Bachelor’s thesis

Bachelor of Engineering, Information and Communications Technology

2022

Tino Nummela

OBJECT DETECTION WITH NVIDIA JETSON NANO

Bachelor's Thesis | Abstract

Turku University of Applied Sciences

Information and communication Technology

2022 | 32, 4

Tino Nummela

OBJECT DETECTION WITH NVIDIA JETSON NANO

Object detection on embedded devices is becoming increasingly more popular in industrial use, as well as in individual use. Object detection can be used in many useful ways, such as security devices, quality assurance devices and many more.

Object detection is a powerful technology to detect object in either images, videos or live video feed. This thesis is introducing the reader in to the world of object detection using Nvidia Jetson Nano as a sole developing environment. Unravel the capabilities of Nvidia Jetson Nano to be used for such object detection applications. Introduce the tools and technologies needed to deploy object detection application on embedded device and also demonstrate of such deployment from ground up.

The reader is also introduced to the outcoming data of such application, meaning of the data and using the data to solve a problem. Also, to consider possible alternative usages of example implementation application.

Keywords:

Object detection, embedded device, Nvidia Jetson Nano

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2022 | 32 sivua, 4 liitesivua

Tino Nummela

ESINEENTUNNISTUS NVIDIA JETSON NANOLLA

Esineentunnistus sulautetuilla laitteilla on tulossa yhä suositummaksi teollisessa käytössä, kuten myös henkilökohtaisessa yksityisessä käytössä. Esineentunnistusta voidaan käyttää monella eri hyödyllisellä tavalla, kuten turvalaitteissa, laadunvarmistuslaitteissa ja lukuisilla muilla tavoilla.

Esineentunnistus on tehokas työkalu esineiden tunnistamiseen kuvissa, videoissa ja videon suoratoistossa. Opinnäytetyö käsitteli esineentunnustusta käyttämällä Nvidia Jetson Nanoa ainoana kehitysympäristönä. Opinnäytetyössä selvitettiin Nvidia Jetson Nanon kykyjä ja ominaisuuksia suorittaa esineentunnistussovellus. Opinnäytetyössä esitettiin työkalut ja teknologiat, joita tarvitaan esineentunnistussovelluksissa sulautetuissa laitteissa, sekä demonstroitiin tällaisen sovelluksen käyttöönotto alusta loppuun.

Opinnäytetyössä tutustuttiin myös tällaisen sovelluksen tuottamaan tietoon, tiedon merkitykseen ja tiedon hyödyntämiseen.

Asiasanat:

Esineentunnistus, sulautettu laite, Nvidia Jetson Nano

CONTENTS

List of abbreviations (or) symbols **7**

1 INTRODUCTION **9**

2 THEORETICAL BACKGROUND **10**

2.1 Nvidia Jetson Nano Developer Kit 10

2.1.1 JetPack and Jetson-inference 11

2.2 Python 12

2.3 SSD-Mobilenet 13

2.3.1 Re-training (Transfer learning) 14

2.3.2 Converting trained model 15

3 IMPLEMENTATION **16**

3.1 Setting up Jetson Nano 16

3.1.1 Flashing Jetpack onto SD Card 16

3.1.2 Initial setup of the Jetpack with Display 18

3.1.3 Setting up Jetson-inference 18

3.1.4 Editor (IDE) 19

3.1.5 Mounting Swap Memory 19

3.2 Gathering data 20

3.2.1 Data Format (Pascal VOC, image/annotation) 22

3.3 Re-training SSD-Mobilenet 23

3.3.1 Resume training 23

3.4 Exporting model 24

3.5 Python program 24

4 TESTING **28**

4.1 Running the program 28

4.2 Performance and Model accuracy 29

5. CONCLUSIONS **30**

REFERENCES **31**

APPENDICES

Appendix 1. Source code of example implementation

FIGURES

Figure 1. Nvidia Jetson Nano. – Source: https://developer.nvidia.com/ 11
Figure 2. Example of Python syntax. 12
Figure 3. SSD-Mobilenet layers. -- Source: https://arxiv.org/abs/1512.02325 13
Figure 4. Transfer learning vizualized. – Source: https://www.topbots.com/transfer-learning-in-nlp/ 14
Figure 5. ONNX logo. – Source: https://en.wikipedia.org/wiki/Open_Neural_Network_Exchange 15
Figure 6. SD Card Formatter interface. 17
Figure 7. Etcher interface. 17
Figure 8. Cloning and runnin jetson-inference container. 18
Figure 9. Mounting swap memory on Nvidia Jetson Nano. 19
Figure 10. Runnin Jetson-inference docker container. 20
Figure 11. Running data collectiong software. 20
Figure 12. Correct data directory. 20
Figure 13. Example of data collection into training dataset. 21
Figure 14. Example of a bounding box annotation in XML format. 22
Figure 15. Example of how to start transfer learning. 23
Figure 16. Example of how to resume training from previous epoch. 23
Figure 17. Exporting the model in .onnx format. 24
Figure 18. Importing required modules. 24
Figure 19. Initializing network, input and output. 25
Figure 20. Start of the main function. 25
Figure 21. Start of the try block and while loop. 26
Figure 22. For loop and time keeping. 26
Figure 23. Expect block with data gathering. 27
Figure 24. Calculating percentage and saving statistics into text file. 27

Figure 25. Running the application. 28
Figure 26. Screencapture of running program. 28
Figure 27. Terminal output. 29
Figure 28. Text file output. 29

LIST OF ABBREVIATIONS AND SYMBOLS

RAM	Random Access Memory
GB	Gigabyte
OS	Operating System
HDMI	High-Definition Multimedia Interface
DP	DisplayPort
USB	Universal Serial Bus
CSI(camera)	Camera Serial Interface
SSD	Single Shot multibox Detector
IDE	Integrated Development Environment
Pascal VOC	Pattern Analysis, Statistical Modeling, Computational Learning Visual Object Challenge
GPU	Graphics Processing Unit
CPU	Central Processing Unit
AI	Artificial Intelligence
®	Registered Trademark
MHz	Megahertz
GB/s	Gigabytes per Second
I2C	Inter-integrated Circuit
I2S	Inter-IC Sound
SPI	Serial Peripheral Interface

UART	Universal Asynchronous Receiver-Transmitter
API	Application Programming Interface
DNN	Deep Neural Network
ONNX	Open Neural Network Exchange
WiFi	Wireless Fidelity
MB/s	Megabytes per Second
L4T	Linux for Tegra
VSCode	Visual Studio Code
SSH	Secure Shell
XML	Extensible Markup Language

1 INTRODUCTION

Object detection becoming more and more popular on everyday usage. Object detection can be used for variety of purposes, for example safety critical and precise applications, but using it for more day-to-day applications that helps the community is becoming more popular and can be carried out on lighter devices. Working with neural networks used to be a very sophisticated field, but technologies are moving forward with very fast pace. Nowadays, there is many powerful pretrained neural networks models. That are available to everyone.

The focus of the thesis is to investigate the elements needed for deploying object detection application on embedded device called Nvidia Jetson Nano and its capabilities to carry out object detection application. Such elements what we are going to examine are Nvidia Jetson Nano itself and its operating system JetPack. Jetson-inference, which is Nvidia's docker container consisting variety of neural network examples and tools to help guide a way to build a custom application. Take a look at programming language which is used heavily in the field of machine learning and neural networks, Python. Brief look at heuristics of SSD-Mobilenet. Deeper functionalities of neural networks are out of the scope of this thesis. Also going over transfer learning and model conversion.

Thesis also includes a detailed instruction of example implementation of object detection application on embedded device.

Thesis is structured as follows. Chapter 2 introduces the reader to the environment, tools and methods of object detection on Nvidia Jetson Nano. In the following chapters, Chapter 3 and Chapter 4, go over of implementing object detection application and its steps. Demonstrating the usage of the application and go over the performance of application created.

2 THEORETICAL BACKGROUND

Everyone has bad habits while doing something important or trying to be productive. Some has more some has less, and everyone's habits are unique. [1] This chapter is focusing on technologies and devices used in the thesis. Thesis is focusing on capabilities and deployment of Nvidia Jetson Nano to run object detection application as an embedded device. Program's ability to help the users become more self-aware of their behavior.

2.1 Nvidia Jetson Nano Developer Kit

Nvidia Corporation is a global technology company from the United States of America. Nvidia's main products include graphics processing units (GPU) and system on a chip units (SoCs). [2]

Nvidia Jetson Nano is a small sized, but powerful computer. Its main focus is to run AI applications in energy-efficient environment. Jetson Nano, even though its small size and reasonable price has impressive hardware inside. Jetson Nano has a GPU which is utilizing Nvidia Maxwell architecture with 128 Nvidia CUDA® cores. Nvidia Jetson Nano also have Quad-Core ARM Cortex-A57 MPCore processor as its CPU. For memory Nvidia Jetson Nano has 4GB 64-bit LPDDR4 RAM, running at speed of 1600MHz and 25.6 GB/s. Nvidia Jetson Nano also has multiple USB 3.0 ports, Ethernet connectivity, HDMI 2.0 and DP 1.4 ports and interfaces to connect GPIO, I2C, I2S, SPI and UART. [3]

Figure 1. Nvidia Jetson Nano. – Source: https://developer.nvidia.com/

2.1.1 JetPack and Jetson-inference

Nvidia Jetson Nano utilizes Nvidia JetPack SDK, which is built on Ubuntu 18.04 operating system. JetPack SDK is a customized OS image and purposely made for building AI applications. JetPack SDK comes with multiple useful libraries prebuilt in the OS image and L4T. L4T stands for Linux for Tegra. L4T is Nvidia Jetson Linux Driver Package, and it is the main support package for the Nvidia Jetson Nano. It comes with Linux Kernel, Nvidia drivers, bootloader, flashing utilities and more based on Ubuntu 18.04 operating system. Such as TensorRT, which is vital part of this project. JetPack also includes APIs for deep learning and computer vision. [4]

Jetson-inference is completely an open-source repository, and its purpose is to guide users on how to use Nvidia Jetson Nano as an embedded device with deep neural network capabilities. Jetson-inference is made by Nvidia, and its main maintainer is Dustin Franklin. Jetson-inference includes many tutorials how to try

out different deep neural networks and some python scripts to get started with own projects. [5]

2.2 Python

Python in general, is a programming language. Python got its name from British comedy show called Monty Python's Flying Circus. Guido van Rossum got the idea while reading scripts of the show. [6] Creator of the Python is aforementioned Guido van Rossum. He is from Netherlands. Python was released on February 20, 1991. [7]

Python can be described as interpreted language, which means that python programs execute programs directly, without compiling program into machine-language. Python programming language also falls in to category of object-oriented languages. Python can be used for many different purposes, for example web development, software development, scripting and many more. It is also used as a language to build to existing applications together, because Python is also high-level language. Python is also very user-friendly and because of its easy to learn syntax it is also really beginner-friendly language. [8]

```
def example():
    x = 10
    y = 10
    result = x + y
    print(result)

example()
```

Figure 2. Example of Python syntax.

This is one of the reasons why Python is on the top of the list of the most popular programming languages. [9] Python is also highly recognized and used by major companies and applications, such as Spotify. [10] As to this day even though Python is over 30-year-old language it is highly respected language among

programmers and many companies are looking for skilled Python programmers according to career advice from techgig.com. [11]

2.3 SSD-Mobilenet

SSD-Mobilenet is an object detection model. Which is using Single-Shot multibox Detector(SSD) neural network architecture. This means that object detection takes one image or one frame at a time and is able to detect multiple boxes. Single Shot multibox Detection is designed for real time object detection. Reason why SSD architecture is highly used for real time detection is that it is fast and can reach higher frames per second while monitoring out output. SSD is faster compared to other reminiscent algorithms because it is not using region proposal network architecture. [12]

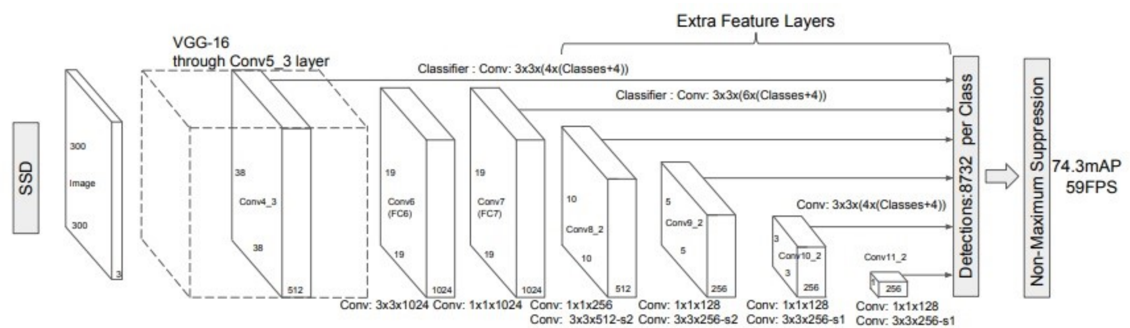

Figure 3. SSD-Mobilenet layers. -- Source: https://arxiv.org/abs/1512.02325

Mobilenet itself is a convolutional neural network made specifically for mobile and embedded devices. Mobilenet is using streamline architecture, which means that it is using depth-wise and point-wise convolutions. Mobilenet has 28 layers. The mobilenet was designed to build the neural network models light enough to be deployed on devices with limited computing power such as aforementioned mobile and embedded devices. [13] Mobilenet architecture was first introduced by Google in 2017. [14]

2.3.1 Re-training (Transfer learning)

Transfer learning is a technique to use existing or pretrained model as a backbone of the new model. Transfer learning also referred as re-training is a quite common approach to create new neural network models. Creating well performing neural networks can be a challenge, because they need an immense amount of data and resources to begin with. [15]

In transfer learning, the re-trained model is trying as much as possible to use the pre-trained model's knowledge and apply it to re-trained model.

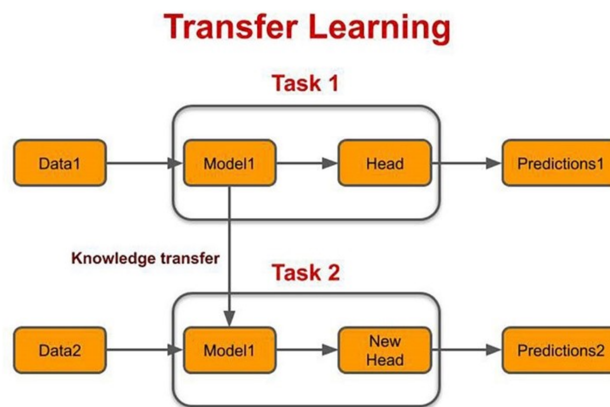

Figure 4. Transfer learning visualized. – Source: https://www.topbots.com/transfer-learning-in-nlp/

In transfer training, we are only using couple of first layers from pre-trained model and use rest of the layers are used for re-training. Pre-trained model is already trained to detect objects in images or frames of live feed and then transfer learning comes in to play. In the latter layers transfer training is training the network to detect new labels. As an example, pre-trained model made to detect cars in images is used to transfer learning into re-trained model detecting trains. [16]

2.3.2 Converting trained model

Converting neural network model is common practice in the field of machine learning. Thousands of experiments with neural networks and machine learning are done on a daily basis around the globe. There are multiple valid frameworks to do those experiments and programs, such as Keras, PyTorch and TensorFlow. Open Neural Network Exchange (ONNX) is a tool or package which can be used to convert trained neural network model for example from PyTorch to be used in multiple different frameworks. So, in other words, you can use any machine learning framework to train your model, and only after training you can convert the trained model into ONNX format. ONNX is also community driven open-source project, which is highly praised by major companies like Microsoft, Hewlett Packard Enterprise, Nvidia and many more minor and major companies. [17]

Figure 5. ONNX logo. – Source: https://en.wikipedia.org/wiki/Open_Neural_Network_Exchange

3 IMPLEMENTATION

Nvidia Jetson Nano is capable of many different types of applications and programs. This chapter contains a closer look into making object detection application. Implementation of such application requires many steps and is quite time consuming. Chapter is structured step by step, from very start of setting up Nvidia Jetson Nano with operating system image to very end of structure of python script that runs the application and gathers data.

3.1 Setting up Jetson Nano

To get started with the application, first we need to make sure we have minimum required peripherals to setup Nvidia Jetson Nano with display attached in to it. Peripheral needed to do this are the following USB or CSI camera, USB mouse, USB keyboard, HDMI or DP cable, compatible display, internet connection with either WiFi dongle or WiFi module, Micro-USB power cable or compatible DC Barrel Jack power supply and of course finally microSD card. Even though recommended minimum for the microSD card is 32GB with at least 100 MB/s bus interface speed. It is preferred to have at least 64GB microSD card with at least 100MB/s bus interface speed. This is recommendation is due to some machine learning applications can possibly be using large datasets, thus more memory is preferred.

3.1.1 Flashing Jetpack onto SD Card

Nvidia Jetson Nano itself is useless without operating system. Next step is to flash JetPack 4.6.1 onto microSD card. JetPack version 4.6.1 is utilizing L4T version R32.7.1. First step is to download the JetPack image from Nvidias developer website. It is recommended to format the card clean, before flashing anything onto it. Thus, next step is the preparation of the microSD card. To format the card program called SD Memory Card Formatter from SD Association come into play. Quick format is preferred. SD Association is organization established in

2000 by Panasonic, SanDisk Corporation and Toshiba Corporation to develop memory card standards.

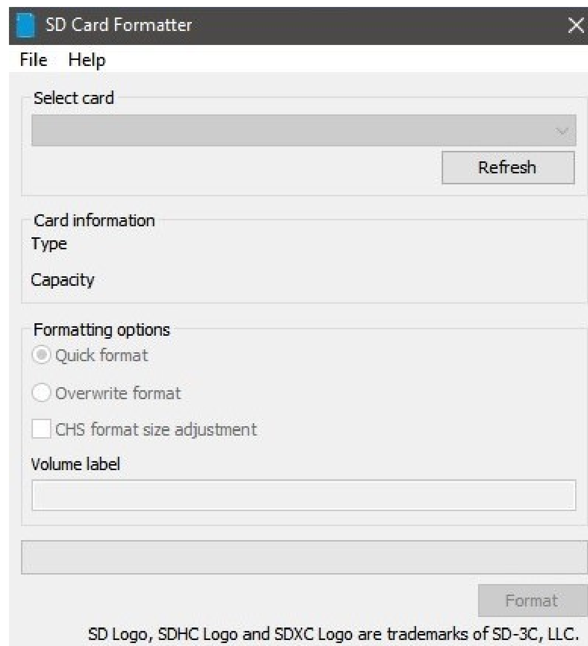

Figure 6. SD Card Formatter interface.

After formatting microSD card, it needs to be flashed with JetPack 4.6.1. To do this it is recommended to use program named Etcher by Balena.

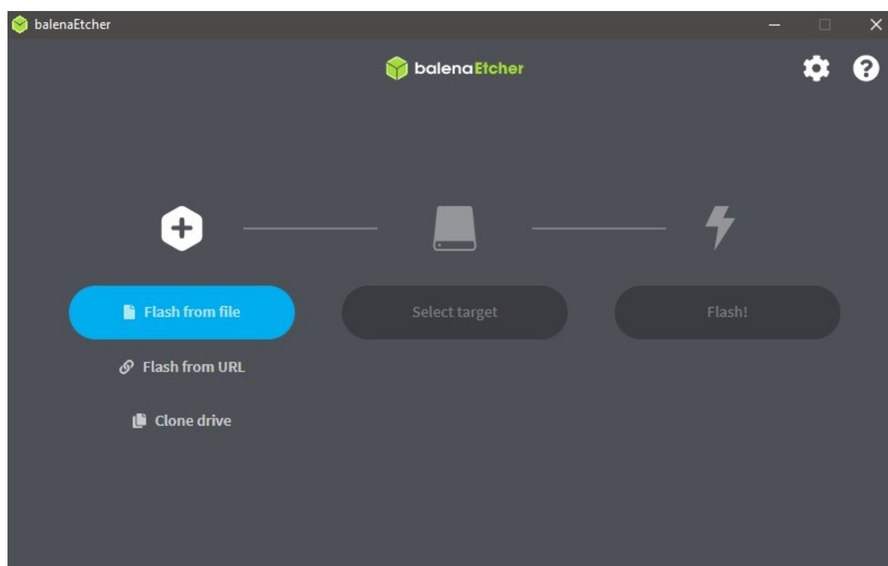

Figure 7. Etcher interface.

After completing flashing the image, Nvidia Jetson Nano is ready to be booted for the first time.

3.1.2 Initial setup of the Jetpack with Display

Booting Nvidia Jetson Nano with fresh JetPack operating system image for the first time with display. First insert the microSD card containing the image into Nvidia Jetson Nano. After this, all of the peripherals aforementioned need to be plugged in. Insert power supply last as Nvidia Jetson Nano will turn on after power is supplied. During the first boot, the developer kit takes user through initial setup.

3.1.3 Setting up Jetson-inference

Setting up the Jetson-inference repository as a Docker container. Jetson-inference containers are using the L4T-pytorch container as a base container. Which means that the containers come with PyTorch and torchvision installed in it. So, Jetson-inference container comes with support to perform transfer learning as it comes. Cloning the jetson-inference repository and running the docker container with the following commands.

```
$ git clone --recursive https://github.com/dusty-nv/jetson-inference
$ cd jetson-inference
$ docker/run.sh
```

Figure 8. Cloning and running jetson-inference container.

When running the Jetson-inference docker container for the first time, it will pull the precise container tag from Docker Hub. Container tag is established on JetPack L4T version. Also, during this step prompt will ask which neural network models to download. Such as SDD-Mobilenet.

3.1.4 Editor (IDE)

As an Editor using Visual Studio Code (VSCode). VSCode has great capabilities for remote development. Using SSH connection to develop on Nvidia Jetson Nano is not required, but highly recommended. Establishing SSH connection to Nvidia Jetson Nano can be achieved with Remote-SSH plugin for VSCode. Remote-SSH software is made by Microsoft.

Though this is not necessary, editor called Gedit that comes with JetPack operating system will work just fine also. It comes down to personal preference working through an SSH connection.

3.1.5 Mounting Swap Memory

As Nvidia Jetson Nano is a small sized embedded device, it comes with some limitations. Memory being one of them. For transfer learning it is recommended to mount more Swap memory. Swap memory is used to support RAM memory when running memory heavy applications. Even though Swap memory is distinctly slower than RAM, it is efficient to have Swap memory to ease off the load on RAM. Mounting Swap memory on Nvidia Jetson Nano can be achieved as follows.

```
$ sudo systemctl disable nvzramconfig
$ sudo fallocate -l 4G /mnt/4GB.swap
$ sudo chmod 600 /mnt/4GB.swap
$ sudo mkswap /mnt/4GB.swap
$ sudo su
$ echo "/mnt/4GB.swap swap swap defaults 0 0" >> /etc/fstab
$ exit
```

Figure 9. Mounting swap memory on Nvidia Jetson Nano.

3.2 Gathering data

At this point forward next steps are in Jetson-inference docker container. Run the docker container with docker/run.sh shell script.

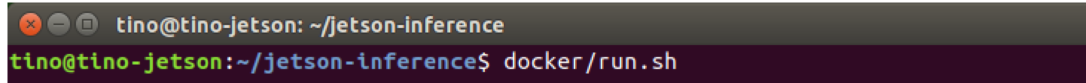

Figure 10. Running Jetson-inference docker container.

Gathering data for custom object detection model can be done in multiple ways. There is not one right way to do this. With Jetson-inference container, comes the camera-capture software, which can be used to obtain images from USB or CSI camera. Object detection needs an image of the object, and also a bounding box. With camera-capture software collecting data of the object is made simple.

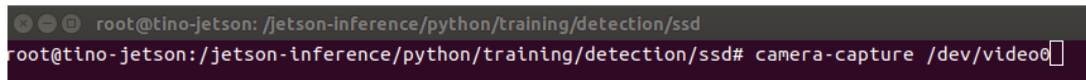

Figure 11. Running data collecting software.

Before starting to collect own dataset, path to the dataset and labels needs to be established. That can be achieved by creating new directory inside /jetson-inference/python/training/detection/ssd/data with labels.txt file inside, which contains labels. In this case there should only be word phone.

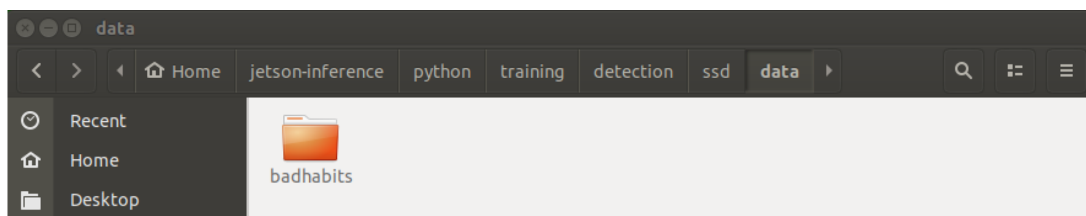

Figure 12. Correct data directory.

Next step is to gather data. Insert correct data path and class labels in camera-capture tool. Freeze the frame and draw bounding box around the object.

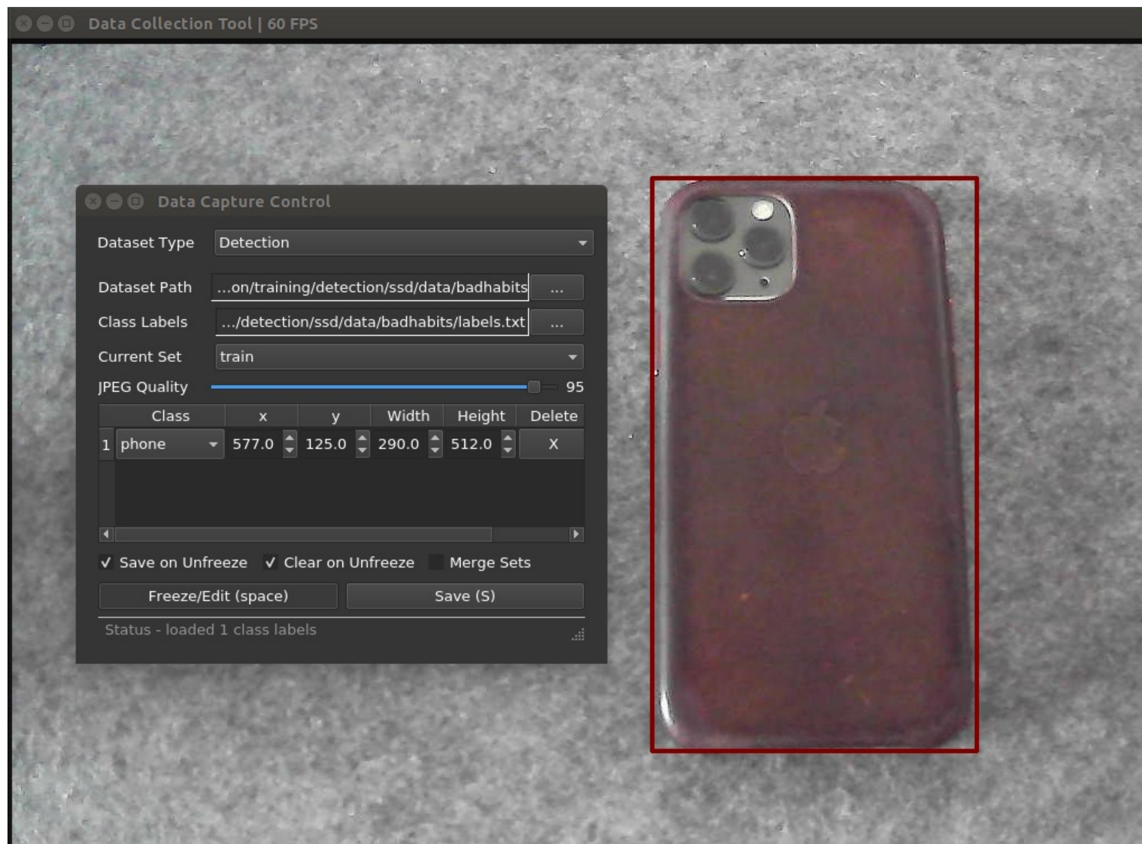

Figure 13. Example of data collection into training dataset.

Then data collection can start. For each image, there needs to be a bounding box around the object, this can make the process time consuming if using this tool to collect large amounts of data. Camera-capture software creates training, validation and testing datasets. Amount of data collected varies a lot. Preferred amount per object is around 1000 images and annotations. To get more reliable results out of the object detection model, it is important to take variety of sample data of the object with different angles and backgrounds. Data should be distributed to 80% of training images, 10% validation images and 10% testing images. 800 training image, 100 validation images, 100 testing images.

3.2.1 Data Format (Pascal VOC, image/annotation)

Pascal VOC stands for Pattern Analysis, Statistical Modeling, Computational Learning Visual Object Challenge. This data format provides standardized data for object detection. Pascal VOC data format includes images and their annotations. Annotations are the bounding boxes in XML format from previous step. Annotation is created automatically by camera-capture tool based on the bounding box drawn by the user.

```
<annotation>
    <filename>20220323-090808.jpg</filename>
    <folder>badhabits</folder>
    <source>
        <database>badhabits</database>
        <annotation>custom</annotation>
        <image>custom</image>
    </source>
    <size>
        <width>1280</width>
        <height>720</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>phone</name>
        <pose>unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>391</xmin>
            <ymin>154</ymin>
            <xmax>600</xmax>
            <ymax>519</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 14. Example of a bounding box annotation in XML format.

3.3 Re-training SSD-Mobilenet

After data has been collected, next step is to use transfer learning to re-train SSD-Mobilenet with the new model. Jetson-inference provides python script to do so. Transfer learning is done by using PyTorch.

```
root@tino-jetson: /jetson-inference/python/training/detection/ssd
root@tino-jetson:/jetson-inference/python/training/detection/ssd# python3 train_ssd.py --dataset-type=voc
 --data=data/badhabits --model-dir=models/badhabits --batch-size=2 --workers --epochs=10
```

Figure 15. Example of how to start transfer learning.

Parameters like –dataset-type=voc declares that data used for re-training is in Pascal VOC format. Parameter –data=data/BadHabits stands for path where data is located. And parameter –model-dir=models/BadHabits includes class labels. Optional parameters are to there to slightly spare memory while training. And –epochs=10 declares how many times PyTorch is going to go through the data during re-training. Training neural networks to produce reliable and accurate results can be time consuming. Example implementation went through 80 epochs. Training was done in multiple parts. Total time rounded up to 4 hours of pure training.

3.3.1 Resume training

For performance point of view. It is sometimes necessary to resume training. Resume training continues from the last successful epoch. Every epoch creates new model and next epoch continues from that checkpoint. Sometimes to get model loss and accuracy to meet the requirements, additional training is needed. Resuming from last epoch can be achieved by the following command. Standard reliable loss is between 2.5-1.0.

```
root@tino-jetson: /jetson-inference/python/training/detection/ssd
root@tino-jetson:/jetson-inference/python/training/detection/ssd# python3 train_ssd.py --dataset-type=voc
--data=data/badhabits --model-dor=models/badhabits --pretrained-ssd=models/badhabits/mb1-ssd-Epoch-9-Loss-
4.972116218794376.pth --batch-size=2 --workers=1 --epochs=10
```

Figure 16. Example of how to resume training from previous epoch.

3.4 Exporting model

When transfer learning is done, the model needs to be exported to .onnx format. Models of .onnx format can be run on many major machine learning frameworks. To export freshly transfer trained model to .onnx format. Jetson-inference provides python script to do so. We can achieve this by executing this terminal command.

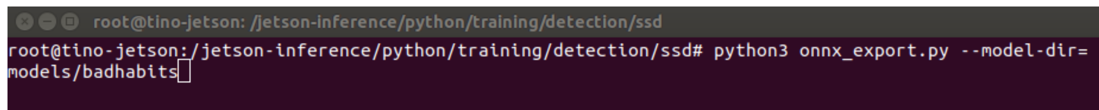

Figure 17. Exporting the model in .onnx format.

This python script outputs ssd-mobilenet.onnx file which is the custom re-trained model.

3.5 Python program

Next, we are structuring the python program that is going to use the custom SSD-Mobilenet model created in the previous steps and gather some useful data out of the program. This program is utilizing jetson.inference, jetson.utils, time and datetime modules.

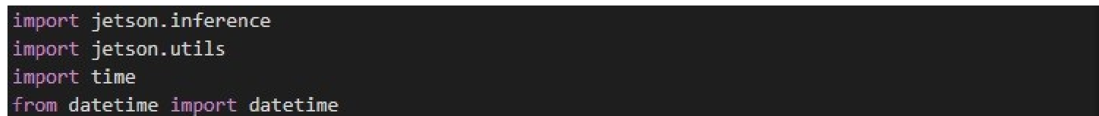

Figure 18. Importing required modules.

Up next is initialization of the network, input source and output source. As a network we are using the model we created above. Parameters in network variable are the location of the model and labels. Also setting confidence percentage and boxes on location of the detected object.

```
network = jetson.inference.detectNet(argv=["--model=BadHabits/ssd-mobilenet.onnx",
                                           "--labels=BadHabits/labels.txt",
                                           "--input-blob=input_0",
                                           "--output-cvg=scores",
                                           "--output-bbox=boxes"],
                                           threshold=0.5)

# Change '/dev/video0' to 'csi://0' if you're using CSI camera.
camera = jetson.utils.videoSource('/dev/video0')

display = jetson.utils.videoOutput('display://0')
```

Figure 19. Initializing network, input and output.

Structuring the main function. First is declared start variable that holds start time of the main function. Which is later used to get the total runtime of the program. Then global variables timeHold and totalTime. Variable timeHold is total time when object is detected and totalTime is total runtime. Variable timeHold is set to zero.

```
def main():
    start = time.time()
    global timeHold
    global totalTime
    timeHold = 0
```

Figure 20. Start of the main function.

After declaring some variables and setting start timer comes the try block. Try block includes another timer which is used to count the value of timeHold variable. Next while loop, which includes many essential features. Variable phoneDetected is set to False so startTimeStamp can be reset in case that frame of the video did not include an object. Setting variable liveFeed to set input source to capture. Detections variable is using provided model to detect input source liveFeed. Then output source display is set to render liveFeed and stream it on the window with SetStatus() function. Capture, Render and SetStatus are part of jetson.utils module and Detect is part of jetson.inference module.

```
try:
    print("\n WELCOME TO BAD HABIT DETECTOR \n")
    #Creating starting time stamp for counting time phone detected
    startTimeStamp = time.time()

    # While loop for live video feed
    while display.IsStreaming():
        # Setting phoneDetected to false and starting live feed
        phoneDetected = False
        liveFeed = camera.Capture()
        detections = network.Detect(liveFeed)
        display.Render(liveFeed)
        display.SetStatus('Bad Habit Detector | {:.0f} FPS'.format(network.GetNetworkFPS()))
```

Figure 21. Start of the try block and while loop.

Next, we have for loop to go through labels within the provided model. Variable label is getting the labels from the model with jetson.inference modules function GetClassDesc(). Then if statement with condition that the label that is detected is phone. When If condition is met that means that in current frame object was detected. Then phoneDetected is set to True and newTimeStamp timer is started to count time passed in current frame. After this startTimeStamp is being reset. Else statement should not happen since no other labels are in the model, if it does happen, it prints error. At the end of the while loop startTimeStamp is reset if phoneDetected value remain False.

```
for detection in detections:
    label = network.GetClassDesc(detection.ClassID)

    # Deciding action when "phone" is detected
    if label == "phone":
        # Counting how long phone is detected
        phoneDetected = True
        newTimeStamp = time.time()
        timeDelta = newTimeStamp - startTimeStamp
        # Adding timeDelta value to timeHold
        timeHold += timeDelta
        # Setting startTimeStamp to newTimeStamp, so time.time() resets
        startTimeStamp = newTimeStamp
        print("BAD HABIT DETECTED")

    else:
        # Print error if something else is detected
        print("An error occurred")

# Resets startTimeStamp if phone is not detected
if phoneDetected == False:
    startTimeStamp = time.time()
```

Figure 22. For loop and time keeping.

To end the program and gather the data program is using except Keyboard Interrupt block. Inside except variable end saves another time stamp and totalTime is calculated between end and start values. Variable without phone is calculated between totalTime and timeHold. Program prints some values to the terminal. Also, percentageHold and saveStats functions are called at this stage.

```
except KeyboardInterrupt:
    print("\n\n\tProgram ended by the user.\n")
    # Ending timer and doing some calculations at to be printed after program is finished
    end = time.time()
    totalTime = end - start
    withoutPhone = totalTime - timeHold
    print("\tProgram was on for", float("{:.1f}".format(totalTime)), "seconds.")
    print("\tTotal time with phone was", float("{:.1f}".format(timeHold)), "seconds")
    print("\tTotal time without phone was", float("{:.1f}".format(withoutPhone)), "seconds")
    percentageHold()
    saveStats()
    pass
```

Figure 23. Expect block with data gathering.

Function to calculate the percentage of time object is detected compared to total runtime of the program. Saving and writing the stats of the program into a text file for further data analysis is done with saveStats function. Function appends into text file called bad_habit_statistics.txt. Including the date and time when program was running, and percentage of how long object was detected and total runtime of the program.

```
# Function to calcultate percentage
def percentageHold():
    global percent
    percent = timeHold / totalTime * 100
    print("\tYou had phone in hand for",
            "{:.0f}".format(percent),
            "% of the time program was running.\n\n")


# Function to save most valuable statisics in to text file.
def saveStats():
    timeNow = datetime.now()
    timeNewFormat = timeNow.strftime("%d/%m/%Y %H:%M:%S")
    f = open("bad_habit_statistics.txt", "a")
    f.write("\nSession at",
            timeNewFormat,
            " ou used phone for",
            "{:.0f}".format(percent),
            "%","of the total runtime. Total runtime of the program was",
            "{:.1f}".format(totalTime),
            "seconds.")
    f.close()
```

Figure 24. Calculating percentage and saving statistics into text file.

4 TESTING

4.1 Running the program

Running the program created above can be little confusing. Python file running the program must be located inside jetson-inference/python/training/detection/ssd/models directory. Running the program is done by running the python script from the terminal using following terminal command.

```
root@tino-jetson: /jetson-inference/python/training/detection/ssd/models
root@tino-jetson:/jetson-inference/python/training/detection/ssd/models# python3 run_bad_habits.py
```

Figure 25. Running the application.

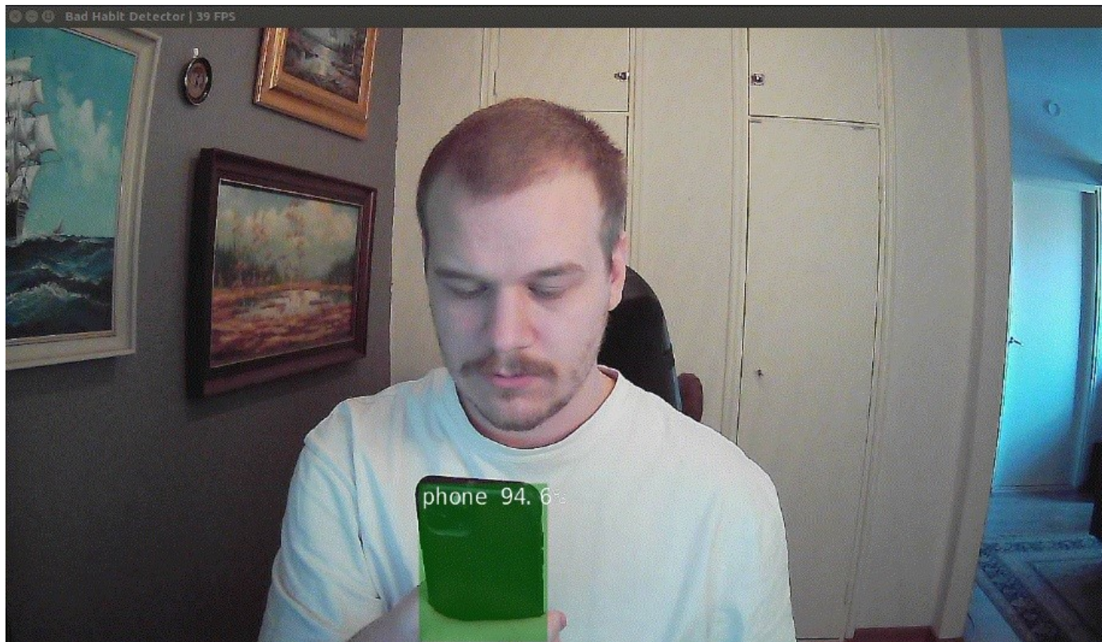

Figure 26. Screen capture of running program.

After running the program for 54 seconds and 186 seconds. And having a phone in hand for time to time each time. Program gives following output in the terminal and in the text file.

```
Program ended by the user.

Program was on for 53.5 seconds.
Total time with phone was 27.7 seconds
Total time without phone was 25.8 seconds
You had phone in hand for 52 % of the time program was running.
```

Figure 27. Terminal output.

```
Session at 31/03/2022 15:27:21 you used phone for 23% of the total runtime. Total runtime of the
program was 185.2 seconds.
```

Figure 28. Text file output.

Running several tests with an external stopwatch. Timer performance proved to be accurate and reliable.

4.2 Performance and Model accuracy

Object detection application is running consistently around 40-50 frames per second. Which is respectable numbers for an embedded device. This proves that SSD-Mobilenet is capable of running lightweight neural network models. Detection of the object is handling well when the object is in certain angle and distance from the input source. Performance also drops if object is covered too much with hand. Can handle different backgrounds quite well. Program is handling bright places notably better than dark. That could be yield of training material being mainly with bright background.

5. CONCLUSIONS

Machine learning and object detection is becoming more and more popular within' embedded devices. For industrial use, as an example for monitoring devices, security devices, quality assurance devices and many more, but also object detection with embedded devices for personal use is accessible. The ability to create such applications using object detection networks can help an individual or industry solve a variety of different problems and tasks.

Purpose of this thesis was to introduce the reader on basic tools and technologies used in such applications and development on embedded device called Nvidia Jetson Nano. Also, to demonstrate deployment of object detection application from ground up. Even though program is not nowhere near perfect, thesis proved the capabilities of Nvidia Jetson Nano to be a sole environment of development for such application. For further improvements of application in question, would be furthering the data set, continue training for even better accuracy, but also add variety of bad habits. Object detection is not limited in any way on which object is decided to detect, thus it is powerful technique to improve day to day life. Nvidia Jetson Nano carried out the application with satisfaction, but it does come with limitations. Such as computing power. Handling even larger object detection applications can be troublesome, in example training large data sets.

Outcoming data of the example implementation application can be used in many different ways. Data can be used to monitor user behavior during any given day and time. Monitor behavior either during work or free time. Compare the gathered data to better the understanding of the behavior in different environments and use the data to understand the issue and help point a way to discard or adapt the unwanted habit.

REFERENCES

[1] Patrick W. L. 2018. How Your Cell Phone Habit Impact Your Productivity. Psychology Today. [cited 15 March 2022] Available from: https://www.psychologytoday.com/us/blog/why-bad-looks-good/201807/how-your-cell-phone-habits-impact-your-productivity

[2] Nvidia. (n.d.). About Us. Nvidia. [cited 15 March 2022] Available from: https://www.nvidia.com/en-us/about-nvidia/

[3] Nvidia. (n.d.). Jetson Nano. Nvidia Developer. [cited 15 March 2022] Available from: https://developer.nvidia.com/embedded/jetson-nano

[4] Nvidia. (n.d.). JetPack SDK 5.0 Developer Preview. Nvidia Developer. [cited 18 March 2022] Available from: https://developer.nvidia.com/embedded/jetpack

[5] Franklin D. 2022. Dusty-nv/jetson-inference: Deploying Deep Learning. GitHub. [cited 18 March 2022] Available from: https://github.com/dusty-nv/jetson-inference

[6] Python. (n.d.). General Python FAQ. Python Docs. [cited 22 March 2022] Available from: https://docs.python.org/3/faq/general.html

[7] Python Institute. (n.d.). What is Python? Python Institute. [cited 22 March 2022] Available from: https://pythoninstitute.org/what-is-python/

[8] Python. (n.d.). Executive Summary. Python. [cited 22 March 2022] Available from: https://www.python.org/doc/essays/blurb/

[9] TIOBE. 2022. TIOBE Index for April 2022. TIOBE. [cited 22 March 2022] Available from: https://www.tiobe.com/tiobe-index/

[10] van der Meer G. 2013. How we use Python at Spotify. Spotify R&D. [cited 22 March 2022] Available from: https://engineering.atspotify.com/2013/03/how-we-use-python-at-spotify/

[11] Yaday I. 2021. Why hiring managers pick Python as most in-demand language of 2022. TechGig. [cited 22 March 2022] Available from: https://content.techgig.com/career-advice/python-most-in-demand-language-for-2022/articleshow/88409336.cms

[12] Hui J. 2018. SSD object detection: Single Shot MultiBox Detector for real-time processing. Medium. [cited 31 March 2022] Available from: https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06

[13] Howard A. G. & Zhu M. & Chen B. & Kalenichenko D. & Wang W. & Weyand T. & Andreetto M. & Adam H. 2017. MobileNets: Efficient Convolutional Neural Networks For Mobile Vision Applications. Cornell University. [cited 31 March 2022] Available from: https://arxiv.org/abs/1704.04861

[14] Howard A. G. & Zhu M. 2017. MobileNets: Open-Source Models for Efficient On-Device Vision. Google AI Blog. [cited 31 March 2022] Available from: https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html

[15] Brownlee J. 2017. A Gentle Introduction to Transfer Learning for Deep Learning. Machine Learning Mastery. [cited 4 April 2022] Available from: https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[16] Sharma P. 2021. Understanding Transfer Learning for Deep Learning. Analytics Vidhya. [cited 4 April 2022] Available from: https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/

[17] Jog C. 2020. The Two Benefits of the ONNX Library for ML models. Medium. [cited 9 April 2022] Available From: https://medium.com/trueface-ai/two-benefits-of-the-onnx-library-for-ml-models-4b3e417df52e

Source code of example implementation

```
'''
Program Name:   Bad Habit Detector
Author:         Tino Nummela
'''


# Importing needed modules
import jetson.inference
import jetson.utils
import time
from datetime import datetime


# Initializing model, input source and output source
network = jetson.inference.detectNet(argv=["--model=BadHabits/ssd-mobilenet.onnx",
                                           "--labels=BadHabits/labels.txt",
                                           "--input-blob=input_0",
                                           "--output-cvg=scores",
                                           "--output-bbox=boxes"],
                                           threshold=0.5)


# Change '/dev/video0' to 'csi://0' if you're using CSI camera.
camera = jetson.utils.videoSource('/dev/video0')
display = jetson.utils.videoOutput('display://0')


# Function to calcultate percentage
def percentageHold():
   global percent
```

```
    percent = timeHold / totalTime * 100
    print("\tYou had phone in hand for", "{:.0f}".format(percent),
           "% of the time program was running.\n\n")



# Function to save most valuable statisics in to text file.
def saveStats():
    timeNow = datetime.now()
    timeNewFormat = timeNow.strftime("%d/%m/%Y %H:%M:%S")
    f = open("bad_habit_statistics.txt", "a")
    f.write("\nSession at " + timeNewFormat + " you used phone for " +
             "{:.0f}".format(percent) +
             "% of the total runtime. Total runtime of the program was " +
             "{:.1f}".format(totalTime) +
             " seconds.")
    f.close()



# Main function
def main():
    # Starting timer and initializing global variables
    start = time.time()
    global timeHold
    global totalTime
    timeHold = 0

    try:
        print("\n WELCOME TO BAD HABIT DETECTOR \n")
```

```
    #Creating starting time stamp for counting time phone detected
    startTimeStamp = time.time()

    # While loop for live video feed
    while display.IsStreaming():
        # Setting phoneDetected to false and starting live feed
        phoneDetected = False
        liveFeed = camera.Capture()
        detections = network.Detect(liveFeed)
        display.Render(liveFeed)
        display.SetStatus('Bad Habit Detector | {:.0f}
FPS'.format(network.GetNetworkFPS()))

        # For loop for going through classes/labels, easy to scale when more bad habits
        are added
        for detection in detections:
            label = network.GetClassDesc(detection.ClassID)

            # Deciding action when "phone" is detected
            if label == "phone":
                # Counting how long phone is detected
                phoneDetected = True
                newTimeStamp = time.time()
                timeDelta = newTimeStamp - startTimeStamp
                # Adding timeDelta value to timeHold variable
                timeHold += timeDelta
                # Setting startTimeStamp to newTimeStamp, so time.time() resets
                startTimeStamp = newTimeStamp
                print("BAD HABIT DETECTED")
```

```
            else:
                # Print error if something else is detected
                print("An error occurred")

        # Resets startTimeStamp if phone is not detected
        if phoneDetected == False:
            startTimeStamp = time.time()

    # CTRL+C to stop the program and print statistics
    except KeyboardInterrupt:
        print("\n\n\tProgram ended by the user.\n")
        # Ending timer and doing some calculations at to be printed after program is
        finished
        end = time.time()
        totalTime = end - start
        withoutPhone = totalTime - timeHold
        print("\tProgram was on for", float("{:.1f}".format(totalTime)), "seconds.")
        print("\tTotal time with phone was", float("{:.1f}".format(timeHold)), "seconds")
        print("\tTotal time without phone was", float("{:.1f}".format(withoutPhone)),
                "seconds")
    percentageHold()
    saveStats()
    pass
# Calling main function
if __name__ == '__main__':
main()
```