OBJECT RECOGNITION: REAL TIME PERSON DETECTION WITH DEEP LEARNING

Pan Yongjing[1], Wen Dequan[2]
[1]Raffles Girls' School (Secondary), 2 Braddell Rise, Singapore 318871
[2]Defence Science and Technology Agency, 1 Depot Road, Singapore 109679

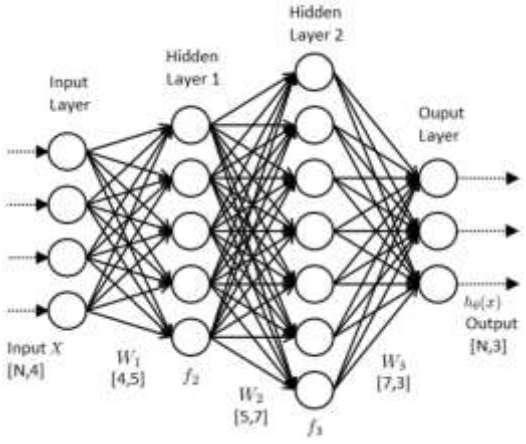
Abstract

A pre-trained object detection deep neural network model was used in creating a security device capable of recognising objects from live video footage. This device notifies security personnel by email and sends them a picture of the scene when person(s) are detected in a location. For the duration in which persons are present in the frame, the device also records footage which is later uploaded to a Google Drive folder, and a link is provided for security personnel to access it. It may be used as an extra security measure for surveillance in locations where access is restricted, such as the DSTA Data Centre. The device can also count the number of people present in the frame, which may serve another purpose of aiding security officers in monitoring traffic in common entry and exit points of a building.

Introduction

This project explores object recognition, one of the applications of deep neural networks that is increasingly relevant in our modern world. Neural networks are used in computer vision to allow programs to achieve a high-level understanding of digital images and videos, through detecting, recognising and classifying objects or scenes. A deep neural network, used in this project, consists of multiple layers between an input and output layer.

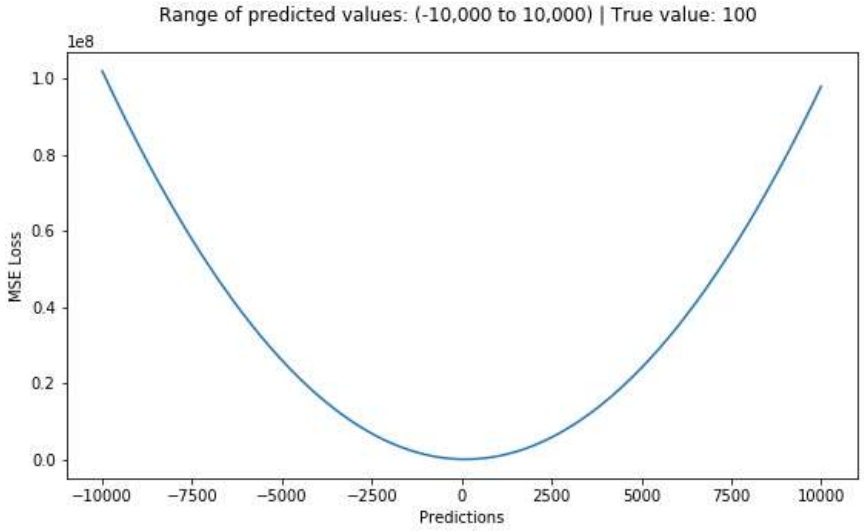
Figure 1: Architecture of a deep neural network
(from Neural Networks and Deep Learning, by Michael Nielsen)



Each layer consists of multiple neurons, and each neuron receives an output from neurons of the previous layer and produces another output, affected by two values, its weight and bias. Each neuron's output is then fed to neurons of the next layer. The output from the last layer is then fed to a loss function, which calculates a loss value comparing the output to a correct target value. A

lower loss indicates a more accurate output closer to the target value, and vice versa. The overall function consisting of all the weights and biases of the network, that maps input to an output and a corresponding loss, may be visualised as a graph with minimum and maximum points. [1][2]

Figure 2: Mean Squared Error Loss Functions



An optimiser algorithm then calculates through backpropagation, how to adjust the network's parameters, to reduce loss and improve the model's accuracy. In other words, the optimiser function adjusts the network's weights and biases such that the loss function is at a minimum point. A commonly used optimiser algorithm is gradient descent, which continuously moves in the direction of steepest descent, which is defined as the negative of the gradient.

Convolutional neural networks (CNNs) are able to assign importance (learnable weights and biases) to various features or objects in the image so as to differentiate one from the other. Thus, the network learns filters and characteristics over time, allowing it to capture spatial and temporal dependencies in an image. Individual neurons respond to stimuli in specific areas in an image. The areas responded to by all the neurons would then overlap to cover the entire image.

A CNN learns features of an image with kernels, matrices that act as a filter applied on portions of the entire image. The network continuously performs matrix multiplication between the kernel K and a portion P of the image it is hovering over, until it covers the entire image. The kernel thus stores a convoluted image of the original, allowing it to extract features of the image. A pooling layer is essential in reducing the spatial size of the convolved feature, thus reducing computational power necessary to process the data. Usually this is done with a max pooling layer, which returns the maximum value from a portion of the image covered by the kernel.

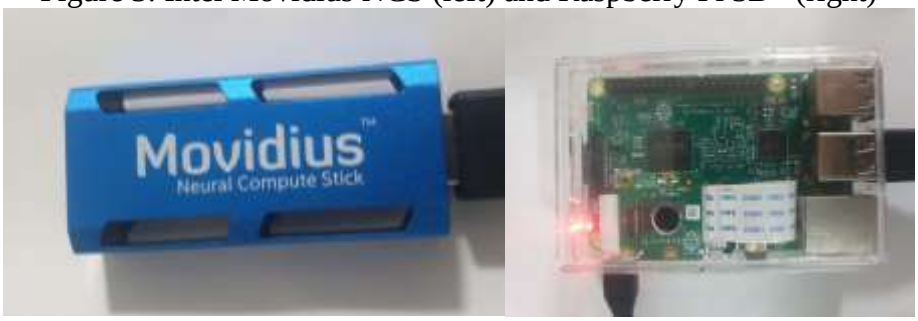
A convolutional and pooling layer usually comes one after another in the CNN, and many such layers may be stacked after one another to deepen the network's understanding of an image. After the convolutional and pooling layers, a fully connected layer is used to classify the input image based on certain categories. This layer allows the network to learn a nonlinear function that consists of combinations of high level features represented as output from the convolutional layer.

Object detection has a wide variety of applications such as in robotics, autonomous driving and human-computer interaction. This project focuses on the use case in security and surveillance. Many organisations may have places where access is restricted to only a few authorised personnel, and is prohibited for most people. Thus, a camera with person-detection capabilities would be able to add an extra layer of security along with other forms of surveillance such as infrared sensors and motion detectors.

Materials and Methods

In this project, we made use of a couple of hardware and software. The hardware includes the Intel Movidius Neural Compute Stick (NCS), with the Myriad 2 Processor that provides the computing power to run our neural net and the Raspberry Pi 3B+, which provides the platform for writing and executing our program. All the code used and written in this project is written in Python3, while making use of the OpenCV2 library and modules such as Matplotlib, Numpy and Imutils.

Figure 3: Intel Movidius NCS (left) and Raspberry Pi 3B+ (right)



The pretrained object recognition model used in this project is an open-source Caffe implementation of a Mobilenet SSD (single-shot detector) by chuanqi305 on Github, with pretrained weights on VOC0712, which is an image dataset that is a training-validation split of the PASCAL VOC 2007 and 2012 datasets. This model has a mean average precision (mAP) of 0.727. Mobilenet refers to a neural network that can recognise and classify objects, while SSD refers to the multibox detector which detects the bounding boxes of objects.

This Mobilenet SSD is able to recognise 21 classes of objects, but for this project we only require its ability to recognise the ‘person’ class. As the model is implemented with Caffe, the model files consist of a ‘.caffemodel’ file and a ‘.prototxt’ file, which contain the weights and structure of the neural network.

This model is then used in a Python script that executes the entire program. First, the necessary modules are imported. Some essential modules include imutils’ video streaming and processing functions, open cv2, for image processing, smtplib, for the sending of emails.

Figure 4: Modules Imported in Main Python Script

```
from __future__ import print_function
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import math
import imutils
import time
import cv2
import smtplib
import datetime
import shutil

import sys
import os
```

Furthermore, two other scripts, pymail.py and up.py are imported for the purposes of sending emails and uploading videos to a Google folder respectively.

Figure 5: Secondary Python Scripts Included

```
sys.path.append(os.path.abspath("/home/pi/securepibot/pymail/"))
import pymail #script for sending email

sys.path.append(os.path.abspath("/home/pi/securepibot/"))
import up #script for uploading video to google drive
```

After importing the modules, the Mobilenet SSD model is read and run with the OpenCV library function cv2.dnn.readNetFromCaffe in Python, with the processor set to Myriad 2 of the NCS.

Figure 6: Reading of Model Files

```
# load our pre-trained model from disk
print("[INFO] loading model...")
modelfile = 'mns.caffemodel'
protofile = 'mns.prototxt'
net = cv2.dnn.readNetFromCaffe(protofile, modelfile)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_MYRIAD)
```

The Raspberry Pi camera then starts receiving video input, with each frame passed as input to the model. The CV2 library functions cv2.VideoStream.start() and cv2.VideoWriter_fourcc() were used to capture and write video data respectively and cv2.imwrite was used to write image data of the screenshot.

Figure 7: Initialisation of Video Stream

```
# initialise the video stream and FPS counter
print("[INFO] starting video stream...")
vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)
fps = FPS().start()
```

Figure 8: Saving of Screenshot

```
if present and writer is None:
    print("[ALERT!] Person present!")

    cv2.imwrite(curdir+'img.jpg', frame)
    print("[IMG] Screenshot saved")
```

When the model detects a person with a confidence value higher than the minimum confidence of 0.5, an email is sent to the user, notifying them that person(s) are present in the scene, along with a screenshot of the frame. At the same time, the script starts writing the video footage to a new file. After that, when the program does not detect persons in the frame for more than 10 seconds, it stops recording and the video file is uploaded to a Google Drive folder. After the video has uploaded, the program notifies the user via email so the file may be accessed.

Figure 9: Email notifying user person has been detected

The sending of emails, in the ‘pymail.py’ script, was done with Python’s email and smtplib modules. When this script is included in ‘exec.py’, some initialisation steps are first performed. First, the emails to which the alerts will be sent to are read from ‘contacts.txt’.

Figure 10: Reading of Contacts

```
curdir = '/home/pi/securepibot/pymail/'
maindir = '/home/pi/securepibot/'

with open(curdir+'mycontacts.txt', mode='r', encoding='utf-8') as contacts_file:
    for a_contact in contacts_file:
        names.append(a_contact.split()[0])
        emails.append(a_contact.split()[1])
```

Then, the script performs a login to a Google account “secureaccessbot@gmail.com”, from which emails will be sent. This was done by establishing a smtp connection to smtp.gmail.com (port 857) and logging in with the email and password of the account.

Figure 11: Login to Google Account

```
s = smtplib.SMTP(host='smtp.gmail.com', port=587)
s.starttls()
s.login(MY_ADDRESS, PASSWORD)
print("Email login done")
```

Two types of alerts can be sent. One type is immediate: it notify the user when a person has been detected, and sends a screenshot of the frame. The second type is sent when a video has been uploaded to the drive, which is after a person is no longer detected for more than 10 seconds, that contains the link to the folder with the uploaded video. The type of alert is indicated by calling the loadalert function in pymail, which can be done in the main program 'exec.py' as if using a library function, with the line pymail.loadalert(num), where num is the type of alert.

Figure 12: Loadalert Function

```
def loadalert(num):
    global message_template, curdir
    if num == 1:
        with open(curdir + 'message.txt', 'r', encoding='utf-8') as template_file:
            message_template = Template(template_file.read())
    else:
        with open(curdir + 'message2.txt', 'r', encoding='utf-8') as template_file:
            message_template = Template(template_file.read())
```

After the type of alert is loaded, 'exec.py' calls pymail.sendmail(), which sends an email to each contact. Each email is put together with a MIMEMultipart() object. "From", "To", "Subject" parameters are specified for this object, then the corresponding message is attached, along with a screenshot, which has been converted to suitable object 'part'.

Figure 13: Sendmail Function

```
def sendmail():
    global names, emails, message_template, s
    part = getfile()
    # For each contact, send the email:
    for name, email in zip(names, emails):
        msg = MIMEMultipart()       # create a message
        # add in the actual person name to the message template

        message = message_template.substitute(PERSON_NAME=name.title())

        # Prints out the message body for our sake
        print("Message: ", message)

        # setup the parameters of the message
        msg['From']=MY_ADDRESS
        msg['To']=email
        msg['Subject']="[Security Alert!] Person(s) detected in data centre"

        # add in the message body
        msg.attach(MIMEText(message, 'plain'))

        msg.attach(part)

        s.send_message(msg)
```

In order to upload videos to Google Drive, the Google Drive API library was utilised to connect the RPi to a Google Account.

Figure 14: Importing of Google Drive API Libraries

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from pydrive.files import GoogleDriveFileList
import googleapiclient.errors
```

An initialisation function is first run before files can be uploaded, where the program calls for the login to Google to be authenticated, and gets the IDs of the parent folder and the destination folder where files will be uploaded to.

Figure 15: Initialisation Function

```
def init():
    global drive, parent_folder_id, folder_id
    drive = authenticate()
    # Get parent folder ID
    parent_folder_id = get_folder_id(drive, 'root', parent_folder_name)
    # Get destination folder ID
    folder_id = get_folder_id(drive, parent_folder_id, dst_folder_name)
```

Figure 16: Authentication Function

```
def authenticate():
    print(" [INFO] Authenticating Google account... ")

    gauth = GoogleAuth()

    return GoogleDrive(gauth)
```

Figure 17: Get Folder ID Function

```
def get_folder_id(drive, parent_folder_id, folder_name):

    # Auto-iterate through all files in the parent folder.
    file_list = GoogleDriveFileList()
    try:
        file_list = drive.ListFile(
            {'q': "'{0}' in parents and trashed=false".format(parent_folder_id)}
        ).GetList()
    # Exit if the parent folder doesn't exist
    except googleapiclient.errors.HttpError as err:
        # Parse error message
        print(type(err))
        message = ast.literal_eval(err.content) ##['error']['message']
        if message == 'File not found: ':
            print(message + folder_name)
            exit(1)
        # Exit with stacktrace in case of other error
        else:
            raise

    for filel in file_list:
        if filel['title'] == folder_name:
            print('title: %s, id: %s' % (filel['title'], filel['id']))
            return filel['id']
```

Then, files may be uploaded from the RPi's local directory "video_footage" to the online folder.

Figure 18: Uploading of Files Function

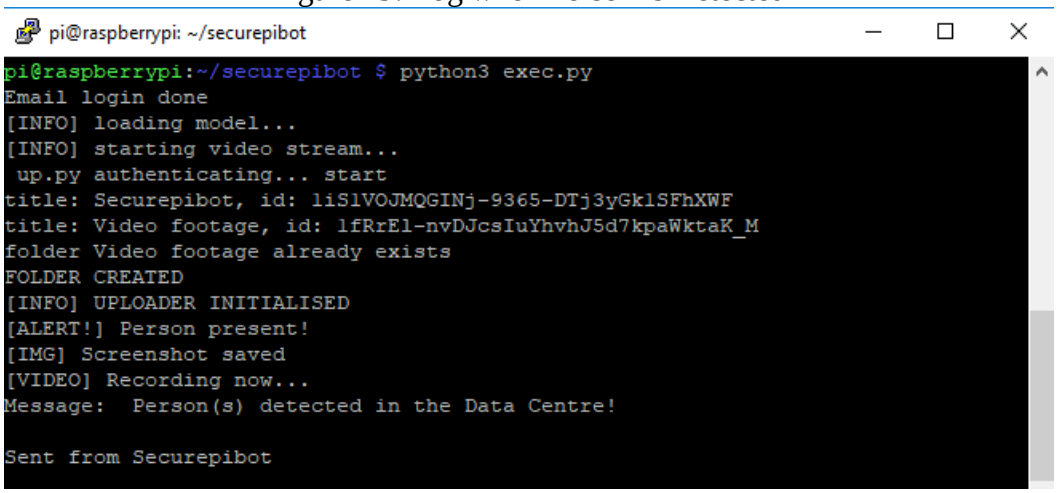
```
def upload_files(drive, folder_id, src_folder_name):

    # Enter the source folder
    try:
        chdir(src_folder_name)
    # Print error if source folder doesn't exist
    except OSError:
        print(src_folder_name + 'is missing')
    # Auto-iterate through all files in the folder.
    for filel in listdir('.'):
        # Check the file's size
        statinfo = stat(filel)
        if statinfo.st_size > 0:
            print('uploading ' + filel)
            # Upload file to folder.
            f = drive.CreateFile(
                {"parents": [{"kind": "drive#fileLink", "id": folder_id}]})
            f.SetContentFile(filel)
            f.Upload()
        # Skip the file if it's empty
        else:
            print('file {0} is empty'.format(filel))

    def uploadnow():
        upload_files(drive, folder_id, src_folder_name)
        print("FILES UPLOADED")
```

Results

The final product is a device consisting of the RPi, connected to the Intel Movidius NCS and a power supply. It requires an internet connection for the sending of email notifications and uploading of video files. The program is initialised by running the Python script exec.py in the directory containing all the program files, /home/pi/securepibot. After the Python script is called, the program initialises the email login, Google login, loads the model and starts the video stream.

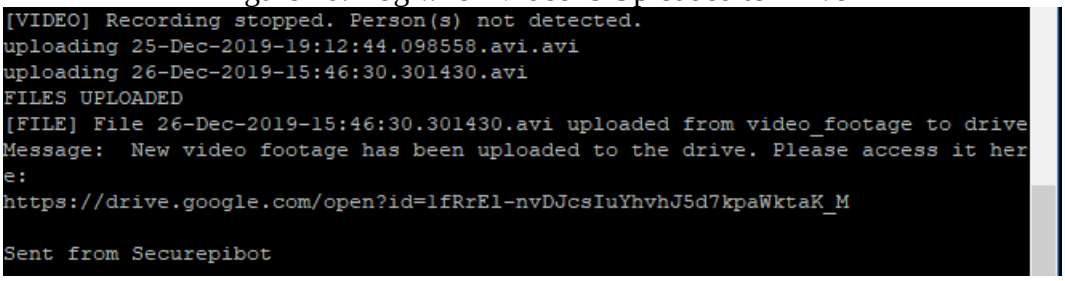
Below is the log of an instance when the program detects a person. A screenshot is taken and immediately sent, along with an email alert to the relevant users.

Figure 19: Log when Person is Detected



When a person is no longer detected for 10 seconds, the program stops recording and uploads a video to the Google Folder, and sends an email alert to the user so it may be accessed.

Figure 20: Log when Video is Uploaded to Drive



A log file is generated on every execution of the program, giving a summary of the timestamps when persons are detected and the number of persons detected.

Figure 21: Log.txt

```
27-Dec-2019-13:41:28.445603 to 27-Dec-2019-13:41:28.445603: 0 persons detected
27-Dec-2019-13:41:28.445603 to 27-Dec-2019-13:41:35.621099: 1 persons detected
27-Dec-2019-13:41:35.621099 to 27-Dec-2019-13:41:35.621099: 0 persons detected
```

Discussion

Object recognition has a wide variety of applications in improving people's lives, improving security systems, education and other areas. This project only scratches the surface of all the possible use cases of convolutional neural networks in recognising objects.

Though this device has achieved the basic task of person detection and alerting users nearly immediately when a person has been detected, it is not without imperfections. There is a delay of 3 to 4 seconds when the video file is being uploaded to the drive, during which the video stream stops and the device is temporarily unable to detect anything. It is also necessary for the RPi to be plugged into a power source when in use, which limits its portability. However, this should not be a major disadvantage when power sources are readily available.

An idea to develop this project further is to integrate face recognition, so that the device may emulate human vision on a higher level. Some other possible research topics are, investigating the different available pre-trained models and their performance on the Neural Compute Stick.

Acknowledgement

I am grateful to my mentor Dequan, for all the help and support he has given me on this project, and for his guidance and expertise that pointed me in the right direction when I met with obstacles. Without him, this project would not have been possible.

References

[1] Chollet François. 2018. Deep learning with Python. Shelter Island, NY: Manning Publications Co.

[2] Goodfellow, I., Bengio, Y., & Courville, A. 2017. Deep learning. Cambridge, MA: The MIT Press.

[3] Redmon, J., & Angelova, A. 2015. Real-time grasp detection using convolutional neural networks. 2015 IEEE International Conference on Robotics and Automation (ICRA). doi: 10.1109/icra.2015.7139361

[4] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. 2016. You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi: 10.1109/cvpr.2016.91

[5] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.