B.TECH. PROJECT REPORT

on

Object Detection in Real-Time Systems using Convolutional Neural Networks with Deep Learning

by

Avnish Bhagwate

Discipline of Computer Science and Engineering

Indian Institute of Technology Indore

December 2016

Object Detection in Real-Time Systems using Convolutional Neural Networks with Deep Learning

A PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

Avnish Bhagwate

Under the guidance of

Dr. Aruna Tiwari, Assistant Professor, IIT Indore

and

Dr. Kapil Ahuja, HOD, Department of Computer Science and Engineering, IIT Indore

Indian Institute of Technology Indore

2016-2017

Contents

List of Figures

List of Tables

CANDIDATE'S DECLARATION

We hereby declare that the project entitled "**Object Detection in Real-Time Systems using Convolutional Neural Networks with Deep Learning**" submitted in partial fulfillment for the award of the degree of Bachelor of Technology in Computer Science and Engineering completed under the supervision of **Dr. Aruna Tiwari, Assistant Professor and Dr. Kapil Ahuja, HOD, Department of Computer Science and Engineering, IIT Indore**, is an authentic work.

Further, we declare that we have not submitted this work for the award of any other degree elsewhere.

Avnish Bhagwate

130001007

CERTIFICATE by BTP Guides

It is certified that the above statement made by the students is correct to the best of my/our knowledge.

Dr. Aruna Tiwari, Assistant Professor, IIT Indore

Dr. Kapil Ahuja, HOD, Department of Computer Science and Engineering, IIT Indore

Acknowlegments

I wish to thank Dr. Aruna Tiwari, Assistant Professor of Computer Science, IIT Indore and Dr. Kapil Ahuja, HOD and Assistant Professor of Computer Science, IIT Indore for providing us support and invaluable guidance over our work on the project. Dr. Kapil's unwavering optimism and cheerfulness kept us striving to work hard. Dr. Aruna's confidence in us was a driving force for our interest in the research. I'm immensely grateful for their direction and willingness to spend extra time with us over discussions and meetings.

Our work could perhaps not have been possible without the external support provided by Mr. Shailendra Verma from the Computer Laboratory for aiding in the procurement and installation of the hardware required. I am profoundly grateful for his help.

I would like to wholeheartedly thank my project partners Mr. Amey Ambade and Ms. Parinita Dudhagundi, whose combined efforts and individual contributions were imperative to the successful completion of the project in time. Our amiable collaboration proved to be incredibly fruitful.

Finally, I would like to extend my gratitude to all the people who were directly or indirectly helpful to us in achieving our goal.

Avnish Bhagwate
B.Tech. IV Year
Discipline of Computer Science and Engineering
IIT Indore

Preface

This report on ‘‘Object Detection in Real-Time Systems using Convolutional Neural Networks with Deep Learning’’ is prepared under the guidance of Dr. Aruna Tiwari and Dr. Kapil Ahuja at IIT Indore during the Autumn Semester of academic year 2016-17.

Through this report we have tried to present a detailed research and analysis of Convolutional Neural Networks and their application in Object Detection which might be useful in detecting suspicious activity in videos.

We have tried to the best of our abilities and knowledge to explain the content in a cogent manner. We have added visual content in an attempt to make the report more illustrative and understandable.

Avnish Bhagwate
B.Tech. IV Year
Discipline of Computer Science and Engineering
IIT Indore

Abstract

In daily loops of life, for the purpose of safety and security, surveillance cameras have proven to be useful. The scenario, when looked from a practical point of view requires a person to monitor the feed and look for suspicious activity. Through this report we have tried to present our experiments in designing a system which helps at reducing the effort of monitoring the surveillance separately.

In the area of Machine Learning, since we have to work on videos i.e., images, Convolutional Neural Networks (CNNs) have proved to be very useful in that area. We have used the concept and structure of CNNs to use it for object detection, starting with images and eventually to videos. Exploiting the idea that a video is a set of frames, we have attempted at scaling the object detection system from images to videos.

We have used CNNs and designed a network, training it on a CPU as well as a high-end GPU and coming to the conclusion that using a GPU saves time on training. Using high end architecture to save time, the algorithm has been trained to give dependable results, taking considerable less time. Having realized that a high-end architecture saves times on training, the network has been expanded and then implemented to detect objects in a video. Taking frames at regular intervals and feeding to the trained network gave us prominent results. Since the algorithm doesn't need much computation after training, the future idea of implementation in a real-time system can be made a reality.

1 Introduction

1.1 Background and Recent Research

Surveillance cameras are video cameras used for the purpose of observing an area. They are often connected to a recording device, and may be monitored by a law enforcement officer or a security guard. Cameras and recording equipment are relatively expensive and usually require human personnel to monitor camera footage to detect any unusual activity or malpractice.

Governments around the world are known to have developed and actively deployed highly trained programs for intrusion detection [16]. A majority of such programs are inaccessible to the wider public, are exclusively proprietary and have an ostensibly large financial dependency [15]. Our approach would be using significantly lesser resources and would be open-sourced.

We intend to develop a system which captures the environment through a video camera, detects any unwanted and malicious activities and alerts personnel thereby reducing the effort and cost required to constantly monitor footage.

1.2 Literature Survey

Convolutional Neural Networks (CNNs) [14] have been demonstrated as an effective class of models for understanding image content [6, 9, 13]. Compared to images, there has been little work on application of CNNs to videos because of (a) the non-availability of large-scale video data sets required for training [2, 3] and (b) the high computational resource requirement for processing videos.

Most of the object detection using CNNs has been made possible due to the extended use of large scale public image repositories like ImageNet [7], and high performance computing architectures like GPUs. Multiple case studies highlight the innovative CNN models developed fairly recently to compete in the ImageNet Large Scale Visual Recognition Competition (ILSVRC) [18]. These widely varying models borrow from each other to improve accuracy.

The AlexNet [13] model (runner-up, ILSVRC 2012) is a derivative of the more rudimentary LeNet [14] architecture, in that it made use of more convolutional (CONV) layers and introduced the concept of extended hyper parameters. The VG-GNet [23] model (runner-up, ILSVRC 2014) focuses on the importance of increasing the number of CONV layers for achieving better accuracy at the cost of additional memory and significantly larger set of parameters.

CNNs have not been extensively applied to videos, and the training done for images has traditionally required the use of very large datasets. We intend to compare a CNN model's computation on a CPU and the GPU. Additionally, we intend to start training with a comparatively small dataset to assess the applicability of our model to videos in general, and later in real-time environments. Our model uses concepts borrows from VGGNet and AlexNet. Our network has smaller depth in order to reduce computation time with hyper parameters associated with each CONV layer.

2 Convolutional Neural Networks

2.1 Design

An Artificial Neural Network is a network inspired by biological neural networks. It is capable of learning particular aspects of the input for predictions after being trained for datasets specific to requirements. Convolutional Neural Networks are a class of ANNs specifically designed for processing an image because of the high computational complexity involved.

We feed test images to a CNN as input, and the network 'learns' the weights of its neurons accordingly, thus becoming better equipped at testing samples of videos with every additional test input. This is called *training and testing.* The intermediate layers in the network adjust their neuron weights to reflect a better prediction for a future input. Each layer is comprised of a two dimensional array of neurons that we call a depth slice.

A CNN transforms the original image, layer by layer from original pixel values

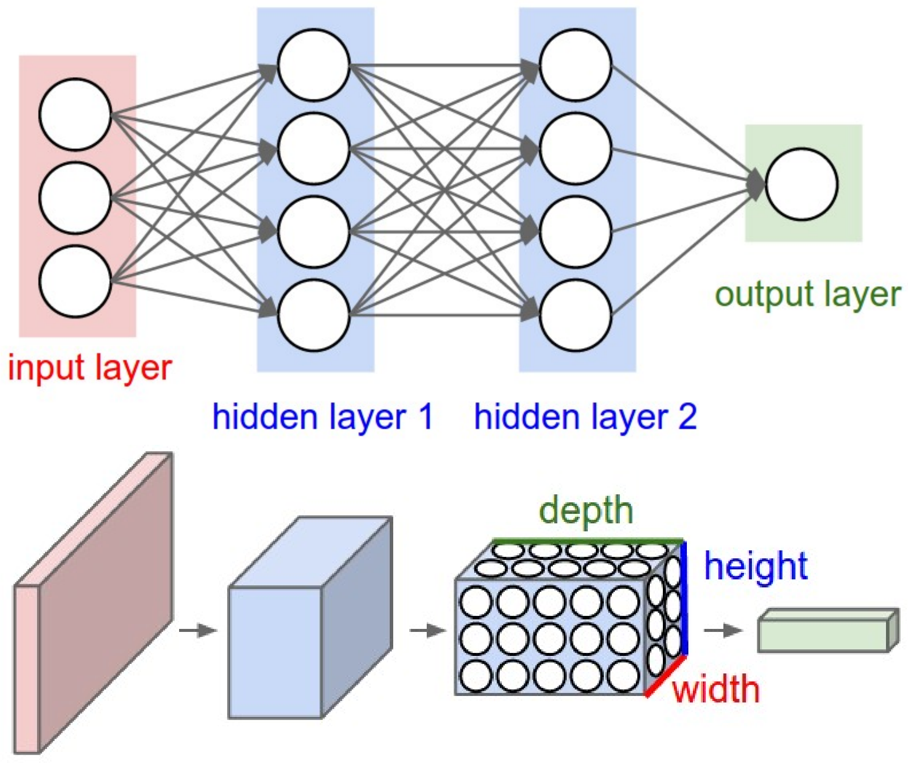


Figure 2.1: Comparison of a regular ANN and a Convolutional NN [1]

to final class values. The input is considered to be three dimensional, so the colors (RGB) represent a new third dimension in addition to the width and the height of the image [19].

2.2 Layers

2.2.1 Convolutional Layer

A CONV layer has a three-dimensional structure, i.e., the neurons in the CONV layer are arranged in three dimensions : width, height and depth. It accepts a 3D input volume and transforms it to a 3D output volume using an activation function. The CONV layer has filters that look at certain regions of the input, the sum of which make up the input. Each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter(weights) and the input and producing a 2-dimensional activation map.

The CONV Layer ensures that the learned filters produce the strongest response to a spatially local input pattern.

The CONV layer makes use of the following parameters:

- *Width (W)* : The Width of the 3D volume of the CONV layer.
- *Height (H)* : The height of the 3D volume of the CONV layer. The width and height of the CONV layer are usually equal.
- *Depth (D)* : The third dimension of the CONV layer, i.e., the number of filters to be used. Each filter corresponds to, and activates on, the presence of a particular feature of interest (horizontal lines, blobs of color, etc.). The 3D CONV layer can be divided into D depth slices having dimensions W and H. The depth assumed for the input is 3, since there are three channels: R, G and B to the image. Generally, the D¿3 for CONV layers.
- *Filter / Receptive Field Size(F)* : The spatial extent of connectivity of each neuron to the input volume, or, more simply, the size of the moving filter.
- *Stride (S)* : The number of pixels skipped when a filter slides over the input. The stride is usually 2, 3 or similar.
- *Zero-padding (P)* : The number of zero values to add around the border of the input volume. This padding is done to preserve the spatial size of the output volume.

  In general, the zero padding is set to:

$$\frac{(F-1)}{2}$$

The spatial size of the output volume can be seen as a relationship between input volume size (W), receptive field size (F), stride (S) and zero-padding (P) which turns out to be equivalent to:

$$\frac{(W-F+2P)}{S}+1$$

For example, for a 7x7 input and a 3x3 filter size, with S=1 and P=0 gives the output volume to be 5x5.

Use of zero padding can be demonstrated by another example: say the input dimension is 5, and we need the output dimension to be equal to the imput dimension. In this case, the absence of zero padding gives the output dimension as 3 (keeping all other parameters same). Therefore, using a padding of 1 ensures the output dimension is 5.

Care should be taken that the padding value should not be such that the output volume dimension calculated manually turns out to be a fractional value.

Suppose we have a convolutional layer of size with WxH as 40x40 and depth as 50 and the filter size is 3x3. For an input image each neuron has 3*3*3=27 weights (assuming filter size is 3x3) and one bias and the total number of neurons are 40*40*50=80,000. Together the number of parameters add up to 80,000*27=2,160,000, which is just for one CONV layer. This number is really high since we have to deal with more layers like this.

We can reduce the amount of parameters by making one reasonable assumption: That if one feature is used to compute at some position (x,y), then it should also be useful to compute at a different position (w,z). Therefore denoting a single 2-dimensional slice of depth as a depth slice (e.g. from our previous example the volume of size [40x40x50] has 50 depth slices, each of size [40x40]), we are going to constrain the neurons in each depth slice to use the same weights and bias. With this parameter sharing scheme, all 40*40 neurons in each depth slice will be using the same parameters, thus saving on the variables.

2.2.2 Pooling Layer

A pooling layer reduces the spatial size of the input 3D volume, using a function (MAX, MIN, AVG) operating on every 2D region in a depth slice of the input. This reduces the amount of parameters and computation in the network, and ultimately

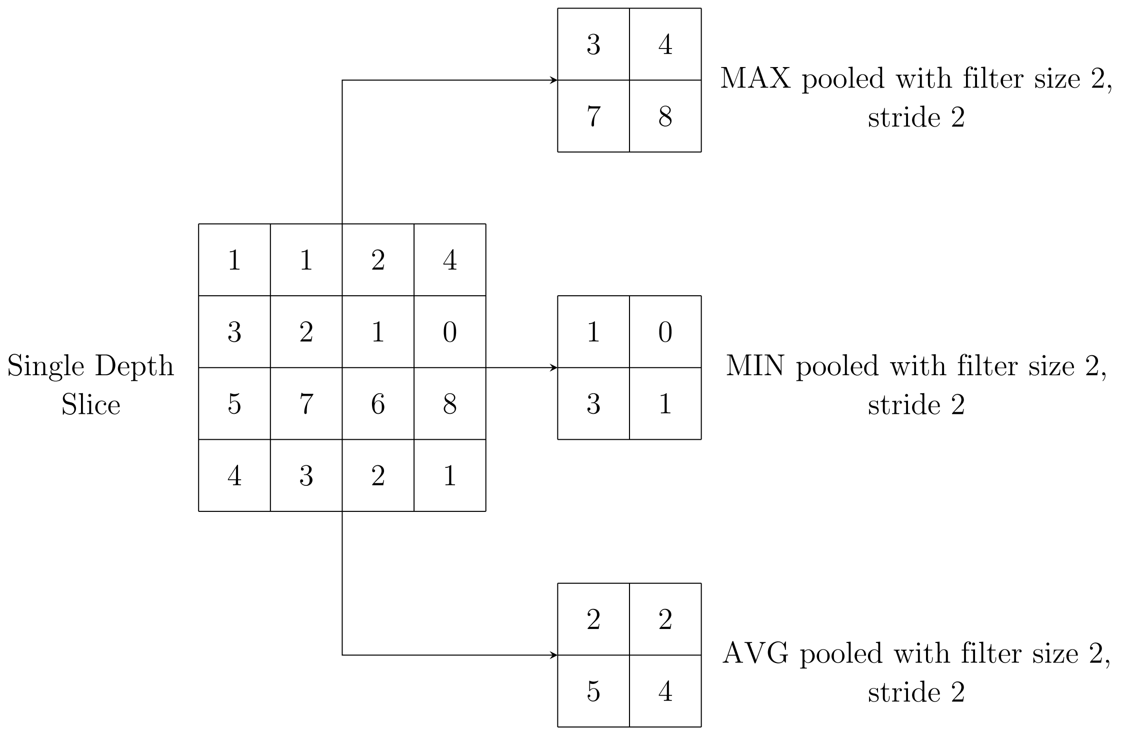

MAX pooled with filter size 2, stride 2
3 4
7 8
Single Depth Slice
1 1 2 4
3 2 1 0
5 7 6 8
4 3 2 1
1 0
3 1
MIN pooled with filter size 2, stride 2
2 2
5 4
AVG pooled with filter size 2, stride 2

Figure 2.2: Pooling : MAX, MIN, AVG

helps control overfitting. The pooling layer performs a joining operation using MAX, MIN or AVG function on each depth slice, reducing its spatial dimensions.

Types of pooling:

- *Max Pooling* : Function for reduction is MAX. The maximum of all values is set as the representative of these values.
- *Min Pooling* : Function for reduction is MIN. The minimum of all values is set as the representative of these values.
- *Average Pooling* : Function for reduction is AVG. The average of all values is set as the representative of these values.
- *Overlapping Pooling* : The stride of the filter motion is less than the complete pooling window.

2.2.3 Rectified Linear Unit (ReLU) Layer

A Rectified Linear Unit or ReLU layer applies an elementwise activation function, like

$$max(0, x)$$

Other functions that can be used are *sigmoid* and *tanh* functions. The MAX function is found to be considerably faster for training purposes [13].

2.2.4 Fully Connected (Dense) Layer

The FC layer in which all neurons are connected to every input activation such that applying simple matrix multiplication gives a numeric score array as output. This layer is almost identical to those in a regular neural network. The high level reasoning is done via this layer, which decides the class scores for an input. The class scores are the numeric identities associated with the classes that the network is being trained to learn.

This layer requires the use of the following parameters:

- *Dense* : The number of neurons in the FC layer. These are typically equal either to powers of 2 or the number of classes for better computation.
- *Activation function* : Softmax is usually used as the activation function for the dense layer. Softmax function is used to normalize data in the interval $[0, 1]$ which is defined as

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for \quad j = 1, ..., K$$

All the above mentioned hyper parameters are subject to additional modifications. The performance measures for the ConvNet model are: time taken for the algorithm to self minimize error using back propagation for stabilizing neuron weights, storage space required for weights and the final output size.

The various features of the network such as number of hidden layers, number of inputs, convolution dimension, max-pooling layers, etc. can be optimized to maximize the efficiency of the training phase. Additionally, optimization algorithms can be applied for selection of frames from the video for testing their quality and color [11].

The CNN is thus trained to 'learn' the features marked in the training images to give a desired output, in our case, the presence of a suspicious object, when run in real-time [20].

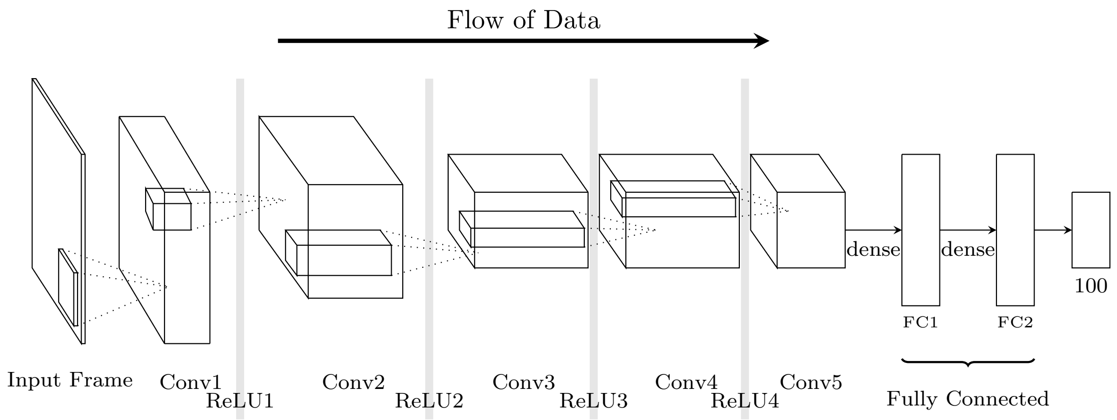


Figure 2.3: Convolutional Neural Network : A Closer Look

Figure 2.3 gives a clearer picture as to how a complete network containing convolutional layers can be visualized.

3 CNN for Images

3.1 AlexNet

AlexNet is the first work to popularize CNNs in Computer Vision developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton [13]. The AlexNet was the runner-up in the ImageNet ILSVRC challenge 2012. The network featured Convolutional Layers stacked on top of each other (The conventional method was to have single CONV layer followed by POOL layer).

AlexNet introduces us to ideas like the usage of Rectified Linear Units (ReLUs) as activation functions for faster training, training on mutiple GPUs, overlapping pooling and concept of dropout to reduce overfitting.

3.2 VGGNet

VGGNet is the network designed by Karen Simonyan and Andrew Zisserman [23], which was the runner-up in ILSVRC 2014. The main contribution of the network was in showing that good performance succeeds with the depth of the network as a critical component. VGGNet has been trained and tested with increasing number of layers and has shown that state-of-the-art accuracy can be obtained by increasing depth of the network.

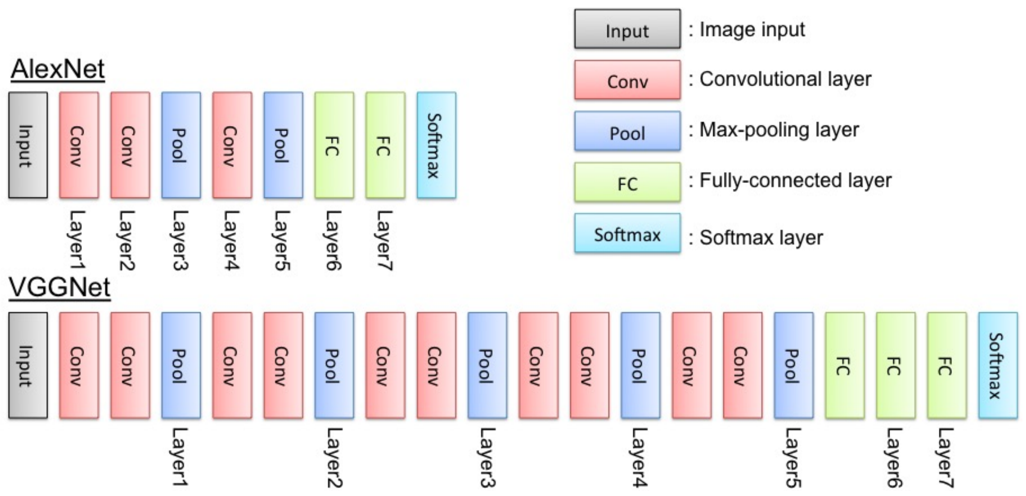


Figure 3.1: A quantified comparison : AlexNet and VGGNet

Figure 3.1 shows the difference between AlexNet and VGGNet in terms of their respective sizes; it can be seen that AlexNet has applied the idea of stacking CONV layers in the first section of the network, and VGGNet has an expanded structure by the increased depth of the network.

3.3 Proposed Model

Our model borrows the idea of hyperparameters from AlexNet and the concept of depth as an important aspect from VGGNet to first create a small network to be run on a CPU and then expanding on a high-end GPU which would be later implemented for videos by exploiting the idea that a video can be treated as a set of frames. The implementation and experimentation will be discussed in further chapters.

4 Analysis

4.1 Problem

Identifying every other object in every frame of a video is a huge task in practicality, usually requiring an exceptionally large dataset. Therefore we would be confining our scope to a controlled environment where specificity is paramount, thus helping us focus on a much clearer target. The controlled environment [17] may be an indoor surveillance area, like a parking lot, or a bank office or an outdoor location, like a protected forest area, or a state highway. Thus, by narrowing our scope we save on computation by training our algorithm for only those objects that can pose an immediate threat to the surroundings.

We would begin by training a small network on an established dataset. Comparisons would be done with respect to time in case of CPU and a high-end architecture. Subsequently the size of the network would be increased and then determined whether the network could be scaled to videos and finally in real-time surveillance.

4.2 Objectives

We have achieved the following by the end of our project:

- Studied the different layers and parameters involved in designing a convolutional neural network such as convolution layer, max-pooling layer, fully connected layer, dropout, activation function, etc.
- Designed an algorithm to detect objects in images by choosing the optimal parameter values and number of layers in order to maximize accuracy and minimize loss.
- Collected the dataset of images required for training and testing the algorithm.
- Compared the execution time of the model on CPU and a high-end GPU.
- Trained and tested the algorithm on GPUs to compare its real-time performance.
- Extended the designed algorithm to videos by exploiting its characteristic that it is made up of sequential collection of frames.

5 Design

The basic structure of feature training for images from the dataset follows the cycle represented in Figure 5.1.

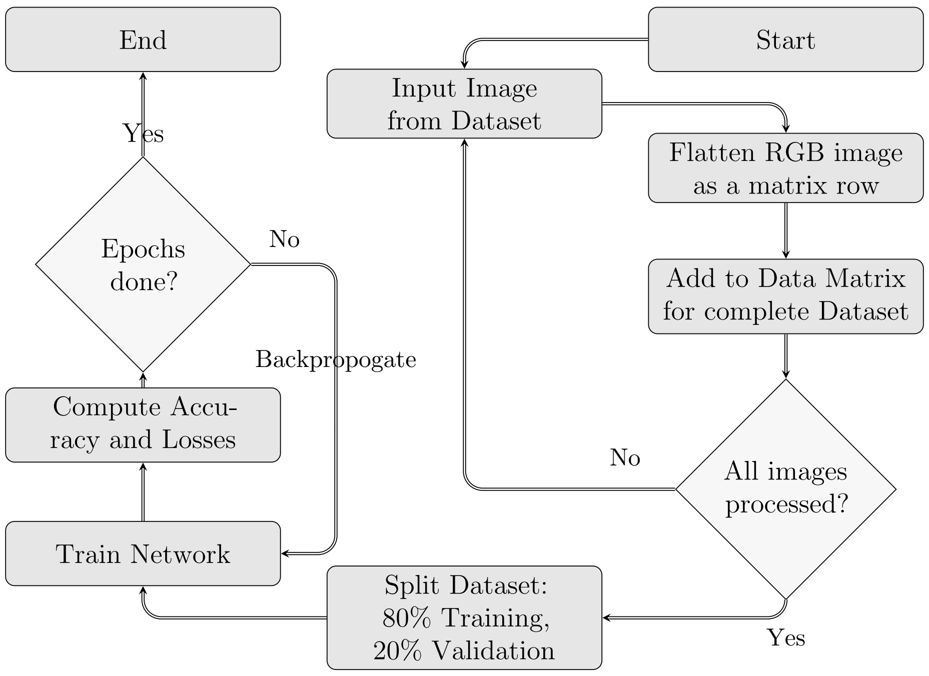

End
Start
Input Image from Dataset
Flatten RGB image as a matrix row
Yes
Epochs done?
No
Add to Data Matrix for complete Dataset
Backpropogate
Compute Accuracy and Losses
No
All images processed?
Train Network
Split Dataset: 80% Training, 20% Validation
Yes

Figure 5.1: Image Training Flow

For all images represented in numerical form, a matrix with the row length of

WxHx3 and column length as $NumberofInputImages$ assuming the images are in RGB format will be created and in case of dataset like CIFAR-10, would be pickled along with the labels associated with every page. The pickled data would then be extracted for training purposes in run-time.

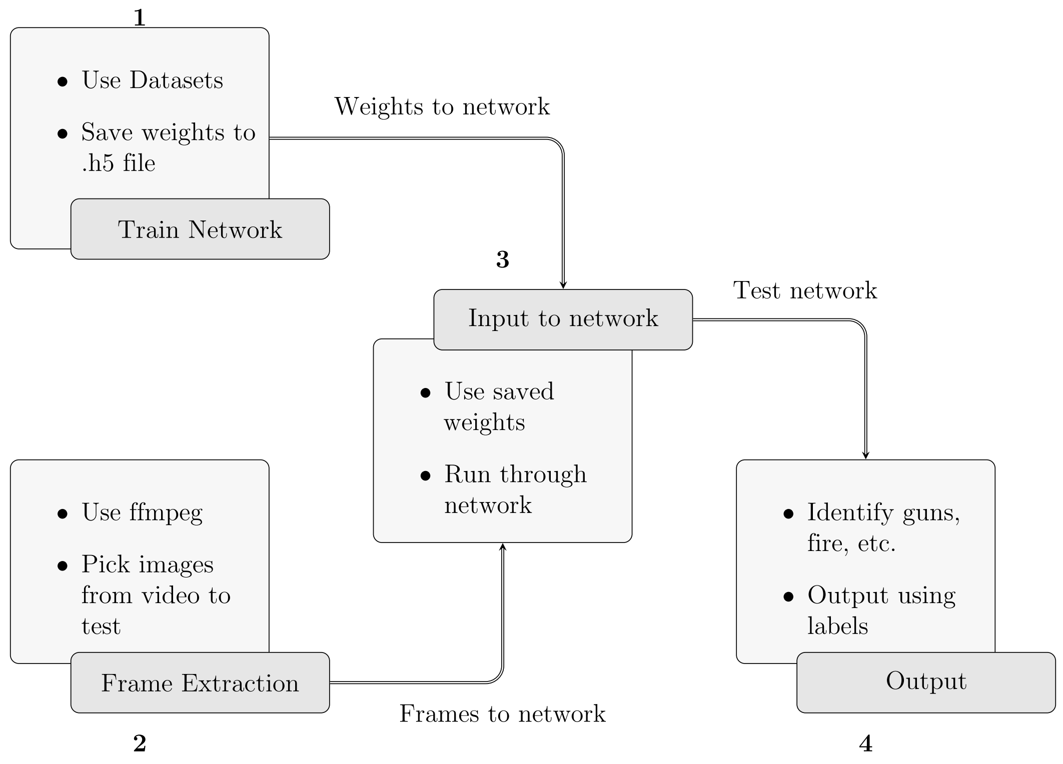


Figure 5.2: Flow Diagram

Figure 5.2 shows the complete flow of our algorithm. The flow is as follows:

1. The network is trained using image datasets and the weights thus learned in the process are saved in a .h5 file.

2. Using the system call of the ffmpeg library, frames are extracted from the video.

3. Extracted frames are then fed into the network which has now the weights received from the .h5 file.

4. The images/frames are tested for objects among the classes if present and the output is generated.

6 Implementation

We started our implementation on two systems with given architectures:

Old Workstation specifications:

- **Processor**: Intel Core i5-4200U CPU @ 1.6GHz
- **RAM**: 6 GB
- **System Type**: Windows x64 based
- **GPU**: Intel HD 4000 Family

On the old system with the above specifications, the following Software and Libraries were installed for the successful implementation of the code :

Python v2.7 — Anaconda v4.1.11 — Spyder v2.0 — Keras v1.0.8 — Theano v0.9.0

New Workstation specifications:

- **Processor**: Intel Xeon(R) CPU E5-1620 v3 @3.50GHz x8
- **RAM**: 64 GB

- **System Type**: Ubuntu 14.04
- **GPU**: NVIDIA GTX 970

We have implemented our algorithm in Python using Theano and Keras deep learning libraries. The software specifications are given below:

- ANACONDA 4.1.11 - It is the leading open data science platform powered by Python. The open source version of Anaconda is a high performance distribution of Python and R and includes over 100 of the most popular Python, R and Scala packages for data science.
- THEANO 0.9.0 - It is a deep learning Python library that allows you to evaluate, optimize, and define mathematical expressions involving multi-dimensional arrays efficiently. Theano features tight integration with NumPy, efficient symbolic differentiation, transparent use of a GPU, speed and stability optimization, dynamic C code generation and extensive unit-testing and self-verification.
- KERAS 1.0.8 - It is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either Theano. It enables fast experimentation, allows for easy and fast prototyping and runs seamlessly on CPU and GPU.

On the new systems, after acquiring the GPU, it became possible for us to use the libraries which can be used with the GPU, the following Softwares and Libraries were installed for the implementation of the code :

Python v2.7 — Anaconda v4.1.11 — Spyder v2.0 — Keras v1.0.8 — Theano v0.9.0 — CUDA v8.0 — CuDNN v5.1 for CUDA.

6.1 Implementation Level Details

Using the above libraries, the main part of implementation comprises of the following functions:

model.add() : This function helps in adding a layer using keras and theano libraries. Following layers can be created-

1. Convolutional Layer (depth, filter size, strides)
2. Pooling Layer (type, filter size, stride)
3. Dense (FC) Layer (Neurons, activation function)

Various other parameters like ReLU activation, ZeroPadding, Dropout and Softmax can be written and thereby implemented in the above function.

extractFramesFromVideo() : This function helps in extracting individual frames, given the number of frames to pick at equal intervals.

resizeImages() : Takes in path of the input image and resizes the image as per the given parameters.

7 Experimentation and Results

7.1 Dataset

The dataset we have used as input is the CIFAR-10 dataset which consists of 60,000 32x32 color images labeled with 10 classes, with 6,000 images per class.There are a total of 50,000 training images and 10,000 test images. The classes used for labeling are: **airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck** The above classes are completely mutually exclusive, there is no overlap between any label.

The data set for python is a 'pickled' object produced using a program called cPickle. The data is stored in a numpy array. Each row of the array stores a 32x32 color image. The first 1024 entries have red channel values, the next 1024, the green and the final 1024 are the blue in row major order.

The labels are in a list of 10,000 numbers ranging from 0–9. The number at index i indicates the lable of the ith image in the array data [12].

7.2 First Approach

We started our implementation with a basic convolutional network. The following code describes the structure of the complete network.

```
#Create the model
model = Sequential()
model.add(ZeroPadding2D((1,1), input_shape=(3,32,32)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(32, activation='relu', W_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

#End of model
```

Listing 7.1: First Model

The model is composed of 2 Convolutional, 2 Padding, 1 Pooling, 2 Dense and 1 Dropout. We use the hyperparameters for padding, pooling, and dropout [3].

We vary the density of the final fully connected layer and the number of epochs to run the network and obtain results for comparison.

7.2.1 Computing with CPU

Working on the CPU with the given architecture, the testing accuracy was found to be dependent on the density of the final fully connected layer of the network and directly proportional to the number of epochs up to the bottleneck limit. The following tables demonstrate our results on the CPU.

Run	**Epoch**	**Dense**	**Accuracy**	**Time(s)**
1	7	64	61.31%	1737
2	15	64	67.85%	6606
3	25	64	69.05%	8497

Table 7.1: Working with CPU - Dense 64

Run	**Epoch**	**Dense**	**Accuracy**	**Time(s)**
1	7	32	58.29%	2593
2	15	32	63.99%	5879
3	25	32	66.89%	6542

Table 7.2: Working with CPU - Dense 32

It is evident from our observations that the time taken and the corresponding accuracy for a particular final dense layer increases with increase in number of epochs. The time taken by the algorithm is noticeably high.

7.2.2 Computing with GPU

The same model was run on the new architecture, this time with the GPU. The code required considerably less amount of time to execute than on the CPU. On comparison, the algorithm took 20x less time on an average to give almost the same, if not higher accuracy rates.

Tables 7.3,7.4 shows the time required for the first model to run on the GPU with Dense 32 and 64.

Run	Epoch	Dense	Accuracy	Time(s)
1	7	64	61.31%	107
2	15	64	67.85%	224
3	25	64	69.05%	371

Table 7.3: Working with GPU - Dense 64

Run	Epoch	Dense	Accuracy	Time(s)
1	7	32	58.29%	104
2	15	32	63.99%	217
3	25	32	66.89%	361

Table 7.4: Working with GPU - Dense 32

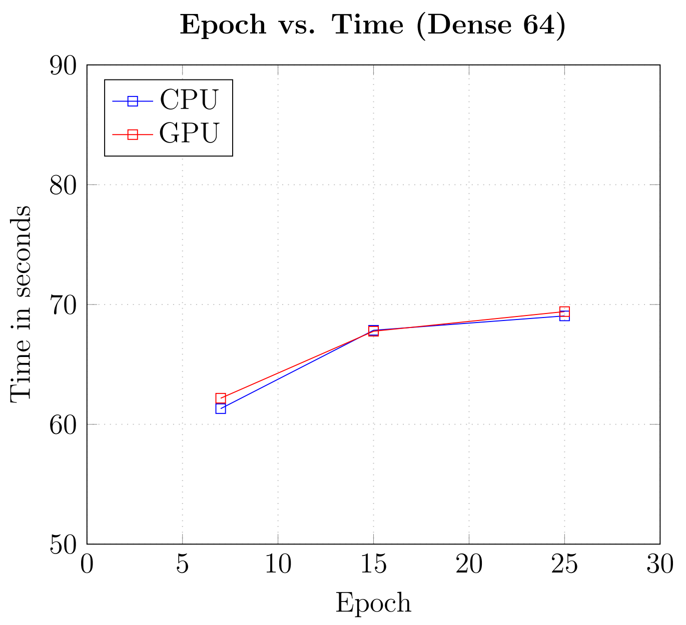


Figure 7.1: Accuracy Comparison: CPU vs. GPU

This led to the conclusion that if we intend to save time then for a specific architecture, we have to compromise on accuracy and in order to increase the accuracy, the depth of the network needs to be increased [23], which would mean the time taken by the algorithm would increase. Since we have saved a lot of time using GPU, this leads us to our next approach.

7.3 Second Approach

We increased the depth of our network to 10 Convolutional Layers, 8 Padding, 3 Pooling, 3 Dense and 2 Dropout Layers. The network model is as follows.

```
#Create the model
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(3, 32, 32)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))


model.add(Flatten())
model.add(Dense(512, activation='relu', W_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu', W_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

#End of model
```

Listing 7.2: Current Model

7.3.1 Working with GPU

We trained this network on the GPU for 25 epochs. We have been able to achieve a testing accuracy of 77%. This is a significant improvement over our first approach which gave a testing accuracy of about 66%.

The observation that GPUs require less time for computation than CPUs is attributed to the fact that GPUs are better equipped at processing and carrying out complex computation on images in comparison to CPUs.

In the next chapter, we discuss our results on working with the algorithm with frames extracted from videos.

8

CNN for Videos

Ideally, the input video for our problem is from surveillance camera footage which is stationary. Such a camera may have a low resolution or gray scale recording. However, since a robust amount of content needs to be extracted in form of frames, we use short video clips containing any of the 10 CIFAR-10 classes to test. We tested our algorithm for 2:30 minute videos each containing pictures of automobiles and airplanes. Our algorithm extracted 16 frames from each video and successfully identified them in all but 4 frames.

We use the versatile FFMPEG library for extracting frames from video. The optimal number of frames is decided based on the following factors:

- Video Resolution
- Video length
- Video frame rate

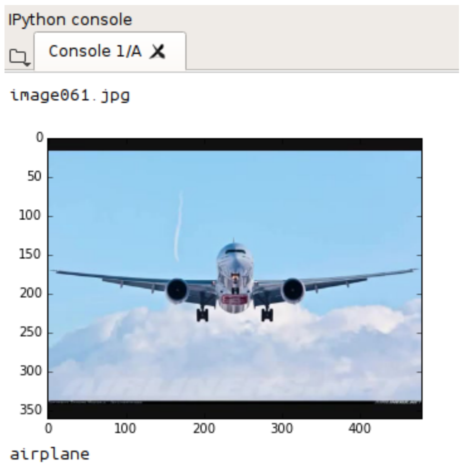
The higher the above parameters, the more processing is required for frame extraction. The images are extracted in the form of video frames at optimal intervals based on the above factors. They are then fed to the neural network to test and finally display the labels of the objects recognized.

```
try:
     ff.extractFramesFromVideo(pathToVideo, pathToFrames, framesToPick,
    everySecond)
     print "Extracted frames"
except:
     print "Couldn't extract frames"

try:
     os.chdir(folderToFrames) ##to set
     for file in glob.glob("*.jpg"):
          pathToFile = folderToFrames + "/" + file
          image = Image.open(pathToFile)
          result = ff.runModel(pathToFile, weightFile)
          print result
except:
     print "Error in retrieving result"
```

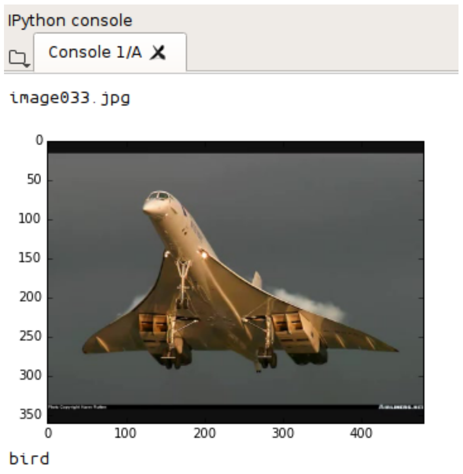
Listing 8.1: Final Code for Video

Following screenshots will show the results aqcquired after feeding the video frames to the code, they contain one false detection and three correct detections of the class *airplane* and *automobile* respectively.



```
image061.jpg
```

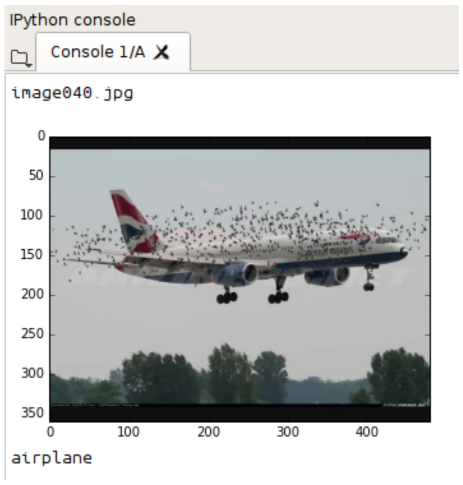
```
airplane
```



```
image033.jpg
```

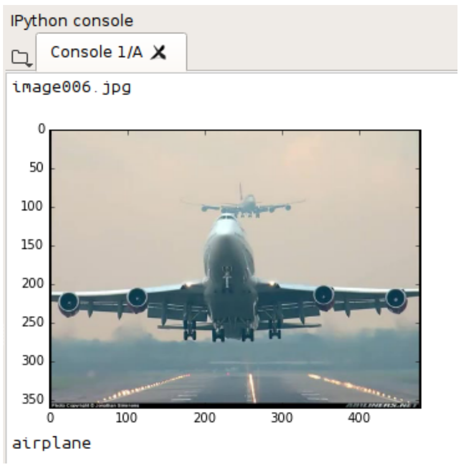
```
bird
```

Figure 8.1: Detect Class : Airplane



```
image040.jpg
```

```
airplane
```

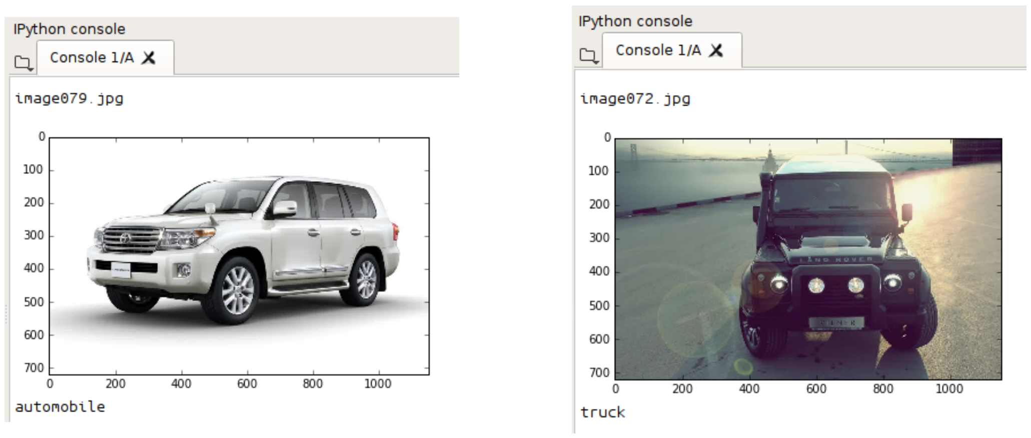


```
image006.jpg
```

```
airplane
```

Figure 8.2: Detect Class : Airplane

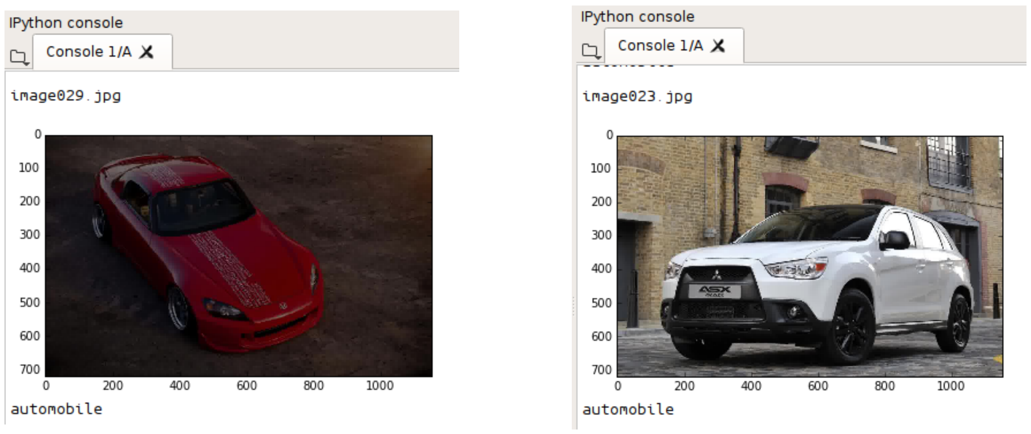
Figures 8.1 and 8.2 contain the frames containing airplanes that were extracted from the video and fed to the network. Attributing to the accuracy and lack of an exhaustible dataset, not every frame has detected an airplane.



```
image079.jpg
automobile
```

```
image072.jpg
truck
```

Figure 8.3: Detect Class : Automobile



```
image029.jpg
automobile
```

```
image023.jpg
automobile
```

Figure 8.4: Detect Class : Automobile

Figure 8.3 and 8.4 contains the frames containing cars extracted from the video and fed to the network. Considering accuracy and lack of an exhaustible dataset, one of the frames showing a false result is also shown.

9 Conclusion

By the observations recorded, the usage of GPU helped in reducing the time needed to train the network. In comparison to processing on the CPU, our model is guaranteed to be trained on the GPU more than twenty times faster on an average. This leads us to conclude that the CNN can be further expanded with its complexity and its training dataset, thereby improving the results of the code and competing with the accuracy of well established networks like AlexNet, VGGNet and LeNet.

Once the network is trained, the established data file containing the weight parameters is used for detection in Videos. Testing is considerably less computationally intensive—the detection can be performed in an architecture with lower specifications—leading to the conclusion that the algorithm can be embedded in hardware of the surveillance equipment for application in real-time environments.

10 Future Work

10.1 Training

Our network has only been trained for 60,000 CIFAR-10 images to give a respectable accuracy of 77%. In the future, this network can be trained for 1.5 million to 10 million distinct images divided into 10,000 classes available for research purposes from the ImageNet database [12].

Furthermore, a multiple GPU architecture may be utilized to efficiently divide the intensive computation, thereby allowing for an even deeper convolutional neural network to be deployed. Such an architecture would require an *NVIDIA* Scalable Link Interface or an AMD Crossfire bridge to connect the GPUs in parallel in order to increase performance.

We unsuccessfully tried to run our algorithm for a self-compiled dataset. Using RGB images from ImageNet and Caltech 101 repositories we scaled them to a 224x224 size and labeled them into 8 classes: **airgun, automatic-weapon, fire, gun, machine-gun, notgun, setgun and weaponsys**. Future work may include building on our existing model for better results with this custom dataset.

Additionally, the detection of environments can also be made possible by adapting the algorithm to simulatneously learn the features of streets, offices, open spaces, monuments, parks, etc. from datasets like the LabelMe dataset.

10.2 Algorithm design

The efficacy of the algorithm can be improved for video input by using multiple optical flow algorithms that have been implemented in the past [24, 21, 10]. The use of motion tracking using 'tracklets' and action recognition in video can also enhance the process of object detection in videos [5].

Four dimensional CNNs using spatio-temporal computation can be used to innovatively process and recognize objects in video frames [11]. Dual stream spatio-temporal CNNs with class score fusion using SVM or averaging have also been proposed [5, 22]. Modeling temporal motion locally (using 3D CNN) or globally (using LSTM/RNN) or a fusion of both may also have scope for implementation [4, 25, 8, 3].

10.3 Scope for applications

The proposed system can be employed in a variety of public environments such as schools, hospitals, railway stations, airports, traffic signals, protected reserves and national parks, and in private environments like banks, police stations. It can be attached to any surveillance camera by training the algorithm with the objects entirely specific to the environment.

It can also be embedded into an instrument to use it as a security alert for blind people. Further, it can also be used as an efficient means for detecting barriers, humans and animals and appropriately apply brakes in self-driven vehicles.

References

[1] Convolutional neural networks for visual recognition. `https://cs231n.github.io/convolutional-networks`. Accessed: 2016-09-30.

[2] Ahn, B. Real-time video object recognition using convolutional neural network. In *2015 International Joint Conference on Neural Networks (IJCNN)* (2015), IEEE, p. 1–7.

[3] Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., and Baskurt, A. Sequential deep learning for human action recognition. In *International Workshop on Human Behavior Understanding* (2011), Springer, p. 29–39.

[4] Ballas, N., Yao, L., Pal, C., and Courville, A. Delving deeper into convolutional networks for learning video representations. *arXiv preprint arXiv:1511.06432* (2015).

[5] Brox, T., and Malik, J. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE transactions on pattern analysis and machine intelligence 33*, 3 (2011), 500–513.

[6] Ciresan, D., Giusti, A., Gambardella, L. M., and Schmidhuber, J. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems* (2012), p. 2843–2851.

[7] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (2009), IEEE, p. 248–255.

[8] Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), p. 2625–2634.

[9] Farabet, C., Couprie, C., Najman, L., and LeCun, Y. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 8 (Aug 2013), 1915–1929.

[10] Farnebäck, G. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis* (2003), Springer, pp. 363--370.

[11] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition* (June 2014), p. 1725–1732.

[12] Krizhevsky, A., and Hinton, G. Learning multiple layers of features from tiny images.

[13] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), p. 1097–1105.

[14] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (Nov 1998), 2278–2324.

[15] Piccardi, M. Video surveillance at the beginning of the third millennium: The viewpoint of research, industry, government bodies, research funding agencies and the community. In *2006 IEEE International Conference on Video and Signal Based Surveillance* (Nov 2006), p. 71–71.

[16] Rajamäki, J., Knuuttila, J., Ruoslahti, H., Viitanen, J., and Patama, P. Transparent surveillance of suspects for building trust between citizens and their governments. In *Intelligence and Security Informatics Conference (EISIC), 2015 European* (Sept 2015), p. 181–181.

[17] Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. Cnn features off-the-shelf: An astounding baseline for recognition. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops* (June 2014), pp. 512--519.

[18] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[19] Sercu, T., and Goel, V. Advances in very deep convolutional neural networks for lvcsr. *arXiv preprint arXiv:1604.01792* (2016).

[20] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229* (2013).

[21] Shi, J., and Tomasi, C. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on* (1994), IEEE, p. 593–600.

[22] Simonyan, K., and Zisserman, A. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems* (2014), p. 568–576.

[23] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[24] Wang, H., Kläser, A., Schmid, C., and Liu, C.-L. Dense trajectories and motion boundary descriptors for action recognition. *International journal of computer vision 103*, 1 (2013), 60–79.

[25] Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., and Toderici, G. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), p. 4694–4702.

Appendix

10.4 Installation Procedure

The installation of GPU is followed by setting up the environment for implementation of algorithm involving installation of NVIDIA, Python and deep learning libraries.

10.4.1 Anaconda

It is an open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment. Its package management system is conda. Open source packages and their dependencies can be installed with a simple

```
conda install <packagename>
```

10.4.2 CUDA

CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia which allows us to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels. It can installed by retrieving the CUDA repository package for Ubuntu 14.04 from the CUDA download site `https://developer.nvidia.com/cuda-downloads` and installing it in a terminal.

10.4.3 Theano

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It can be installed in terminal.

```
conda install -c trung theano=0.8.2.7
```

10.4.4 Keras

Keras is a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. It can be installed in a terminal using the following in terminal.

```
conda install -c conda-forge keras=1.0.7
```

10.4.5 CuDNN

The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. It can be downloaded by registering on `https://developer.nvidia.com/cudnn`.

10.5 Resize Images

Every image is resized into 32x32 before being fed to the algorithm. This is achieved in by using the opencv function in python.

```
resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])
```

10.6 Extract Frames

In order to detect objects in videos, frames are extracted at fixed intervals by making a system call as

```
ffmpeg -i <pathToVideo> -r <framesToPick>/<everySecond> <pathToFrames>
```

where pathToVideo is path to video, framesToPick is number of frames to pick at the moment, everySecond is the number of seconds after which the specified number of frames are picked and pathToFrames is the destination to store the frames.