18CSE353T

DIGITAL IMAGE PROCESSING

MINI PROJECT

PROJECT NAME:

Development of a hybrid model for Automatic Weapon Detection/Anomaly Detection in Surveillance Applications

GROUP MEMBERS:

RA1811003010680 - VAIDYA VISHAL
RA1811003010688 - T PRAVANTH

ABSTRACT

Adoption of CCTV cameras for security purposes is growing around the world. Surveillance methodologies are one of the fast-growing fields in security systems. Monitoring systems play an important role in detecting abandoned objects, anomaly detection, suspect identification etc. Increase in demand for advanced systems led to the development of new surveillance methodologies.

Newer technologies in artificial intelligence have helped us to create smart and automated security measures. Since there has been an increase in the amount of weapon related violence, one of the main factors in safety measures is protection against weapons.

This project aims to develop a hybrid model (designing with Image Processing Techniques and Deep Learning Models) to automatically check images to detect weapons such as knives and guns. If it detects a weapon, alerts the human personal, so that he can focus his attention on the particular footage and take immediate action as fast as possible. It will be experimented and validated to achieve state of art results.

The project work uses YOLO (You Only Look Once) object detection system which uses convolution neural networks for object detection. It is one of the faster algorithms that performs without much degradation in accuracy.

The training of this model has been done on the cloud to save hundreds of hours of GPU time on a local runtime.

LITERATURE SURVEY

Several weapon detection algorithms have been developed and studied over years. Each of them consists of different types of methods and techniques and has its own pro and con. However, all of them have the same goal which is trying to enhance the existing method or create a new method to achieve a better result.

G. K. Verma and A. Dhillon proposed a R-CNN based method for handheld gun detection using a pre-trained VGG model. In the segmented images, Fast Retina Keypoint (FREAK) and Harris Interest Point Detector are used to find the weapons. Testing was done on a dataset built from the Internet Movie Firearm Database (IMFDB). The model could detect and classify three types of guns namely revolvers, rifles, and shotguns. However, for this method to detect guns, it has to be held by humans and not otherwise. The visual gun detection system using SIFT (Scale Invariant Feature Transform) and Harris interest point detector was proposed which utilized color-based segmentation to take out a distinct object from an image using K-Means clustering algorithm.

Grega et al. proposed a method for automatic detection of dangerous situations in CCTV systems, through the use of image processing and machine learning. Firearms and knives were detected in video using sliding window techniques, fuzzy classifiers and canny detectors. The dataset and detection system constructed by the authors were made available.

FEATURE EXTRACTION

The project employs You Only Look Once (YOLO) v3 model, which is a deep learning framework based on Darknet, an open-source neural network in C. YOLOv3 is the best choice as it provides real-time detection without losing too much accuracy. The architecture used is darknet53 which consists of 53 convolutional layers each followed by Leaky ReLu activation and batch normalization layers, making it a fully convolutional network (FCN).

For the task of detection, the total layers used are 106 which makes the model bulkier than its previous variants. The model doesn't use pooling to prevent loss of low-level features. Also, the unsampled layers are concatenated with the previous layers to help detect small objects by preserving the minute features. Unlike the sliding window and region proposal- based techniques, YOLO detects objects in an image very well as it gets every detail about the whole image and the object by seeing the entire image.

YOLOv3 algorithm uses the Feature Pyramid Networks (FPN) idea to idea to achieve multi-scale prediction and uses deep residual network (ResNet) ideas to extract achieve multi-scale prediction and uses deep residual network (ResNet) ideas to extract image image features to achieve a certain balance between detection speed and detection accuracy.

TRAINING THE YOLOv3 MODEL

The YOLOv3 model was entirely trained on Google Colab using Python 3 and NVidia GPU Backend.

Dataset used in this Project is Downloaded from Kaggle, this dataset contains more than 4000 images. The dataset can be viewed/downloaded from here.

After we have downloaded the Dataset, we can now proceed to train our YOLOv3/Darknet model.

The python notebook in which we trained our model can be viewed/Downloaded from here.

The first step in training the model is to clone into the darknet repository on GitHub, which can be achieved using the command:

```
!git clone https://github.com/AlexeyAB/darknet
```

After cloning the repository, we build the Darknet using the commands:

```
# change makefile to have GPU, OPENCV, and CUDNN enabled
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
# build darknet
!make
```

The second step is to define a helper function, as Colab doesn't directly support cv2.imshow() method, which can be done using:

```
def imShow(path):
 import cv2
 import matplotlib.pyplot as plt
 %matplotlib inline
```

```
 image = cv2.imread(path)
 height, width = image.shape[:2]
 resized_image = cv2.resize(image,(3*width, 3*height), interpolation
= cv2.INTER_CUBIC)
 fig = plt.gcf()
 fig.set_size_inches(18, 10)
 plt.axis("off")
 plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
 plt.show()
```

After this, we connect to our Google Drive using the commands:

```
%cd ..
from google.colab import drive
drive.mount('/content/gdrive')
# this creates a symbolic link so that now the path
/content/gdrive/My\ Drive/ is equal to /mydrive
!ln -s /content/gdrive/My\ Drive/ /mydrive
```

In the next step, we train the model using the dataset and YOLOv3 configurations files, There are essentially 3 YOLOv3 configuration files: obj.names, obj.data, yolov3_custom.cfg.

These 3 files can be viewed/downloaded from here.

obj.data file contains the data, such as the number of classes, the location of the train.txt (which holds the location of all training images), location of the test.txt (which holds the location of all test images), location of obj.names (which holds the names of all the classes) and a backup file location (to backup the weights of the YOLOv3 model)

obj.names contains the name of all the classes

yolov3_custom.cfg contains all the parameters, such as number of classes, max number of iterations,.etc, which is required to train the model. More information on how to create a custom training yolov3 configuration file can be found here.

We then extract the dataset downloaded using the command:

```
!unzip ../obj.zip -d data/
```

After this we copy the remaining 3 config files, using the command:

```
# uploading the custom .cfg back to colab from Google Drive
!cp /mydrive/yolo-2/yolov3_custom.cfg ./cfg
# uploading the obj.names and obj.data files to colab from Google
Drive
!cp /mydrive/yolo-2/obj.names ./data
!cp /mydrive/yolo-2/obj.data  ./data
```

We also create a train.txt which holds the relative paths to all our training images, using the following python code:

```
import os

image_files = []
os.chdir(os.path.join("data", "obj"))
for filename in os.listdir(os.getcwd()):
    if filename.endswith(".jpg"):
        image_files.append("data/obj/" + filename)
os.chdir("..")
with open("train.txt", "w") as outfile:
    for image in image_files:
        outfile.write(image)
        outfile.write("\n")
    outfile.close()
os.chdir("..")
```

In the next step, we Download a pre-trained weights for the training of our convolutional layers, which can be done using,

```
!wget http://pjreddie.com/media/files/darknet53.conv.74
```

Next, we train our custom model on the existing pre-trained weights file, using the command,

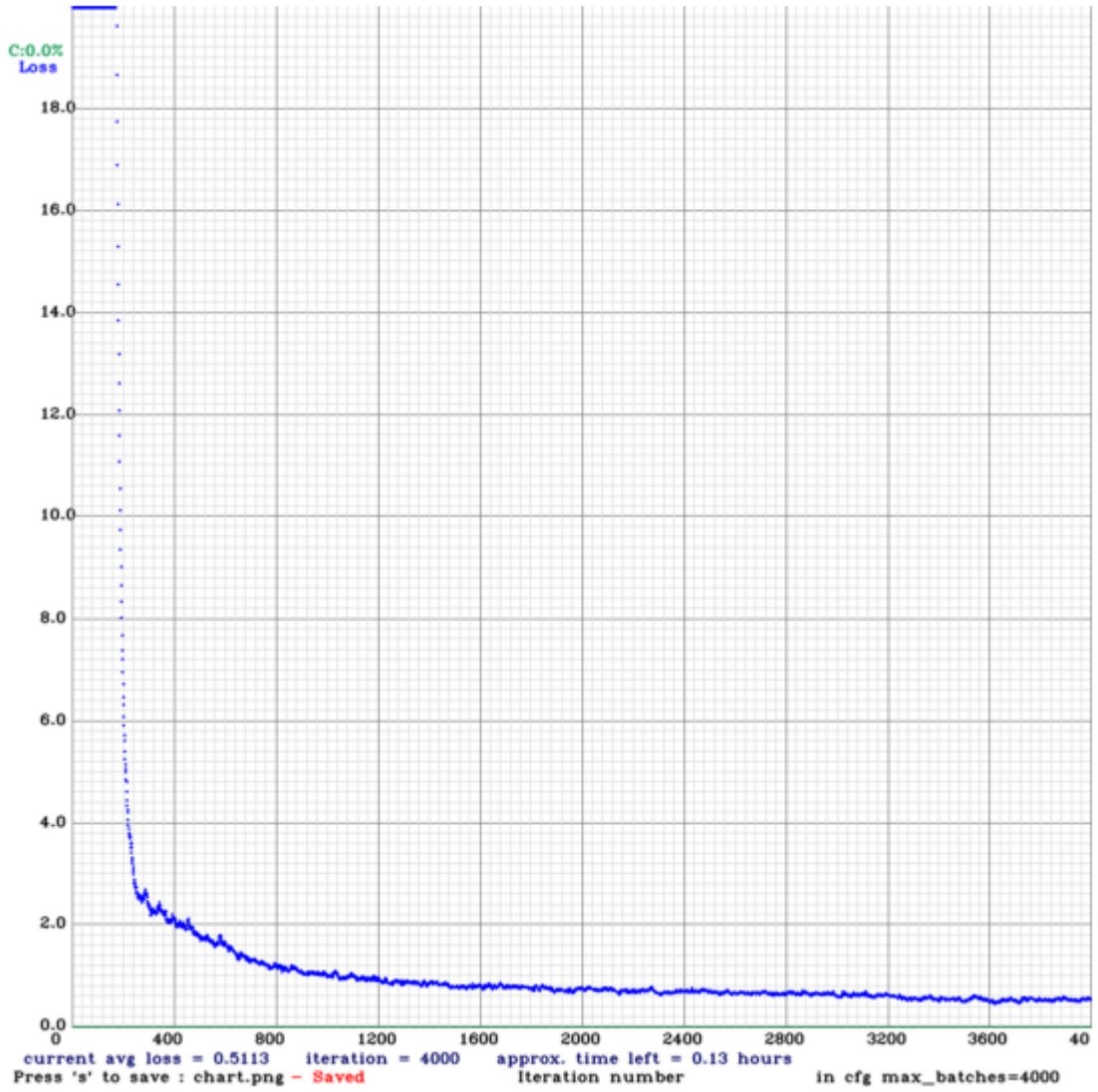
```
!./darknet detector train data/obj.data cfg/yolov3_custom.cfg
darknet53.conv.74 -dont_show
```

The training will take several hours depending on the CPU/GPU resources.

After the training we can view the YOLOv3 loss chart using the command:

```
imShow('chart.png')
```

After training the model, we obtained the following loss graph.



The trained YOLOv3 weights file can be downloaded from here.

TESTING THE YOLOv3 MODEL

First we need to set our custom configuration file to test mode, which can be done using the command:

```
%cd cfg
!sed -i 's/batch=64/batch=1/' yolov3_custom.cfg
!sed -i 's/subdivisions=16/subdivisions=1/' yolov3_custom.cfg
%cd ..
```

Now we test our model on the images provided, the images i used to test can be viewed/downloaded from here. The command to test the model on image is:

```
#Testing the model on a gun image
!./darknet detector test data/obj.data cfg/yolov3_custom.cfg
/mydrive/yolo-2/backup/yolov3_custom_last.weights
/mydrive/yolo/images/gun.jpg
imShow('predictions.jpg')
```

The result which was obtained on the gun image is,

Test the model on a low resolution image gives the following result,

Testing the model on a CCTV video:

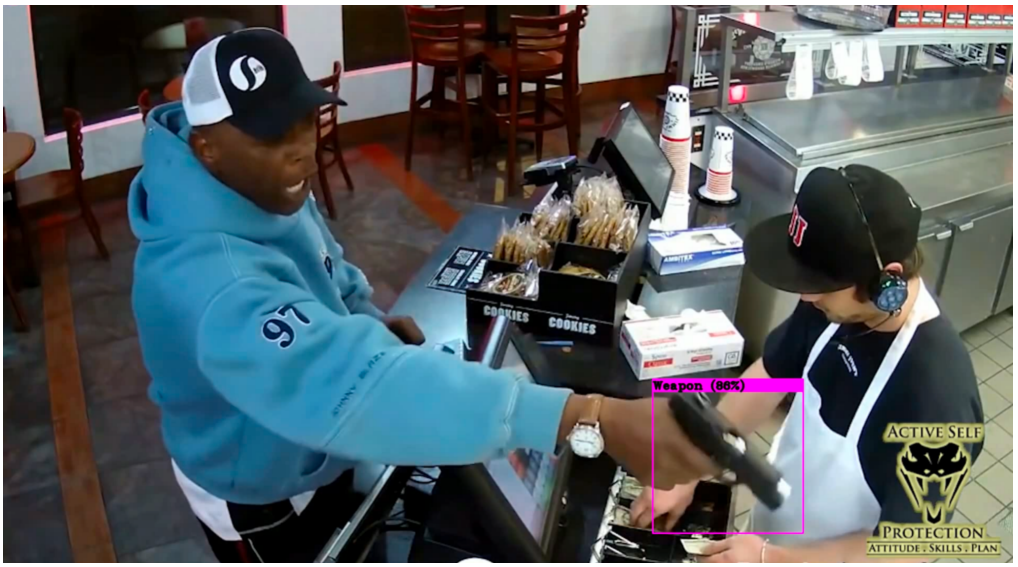
We use the following command to test the model on a video:

```
!./darknet detector demo data/obj.data cfg/yolov3_custom.cfg
/mydrive/yolo-2/backup/yolov3_custom_last.weights -dont_show
/mydrive/yolo/images/cctv1.mp4 -i 0 -out_filename
/mydrive/yolo-2/images/result1.avi -thresh 0.4
```

Note: we use -thresh to set the threshold for the test image/video.

The test images and cctv videos we used and their result using the model can be found here.

Some frames from the resultant CCTV video file:



RESULT

A YOLOv3 model to detect weapons is created successfully using Google Colab, using Python and a GPU Backend.

This model successfully detected weapons in an image and also in a CCTV video file.

Although this model isn't perfect, it does detect weapons in a HD image and also on a SD image, it sometimes fails to detect weapons in a LD image.

It also recognizes weapons in a video file as well, but this model also detects objects which are not weapons. This can be solved by using a good dataset, which differentiates between objects of different types.

This model took around 5 hours to train using a 4000+ images dataset. Considering the time and work it took to train this model, the results are pretty impressive.

The github link for the project can be found here.