and person key point's localization. We present a new dataset with the goal of advancing the state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding. This is achieved by gathering images of complex everyday scenes containing common objects in their natural context. Objects are labelled using per-instance segmentations to aid in precise object localization. Our dataset contains photos of 91 objects types that would be easily recognizable. With a total of 2.5 million labelled instances in 328k images, the creation of our dataset drew upon extensive crowd worker involvement via novel user interfaces for category detection, instance spotting and instance segmentation [1], [5].

The various kinds of COCO image data set are

- **Image data** are, Facial recognition, Action recognition, Object detection and recognition, Handwriting and character recognition, Aerial images, other images.
- **Text data** are, Reviews, News articles, Messages, Twitter and tweets, Other text.
- **Sound data** are, Music, Other sounds

3. Design

This chapter describes in detail the development process of the implementation for realizing object recognition with CNNs on Deep Learning. All the developed modules will be explained, how they interact, as well as the general architecture of the solution. Where appropriate, use cases are described for clarification. The chapter's subsections are arranged chronologically in the order in which the respective parts of the implementation were developed.

The entire implementation's source code was written in Python 3.5, or uses Python wrappings in case of most of the code parts directly relying on other frameworks like Tenserflow, OpenCV, MatplotLib, COCO and Numpy.

Besides developing an implementation for object recognition itself on Raspberry Pi 3, a goal for this work is to provide code in a highly modular fashion, so that it can easily be reused as a whole or in parts in other projects or solutions for object recognition.

3.1 Initial Design

The initial design contained the prototypes for the core functionality, namely the forwarding of images to the COCO models for object recognition. Two use cases or modes of operation were identified from the start,

- **Local classification** the image recognition is run directly on the device used for executing the scripts. This requires the COCO framework and the needed pertained CNN models to be installed on the device.
- **Remote classification** the image, which is to be classified, is sent to a remote service running the COCO web demo.

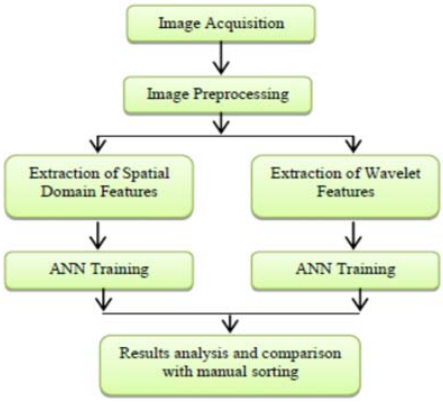

Fig. 5 Shows Flow Chart of Classification of Objects in ANN

3.2 Local Classification

The script local Classification.py provides the method classify image local, which is based on the CNN's basic usage and for calculating the maximally accurate results. The method expects the absolute system path to the directory in which the COCO framework is installed, the relative paths to the pre trained model to be used, its file and the absolute path to the image, which shall be classified.

The script further sets internal parameters for the object recognition and forwards the image to the CNN, which identifies the object on the image. Additionally to the five top labels found, which form the maximally specific result set, five more labels are calculated as mentioned before by "hedging the bets" for the maximally accurate results.

3.3 Locating Multiple Objects

Using the classification modes described the calculate results for one object in a given image, which was forwarded to the CNN. However, if there are multiple objects on the image, it is not possible to directly control, which object is identified, or to get classifications for all objects at once. Multiple passes of the same image through the same CNN will always return identical results. Moreover, no information is returned about the identified object's location in the image.

As such, a way is needed to potentially detect additional objects in the image, and pass those objects, or the image regions that contain them, individually to the CNN. There are many well documented approaches and algorithms for detecting multiple objects and their locations in images, and discussing all of them would be out of scope for the current work. Therefore the focus will largely remain on concrete implementations of those algorithms, which were more closely considered for incorporation into the current object recognition pipeline.

The best method to detect a multiple objects in a image is Sliding Window.

Sliding Window

One of the more naive approaches is to use a simple sliding window implementation, that is, to move a rectangular window (with a size smaller than the target image) in a certain pattern over the image as shown in figure and applying one of the implemented image classifiers to each of the segments obtained this way.

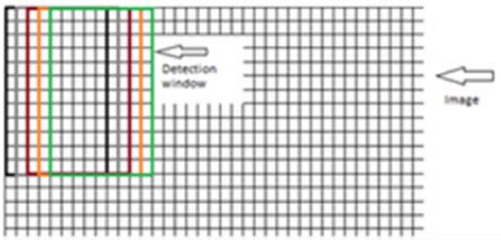

Fig. 6 The detection window is moved over the target image in strides of one image source

While this approach is relatively simple to implement as described, it has numerous disadvantages. Obviously, depending on the size of the window and the exact pattern and stride used to slide it over the

target image, there might be lots of segments, which have to be classified by the CNN. Apart from being prohibitively wasteful with regard to the performance, the same object could be detected multiple times on different segments, or not at all, depending on how it was fitted into the segments. This is especially the case if one single large object covers most of the image.

For example, each segment could only contain a small part of the object, resulting in the CNN being unable to classify it at all, due to having not enough information provided at once by each segment seen separately.

On the other hand, if the detection window or the strides are too big, there might be several objects in one segment, leading back to the initial problem. While there are refinements for the sliding window approach alleviating these points, a more computationally efficient method was sought.

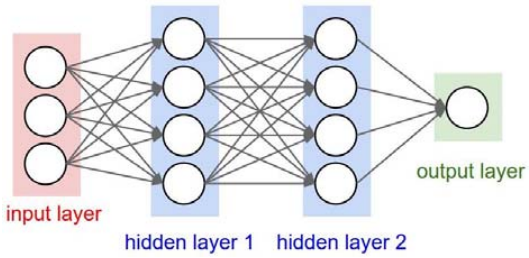

Fig.7 Shows Convolutional Neural Network

3.4 Image Handling and Processing in Python

This chapter is an introduction to handling and processing images. It explains the central Python packages you will need for working with images. This chapter introduces the basic tools for reading images, converting and scaling images, computing derivatives, plotting or saving results, and so on.

Python Imaging Library

The Python Imaging Library (PIL) provides general image handling and lots of useful Basic image operations like resizing, cropping, rotating, color conversion and much More. With PIL you can read images from most formats and write to the most common Ones. The most important module is the Image module. The most important class in the Python Imaging Library is the Image class, defined in the module with the same name. You can create instances of this class in several ways; either by loading images from files, processing other images, or creating images from scratches. To load an image from a file, use the open function in the Image module.

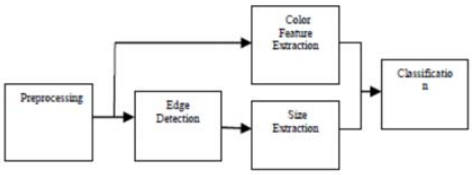

Fig. 8 Shows How Images are processed

If successful, this function returns an Image object. You can now use instance attributes to examine the file contents. The format attribute identifies the source of an image. If the image was not read from a file, it is set to none. The size attribute is a 2-tuple containing width and height (in pixels). The mode attribute defines the number and names of the bands in the image, and also the pixel type and depth. Common modes are "L" (luminance) for grey scale images, "RGB" for true color images and "CMYK" for pre-press images. If the file cannot be opened, an IO Error exception is raised. Once you have an instance of the Image class, you can use the methods defined by this class to process and manipulate the image. The basic tools for processing an image are,

- Reading and Writing images,
- Converting between modes,
- Image enhancement,
- Filters.

COCO object recognition datasets have

1. Image classification,
2. Object bounding or box localization,
3. Semantic pixel-level segmentation,

a. Image classification

The task of object classification requires binary labels indicating whether objects are present in an image, early datasets of this type comprised images containing a single object with blank backgrounds, such as the MNIST handwritten digits or COIL household objects. Caltech 101 and Caltech 256 marked the transition to more realistic object images retrieved from the internet while also increasing the number of object categories to 101 and 256, respectively. Popular datasets in the machine learning community due to the larger number of training examples, CIFAR-10 and CIFAR-100 offered 10 and 100 categories from a dataset of tiny 32_32 images. While these datasets contained up to 60,000 images and hundreds of categories, they still only captured a small fraction of our visual world. Recently, Image-Net made a striking departure from the incremental increase in dataset sizes. They proposed the creation of a dataset containing 22k categories with 500-1000 images each. Unlike previous datasets containing entry-level categories, such as "dog" or "chair," like, Image-Net used the Word-Net Hierarchy to obtain both entry-level and fine-grained categories. Currently, the Image-Net dataset contains over 14 million labelled images and has enabled significant advances in image classification.

b. Object detection

Detecting an object entails both stating that an object belonging to a specified class is present, and localizing it in the image. The location of an object is typically represented by a bounding box. Early algorithms focused on face detection using various ad hoc datasets. Later, more realistic and challenging face detection datasets were created. Another popular challenge is the detection of pedestrians for which several datasets have been created. The Caltech Pedestrian Dataset contains 350,000 labelled instances with bounding boxes. For the detection of basic object categories, a multiyear effort from 2005 to 2012 was devoted to the creation and maintenance of a series of benchmark datasets that were widely adopted. The PASCAL VOC datasets contained 20 object categories spread over 11,000 images. Over 27,000 object instance bounding boxes were labelled, of which almost 7,000 had detailed segmentations. Recently, a detection challenge has been created from 200 object categories using a subset of 400,000 images from Image-Net. An impressive 350,000 objects have been labelled using bounding boxes. Since the detection of many objects such as sunglasses, Cell phones or chairs is highly dependent on contextual information, it is important that detection datasets contain objects in their natural environments. In our dataset we strive to collect images rich in contextual information. The use of bounding boxes

also limits the accuracy for which detection algorithms may be evaluated. We propose the use of fully segmented instances to enable more accurate detector evaluation.

c. Semantic Pixel-level Segmentation

The task of labelling semantic objects in a scene requires that each pixel of an image be labelled as belonging to a category, such as sky, chair, floor, street, etc. In contrast to the detection task, individual instances of objects do not need to b segmented. This enables the labelling of objects for which individual instances are hard to define, such as grass, streets, or walls. Datasets exist for both indoor and outdoor scenes. Some datasets also include depth information. Similar to semantic scene labelling, our goal is to measure the pixel-wise accuracy of object labels. However, we also aim to distinguish between individual instances of an object, which requires a solid understanding of each object's extent. A novel dataset that combines many of the properties of both object detection and semantic scene labelling datasets is the SUN dataset for scene understanding. SUN contains 908 scene categories from the Word-Net dictionary with segmented objects. The 3,819 object categories span those common to object detection datasets (person, chair, car) and to semantic scene labelling (wall, sky, floor). Since the dataset was collected by finding images depicting various scene types, the number of instances per object category exhibits the long tail phenomenon. That is, a few categories have a large number of instances (wall: 20,213, window: 16,080, chair: 7,971) while most have a relatively modest number of instances (boat: 349, airplane: 179, floor lamp: 276). In our dataset, we ensure that each object category has a significant number of instances.

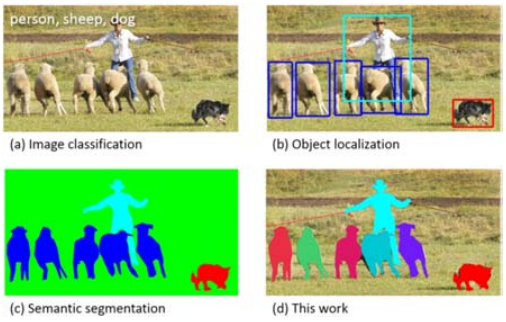

Fig. 9 COCO object recognition process of an image

4. Results and Discussions

The behavior of the image classification models, we evaluate the performance of DNNs on negative images of the training data. A negative is referred to an image with reversed brightness, i.e., the lightest parts appear the darkest and the darkest parts appear lightest. These complemented images are often easily recognizable by humans. We show when testing on negative images, the accuracy of DNNs drops to the level of the random classification, i.e., the network maps the inputs randomly to one of the output classes. This shows that the DNNs, which are merely trained on raw data, do not learn the structures or semantics of the objects and cannot generalize the concepts.

Example -1

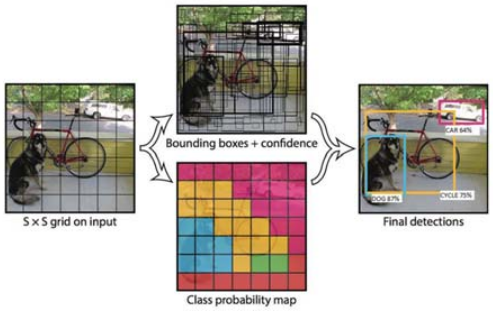

Fig. 10 Methodology of Object Detection Using Deep Learning

- The given image is converted to (S x S) grid view in which foreground, background and many other hidden layers get extracted. In which the images are converted to grey level values.
- The grey level values are considered and the bounding boxes are created. Further probabilities of different kinds of objects are validated with test image. If the probability is greater than or equal to 0.6 then the image is considered to be a matched with an test image.
- After the validation with test image, the probability values are considered and if it is recognized the object is named.

Example -2

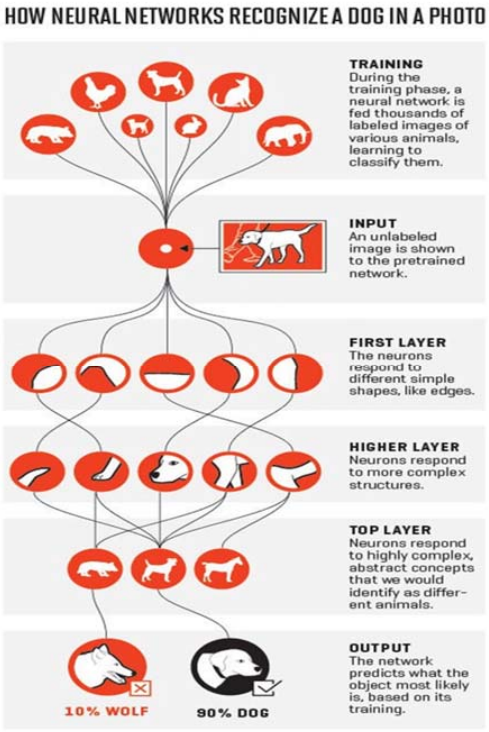

Fig.11 Methodology of Object Detection Step by Step Using Deep Learning

5. Conclusion

In this paper, we present a multi-model Neural Network that automatically learns to describe the content of images. Our model first extracts the information of objects and their spatial locations in an image, and then a deep recurrent neural network (RNN) based on LSTM units with attention mechanism generates a description sentence. Each word of the description is automatically aligned to different objects in the input image when it is generated. The proposed model is more optimized compared to other benchmark algorithms on the ground that its implementation is totally made on human visual system. We hope that this paper will serve as a reference guide for re- searchers to facilitate the design and implementation image captioning.

Hence, Deep learning can be a time and cost efficient way to determine malignancy associated changes. It can provide better result for recognition of different kinds of objects in given premises.

Acknowledgement: This work is carried out under Karnataka State Council for Science and Technology (KSCST) funded Project Sponsorship.

References

[1] Application of Deep Learning in Object Detection Xinyi Zhou1, Wei Gong2, WenLong Fu3, Feng tong Du4 1,2Information Engineering School, Communication University of China, CUC 3,4Neuroscience and Intelligent Media Institute, Communication University of China Beijing, China

[2] Artificial Intelligence, Machine Learning and Deep Learning Pariwat Ongsulee Department of Computer Science Faculty of Science, Siam University Bangkok, Thailand

[3] Krizhevsky I. Sutskever G. Hinton "ImageNet classification with deep convolutional neural networks" NIPS 2012.

[4] K. Simonyan A. Zisserman "Very deep convolutional networks for large-scale image recognition" ICLR2015.

[5] K. He X. Zhang S. Ren J. Sun "Deep residual learning for image recognition"CVPR2016.

[6] R. Girshick J. Donahue T. Darrell J. Malik "Rich feature hierarchies for accurate object detection and semantic segmentation" CVPR 2014.

[7] J. Deng W. Dong R. Socher L.-J. Li K. Li L. FeiFei ImageNet: A large-scale hierarchical image database. In CVPR 2009.

[8] J. Deng A. Berg S. Satheesh H. Su A. Khosla L. FeiFei ImageNet Large Scale Visual Recognition Competition 2012.

[9] M. Everingham L. Van Gool C. K. I. Williams J. Winn A. Zisserman The PASCAL Visual Object Classes Challeng.