CAFÉ OCCUPANCY MONITORING SYSTEM BASED ON DEEP LEARNING FOR VISUAL RECOGNITION

A PREPRINT

December 11, 2020

ABSTRACT

The project documented by this report presents a concept for avoiding long queues at downtown cafés during lunch hours in large cities. Based on deep learning for visual recognition, it uses a quantized convolutional neural network in order to monitor the occupancy in all participating cafés. Thus, customers can be distributed equally and waiting queues can be avoided. The report also documents a primitive, yet functional prototype, which uses Tensorflow Lite to deploy a custom-trained convNet on a Raspberry Pi with an attached camera. The project serves as an experimental and learning environment for the student to apply knowledge and skills acquired in the Deep Learning for Visual Recognition course taught at Aarhus University, Fall 2020.

***K*eywords** Occupancy Monitoring · Deep Learning for Visual Recognition · Convolutional Neural Networks · Tensorflow Lite · MobileNet v2

1 Introduction

Downtown cafés in large cities such as Aarhus experience several rush hours throughout the day. Especially during lunch time, one must stand in line for several minutes, sacrificing precious free time for some cake and a cup of coffee. Not that it wouldn't be worth it – it is! Yet, much nicer would it be to get the treat right away and enjoy it sitting on a terrace rather than spending all the free time standing in a line [1]. And lately, more than ever, the importance of avoiding tight gathering of people became evident, due to the COVID-19 pandemic.

A simple concept based on Deep Learning methods for object detection could allow customers to know, which cafés are currently hotspots to avoid and at which ones they can expect the least waiting time. By installing an occupancy monitoring system in each café, live data can be used to suggest the best cafés close by. This data can also be processed to give an estimation for the future, making it easy for customers to plan their lunch hours. This would allow cafés to relief their workers' stress during those hours and enhance the user experience of their customers, which is also in the best interest of the cafés.

This project documents a prototype, which can be installed in a room. It consists of a Raspberry Pi and a camera, which monitors the the room. The quantized convNet MobileNet v2 is used in combination with Google's machine learning platform for mobile devices, TensorFlow Lite, to recognize the amount of people within camera view. It has been trained with a custom data set to differentiate between people already served and those still waiting. Thus it avoids wrong estimations for the actual waiting time. The prototype also involves the deployment of a webserver, which displays the amount of people in the room.

2 Related Work

2.1 OpenCV People Counter

This program counts the number of people incoming and outgoing a particular door from a video file [2]. If using a live stream and once calibrated, it could thus provide an accurate number of people in a room, such as a Café. It can be run on a Raspberry Pi, making it suitable for low-end devices and small-scale applications. While it does count people in a room, it cannot know, how many of those have already been served or how long the queue is. Likewise, if the queue goes beyond the entrance into the open of a very small café, the program gives unsatisfactory results. It does, however, give a good example of using Numpy and OpenCV in relation to counting people.

2.2 TensorFlow Object Detection API

Creating accurate machine learning models capable of localizing and identifying multiple objects in a single image remains a core challenge in computer vision [3]. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. The API could for example be used to count the amount of people in a room. Additionally, it could count the amount of coffee cups and compare the amount with the amount of people, thus giving some kind of estimation on how many customers have been served. Of course this method is open to errors, because there might be plenty of coffee cups not in use. Still, by counting only in a certain area of the camera's view, it has the potential to fit the frame of the project. TensorFlow models are easiest to convert into TensorFlow Lite. That makes it suitable for a Raspberry Pi application.

2.3 SSD-MobileNet V2 Trained on MS-COCO Data

This model is a single-stage object detection model that goes straight from image pixels to bounding box coordinates and class probabilities [4]. The model architecture is based on inverted residual structure where the input and output of the residual block are thin bottleneck layers as opposed to traditional residual models. Moreover, nonlinearities are removed from intermediate layers and lightweight depthwise convolution is used. The model's efficiency makes it highly suitable for any mobile application. To be applicable for this project, it needs the camera to monitor the whole room and potentially a second camera for the outside area. Transfer learning could be applied to integrate a custom data set into COCO, to fit this project's frame.

3 Methods

This section describes the workflow of the project. First, it presents the requirements for the project and why certain choices have been made concerning the use of platform, convNet and data set. Then, it describes the deployment of the chosen framework, in this order:

1. Use of MobileNet sample model with TensorFlow Lite on Raspberry Pi
2. Transfer Learning to enhance sample model
3. Custom data set creation
4. Custom model creation
5. Custom model training

3.1 Project Requirements & Framework

The proposed concept should be lightweight and autonomous, that is, it should not depend on or be integrated into the café's general IT system (especially since some cafés might not even have any IT system). The hardware requires a computer that is powerful enough for machine learning image processing, a camera, and needs to be WIFI-enabled. It should be small and inattentive. To prototype for such a purpose, the Raspberry Pi seems perfectly fit. As a machine learning platform, TensorFlow Lite was chosen. As the name infers, it is a light version of Google's TensorFlow API and designed and optimized specifically for mobile applications. As a convNet, the SSD MobileNet v2 was chosen, as it too is optimized for mobile applications. Throughout the project, this network has been used for basic experimentation, transfer learning attempts and custom model training.
All programming has been done in Google Colab. Multiple links to Colab notebooks will be shared throughout the project for the reader to follow the programming steps. The created models have then been exported to be used on a Raspberry Pi.

3.2 Raspberry Pi Data Structure

The general data structure on the Raspberry Pi is quite simple. The concept is initialized by running an executable main.sh program. This program opens two subprograms. One written in Python to run the image processing and deep learning algorithm. This program writes the data of interest (amount of people) to a text document. This data is accessed by the second program, written in JavaScript, which also displays it on a simple webserver. Even though the second program is not the focus of the project it is still presented for the purpose of completion. The data structure and program code can be found in this **notebook**.

3.3 MobileNet sample model with TensorFlow Lite on Raspberry Pi

TensorFlow is an open source, state-of-the-art machine learning platform. It provides a comprehensive, flexible ecosystem of tools, libraries and community resources. Its light version, TensorFlow Lite, is optimized for mobile and IoT applications and can run efficiently on the Raspberry Pi. In combination with the SSD-MobileNet v2, it runs with a frame rate of 4-5 fps. This is sufficient for the project requirements.
The basic idea of the MobileNet architecture is to use depthwise separable convolution to reduce the model size and complexity. It has two steps, first depthwise convolution and second, pointwise convolution. This makes it much more efficient, with little loss in accuracy.The MobileNet v2 builds upon the ideas from MobileNet v1, using depthwise separable convolution as efficient building blocks. However, V2 introduces two new features to the architecture: 1) linear bottlenecks between the layers, and 2) shortcut connections between the bottlenecks. The bottlenecks encode the model's intermediate inputs and outputs while the inner layer encapsulates the model's ability to transform from lower-level concepts, such as pixels, to higher level descriptors, such as image categories. Finally, as with traditional residual connections, shortcuts enable faster training and better accuracy.
As a first experimentation, the sample model SSD MobileNet v2 was deployed on the Raspberry Pi, as described in the before-mentioned **notebook** ($occupancy_monitor.py$). The model is pre-trained on COCO and detects 80 different objects with a confidence score between 0 and 1. With regard to people detection, it seemed a bit clumsy. An example output can be seen in Figure 1. The confidence score is rather low and more people are counted than currently in the picture present. This leaves great room for improvement!

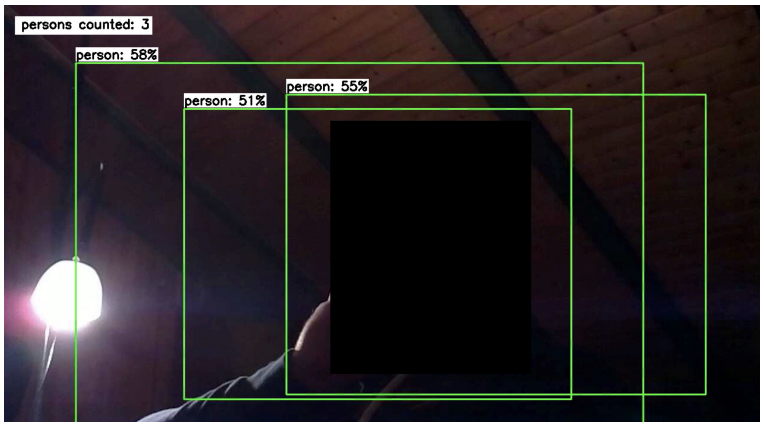

Figure 1: Sample model output image

3.4 Transfer Learning

As an attempt to improve the model with regard to people detection, a method called "transfer learning" was applied. For that matter, the guide in this **notebook** has been followed. It creates a tiny data set with custom images. Five random images of persons from Google Images with similar coloring and proportion as in Figure 1 have been used to fine-tune the pre-trained model. After conversion to the TFLite format it was deployed and gave a significantly improved output, as shown in Figure 2. The score is almost 20% higher, and no more ghosts are being counted.

3.5 Model Training

The customized model now detects people with a satisfactory confidence score. In order to differentiate between people that have already been served and those that have not, the model needs further customization. A first attempt was made to differentiate between normal people and those that have a coffee in their hands.

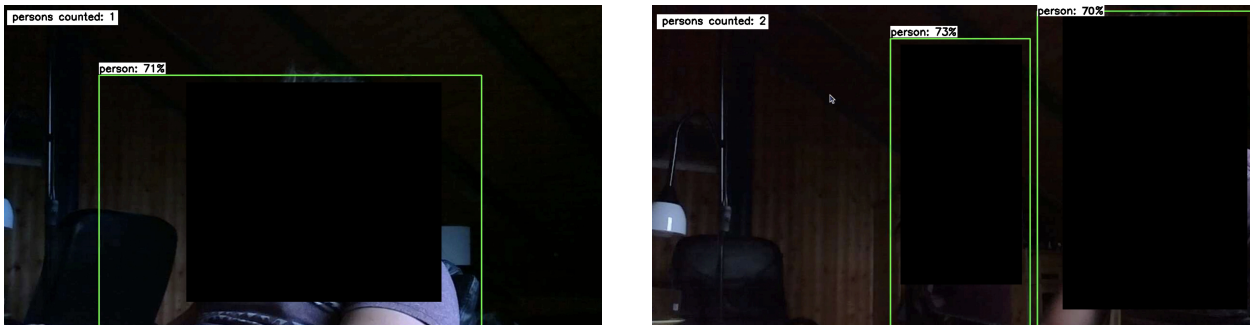

Figure 2: Transfer trained output images

3.5.1 Custom Data Set Creation

Using the online image labeling tool **CVAT**, a total of 59 images of diverse people with or without coffee have been labeled, using three different labels: coffee, person, and person-coffee. Figure 3 shows an example of one of those labeled images.

3.5.2 Custom Model Creation

The new custom data set was imported into **Roboflow**, an online tool for creating computer vision models of various formats. Roboflow allows to include essential preprocessing and augmentation steps and splits the set into training, validation and test images.
Preprocessing can reduce training time and increase inference speed by normalizing the images in the data set. The free version of Roboflow only allows auto-orienting and resizing, so those are the only steps taken in this set.
Augmentations create new training examples from the images in the data set for the model to learn from. By copying and then manipulating the images while keeping the label, the number of images in the set can be exponentially increased with little manual effort. Because most augmentation steps offered by Roboflow are not open source, the data set created in this project does not include any augmentation steps. But it seems worth mentioning that this would improve the final results significantly.
After preprocessing, the model was exported for the TensorFlow API, as there is no option for the TFLite format. Instead, the TensorFlow format was used for the training and then later converted into the TFLite format.

Figure 3: Example images from the data set

3.5.3 Custom Model Training

Because this essential step requires a lot of programming, a **notebook** has been created that describes the process in detail. It takes as input any model in the TensorFlow-format, allows to set certain training parameters, performs the training, and allows to download it. Additionally, steps for converting it into TFLite-format have been included to make the model ready for the Raspberry Pi. The notebook is an essential part of this report, as it combines multiple findings and applications from various sources into one complete source.
The goal was to train the model in such a way that the loss is constantly below 2. Even though that is still not incredibly good, it seems sufficient and adequate with regard to the data set size. Previous tests allowed to estimate that around 50,000 training steps would be suitable for those 59 images to get to that loss value. The initial step had a loss value of 20.203. In average, 4.72 steps per second were performed with a final loss value of 1.564.
Tensorboard is a tool offered by Google to visualize and monitor the training process. Unfortunately, it was discovered late in the process of the project, which is why the training process can not be documented into more detail. The only documentation is the written output inside the training section of the before-mentioned notebook. The steps for using Tensorboard have been implemented into the notebook after the training of the model for this project, for future use.

4 Results

To see the results of the conducted transfer learning , compare Figures and . The improvements are substantial, given that a very small data set of only five images was used. Even though it must also be stated, that the lighting in Figure was better. This leads to the first conclusion:

The quality of machine learning object detection depends to a large degree on the quality of the image.

It is also probable, that the improvement was so considerable because of the similarity between the output image and the images used for transfer learning. The second conclusion is therefore:

The greater the data set and the more similar to the object in camera, the better the result.

The training of a custom model was a bit less successful with regard to the confidence score, but it proved to be fairly easy to implement a complete new class of objects. Figure 4 shows the output of a video stream after running through the newly trained custom model.

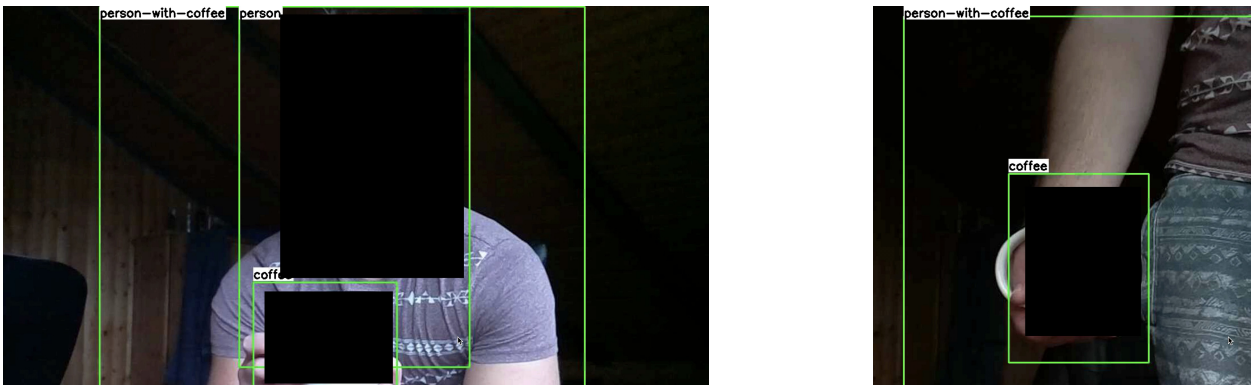

Figure 4: Output images from the trained custom model

The program successfully detects the newly introduced classes in the image. The confidence score (not displayed in the image) was between 15-35%. This is due to the small size of the data set. If the lighting is worsened, the program will not be able to detect the classes anymore. It is noteworthy, that the program can even differentiate between coffee and a person holding the coffee, as seen in image 2 of Figure 4. This leads to a valuable conclusion for a next iterative step that would be undertaken if the scope of the project was extended: the bounding boxes during the labeling process of creating a custom data set need not be drawn around the whole person, but rather just around the coffee with the attached hand or arm. This would blend out a lot of noise and probably improve the results further.

5 Discussion

The overall results are satisfying. Of course the model in its current stage would not be ready to be applied in a café yet and fulfil its purpose to a tolerable degree. But the presented use case is only an environment for the student to apply concepts and experiment with deep learning for visual recognition, not an actual product development process. Two highly valuable methods have been successfully applied: transfer learning and custom model training. These two methods allow a solid utilization of deep learning. Essential conclusions could be drawn with regard to how things could be improved and theoretical knowledge about network architecture could be confirmed in a practical application of an interesting and fun use case.
That being said, the scope of the project would probably have allowed the student to enhance the application further and dive deeper into the matter. Especially the training steps in detail could have been investigated more closely, and methods about how to prepare data sets could have been elaborated.
The project uses a state-of-the-art platform and convNet, but only applies them to a use case, not pushing it further or improving it in any way. Yet, necessary knowledge and skills for how to do that have been learned, especially in mobile applications. All together, the project's outcome fits well into the intended learning outcome of the Deep Learning for Visual Recognition course and possibly provides a suitable platform for future students to elaborate on.

References

[1] Andrew and Claire Bowen. Why the Wait. 2019.

[2] Narayanan Ramanathan. OpenCV People Counter. 2017.

[3] Google TensorFlow. TensorFlow Object Detection API. 2020.

[4] Julian W. Francis. SSD-MobileNet V2 Trained on MS-COCO Data. 2019.