applied sciences

MDPI

Article

TinyML-Based Concept System Used to Analyze Whether the Face Mask Is Worn Properly in Battery-Operated Conditions

Dominik Piątkowski and Krzysztof Walkowiak *

Faculty of Information and Communication Technology, Wrocław University of Science and Technology, 50-370 Wrocław, Poland; 248833@student.pwr.edu.pl
* Correspondence: krzysztof.walkowiak@pwr.edu.pl

Abstract: As the COVID-19 pandemic emerged, everyone's attention was brought to the topic of the health and safety of the entire human population. It has been proven that wearing a face mask can help limit the spread of the virus. Despite the enormous efforts of people around the world, there still exists a group of people that wear face masks incorrectly. In order to provide the best level of safety for everyone, face masks must be worn correctly, especially indoors, for example, in shops, cinemas and theaters. As security guards can only handle a limited area of the frequently visited objects, intelligent sensors can be used. In order to mount them on the shelves in the shops or near the cinema cash register queues, they need to be capable of battery operation. This restricts the sensor to be as energy-efficient as possible, in order to prolong the battery life of such devices. The cost is also a factor, as cheaper devices will result in higher accessibility. An interesting and quite novel approach that can answer all these challenges is a TinyML system, that can be defined as a combination of two concepts: Machine Learning (ML) and Internet of Things (IoT). The TinyML approach enables the usage of ML algorithms on boards equipped with low-cost, low-power microcontrollers without sacrificing the classifier quality. The main goal of this paper is to propose a battery-operated TinyML system that can be used for verification whether the face mask is worn properly. To this end, we carefully analyze several ML approaches to find the best method for the considered task. After detailed analysis of computation and memory complexity as well as after some preliminary experiments, we propose to apply the K-means algorithm with carefully designed filters and a sliding window technique, since this method provides high accuracy with the required energy-efficiency for the considered classification problem related to verification of using the face mask. The STM32F411 chip is selected as the best microcontroller for the considered task. Next, we perform wide experiments to verify the proposed ML framework implemented in the selected hardware platform. The obtained results show that the developed ML-system offers satisfactory performance in terms of high accuracy and lower power consumption. It should be underlined that the low-power aspect makes it possible to install the proposed system in places without the access to power, as well as reducing the carbon footprint of AI-focused industry which is not negligible. Our proposed TinyML system solution is able to deliver very high-quality metric values with accuracy, True Positive Ratio (TPR), True Negative Ratio (TNR), precision and recall being over 96% for masked face classification while being able to reach up to 145 days of uptime using a typical 18650 battery with capacity of 2500 mAh and nominal voltage of 3.7 V. The results are obtained using a STM32F411 microcontroller with 100 MHz ARM Cortex M4, which proves that execution of complex computer vision tasks is possible on such low-power devices. It should be noted that the STM32F411 microcontroller draws only 33 mW during operation.

Keywords: TinyML; Machine Learning; embedded systems; computer vision; COVID-19; face masks

Citation: Piątkowski, D.; Walkowiak, K. TinyML-Based Concept System Used to Analyze Whether the Face Mask Is Worn Properly in Battery-Operated Conditions. *Appl. Sci.* **2022**, *12*, 484. https://doi.org/10.3390/app12010484

Academic Editor: Antonio Fernández-Caballero

Received: 15 September 2021
Accepted: 21 December 2021
Published: 4 January 2022

1. Introduction

Machine Learning (ML) is one of the methods used in the field of Artificial Intelligence (AI). In recent years, the computational complexity of widely used algorithms has been growing exponentially [1]. This is caused by the combination of increasingly fast processor

speeds, and the presence of Big Data and progressively powerful graphics processing units. On one hand, allocating more resources enables training and using state-of-the-art models capable of achieving results similar to humans, or even surpassing them. However, on the other hand, training and using such models requires increasingly advanced hardware and enormous amounts of electricity [1,2]. This poses a handful of problems, and one of them is the carbon footprint of an AI-focused industry, which is not negligible [1]. The popularization of cloud computing is not helping either, as it additionally strains the network infrastructure. One of the solutions to these problems is TinyML—a combination of ML and embedded Internet of Things (IoT) devices. This approach evades possible concerns caused by cloud solutions, such as a potential lack of privacy, introduction of latency, additional network strain, as well as non-negligible power consumption [3].

The latency happens because of the nature of cloud computing; a typical process consists of gathering the data, sending it to the server, waiting for a response, and acting accordingly. This is heavily dependent on network speed and load. In case of slow and/or loaded network infrastructures, the device will be less responsive and less reliable than the TinyML one that does all the computations by itself [3]. Another factor connected to that topic is the network strain—an inherent part of cloud-computing—as the data acquired by the sensors need to be sent to a server, and the response from that server also needs to be sent back to the device, and this can generate unnecessary network traffic that could be avoided by using the TinyML approach.

Power consumption is also important, as factor-data transmission requires more energy than locally computing the results, even if the microcontroller is at the maximum workload level for a significant amount of time [3,4]. As the WeMOS D1 Mini board based on the ESP8266EX Wi-Fi chip requires 185 mA during operation [5], this is current consumption that is an order of magnitude higher than any TinyML-capable microcontroller, for example, STM32F411 [6].

The system reliability is also important—transmitting the data over the unpredictable, lossy wireless channels from the device to the cloud poses a handful of problems, including privacy breach/data loss, data compromise, as well as malicious data modification [3]. The TinyML approach is free from these problems, as its working principles are very different from cloud computing—the data are kept within the device and the communication is reduced to minimum, effectively mitigating the stated problems.

TinyML provides a way to counter the mentioned problems. The proposed approach makes it possible to create cheap, widely available, energy-efficient sensor devices with long battery-operated lifespan [3]. To the best of our knowledge, there exists no research papers nor any device consisting of such a tiny microcontroller used in this paper that is capable of carrying out the task of analyzing whether the face mask is being worn properly.

The motivation of our work is associated with the COVID-19 pandemic. As it has been proven that wearing a face mask can help limit the spread of the virus [7–12], the face mask became mandatory in many public places, especially indoor ones, for example, in shops, cinemas and theaters. For the face masks to be effective, they have to be worn correctly; covering both the nose and the chin. Despite dissemination of these facts, there still exists a group of people that wear face masks incorrectly. Due to the fact that the security guards can only handle a limited area of the frequently visited objects, we propose the usage of TinyML intelligent sensors. With the aim of mounting them on the shelves in shops or near the cinema cash register queues, the battery operation capability is a must. This enforces the power efficiency of such devices in order to prolong the battery life to a reasonable extent. The other important factor is the device cost, as cheaper devices will result in higher accessibility.

The main hypothesis of our research presented in this paper is formulated as follows: it is possible to create a cheap TinyML system with a long uptime that can classify the masked face with satisfactory quality. The key challenge in our research is the fact that candidate hardware platforms are limited by the cost and power consumption, as they are also inherently resource-limited—a small, power-efficient microcontroler will not have

much memory nor computational power. To answer the main question, a hardware platform must be chosen and an effective classifier must be developed in order to conduct the research and check if the resulting TinyML system meets the quality requirements. The main contribution of this paper is the enablement of a severely resource-constrained microcontroller to perform complex computer vision tasks, as well as training, testing and implementing the model for the used algorithm that will fit in the Flash and RAM memory of the chosen microcontroller and will work within a reasonable time and with reasonable accuracy. The image preprocessing methods enhancing the system accuracy are also worth mentioning.

This paper is organized as follows. In Section 2, we discuss the related works, emphasizing the discovered TinyML niche in the computer vision field. In Section 3, we describe the used algorithms and datasets, and also propose the image preprocessing steps that gave the best observed results. We propose the method of data extraction that made the classification possible. As the TinyML is meant to run on a microcontroller, we evaluate the chosen platform candidates and pick the most suitable one. We discuss the limitations linked to the chosen platform and TinyML overall and describe the method the problem was solved with, as well as the applied improvements. We measure the time required in order to perform the complete face detection and classification and calculate the required energy and estimated battery life. In Section 4, we present and discuss the results, including the accuracy of the trained models and the overall system accuracy. The last section concludes this work.

2. Related Works

TinyML is an edge computing crossover between the IoT devices and the ML. This approach refines the low power aspect of the IoT devices and combines it with the algorithms capable of performing complex tasks, for example decision-making, prediction or classification. There are numerous established TinyML use cases [4], including: audio wake words, context recognition, control words, keyword detection, visual wake words, object detection, image classification, gesture recognition, object counting, text recognition, segmentation, forecasting, activity detection, sensing environmental factors (e.g., light or temperature), anomaly detection, motor control or predictive maintenance. Use cases similar to this paper are in the image classification group—we will discuss them in this section. One of the examples can be seen in the Ref. [13], where TinyML helps achieve better results in autonomous driving. Another example comes from the Ref. [14], where a person detector is built.

There are also other works that are linked with the topic of this paper. The problem of face detection is solved in the Ref. [15]. The researchers use a state-of-the-art object detection system called You Only Look Once (YOLO) v3 [16], capable of real-time object detection. In the mentioned paper, researchers used a seventh-generation Intel i7 Central Processing Unit (CPU) in combination with GTX 1080 Graphics Processing Unit (GPU) and 7.7 GB of RAM memory, achieving the results in less than 30 ms. The other work solving the problem of face detection is the Ref. [17]. The researchers proposed a method based on Faster Region-based Convolutional Neural Networks (R-CNN) [18] that is capable of providing the results in 130 ms using Intel Xeon E5 8-core processor with GTX TITAN-X GPU. The authors of the Ref. [19] solved the problem of face mask detection. The researchers applied YOLOv2 [20] combined with ResNet-50 [21] to create the detector. As the YOLOv2 uses a custom network based on Googlenet [22], it needs 8.52 billion floating-point operations for a forward pass [20]. Another way of solving the problem of face mask detection is presented in the Ref. [23]. This paper evaluates the approach based on ResNet [21] and MobileNet [24] respectively. The ResNet requires billions of floating-point operations for a forward pass [21], while MobileNet requires hundreds of millions of floating-point operations [24] for a forward pass.

There are very few recent works covering the topic of embedded systems used for face mask detection, for example, the Refs. [25–27]. The authors used powerful processors

in order to achieve impressive results. Our paper focuses on energy efficiency, taking the energy-efficient battery-operated device concept into consideration, which requires the usage of less powerful platforms. The chosen microcontroller is also cheaper than the ones proposed by the researchers, bringing the device cost down and making it more accessible in the process. Platform cost comparison is shown in Table 1. In the mentioned papers [25–27], the researchers focus on the binary classification problem: whether the person wears a face mask or not. Our paper focuses on the classification of the correctly masked face and incorrectly masked face. It should be underlined that, taking the existing ML and TinyML studies and devices into account, to the best of our knowledge, there are no TinyML research papers nor devices with resources comparable to the used STM32F411 microcontroller that were used to implement complex computer vision tasks. After discovering this field research gap, we propose a power-efficient TinyML system that can be used to verify whether the face mask is worn properly using strictly limited resources.

Table 1. Platform cost comparison.

Work	Platform Used	Price
[25]	STM32H743VI	$15.91
[26]	Sipeed Maixduino	$34.90
[27]	STM32H74VI	$15.91

Considering these approaches, despite splendid results, none of them can be transferred into the TinyML domain. This is due to the processing power and memory constraints, as well as immense energy consumption that is not acceptable on battery-operated devices.

3. Materials and Methods

This section presents the methods used in the proposed concept system, as well as the data used to train the classifier models. There are numerous ML classifiers, each with its own benefits and drawbacks. For the task of classification of the masked face, a K-means algorithm was chosen because it is computationally lightweight and requires a relatively small amount of Flash memory for the model in comparison with Neural Networks (NN), which are a popular method in image recognition tasks. The RAM requirement is also remarkably small, as the only calculated intermediate values are the distances between the classified object and the centroids compared to the activation values of every neuron in the network.

For the task of detecting a face, a sliding window technique combined with a K-means classifier was chosen, due to the fact that this approach is lightweight enough to fit into the memory of the microcontroller unlike most of the sophisticated models, executes reasonably fast, and can be further tuned in order to achieve either better results or higher speed.

3.1. K-Means

The K-means algorithm is a method capable of automatically clustering similar data examples together using a training set $\{x^{(1)}, ..., x^{(m)}\}$, where $x^{(i)} \in \mathcal{R}^n$ [2,28]. The algorithm is an iterative procedure that starts by randomly initializing the centroids, and then proceeds to loop consisting of two parts. The first part is the example-centroid assignment procedure, which means that every example gets assigned to the closest centroid using a certain distance metric. The second part is recomputing the centroids, which is done by calculating the mean value of all examples assigned to currently evaluated centroid and shifting the centroid to the computed mean value. This procedure is applied to every centroid, which ends the last step of the loop [2,28]. The algorithm can be represented by the pseudocode shown on the Listing 1.

Listing 1. K-means algorithm.

```
centroids = initialize_random();
for (int i = 0; i < iterations; i++)
{
example_assignments = assign_closest_centroid();
centroids = compute_means();
}
```

The K-means algorithm will always converge to a certain set of centroid values, but the solution may not be ideal, because it depends on the initial centroid values. Therefore, the K-means algorithm should be run multiple times with different random initializations. After running the algorithm several times, the best centroid set can be chosen from the results [2,29]. The typical K-means algorithm implementation uses Euclidean distance as the distance metric in the example-centroid assignment procedure.

The K-means algorithm was chosen for its overall lightness as it requires small amounts of both energy and memory to execute. The drawbacks of the algorithm were mitigated by careful design and testing of the preprocessing filters. Other methods that were considered have not been used due to the computational complexity and memory constraints. Notable methods other than the used K-means with $O(n^2)$ computational complexity are Support Vector Machines (SVM) with a computational complexity of $O(n^3)$, and Neural Networks with a computational complexity of $O(n^4)$. The algorithm we chose supported by carefully designed filters gave the best tradeoff between complexity (both computational complexity and memory complexity) and the given results. This is very important due to the strict TinyML limitations regarding the power consumption and available device resources. This approach gave us satisfactory results, and the other methods can be evaluated in future work. For more details, please refer to Sections 3.2.2 and 3.3.

3.2. Data

3.2.1. Datasets

An image is considered a two-dimensional array of values in range $[0, 255]$. The following datasets were used in this work:

- MaskedFace-Net [30,31]
- Natural images [32]

The MaskedFace-Net [30,31] dataset was used to train and test the K-means classifier that is capable of detection if the face mask is worn correctly or incorrectly. This dataset consists of 137,013 images (as of 5 November 2020) that is based on the Flickr-Faces-HQ (FFHQ) dataset, that was originally created as a benchmark for the generative adversarial networks (GAN) [33]. The dataset consists of two subsets:

- Correctly Masked Face Dataset (CMFD)-67192 images,
- Incorrectly Masked Face Dataset (IMFD)-69821 images.

The Incorrectly Masked Face Dataset contains three subclasses:

- Face masks worn only on chin, leaving the mouth and nose uncovered—6243 images;
- Face masks covering both chin and mouth, leaving the nose uncovered—57,224 images;
- Face masks covering both nose and mouth, leaving chin uncovered—6354 images.

The images are of high-quality, with a size of 1024 × 1024 px. They also incorporate a large enough variation regarding age, ethnicity, and background, and also introduce glasses and headgear.

The Natural images [32] dataset was used to train and test K-means classifiers supported by the sliding window technique that is capable of face detection. This dataset consists of 6899 images composed of eight subsets:

- airplane,
- car,
- cat,
- dog,
- flower,
- fruit,
- motorbike,
- person.

The images of people are of good quality, with a size of 256 × 256 px. The images incorporate similar variation as the MaskedFace-Net [30,31] dataset. In this paper, the other classes are treated as one non-person class.

3.2.2. Data Preprocessing

Data for the algorithm consist of preprocessed images from the datasets. The first step is filtering, which is shown on the Listing 2 for the MaskedFace-Net [30,31] dataset and on the Listing 3 for the Natural images [32] dataset. The examples are shown in the Figures 1 and 2. The downscaling size of 32 × 32 px was chosen as it was the smallest size that did not impact accuracy.

Listing 2. Filtering of the MaskedFace-Net [30,31] dataset.

- Downscaling the image containing a face to 32 × 32 px
- Calculating the intermediate color value using the procedure:

```
value = max(R, G, B)
```

- As most of the face masks are cyan, the intermediate color value is changed according to the procedure:

```
if (B > G && G > R && R < 200) value += 50
else value = 0
if (value > 255) value = 255
```

- Applying the threshold filter to the intermediate values with the boundary value of 240 with the procedure:

```
if (value < 240) value = 0
else value = 255
```

Listing 3. Filtering of the Natural images [32] dataset.

- Downscaling the image to 32 × 32 px
- Conversion to grayscale by averaging the R, G and B values

Figure 1. Example of image filtering of the MaskedFace-Net [30,31] dataset.

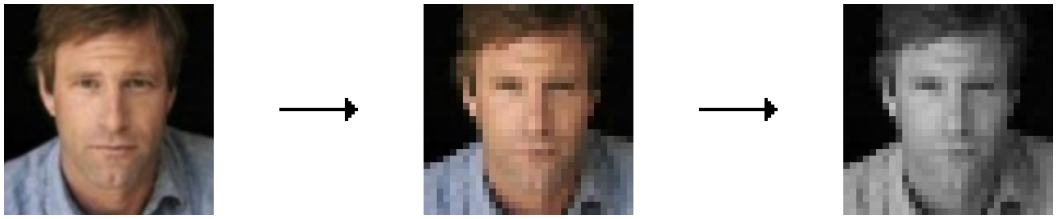


Figure 2. Example of image filtering of the Natural images [32].

Due to the fact that the data for the K-means algorithm is meant to be a normalized one-dimensional array, as ML algorithms typically work better on normalized data [2], the next step is feature normalization—every pixel of the image becomes a floating point value in range $[0, 1]$ stored in a two-dimensional array. The resulting array is then unfolded to a one-dimensional array using the procedure shown on the Listing 4. An example is shown in Figure 3.

Listing 4. Unfolding procedure.

- Two-dimensional array is split into n one-dimensional rows
- One-dimensional rows are concatenated into one one-dimensional array

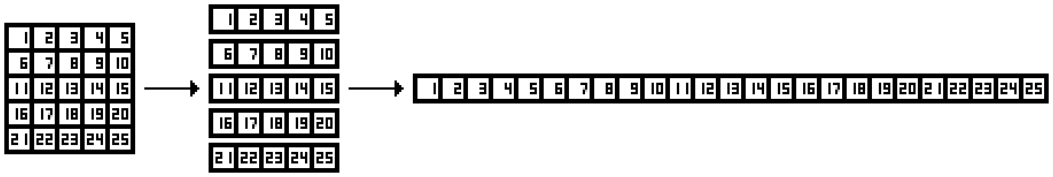


Figure 3. Example of the unfolding procedure.

3.3. Data Extraction

3.3.1. Sliding Window

The sliding window is a technique allowing for an easy detection of objects in computer vision ML tasks [34] that can run on platforms with limited resources. The window is a rectangle that moves horizontally and vertically through the image [2]. An example of the sliding window output is shown in Figure 4.

In order to find the desired object, each fragment of the image limited by the window is passed as an input to the classifier. To enable detection of objects of different sizes, scaling is mandatory [2].

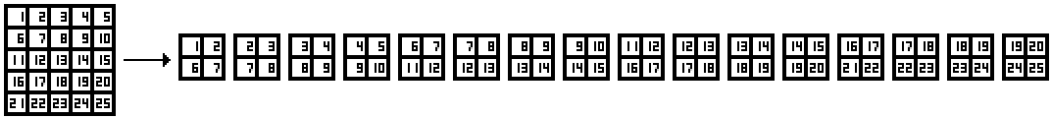


Figure 4. Example of the sliding window of size 2 × 2 with stride value of 1.

3.3.2. Face Detection

As the system was designed to work with a 640 × 480 px camera, there are many possible face sizes that can be present on the image. The detection is achieved by applying the sliding window technique supported by the K-means classifier with one of the window parameters shown in the Table 2. The best result, which is a rectangle with the highest probability of containing a face, is passed as the output of the detector.

Table 2. Face detection sliding window parameters.

Window Size	Stride	Window Size	Stride	Window Size	Stride
64 × 64	16	224 × 224	48	384 × 384	48
96 × 96	24	256 × 256	48	416 × 416	48
128 × 128	32	288 × 288	48	448 × 448	48
160 × 160	40	320 × 320	48	480 × 480	48
192 × 192	48	352 × 352	48		

3.4. TinyML Platform

3.4.1. Picking the Platform

An important factor impacting the whole system design is the choice of the TinyML platform. Among the major features required for consideration are price, accessibility, speed, power consumption and memory size. Three candidates were examined in order to choose the best-fitting TinyML platform:

- STM32F103CB, representing the STM32 F1xx family,
- STM32F411CE, representing the STM32 F4xx family,
- STM32H743VI, representing the STM32 H7xx family.

First two microcontrollers are widely available in the form of small evaluation boards, popular among tinkerers, called Bluepill for STM32F103 and Blackpill for STM32F411. The boards are shown in Figures 5 and 6. The third microcontroller is not as popular as the two previous ones, due to the lack of availability of small form-factor boards incorporating the STM32H743 chip, although it can be found in expensive, full-sized development boards.

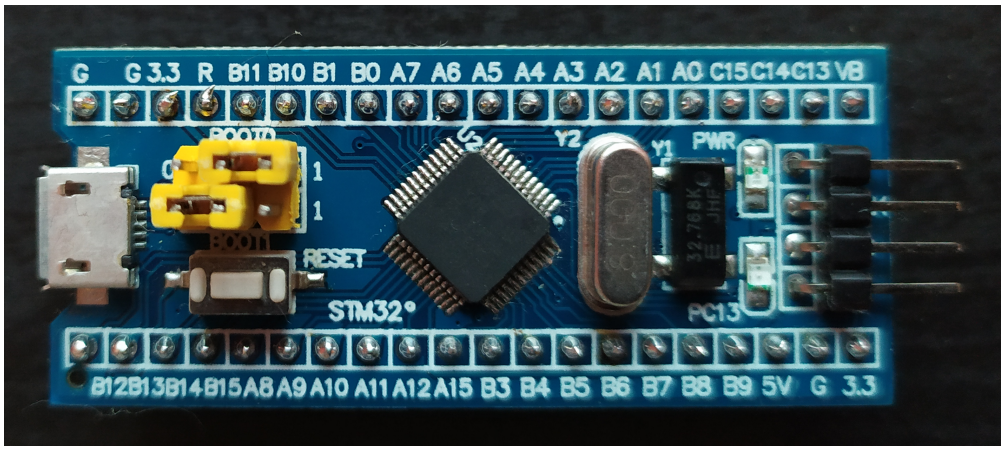


Figure 5. Bluepill board with STM32F103CB microcontroller.

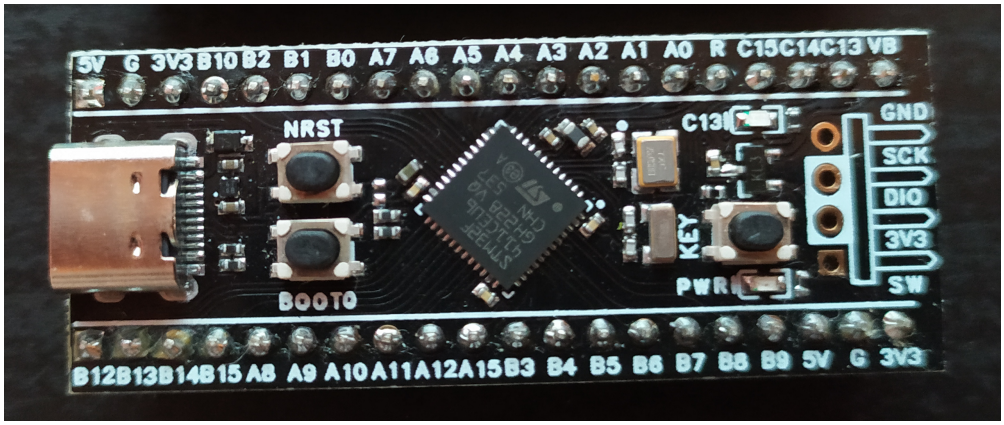


Figure 6. Blackpill board with STM32F411CE microcontroller.

As seen in the Table 3, the microcontrollers vastly differ from each other. The STM32F103 chip is the cheapest, but it also incorporates the worst power-consumption-to-speed ratio of 0.44 mA/MHz, compared to 0.1 mA/MHz for STM32F411 and 0.275 mA/MHz for STM32H743. It is also the only candidating microcontroller without FPU and with only 20 KB of RAM, what renders the chip unsuitable for this task.

Table 3. Platform comparison.

Microcontroller	Price	Speed	Power Consumption	Memory Size
STM32F103CB	\$7.17	72 MHz ARM Cortex M3 without FPU	32 mA @ 72 MHz	128 KB Flash 20 KB RAM
STM32F411CE	\$7.83	100 MHz ARM Cortex M4 with FPU	10 mA @ 100 MHz	512 KB Flash 128 KB RAM
STM32H743VI	\$15.91	480 MHz ARM Cortex M7 with FPU	132 mA @ 480 MHz	2 MB Flash 1 MB RAM

The STM32H743 chip is the most expensive, being over double the STM32F103 price. It contains more than enough memory and is also the fastest of the candidating microcontrollers, containing the ARM Cortex M7 core capable of reaching a speed of 480 MHz. However, in terms of TinyML, it is not suitable due to a worse power-consumption-to-speed ratio than STM32F411, and not being as widely available as the STM32F103 and STM32F411 chips.

3.4.2. Chosen Platform

Considering all the factors, the STM32F411 chip was chosen as the best suitable microcontroller for this task. It is relatively cheap, widely available as a Blackpill evaluation board, and fast enough for computational tasks due to the fast ARM Cortex M4 core, while being the most energy-efficient option out of all candidates. It also contains sufficient onboard memory to have a big enough DMA-driven buffer connected to the external RAM, resulting in no impact on the computation time due to the fact that all needed data can be present in the internal RAM on demand.

3.4.3. Limitations

TinyML is a great approach for creating energy-efficient battery-operated intelligent sensors. Despite many advantages, it also comes with significant drawbacks. There are many ML state-of-the-art algorithms and models, but they require an immense amount of resources which are not present on any typical microcontroller. One of the limiting factors

is the amount of memory for both Flash and RAM [4]. The STM32F411 chip has 512 KB of Flash memory, in which the code and trained models must fit. This makes the usage of the high-precision complex models impossible. Even if this was possible, usage of such models would be impractical, as intricate models would need a tremendous amount of time to calculate the result.

As the STM32F411 microcontroller has 128 KB of RAM, the presence of an external memory chip is required, due to the fact that the image of size 640 × 480 px encoded in the RGB565 format requires 600 KB of storage.

3.5. Masked Face Classification

3.5.1. Solving the Masked Face Problem

The first step in solving the problem of classification of the masked face is the detection of the face in an image. This is achieved by applying the sliding window technique to the image, obtaining the image fragment in the process. The preprocessing filter is then applied and the image fragment is unfolded. The unfolded array is passed to the K-means face/non-face classifier, and the procedure is repeated until the sliding window positions are exhausted. At this point, the image of a face is obtained. Then, the preprocessing filter is applied and the image of a face is unfolded. The unfolded array is passed to the K-means masked face classifier, which produces the system output. The system flow diagram is shown in Figure 7.

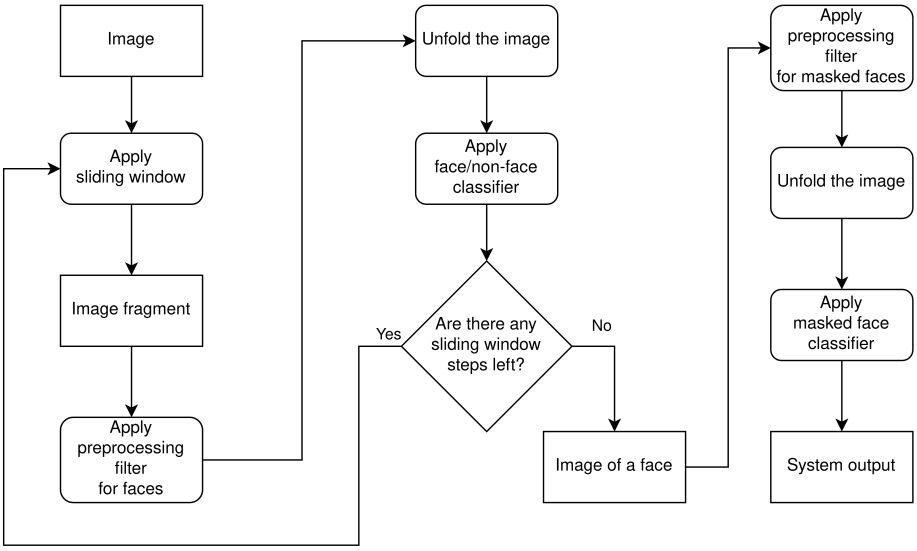


Figure 7. System flow diagram.

3.5.2. Improving the Algorithm

The goal is to create a TinyML device that is capable of detecting whether the face mask is being worn correctly or incorrectly. Due to the TinyML nature and the fact that this device is meant to be powered by a battery, saving every possible bit of energy is a must. This can be done by algorithm simplification that can either influence the results or not, depending on the action. An example of algorithm simplification that does not influence the results is distance metric replacement. The conventionally used Euclidean metric gives precise distance values, but in order to classify a vector using K-means algorithm, precise distance value is not required. The only information that is required in order to correctly classify the given example to one particular centroid is the answer to the question: "Which centroid is the closest to the given example?"

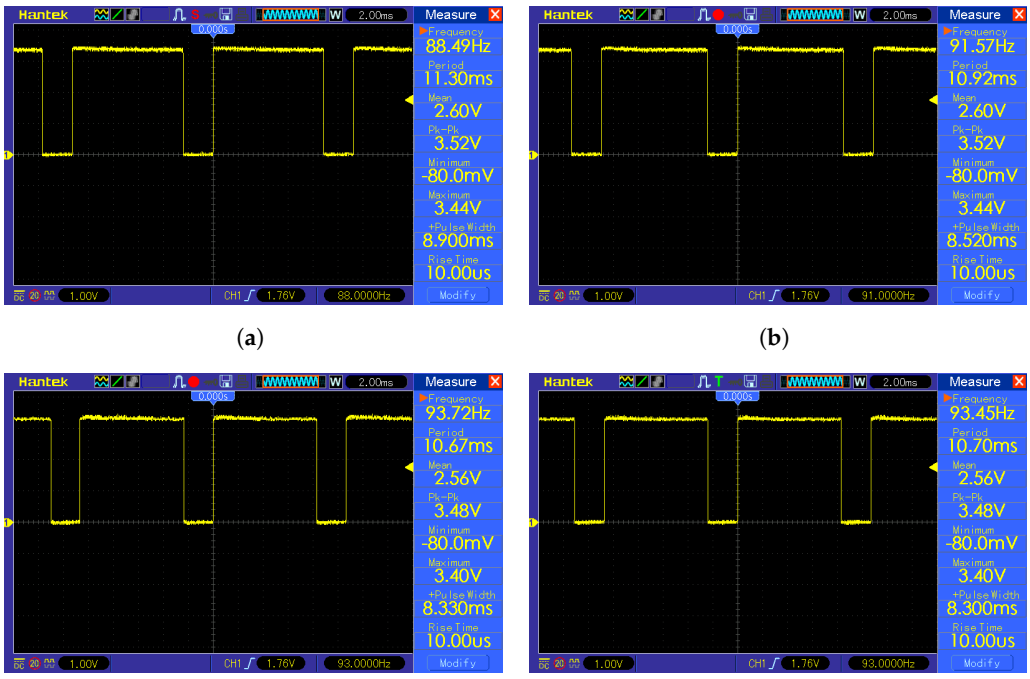
This can be done by replacing the Euclidean metric, described with the Formula (1) with the squared Euclidean metric, described with the Formula (2).

$$ED(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} \quad (1)$$

$$SED(p,q) = (p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2 \quad (2)$$

It can also be proven mathematically. The square root function monotonically increases in the entire domain, that is, $x \in (0, \infty)$. As the distance is never negative, the square root can be removed from the Euclidean metric, creating the squared Euclidean metric, as this metric still allows for explicit comparison of the distance values. This means that replacing the Euclidean metric with a squared Euclidean metric will distort distance values (due to nonlinearity), but will not change the "greater than" and "smaller than" relationships between them, which is the only information needed.

Square root removal is beneficial in the context of TinyML because it is a computationally heavy mathematical operation, which results in an excessive amount of required time and energy. As shown in the Figure 8, the time difference is non-negligible. The K-means classification with a squared Euclidean metric is a baseline with 8.30 ms calculation time. The K-means classification with a Euclidean metric and bisection algorithm for square root calculation resulted in 8.90 ms of total computation time, which translates into 0.60 ms calculation time for the bisection square root calculation. The K-means classification with a Euclidean metric and Babylonian algorithm resulted in 8.52 ms total computation time, which translates into 0.22 ms calculation time for the Babylonian square root calculation. This translates into a 7.23% overall time increase for K-means classification with a Euclidean metric with a bisection algorithm for square root calculation and 2.65% for K-means classification with a Euclidean metric with a Babylonian algorithm for square root calculation in comparison with K-means classification with a squared Euclidean metric. The K-means classification with a Euclidean metric and optimized $sqrt()$ function for square root calculation resulted in 8.33 ms total computation time, which translates into 0.03 ms calculation time for the optimized $sqrt()$ function root calculation and a 0.36% overall time increase for K-means classification with a Euclidean metric with an optimized $sqrt()$ function for square root calculation in comparison with K-means classification with the squared Euclidean metric.



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

(**c**) (**d**)

Figure 8. Time difference between K-means classification time with Euclidean distance metric and squared Euclidean distance metric. (**a**) K-means with Euclidean distances computation time using bisection algorithm; (**b**) K-means with Euclidean distances computation time using Babylonian algorithm; (**c**) K-means with Euclidean distances computation time using optimized sqrt() function; (**d**) K-means with squared Euclidean distances computation.

3.6. Measurements

3.6.1. Face Detection

The sliding window was used in combination with the K-means classifier to detect a face in the image. The image fragment limited by the window was downscaled to 32 × 32 px in order to pass it to the classifier after applying the grayscale filter. The most computationally heavy task in this method is the downscaling, which must be measured and taken into consideration. Another procedure to measure is the grayscale filter. The sliding window technique measurement results are shown in Table 4. The Figures A1 and A2 from the Appendix A contain the oscillograms from the measurements. Due to the fact that during measurements, the microcontroller was programmed to output a logical one during the desired operation to measure and logical zero otherwise, the time measurement of the operation is equivalent to measuring the pulse width. The grayscale conversion measurement shown in the Figure 9 resulted in 1.45 ms of calculation time. The K-means classification measurement shown in the Figure 10 resulted in a 8.30 ms calculation time.

Table 4. Sliding window measurements.

Window Size	Stride	Window Positions	Stride: Window Size Ratio	Downscaling Time
64 × 64	16	999	25%	6.920 ms
96 × 96	24	391	25%	12.95 ms
128 × 128	32	204	25%	21.30 ms
160 × 160	40	117	25%	32.00 ms
192 × 192	48	70	25%	44.96 ms
224 × 224	48	54	21.43%	60.24 ms
256 × 256	48	45	18.75%	77.90 ms
288 × 288	48	40	16.67%	97.80 ms
320 × 320	48	28	15%	120.0 ms
352 × 352	48	21	13.64%	144.6 ms
384 × 384	48	18	12.5%	171.6 ms
416 × 416	48	10	11.54%	200.8 ms
448 × 448	48	5	10.71%	232.4 ms
480 × 480	48	4	10%	266.2 ms

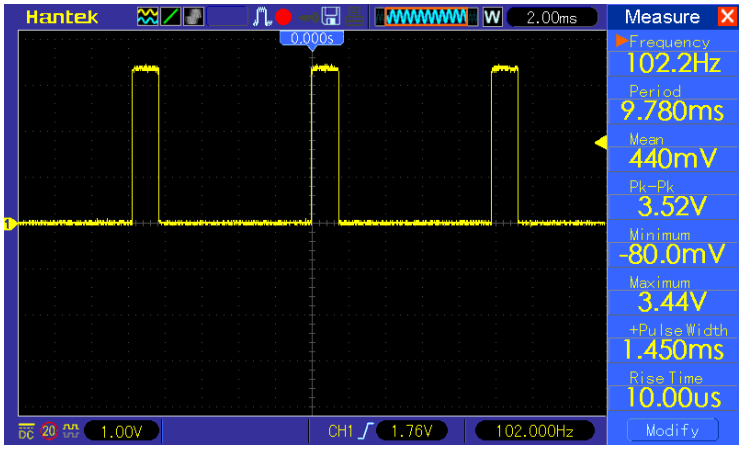


Figure 9. Measurement of time required for grayscale conversion.

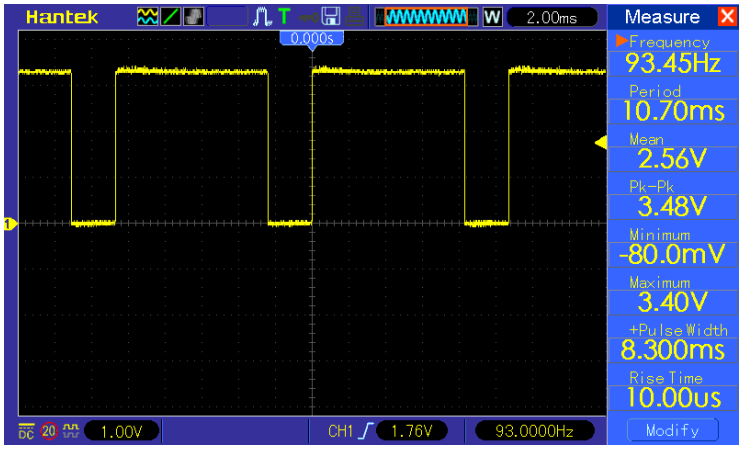


Figure 10. Measurement of time required for K-means classifier.

3.6.2. Masked face Classification

The masked face classification consists of a thresholding part and K-means algorithm part. The measurement of thresholding shown in the Figure 11 resulted in 2.40 ms of calculation time, and the K-means measurement is shown in the Figure 10.

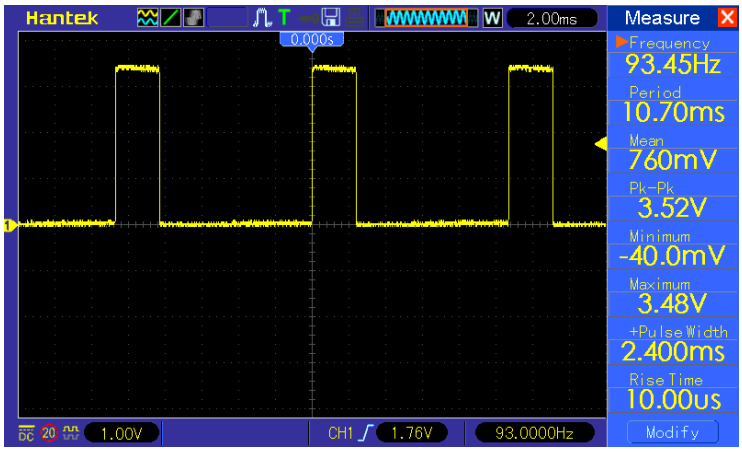


Figure 11. Measurement of time required for thresholding.

4. Results

This section presents a performance evaluation of each system component, including the overall system accuracy and the face detection time calculation results. The tuning of the masked face classifier and sliding window is discussed and presented. Using the acquired data, the energy consumption of one full masked face classification operation is calculated, and the device battery life is estimated.

All datasets used for training the models were split into three subsets:

- Training subset consisting of 60% randomly chosen images
- Cross-validation subset consisting of 20% randomly chosen images
- Testing subset consisting of 20% randomly chosen images

The models were trained using the training subsets and validated using cross-validation subsets. The best-performing model in the validation stage was chosen for testing using the testing subset. As the K-means classifier training is heavily dependent on the initial values, the training was executed 1000 times [2].

4.1. Masked Face Classifier Tuning

The K-means masked face classifier was trained and tested with different parameters and filtering methods in order to create a model with the best results possible. Tested

approaches are described in the Table 5. There were 14 tuning approaches with three varying parameters: the number of centroids, filtering method and whether the subset of the masked face set was used. The amount of centroids that was tested was in the range 2–4 due to the existence of two major dataset classes (correctly masked face and incorrectly masked face) and four total dataset subclasses (correctly masked face, incorrectly masked face-mouth and nose uncovered, incorrectly masked face-nose uncovered, incorrectly masked face-chin uncovered) in order to check which value gave the best results. Four types of filtering were tested. The first filtering method was composed of a downscaling module that outputs a. 32×32 px image, the calculation of an intermediate color value with $max(R, G, B)$, a color-enhancing module sensitive only to the face mask color range, and a thresholding module with a threshold of 240. The second filtering method is composed of downscaling module that outputs 32×32 px image, calculation of intermediate color value with $max(R, G, B)$, a color-enhancing module sensitive to the face mask color range and less sensitive to other color ranges, and a thresholding module with threshold levels of 160, 192, and 240. The third filtering method was composed of a downscaling module that outputs a 32×32 px image, calculation of an intermediate color value with $max(R, G, B)$ and thresholding module with a threshold levels of 160, 176, 192, 208, 224 and 240. The fourth filtering method was composed of a downscaling module that outputs a 32×32 px image. The first two described filters were designed to take advantage of the fact that most of the face masks were cyan. The third filter did not have a color-enhancing module sensitive to the face mask color range in order to discover the importance of this module in results. The fourth filter only downscaled the image and was the only filter that resulted in a non-grayscale image. It was also introduced in order to discover the performance of other filtering operations (calculating intermediate value and thresholding).

The comparison of all proposed tuning approaches is shown in Figures 12–16 that present the obtained results of the following metrics: accuracy, TPR, TNR, precision and recall, respectively. As seen in the mentioned Figures, the metrics vary significantly between the tested approaches. The obtained results for the chosen approach (number 4) are of very high-quality.

The comparison of filtering methods is shown in the Figure 17. The approach numbers 1, 2, 3, 6 and 7 use a subset consisting only of the images where masks are worn on the chin and a similar amount of randomly chosen images where masks are worn correctly. This results in a subset of size 12,226 instead of the complete dataset of 137,013 images. The subset was used to discover the difference in results between "easy data" consisting of only two dataset subclasses (correctly masked face and incorrectly masked face-mouth and nose uncovered; the subclasses are balanced in size) and the full dataset.

Table 5. Tested approaches for the masked face classifier.

Approach	Filtering Method	Centroid Amount	Masked Face Subset
1	Proposed in this paper	2	Yes
2	Proposed in this paper	3	Yes
3	Proposed in this paper	4	Yes
4	Proposed in this paper	2	No
5	Proposed in this paper	3	No
6	Shown on the Listing 5	2	Yes
7	Shown on the Listing 5	3	Yes
8	Shown on the Listing 5	2	No
9	Shown on the Listing 5	3	No
10	Shown on the Listing 6	2	No
11	Shown on the Listing 6	3	No
12	Shown on the Listing 6	4	No
13	Downscaling to 32×32 px	2	No
14	Downscaling to 32×32 px	3	No

Listing 5. Alternative filtering of the MaskedFace-Net [30,31] dataset-method 1.

- Downscaling the image containing a face to 32 × 32 px
- Calculating the intermediate color value using the procedure:

```
value = max(R, G, B)
```

- As most of the face masks were cyan, the intermediate color value was changed according to the procedure:

```
if (B > G && G > R) value += 50
else if (value > 200) value -= 50
if (value > 255) value = 255
```

- Applying the threshold filter to the intermediate values with the boundary values of 160, 192 and 240 with the procedure:

```
if (value < 160) value = 0
else if (value < 192) value = 85
else if (value < 240) value = 170
else value = 255
```

Listing 6. Alternative filtering of the MaskedFace-Net [30,31] dataset-method 2.

- Downscaling the image containing a face to 32 × 32 px
- Calculating the intermediate color value using the procedure:

```
value = max(R, G, B)
```

- Applying the threshold filter to the intermediate values with the boundary values of 160, 176, 192, 208, 224 and 240 with the procedure:

```
if (value < 160) value = 0
else if (value < 176) value = 42
else if (value < 192) value = 85
else if (value < 208) value = 127
else if (value < 224) value = 170
else if (value < 240) value = 212
else value = 255
```

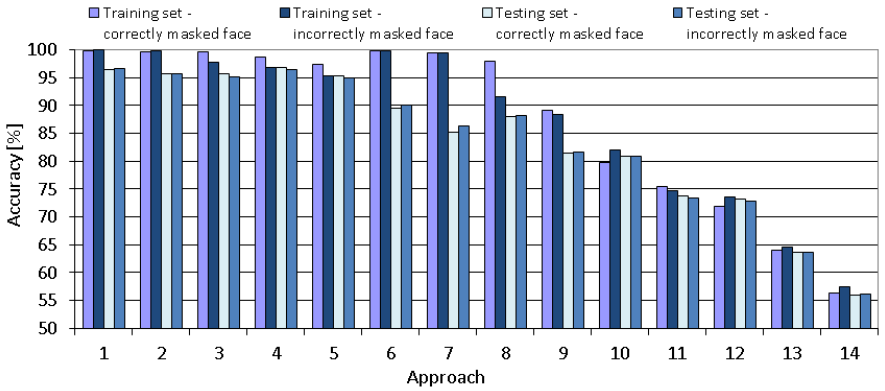


Figure 12. Comparison of the accuracy of the K-means classifier vs. parameters and filtering methods.

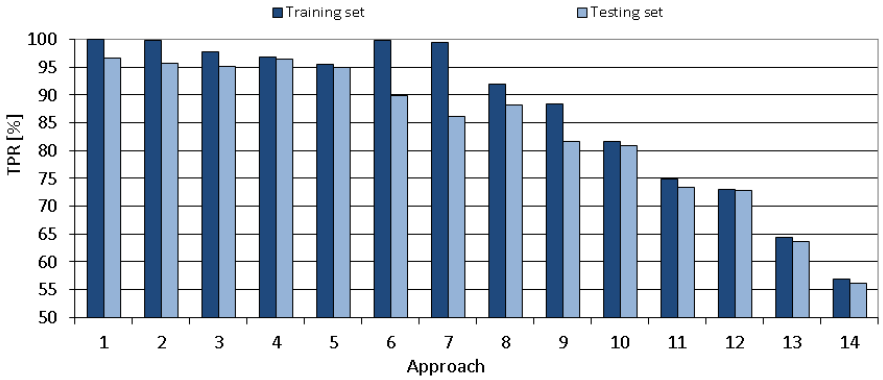


Figure 13. Comparison of the TPR of the K-means classifier vs. parameters and filtering methods.

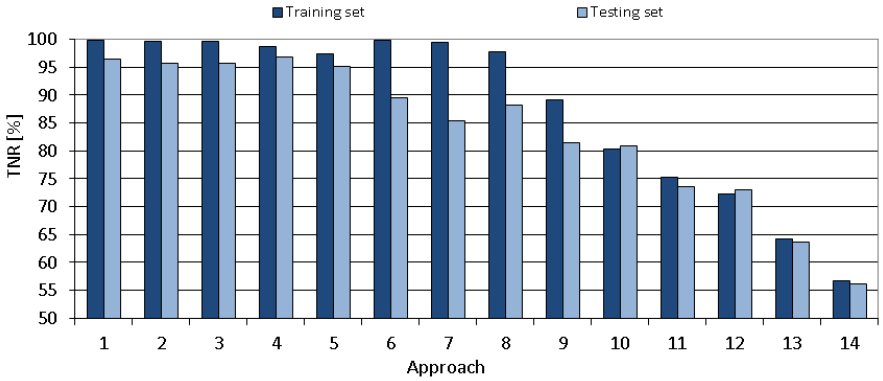


Figure 14. Comparison of the TNR of the K-means classifier vs. parameters and filtering methods.

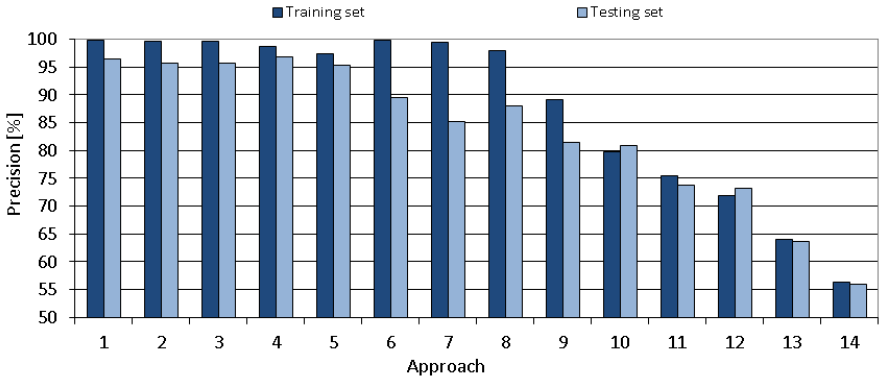


Figure 15. Comparison of the precision of the K-means classifier vs. parameters and filtering methods.

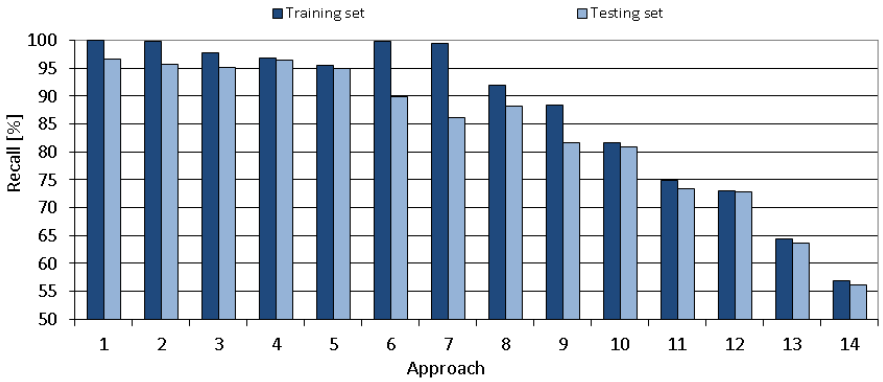


Figure 16. Comparison of the recall of the K-means classifier vs. parameters and filtering methods.

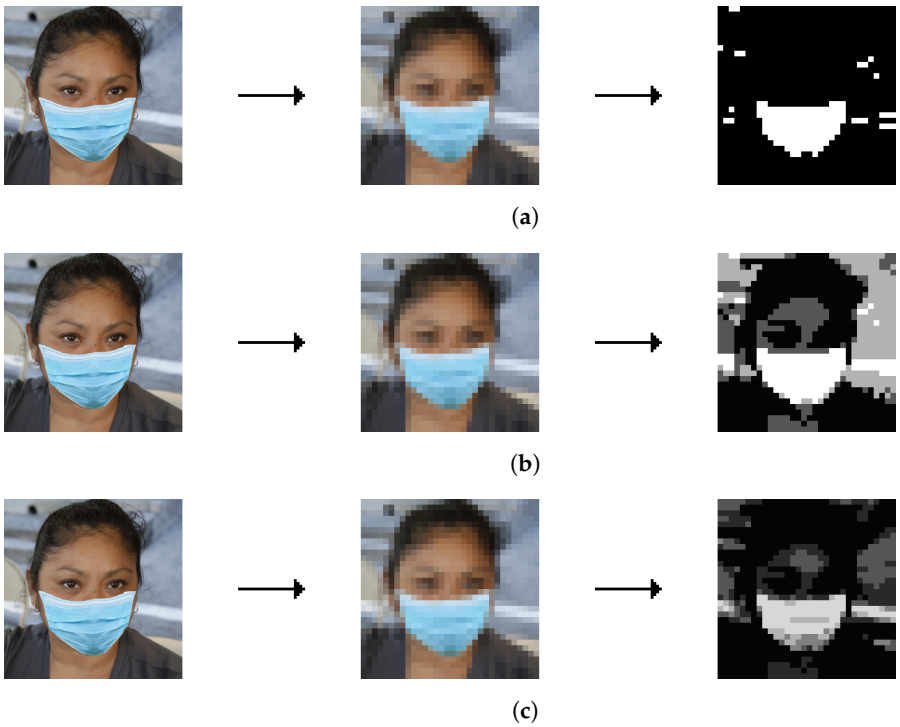


Figure 17. Comparison of the filtering methods. (**a**) Proposed filtering; (**b**) Filtering from Listing 5; (**c**) Filtering from Listing 6.

The results of all analyzed 14 approaches were validated using the Analysis of Variance (ANOVA) test. For each analyzed approach, we applied the bootstrapping method to obtain 30 results of the accuracy metric. The ANOVA test of the obtained results is shown in Table 6. According to the reported values, the null hypothesis is rejected, that is, there is sufficient evidence to conclude that not all of the means of analyzed 14 approaches are equal and that there are statistical differences between the tested approaches. Based on the obtained results, the model from Approach 4 was chosen as the best masked face classifier due to the fact that this model performed the best in the test results, accurately classifying the correctly masked face, while being able to achieve almost the best results for accurately classifying the incorrectly masked face, even if only the nose was uncovered. To verify the performance of Approach 4, we made the *t*-test comparing Approach 4 against other approaches. In all cases, the obtained *p*-value is <0.00001, which means that the differences between the results of Approach 4 and results of other approaches are significant. Therefore, in the remainder of the paper, we present results are obtained using Approach 4.

Table 6. The results of the ANOVA test of analyzed approaches for the masked face classifier

Source	Degrees of Freedom	Sum of Squares	Mean Square	F-Stat	*p*-Value
Between groups	13	68,823.4871	5294.1144	143,244.7783	0
Within groups	406	15.0052	0.037		
Total	419	68,838.4922			

4.2. Sliding Window Tuning

The sliding window technique can be tuned in two ways—either by maximizing speed, or maximizing accuracy. By using more window sizes, the amount of passes through the image rises, especially with smaller window sizes. With this approach, objects of more sizes can be detected, increasing accuracy. By using less window sizes, the amount of passes through the image falls, trading accuracy for speed. Similar observations can be noticed by modifying the stride value, lowering the stride translates into more window

positions, increasing accuracy, and lowering speed, while increasing the stride translates into less window positions, decreasing accuracy and increasing speed. The sliding window parameters were chosen to achieve enough detection coverage while keeping the time relatively low.

4.3. System Accuracy

4.3.1. Masked Face Classification

The masked face classification accuracy results are shown in Table 7. The results presented here are results of Approach 4 chosen in the tuning section (refer to Section 4.1). There are values present for correct detection of the correctly masked face, correct detection of the incorrectly masked face, the True Positive Rate (TPR), True Negative Rate (TNR), precision and recall. Considering the fact that all of these values are over 96% and that the dataset classes are balanced in terms of sample amount, this translates into a high-quality classifier.

Table 7. Masked face classification metrics.

Metric	Value
Accuracy of the correct detection of the correctly masked face	96.86%
Accuracy of the correct detection of the incorrectly masked face	96.40%
TPR	96.58%
TNR	96.41%
Precision	96.40%
Recall	96.58%

4.3.2. Face Detection

The face detection metric values are shown in the Table 8. As with the masked face classification, there are values present for correct detection of the face, correct detection of the non-face, TPR, TNR, precision and recall. As all of these values are over 79%, this also translates into a high-quality classifier, although not as high as the masked face classifier.

Table 8. Face detection metrics.

Metric	Value
Accuracy of the correct detection of the face	82.09%
Accuracy of the correct detection of the non-face	79.85%
TPR	80.29%
TNR	81.68%
Precision	82.09%
Recall	80.29%

Using the acquired data, the time required to detect a face was calculated by multiplying the amount of the window positions with appropriate downscaling time and adding the amount of the window positions multiplied by the K-means processing time, consisting of the grayscale conversion and K-means classification. The results for the proposed window sizes are shown in the Table 9.

Depending on the device mounting position, one of the proposed window sizes can be used. Smaller window size allows the detection of the face in the scenario where the examined person is standing farther away from the camera, but at the cost of higher computation time. A bgger window size can be used in situations where the examined person is standing closer to the camera, resulting in a shorter computation time.

Table 9. Face detection time calculation results.

Window Size	Total Downscaling Time	Total K-Means Time	Total Detection Time
320 × 320	3360.00 ms	273.0 ms	**3633.00 ms**
352 × 352	3036.60 ms	204.8 ms	**3241.35 ms**
384 × 384	3088.80 ms	175.5 ms	**3264.30 ms**
416 × 416	2008.00 ms	97.5 ms	**2105.50 ms**
448 × 448	1162.00 ms	48.8 ms	**1210.75 ms**
480 × 480	1064.80 ms	39.0 ms	**1103.80 ms**

4.3.3. Total System Accuracy

Taking the results from the Tables 7 and 8, and also considering the fact that the accuracy of the system composed of two systems connected in series is the product of multiplication of the subsystem accuracies, the total system accuracy is shown in the Table 10. Considering the fact that both accuracy values are over 79%, this system has good-quality predictions.

Table 10. Overall system accuracy.

Scenario	Accuracy
Correct detection of the correctly masked face	79.51%
Correct detection of the incorrectly masked face	79.13%

4.4. Energy Consumption

The amount of consumed energy can be calculated using the Formula (3).

$$E = P * t, \tag{3}$$

where E-energy, P-power and t-time. The power can be calculated using the Formula (4).

$$P = U * I, \tag{4}$$

where U-voltage and I-current. The Formulas (3) and (4) can be combined together, forming the Formula (5).

$$E = U * I * t. \tag{5}$$

As the time required to detect a face in image depends on the chosen window size, the results are shown in the Table 11. The STM32F411 microcontroller is powered by 3.3 V DC voltage and consumes around 10 mA of current. The time required to classify a masked face is equal to 9.75 ms.

Table 11. System time and energy consumption measurements.

Window Size	Face Detection Time	Total Time	Energy Consumption
320 × 320	3633.00 ms	3642.75 ms	0.120 J
352 × 352	3241.35 ms	3251.10 ms	0.107 J
384 × 384	3264.30 ms	3274.05 ms	0.108 J
416 × 416	2105.50 ms	2115.25 ms	0.070 J
448 × 448	1210.75 ms	1220.50 ms	0.040 J
480 × 480	1103.80 ms	1113.55 ms	0.037 J

4.5. TinyML Battery Life

Considering the calculated amount of energy required for one face mask operation, the battery life can be estimated. The 18650 battery with a typical capacity of 2500 mAh and nominal voltage of 3.7 V will be used for calculations. As there are high-efficiency

power supply chips available with several μA of quiescent current and over 90% efficiency, for example, for TPS63806, the energy loss can be pessimistically modeled as 10%. The singular face mask operation energy consumption results are shown in Table 12.

Table 12. System time and energy consumption results.

Window Size	Energy Consumption (90% Efficiency)	Possible Operations on One Battery Charge
320 × 320	0.011 mAh	222,359
352 × 352	0.010 mAh	249,146
384 × 384	0.010 mAh	247,400
416 × 416	0.007 mAh	382,933
448 × 448	0.004 mAh	663,662
480 × 480	0.003 mAh	727,403

Considering the power consumption of the microcontroller, the system could continuously operate for 225 h. With the example of shops visited by 5000 customers every day, the proposed system uptime without recharging the battery is shown in Table 13. This is especially important, as our low-power design makes the mounting of the proposed system possible wherever it is needed, regardless of the presence of mains power. In the example with a shop, it would be inconvenient to install the mains power cables on market shelves, where the proposed system could be mounted.

Table 13. System uptime results.

Window Size	System Uptime
320 × 320	44.47 days
352 × 352	49.83 days
384 × 384	49.48 days
416 × 416	76.59 days
448 × 448	132.73 days
480 × 480	145.48 days

5. Conclusions

According to the measurements and results shown in this paper, we have shown that it is possible to create an energy-efficient battery-operated system capable of executing complex computer vision tasks in a reasonable time with a low amount of available resources by using carefully selected methods and algorithms. This approach can help to reduce the energy usage of the intelligent sensor grids, as well as to decrease the network traffic by replacing cloud computing where it is worthwhile.

It should be underlined that taking the existing ML and TinyML studies and devices into account, to the best of our knowledge, there are no TinyML research papers nor devices with the resources comparable to the STM32F411 microcontroller that was used to implement complex computer vision tasks. Meanwhile, there exist research papers handling the topic of such tasks, yet with the use of overly powerful processors with more than enough resource memory and computational capabilities, resulting in higher cost and excessive power consumption of such devices in comparison to the proposed solution.

The main conclusion of this paper is that it is possible to create a cost-effective and battery-operated TinyML system with a long uptime and that provides satisfactory results of masked face classification. In more detail, we created the TinyML system consisting of K-means supported by a sliding window face detection module, a carefully designated preprocessing filter that is responsible for feature extraction, and a K-means masked face classifier. The preprocessing filter transforms the detected face into a downscaled 32 × 32 px image that is converted to intermediate values and passed to the thresholding stage. After unfolding, this results in a feature vector with 1024 dimensions. Our proposed

TinyML system solution managed to deliver very high-quality metric values, with accuracy, TPR, TNR, precision and recall being over 96% for masked face classification, while being able to reach up to 145 days of uptime using a typical 18650 battery with a capacity of 2500 mAh and nominal voltage of 3.7 V. The results were achieved using a STM32F411 microcontroller with 100 MHz ARM Cortex M4, which proves that execution of complex computer vision tasks is possible on such low-power devices. It should be noted that the STM32F411 microcontroller draws only 33 mW during the operation. The accuracy of the whole system is over 79%.

There is still room for improvement—future work may include face detection accuracy enhancement and face detection speedup. Further work can also cover other classification algorithms and methods in combination with different preprocessing methods, as well as the research of other hardware platforms that are capable of fulfilling the strict limitations of TinyML.

Author Contributions: Conceptualization, D.P. and K.W.; methodology, D.P.; software, D.P.; validation, D.P. and K.W.; formal analysis, D.P. and K.W.; resources, D.P.; data curation, D.P.; writing—original draft preparation, D.P.; writing—review and editing, D.P. and K.W.; visualization, D.P.; supervision, K.W.; project administration, K.W.; funding acquisition, K.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Ministry of Education and Science, grant number POWR.03.01.00-00-P015/18.

Data Availability Statement: The data that support the findings of this study are available within this article.

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

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
ANOVA	Analysis of Variance
CPU	Central Processing Unit
FPU	Float Processing Unit
GAN	Generative Adversarial Networks
GPU	Graphics Processing Unit
IoT	Internet of Things
ML	Machine Learning
NN	Neural Network
R-CNN	Region-based Convolutional Neural Networks
TinyML	Tiny Machine Learning
TNR	True Negative Rate
TPR	True Positive Rate
YOLO	You Only Look Once

Appendix A. Downscaling Time Measurements

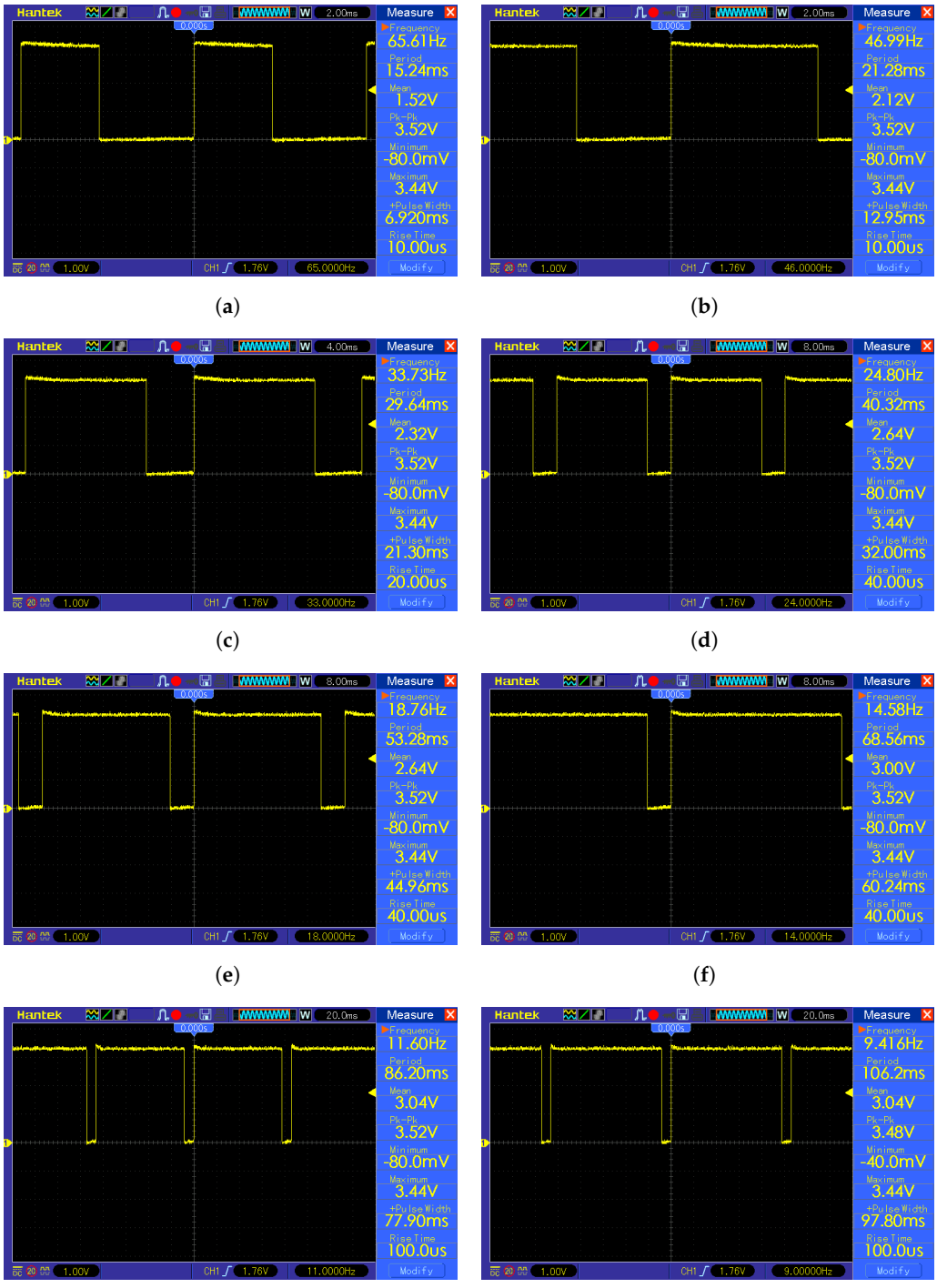


Figure A1. Measurements of time required to downscale the image of given size to 32 × 32 px (1). (**a**) 64 × 64 px; (**b**) 96 × 96 px; (**c**) 128 × 128 px; (**d**) 160 × 160 px; (**e**) 192 × 192 px; (**f**) 224 × 224 px; (**g**) 256 × 256 px; (**h**) 288 × 288 px.

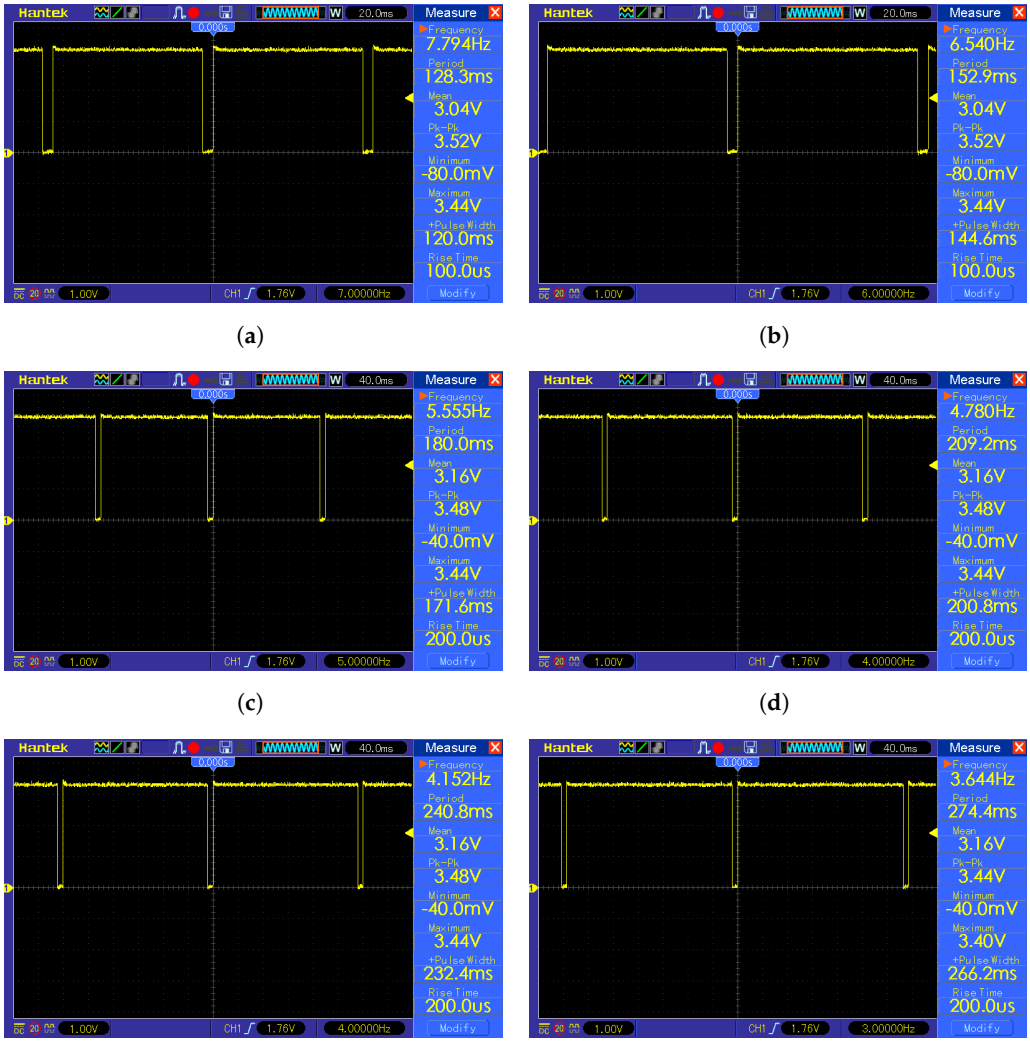


Figure A2. Measurements of time required to downscale the image of given size to 32 × 32 px (2). (**a**) 320 × 320 px; (**b**) 352 × 352 px; (**c**) 384 × 384 px; (**d**) 416 × 416 px; (**e**) 448 × 448 px; (**f**) 480 × 480 px.

References

1. Patterson, D.; Gonzalez, J.; Le, Q.; Liang, C.; Munguia, L.M.; Rothchild, D.; So, D.; Texier, M.; Dean, J. Carbon emissions and large neural network training. *arXiv* **2021**, arXiv:2104.10350.
2. Ng, A. Machine Learning Course. Coursera [Online]. 2017. Available online: https://www.coursera.org/learn/machine-learning (accessed on 29 August 2021).
3. Sanchez-Iborra, R.; Skarmeta, A.F. Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits Syst. Mag.* **2020**, *20*, 4–18. [CrossRef]
4. Banbury, C.R.; Reddi, V.J.; Lam, M.; Fu, W.; Fazel, A.; Holleman, J.; Huang, X.; Hurtado, R.; Kanter, D.; Lokhmotov, A.; et al. Benchmarking TinyML systems: Challenges and direction. *arXiv* **2020**, arXiv:2003.04821.
5. Gowda, M.; Gowda, J.; Iyer, S.; Pawar, M.; Gaikwad, V. Power Consumption Optimization in IoT based Wireless Sensor Node Using ESP8266. In *ITM Web of Conferences*; EDP Sciences: Ulis, France, 2020; Volume 32.
6. Bertuletti, S.; Cereatti, A.; Comotti, D.; Caldara, M.; Della Croce, U. Static and dynamic accuracy of an innovative miniaturized wearable platform for short range distance measurements for human movement applications. *Sensors* **2017**, *17*, 1492. [CrossRef]
7. National Academies of Sciences, Engineering and Medicine. Rapid expert consultation on the effectiveness of fabric masks for the COVID-19 Pandemic (8 April 2020). In *Rapid Expert Consultations on the COVID-19 Pandemic: 14 March–8 April 2020*; National Academies Press (US): Washington, DC, USA, 2020.
8. Ueki, H.; Furusawa, Y.; Iwatsuki-Horimoto, K.; Imai, M.; Kabata, H.; Nishimura, H.; Kawaoka, Y. Effectiveness of face masks in preventing airborne transmission of SARS-CoV-2. *MSphere* **2020**, *5*, e00637-20. [CrossRef] [PubMed]
9. Li, Y.; Liang, M.; Gao, L.; Ahmed, M.A.; Uy, J.P.; Cheng, C.; Zhou, Q.; Sun, C. Face masks to prevent transmission of COVID-19: A systematic review and meta-analysis. *Am. J. Infect. Control* **2020**, *49*, 900–906. [CrossRef]

10. Verma, S.; Dhanak, M.; Frankenfield, J. Visualizing the effectiveness of face masks in obstructing respiratory jets. *Phys. Fluids* **2020**, *32*, 061708. [CrossRef]
11. Swain, I.D. Why the mask? The effectiveness of face masks in preventing the spread of respiratory infections such as COVID-19–a home testing protocol. *J. Med. Eng. Technol.* **2020**, *44*, 334–337. [CrossRef] [PubMed]
12. Eikenberry, S.E.; Mancuso, M.; Iboi, E.; Phan, T.; Eikenberry, K.; Kuang, Y.; Kostelich, E.; Gumel, A.B. To mask or not to mask: Modeling the potential for face mask use by the general public to curtail the COVID-19 pandemic. *Infect. Dis. Model.* **2020**, *5*, 293–308. [CrossRef]
13. de Prado, M.; Rusci, M.; Capotondi, A.; Donze, R.; Benini, L.; Pazos, N. Robustifying the Deployment of tinyML Models for Autonomous mini-vehicles. *Sensors* **2021**, *21*, 1339. [CrossRef]
14. Warden, P.; Situnayake, D. *Tinyml: Machine Learning with Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers*; O'Reilly Media: Newton, MA, USA, 2019.
15. Yang, W.; Jiachun, Z. Real-time face detection based on YOLO. In Proceedings of the 2018 1st IEEE international conference on knowledge innovation and invention (ICKII), Jeju Island, Korea, 23–27 July 2018; pp. 221–224.
16. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
17. Wu, W.; Yin, Y.; Wang, X.; Xu, D. Face detection with different scales based on faster R-CNN. *IEEE Trans. Cybern.* **2018**, *49*, 4017–4028. [CrossRef]
18. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 91–99. [CrossRef] [PubMed]
19. Loey, M.; Manogaran, G.; Taha, M.H.N.; Khalifa, N.E.M. Fighting against COVID-19: A novel deep learning model based on YOLO-v2 with ResNet-50 for medical face mask detection. *Sustain. Cities Soc.* **2021**, *65*, 102600. [CrossRef] [PubMed]
20. Redmon, J.; Farhadi, A. YOLO9000: Better, faster, stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 7263–7271.
21. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
22. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 July 2015; pp. 1–9.
23. Jiang, M.; Fan, X.; Yan, H. Retinamask: A face mask detector. *arXiv* **2020**, arXiv:2005.03950.
24. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
25. Mohan, P.; Paul, A.J.; Chirania, A. A tiny CNN architecture for medical face mask detection for resource-constrained endpoints. In *Innovations in Electrical and Electronic Engineering*; Springer: Berlin, Germany, 2021; pp. 657–670.
26. Lim, H.; Ryoo, S.; Jung, H. Face-Mask Detection with Micro processor. *J. Korea Inst. Inf. Commun. Eng.* **2021**, *25*, 490–493.
27. Raza, W.; Osman, A.; Ferrini, F.; Natale, F.D. Energy-Efficient Inference on the Edge Exploiting TinyML Capabilities for UAVs. *Drones* **2021**, *5*, 127. [CrossRef]
28. Ng, A. Clustering with the k-means algorithm. *Mach. Learn.* **2012**, *36*, 451–461.
29. Ng, A. Advice for applying machine learning. In *Machine Learning*; 2011. Available online: https://see.stanford.edu/materials/aimlcs229/ml-advice.pdf (accessed on 29 August 2021).
30. Cabani, A.; Hammoudi, K.; Benhabiles, H.; Melkemi, M. MaskedFace-Net—A Dataset of Correctly/Incorrectly Masked Face Images in the Context of COVID-19. *Smart Health* **2020**, *19*, 100144. [CrossRef]
31. Hammoudi, K.; Cabani, A.; Benhabiles, H.; Melkemi, M. Validating the Correct Wearing of Protection Mask by Taking a Selfie: Design of a Mobile Application "CheckYourMask" to Limit the Spread of COVID-19. *Comput. Model. Eng. Sci.* **2020**, *124*, 1049–1059. [CrossRef]
32. Roy, P.; Ghosh, S.; Bhattacharya, S.; Pal, U. Effects of Degradations on Deep Neural Network Architectures. *arXiv* **2018**, arXiv:1807.10108.
33. Karras, T.; Laine, S.; Aila, T. A style-based generator architecture for generative adversarial networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 4401–4410.
34. Chen, R.C. Automatic License Plate Recognition via sliding-window darknet-YOLO deep learning. *Image Vis. Comput.* **2019**, *87*, 47–56.