energies

MDPI

Article

Embedded Real-Time Clothing Classifier Using One-Stage Methods for Saving Energy in Thermostats

Adán Medina [1,†], Juana Isabel Méndez [1,2,†], Pedro Ponce [1,2*,†], Therese Peffer [3,†] and Arturo Molina [1,2,†]

[1] Institute of Advanced Materials for Sustainable Manufacturing, Tecnologico de Monterrey, Monterrey 64849, NL, Mexico
[2] School of Engineering and Sciences, Tecnologico de Monterrey, Monterrey 64849, NL, Mexico
[3] Institute for Energy and Environment, University of California, Berkeley, CA 94720, USA
* Correspondence: pedro.ponce@tec.mx
† These authors contributed equally to this work.

Abstract: Energy-saving is a mandatory research topic since the growing population demands additional energy yearly. Moreover, climate change requires more attention to reduce the impact of generating more CO_2. As a result, some new research areas need to be explored to create innovative energy-saving alternatives in electrical devices that have high energy consumption. One research area of interest is the computer visual classification for reducing energy consumption and keeping thermal comfort in thermostats. Usually, connected thermostats obtain information from sensors for detecting persons and scheduling autonomous operations to save energy. However, there is a lack of knowledge of how computer vision can be deployed in embedded digital systems to analyze clothing insulation in connected thermostats to reduce energy consumption and keep thermal comfort. The clothing classification algorithm embedded in a digital system for saving energy could be a companion device in connected thermostats to obtain the clothing insulation. Currently, there is no connected thermostat in the market using complementary computer visual classification systems to analyze the clothing insulation factor. Hence, this proposal aims to develop and evaluate an embedded real-time clothing classifier that could help to improve the efficiency of heating and ventilation air conditioning systems in homes or buildings. This paper compares six different one-stage object detection and classification algorithms trained with a small custom dataset in two embedded systems and a personal computer to compare the models. In addition, the paper describes how the classifier could interact with the thermostat to tune the temperature set point to save energy and keep thermal comfort. The results confirm that the proposed real-time clothing classifier could be implemented as a companion device in connected thermostats to provide additional information to end-users about making decisions on saving energy.

Keywords: energy saving; clothing insulation; embedded system; thermal comfort; deep learning; computer vision; connected thermostat

Citation: Medina, A.; Méndez, J.I.; Ponce, P.; Peffer, T; Molina A. Embedded Real-Time Clothing Classifier Using One-Stage Methods for Saving Energy in Thermostats. *Energies* **2022**, *15*, 6117. https://doi.org/10.3390/en15176117

Academic Editor: Dimitrios Katsaprakakis

Received: 5 July 2022
Accepted: 18 August 2022
Published: 23 August 2022

1. Introduction

Since the United Nations proposed an annual meeting between representatives of the 190 nations to discuss climate change and address carbon dioxide (CO_2) emissions from fossil fuels, countries have become more engaged in addressing the subject seriously. Researchers have proposed alternative energy sources such as using the sun, the wind, nuclear energy, or harvesting geothermal energy. Nevertheless, despite recent implementations sanctioned by various governments, CO_2 emissions continue rising, and it took a global pandemic to change the expected 1% growth for CO_2 emissions. However, activities seem to be returning to normal, heightening the need for cleaner energy and making reducing unnecessary energy consumption more urgent.

Moreover, the number of household appliances has kept increasing since the 1960s, when the television, refrigerator, and washing machine were the main elements in a home.

Modern households currently contain many more energy-consuming appliances, depending on the number of persons in the household. Research indicates an average of 10 or more appliances inside a modern household with only one person living inside [1]. Therefore, it is imperative to evaluate which are the ones that consume the most energy and address their use to help the user operate them in the most energy-efficient manner.

One of the most energy-consuming appliances is the Heating and Ventilation Air-Conditioning System (HVAC). Research indicates that more than 60% of homes have these systems, and more than 85% have thermostats in their home [2]. Therefore, it is vital to avoid the unnecessary use of these systems to save energy. The method developed to approach an efficient use of such appliances is the concept of thermal comfort. The American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) defines Thermal Comfort as "the condition of the mind in which satisfaction is expressed with the thermal environment" [3]. Moreover, it takes into account six different factors to calculate thermal comfort: air temperature, radiant temperature, air speed, humidity, metabolic rate, and clothing insulation. Medina et al. [4] proposed to analyze clothing insulation to learn about householder garments' preferences and analyze their thermal comfort to send messages to interactive interfaces that suggest the householder use warmer or cooler garments to increase, maintain, or decrease the set point to promote energy reduction. However, this proposal lacked implementation on an embedded system. It just proposed an interactive interface in the MATLAB/Simulink R2021a environment.

Research about implemented interactive interfaces to engage householders in game-like activities that help them reduce energy have been proposed [2,5–8]. However, these proposals consider static activities and garments. The PMV/PPD and adaptive models calculate thermal comfort depending on several factors, considering the metabolic rate and clothing insulation in both models. Activity affects the metabolic rate and therefore affects the amount of heat the body produces. Moreover, different clothing garments affect the heat transfer between the user and the ambient air, representing a clothing insulation value. Some studies calculate the metabolic rate using thermal cameras or Kinect [9–13]. Regarding the clothing insulation classifier, Liu et al. [14] proposed using a thermal camera to dynamically estimate each person's clothing insulation and metabolic rate. Choi et al. [15] classified five classes of garments and proposed a method for estimating clothing insulation using deep-learning-based vision recognition by classifying five classes of garments. Medina et al. [4] classified eight classes of garments using the YOLOv3 model to propose its implementation in a connected mock-up thermostat.

1.1. One-Stage Object Detector and Classifiers

Object detection and classification have two main research lines: one-stage and two-stage algorithms. Two-stage algorithms detect objects and delimit them with bounding boxes; then, a second stage classifies the objects detected inside that image and better delimits the detected object; an example is the Mask R-CNN [16]. On the other hand, the one-stage detectors provide object detection and classification in a single pass; therefore, they are faster but less accurate than the two-stage algorithms. Hence, the one-stage methods are faster and require less processing power, making them suitable for a small system such as an embedded system.

Inside the one-stage classifiers, two main algorithms are presented as solutions: the Single Shot Detector (SSD) [17] and the You Only Look Once (YOLO) [18] algorithms. SSD is faster but less accurate, according to the last comparisons by YOLO authors [19]. YOLO has been integrating and outputting official variations to its algorithm more often, improving its performance.

YOLO is one of the most popular object detector algorithms, and it was created in 2016 by Redmon et al. [18]. The authors' main goal was to create an accurate deep learning algorithm that could be faster than the existing algorithms. They proposed to look at the entire image and produce predictions for bounding boxes as well as class probabilities for those boxes. Figure 1 exemplifies this concept by using a grid system to look into each

cell in the grid to produce the probabilities for the bounding boxes and classes. The different algorithms of YOLO change because different authors propose new versions of YOLO. Thus, a general review of their architectures and how they have evolved is presented.

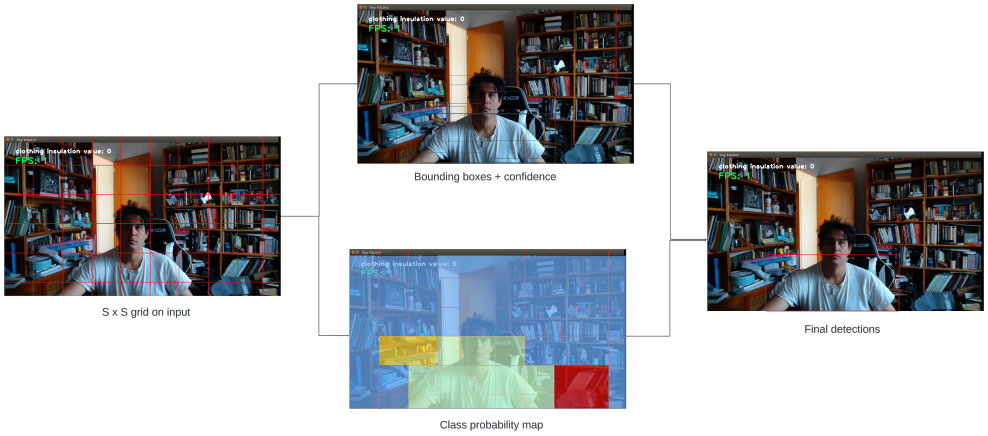


Figure 1. YOLO method.

1.1.1. YOLOv3 and Tiny YOLOv3

The YOLOv3 model was implemented in a Linux environment, allowing training with custom datasets. Moreover, this model includes the weights of pre-trained algorithms using the COCO dataset [20] to perform transfer learning or to train existing models.

The TinyYOLOv3 is a smaller version of the YOLOv3 with a minor architecture that make its implementation easy in a more computationally restricted system [19].

The main difference between the full YOLOv3 model and the Tiny YOLOv3 model is that the full model does not use pooling layers, and the Tiny YOLOv3 model does. Tiny YOLOv3 only performs recognition at two different sizes instead of three; however, due to the algorithm's size, it is much faster than the full model. Figure 2 pictures the YOLOv3 Model, and Table 1 depicts the Tiny YOLOv3 architecture.

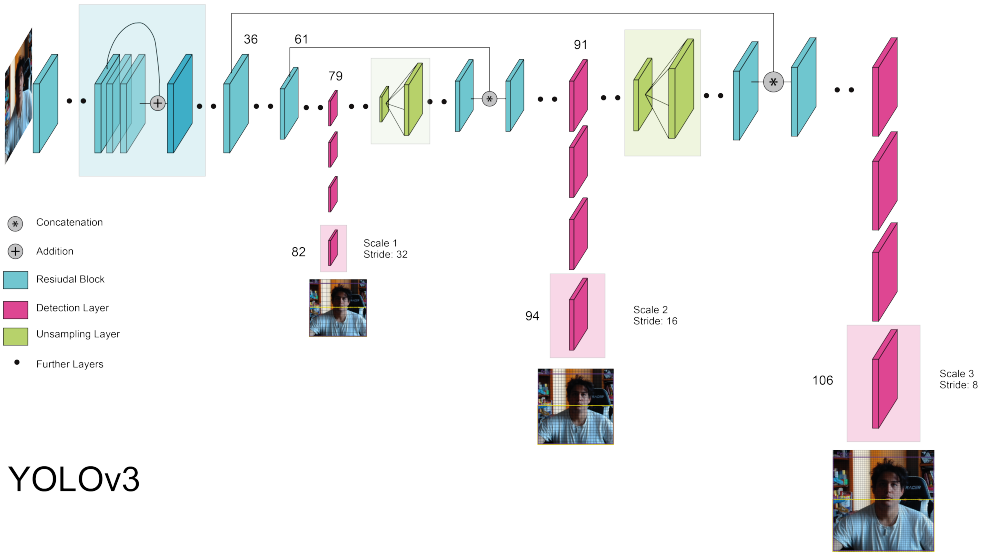


Figure 2. YOLOv3 architecture [19].

Table 1. Tiny YOLOv3 architecture.

Layer	Type	Filters	Size/Stride	Input	Ouput
0	Convolutional	16	3 × 3/1	416 × 416 × 3	416 × 416 × 16
1	Maxpool		2 × 2/2	416 × 416 × 16	208 × 208 × 16
2	Convolutional	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
3	Maxpool		2 × 2/2	208 × 208 × 32	104 × 104 × 32
4	Convolutional	64	3 × 3/1	104 × 104 × 32	104 × 104 × 64
5	Maxpool		2 × 2/2	104 × 104 × 64	52 × 52 × 64
6	Convolutional	128	3 × 3/1	52 × 52 × 64	52 × 52 × 128
7	Maxpool		2 × 2/2	52 × 52 × 128	26 × 26 × 128
8	Convolutional	256	3 × 3/1	26 × 26 × 128	26 × 26 × 256
9	Maxpool		2 × 2/2	26 × 26 × 256	13 × 13 × 256
10	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
11	Maxpool		2 × 2/1	13 × 13 × 512	13 × 13 × 512
12	Convolutional	1024	3 × 3/1	13 × 13 × 512	13 × 13 × 1024
13	Convolutional	256	1 × 1/1	13 × 13 × 1024	13 × 13 × 256
14	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
15	Convolutional	255	1 × 1/1	13 × 13 × 512	13 × 13 × 255
16	YOLO				
17	Route 13				
18	Convolutional	128	1 × 1/1	13 × 13 × 256	13 × 13 × 128
19	Up-sampling		2 × 2/1	13 × 13 × 128	26 × 26 × 128
20	Route 19 8	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
21	Convolutional	256	3 x 3/1	13 × 13 × 384	13 × 13 × 256
22	Convolutional	255	1 × 1/1	13 × 13 × 256	13 × 13 × 256
23	YOLO				

1.1.2. YOLOv4 and Tiny YOLOv4

Since the principal author of the original YOLO algorithm decided to stop the research on computer vision algorithms, some authors decided to take the algorithm and try to keep improving it since it was a high-speed algorithm. However, the accuracy was still below the two-stage algorithms. One of the models proposed is YOLOv4, developed by Bochovskiy et al. [21].

Hence, YOLOv4 has a variation of the DarkNet-53 used as backbone called CSPDarkNet53. This backbone was chosen after comparing the most common backbones, such as CSPDarkNet53 and CSPResNext50. In addition, the author followed Redmon's footsteps and provided the Tiny YOLOv4. There is no current comparison between the YOLOv4 and the Tiny YOLOv4.

Consequently, Figure 3 shows both algorithms. Figure 3a shows that YOLOv4 uses pooling layers but only for the newly added Spatial Pyramid Pooling (SPP) to improve the detection method by not modifying the input image size. The DarkNet53 backbone remains free of pooling layers. Figure 3b shows that the Tiny YOLOv4 continues to work with pooling layers; however, this newer version skips connections to try and help the algorithm retain more feature information.

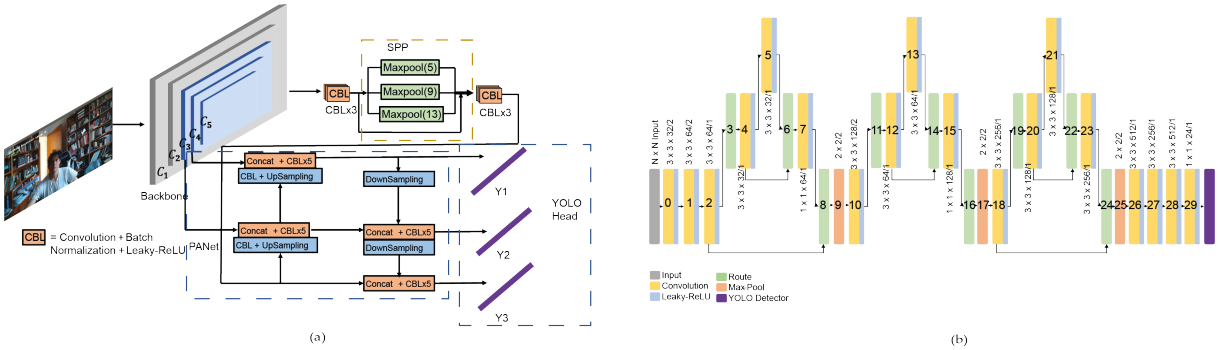


Figure 3. Architectures: (**a**) YOLOv4; and (**b**) Tiny YOLOv4.

1.1.3. YOLOv5

Jocher et al. [22,23] proposed the recent version of the YOLO algorithm, the YOLOv5. This algorithm is for public use because they changed the Linux environment requirement into an open source approach. They implemented YOLOv5 in Python through PyTorch rather than DarkNet. Therefore, they allowed Windows users to create virtual environments and to train it locally, for instance, in Anaconda, instead of Google Colab.

These authors proposed five models, from smallest to largest: YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x. Nevertheless, due to the lack of a scientific paper, there is no apparent difference in the models; Azam et al. [24] indicated that the difference relied on the scaling multipliers of the width and depth of the network. Therefore, the architecture is the same but the size changes. Thus, YOLOv5 algorithms enable more accessible training in computationally restricted environments than the previous versions of YOLO (See Figure 4).

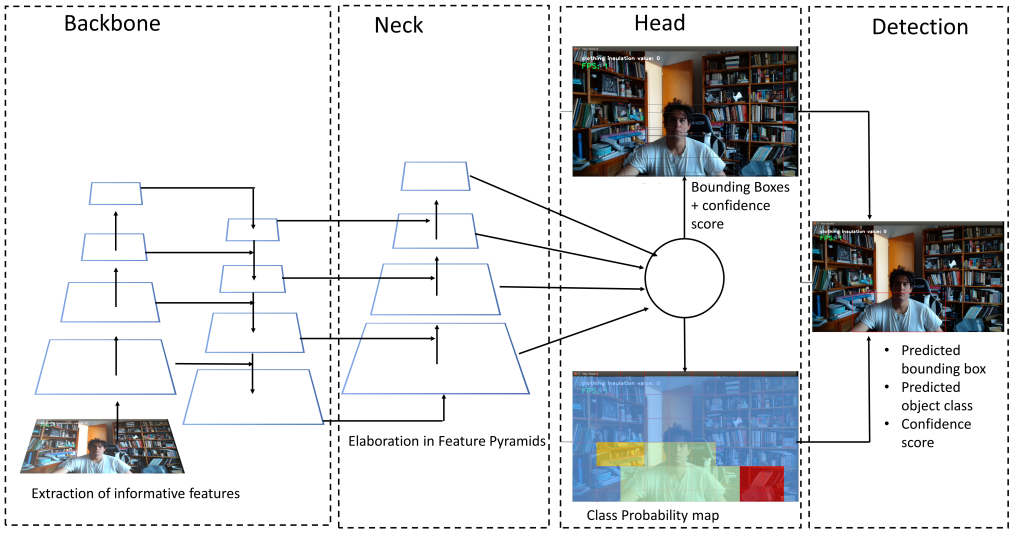


Figure 4. YOLOv5 architecture [24].

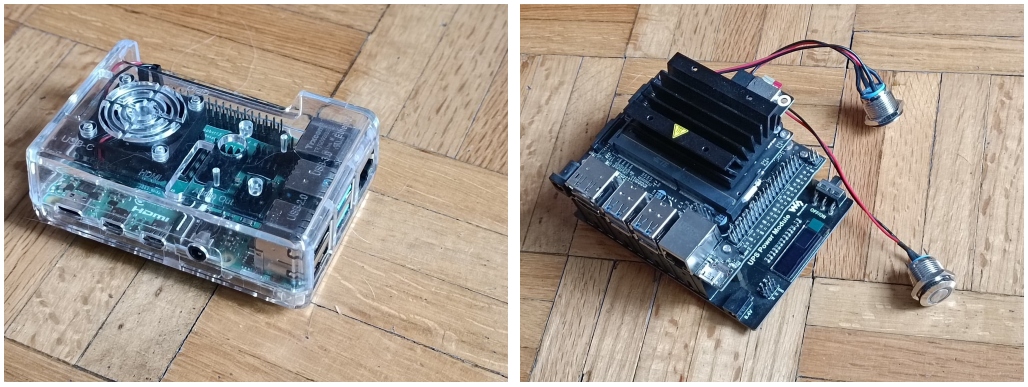
1.2. Embedded Systems

An embedded system has minimum hardware requirements based on a microprocessor designed for a specific purpose. Thus, most times it is custom-made. There are some development boards to perform tests before assembling the custom board, and these systems test simple functionalities and different sensor and actuator capabilities. Therefore, the main benefit of these boards is that they come with some Light Emitting Diodes (LEDs) to simulate the actuators' activation and some push buttons to simulate the inputs from a sensor. Moreover, compatible extension boards have been developed for different sensors or actuators, such as cameras, motor drivers, and wireless communications. Even though the development boards are not fully customized for a certain solution, they are microprocessor-based and practically next to the final custom embedded system solution. Hence, they are considered embedded systems.

Figure 5 shows the embedded systems used for this research. Figure 5a depicts the Raspberry Pi 4 board, and Figure 5b portrays the Jetson Nano board.

Raspberry Pi [25] focuses on offering a more robust development board while competing with the most expensive Arduino boards on the market. This development board does not have its IDE. However, the developers advertised it as a mini-computer, so instead, they developed its operating system called Raspbian. This operating system uses free distribution operating system called Debian, which is built on top of a Linux kernel [26]. Raspberry Pi 4 Model B is the most recent board; it has a Broadcom BCM2711B0 quad-core Cortex-A72 processor running at 1.5 GHz with a VideoCore VI 500 MHz GPU with 4 GB LPDDR4-3200 memory.

On the other hand, the company NVIDIA proposed their line of embedded system boards called Jetson. They are oriented toward Artificial Intelligence (AI) solutions. One of these boards is the Jetson Nano [27]; this board has a Quad-core ARM Cortex-A57 MPCore processor with an NVIDIA Maxwell with 128 CUDA cores GPU a 4 GB 64-bit LPDDR4-1600 memory.



(**a**) (**b**)

Figure 5. Embedded systems using in this research: (**a**) Raspberry Pi 4 board; and (**b**) Jetson Nano board.

1.3. Computer Vision in Embedded Systems

Some researchers have worked with embedded systems to implement computer vision algorithms for different purposes. According to Stankovic [28], there have been three major trends involving embedded systems and real-time solutions:

1. The development of the architecture used in the embedded systems has been growing fast and has achieved real-time constraints.
2. Achieve hard real-time results where missing a deadline means the failure of the solution.
3. Soft real-time solutions: If there is some failure in the real-time part of the solution, it is possible to manage and provide results avoiding the system's failure.

This paper focuses on soft real-time solutions. Jin et al. [29] modified the YOLOv3 algorithm to detect and classify pedestrians using a Jetson TX2 embedded system. With their customized architecture, the authors reached 9-10 frames per second (FPS) on the system. Chen et al. [30] customized another version of the YOLOv3 to detect cars in the street. They also work around the bit floating point difference for the embedded system to use resources in an embedded system efficiently. Thus, with their custom architecture and bit floating point calculation quantization, they achieved 28 FPS. However, the authors do not specify which embedded system they used for their tests. Therefore, it is feasible to implement solutions for energy savings at home, for instance, in a connected thermostat, to reduce energy consumption at home.

Murty et al. [31] discussed different algorithms tested with different embedded systems. Unfortunately, the results comparing such systems present older algorithms and only Powerful AI-oriented Boards such as Jetson TX2. Therefore, a new comparison is needed to assess whether existing algorithms can be used for object detection and classification to avoid needing custom-built architectures, hardware enhancers, or expensive AI-oriented embedded systems.

Taylor et al. [32] identified the best algorithm depending on the image input on the Jetson TX2 and compared the models with the Inception and ResNet algorithms. Dedeoglu et al. [33] proposed special computer vision libraries for embedded systems. Medina et al. [4] suggested that YOLO algorithms classified better clothing garments in real-time than the ResNet18, Inceptionv4, and VGG16 architectures. Therefore, this paper focuses on determining the best YOLO architecture with the fastest response in an embedded system. Thus, the computer vision system deploys the algorithms and analyzes the experiments to obtain

enough information to implement the clothing classification as a companion system in connected thermostats.

Figure 6 proposes how the connected thermostat interacts with the real-time clothing classifier and the end-user for reducing energy consumption at home according with [4,34,35]. The connected thermostat uses the real-time clothing classifier to determine the thermal comfort zone. Consequently, the temperature set point is adjusted to save energy. A message is displayed in the connected thermostat asking the householder to accept this new set point value. This paper focuses on the real-time clothing classifier as it is the main component in the proposed structure. This is carried out by determining which YOLO algorithm best fits the embedded system to provide information in real time. As a result, we obtain the best solution between the tested options, concluding that it is feasible to deploy this in a real environment.

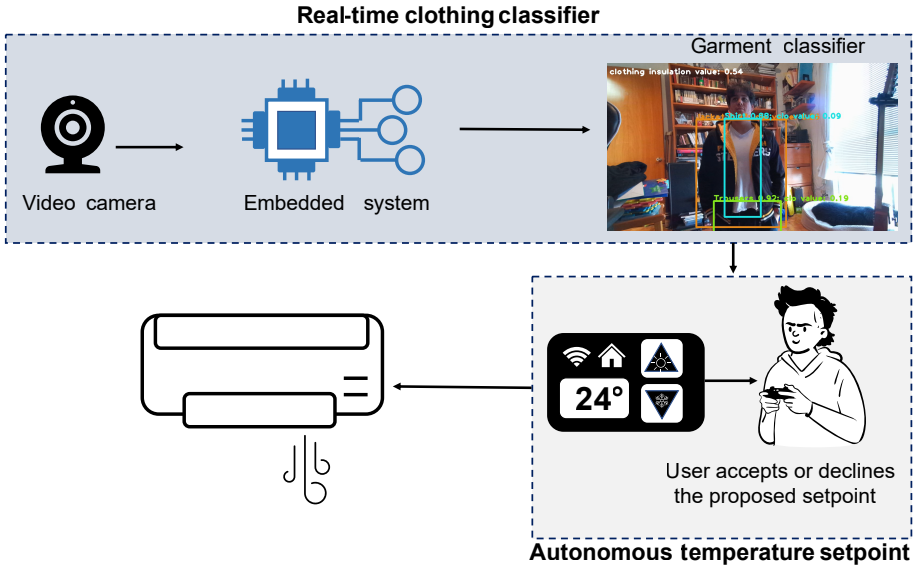


Figure 6. Proposed block diagram.

The remainder of this paper is as follows. Section 2 describes the methodology used for the clothing classifier and the embedded systems. Consequently, Section 3 describes and compares the results of each YOLO algorithm with each embedded system. Section 4 describes the scope of the research and discusses the advantages and disadvantages of each comparison. Finally, conclusions and suggestions for future work are presented in Section 5.

2. Methodology

Figure 7 presents the flow diagram of the real-time clothing classifier in which the programmable code loads the corresponding algorithm weights in order to just dedicate itself to the classification. then the whole loop starts, and the video capture commences to start obtaining the information from the camera, the time stamp is taken for later comparison, and the classification information such as bounding boxes, class, and confidence values are produced. Then the new time stamp is taken to obtain the difference in time and obtain the FPS with an equation presented later. This process repeats itself until the user stops the program and resets and restarts the variables.

This research compares already existing algorithms and applies them to embedded systems [19,21]. This comparison helps those who look for an algorithm to be implemented in an embedded system and even sees if the existing algorithms may help solve their problems. Nevertheless, many possibilities and many variables may affect the results presented in this paper. For example, hardware enhancers may increase the amount of FPS obtained, and code optimization or a network trained with different settings or datasets may improve or worsen its accuracy values.

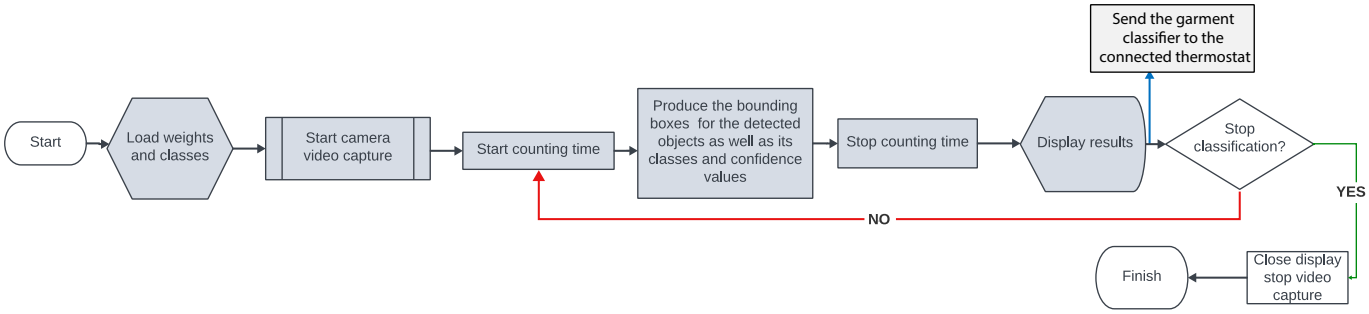


Figure 7. Real Time clothing classifier flow diagram.

Therefore, this section compares YOLO v3 with the Tiny YOLOv3, YOLOv4 with the Tiny YOLOv4, and three different sizes of YOLOv5 to determine which of these architectures has the fastest response in the implementation of the embedded system without losing accuracy. The accuracy test was performed multiple times in the same room, with variations on clothing garments of the same class, one hardware system after the other, to avoid significant lighting differences between classifications. Thus, the algorithms were tested on the computer and in the Raspberry Pi 4 and Jeston Nano.

One-stage classifiers were chosen to compare different algorithms for object detection and classification because they are faster and have fewer parameters to train, so the most common one-stage algorithms are Tiny YOLOv3 [19], YOLOv3 [19], YOLOv4 [21], Tiny YOLOv4 [21], YOLOv5n [22], YOLOv5s [22], and YOLOv5m [22], among other algorithms. Although the literature review showed little to few implementations of these algorithms in embedded systems and better customized the YOLO versions [29,30,36–38], this paper prefers using algorithms and training datasets for clothing garments to modifying them because it is easier for other researchers to replicate this research methodology than adapt algorithms.

For this custom dataset for clothing recognition, images were taken from the Internet and labeled. Figure 8 shows an example of this classification. Since the number of training images is a factor to be considered when choosing the hypertraining parameters, 2000 images were selected where there were multiple persons or just a single person in a frame and with different backgrounds. However, these images have a small dataset, so image transformations such as flipping, contrast changes, color changes, blurring, and other transformations were performed on all images to increase the training dataset and obtain a final 15,000-image dataset. Figure 9 shows an example of this type of transformation. These transformations also address the difference in camera resolution and lighting settings so that when implemented, cameras with less resolution or darker settings do not struggle too much performing the classifications, as well as removing the orientation variable out of the equation by using the flipping transformations.

The algorithms are trained using 16,000 epochs ashyperparameters, with alearning rate of0.01, astheauthors recommend forcustom training, and using theAdam Optimizer to have amore objective comparison. Then, these algorithms are implemented inacomputer that has aGeForce RTX 2080 Ti GPU with 11 GB GGDR6 RAM and anAMD Ryzen 3950 12 core 3.5 GHz processor with 64 GB ofRAM memory (PC). Thetrained models run using OpenCV toaccess thecamera and theresulting weights for each model toobtain theprediction inareal-time implementation and measure accuracy and Frames Per Second (FPS).

The YOLOv3, Tiny YOLOv3, and Tiny YOLOv4 algorithms were trained using Google Colab due to the fact that there was no access to a Linux environment, and Colab allows us to use the Darknet framework these algorithms have. So, in order to keep the comparison as fair as possible (since the author of the YOLOv4 algorithm has stated that the comparison presented by the author of the YOLOv5 model was using the version he implemented with PyTorch framework and not the native Darknet framework which affects the algorithms results) the authors selected the one using the Darknet framework.

Figure 8. Original image.

Figure 9. Image transformations.

The training time for the YOLOv3 algorithm was around 300 h (an approximate time is used since Colab allows access to Cloud GPU computing for 4 h a day, and you cannot choose the GPU, so it may change from day to day), the Tiny YOLOv3 took around 120 h, and the Tiny YOLOv4 took around 90 h.

YOLOv5 implemented an early stop in the training code, so if the Mean Average Precision (mAP) does not change after 100 epochs, the training stops; therefore, training for the YOLOv5 algorithms was shorter and only reached an average of 350 epochs. There is the option to change this parameter. However, after changing it to have an early stop after 500 epochs, the results of the mAP were the same for the trained algorithm, and since the point is to train all the algorithms in the same manner, no more tuning to the param-

eters was performed. Before the comparison, a quick test was made to see if the YOLOv5 algorithms managed to detect and classify objects with such short training, and since they did, the comparison continued as planned. The training times for the YOLOv5 algorithms were all around the 3 h mark.

The comparison focuses on two metrics: accuracy and FPS. The accuracy was measured using an object detector threshold of 70 to avoid miss-classifications of detected objects that are not within the desired results. Therefore, the accuracy uses the percentage of different clothing garments recognized with different real-time tests. It is performed with a Logitech HD Pro C920 Web Camera connected to the different systems in the same room at different times during the day to leave out possible bias due to different lighting settings caused by the difference in day times.

The FPS were measured using the time library available in Python and is calculated measuring the time stamp of the model when it loads the image and after then applying Equation (1). Since the program loads the image again, a measurement of frames per second, analysis can be obtained.

$$FPS = 1/(newtimestamp - previoustimestamp) \quad (1)$$

Three different clothing garments are chosen for eight classes in order to have some different colors and textures to present different possibilities, such as the examples shown in Figure 10. Table 2 depicts the classes analyzed. Each class was tested for every hardware system in a consecutive manner to avoid having too much difference in light settings due to the difference in day times.

Table 2. Clothing insulation values considered for the classes.

Label	Garment	(clo) [1]
0	Highly insulating jacket, multicomponent	0.40
1	Highly insulating shoes, boots	0.10
2	Jacket, no buttons	0.26
3	T-Shirt	0.09
4	Trousers (straight, fitted)	0.19
5	Shoes	0.04
6	Warm winter cap	0.03
7	A-Line, knee-length	0.15

[1] 1 clo = 0.155 m^2 C/W.

Figure 10. Example of three different garments for two classes.

3. Results

Figure 11 shows the results in the PC. Figure 11 shows the Jetson Nano results for a seated individual with a white T-shirt and with the display turned on. Figure 12 depicts the Raspberry Pi 4 results with the display turned on, too. These figures show how the system worked in each one. Although the computer is above the hardware capabilities of an average personal computer (PC), it is still not good enough for an AI-oriented computer that requires at least a Titan X or Tesla V100 GPU.

Tables 3 and 4 present the information on both embedded systems but with the display turned off. Thus, the results were printed on the console. The columns correspond to the model the results refer to, the Highest FPS value the model reached, the Average FPS value the model had in 10 seconds run time, the accuracy of the model recognizing different clothing garments, and the average class probability the model produced for a white shirt test.

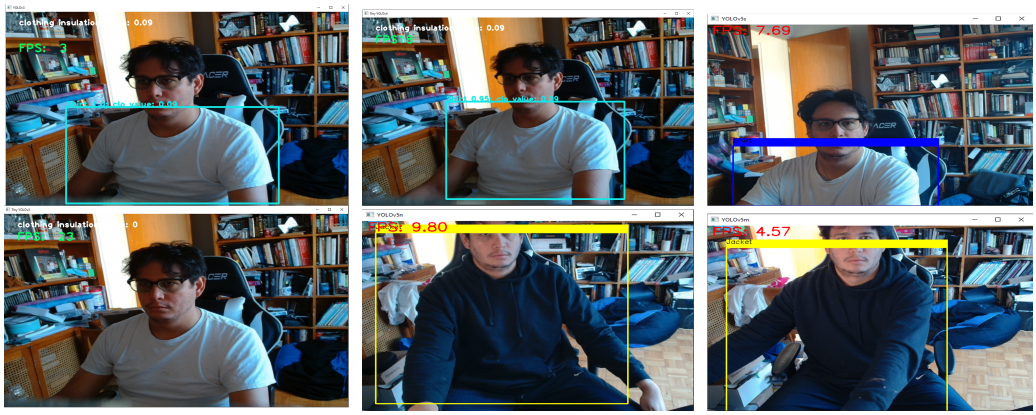


Figure 11. PC Results.

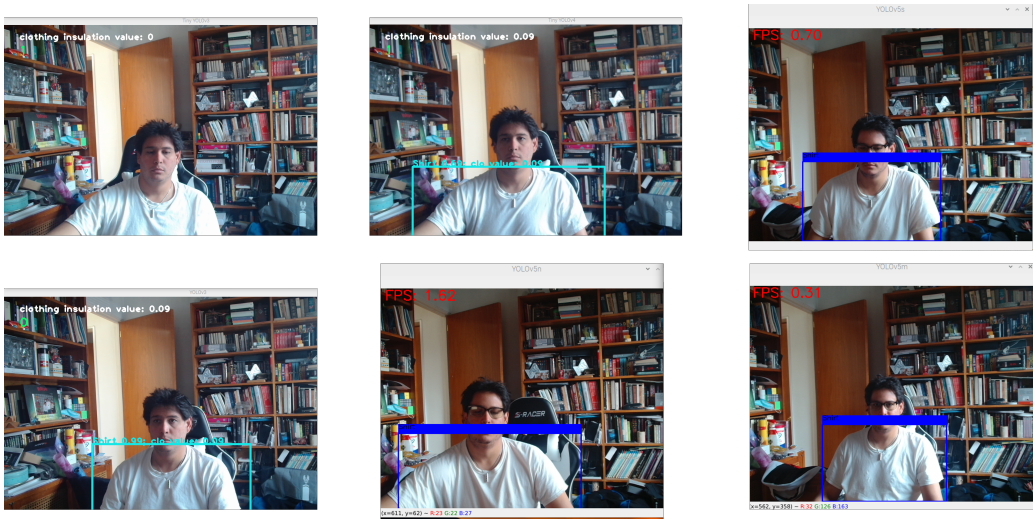


Figure 12. Raspberry Pi 4 Results.

Table 3. Jetson Nano results comparing the YOLO models.

Model	Highest FPS	Average FPS	Accuracy (%)	Class Probability Average (%)
Tiny YOLOv3	2.01	1.97	0	0
YOLOv3	0.23	0.22	76	99
Tiny YOLOv4	0.98	0.96	76	96
YOLOv5n	1.49	1.47	71	86
YOLOv5s	0.71	0.69	71	91
YOLOv5m	0.30	0.29	76	93

Table 4. Raspberry Pi 4 results comparing the YOLO models.

Model	Highest FPS	Average FPS	Accuracy (%)	Class Probability Average (%)
Tiny YOLOv3	2.01	1.97	0	0
YOLOv3	0.23	0.23	82	1.0
Tiny YOLOv4	1.88	1.84	79	97
YOLOv5n	1.60	1.59	76	80
YOLOv5s	0.72	0.70	67	92
YOLOv5m	0.30	0.29	70	90

After observing the results with the display active, the clear difference between the computer and both embedded systems is noticeable. All models suffer a loss of about 80% in the FPS metric they could produce. However, Raspberry Pi 4 seems to perform better than the Jetson Nano, which was unexpected. A clear difference can be seen comparing Figures 12 and 13, where the full YOLOv3 model crashed when running in the Jetson Nano (that is why there is a grayscale image), and even though it had its fair share of problems, it does not crash in the Raspberry Pi 4.

In addition, comparing the tests performed with the display turned off with the displays turned on, the FPS metric does not differ by a significant amount in the Raspberry Pi 4, and even though in the Jetson Nano, it helped the bigger models such as YOLOv3, YOLOv5s, and YOLOv5m the difference was not that great. The only difference made was to turn off the display, so a code optimization to eliminate unnecessary things may increase the models' performance. However, the display alone was not responsible for downgrading the smaller models' performance by comparing the FPS values in Figures 12 and 13 with those in Tables 3 and 4.

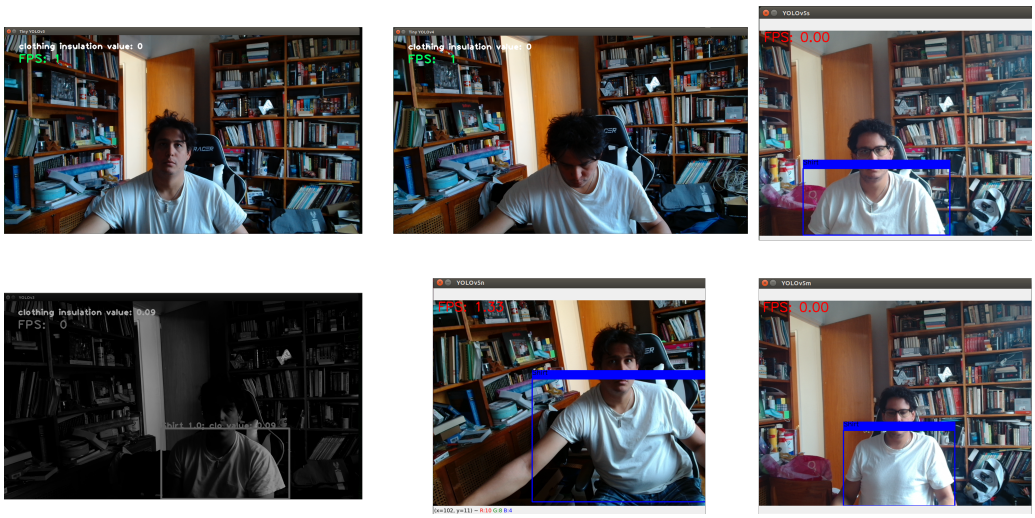


Figure 13. Jetson Nano Results.

4. Discussion

Both PMV/PPD and adaptive methods share the met and clothes factors. However, this proposal considers the activity factor static, whereas the clothing insulation is dynamic. In Medina et al. [4], the authors had previously simulated the clothing insulation using MATLAB/Simulink. They found that the HVAC energy consumption decreased from 18% to 47% by providing feedback about the clothing insulation. Thus, succeeding in those simulated reductions, this paper focuses on implementing this solution in embedded systems. Therefore, future work includes deploying this embedded system into a connected thermostat and implementing it in a physical space to analyze the impact of reducing energy while maintaining thermal comfort.

These FPS results are similar to those presented in [39]. Although 3 to 7 FPS is a significant difference, it would not be enough for implementation in an autonomous car. It is not necessary to achieve these FPS for calculating the clothing insulation value. Thermal comfort models usually consider the clothing for larger periods of time, for instance, during a day, week, or month. Hence, the FPS achieved in this proposal helps provide new ways of saving energy by taking advantage of current technology and using deep learning computer vision algorithms in embedded systems.

In addition, the accuracy results showed little change between all hardware systems. Although the models implemented in the Raspberry Pi 4 showed better performance compared to the models implemented in the Jetson Nano by about a 3% difference, the number of tests is not enough to conclude that the hardware system affects the accuracy of the model significantly. Recognizing one of the three chosen clothing garments for each class would increase the accuracy value by almost 5%. Moreover, comparing each model with the PC results, the difference is practically non-existent, only 1% for each model between hardware implementations and the PC results.

5. Conclusions

This paper proposes a real-time clothing classifier to be implemented in a connected thermostat that does not require more than seven or eight FPS. Six YOLO models were successfully implemented and evaluated in a Raspberry Pi 4 and the Jetson Nano. A complete analysis of these embedded systems was conducted.

The results show that embedded systems and existing algorithms achieve soft real-time acquisition. In this application, one or two FPS is sufficient to classify clothing. The speed and accuracy of algorithms such as Tiny YOLOv4 and YOLOv5n are enough to be considered a good alternative in this proposal.

YOLOv3 is discarded from this application because even though it had one of the best accuracy results, the processing speed required to obtain classifications is not fast enough without a hardware enhancer. It crashed on the Jetson Nano and ran very slowly on the Raspberry Pi 4. For the Tiny YOLOv3 model, even though it is the fastest model since it achieved FPS values that doubled the FPS of the rest of the algorithms, it lacks accuracy since it does not deliver almost any classifications. Therefore, a more robust training dataset is sufficient to fix this problem. As for the YOLOv5 models, choosing which one is the best for any situation would be better considering that it is the same base architecture, and only the depth changes with each version. However, for embedded systems, the best is YOLOv5n since the accuracy results do not change much, but the FPS is higher on the YOLOv5n algorithm.

The option for choosing a model between Tiny YOLOv4 and YOLOv5n would be Tiny YOLOv4 because it had fewer missed object detections and showed the same speed and increased accuracy as the YOLOv5n model. Nevertheless, the training time difference between models was vastly different. When Google Colab has used, the Tiny YOLOv4 needed more time to be trained than YOLOv5n. The FPS values were similar and presented the most variations between embedded systems, with almost one FPS difference in the case of Tiny YOLOv4. In addition, the results of both embedded systems showed that these systems have the requirements for running computer vision deep learning al-

gorithms. They present results up to one FPS without any help of hardware enhancers or custom algorithms.

Author Contributions: Conceptualization, A.M. (Adan Medina), J.I.M. and P.P.; methodology, A.M. (Adan Medina); software, A.M. (Adan Medina); validation, P.P., T.P. and A.M. (Arturo Molina); formal analysis, A.M. (Adan Medina); investigation, A.M. (Adan Medina) and J.I.M.; resources, P.P. and A.M. (Arturo Molina); data curation, A.M. (Adan Medina); writing—original draft preparation, A.M. (Adan Medina), J.I.M. and P.P.; writing—review and editing, A.M. (Adan Medina), J.I.M. and P.P.; visualization, A.M. (Adan Medina); supervision, P.P., T.P. and A.M. (Arturo Molina); project administration, P.P. and A.M. (Arturo Molina); funding acquisition, P.P. and A.M. (Arturo Molina). All authors have read and agreed to the published version of the manuscript.

Funding: This research project is supported by the Institute of Advanced Materials for Sustainable Manufacturing, Tecnologico de Monterrey and CITRIS under the collaboration ITESM-CITRIS Smart thermostat, deep learning, and gamification project (https://citris-uc.org/2019-itesm-seed-funding/ (accessed on 25 October 2021)). Agreement: TECNOLÓGICO DE MONTERREY—CITRIS 2019.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to acknowledge the financial and the technical support of the Institute of Advanced Materials for Sustainable Manufacturing and Tecnologico de Monterrey, in the production of this work.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MDPI	Multidisciplinary Digital Publishing Institute
DOAJ	Directory of open access journals
PMV	Predicted Mean Vote
PPD	Predicted Percentage of Dissatisfied
AHSRAE	American Society of Heating, Refrigerating and Air-Conditioning Engineers
HVAC	Heating and Ventilation Air-Conditioning System
DNN	Deep Neural Network
CNN	Convolutional Neural Network
YOLO	You Only Look Once
SSD	Single Shot Detector
SPP	Spatial Pyramid Pooling
LED	Light Emitting Diode
IDE	Integrated Development Environment
AI	Artificial Intelligence
FPS	Frames Per Second
R-CNN	Region-based Convolutional Neural Network
mAP	Mean Average Precision

References

1. Won, A.N.; Hong, W.H. A survey on ownership of home appliances and electric energy consumption status according to the number of household member. *Appl. Mech. Mater.* **2014**, *672*, 2165–2168. [CrossRef]
2. Méndez, J.I.; Peffer, T.; Ponce, P.; Meier, A.; Molina, A. Empowering saving energy at home through serious games on thermostat interfaces. *Energy Build.* **2022**, *263*, 112026. [CrossRef]
3. *ANSI/ASHRAE Standard 55-1992*; Thermal Environmental Conditions for Human Occupancy. American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc.: Atlanta, GA, USA, 1992. Available online: https://www.ashrae.org/about/ashrae-en-espa%C3%B1ol (accessed on 6 June 2022).
4. Medina, A.; Méndez, J.I.; Ponce, P.; Peffer, T.; Meier, A.; Molina, A. Using Deep Learning in Real-Time for Clothing Classification with Connected Thermostats. *Energies* **2022**, *15*, 1811. [CrossRef]

5. Ávila, M.; Méndez, J.I.; Ponce, P.; Peffer, T.; Meier, A.; Molina, A. Energy Management System Based on a Gamified Application for Households. *Energies* **2021**, *14*, 3445. [CrossRef]
6. Méndez, J.I.; Ponce, P.; Medina, A.; Meier, A.; Peffer, T.; McDaniel, T.; Molina, A. Human-machine interfaces for socially connected devices: From smart households to smart cities. In *Multimedia for Accessible Human Computer Interfaces*; Springer: Cham, Switzerland, 2021; pp. 253–289. [CrossRef]
7. Méndez, J.I.; Ponce, P.; Miranda, O.; Pérez, C.; Cruz, A.P.; Peffer, T.; Meier, A.; McDaniel, T.; Molina, A. Designing a consumer framework for social products within a gamified smart home context. In Proceedings of the International Conference on Human-Computer Interaction, Washington, DC, USA, 14–19 March 2021; Springer International Publishing: Cham, Switzerland, 2021; pp. 429–443. [CrossRef]
8. Ponce, P.; Meier, A.; Méndez, J.I.; Peffer, T.; Molina, A.; Mata, O. Tailored gamification and serious game framework based on fuzzy logic for saving energy in smart thermostats. *J. Clean. Prod.* **2020**, *262*, 121167. [CrossRef]
9. Na, H.; Choi, H.; Kim, T. Metabolic rate estimation method using image deep learning. In *Building Simulation*; Springer: Beijing, China, 2020; Volume 13; pp. 1077–1093.
10. Na, H.; Choi, J.H.; Kim, H.; Kim, T. Development of a human metabolic rate prediction model based on the use of Kinect-camera generated visual data-driven approaches. *Build. Environ.* **2019**, *160*, 106216. [CrossRef]
11. Na, H.; Kim, T. Development of metabolic rate prediction model using deep learning via Kinect camera in an indoor environment. In Proceedings of the IOP Conference Series: Materials Science and Engineering, Singapore, 5–11 November 2019; Volume 609, p. 42036.
12. Vrigkas, M.; Nikou, C.; Kakadiaris, I.A. A review of human activity recognition methods. *Front. Robot. AI* **2015**, *2*, 28. [CrossRef]
13. Hasan, M.H.; Alsaleem, F.; Rafaie, M. Sensitivity study for the PMV thermal comfort model and the use of wearable devices biometric data for metabolic rate estimation. *Build. Environ.* **2016**, *110*, 173–183. [CrossRef]
14. Liu, J.; Foged, I.W.; Moeslund, T.B. Clothing Insulation Rate and Metabolic Rate Estimation for Individual Thermal Comfort Assessment in Real Life. *Sensors* **2022**, *22*, 619. [CrossRef]
15. Choi, H.; Na, H.; Kim, T.; Kim, T. Vision-based estimation of clothing insulation for building control: A case study of residential buildings. *Build. Environ.* **2021**, *202*, 108036. [CrossRef]
16. Dollár, K.H.G.G.P.; Girshick, R. Mask r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 19–22 October 2017; pp. 2961–2969.
17. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 8–16 November 2016; Springer: Cham, Switzerland, 2016; pp. 21–37.
18. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on computer vision and pattern recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
19. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
20. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft coco: Common objects in context. In Proceedings of the European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; Springer: Cham, Switzerland, 2014; pp. 740–755.
21. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv* **2020**, arXiv:cs.CV/2004.10934.
22. Jocher, G.; Chaurasia, A.; Stoken, A.; Borovec, J.; NanoCode012; Kwon, Y.; Xie, T.; Michael, K.; Fang, J.; imyhxy; et al. Ultralytics/yolov5: v6.1—TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference, 2022. Available online: https://zenodo.org/record/7002879#.Yv_1ihzMJD8 (accessed on 6 June 2022).
23. Differences between YOLOv5 Models. Available online: https://github.com/ultralytics/yolov5/issues/7152 (accessed on 20 June 2022).
24. Azam, M.A.; Sampieri, C.; Ioppi, A.; Africano, S.; Vallin, A.; Mocellin, D.; Fragale, M.; Guastini, L.; Moccia, S.; Piazza, C.; et al. Deep Learning Applied to White Light and Narrow Band Imaging Videolaryngoscopy: Toward Real-Time Laryngeal Cancer Detection. *Laryngoscope* **2021**, *132*, 9. 10.1002/lary.29960 . [CrossRef] [PubMed]
25. Raspberry Pi 4 Model B. Available online: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (accessed on 6 June 2022).
26. Severance, C. Eben upton: Raspberry pi. *Computer* **2013**, *46*, 14–16. [CrossRef]
27. NVIDIA Jetson Nano. Available online: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (accessed on 6 June 2022).
28. Stankovic, J.A. Real-time and embedded systems. *ACM Comput. Surv. (CSUR)* **1996**, *28*, 205–208. [CrossRef]
29. Jin, Y.; Wen, Y.; Liang, J. Embedded real-time pedestrian detection system using YOLO optimized by LNN. In Proceedings of the 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), Istanbul, Turkey, 12–13 June 2020; pp. 1–5.
30. Chen, S.; Lin, W. Embedded system real-time vehicle detection based on improved YOLO network. In Proceedings of the 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Chongqing, China, 11–13 October 2019; pp. 1400–1403.

31. Murthy, C.B.; Hashmi, M.F.; Bokde, N.D.; Geem, Z.W. Investigations of object detection in images/videos using various deep learning techniques and embedded platforms—A comprehensive review. *Appl. Sci.* **2020**, *10*, 3280. [CrossRef]
32. Taylor, B.; Marco, V.S.; Wolff, W.; Elkhatib, Y.; Wang, Z. Adaptive deep learning model selection on embedded systems. *ACM Sigplan Not.* **2018**, *53*, 31–43. [CrossRef]
33. Dedeoğlu, G.; Kisačanin, B.; Moore, D.; Sharma, V.; Miller, A. An optimized vision library approach for embedded systems. In Proceedings of the CVPR 2011 WORKSHOPS, Colorado Springs, CO, USA, 20–25 June 2011; pp. 8–13.
34. Méndez, J.I.; Medina, A.; Ponce, P.; Peffer, T.; Meier, A.; Molina, A. A real-time adaptive thermal comfort model for sustainable energy in interactive smart homes: Part I. In Proceedings of the International Conference on Smart Multimedia, Marseille, France, 25–27 August 2022; Springer: Berlin/Heidelberg, Germany, 2022.
35. Medina, A.; Méndez, J.I.; Ponce, P.; Peffer, T.; Meier, A.; Molina, A. A real-time adaptive thermal comfort model for sustainable energy in interactive smart homes: Part II. In Proceedings of the International Conference on Smart Multimedia, Marseille, France, 25–27 August 2022; Springer: Berlin/Heidelberg, Germany, 2022.
36. Mao, H.; Yao, S.; Tang, T.; Li, B.; Yao, J.; Wang, Y. Towards real-time object detection on embedded systems. *IEEE Trans. Emerg. Top. Comput.* **2016**, *6*, 417–431. [CrossRef]
37. Said, Y. Pynq-YOLO-Net: An embedded quantized convolutional neural network for face mask detection in COVID-19 pandemic era. *Int. J. Adv. Comput. Sci. Appl.* **2020**, *11*, 100–106. [CrossRef]
38. Shafiee, M.J.; Chywl, B.; Li, F.; Wong, A. Fast YOLO: A fast you only look once system for real-time embedded object detection in video. *arXiv* **2017**, arXiv:1709.05943.
39. Mazzia, V.; Khaliq, A.; Salvetti, F.; Chiaberge, M. Real-time apple detection system using embedded systems with hardware accelerators: An edge AI application. *IEEE Access* **2020**, *8*, 9102–9114. [CrossRef]