A Novel Approach to Detect Objects in Videos Using Machine Learning.

Ravindra yadav
Assistant professor
IT Department ,IET DAVV

Vikas Vankhede
Assistant professor
IT Department ,IET DAVV

Jay Singh
Assistant professor
IT Department,IET DAVV

Abstract— This Paper basically contains applications of computer vision. In this Paper we have implemented computer vison and its three major application. First of all we have implemented face recognition system using Rapid object detection using Boosted Cascade of Simple Features paper published by Paul Viola and Michael Jones. Then we further implemented SSD algorithm for object detection using paper SSD: Single Shot MultiBox detection published by Wei Liu. At last we have implemented Generative Adversarial Network using paper publish by Ian Goodfellow. For implementing SSD we have used VOC dataset which is mentioned in paper by Wei Liu. SSD is much better than other normal CNN technique having 76.9% accuracy on 512 x 512 model.

Keywords— CNN, SSD, Generative Adversarial Network, Object detection.

I. INTRODUCTION

The main purpose of this Paper was to get knowledge about applications of computer vision which can be beneficial in real life for example self driving cars are currently implementing computer vision, security systems are implementing computer vision, etc.

Issue involved in this area is how to recognize image correctly so that it could be implemented in easy way. For this we have used several algorithms for getting better efficiency for example SSD algorithm for object detection. Initially it was very difficult to apply such an algorithm due to lack of knowledge in people also older algorithms like viola jones take more time in execution having less efficiency. But SSD and GAN has done much better work as compared to ancient algorithms The main problem on which we are working on is object detection and image generation using features of computer vision. It is a critical task to train a computer model to recognize face, objects and generate images according to that. So, for this we have used several frameworks to implement this challenging situation. It may take several hours to train our model depending on GPU and processor used for implementation. Best GPU is Nvidia and minimum core i5 processor is needed. Although implementation is same for MAC OS, Windows, Linux.

II. RELATED WORKS

We have applied viola Jones algorithm, Generative adversarial Networks and SSD for better result. First of all, we have applied viola- jones research paper for practical implementation. The first contribution of this paper in this Paper is a new image representation called an integral image that allows for very fast feature evaluation. Motivated in part by the work of Papageorgiou et al. our detection system does not work directly with image intensities.

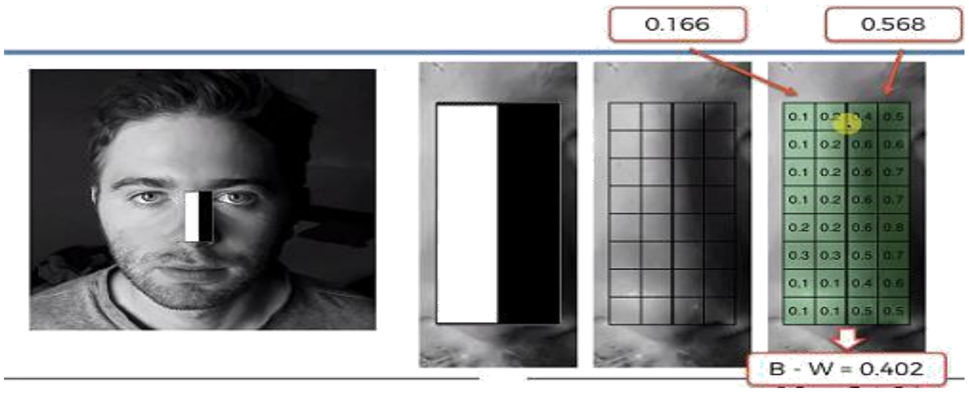


Fig 1.1 Ref: https://www.superdatascience.com/

Then further proposed adversarial nets framework, the generative model is pitted against an adversary: a discriminative model that learns to determine whether a sample is from the model distribution or the data distribution.

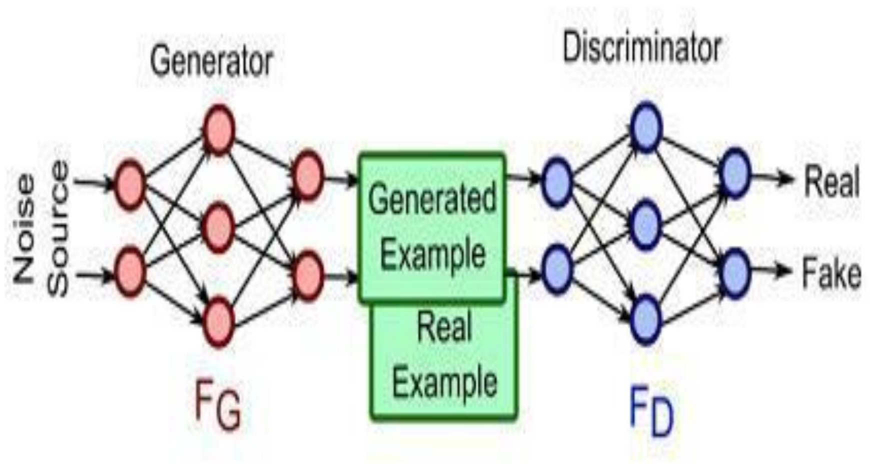


Fig: 1.2 Ref: https://www.arya.org/archievs

At last we introduce SSD, a single-shot detector for multiple categories that is faster than the previous state-of-the-art for single shot detectors (YOLO), and significantly more accurate, in fact as accurate as slower techniques that perform explicit region proposals and pooling (including Faster R-CNN).

III. Implementing Object Detection algorithm

First of all, we have worked on **viola jones Research paper** it has several features discussed below:

i) Basically, it has 2 phases named training and detection. First, we train algorithm using face and Non face images after that we perform detection.

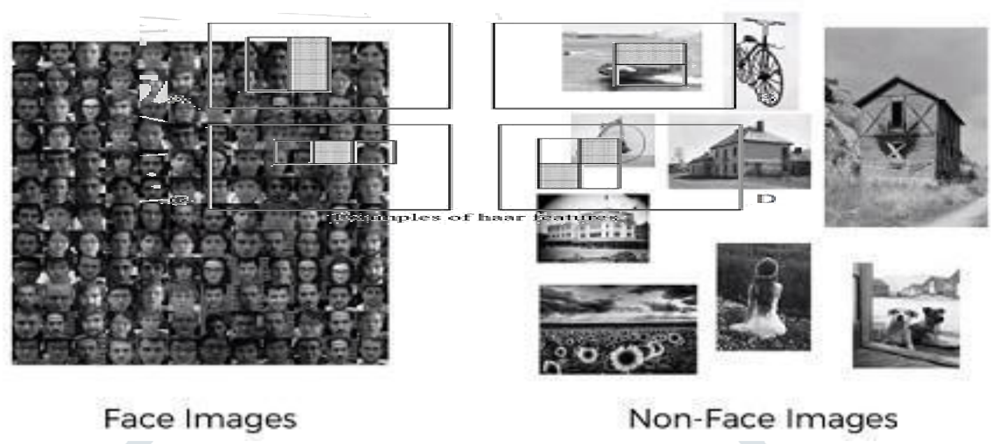


Fig 2.1 Ref: https://www.superdatascience.com/

ii) For training we shrink image into 24 X 24 pixels and then apply Haar Features on it.

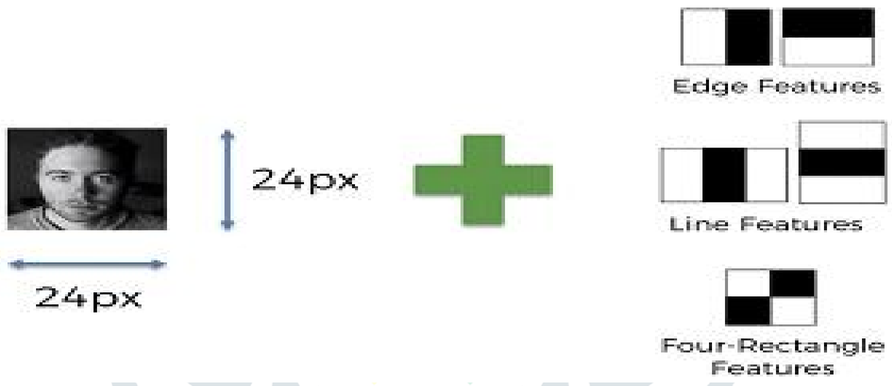


Fig 2.2Ref: https://www.supeerdatascience.com/

Haar Features: These are Haar wavelets based on Fourier transformation. These are of many types some of them is listed below, shown in figure.

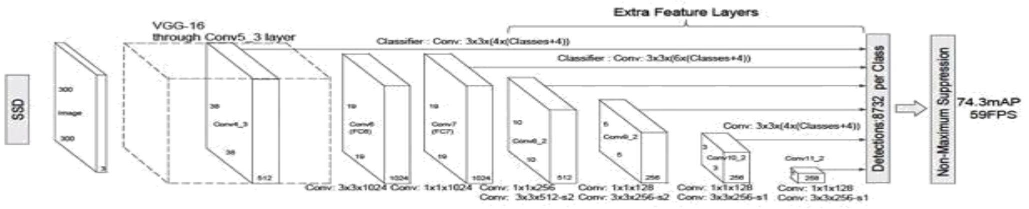


Fig 2.3 Ref: Rapid Object Detection using a Boosted Cascade of Simple Feature paper by Paul viola and Michael Jones.

These features are applied on human face at several area to detect parts of face.

As shown in figure.

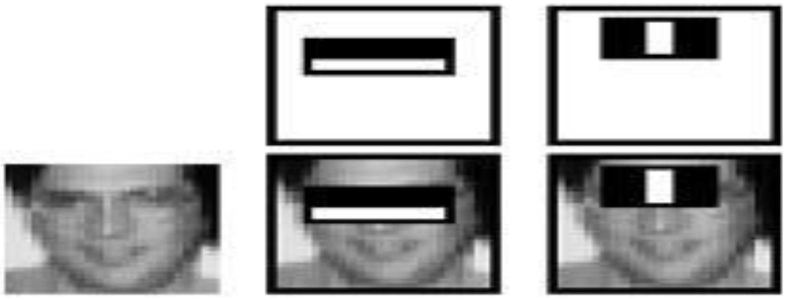
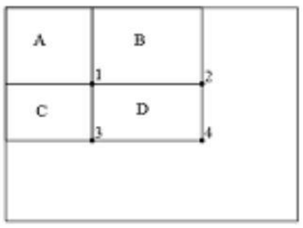


Fig 2.4 Ref: Rapid Object Detection using a Boosted Cascade of Simple Feature paper by Paul viola and Michael Jones

Our object detection procedure classifies images based on the value of simple features. There are many motivations for using features rather than the pixels directly. The most common reason is that features can act to encode ad-hoc domain knowledge that is difficult to learn using a finite quantity of training data.Rectangle features can be computed very rapidly using an intermediate representation for the image which we call the integral image.



$$\begin{aligned} s(x,y) &= s(x,y-1)+i(x,y) \\ ii(x,y) &= ii(x-1,y)+s(x,y) \end{aligned}$$

Fig 2.5 Ref: Rapid Object Detection using a Boosted Cascade of Simple Feature paper by Paul viola and Michael Jones

iii) After Integral image process we perform learning of algorithm

Given a feature set and a training set of positive and negative images, any number of machine learning approaches could be used to learn a classification function. In our system a variant of AdaBoost is used both to select a small set of features and train the classifier. In its original form, the AdaBoost learning algorithm is used to boost the classification performance of a simple (sometimes called weak) learning algorithm.Recall that there are over 180,000 rectangle features associated with each image sub-window, a number far larger than the number of pixels.Even though each feature can be computed very efficiently, computing the complete set is prohibitively expensive.Many general feature selection procedures have been proposed. Ourfinal application demanded a very aggressive approach which would discard the vast majority of features.

Learning Result:

Initial experiments demonstrated that a frontal face classifier constructed from 200 features yields a detection rate of 95% with a false positive rate of 1 in 14084. These results are compelling, but not sufficient for many real-world tasks.

In terms of computation, this classifier is probably faster than any other published system, requiring 0.7 seconds to scan a 384 by 288-pixel image.

iv) After Learning we perform cascading on images.

This section describes an algorithm for constructing a cascade of classifiers which achieves increased detection performance while radically reducing computation time

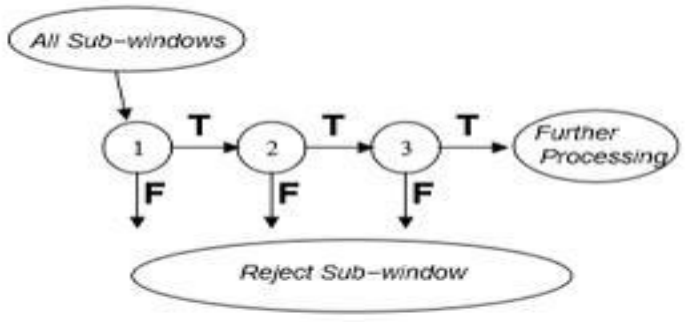


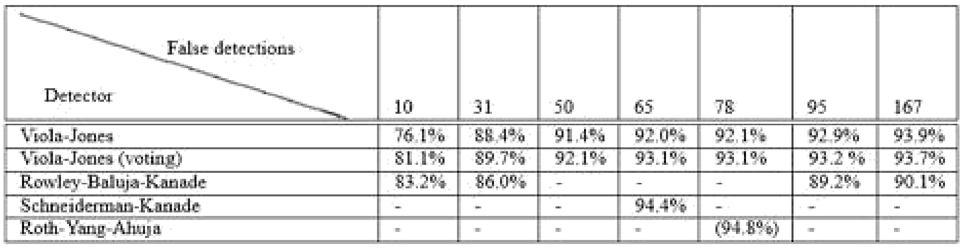
Fig 2.6 Ref: Rapid Object Detection using a Boosted Cascade of Simple Feature paper by Paul viola and Michael Jones

For example, an excellent first stage classifier can be constructed from a two-feature strong classifier by reducing the threshold to minimize false negatives. Measured against a validation training set, the threshold can be adjusted to detect 100% of the faces with a false positive rate of 40%.The structure of the cascade reflects the fact that within any single image an overwhelming majority of subwindows are negative.The cascade training process involves two types of tradeoffs. In most cases classifiers with more features will achieve higher detection rates and lower false positive rates. At the same time classifiers with more features require more time to compute. In principle one could define an optimization framework in which: i) the number of classifier stages, ii) the number of features in each stage, and iii) the threshold of each stage, are traded off in order to minimize the expected number of evaluated features. Unfortunately, finding this optimum is a tremendously difficult problem.The complete face detection cascade has 38 stages with over 6000 features. Nevertheless, the cascade structure results in fast average detection times. On a difficult dataset, containing 507 faces and 75 million sub-windows, faces are detected using an average of 10 feature evaluations per subwindow.

Fig 2.7 Ref: Rapid Object Detection using a Boosted Cascade of Simple Feature paper by Paul viola and Michael Jones

v) Results:

A 38-layer cascaded classifier was trained to detect frontal upright faces. To train the detector, a set of face and nonface training images were used. The face training set consisted of 4916 hand labeled faces scaled and aligned to a base resolution of 24 by 24 pixels.The speed of the cascaded detector is directly related to the number of features evaluated per scanned sub-window.



Detector \ False detections	10	31	50	65	78	95	167
Viola-Jones	76.1%	88.4%	91.4%	92.0%	92.1%	92.9%	93.9%
Viola-Jones (voting)	81.1%	89.7%	92.1%	93.1%	93.1%	93.2 %	93.7%
Rowley-Baluja-Kanade	83.2%	86.0%	-	-	-	89.2%	90.1%
Schneiderman-Kanade	-	-	-	94.4%	-	-	-
Roth-Yang-Ahuja	-	-	-	-	(94.8%)	-	-

Fig 2.8 Ref: Rapid Object Detection using a Boosted Cascade of Simple Feature paper by Paul viola and Michael Jones

So, we can see that how Viola and jones implemented Object detection feature.Its practical implementation is shown in next section.The next paper we studied and implemented was based on **SSD: Single Shot MultiBox Detector** was published by Wei Liu et al.They presented a method for detecting objects in images using a single deep neural network.

Their approach, named SSD, discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. SSD is simple relative to methods that require object proposals because it completely eliminates proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network.

This Paper presents the first deep network based object detector that does not resample pixels or features for bounding box hypotheses and is as accurate as approaches that do. The fundamental improvement in speed comes from eliminating bounding box proposals and the subsequent pixel or feature resampling stage.The whole procedure of implementation of SSD is discussed below:

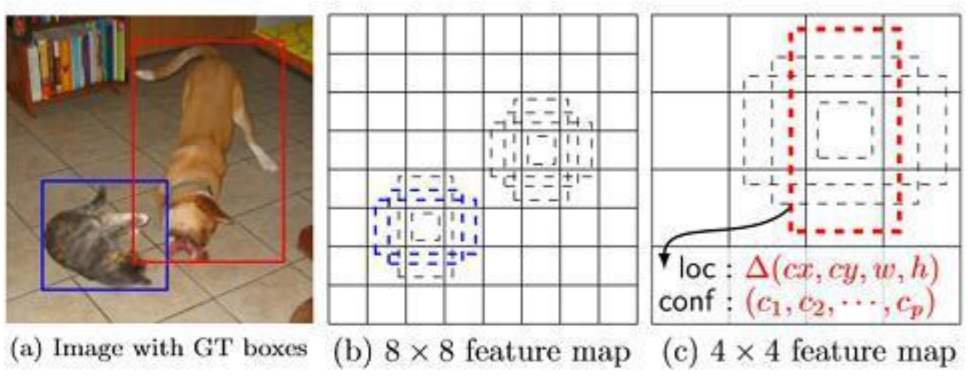


Fig 2.9 Ref: SSD: Single Shot MultiBox Detector paper by Wei Liu et al

i) First of all, we selected Model for implementation.

The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detectionWe add convolutional feature layers to the end of the truncated base network. These layers decrease in size progressively and allow predictions of detections at multiple scales.Each added feature layer can produce a fixed set of detection predictions using set of convolution filters. As shown in figure

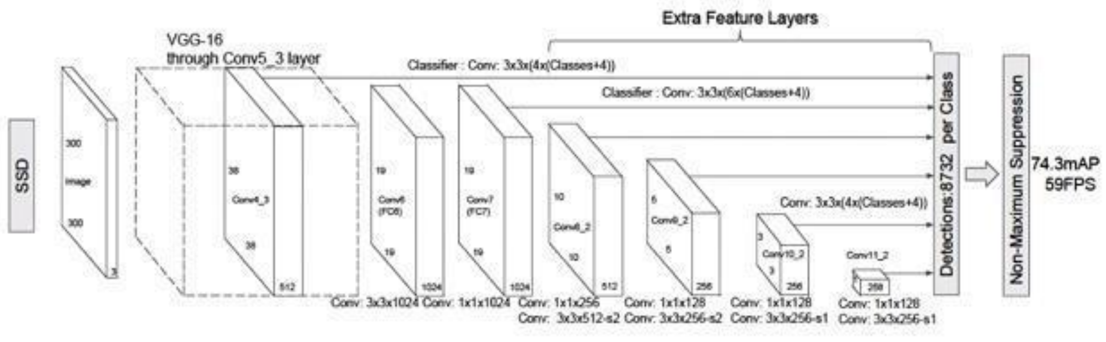


Fig Ref: SSD: Single Shot MultiBox Detector paper by Wei Liu et al

ii)After this we try to **train the model**

The key difference between training SSD and training a typical detector that uses region proposal is that ground truth information needs to be assigned to specific outputs in the fixed set of detector outputs.During training we need to determine which default boxes correspond to a ground truth detection and train the network accordingly. For each ground truth box we are selecting from default boxes that vary over location, aspect ratio, and scale. We begin by matching each ground truth box to the default box with the best Jaccard overlap.**Training objective** The SSD training objective is derived from the MultiBox objective but is extended to handle multiple object categories. During training, the scale of the default boxes for each feature map is computed as:

$$s_k = s_{\text{min}} + \frac{s_{\text{max}} - s_{\text{min}}}{m-1}(k-1), \quad k \in [1, m]$$

where smin is 0.2 and smax is 0.9, meaning the lowest layer has a scale of 0.2 and the highest layer has a scale of 0.9, and all layers in between are regularly spaced.During training we apply multiple boxes to detect the position of the object, as shown in diagram.

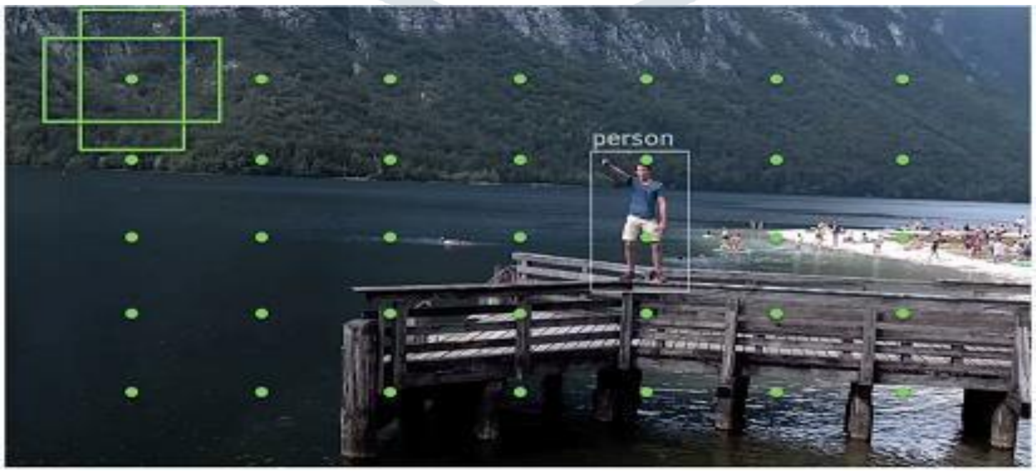


Fig 2.10 Ref: https://www.superdatascience.com/

iii)We tried to implement it on 2 datasets VOC2007 and VOC2012 and found results.

Here we mainly worked on 20 different objects using above mentioned datasets.

PASCAL VOC2007: on this dataset we tried to visualize the result generated as shown in figure below.

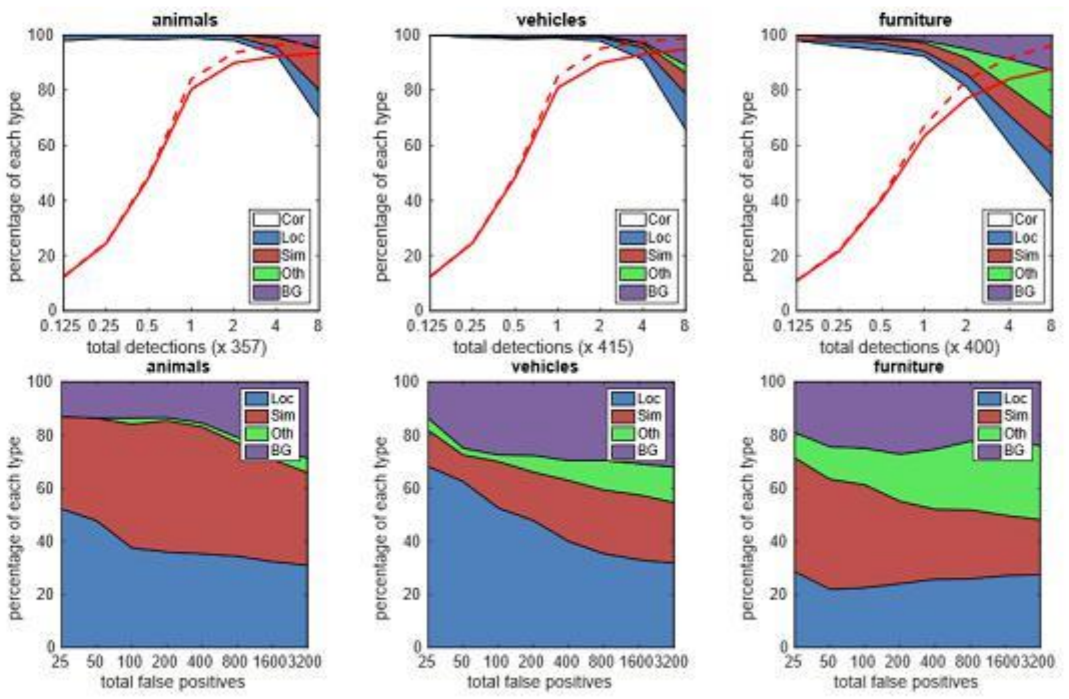
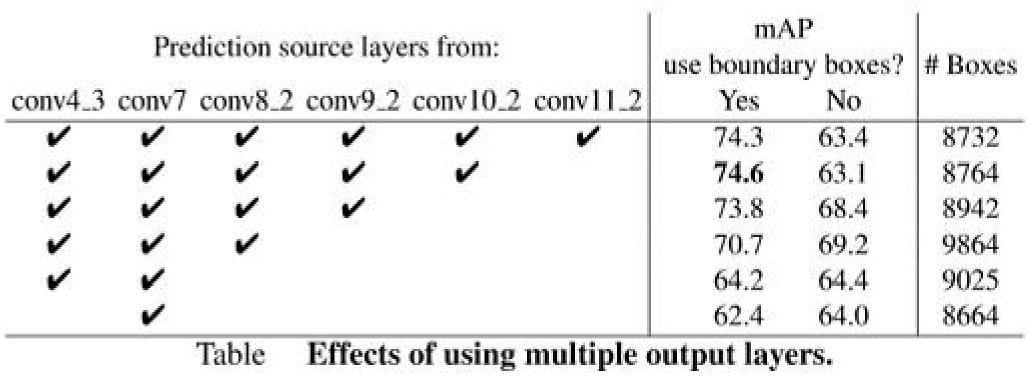


Fig 2.11 Ref: SSD: Single Shot MultiBox Detector paper by Wei Liu et al

iv) Finally, we found that **Multiple output layers at different resolutions is better**. A major contribution of SSD is using default boxes of different scales on different output layers

PASCALVOC2012: We use the same settings as those used for our basic VOC2007 experiments above except that we use VOC2012 trainval and in VOC2007 we used train and test.



Prediction source layers from:						mAP use boundary boxes?		# Boxes
conv4_3	conv7	conv8_2	conv9_2	conv10_2	conv11_2	Yes	No	
✔	✔	✔	✔	✔	✔	74.3	63.4	8732
✔	✔	✔	✔	✔		**74.6**	63.1	8764
✔	✔	✔	✔			73.8	68.4	8942
✔	✔	✔				70.7	69.2	9864
✔	✔					64.2	64.4	9025
	✔					62.4	64.0	8664

Table **Effects of using multiple output layers.**

Fig 2.12 Ref: SSD: Single Shot MultiBox Detector paper by Wei Liu et al Above figure shows the prediction of all layers.

At last we applied GENERATIVE ADVERSARIAL NETWORK in our Paper for generating some random images.

Although there are various applications of GANs, some of them are listed below:

i) Generating images
ii) Image modifications
iii) Super resolution
iv) Assisting artist
v) Photo Realistic images
vi) Speech generation
vii) Face aging

We worked on a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game.

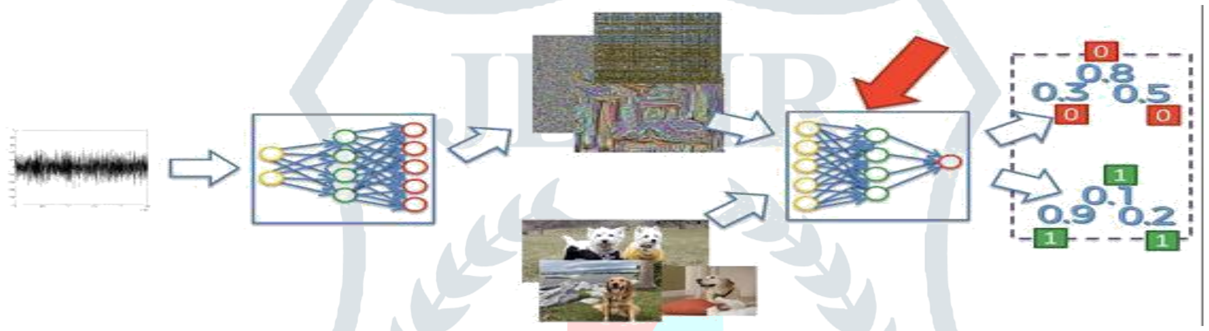


Fig 2.13 Ref: https://www.superdatscience.com/

As shown in image there are 2 subparts of GAN

i) Generator
ii) Discriminator

This framework can yield specific training algorithms for many kinds of model and optimization algorithm. In this Paper , we explore the special case when the generative model generates samples by passing random noise through a multilayer perceptron, and the discriminative model is also a multilayer perceptron.

Experiment: We trained adversarial nets on a range of datasets including CIFAR-10. We estimate probability of the test set data under pg by fitting a Gaussian Parzen window to the samples generated with G and reporting the log-likelihood under this distribution.Output after training the model is shown below

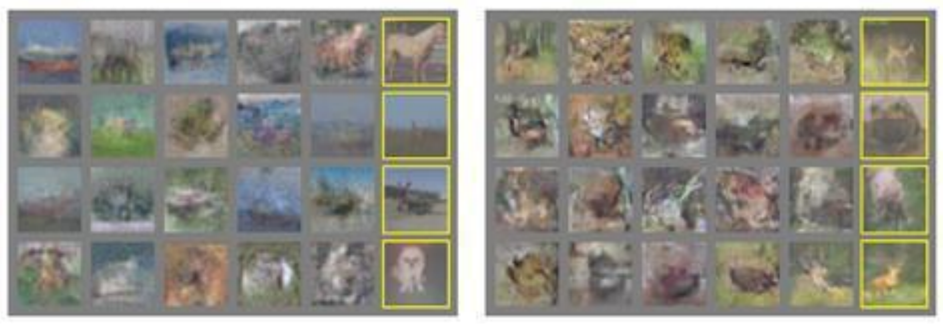


Fig 2.14 Ref: Generative Adversarial Nets paper by Ian G Goodfellow

General steps performed in implementation of this framework is mentioned below:

i) We first try to train the discriminator on real images.

ii) Generator try to generate random images in parallel while training the discriminator.

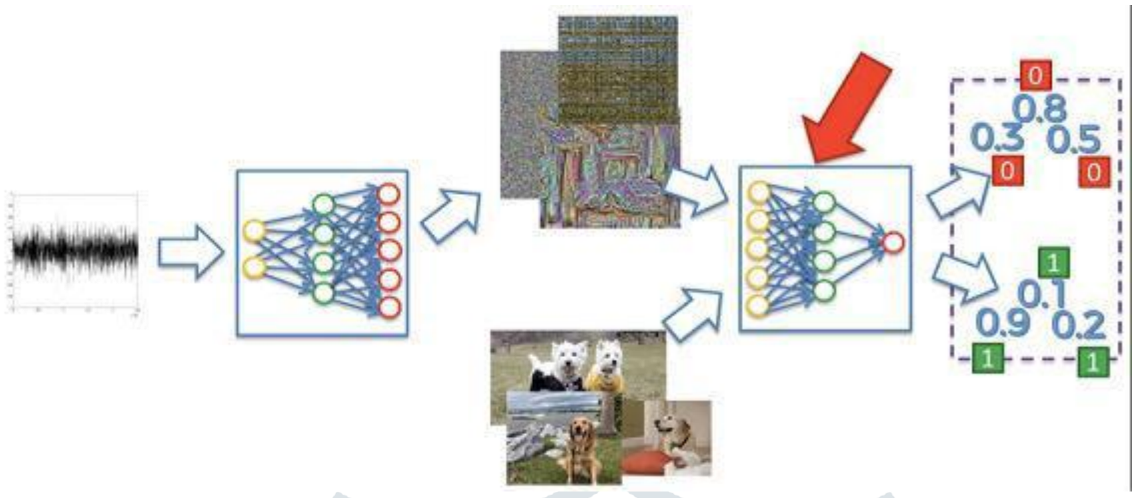


Fig 2.15 Ref: https://www.superdatascience.com/

As shown in figure discriminator is being trained as well as generator is generating images.

iii) Discriminator is trained on fake images too so that it could be easy for it to recognize fake as well as real images.

iv) If the images generated by generator will not match the threshold value the backpropagation is performed by Discriminator.

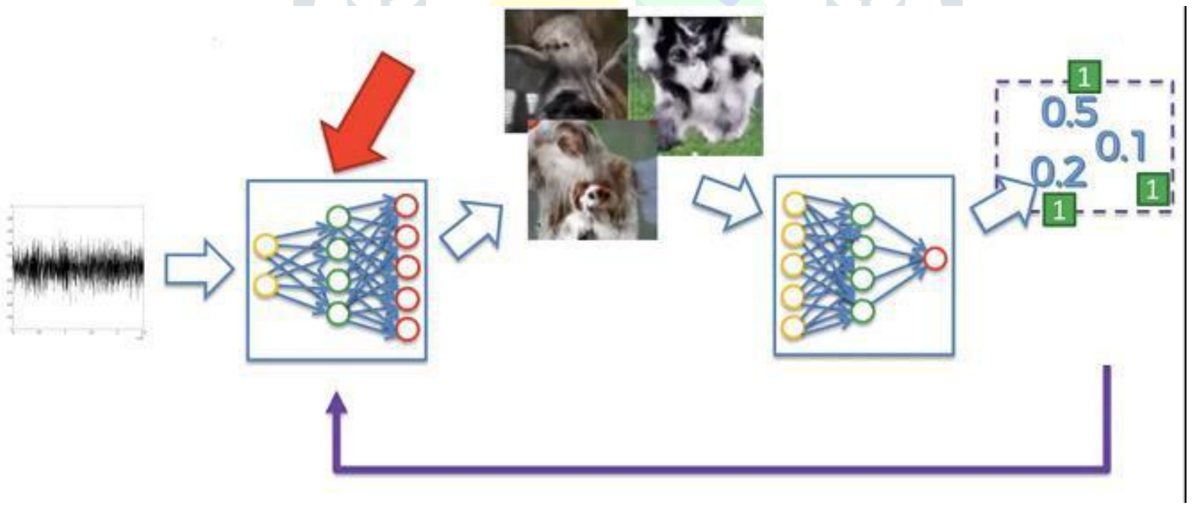


Fig 2.16 Ref: https://www.superdatascience.com/

This procedure is continued till we don't get the desired output.

Result: We have applied it on CIFAR-10 dataset to generate some random images. We have executed it till 25 epochs to get better result.

After 1st epoch the result was

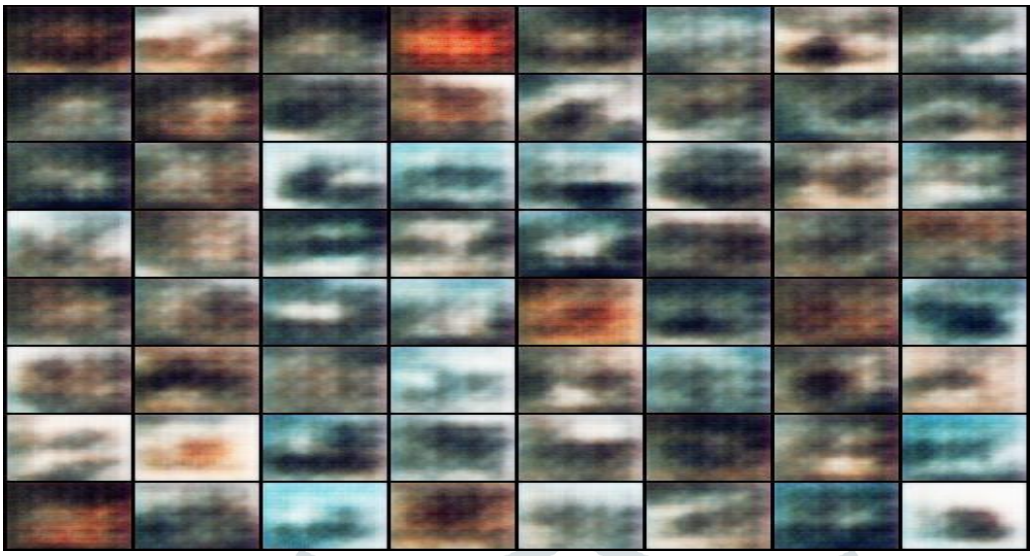


Fig 2.18(applied on anaconda spyder 3.3.3 using pytorch library)

After 25th epoch result was

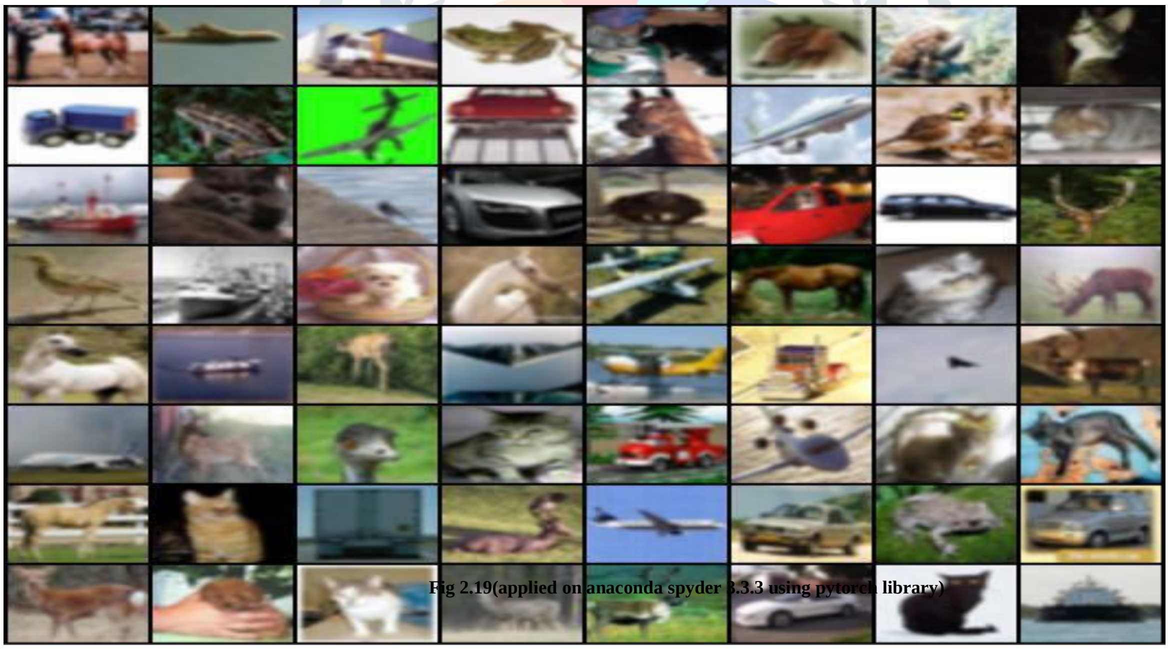


Fig 2.19(applied on anaconda spyder 3.3.3 using pytorch library)

IV. Design

4.1 Technology Selection

We basically worked on data science technology in which we have downloaded datasets from various sites and various types like CIFAR10, VOC2007, VOC2012.

We implemented algorithm on these datasets for training them and further tested them by real time data.All the tests are performed on spyder IDE which supports python. We have used python3.5 and python 3.6 for different purposes.Most of the part of this Paper is based on mathematics implementation mainly on **probability distribution**. So, one should have at least knowledge of basic probability and **trigonometry** to implement this Paper model.For implementing neural network, one should have knowledge of trigonometric functions like **Hyperbolic Tan, Sigmoid, SoftMax**.One should have at least knowledge of perceptron or working of biological neural network and its structure. Because the technology that we have selected is based on working of neural network.

4.2 Datasets Design

In this Paper , we have used image datasets to feed neural networks. For a single model approx. 10000 image datasets are needed to implement. We have mainly used three datasets:

i) CIFAR-10
ii) PASCAL VOC2007
iii) PASCAL VOC2012

CIFAR10 has basically 5 batch files which include approx. 10000 images. We have used this to feed our GAN for generating random images after train of 25 epochs.

One can download it from

https://www.kaggle.com/c/cifar-10

PASCAL VOC2007 and PASCAL VOC2012 datasets used for implementing SSD network.We have separately implemented both datasets to train our model on 20 type of objectsOne can download VOC dataset it from https://github.com/DrSleep/tensorflow-deeplab-resnet/issues/128

5. Implementation and Testing

5.1 Subsystem and their dependancies

We have implemented all three frameworks on anaconda framework using python 3.5 and find out the output results which we are including here.

Source Code for viola Jones Algorithm for face detection:

```
1. -*- coding: utf-8 -*-
"""Created on Thu Mar 28 01:50:45 2019

@author: Ravindra Yadav

"""

import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

smile_cascade = cv2.CascadeClassifier('haarcascade_smile.xml')
```

#define function that will detect

def detect(gray, frame):

faces = face_cascade.detectMultiScale(gray, 1.3, 5)

for(x, y, w, h) in faces:

cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

roi_gray = gray[y:y+h, x:x+w]

roi_color = frame[y:y+h, x:x+w]

eyes = eye_cascade.detectMultiScale(roi_gray, 1.1, 22)

for(ex, ey, ew, eh) in eyes:

cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2) smiles = smile_cascade.detectMultiScale(roi_gray, 1.7, 22) for(sx, sy, sw, sh) in smiles:

cv2.rectangle(roi_color, (sx, sy), (sx+sw, sy+sh), (0, 0, 255), 2) return frame

#Doing face detection using webcam

video_capture = cv2.VideoCapture(0)

while True:

_, frame = video_capture.read()

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

canvas = detect(gray, frame)

cv2.imshow('Video', canvas)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

video_capture.release()

cv2.destroyAllWindows()

Source code for SSD implementation:

2. , epoch), normalize = True) Object Detection
3. Importing the libraries import torch

from torch.autograd import Variable import cv2
from data import BaseTransform, VOC_CLASSES as labelmap from ssd import build_ssd
import imageio
Defining a function that will do the detections

def detect(frame, net, transform):

height, width = frame.shape[:2]

frame_t = transform(frame)[0]

x = torch.from_numpy(frame_t).permute(2, 0, 1) x = Variable(x.unsqueeze(0))

y = net(x)

detections = y.data

scale = torch.Tensor([width, height, width, height])

ii) detections = [batch, number of classes, number of occurence, (score, x0, Y0, x1, y1)]

for i in range(detections.size(1)):

j = 0

while detections[0, i, j, 0] >= 0.2:

pt = (detections[0, i, j, 1:] * scale).numpy()

cv2.rectangle(frame, (int(pt[0]), int(pt[1])), (int(pt[2]), int(pt[3])), (255, 0, 0), 2)

cv2.putText(frame, labelmap[i - 1], (int(pt[0]), int(pt[1])), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255), 2, cv2.LINE_AA)

j += 1

return frame

iii) Creating the SSD neural network net = build_ssd('test')

net.load_state_dict(torch.load('ssd300_mAP_77.43_v2.pth', map_location = lambda storage, loc: storage))

# Creating the transformation

transform = BaseTransform(net.size, (104/256.0, 117/256.0, 123/256.0))

v) Doing some Object Detection on a video reader = imageio.get_reader('boatrace.mp4') fps = reader.get_meta_data()['fps']

writer = imageio.get_writer('outputrace.mp4', fps = fps) for i, frame in enumerate(reader):

frame = detect(frame, net.eval(), transform) writer.append_data(frame)

print(i)

writer.close()

<u>Source code for GANs implementation:</u>

vi) Deep Convolutional GANs
vii)
viii) Importing the libraries

from __future__ import print_function

import torch

import torch.nn as nn

import torch.nn.parallel

import torch.optim as optim

import torch.utils.data

import torchvision.datasets as dset

import torchvision.transforms as transforms import torchvision.utils as vutils from torch.autograd import Variable

# Setting some hyperparameters

batchSize = 64 # We set the size of the batch.

imageSize = 64 # We set the size of the generated images (64x64).

# Creating the transformations

transform = transforms.Compose([transforms.Scale(imageSize), transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),]) # We create a list of transformations (scaling, tensor conversion, normalization) to apply to the input images.

# Loading the dataset

dataset = dset.CIFAR10(root = './data', download = True, transform = transform) # We download the training set in the ./data folder and we apply the previous transformations on each image.

dataloader = torch.utils.data.DataLoader(dataset, batch_size = batchSize, shuffle = True, num_workers = 2) # We use dataLoader to get the images of the training set batch by batch.

ii) Defining the weights_init function that takes as input a neural network m and that will initialize all its weights.

def weights_init(m):

classname = m.__class__.__name__

if classname.find('Conv') != -1:

m.weight.data.normal_(0.0, 0.02)

elif classname.find('BatchNorm') != -1:

m.weight.data.normal_(1.0, 0.02)

m.bias.data.fill_(0)

ii) Defining the generator

class G(nn.Module):

def __init__(self):

super(G, self).__init__()

self.main = nn.Sequential(

nn.ConvTranspose2d(100, 512, 4, 1, 0, bias = False),

nn.BatchNorm2d(512),

nn.ReLU(True),

nn.ConvTranspose2d(512, 256, 4, 2, 1, bias = False),
nn.BatchNorm2d(256),

```
nn.ReLU(True),

nn.ConvTranspose2d(256, 128, 4, 2, 1, bias = False),

nn.BatchNorm2d(128),

nn.ReLU(True),

nn.ConvTranspose2d(128, 64, 4, 2, 1, bias = False),

nn.BatchNorm2d(64),

nn.ReLU(True),

nn.ConvTranspose2d(64, 3, 4, 2, 1, bias = False), nn.Tanh()
)

def forward(self, input):

output = self.main(input)

return output
```

iv) Creating the generator netG = G() netG.apply(weights_init)

v) Defining the discriminator

```
class D(nn.Module):

def __init__(self):

super(D, self).__init__()
self.main = nn.Sequential( nn.Conv2d(3, 64, 4, 2, 1, bias = False), nn.LeakyReLU(0.2, inplace =
True), nn.Conv2d(64, 128, 4, 2, 1, bias = False), nn.BatchNorm2d(128), nn.LeakyReLU(0.2,
inplace = True), nn.Conv2d(128, 256, 4, 2, 1, bias = False), nn.BatchNorm2d(256),
nn.LeakyReLU(0.2, inplace = True), nn.Conv2d(256, 512, 4, 2, 1, bias = False),
nn.BatchNorm2d(512), nn.LeakyReLU(0.2, inplace = True), nn.Conv2d(512, 1, 4, 1, 0, bias =
False), nn.Sigmoid()
)

def forward(self, input):

output = self.main(input)

return output.view(-1)
```

v) Creating the discriminator netD = D() netD.apply(weights_init)

vi) Training the DCGANs

```
criterion = nn.BCELoss()

optimizerD = optim.Adam(netD.parameters(), lr = 0.0002, betas = (0.5, 0.999))

optimizerG = optim.Adam(netG.parameters(), lr = 0.0002, betas = (0.5, 0.999))
```

```
for epoch in range(25):

for i, data in enumerate(dataloader, 0):
```

vi) 1st Step: Updating the weights of the neural network of the discriminator

```
netD.zero_grad()
```

vii) Training the discriminator with a real image of the dataset real, _ = data

```
input = Variable(real)

target = Variable(torch.ones(input.size()[0])) output = netD(input)

errD_real = criterion(output, target)
```

viii) Training the discriminator with a fake image generated by the generator noise = Variable(torch.randn(input.size()[0], 100, 1, 1))

```
fake = netG(noise)

target = Variable(torch.zeros(input.size()[0])) output = netD(fake.detach())

errD_fake = criterion(output, target)
```

viii) Backpropagating the total error errD = errD_real + errD_fake errD.backward()

```
optimizerD.step()
```

ix) 2nd Step: Updating the weights of the neural network of the generator

```
netG.zero_grad()

target = Variable(torch.ones(input.size()[0]))

output = netD(fake)

errG = criterion(output, target)


errG.backward()

optimizerG.step()
```

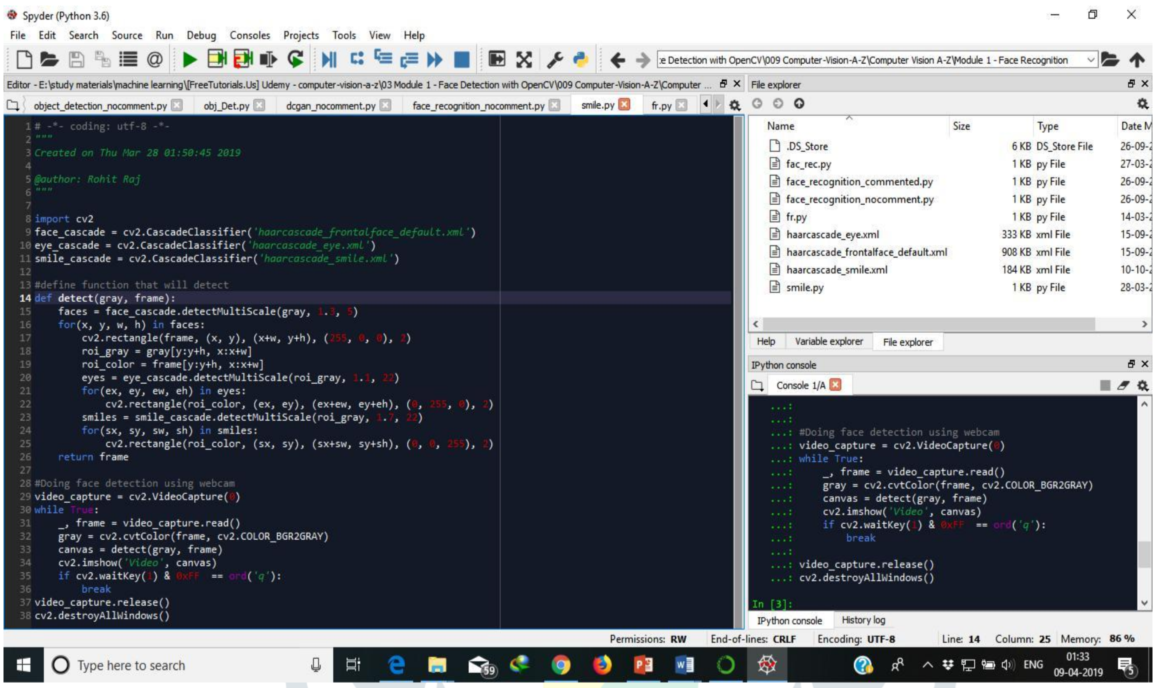
iii) 3rd Step: Printing the losses and saving the real images and the generated images of the minibatch every 100 steps

```
print('[%d/%d][%d/%d] Loss_D: %.4f Loss_G: %.4f' % (epoch, 25, i, len(dataloader),
(errD.data).item(), (errG.data).item()))

if i % 100 == 0:

vutils.save_image(real, '%s/real_samples.png' % "./result1", normalize =
True)

fake = netG(noise)

vutils.save_image(fake.data, '%s/fake_samples_epoch_%03d.png' % ("./result1"
```

Implementation of viola – Jones research paper Rapid Object Detection using a Boosted Cascade of Simple Feature given below:

i) We implemented Haar features for face, smile and eye detection, as shown in screenshot.

ii) For this we have to set the environment in spyder as base environment.



```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 28 01:50:45 2019

@author: Rohit Raj
"""

import cv2
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
smile_cascade = cv2.CascadeClassifier('haarcascade_smile.xml')

#define function that will detect
def detect(gray, frame):
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for(x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray, 1.1, 22)
        for(ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)
        smiles = smile_cascade.detectMultiScale(roi_gray, 1.7, 22)
        for(sx, sy, sw, sh) in smiles:
            cv2.rectangle(roi_color, (sx, sy), (sx+sw, sy+sh), (0, 0, 255), 2)
    return frame

#Doing face detection using webcam
video_capture = cv2.VideoCapture(0)
while True:
    _, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    canvas = detect(gray, frame)
    cv2.imshow('Video', canvas)
    if cv2.waitKey(1) & 0xFF  == ord('q'):
        break
video_capture.release()
cv2.destroyAllWindows()
```

Name	Size	Type	Date M
.DS_Store	6 KB	DS_Store File	26-09-2
fac_rec.py	1 KB	py File	27-03-2
face_recognition_commented.py	1 KB	py File	26-09-2
face_recognition_nocomment.py	1 KB	py File	26-09-2
fr.py	1 KB	py File	14-03-2
haarcascade_eye.xml	333 KB	xml File	15-09-2
haarcascade_frontalface_default.xml	908 KB	xml File	15-09-2
haarcascade_smile.xml	184 KB	xml File	10-10-2
smile.py	1 KB	py File	28-03-2

```
   ...:
   ...:
   ...: #Doing face detection using webcam
   ...: video_capture = cv2.VideoCapture(0)
   ...: while True:
   ...:     _, frame = video_capture.read()
   ...:     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
   ...:     canvas = detect(gray, frame)
   ...:     cv2.imshow('Video', canvas)
   ...:     if cv2.waitKey(1) & 0xFF  == ord('q'):
   ...:         break
   ...:
   ...: video_capture.release()
   ...: cv2.destroyAllWindows()

In [3]:
```

iii) We executed the code and get output which is mentioned below.

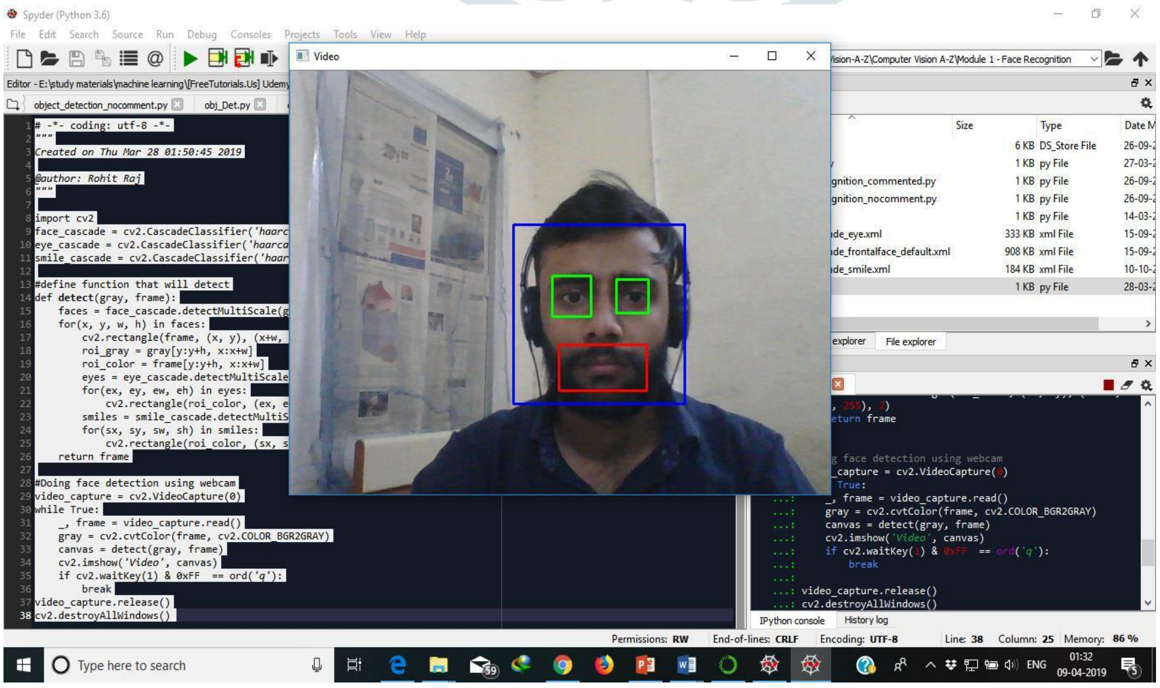
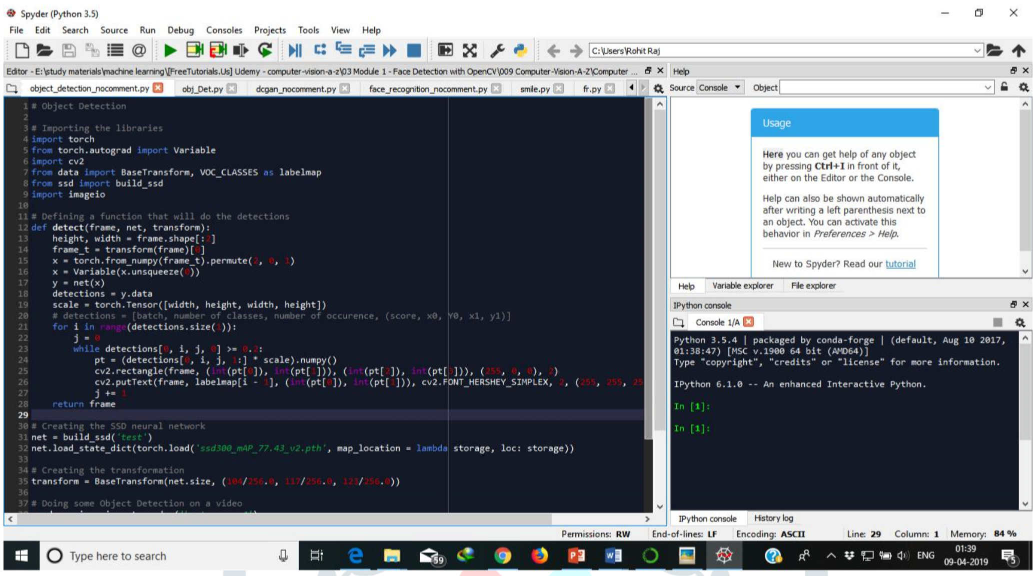


Fig 4.2

Implementation of **SSD: Single shot MultiBox Detector** paper by Wei Liu is mentioned below:

i) We implemented SSD using pytorch library and downloaded datasets for training

ii) For this we used different environment in anaconda named virtual_environment on which have installed all required library for implementing SSD.


Spyder (Python 3.5)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Rohit Raj
Editor - E:\study materials\machine learning\[FreeTutorials.Us] Udemy - computer-vision-a-z\03 Module 1 - Face Detection with OpenCV\009 Computer-Vision-A-Z\Computer ...
object_detection_nocomment.py
obj_Det.py
dcgan_nocomment.py
face_recognition_nocomment.py
smile.py
fr.py
Help
Variable explorer
File explorer
IPython console
Console 1/A
History log
Permissions: RW
End-of-lines: LF
Encoding: ASCII
Line: 29
Column: 1
Memory: 84 %
Type here to search
ENG
01:39
09-04-2019

```
# Object Detection

# Importing the libraries
import torch
from torch.autograd import Variable
import cv2
from data import BaseTransform, VOC_CLASSES as labelmap
from ssd import build_ssd
import imageio

# Defining a function that will do the detections
def detect(frame, net, transform):
    height, width = frame.shape[:2]
    frame_t = transform(frame)[0]
    x = torch.from_numpy(frame_t).permute(2, 0, 1)
    x = Variable(x.unsqueeze(0))
    y = net(x)
    detections = y.data
    scale = torch.Tensor([width, height, width, height])
    # detections = [batch, number of classes, number of occurence, (score, x0, Y0, x1, y1)]
    for i in range(detections.size(1)):
        j = 0
        while detections[0, i, j, 0] >= 0.2:
            pt = (detections[0, i, j, 1:] * scale).numpy()
            cv2.rectangle(frame, (int(pt[0]), int(pt[1])), (int(pt[2]), int(pt[3])), (255, 0, 0), 2)
            cv2.putText(frame, labelmap[i - 1], (int(pt[0]), int(pt[1])), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 25
            j += 1
    return frame

# Creating the SSD neural network
net = build_ssd('test')
net.load_state_dict(torch.load('ssd300_mAP_77.43_v2.pth', map_location = lambda storage, loc: storage))

# Creating the transformation
transform = BaseTransform(net.size, (104/256.0, 117/256.0, 123/256.0))

# Doing some Object Detection on a video
```

Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
Python 3.5.4 | packaged by conda-forge | (default, Aug 10 2017,
01:38:47) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.1.0 -- An enhanced Interactive Python.

In [1]:

In [1]:
```

Fig 4.3 Ref: spyder 3.3.3 from anaconda framework.

iii) Output as shown in figure below:
When we implemented using boat video, output is:

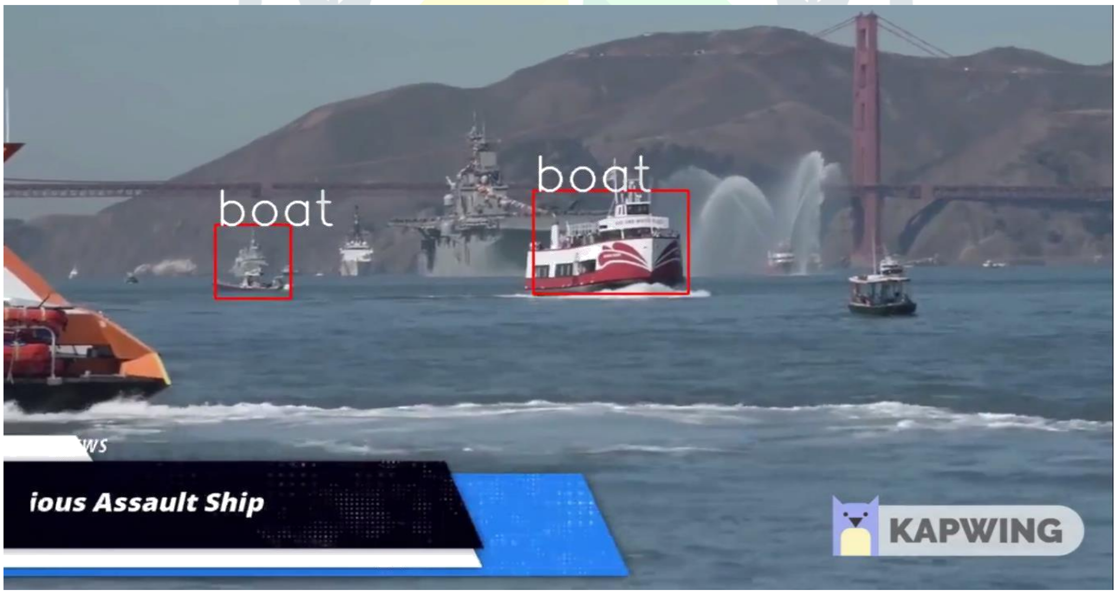


Fig 4.4

When we implemented using Dog video, output is:

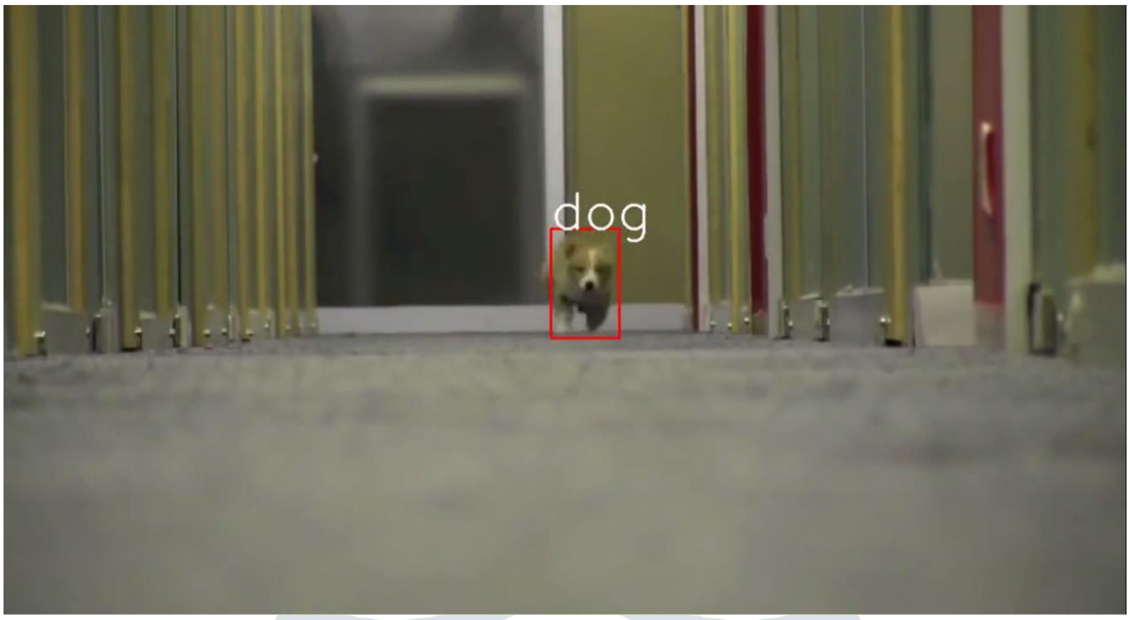
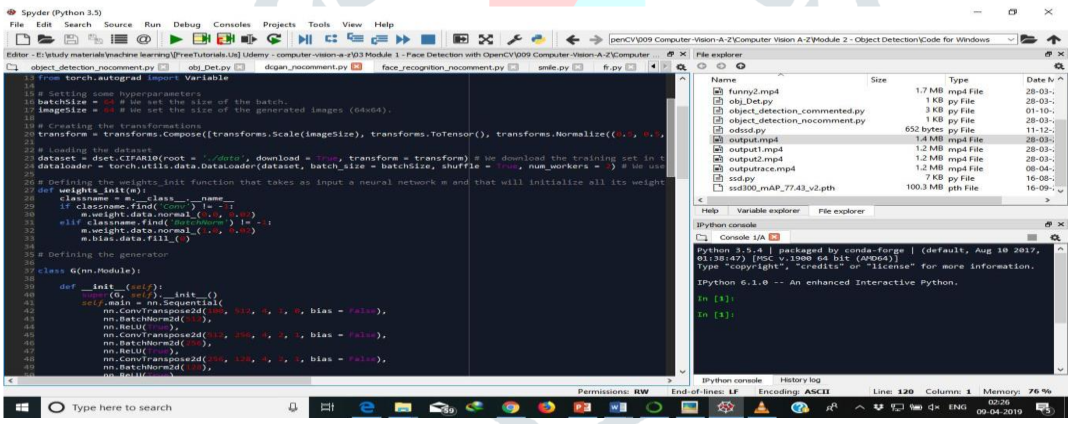


Fig 4.5

Implementation of GENERATIVE ADVAERSARIAL NETWORK paper by Ian G. Goodfellow:

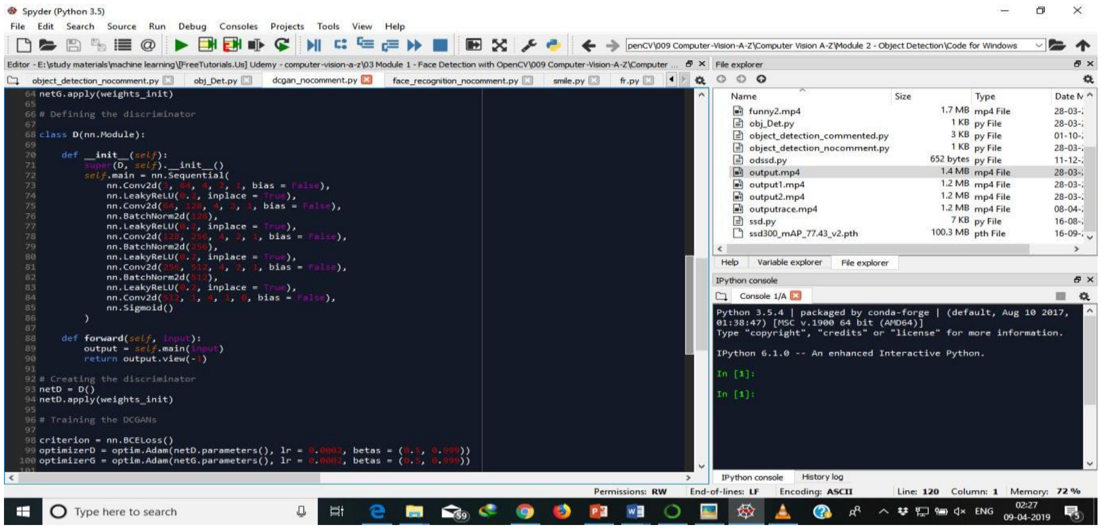
i) We have implemented it using python 3.5 in spyder on anaconda framework.
ii) Used AMD graphics card for training neural network
iii) Code is shown below,



```
from torch.autograd import Variable

# Setting some hyperparameters
batchSize = 64 # We set the size of the batch.
imageSize = 64 # We set the size of the generated images (64x64).

# Creating the transformations
transform = transforms.Compose([transforms.Scale(imageSize), transforms.ToTensor(), transforms.Normalize((0.5, 0.5,

# Loading the dataset
dataset = dset.CIFAR10(root = './data', download = True, transform = transform) # We download the training set in t
dataloader = torch.utils.data.DataLoader(dataset, batch_size = batchSize, shuffle = True, num_workers = 2) # We use

# Defining the weights_init function that takes as input a neural network m and that will initialize all its weight
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        m.weight.data.normal_(0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)

# Defining the generator

class G(nn.Module):

    def __init__(self):
        super(G, self).__init__()
        self.main = nn.Sequential(
            nn.ConvTranspose2d(100, 512, 4, 1, 0, bias = False),
            nn.BatchNorm2d(512),
            nn.ReLU(True),
            nn.ConvTranspose2d(512, 256, 4, 2, 1, bias = False),
            nn.BatchNorm2d(256),
            nn.ReLU(True),
            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias = False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
```

```
Python 3.5.4 | packaged by conda-forge | (default, Aug 10 2017,
01:38:47) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.1.0 -- An enhanced Interactive Python.

In [1]:

In [1]:
```

Fig 4.6



```
netG.apply(weights_init)

# Defining the discriminator

class D(nn.Module):

    def __init__(self):
        super(D, self).__init__()
        self.main = nn.Sequential(
            nn.Conv2d(3, 64, 4, 2, 1, bias = False),
            nn.LeakyReLU(0.2, inplace = True),
            nn.Conv2d(64, 128, 4, 2, 1, bias = False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace = True),
            nn.Conv2d(128, 256, 4, 2, 1, bias = False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace = True),
            nn.Conv2d(256, 512, 4, 2, 1, bias = False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace = True),
            nn.Conv2d(512, 1, 4, 1, 0, bias = False),
            nn.Sigmoid()
        )

    def forward(self, input):
        output = self.main(input)
        return output.view(-1)

# Creating the discriminator
netD = D()
netD.apply(weights_init)

# Training the DCGANs

criterion = nn.BCELoss()
optimizerD = optim.Adam(netD.parameters(), lr = 0.0002, betas = (0.5, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr = 0.0002, betas = (0.5, 0.999))
```

```
Python 3.5.4 | packaged by conda-forge | (default, Aug 10 2017,
01:38:47) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.1.0 -- An enhanced Interactive Python.

In [1]:

In [1]:
```

Fig 4.7

iv) Output images are shown below

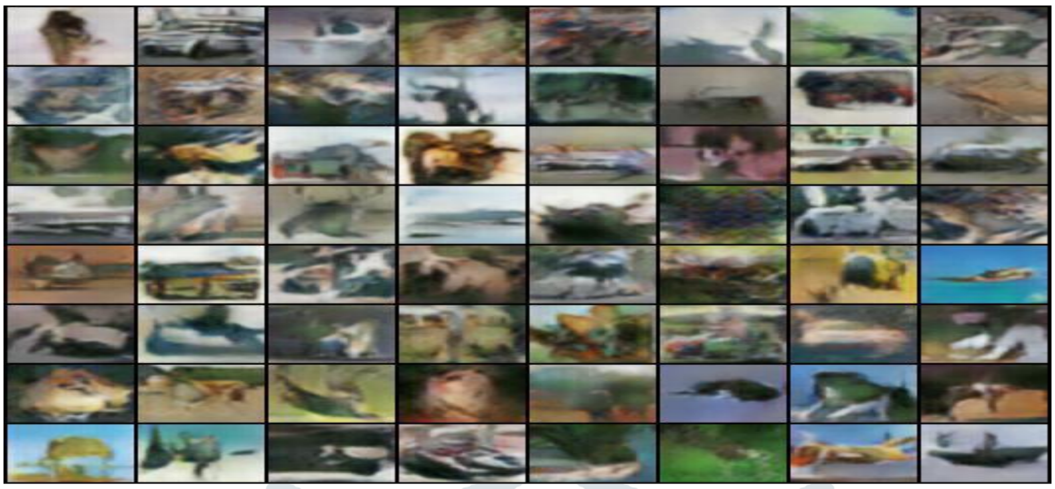


Fig 4.8

Conclusion

After implementing all three frameworks we came to conclude each of these three as:

Viola Jones Algorithm:

We have presented an approach for object detection which minimizes computation time while achieving high detection accuracy. The approach was used to construct a face detection system which is approximately 15 times faster than any previous approach. But it has some demerits like it can't work properly with video streaming frames. So, for this we have implemented SSD algorithm.

SSD: Single shot MultiBox detection framework:

A key feature of our model is the use of multi-scale convolutional bounding box outputs attached to multiple feature maps at the top of the network. This representation allows us to efficiently model the space of possible box shapes. Unlike CNN it can detect whole image in single shot. So, it saves our computation time.

But it hasn't any feature of generating a random image. So for this feature we used GANs.

Generative Adversarial Network

Although GAN has several advantages but it is the best network for generating random images in less amount of time. It can also be used for making AI chatbots and increasing or decreasing resolution of an image with most efficient way.

REFERENCES

1	W Liu, D Anguelov, D Erhan, C Szegedy… - European conference on …, 2016 – Springer Ssd: Single shot multibox detector.
2	CP Papageorgiou, M Oren, T Poggio - Sixth International Conference …, 1998 - acius.co.uk , A general framework for object detection
3	K Tieu, P Viola - International Journal of Computer Vision, 2004 – Springer Boosting image retrieval
4	P Viola, M Jones - CVPR (1), 2001, Rapid object detection using a boosted cascade of simple features
5	A Radford, L Metz, S Chintala - arXiv preprint arXiv:1511.06434, 2015 - arxiv.org, Unsupervised representation learning with deep convolutional generative adversarial networks
6	I Goodfellow, J Pouget-Abadie, M Mirza… - Advances in neural …, 2014 - papers.nips.cc, Generative adversarial nets
7	https://www.superdatascientist.com/
8	https://www.github.com/
9	https://www.anaconda.org/
10	http://host.robots.ox.ac.uk/pascal/VOC/