

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Evaluating Embedded Wildfire Detection Algorithms for the Wildland-Urban Interface

Rodrigo Santa Comba Coelho da Silva



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Daniel Gouveia Costa

October 23, 2024

Resumo

Os incêndios florestais na Interface Urbano-Florestal (IUF) representam uma ameaça significativa à segurança pública e à propriedade, indicando a necessidade de sistemas avançados de detecção atempada de incêndios que não sejam apenas económicos e de baixa manutenção, mas também capazes de operar em locais remotos. Esta tese aborda essa necessidade, desenvolvendo um sistema de detecção de incêndios florestais incorporado e otimizado para a IUF, utilizando modelos de redes neurais convolucionais implementados em hardware com recursos limitados.

A abordagem envolve a criação de uma base de dados de imagens, abrangendo diversos cenários de incêndios florestais, o treino e avaliação de múltiplos modelos de redes neurais convolucionais neste conjunto de dados e, em seguida, a implementação do modelo mais eficaz num Arduino Nano 33 BLE Sense. A base de dados de imagens é criada de modo a garantir o equilíbrio entre várias categorias, assegurando o desempenho do modelo e melhorando a sua capacidade de generalização. Além disso, são empregadas técnicas de aprendizagem por transferência para acelerar o processo de treino e melhorar o desempenho do modelo.

Os modelos são testados e comparados utilizando métricas como a precisão global, a precisão de verdadeiros positivos, a área sob a curva ROC e o tempo de inferência. O modelo com melhor desempenho é selecionado com base nestas métricas e é então implementado no hardware alvo. Adicionalmente, é concebida uma caixa protetora personalizada para proteger o hardware de fatores ambientais, garantindo assim a sua funcionalidade e durabilidade em cenários de implementação no mundo real.

Esta pesquisa procura superar as limitações dos métodos de detecção de incêndios florestais existentes, aumentando a segurança pública e reduzindo os danos materiais na IUF.

Palavras-chaves: Detecção de Incêndios Florestais, Aprendizagem Computacional, Classificação de Imagens, Arduino Nano 33 BLE Sense, Embarcado, MobileNetV2

Abstract

Wildfires in the Wildland-Urban Interface (WUI) pose a significant threat to public safety and property, indicating the need for advanced early detection systems that are not only cost-effective and low-maintenance but also capable of operating in remote locations. This thesis addresses this need by developing an embedded wildfire detection system optimized for the WUI, using Convolutional Neural Network (CNN) models deployed on resource-constrained hardware.

The approach involves building a balanced image dataset encompassing diverse wildfire scenarios, training and evaluating multiple CNN models on this dataset and then deploying the most effective model onto an Arduino Nano 33 BLE Sense. The dataset is curated to ensure balance across various categories, improving its reliability and generalization ability. Additionally, transfer learning techniques are employed to expedite the training process and enhance model performance.

The models are tested and compared using metrics such as global accuracy, true positive accuracy, area under the ROC curve, and inferencing time. The best-performing model is selected based on these metrics and is then deployed on the target hardware. Additionally, a custom enclosure is designed to safeguard the hardware from environmental factors, thereby ensuring its functionality and durability in real-world deployment scenarios.

This research seeks to overcome the limitations of existing wildfire detection methods, enhancing public safety and reducing property damage in the WUI.

Key-words: Wildfire Detection, Machine Learning, Image Classification, Arduino Nano 33 BLE Sense, Embedded, MobileNetV2

Sustainable Development Goals (SDG)

The United Nations Sustainable Development Goals (SDGs) are a collection of 17 interconnected goals designed to be a "blueprint to achieve a better and more sustainable future for all" [1]. The SDGs address global challenges, including those related to poverty, inequality, climate change, environmental degradation, peace, and justice.

Wildfire detection technologies and the associated research and development efforts can significantly contribute to several SDGs.

SDG 3: Good Health and Well-being

Wildfires pose significant health risks due to smoke inhalation and injuries. Early detection can help mitigate these risks by enabling faster response and evacuation, thus safeguarding public health.

SDG 6: Clean Water and Sanitation

Wildfires can contaminate water sources and disrupt sanitation systems. Effective wildfire detection and suppression can help protect water resources and ensure access to clean water for affected communities.

SDG 9: Industry, Innovation, and Infrastructure

Wildfire detection systems often rely on cutting-edge technologies, such as machine learning, remote sensing, and sensor networks. Developing and deploying these technologies drive innovation in the engineering and technology sectors, fostering industrial growth and creating new economic opportunities. Additionally, these systems can be integrated into existing infrastructure, such as communication networks and data centers, to enhance monitoring and response capabilities, contributing to the development of resilient infrastructure.

SDG 11: Sustainable Cities and Communities

Wildfires pose a significant threat to communities, particularly in the wildland-urban interface. Effective wildfire detection systems are crucial for protecting these communities by providing early warnings, enabling timely evacuations, and minimizing the loss of life and property. By safeguarding communities and their infrastructure, these technologies contribute to the development of sustainable and resilient cities.

SDG 13: Climate Action

Climate change is a major driver of increased wildfire risk, as rising temperatures and prolonged droughts create conditions conducive to fire ignition and spread. Wildfire detection systems are

essential for climate action because they enable rapid response and containment, reducing greenhouse gas emissions from prolonged burning. Early detection also helps to minimize the area burned, preserving carbon sinks like forests and mitigating the overall impact of wildfires on the climate.

SDG 15: Life on Land

Wildfires have devastating consequences for terrestrial ecosystems, destroying habitats, displacing wildlife, and contributing to biodiversity loss. Early detection and rapid response are crucial for minimizing the ecological damage caused by wildfires. By enabling firefighters to suppress fires before they spread extensively, these technologies help protect forests, grasslands, and other vital ecosystems.

The work on wildfire detection detailed in this study demonstrates the potential for technology to positively impact various SDGs. By protecting communities, safeguarding public health, preserving water resources, fostering innovation, and mitigating climate change, wildfire detection technologies are not merely tools for disaster management but also instruments for sustainable development. As the world faces the growing threat of wildfires, continued investment in the development of these technologies is essential for building a more resilient and sustainable future.

Agradecimentos

Muito obrigado a todos o que acompanharam o meu percurso académico. Obrigado por toda a ajuda e apoio nos estudos e trabalhos e também por todos os momentos de relaxamento e diversão. Um especial obrigado à minha família por tudo.

Rodrigo Santa Comba

“Eles sabem coisas...!”

João Nogueira

Contents

1	Introduction	1
1.1	Context	1
1.2	Evolution of Wildfire Detection	2
1.3	Motivation	3
1.4	Goals	3
1.4.1	System Requisites	3
1.4.2	Methodology	4
1.5	Document Structure	5
2	Literature Review	7
2.1	Fundamentals of Deep Learning	7
2.2	Fundamentals of Convolutional Neural Networks (CNNs)	8
2.3	Evolution	10
2.4	The Rise of Lightweight CNNs	11
2.5	Transfer Learning (TL)	13
2.6	Visual Sensors and TinyML	14
2.6.1	Key Components and Advancements	14
2.6.2	Applications in Wildfire Detection and Prevention	15
2.7	Wildfire Image Dataset	15
2.7.1	Dataset Splitting for Model Evaluation	17
2.8	Edge Impulse Platform	18
2.8.1	Studio	18
2.8.2	API	19
2.8.3	SDK	19
2.8.4	EON-Compiler	19
2.9	Bringing Machine Learning (ML) to the Edge	19
3	Data & Methodology	21
3.1	Constructing the datasets	22
3.1.1	D-Fire	22
3.1.2	FF Dataset	22
3.1.3	Hand-Captured and Searched Photographs	22
3.1.4	Dataset Categories	23
3.1.5	Image Inclusion and Dataset Cleaning	23
3.1.6	Final Dataset	24
3.2	<i>Random Sample Maker (RSM)</i>	26
3.2.1	Requirements	26
3.2.2	Usage and Functionality	27

3.2.3	Downloading and Running <i>Random Sample Maker</i>	30
3.3	Testing and Training Datasets	30
3.3.1	Testing Dataset	31
3.3.2	Training Dataset	32
3.4	Training, Testing and Deploying CNN Models on the Edge Impulse Studio	33
3.4.1	Uploading the Dataset	33
3.4.2	Training	34
3.4.3	Testing	35
3.4.4	Evaluation Metrics	35
3.4.5	Deploying the CNN models	36
3.5	Choice of Models and Training Dataset Image Sizes	37
3.5.1	Comparisons and Choice of Model and Image Size	38
3.6	Final Model Optimization and Tunning	38
3.7	Model Comparison with the Literature	40
4	Hardware, Enclosure Build and Assembly Process	41
4.1	Choosing the hardware platform for TinyML-based visual sensing	41
4.2	Arduino Tiny Machine Learning Kit	42
4.3	Enclosure build using SolidWorks	43
4.3.1	Base	44
4.3.2	Lid	44
4.3.3	Assembly and Preview	45
4.4	Final 3D printed enclosure and assembly process	45
4.4.1	Assembly process	46
4.4.2	Power supply and charging	47
5	Tests & Results	48
5.1	Testing Different Approaches for Generating the Training Dataset	48
5.1.1	An approach with balanced categories	49
5.1.2	An approach with balanced categories and slight augmentation	51
5.1.3	An approach without considering balance for training	52
5.1.4	Comparison between approaches and choice of approach for generating the training dataset	54
5.2	Testing Various Models and Image Sizes	56
5.2.1	Estimated Inferencing Time, RAM Usage and Flash Usage	56
5.2.2	Best Models Complying with the Hardware Constraints	60
5.2.3	MobileNetV1, 128x128p, alpha=0.25	60
5.2.4	MobileNetV2, 96x96 alpha=0.1	61
5.2.5	MobileNetV2, 32x32, alpha=0.35	61
5.2.6	Best Performing Model for Each Configuration	62
5.3	Final Model Optimization and Tunning	63
5.4	Testing Final Model on Available Datasets	65
6	Discussion	67
7	Conclusion and Future Work	69
7.1	Objectives Achieved	69
7.1.1	Dataset Created	69
7.1.2	Random Sample Maker (RSM)	70

7.1.3	Models Trained	70
7.1.4	Deployment of Models	70
7.1.5	Enclosure Build	70
7.2	Future Works	71
A		80
A.1	Batch File	80
A.2	Dataset Folder Structure	81
A.3	On-Device Performance Metrics	82
A.3.1	Dataset References	83
A.3.2	Output Excel File from <i>RSM</i> tool	84
A.4	Training Hyperparameters for Different Numbers of Neurons	85

List of Figures

2.1	The convolution operation	9
2.2	MobileNet's building block	12
2.3	ShuffleNet's residual block	12
2.4	EfficientNet's scaling method	13
2.5	SqueezeNet's fire module	13
3.1	General methodology for the construction of TinyML-based wildfire detection systems.	21
3.2	Process for adding images from existing datasets to the main dataset	24
3.3	Process for adding individual images to the main dataset	24
3.4	Example images of wildfire and non-wildfire per category	25
3.5	Dataset building process	25
3.6	<i>Random Sample Maker's</i> example	26
3.7	<i>Random Sample Maker's</i> UI	29
3.8	<i>Random Sample Maker's</i> UI warning	30
3.9	Generating the testing dataset.	31
3.10	Generating training dataset	33
3.11	Edge Impulse: Data Acquisition	34
3.12	Edge Impulse: Training Expert Mode	34
3.13	Arduino IDE's serial monitor displaying class probabilities directly from the board	36
3.14	On-Device estimates testing process	37
4.1	Arduino Tiny Machine Learning Kit	43
4.2	Base design in SolidWorks	44
4.3	Lid design in SolidWorks	45
4.4	SolidWorks previews	45
4.5	System Assembly	46
4.6	Charging process	47
5.1	Accuracies obtained by random sampling from categories and corresponding variance. The curves are polynomial approximations of degree 3 and degree 4, respectively.	50
5.2	Accuracies obtained by random sampling from categories with augmentation for under-represented categories and corresponding variance. The curves are polynomial approximations of degree 5 and degree 3, respectively.	51
5.3	Accuracies obtained by random sampling without categories and corresponding variance. The curves are polynomial approximations of degree 4 and degree 3, respectively.	53

5.4	Average accuracies obtained by random sampling using 3 different methods. The curves are simple smooth lines connecting the dots for representation.	54
5.5	Change in inferencing time, RAM usage and flash usage with respect to MobileNetV1's alpha value	56
5.6	Change in inferencing time, RAM usage and flash usage with respect to MobileNetV1's input image size	57
5.7	Change in inferencing time, RAM usage and flash usage with respect to MobileNetV2's alpha value	58
5.8	Change in inferencing time, RAM usage and flash usage with respect to MobileNetV2's input image size	59
5.9	Learning rate optimization	64
A.1	Final dataset: wildfire and non-wildfire folder structures	81
A.2	Tuned training hyperparameters for different numbers of neurons	85

List of Tables

3.1	D-FIRE: number of images and corresponding labels	22
3.2	Final model's training hyperparameters	39
3.3	Final model's performance metrics	39
5.1	Accuracy and average accuracy comparison	55
5.2	MobileNetV1, alpha=0.125, image size = 128x128p - Performance metrics for different numbers of neurons in one dense layer prior to the classification layer.	61
5.3	MobileNetV2, alpha = 0.1, image size = 96x96 - Performance metrics for different numbers of neurons in one dense layer prior to the classification layer.	61
5.4	MobileNetV2, alpha = 0.35, image size = 32x32 - Performance metrics for different numbers of neurons in one dense layer prior to the classification layer.	62
5.5	Summary of performance metrics for best configurations	62
5.6	Performance metrics for different dropout rates.	63
5.7	Performance metrics for different dropout rates with batch normalization.	63
5.8	Performance metrics for different learning rates.	64
5.9	Comparison of performance metrics with the literature	66
A.1	Testing different models on different sized images (NxN pixels) using different methods for compiling the model: TFlite (TFL), EON-Compiler (EC) and EON-Compiler RAM Optimised (ECR), to obtain Inferencing Time (IF), Peak RAM Usage (PRU) and Flash Usage (FU). Models selected for testing are displayed in yellow background colour.	82
A.2	List of datasets and sources used and respective references	83
A.3	Number of images in each folder	84

Abbreviations & Symbols

AUC-ROC	Area Under the ROC Curve
API	Application Programming Interface
BLE	Bluetooth Low Energy
CNN	Convolutional Neural Network
CPU	Central Processing Unit
FS	Folder Structure
GIS	Geographic Information System
IoT	Internet of Things
ILSRVC	ImageNet Large-Scale Visual Recognition Challenge
ML	Machine Learning
NAS	Neural Architecture Search
RAM	Random Access Memory
ROC	Receiver Operating Characteristic
RSM	Random Sample Maker
SDG	Sustainable Development Goals
SRAM	Static Random-Access Memory
TL	Transfer Learning
UAV	Unmanned Aerial Vehicle
WUI	Wildland-Urban Interface
YOLO	You Only Look Once

Chapter 1

Introduction

In this chapter, the motivations and objectives of this work are explained and the document structure is outlined. This chapter provides an insight into wildfire detection at the Wildland-Urban Interface (WUI), the challenges faced by current detection methods and the strategies proposed in this work to enhance detection capabilities. The focus is on developing CNN-based fire detection models, leveraging the Edge Impulse platform for efficient model development and deployment. For that, a comparative analysis of various techniques and model architectures is conducted to identify the most effective wildfire detection models and the most promising one is selected for further optimization and deployment.

1.1 Context

Wildfires pose a significant threat, especially at the WUI, where urban areas meet natural vegetation. The increasing frequency and intensity of wildfires emphasize the need for early detection to ensure public safety and minimize property damage.

Effective wildfire detection requires the ability to identify fire and smoke under diverse environmental conditions and development stages of the wildfire. Ideal detection systems should be cost-effective, low-maintenance, and capable of operating in remote locations.

The WUI presents unique challenges due to the proximity of human-made structures to flammable vegetation, requiring specialized detection systems to mitigate the risk of fire spreading from wildlands into urban areas. Research indicates that the WUI is a high-risk environment for wildfires [2].

To enhance wildfire detection capabilities, advancements in technology offer promising solutions. Satellite imagery, unmanned aerial vehicles (UAVs), and ground-based sensor networks have the potential to provide real-time data on fire initiation, spread, and behavior [3, 4, 5]. However, the diversity of such technologies fosters the evaluation of effective approaches in WUI scenarios.

1.2 Evolution of Wildfire Detection

The evolution of fire detection systems for wildfires has seen significant advancements over the years, driven by the increasing need to protect both natural and human habitats from their devastating impacts.

Historically, fire detection relied heavily on human observation from watchtowers and look-out stations. These methods, while somewhat effective, were labor-intensive and limited by the observer's range of vision and weather conditions, such as the Osborne Fire Finder [6]. In the mid-20th century, advancements in technology introduced aerial reconnaissance using manned aircrafts, which expanded the area that could be monitored but still depended heavily on human interpretation.

Before the advent of Machine Learning (ML) and advanced computer vision techniques, early automated fire detection systems utilized handmade models based on simple algorithms and rule-based systems. These models were designed to identify specific patterns of smoke or fire in images and were often implemented in ground-based sensors and cameras. These rule-based systems typically analyzed color, motion, and shape features to detect fire and smoke. For example, algorithms were developed to recognize the unique color signatures of flames and smoke [7]. Despite their innovative approach at the time, these systems faced limitations in sensitivity and accuracy due to the complexity of distinguishing fire from other similar visual phenomena such as fog or clouds. The models often required manual calibration and were not adaptive to varying environmental conditions, which limited their effectiveness and reliability in early fire detection.

The development of remote sensing technologies marked a significant leap forward in wildfire detection. Satellites equipped with thermal and optical sensors began to provide real-time data on fire hotspots, smoke plumes, and burned areas. These systems allowed for continuous monitoring over large geographic areas, improving early detection and response times. Key satellites like NASA's MODIS (Moderate Resolution Imaging Spectroradiometer) and VIIRS (Visible Infrared Imaging Radiometer Suite) have been instrumental in global fire monitoring efforts [8].

The integration of Geographic Information Systems (GIS) with remote sensing data further enhanced wildfire management capabilities. GIS enables the mapping of fire risk areas, the tracking of fire spread, and the allocation of firefighting resources.

In recent years, ML has emerged as a powerful tool in the field of wildfire detection. ML algorithms, particularly CNNs, have shown great promise in analyzing images from various sources, including satellites, drones, and ground-based cameras, to detect early signs of wildfires. These systems can identify fire and smoke patterns with high accuracy, significantly reducing false alarms and enabling quicker responses [9, 10]. CNN architectures such as YOLO (You Only Look Once) neural networks [11], have been applied to drone applications [12], providing real-time analysis and improving the speed and accuracy of fire detection.

Therefore, autonomous systems for wildfire detection and control represent the cutting edge of current research and development. These systems combine advanced sensors, ML algorithms, and

autonomous platforms such as drones and ground robots to detect and even combat fires without direct human intervention [13].

1.3 Motivation

Developing a wildfire detection system is inherently challenging. It must identify fire and smoke across various distances, lighting conditions, and even different stages of the wildfire's development. The challenge scales exponentially when considering vast areas of land and low maintenance, cost-effective deployment and functionality in remote locations further complicates the design.

While various effective wildfire detection systems exist, none have yet been developed that can resolve all these issues. To address the previously mentioned challenges, this work proposes a system utilizing a camera as the primary sensor for wildfire detection. The captured image would then be fed into an image classification model residing on a low-power microprocessor. This model would analyze the image and classify it as either 'wildfire' or 'non-wildfire'.

There has been much recent research on detecting wildfires, however, some areas still require further development. For instance, there has not been a sufficient focus on developing a proper well-balanced database of images depicting many different wildfire scenarios. Without such a dataset, attempts to evaluate a given system to determine its real-world accuracy are futile. Developing this comprehensive dataset is also a key objective of this work.

Equally important is ensuring that the system can operate in real-world environments. Wildfire detection systems must be tested under actual environmental conditions, including exposure to heat, moisture, and dust. Without protection, the system's hardware could fail during such tests, preventing the collection of valuable performance data. A protective enclosure is therefore crucial to shield the hardware, allowing it to function throughout field testing. This enables the gathering of data that will inform refinements in both hardware durability and algorithm performance, ultimately contributing to the development of a more robust, field-ready solution.

1.4 Goals

The goal of this work is to build an embedded wildfire detection system for the WUI by training and evaluating different CNN models on a balanced image dataset and deploying the most effective model on a low-power embedded hardware system.

1.4.1 System Requisites

The system requirements relate to the essential criteria and specifications that the final system must fulfill. These requirements define the constraints and capabilities that the system must adhere to in order to ensure its proper functionality and reliability.

- **Image-based detection:** The wildfire detection system must be capable of detecting the presence of fire and smoke using images captured by a camera. The system should be able to identify fire-related events in a variety of scenarios, including different lighting conditions (e.g., day, night, evening) and distances (close, medium, far).
- **Capture and processing interval:** The system must capture an image and perform fire detection at least every 10 seconds. This frequency ensures a timely response to potential fire hazards while balancing computational load and power consumption.
- **Battery-powered operation:** The system must be able to function autonomously, powered by batteries. This requirement is crucial for deployment in remote WUI areas, where access to grid power may not be feasible. The hardware design must therefore prioritize energy efficiency to extend operational time.
- **Low maintenance:** The system must be designed to operate with minimal maintenance requirements. This includes durable hardware and long-lasting battery life, as well as the ability to function in harsh environments without frequent human intervention.
- **Scalable and generalizable detection:** The system must be capable of detecting wildfires in diverse environmental conditions, with models that generalize well to various wildfire scenarios. The system's dataset and detection algorithms should ensure robust performance across different types of fires and geographical areas.

1.4.2 Methodology

The methodology relates to the specific procedures and actions required to achieve the overall objectives of the study which are developing, evaluating and deploying an effective wildfire detection system for the WUI. These goals outline the practical steps necessary to conduct the research effectively.

1. **Building a Balanced Dataset:** Developing a comprehensive dataset with a focus on balance, ensuring it accurately represents various wildfire scenarios. This includes collecting images depicting fires and smoke during different times of the day and night, at varying distances, and in both urban and rural settings.
2. **Employing Different CNN Models:** Finding CNN models that respect the hardware constraints and train them on the dataset created.
3. **Testing and Comparing Different Models:** Conducting a comparative analysis of the different CNN models to evaluate their performance. This involves assessing metrics such as global accuracy, true positive accuracy and Area Under the ROC Curve (AUC-ROC) [14] and also the inferencing time, in order to determine the most effective model for wildfire detection.

4. **Selecting and Deploying the Best Model:** Based on the comparative analysis, selecting the best-performing CNN model and deploying it on the Arduino Nano 33 BLE Sense, which is the chosen hardware platform. This step ensures that the model can operate efficiently on resource-constrained hardware.
5. **Choosing the Hardware and Building an Enclosure:** Designing and construct an enclosure for the Arduino Nano 33 BLE Sense and its associated hardware. The enclosure must protect the system from environmental factors, ensuring its durability and functionality for real-world testing.

1.5 Document Structure

The structure of this document is organized to provide a comprehensive understanding of the development, testing and deployment of wildfire detection algorithms using CNNs. Each chapter is designed to build upon the previous one, ensuring a logical flow of information.

Introduction: This chapter introduces the motivations and objectives of the work, providing insight into wildfire detection at the WUI. It discusses the challenges faced by current detection methods and outlines the strategies proposed in this work to enhance detection capabilities using CNN-based models.

Literature Review: The literature review chapter discusses the most appropriate lightweight architectures for object detection and image classification, focusing on CNN models and Transfer Learning (TL) techniques. It also details the building process for image datasets, focusing on wildfire detection. Finally, it reviews the Edge Impulse platform and the hardware considerations for a wildfire detection system.

Methodology: The methodology chapter describes the development of a new wildfire image dataset from existing sources, the use of the developed Random Sample Maker (RSM) tool to balance the dataset, the specific training and testing details for developing and evaluating CNN models. Additionally, it covers the hardware and enclosure build for real-world testing.

Hardware and Enclosure Build: The hardware and enclosure build chapter discusses the selection of a suitable hardware for TinyML-based visual sensing, focusing on processing power, energy efficiency, and usability. It also covers the design and construction of an enclosure to protect the hardware in real-world conditions.

Tests & Results: The tests and results chapter presents the experiments conducted for evaluating different approaches of generating the training dataset and for testing various models trained and various image sizes. It provides a comparative analysis of the approaches and models, detailing their performance metrics, including inferencing time, Random Access Memory (RAM) usage, and flash usage.

Conclusion: The conclusion chapter summarizes the work done, highlighting the objectives achieved. It also discusses potential future work and improvements to the system.

Appendix: The appendix is designed to support the main content of the thesis with supplementary data and resources.

Chapter 2

Literature Review

This chapter describes the fundamental theories and recent advancements in computer vision, focusing on CNNs and their applications in wildfire detection. It explores the evolution of CNNs, highlighting significant milestones and innovations that have enhanced image detection systems. This chapter also examines the emergence of lightweight CNN models, designed to optimize accuracy and efficiency, making them ideal for deployment on devices with limited computational resources. Furthermore, it covers TL principles and its benefits for improving model performance with limited data. Additionally, it reviews the integration of TinyML with visual sensors for intelligent edge applications and the use of the Edge Impulse platform for deploying ML models on resource-constrained hardware. Finally, it reviews the considerations to be had regarding the building and usage of datasets, the Edge Impulse platform, and the hardware constraints associated with embedded wildfire detection systems

2.1 Fundamentals of Deep Learning

Deep learning involves training artificial neural networks to learn representations and perform complex tasks [15]. Understanding the following fundamental concepts is crucial for comprehending the inner workings and effective utilization of deep neural networks:

Learning Rate:

The learning rate is a hyperparameter that governs the magnitude of updates applied to the model's weights during training [15]. A suitable learning rate is essential for efficient and stable convergence to an optimal solution.

Overfitting and Underfitting:

Overfitting occurs when a model becomes overly specialized to the training data, capturing noise and irrelevant details, leading to poor generalization on unseen data. On the other hand, underfitting refers to a model's inability to capture the underlying patterns in the data, resulting in subpar

performance on both training and testing data [16].

Loss Functions:

Loss functions quantify the discrepancy between the model's predictions and the ground truth labels. They provide the feedback signal that guides the optimization process, steering the model towards learning the desired patterns.

Optimization Algorithms:

Optimization algorithms are used to iteratively update the model's weights based on the calculated gradients of the loss function with respect to the weights

2.2 Fundamentals of Convolutional Neural Networks (CNNs)

CNNs have revolutionized the field of computer vision by automating feature extraction, leading to significant improvements in accuracy and efficiency across various tasks [17]. Unlike traditional methods that rely on manually engineered features, CNNs learn these features directly from the data through a series of layers and mathematical operations.

Convolutional Layer:

The convolutional layer is the fundamental building block of a CNN. It employs a filter, also known as a kernel, that slides across the input image, performing element-wise multiplications and summations to detect specific features like edges, textures, or patterns. This operation is mathematically expressed as a convolution [18]:

$$(I * K)(i, j) = \sum_m \sum_n I(m, n) \cdot K(i - m, j - n) \quad (2.1)$$

Where:

- I is the input image or feature map
- K is the convolution kernel or filter
- (i, j) are the coordinates of the output feature map
- (m, n) are the coordinates of the current element in the kernel
- (x - m, y - n) are the coordinates of the element in the input being multiplied

By applying multiple filters, each designed to detect different features, the convolutional layer produces a set of feature maps that capture the spatial relationships within the image. This process is exemplified in Figure 2.1.

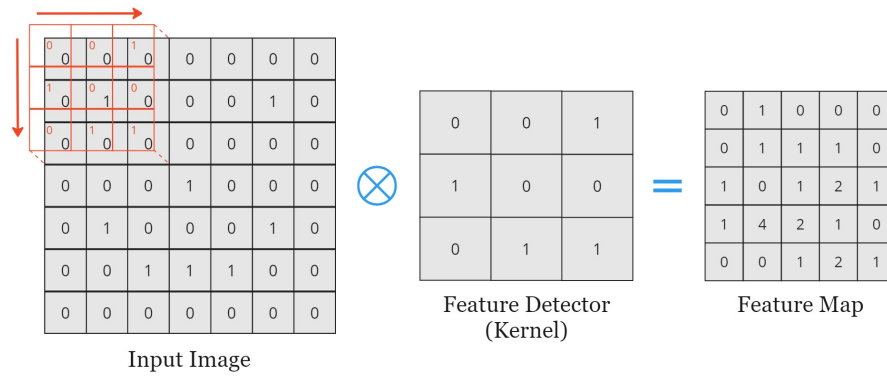


Figure 2.1: The convolution operation, adapted from [19]

Types of Layers

In addition to the convolutional layer, CNNs employ various other layers to enhance their learning capabilities and performance. These layers combine to extract, process, and refine features from the input data, ultimately leading to accurate predictions.

Activation Layer:

Following the convolutional layer, an activation function is applied to introduce non-linearity into the model. This allows the CNN to learn complex and intricate patterns that cannot be captured by linear transformations alone. A commonly used activation function is the Rectified Linear Unit (ReLU), which outputs the input if it is positive and zero otherwise.

Pooling Layers:

Pooling layers are often inserted between convolutional layers to reduce the spatial dimensions of the feature maps while preserving the most salient features. This helps to decrease the computational complexity of the model and prevent overfitting. Common pooling operations include max pooling, which selects the maximum value within a local region, and average pooling, which computes the average value within a local region.

Normalization Layers:

Normalization layers, such as Batch Normalization, help stabilize training by normalizing the activations within each batch [20], allowing for higher learning rates.

Dropout Layers:

Dropout layers are a regularization technique that randomly deactivates a fraction of neurons during training [21]. This prevents overfitting by reducing the interdependence of neurons and forcing the network to learn more robust features.

Fully Connected Layers:

Fully connected layers are typically found towards the end of a CNN and are used for high-level reasoning [22]. Each neuron in a fully connected layer is connected to every neuron in the previous layer, allowing the network to combine the extracted features and make the final prediction for tasks like image classification.

CNN Architectures:

CNN architectures often comprise multiple convolutional, activation, and pooling layers stacked sequentially, with fully connected layers at the end [23]. This hierarchical structure enables the CNN to learn progressively more abstract features from the input data, from basic edges and textures to high-level concepts.

Impact on Computer Vision:

The introduction of CNNs has marked a paradigm shift in computer vision. Their ability to automatically learn task-specific features has led to remarkable achievements in various applications, including image classification, object detection, semantic segmentation, and image generation [24]. CNNs continue to be a driving force in advancing the state-of-the-art in computer vision research and applications.

2.3 Evolution

CNNs have transformed the field of computer vision, particularly in image detection. Prior to their emergence, image detection heavily relied on hand-crafted features and complex algorithms, which often proved to be computationally expensive and limited in accuracy [25].

The year 2012 marked a turning point with the introduction of AlexNet [17]. This deep learning model achieved a remarkable breakthrough in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [26], significantly outperforming previous methods. AlexNet's success demonstrated the power of CNNs in learning complex features directly from raw image data paving the way for further advancements.

Following AlexNet's success, numerous CNN architectures emerged, each contributing to advancements in image detection accuracy and efficiency. VGGNet [27] explored the concept of depth in CNNs, stacking multiple convolutional layers to achieve even better results on the ILSVRC.

Building upon these advancements, the YOLO framework [11] introduced a novel approach to object detection. Unlike traditional methods, YOLO predicts bounding boxes and class probabilities for objects in a single forward pass, making it significantly faster for real-time applications.

Subsequent iterations of YOLO, such as YOLOv2, YOLOv3, and YOLOv4, refined the approach, improving both speed and accuracy. These advancements, fueled by the power of CNNs,

have enabled real-time object detection in various applications, including autonomous vehicles [28], surveillance systems [29], and robotics [30].

2.4 The Rise of Lightweight CNNs

While traditional CNN architectures achieved remarkable performance, their computational demands often limited their deployment on resource-constrained devices like mobile phones, drones and robots. This sparked the development of lightweight CNN models, specifically designed to balance accuracy with efficiency.

These models achieve this balance through various architectural optimizations:

- **Depthwise Separable Convolutions:** This technique decomposes standard convolutions into depthwise and pointwise convolutions, significantly reducing the number of parameters and computations while maintaining comparable performance [31].
- **Reduced Filter Size and Channels:** By employing smaller filter sizes (e.g., 3x3) and fewer channels, lightweight models decrease the computational complexity without sacrificing essential feature extraction capabilities [17].
- **Pruning and Quantization:** These methods remove redundant parameters and reduce the precision of weights and activations, further compressing the model size and accelerating inference speed [32].
- **Efficient Network Architectures:** New architectures like MobileNet, ShuffleNet and SqueezeNet have been specifically designed for mobile and embedded applications, incorporating innovative design principles such as inverted residual blocks, channel shuffling, and bottleneck layers, to optimize resource utilization and efficiency while preserving accuracy [33, 34, 35].

These advancements have led to the development of several successful lightweight CNN models

MobileNet

MobileNet is a lightweight model specifically designed to achieve high accuracy on mobile and embedded devices [33]. It employs depthwise separable convolutions, which split the convolution operation into two layers: a depthwise convolution that applies a single filter per input channel, and a pointwise convolution that uses a 1x1 convolution to combine the outputs of the depthwise layer. This significantly reduces the number of parameters and computational cost, making MobileNet highly efficient for real-time applications on devices with limited processing power. Subsequent versions, MobileNetV2 and MobileNetV3, further improved upon the original architecture. MobileNetV2 introduced the inverted residual block and linear bottlenecks, enhancing the network's efficiency and accuracy [36]. MobileNetV3 incorporated a combination of network architecture search (NAS) and other techniques resulting in even faster and more accurate models [37].

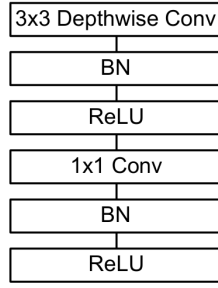


Figure 2.2: MobileNet's building block [33]

ShuffleNet

ShuffleNet introduces the concept of channel shuffling to enhance the flow of information between different feature channels [34]. It uses pointwise group convolutions to reduce the number of parameters, and then shuffles the channels between group convolutions to ensure that the outputs from different groups are mixed, thus maintaining a high level of information exchange. This design allows ShuffleNet to achieve competitive accuracy with fewer parameters and lower computational cost, making it suitable for resource-constrained environments.

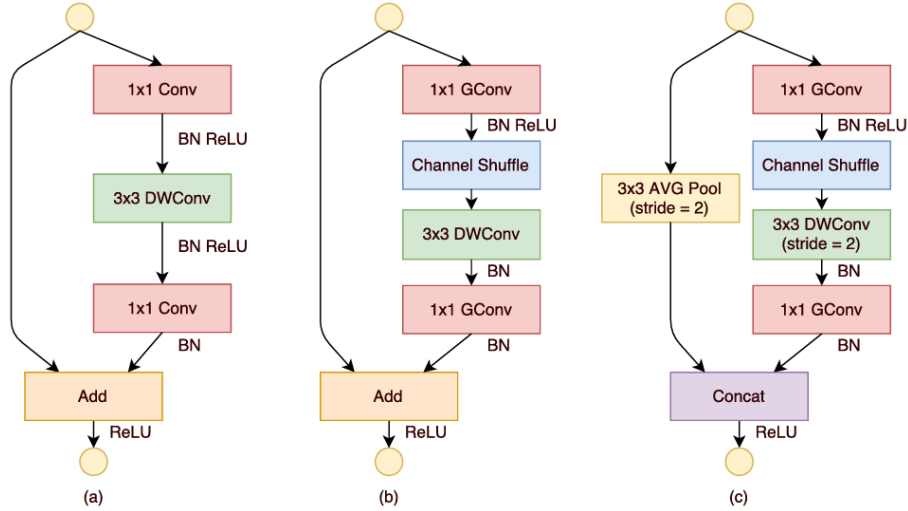


Figure 2.3: ShuffleNet's residual block [34]

EfficientNet

EfficientNet explores a compound scaling method, automatically scaling network depth, width, and resolution based on a fixed resource constraint (Fig. 2.4). These models are derived from a baseline model optimized through neural architecture search (NAS), an automated process to discover optimal network structures. EfficientNets, derived from this baseline, achieve state-of-the-art accuracy while being computationally efficient compared to previous architectures [38].

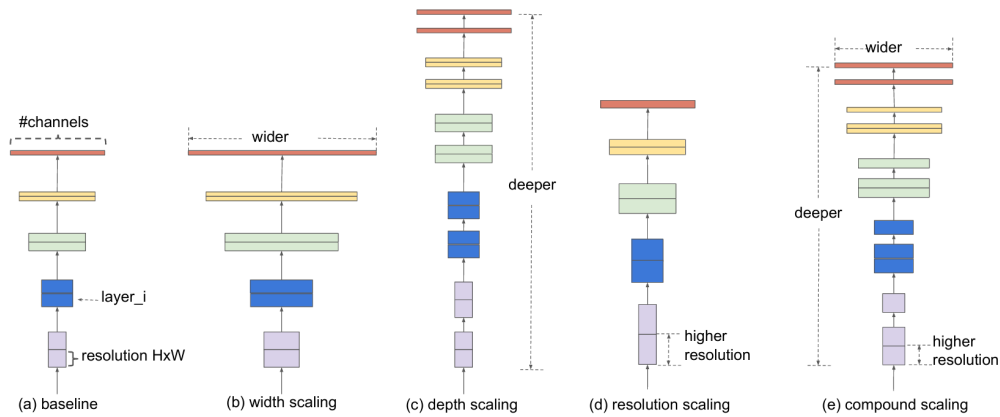


Figure 2.4: EfficientNet's scaling method [38]

SqueezeNet

SqueezeNet is designed to achieve AlexNet-level accuracy with 50 times fewer parameters, SqueezeNet employs a "fire module" which consists of a squeeze convolution layer with fewer filters feeding into an expand layer with more filters [35] as shown in Figure 2.5. This architecture reduces the number of parameters while maintaining performance, making it highly suitable for deployment on devices with limited computational resources .

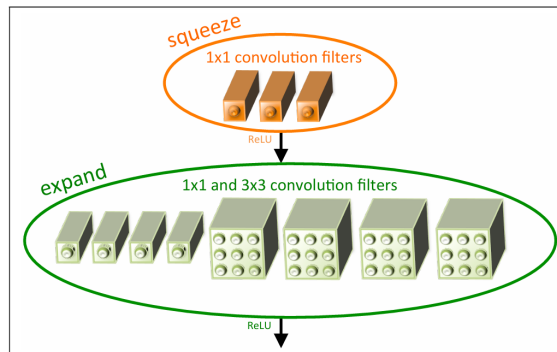


Figure 2.5: SqueezeNet's fire module [35]

A comprehensive comparison of various lightweight CNN models is presented in [19]. This survey analyzes different models, their design choices, and their performance on various tasks.

By leveraging these advancements, lightweight CNNs bridge the gap between powerful deep-learning models and resource-constrained environments.

2.5 Transfer Learning (TL)

TL is a machine learning technique that utilizes knowledge gained from a pre-trained model on a source task to enhance the learning process of a new target task [39]. The pre-trained model,

typically a deep neural network, is trained on a large dataset to extract relevant features from the input data.

TL can be implemented through several methods, including:

- **Freezing Pre-trained Layers:** The initial layers of the pre-trained model are kept unchanged, while only the final layers are fine-tuned on the target task data. This is suitable when the source and target tasks share similar low-level features.
- **Fine-tuning the Entire Model:** The entire pre-trained model is retrained on the target task data. This approach allows for greater adaptation to the target task but requires more data and computational resources.

TL offers several advantages, including:

- **Reduced Training Time:** Utilizing pre-trained knowledge significantly reduces the training time required for the target task.
- **Improved Performance with Limited Data:** TL is particularly beneficial when the target task has limited available data.
- **Faster Experimentation:** TL enables researchers to quickly explore new tasks by adapting existing models, accelerating the development cycle.

In the context of wildfire detection, TL is particularly relevant due to the scarcity of wildfire image datasets. By utilizing pre-trained models on large image datasets, such as ImageNet [40], researchers can extract relevant features for wildfire detection with minimal target data. This approach facilitates the development of efficient and accurate wildfire detection systems even with limited computational resources.

2.6 Visual Sensors and TinyML

TinyML, a field focused on deploying ML models on resource-constrained devices, is reshaping the capabilities of visual sensors [41, 42]. This technology enables visual sensing directly on edge devices like microcontrollers, opening up possibilities for real-time, intelligent sensing applications. By integrating advanced algorithms with embedded systems, TinyML is driving advancements in environmental monitoring, including wildfire detection and prevention.

2.6.1 Key Components and Advancements

Visual sensors empowered by TinyML combine efficient algorithms, specialized hardware, and optimized software frameworks to perform intelligent visual analysis on the edge:

- **Efficient Algorithms:** Lightweight CNN architectures, discussed in Section 2.4, are crucial for real-time visual sensing on resource-constrained devices. These models prioritize

computational efficiency while maintaining accuracy, making them suitable for edge deployment.

- **Specialized Hardware:** Microcontrollers with integrated image sensors and dedicated hardware accelerators for ML operations are being developed to meet the computational demands of visual processing on edge devices [43]. These advancements enable efficient execution of CNN models, allowing for real-time inference and analysis of visual data.
- **Optimized Software Frameworks:** Frameworks like TensorFlow Lite for Microcontrollers provide a streamlined environment for deploying and running ML models on microcontrollers [44]. These frameworks optimize model execution, memory usage, and power consumption, making visual sensing on embedded systems feasible.

2.6.2 Applications in Wildfire Detection and Prevention

There have been numerous applications of visual sensors, enhanced by TinyML algorithms, in the realm of wildfire detection and prevention. These systems offer promising solutions for early detection and proactive risk mitigation. Some of these include:

- **Smoke and Flame Detection:** TinyML-powered cameras equipped with CNN models can be strategically positioned in high-risk areas to detect the initial signs of smoke or flames [45]. This early warning system allows for rapid response and containment, potentially mitigating the impact of wildfires.
- **Thermal Imaging:** By integrating thermal imaging cameras with TinyML algorithms, visual sensors can identify hotspots and temperature anomalies, indicative of potential wildfire risk [4]. This early identification allows for targeted assessment and intervention efforts, reducing the risk of uncontrolled fires.
- **Monitoring Vegetation Health:** Visual sensors can monitor the health of vegetation by analyzing changes in color and texture [46]. This analysis helps identify areas susceptible to wildfires due to drought or disease, allowing for targeted preventative measures.

2.7 Wildfire Image Dataset

The effectiveness of ML models in wildfire detection is directly related to the quality and representativeness of the underlying datasets. The selection and curation of datasets significantly influence a model's ability to generalize to real-world scenarios, a crucial factor in the dynamic and unpredictable nature of wildfires.

- **Dataset size:** Models trained on larger datasets are exposed to a wider variety of fire scenarios, allowing the models to learn more complex patterns and improve their ability to detect wildfires in unseen data. This principle, called generalization, is well-documented

in the field of deep learning [47, 48] and is especially important in the context of wildfire detection, as wildfires can have an extensive range of shapes, colors and textures.

- **Diversity in image selection:** The diversity of images within a training dataset is crucial for the development of effective computer vision models. For wildfire detection, this diversity should encompass variations in lighting conditions (day/night, shadows, direct sunlight), weather patterns (clear skies, rain, fog, smoke), seasonal changes (lush greenery, dry foliage) and geographic locations (forests, grasslands, urban areas). Such a diverse dataset enables the model to learn the varying visual characteristics of wildfires under different real-world conditions, enhancing its accuracy and adaptability. Additionally, including images with varying resolutions and quality prepares the model to effectively process input data from diverse sources.
- **Class balance:** Real-world datasets often exhibit class imbalance, where the frequency of one class significantly outweighs others. This imbalance can hinder the performance of ML models, particularly for minority classes. In the context of wildfire detection, this imbalance can manifest in several ways, such as the under-representation of specific fire stages, wildfire types or environments (e.g., urban areas). For example, a dataset might contain numerous images of smoke, but relatively few of active flames.

To mitigate the effects of class imbalance in wildfire datasets, various techniques can be employed:

- **Oversampling:** This technique involves duplicating images from the under-represented classes to increase their frequency in the dataset, creating a more balanced distribution. Additionally, generating synthetic examples of the minority class can further improve balance in the dataset [49].
- **Undersampling:** This technique reduces the number of images from the over-represented classes to match the size of the minority class. This can be done randomly or strategically, such as selecting representative images from the majority class [50].
- **Cost-sensitive learning:** This approach assigns different misclassification costs to each class during training. By penalizing misclassifications of the minority class more heavily, the model is encouraged to focus on these rarer examples, improving its ability to detect them [51].
- **Data augmentation:** While not directly addressing class imbalance, data augmentation techniques like rotation, cropping, and brightness adjustments can generate variations of existing images, effectively increasing the diversity of the minority class and improving model generalization [52].

By addressing class imbalance through these techniques, wildfire detection models can be trained on a more representative distribution of real-world data, ultimately leading to improved performance in identifying and classifying wildfires across diverse scenarios.

- **Data Preprocessing:** Data preprocessing is essential for ensuring the quality and suitability of image datasets for wildfire detection models [53]. This process involves several key steps:
 - Standardization: Resizing and cropping images to a consistent size and aspect ratio ensures compatibility with various model architectures and optimizes computational efficiency.
 - Normalization: Scaling pixel values to a standard range mitigates the impact of varying lighting conditions, making the model less sensitive to changes in brightness and contrast.
 - Noise reduction: Applying filters such as Gaussian or median filters removes unwanted noise and artifacts that may hinder model performance.
 - Data augmentation: As stated for class balance, artificially expanding the dataset through techniques like rotation, flipping, cropping, and color adjustments improves model generalization and robustness to variations in real-world data.
 - Label verification: Ensuring the accuracy of labels associated with each image is essential, as incorrect labels can mislead the model during training.

2.7.1 Dataset Splitting for Model Evaluation

The partitioning of a dataset into training and testing sets is a fundamental step in ML model development and evaluation. This process ensures that the model is not simply memorizing the training data but can generalize to unseen data, a crucial indicator of its real-world performance.

Training Set

This subset of the data is used to train the model, allowing it to learn patterns and relationships within the data. The size of the training set typically ranges from 70% to 80% of the total dataset, depending on the size of the dataset and the complexity of the model.

Testing Set

The testing set is a separate, unseen portion of total dataset that is used to evaluate the model's performance after training. The testing set usually comprises 20% to 30% of the total dataset.

Validation Set:

In some cases, a third subset called the validation set is used to tune hyperparameters and make decisions about the model architecture during training. This is often used in conjunction with k-fold cross-validation [54].

Strategies for Dataset Splitting

The choice of dataset splitting strategy directly impacts the model's performance evaluation and ability to generalize to new data. Several strategies can be employed, each with its own advantages and considerations:

- **Random Splitting:** This is the most common and straightforward approach, where the dataset is randomly divided into training and testing sets. While simple, it can lead to imbalances in class distribution, particularly in datasets with class imbalance.
- **Stratified Splitting:** This method ensures that the class distribution in the training and testing sets is proportional to the original dataset. This is particularly important for imbalanced datasets, like those often found in wildfire detection, where the number of fire images may be significantly lower than non-fire images.
- **Time-Based Splitting:** For time-series data, where the order of observations is important, a time-based split is used. In this case, earlier data is used for training and later data for testing. This is relevant for wildfire detection, as it allows the model to be tested on more recent fire events.

Considering these approaches the following factors must be taken into account:

- **Reproducibility:** Ensuring that the splitting process is reproducible by using a fixed random seed when randomly splitting the dataset.
- **Data Leakage:** Avoiding data leakage, where information from the testing set leaks into the training set. This can lead to overly optimistic performance estimates and poor generalization.
- **Class Imbalance:** For imbalanced datasets, considering the use of stratified splitting or over-sampling/undersampling techniques to ensure a balanced representation of classes in both training and testing sets.

2.8 Edge Impulse Platform

Edge Impulse is an advanced platform designed to develop and deploy ML models on edge devices [55]. It supports multiple functionalities that make it an ideal choice for applications like wildfire detection for resource-constrained hardware. This section explores the various functionalities of the Edge Impulse platform, including the Studio, API, SDK, and EON-Compiler.

2.8.1 Studio

Edge Impulse Studio is a user-friendly interface, accessible through a browser, that facilitates the end-to-end process of developing ML models, by streamlining data acquisition, model training, and deployment.

The Studio is equipped with tools for data labeling, which is crucial for creating accurate models for image-based wildfire detection. Additionally, it provides visual insights into model performance and allows for easy debugging and optimization of algorithms.

2.8.2 API

The Edge Impulse API enables seamless integration with various data sources and deployment targets. It allows developers to interact with the platform programmatically, facilitating automated workflows and custom integrations. For wildfire detection, the API can ingest real-time data from remote sensors and cameras, enabling continuous monitoring and rapid response to potential fire events. This flexibility ensures the detection system remains up-to-date with the latest data.

2.8.3 SDK

The Edge Impulse SDK (Software Development Kit) provides a collection of libraries and tools that simplify the deployment of ML models on edge devices. It supports multiple programming languages and platforms, ensuring broad compatibility.

2.8.4 EON-Compiler

The EON-Compiler is a unique feature of Edge Impulse that optimises machine-learning models for edge devices. It reduces the memory footprint and increases the inference speed of models, making them suitable for deployment on hardware with limited resources [56].

The EON-compiler won the 2021 Edge AI and Vision Product of the Year by the Edge AI and Vision Alliance at the Embedded Vision Summit [57].

2.9 Bringing Machine Learning (ML) to the Edge

Edge computing refers to the processing of data at or near the source of data generation rather than relying on a centralized data-processing warehouse [58, 59]. Edge computing is becoming increasingly significant in applications requiring real-time decision-making and processing, such as autonomous vehicles, smart cities, and industrial automation.

In the Edge of IoT systems, small, cheap, low-power hardware devices with limited computational resources are commonly used. These hardware devices are prime candidates for running compressed ML models. The advantage of using these embedded devices is their ability to perform specific tasks efficiently without the need for significant computational power.

This development can be a positive turning point for fire detection, especially in the WUI, which is an area of particular importance in this work. By deploying these low-power, cost-effective embedded systems with compressed ML models, it becomes feasible to implement widespread, real-time fire detection networks. This can significantly enhance the ability to detect and respond to fires quickly, potentially saving lives and reducing property damage.

The specific embedded system devices used for machine learning vary depending on the application. Some commonly used devices include [60, 61]:

- **NVIDIA Jetson:** A series of embedded systems designed for autonomous devices and various embedded applications. They are known for their reliability in implementing ML tasks, especially for graphically intensive applications, but they tend to be more expensive than other options.
- **Raspberry Pi:** A series of popular embedded computers developed by the Raspberry Pi Foundation. They have a wide range of applications, including ML, and are known for their affordability.
- **Arduino Nano 33 BLE Sense:** A compact and versatile embedded system with various integrated sensors (temperature, humidity, pressure, inertial measurement unit (IMU), microphone, and gesture sensors) and Bluetooth Low Energy (BLE) connectivity for wireless communication.
- **Arduino Portenta:** A high-performance embedded system featuring an ARM Cortex-M7 and Cortex-M4 dual-core processor, making it suitable for complex ML tasks. The Portenta also supports Wi-Fi, Bluetooth, and various interfaces for extensive connectivity options.
- **ESP32:** A cost-effective embedded system with integrated Wi-Fi, Bluetooth, and a dual-core processor. While not as computationally powerful as higher-end devices, its versatility and low power consumption make it suitable for lightweight ML tasks and IoT edge computing.

Embedded computing devices typically include a microcontroller unit (MCU), which integrates a central processing unit (CPU) along with other components such as memory and peripherals. The CPU within the MCU is responsible for executing programs and handling control tasks, while some MCUs also incorporate specialized accelerators for tasks such as graphical rendering or machine learning. The choice of MCU depends on the specific requirements of the application, with more advanced MCUs providing greater computational power and capabilities for complex tasks.

Chapter 3

Data & Methodology

The major goal of this Dissertation is to evaluate the feasibility of employing compressed machine learning models in resource-constrained affordable hardware platforms in order to better support the detection of wildfire incidents, mostly considering the characteristics of the WUI. For that, a particular methodology was defined for this purpose, which will be further assessed in Chapter 5.

Building upon this primary objective, this chapter outlines the process of wildfire detection using CNNs and incorporating TL principles. Firstly, it describes the process of developing a consistent dataset from existing publicly available datasets and other sources, detailing a specific process used for balancing the dataset and the tool developed for that purpose. Secondly, it presents the specific training details and the tools employed for the model's development and deployment, along with the various evaluation metrics utilized to assess model efficacy and computation efficiency. Finally, the hardware used and the construction of the enclosure for enabling real-world testing are described. This process is represented by the methodology architecture of Figure 3.1

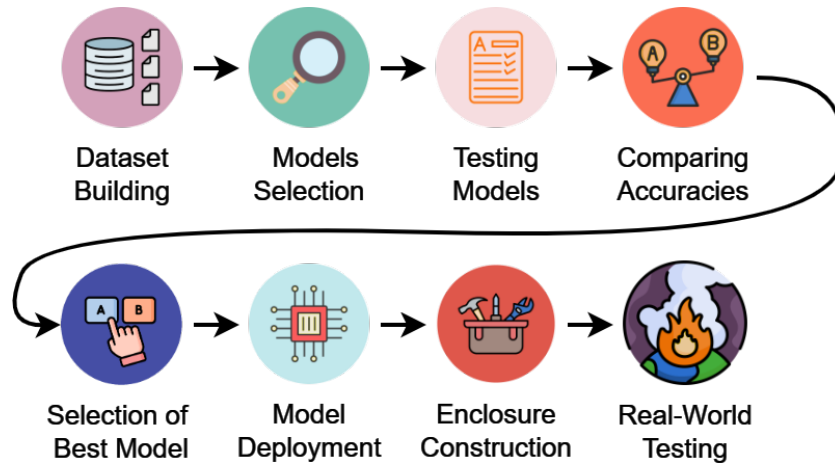


Figure 3.1: General methodology for the construction of TinyML-based wildfire detection systems.

3.1 Constructing the datasets

As with any machine learning application, the success of a CNN for wildfire detection is heavily dependent on the quality and suitability of the training data. This training data is in the form of an image dataset, and as such, various datasets were gathered in search of one that fulfilled the prerequisites outlined in Section 2.7. Despite the existence of numerous datasets referenced in the literature, they do not meet the requirements, and therefore, a new dataset was created, prioritizing factors such as diversity, class and category balance, and overall size. A total of 19 datasets were consulted, with the reference list being provided in Table A.2. Two prominent datasets, D-Fire [62, 63] and FF Dataset [64], provided the majority of images. Additionally, to increase the number of images of under-represented categories (these categories are explained in Section 3.1.4), the dataset was further supplemented with images found online and with hand-captured photographs.

3.1.1 D-Fire

The D-Fire dataset consists of over 21,000 images encompassing various fire and smoke scenarios. As illustrated in Table 3.1, the dataset includes images containing only fire, only smoke, both fire and smoke, and those devoid of fire and smoke. These images are labeled according to the categories presented in the table. Additionally, the dataset includes bounding boxes for object detection tasks. However, the current work focuses on image classification and, therefore, does not utilize these bounding boxes.

Table 3.1: D-FIRE: number of images and corresponding labels

Category	#Images
Only fire	1,164
Only smoke	5,867
Fire and smoke	4,658
None	9,838

3.1.2 FF Dataset

The FF Dataset consists of over 8,322 images encompassing various fire and smoke scenarios. A large percentage of this dataset depicts fire and smoke in indoor, urban and industrial situations that are not relevant to this work. Nevertheless, more than 400 images were taken from the FF Dataset and used in the final dataset.

3.1.3 Hand-Captured and Searched Photographs

After collecting over 2500 images from various datasets, including the D-Fire and FF datasets, and categorizing them as described in 3.1.5, it became apparent that some categories were under-represented. For example, the Day-Far-Smoke category contained only 2% of the number of

images in the Evening-Far-Smoke category. To address this imbalance, stock photo websites such as StockSnap [65], Picryl [66] and Pexels [67] were explored to find additional relevant images. The search for images was filtered by public domain photos in order to respect copyright laws. Additionally, photographs were captured manually to further improve these under-represented categories. Around 5% of the non-wildfire images of the final dataset consists of hand-captured photographs.

3.1.4 Dataset Categories

To ensure thorough representation and meet the requisites outlined in Section 2.7, particularly dataset balance, the dataset was organized into a multi-layered, tree-like Folder Structure (FS) as shown in Figure A.1. This structure utilizes five categories for image categorization, which are described in detail below:

- **First Layer (Root):** The entire collection of the dataset.
- **Second Layer:** Binary classification of images where “wildfire” represents detection (presence of fire or smoke) and “non-wildfire” represents non-detection.
- **Third Layer:** Categorization based on the time of day — day, night, and evening (evening represents both sunset and sunrise situations). This layer accounts for varying lighting conditions that significantly affect the visual characteristics of fire and smoke.
- **Fourth Layer:** Categorization based on the distance of the fire or smoke from the camera — close, medium, and far. This helps in tuning the model’s ability to detect wildfires at different scales.
- **Fifth Layer (Applicable only to ‘Wildfire’):** Further differentiation between fire and smoke. This specificity is vital for models to accurately identify wildfires in different scenarios where only smoke or fire may be present. Smoke was given priority for categorizing images where both fire and smoke were present, because, as found in preliminary testing, CNN models seemed to detect fire more easily than smoke. This helps increase the representation of smoke in the dataset, with or without fire.

This categorization not only enhances the quality of the dataset but also enables targeted evaluation of models during testing. For instance, one model might excel at detecting wildfires during the day, while another might perform better at detecting wildfires at night.

3.1.5 Image Inclusion and Dataset Cleaning

To create the final dataset, all the datasets mentioned in this chapter were carefully reviewed. Firstly, some datasets had images that were related to fire or smoke, but not fire nor smoke, such as fire-hydrants, exhaust pipes from vehicles, emergency helicopters and many others. These mislabelled images were discarded. Secondly, datasets comprised entirely of images captured

in the same location were identified, and only a random portion of these datasets were utilized. Including all images from such datasets in the final dataset could result in an over-representation of that specific location, potentially biasing models trained on this data and preventing them from generalizing well. Thirdly, some datasets contained images from news channels, these images were also discarded in order to prevent any copyright infringement issues. Finally, the datasets being reviewed were built from mostly the same sources, therefore, many images were duplicated multiple times. Using the *Duplicate Photo Cleaner* tool [68], duplicate or almost duplicate images were excluded.

The process by which images were added to the dataset was conducted iteratively as presented in Figure 3.2.

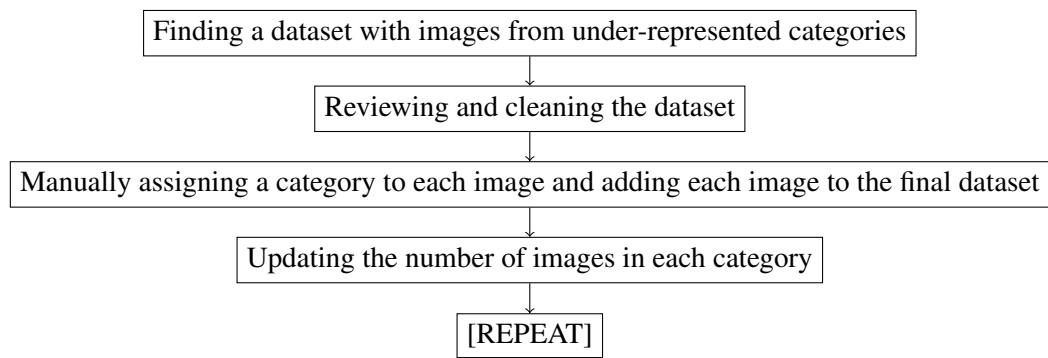


Figure 3.2: Process for adding images from existing datasets to the main dataset

Once no more significant datasets were found, the process presented in Figure 3.3 was conducted to add images searched on the web or captured with a camera:

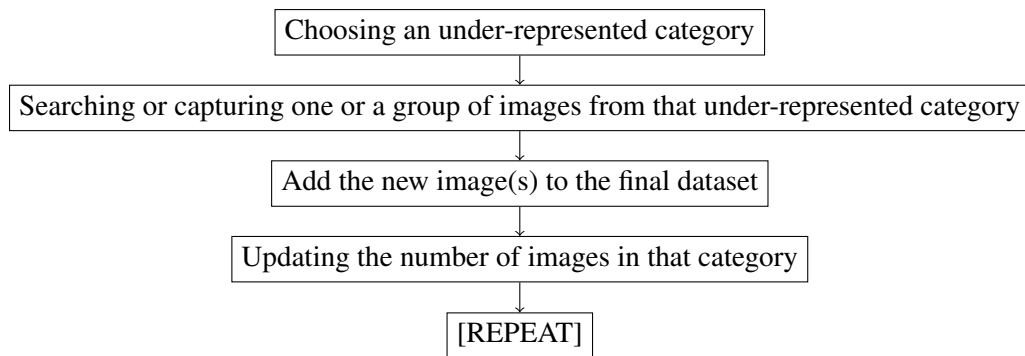


Figure 3.3: Process for adding individual images to the main dataset

3.1.6 Final Dataset

The final dataset is comprised of 2,930 images, each assigned to one of the categories described in Section 3.1.4. The number of images in each category can be observed in Figure A.1.

For better visualization of the different categories, examples for every wildfire and non-wildfire categories are presented in Figure 3.4. There is no image for the Wildfire-Close-Day-Smoke

category, as this never happens. For example, if an observer is close to a wildfire at night and can see the smoke, the flames of the fire will always be visible. Therefore, this category is not used.

A simplified flowchart representing the dataset's building process is presented in Figure 3.5.

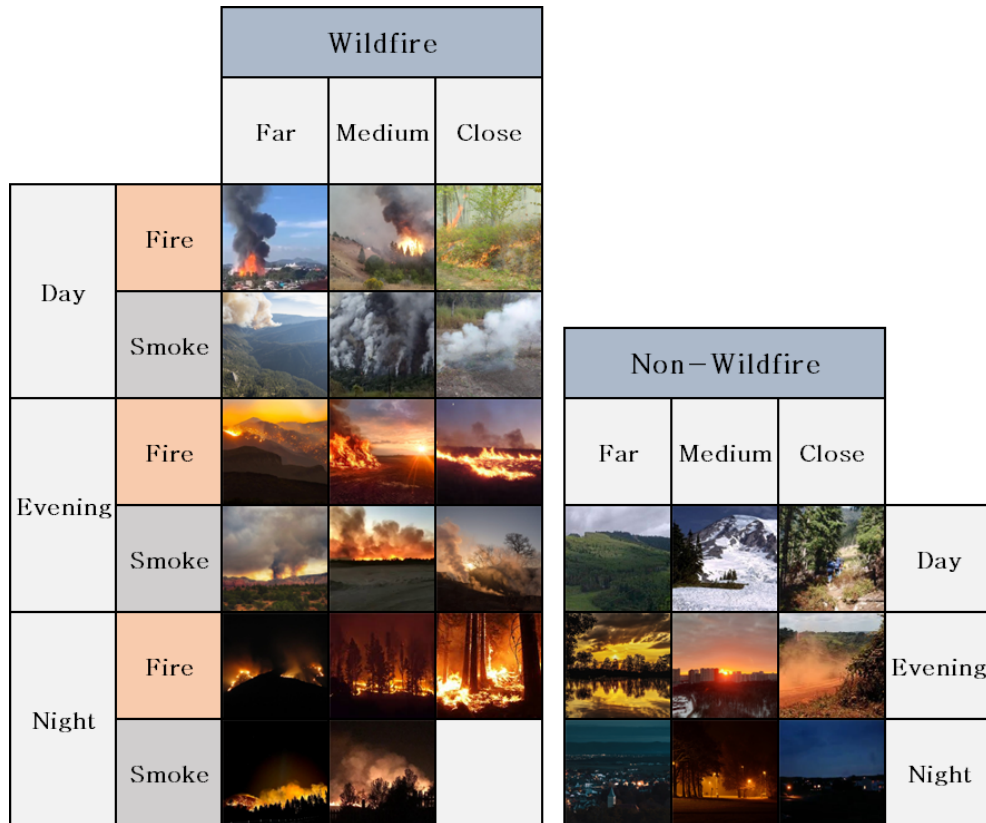


Figure 3.4: Example images of wildfire and non-wildfire per category

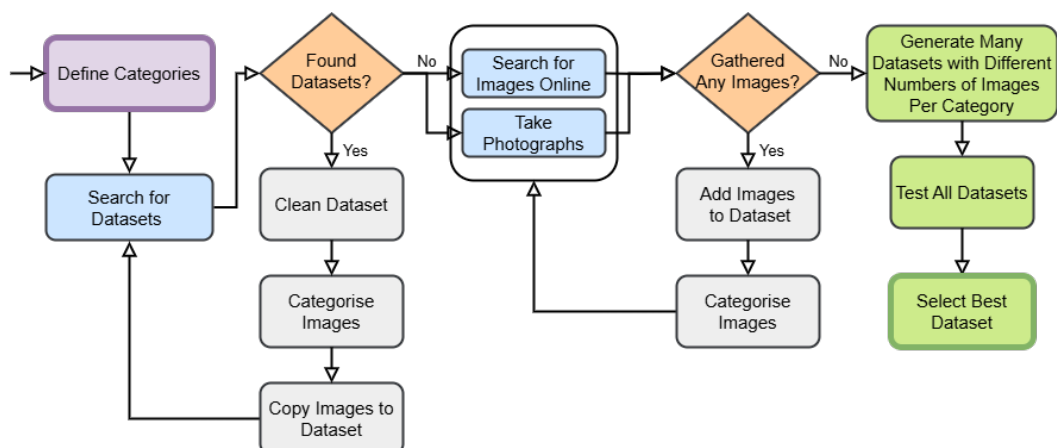


Figure 3.5: Dataset building process

3.2 Random Sample Maker (RSM)

Managing thousands of images is a complex task, and while many tools exist for creating image datasets and labeling images, there is a lack of a straightforward tool specifically designed for image classification tasks that allows users to easily generate a random subset of a dataset by extracting a fixed number of images from each category in the original dataset.

This functionality is datasets are often unbalanced, containing varying numbers of images across different categories. As discussed in Section 2.7, to train a model effectively, it is essential to have a balanced dataset where each category is equally represented.

Manually performing this task is not only time-consuming but also prone to human bias. Individuals might select images they believe will yield the best results, thereby introducing unintended bias into the dataset.

To address these challenges, *RSM* was developed using Python. Although initially designed for managing images in folders on Windows operating systems, it leverages cross-platform libraries, making it compatible with both Linux and macOS. This tool automates the process of balancing datasets and includes other relevant options for datasets used in image classification tasks. An example of this process is provided in Figure 3.6.

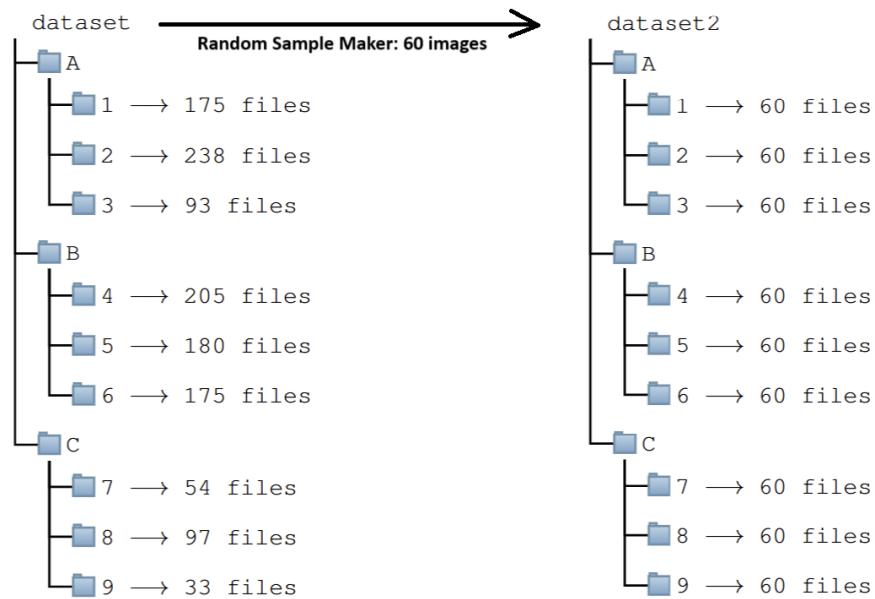


Figure 3.6: Random Sample Maker's example

3.2.1 Requirements

RSM has the following requirements to work correctly:

- **Dependencies and Libraries:** The user's system running *RSM* must have Python 3.8 or above installed, as well as the following libraries: Pillow (Fork of the Python Imaging Library by Jeffrey A. Clark) [69], openpyxl [70], tkinter [71], imagehash [72] and OpenCV [73].

- **Hierarchical Organization:** The program requires the original dataset's FS to follow a hierarchical approach with nested folders. It must have only one root folder, which can contain many subfolders. Each subfolder can also contain subfolders.
- **Unique Category:** Each folder represents a unique category for the images contained immediately inside that folder.
- **Supported image types:** `'.png'`, `'.jpg'`, `'.jpeg'`, `'.gif'`, `'.bmp'` and `'.tiff'`

Example of an image path based on Figure A.1: `wildfire/day/close/smoke/image42.jpg`, in this example, "image42.jpg" is an image of smoke taken during daylight and where the smoke is close to the camera.

It should be noted that images can be located both in the leaf folders (folders with no nested subfolders) and intermediate folders (folders with nested subfolders). For example, `1/day/far/`, `1/day/far/fire` and `1/day/far/smoke` could all contain images immediately inside them. In this example, the reason could be that there might be images where there is only fire, only smoke or both fire and smoke.

3.2.2 Usage and Functionality

RSM is a tool for creating a subset of a dataset structured as described in Section 3.2.1. It allows the creation of a new dataset from an existing dataset by copying a fixed number of images from each folder in the original dataset into a new dataset. The generated FS of the new dataset is the same as that of the original dataset.

The images in the new dataset are renamed according to the original FS, in case the user does not want to utilize the FS and needs to know the category of an image from its name. For example, if an image named "image.png" is originally inside the following FS: `RootFolder` → `SubFolder1` → `SubFolder2`, and it's the forth image copied from that folder to the new FS, the name of that image inside the new FS would be:

`"4_RootFolder_SubFolder1_SubFolder2_image.png"`

This tool functions through the User Interface (UI) presented in Figure 3.7 or through a Command Line Interface (CLI), an example of the CLI usage is provided in Bash Example 3.1.

The following options can be selected and defined depending on the user's intended functionality:

- **"Delete copied images from original folder structure":** If this option is selected, the images taken from the original dataset are deleted from the original FS.
(CLI: `-d` or `--delete`)
- **"Keep existing images in destination folder structure":** If this option is selected and the new dataset is being created in an already existent FS containing images, these images are not deleted when generating the new dataset in that destination.

(CLI: -k or --keep)

- **"Ignore duplicate images in original folder"**: If this option is selected, the program will ignore duplicate images in the original folder and only use different images to constitute the new dataset. It utilises a technique called *perceptual hashing* through the 'imagehash' library.

This feature is currently not optimised, and being active makes the program at least 10 times slower, depending on the number of images in the original dataset. If the user intends to use *RSM* to manage hundreds of thousands of images and intends to check all of them for duplicates, other tools such as the *Duplicate Photo Cleaner* would be better suited to this specific functionality. After removing duplicates in the dataset, the user could run *RSM* with this option deactivated to utilize the remaining features.

(CLI: -id or --ignoreduplicates)

- **"Ignore folder structure in destination"**: If this option is selected, all the images used for the new dataset will be placed directly under the Root Folder in the destination.

(CLI: -ifs or --ignorefolderstructure)

- **"Augment images in under-represented categories"**: If this option is selected and some categories of the original dataset have fewer images than required, those images will be duplicated until the necessary number of images is satisfied, flipped on the vertical axis (the flipping only occurs 50% of the time, using a random function) and randomly rotated between -5 and 5 degrees, this rotation forces the image to be scaled and cropped to fill the expected image size. These operations use the Pillow and the OpenCV libraries.

(CLI: -md or --makeduplicates)

- **"Select output image type"**: This field allows the user to select PNG, JPEG, BMP, GIF or TIFF as the output format of the images. This conversion uses the Pillow library, it firstly converts every image to RGB and then saves that image in the specified format.

(CLI: -of or --outputformat followed by image type, eg. png)

- **"Image Size (WIDTHxHEIGHT)"**: This field expects two numbers separated by an 'x'. The values inserted define the size of the images copied to the new dataset. This new size is obtained by resizing and cropping the original images using the Pillow library's Lanczos Resampling.

(CLI: -s or --size followed by size format, eg. 96x96)

- **"Output Excel File Name"**: This field expects a filename followed by the '.xls' extension. After the new dataset has been created, an Excel file with that name will be created in the new dataset's Root Folder. This file includes a representation of this new dataset's FS and the number of images immediately inside each folder. An example of this output file representing the same FS shown in Figure A.1 is presented in Table A.3.

(CLI: -o or --output followed by filename, eg. output.xlsx)

- **"Select Root Folder"** and **"Select Destination Folder"**: These buttons allow the user to select the Root Folder where the original dataset is located and the Destination Folder where the new dataset is to be located. The Destination Folder should be the Root Folder of the new dataset. The FS does not need to exist as it is generated automatically by the program.

(CLI: -rd or --rootdirectory followed by computer path, eg. C:\User\path\dataset)

(CLI: -dd or --destdirectory followed by computer path, eg. C:\User\path\sampled_dataset)

- **"Number of Samples"**: This field expects the number of images to be copied from each folder in the original dataset's FS to each corresponding folder in the new dataset's FS.

(CLI: -ns or --numbersamples followed by the number of samples to make, eg. 20)

- **"Make Random Sample"**: This button runs the program.
- **Running in CLI mode**: This feature for running the program in the CLI only requires the following parameter written in the CLI when running the program:

(CLI: --cli)

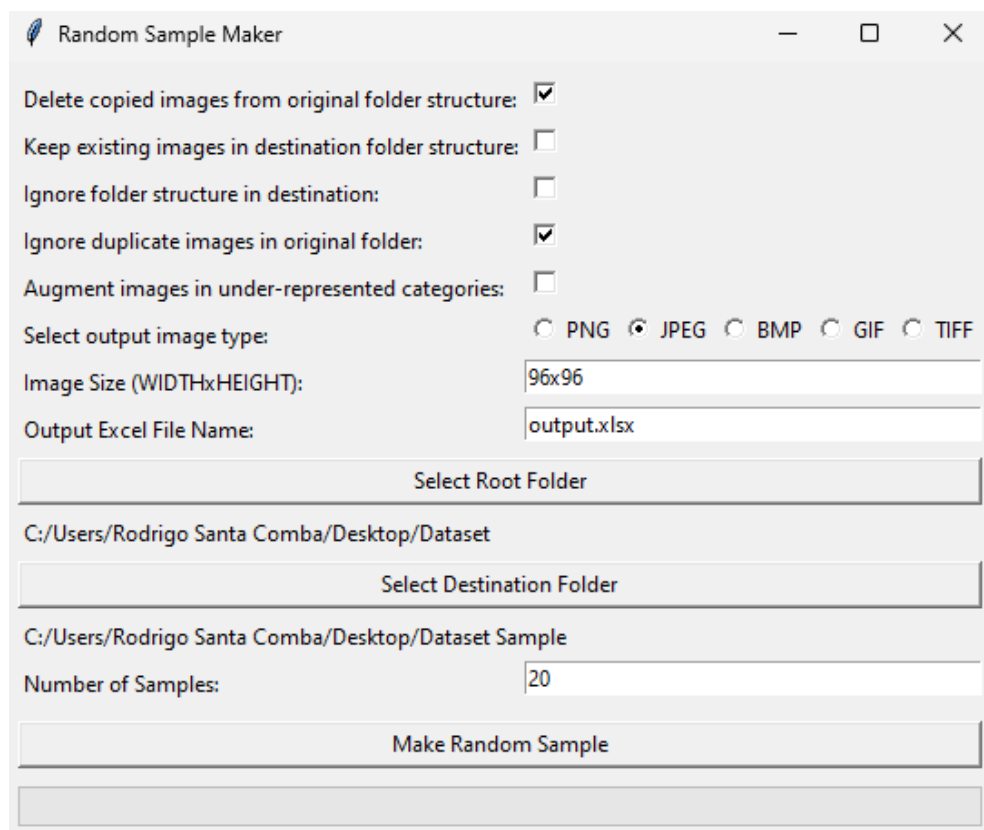


Figure 3.7: *Random Sample Maker's* UI

RSM has several error-checking mechanisms for incorrectly filled fields or chosen folders in the UI. These include displaying a warning if the user selects an invalid root or destination folder, selects the same folder for both root and destination folders or writes an invalid size format. Most importantly, it displays an alert if the "Delete copied images from root folder" is selected when the "Make Random Sample" button is pressed. This alert is presented in Fig 3.8.

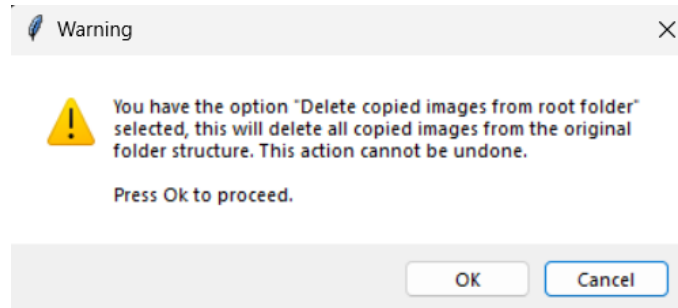


Figure 3.8: *Random Sample Maker*'s UI warning

CLI Usage: Running *RSM* on the CLI allows for personalized scripting. All the options displayed in the UI can also be selected using the CLI through the use of the codes presented previously in this section. Because of this, it is possible to create scripts that run the program multiple times sequentially with different options, if many datasets need to be generated. Following is an example of a CLI command to run *RSM*:

```
1 python randomnessmaker.py --cli --outputformat PNG --size 96x96 --
  output output.xlsx --rootdirectory C:/Users/Username/Desktop/
  Dataset --destdirectory C:/Users/Username/Desktop/Sampled_Dataset
  --numbersamples 10
```

3.2.3 Downloading and Running *Random Sample Maker*

To run the program, it is necessary to follow the requirements presented in Section 3.2.1, open the Interface, fill in all the fields and options described in Section 3.2.2 and select "Make Random Sample" in the UI, or run it directly through the CLI.

The program's source code can be downloaded as a .py file through the author's GitHub [74], where more information is also detailed.

3.3 Testing and Training Datasets

For training and evaluating an image classification model, it is necessary to divide the available data into two distinct datasets: a training dataset and a testing dataset. This division, where none of

the images are shared between both datasets, is essential to ensure the model’s ability to generalize to new, unseen data, in order to evaluate its real-world performance.

To ensure that no images are shared between the training and testing datasets, the final dataset should be carefully split, typically allocating 70-80% of the data to the training set and the remaining 20-30% to the testing set. Although this is the most common practice in the literature, for consistency between the testing of different models on datasets with different sizes, a fixed size of the testing dataset was defined. Section 3.3.1 will describe this in more detail. Additionally, randomizing the images selected during the split ensures that the images in both datasets represent the overall data distribution, further enhancing the reliability of the model trained and tested on these datasets.

3.3.1 Testing Dataset

The testing dataset is used to evaluate the model’s performance. It must consist of images that the model has never encountered during training. This separation is critical because it allows for an unbiased assessment of the model’s generalization ability. By testing the model on unseen images, we can determine how well it performs in real-world scenarios where it encounters new and varied scenarios.

The *RSM* tool described in Section 3.2 was used to generate the testing dataset. It extracted a set number of random images from each category from the final dataset described in Section 3.1.5 into the testing dataset. This process is exemplified in the form of a diagram in Figure 3.9. The images extracted were deleted from the final dataset, so that when the final dataset is used to generate the training dataset, none of the images are shared by both the training and the testing datasets.

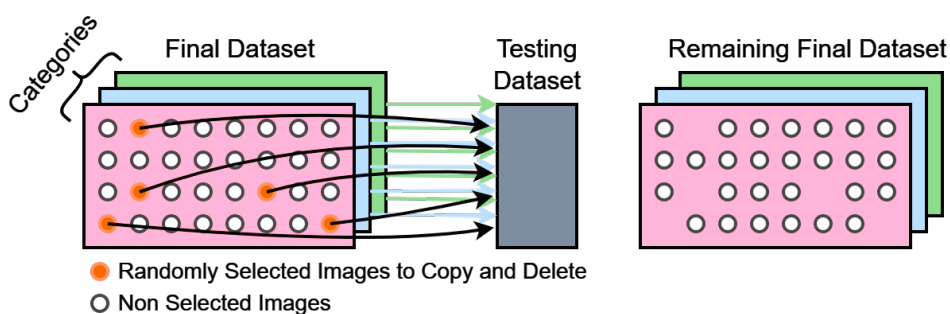


Figure 3.9: Generating the testing dataset.

The testing dataset needs to be made prior to the training dataset to guarantee a balance in the number of images per category in the testing dataset. This will not happen if the training dataset is generated prior to the testing dataset and all or almost all of the images in a particular category are used for the training dataset, leaving no images left from that category for the testing dataset.

Additionally, the choice of the number of images per category is extremely important. A very high number means the training dataset may not have enough images in a certain category, and

a very low number means the accuracy obtained by testing a model on that dataset might not be representative of the true accuracy of the model.

Ten was chosen as the number of images per category to be used in the testing dataset according to the results obtained in Section 5.1.

3.3.2 Training Dataset

The training dataset consists of a large set of labeled images that the model uses to learn and identify patterns and features relevant to the task, such as wildfire detection in this work. During training, the model adjusts its internal parameters to minimize errors in predicting the labels of these training images.

Although the remaining final dataset, obtained after extracting the images used for testing, could be used as is to train an image classification model, the imbalance in category representation might not yield the best results.

Given these considerations, the following approaches could be used to obtain the training dataset: where,

k = number of images to be extracted per category

i = a specific category

j = number of existing images in a specific category i

n = number of categories

N = total number of images extracted

1. An approach with balanced categories:

Extracting the testing dataset from the final dataset and using the *RSM* tool to extract k images per category from the resulting final dataset. $N_1 = \sum_i \max(k, j_i)$

2. An approach with balanced categories and slight augmentation:

Extracting the testing dataset from the final dataset and using the *RSM* tool to extract k images per category from the resulting final dataset, using the option "Augment images for under-represented classes". $N_2 = k \times n$

3. An approach without considering balance for training:

Extracting the testing dataset from the final dataset and extracting N_1 random images from the resulting final dataset. $N_3 = N_1$

The tests and results obtained for the different approaches with varying values of k are discussed in Section 5.1. These tests help to determine what approach to use and how many images should be considered when generating the training dataset.

The approach taken was "2. An approach with balanced categories and slight augmentation" according to the results obtained in Section 5.1.4. This approach is exemplified in the diagram presented in Figure 3.10.

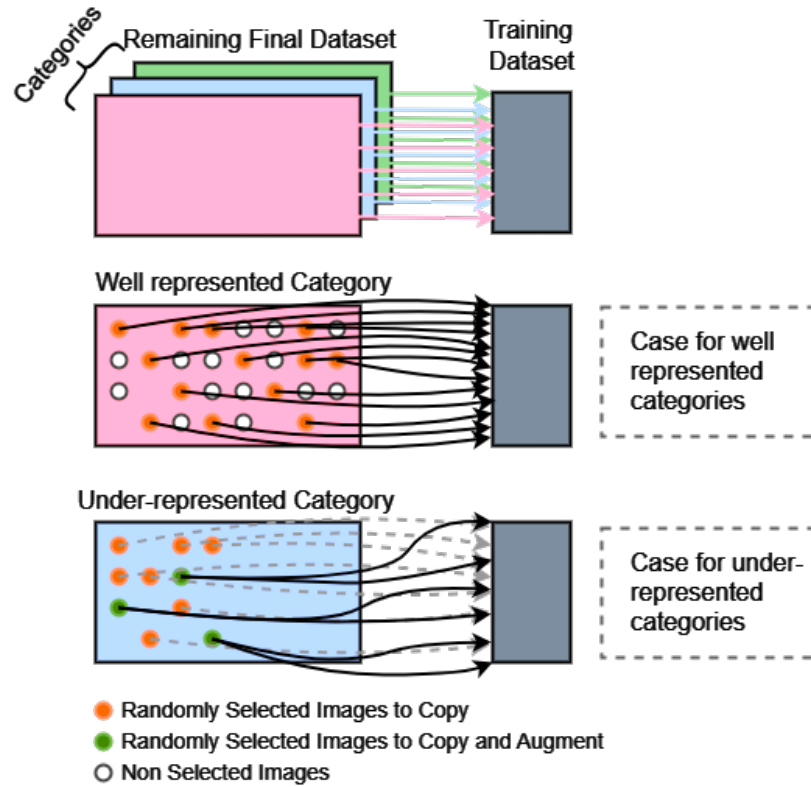


Figure 3.10: Generating training dataset

3.4 Training, Testing and Deploying CNN Models on the Edge Impulse Studio

This section describes how the models used in this work were trained to detect wildfires and how they were subsequently tested and evaluated. This includes the training, the testing, the evaluation metrics used to compare different models and approaches, and deployment of the model on the chosen hardware, the Arduino Nano 33 BLE Sense, described in Section 4.1. All these steps were executed on the Edge Impulse Studio (Section 2.8.1), as it allows for fast training, testing and deployment of the model.

3.4.1 Uploading the Dataset

The first step in the process was to upload the testing dataset (Section 3.3.1) and training dataset (Section 3.3.2) to Edge Impulse Studio. The datasets consists of labeled images, categorized into wildfire and non-wildfire classes.

Edge Impulse Studio facilitated this process by offering intuitive upload tools and storage for the datasets (Fig. 3.11), ensuring that they were ready for subsequent training and testing phases. Additionally, it provided options for the images to be resized to a specific size and their pixel values normalized.

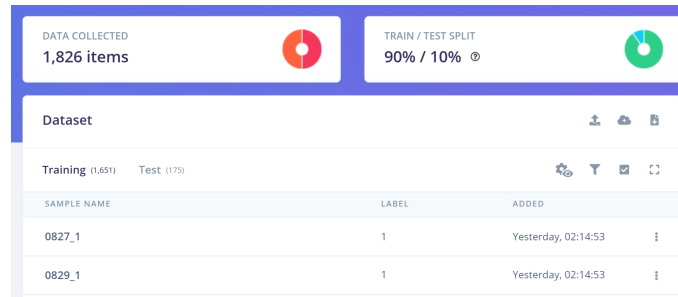


Figure 3.11: Edge Impulse: Data Acquisition

3.4.2 Training

Model training was conducted within the Edge Impulse Studio platform. To expedite the training process and enhance model performance, TL was employed. This technique leverages pre-trained models originally trained on the ImageNet dataset [40] by freezing their initial layers and retraining only the final layers with the wildfire dataset.

Hyperparameters such as learning rate, batch size, and epochs were configured using the platform's Expert Mode (Fig.3.12). This level of control was necessary to fine-tune the model architecture and training process.

Neural network architecture Save

```

1
2 import math, requests
3 from pathlib import Path
4 import tensorflow as tf
5 from tensorflow.keras import Model
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import (
8     Dense, InputLayer, Dropout, Conv1D, Flatten, Reshape, MaxPooling1D,
9     BatchNormalization,
10     Conv2D, GlobalMaxPooling2D, Lambda, GlobalAveragePooling2D)
11 from tensorflow.keras.optimizers.legacy import Adam, Adadelta
12 from tensorflow.keras.losses import categorical_crossentropy
13
14 sys.path.append('./resources/libraries')
15 import ei_tensorflow.training
16
17 WEIGHTS_PATH = './transfer-learning-weights/keras
18     /mobilenet_v2_weights_tf_dim_ordering_tf_kernels_0.35_96.h5'
19
20 # Download the model weights
21 root_url = 'https://cdn.edgeimpulse.com/'
22 p = Path(WEIGHTS_PATH)
23 if not p.exists():
24     print(f"Pretrained weights {WEIGHTS_PATH} unavailable; downloading...")
25     if not p.parent.exists():
26         p.parent.mkdir(parents=True)
27     weights_data = requests.get(root_url + WEIGHTS_PATH[2:]).content
28     with open(WEIGHTS_PATH, 'wb') as f:
29         f.write(weights_data)
30     print(f"Pretrained weights {WEIGHTS_PATH} unavailable; downloading OK")
31     print("")
32 INPUT_SHAPE = (96, 96, 3)

```

Figure 3.12: Edge Impulse: Training Expert Mode

The training process utilized the platform's built-in tools to provide real-time monitoring of metrics like accuracy and loss. This enabled close observation of the model's performance, allowing for precise hyperparameter tuning. By paying close attention to the validation accuracy and loss, it was possible to reduce the time taken to tune the hyperparameters in order to avoid issues such as overfitting, underfitting, and non-linear learning patterns.

After the initial TL phase, the full model underwent fine-tuning to further enhance performance. This involved training the full model with a very small learning rate, which can also be adjusted in the "Expert Mode".

3.4.3 Testing

Testing was also executed within Edge Impulse Studio. The testing dataset, composed of unseen images, was used to evaluate the model.

The platform facilitated the testing process by providing immediate feedback on model performance. Metrics such as global accuracy, AUC-ROC, precision, recall, and F1-score were automatically calculated.

Some additional considerations include the usage of a threshold of 0.6 for classification. By setting this threshold, only classifications where the model's confidence exceeds 60% are considered valid, effectively filtering out uncertain or random guesses.

Furthermore, all test results presented in this work were obtained using a quantized model, using higher precision floating-point numbers (float32) is unsuitable for the hardware constraints of the system, thus, quantization was applied, also within Edge Impulse, converting the models parameters to a lower precision integer representation (int8).

3.4.4 Evaluation Metrics

A comprehensive evaluation of the CNN models was conducted using Edge Impulse Studio. The primary focus was on the following metrics:

Global Accuracy: This metric quantifies the overall proportion of correctly classified images as either containing or not containing wildfire activity. In the context of wildfire detection, it provides a general assessment of the model's ability to distinguish between images with and without a wildfire present.

True Positive Accuracy: This metric calculates the proportion of accurately identified wildfire instances within the dataset. For wildfire detection, it is crucial to prioritize the correct identification of true positives, as false negatives can lead to delayed response efforts.

Area Under the ROC Curve (AUC-ROC): This metric assesses the model's capacity to discriminate between wildfire and non-wildfire images across varying classification thresholds.

The selection of these metrics was informed by the challenges inherent in wildfire image classification. Accurate and timely detection of wildfires is crucial for effective response and prevention. Global accuracy provides a general assessment of the model's performance, while true positive accuracy directly addresses the core objective of identifying wildfire occurrences. The AUC-ROC offers a nuanced evaluation of the model's discriminative capability, essential for robust wildfire detection across diverse image conditions, including varying fire intensities, smoke levels, and environmental factors.

While additional metrics such as precision, recall, F1-score, and confusion matrices can provide supplementary insights, the chosen metrics were deemed more directly aligned with the primary objective of this study: accurate wildfire detection in images.

3.4.5 Deploying the CNN models

Once the CNN models were trained and optimized, the next step involved deploying them to an embedded platform for real-time wildfire detection. Edge Impulse was utilized as the development platform for this purpose.

Edge Impulse offers several options for compiling and deploying machine learning models to embedded devices. For this project, the following approaches were considered:

TFLite: This open-source framework provides a lightweight interpreter for running TensorFlow models on various platforms.

EON Compiler: Edge Impulse's proprietary compiler optimizes neural networks for embedded devices, resulting in smaller code size and RAM usage.

EON Compiler RAM Optimized: A variant of the EON Compiler specifically designed to minimize memory usage, suitable for devices with limited RAM.

The Arduino Nano 33 BLE Sense was employed as the target hardware for this work (Section 4.1). As such, the deployment option must be compatible with the board. Consequently, the trained CNN models were exported from Edge Impulse as Arduino libraries, using the EON Compiler RAM Optimized to comply with the maximum RAM usage, 200KB as defined in Section 4.1. Exporting the models as Arduino libraries allowed for seamless integration into the Arduino IDE development environment. The libraries were imported into the Arduino IDE with the necessary code to interface with the OV7675 camera and execute the model inference. The resulting application was then deployed to the Arduino Nano 33 BLE Sense board.

The board could then be turned on while connected to a computer, and for each captured image, the wildfire and non-wildfire percentages could be observed using the Arduino IDE's Serial Monitor (Fig. 3.13).

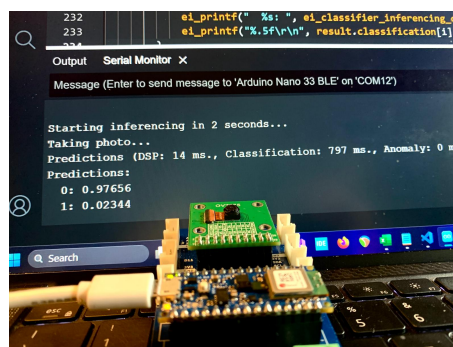


Figure 3.13: Arduino IDE's serial monitor displaying class probabilities directly from the board

3.5 Choice of Models and Training Dataset Image Sizes

Edge Impulse has many available models for image classification and object detection tasks. However, considering the hardware constraints defined in Section 4.1, the only models that could possibly fit in the Arduino Nano 33 BLE Sense are MobileNetV1 and MobileNetV2. More complex models commonly used for these kinds of tasks such as YOLO [75, 76], do not meet the requirements in terms of memory usage and computational complexity.

MobileNet utilizes an alpha parameter that changes the computational and memory needs of the model. The different values of alpha need to be tested along with all possible image sizes. The objective is to find a model that fits the hardware constraints and has the best possible combination of global accuracy, wildfire accuracy (true positives), and area under ROC curve.

The models were tested according to different configurations of model type, alpha value and image size on Edge Impulse:

- Available values of alpha for MobileNetV1: 0.05, 0.1, 0.25, 0.5, 0.75
- Available values of alpha for MobileNetV2: 0.1, 0.35, 0.75, 1
- Available image sizes: 32x32, 96x96, 128x128, 160x160...

The tests were done in ascending order of alpha and image size, as described in Figure 3.14. This process was chosen based on the results of Section 5.2, where an increase in either alpha value or image size increases the inferencing time, the peak RAM usage and the flash usage.

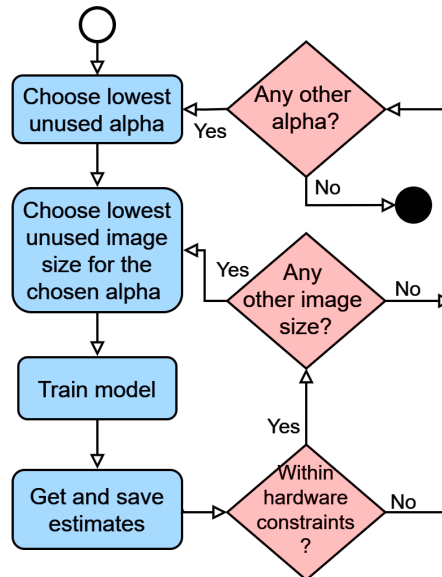


Figure 3.14: On-Device estimates testing process

The models selected for further testing and development were those with higher alpha and image size, and that complied with the hardware constraints of the Arduino Nano 33 BLE Sense.

The choice for a higher alpha and image size is based on the expectation that accuracy increases with the increase of these parameters [33].

Section 5.2.2 describes the selection of the best models that comply with the hardware constraints. The results obtained are presented in Table A.1.

The models chosen for further testing were:

- MobileNetV1 with 0.25 as the value for alpha using images of 128 by 128 pixels compiled using the EON Compiler RAM Optimized.
- MobileNetV2 with 0.1 as the value for alpha using images of 96 by 96 pixels compiled with the EON Compiler RAM Optimized.
- MobileNetV2 with 0.35 as the value for alpha using images of 32 by 32 pixels compiled using the TFLite model directly.

3.5.1 Comparisons and Choice of Model and Image Size

To determine the best model and image size for the application, performance metrics of different configurations of MobileNetV1 and MobileNetV2 models were evaluated and compared as described in Section 5.2.2. The primary metrics considered were global accuracy, wildfire accuracy, non-wildfire accuracy and AUC-ROC. Additionally, the inferencing time was also considered.

Considering the evaluations and the comparative analysis presented in Section 5.2.2, and given the importance of accurately detecting wildfires along with the need for efficient inferencing speed to reduce battery usage, the MobileNetV2 with a 96x96 image size, alpha of 0.1 and 4 neurons in the final dense layer prior to the classification layer was chosen for further optimization and deployment. The methodology for this further optimization is described in Section 3.6

3.6 Final Model Optimization and Tunning

To enhance the performance of the selected model (Section 3.5.1), MobileNetV2 with an input image size of 96x96 and 4 neurons in the final dense layer prior to the classification layer, several optimization and tuning steps were carried out. These steps included adjusting the dropout rate, incorporating batch normalization and tuning the learning rate. It is important to notice that given the target hardware platforms are resource constrained in visual sensing, processing, and memory and storage capacity, this model configuration is adequate.

Dropout Rate Adjustment

The first optimization step involved testing various dropout rates to mitigate potential overfitting and observe the impact on the model's performance. The tested dropout rates were 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, and 0.7. The performance metrics analysed from these tests were global accuracy, non-wildfire accuracy, wildfire accuracy, and the area under the ROC curve.

These tests are presented in Section 5.3.

Adding Batch Normalization

After determining the optimal dropout rate, batch normalization was introduced between the dense layer and the dropout layer to examine its effect on model performance. The model performance was analyzed for all the dropout rates tested previously, as presented in Section 5.3.

Learning Rate Tuning

Finally, once the best model was selected from the all the previous tests, the learning rate was tuned for that particular configuration. An inappropriate learning rate can lead to slow convergence or even divergence of the model. A series of experiments were conducted to find the optimal learning rate, which involved testing multiple values and selecting the one that resulted in the best performance metrics.

These tests are described in Section 5.3.

Final Model Configuration

According to the results obtained in Section 5.3 the final model configuration was MobileNetV2 with a 96x96 image size, 4 neurons in the final dense layer prior to the activation layer, using batch normalization and a dropout rate of 0.3. The model was trained with the hyperparameters presented in Table 3.2. The final performance metrics are presented in Table 3.3. Where:

GA: Global Accuracy (%); TP: True Positive Rate (%); FP: False Positive Rate (%);
 TN: True Negative Rate (%); FN: False Negative Rate (%);
 AUC: Area Under the ROC Curve (%) WAP: Weighted Average Precision;
 WAR: Weighted Average Recall; F1: Weighted Average F1 score;
 IT: Inferencing Time; PRU: Peak RAM Usage; FU: Flash Usage;

Table 3.2: Final model's training hyperparameters

Hyperparameters	
Training Cycles	50
Learning Rate	0.01
Validation Size	0.2
Batch Size	32

Table 3.3: Final model's performance metrics

GA	TP	FP	TN	FN	AUC	WAP	WAR	F1	IT	PRU	FU
88.6	94.1	10.0	88.9	4.7	0.93	0.93	0.93	0.93	992ms	169.5K	223.9K

3.7 Model Comparison with the Literature

To validate the proposed model's real-world applicability and generalization ability, its performance was assessed on multiple datasets and compared against models reported in the existing literature.

The selection of appropriate datasets is challenging, as most publications do not share them publicly. Three datasets were identified for this evaluation: Corsican Fire Database [77], DeepFire [78] and BowFire [79].

It should be noted that BoWFire is not focused wildfires, but fires in general, containing mostly images of urban and indoor scenarios that the model developed in this work was not trained on. Regardless, testing the final model on this dataset could prove useful to evaluate the generalization ability of the model.

Moreover, it was noticed that two of the images in the BoWFire dataset contained only smoke from a fire event, and as the BoWFire is a fire dataset, these images were labeled as non-fire. To align with the focus of this study on detecting both fire and smoke occurrences, these images were relabeled as fire.

Additionally, some images in BoWFire and in DFire were also present in the training dataset for the final model developed. To prevent any data leakage, these images were removed from the testing dataset, only 1 in the case of the BoWFire and 64 in the case of the DFire. In order to maintain a 50/50 balance in the number of wildfire and non-wildfire images in the DFire testing dataset, some images from the non-wildfire class were deleted resulting in 147 images from each class.

The model was then tested on these datasets, the results obtained are presented and discussed in Section 5.4.

Chapter 4

Hardware, Enclosure Build and Assembly Process

This section examines the choice of hardware for the wildfire detection system, the process of designing a suitable enclosure to protect the hardware components in various environmental conditions and the assembly of the final system.

4.1 Choosing the hardware platform for TinyML-based visual sensing

Selecting the appropriate hardware for a TinyML-based visual sensing system is essential to ensure reliable and efficient wildfire detection. The hardware platform must balance processing power, energy efficiency, and environmental resilience. When deploying CNN-based wildfire detection systems on embedded devices, several hardware factors demand careful consideration:

- **Processing Units:** The choice of MCU, microprocessor (MPU), or specialized accelerator depends on the complexity of the chosen CNN architecture, the required inference speed, and the overall power budget.
- **Memory:** Sufficient RAM and flash memory are needed to store the CNN model, intermediate feature maps, and input/output image data. Model compression techniques are often employed to reduce the memory footprint.
- **Image Sensors:** High-resolution cameras are desirable for capturing subtle smoke and flame patterns, but they also generate large amounts of data that the system must process efficiently.
- **Environmental Ruggedness:** Wildfire detection systems are deployed in harsh outdoor conditions. Components and enclosures must be able to withstand extreme temperatures, dust, humidity, and potentially corrosive elements.

- **Connectivity:** Reliable communication is essential for transmitting alerts and data. Options like cellular, LoRaWAN, or satellite links offer different trade-offs regarding range, bandwidth, and power consumption.
- **Power Source:** Depending on the deployment scenario, wildfire detection systems may be powered by batteries, solar panels, or a combination of both. Power optimization is essential for extending operational life in remote areas.

Based on these factors the Arduino Nano 33 BLE Sense board was chosen for this project. To the authors knowledge, it has not yet been used in the literature for wildfire detection using images. However, it has been proven effective in other applications, including image based ML for UAVs [80].

The board is powered by the Nordic nRF52840 microcontroller, featuring a 32-bit ARM Cortex-M4 CPU running at 64 MHz. This processor provides a significant performance boost over earlier Arduino models and includes 1 MB of flash memory and 256 KB of SRAM, allowing for more complex programs and data handling.

The Arduino Nano 33 BLE Sense is equipped with a comprehensive set of onboard sensors:

- **9-axis IMU:** Combines an accelerometer, gyroscope, and magnetometer, useful for motion tracking and orientation detection.
- **Environmental Sensors:** Includes temperature, humidity, pressure, and a microphone for sound detection, enabling a wide range of environmental monitoring applications.
- **Color and Gesture Sensor:** Allows for basic color detection and gesture recognition, which can be used in user interface applications.

As for connectivity, the board includes Bluetooth Low Energy (BLE) capabilities, facilitating wireless communication for IoT applications. This BLE functionality is powered by the integrated nRF52840 microcontroller, which supports BLE 5.0, offering improved speed and range over previous Bluetooth standards.

Finally, the Arduino Nano 33 BLE Sense is compatible with the Edge Impulse platform, simplifying model deployment (Section 3.4.5).

It is important to notice that the Arduino Nano 33 BLE Sense has 256 KB of SRAM. To allow for additional overhead and functionalities, the maximum SRAM usage made available for image capture and processing and the wildfire detection algorithm was 200KB. If a model uses more than 200KB of the SRAM, it is considered not to be compliant with the hardware constraints.

4.2 Arduino Tiny Machine Learning Kit

The hardware chosen for this project was the Arduino Tiny Machine Learning Kit. It consists of an Arduino Nano 33 BLE Sense board, an OV7675 camera board and a shield that allows for easy integration between these components and the power supply. This kit is presented in Figure 4.1.

The hardware's affordability, around 50€ for the kit, allows for deploying this system on multiple devices, enhancing scalability. Additionally, having a small form factor ensures transportation and deployment ease and minimizes the physical footprint in natural environments.



Figure 4.1: Arduino Tiny Machine Learning Kit [81]

4.3 Enclosure build using SolidWorks

To assess model performance under real-world conditions, the system must be shielded from environmental factors such as rain, sunlight, dust, and wildlife. Therefore, an enclosure is necessary to protect the system's circuitry.

Building this enclosure needs to consider the placement of the following components and their respective size and some additional requirements:

Components

- Arduino's Tiny Machine Learning Shield connected to the Arduino Nano 33 BLE Sense and the OV7675 camera.
- Batteries for powering the shield.
- A button for turning the shield on and off.
- Wires to connect the shield to the button and the button to the batteries.
- An enclosure to protect the circuitry.

Requirements

- An opening on the enclosure for the camera.
- Water-resistant to rain.
- A method for attaching the enclosure to a physical support.

Based on these requirements, an enclosure was designed in SolidWorks. It consists of a base, a lid that can be screwed to the base, an opening for the camera where a lens is to be placed for protection and a hole meant for an insert to allow for the enclosure to be mounted on a physical structure.

4.3.1 Base

The base guards all the electrical components. It was designed as a container opened at the top, with screws for mounting the lid. For extra protection, following the design of many junction boxes used in outdoor scenarios, a horizontal edge was added around the top of the base, this prevents water from running along the sides into the enclosure and increases the strength of the corners between the lid and the base. Additionally, a vertical edge was added at the top of the base (that fits tightly to a similar edge at the lid), this further prevents water from easily leaking into the enclosure.

Regarding the components and requirements stated previously in this Section, it has a center opening for the batteries, four elevated holed cylinders for mounting the Arduino's shield over the batteries, space for a button, and a hole for a threaded insert for a $1/4'' - 20 \times 1/2$ screw at the bottom.

The final base design can be seen in Figure 4.2

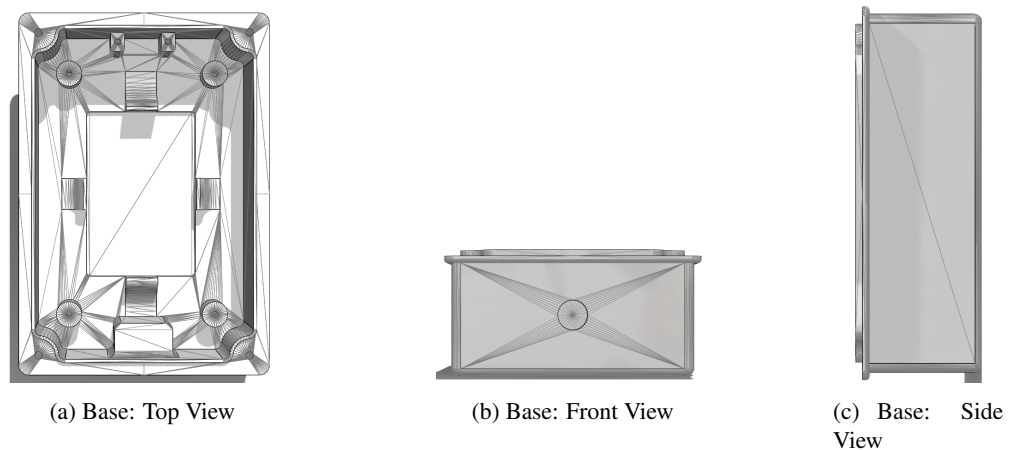


Figure 4.2: Base design in SolidWorks

4.3.2 Lid

The lid closes the enclosure, as such, it has a hole in each corner so it can be screwed to the base. For extra protection, it has a vertical edge that fits tightly with the vertical edge at the top of the base. This prevents water from leaking into the enclosure.

Additionally, the lid contains a hole for the camera and a protrusion for mounting a lens over it. This is necessary to prevent water and dust from entering the enclosure. It also contains a small visor above the camera to deflect rainwater, thereby mitigating image distortion on the camera, and serving as extra protection.

The final lid design can be seen in Figure 4.3.

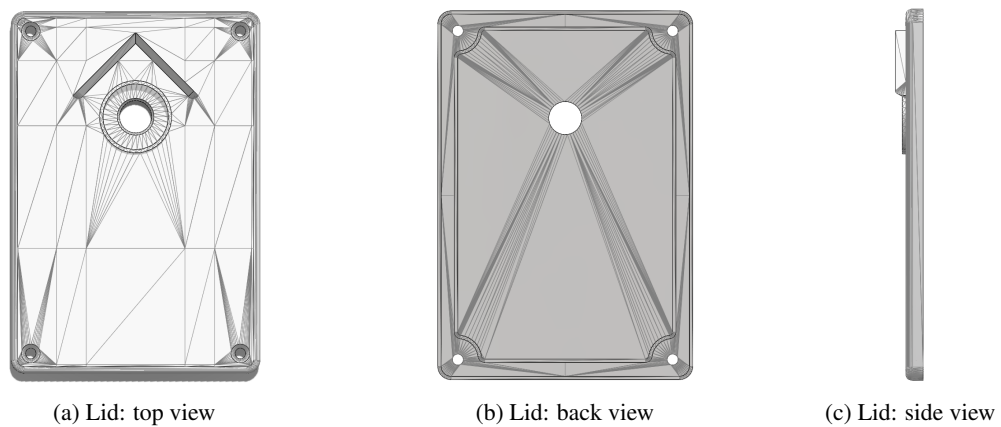


Figure 4.3: Lid design in SolidWorks

4.3.3 Assembly and Preview

Using SolidWorks the base, lid, necessary screws and the threaded insert can be put together into one Assembly file. This allows for the visualization of the full enclosure and the guarantee that everything fits together perfectly. The batteries and shield were not added to the Assembly as there were no available 3d models of them, and building those would be time-consuming and out of the scope of this work.

A realistic preview of the lid, base and assembly can be seen in Figure 4.4.

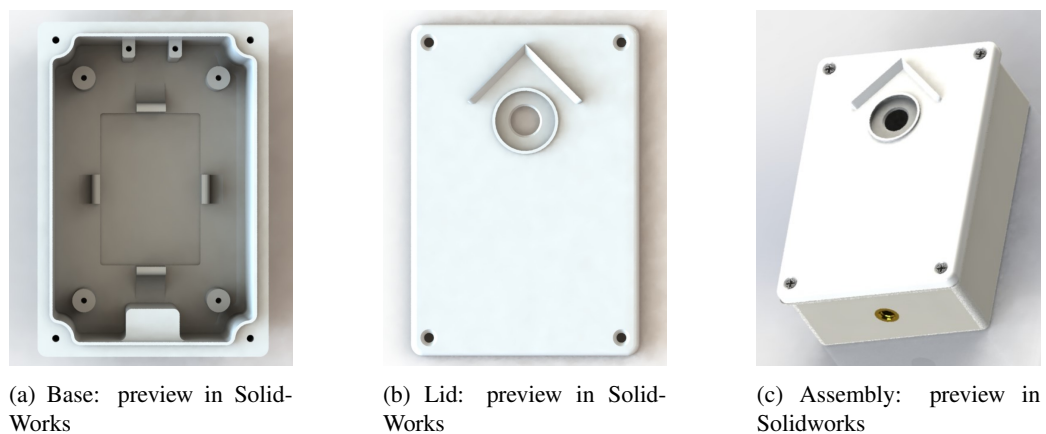


Figure 4.4: SolidWorks previews

4.4 Final 3D printed enclosure and assembly process

Following the design phase, the enclosure was fabricated using 3D printing technology, with PLA filament chosen for its balance of strength and ease of use in prototyping applications. This step was essential in creating a functional enclosure for field testing and proof-of-concept validation.

From the outset, the design incorporated several key features to ensure both functionality and durability. Four threaded inserts were integrated into the base to allow for the secure screwing

of the lid. This feature ensures that the screw holes maintain their shape and integrity over time, preventing wear from repeated assembly and disassembly. Additionally, a threaded insert was added to the bottom of the enclosure, designed to accommodate a $1/4'' - 20 \times 1/2$ screw and allow the entire system to be mounted securely onto commercially available supports, enhancing its flexibility in real-world applications.

During the 3D printing process, minor adjustments were made to improve the assembly process, such as refining the fit of the lid and its ease of printing, and increasing the strength of the support structure for the switch. These modifications ensured smooth assembly while preserving the protective features essential for safeguarding the system's electronics.

4.4.1 Assembly process

The assembly of the enclosure involved integrating all components, including the batteries, on-off switch, and the Arduino Tiny Machine Learning Kit. The assembly sequence, as shown in Figure 4.5, ensures that all components are securely housed while maintaining accessibility for maintenance and testing.

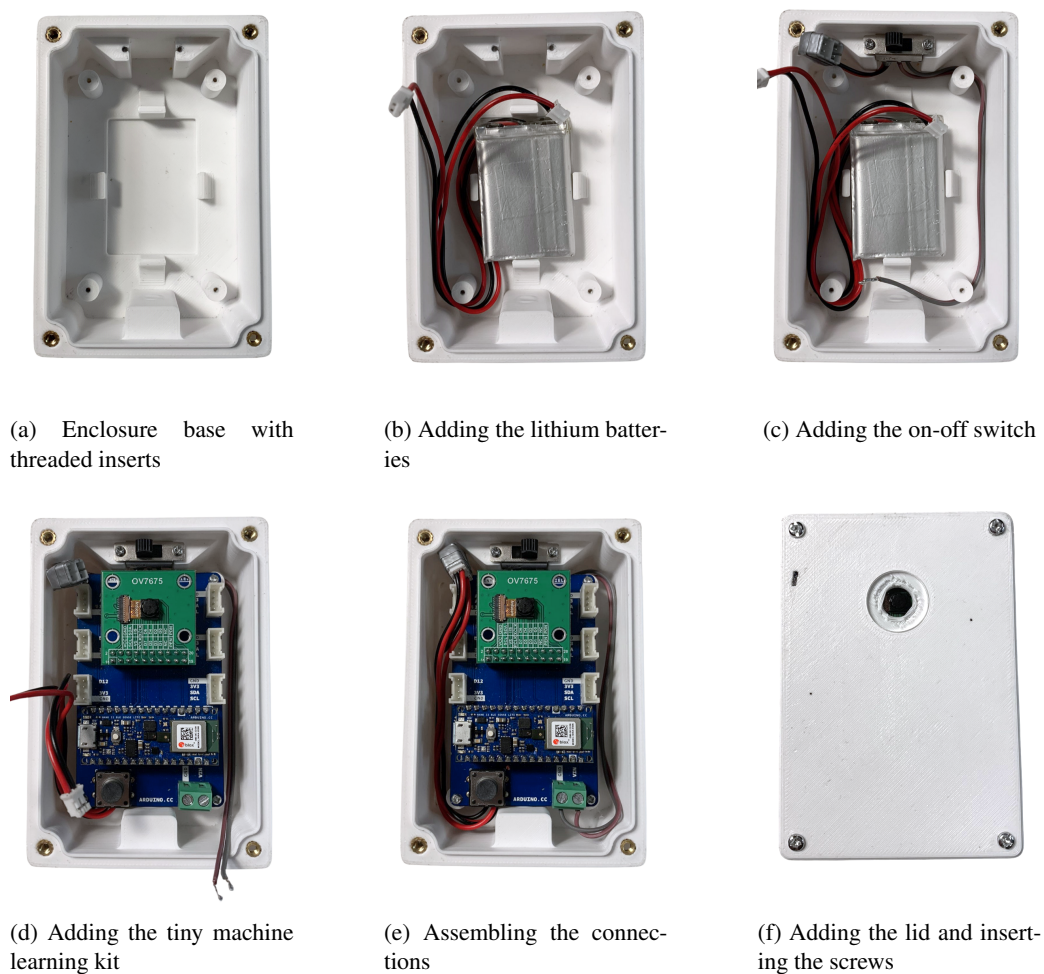


Figure 4.5: System Assembly

4.4.2 Power supply and charging

For sustained operation, the system relies on 3.7V lithium-ion batteries, which are easily rechargeable. Figure 4.6 illustrates the charging process, which utilizes a standard lithium battery charger. This flexible power solution ensures prolonged usage during field tests while maintaining portability.

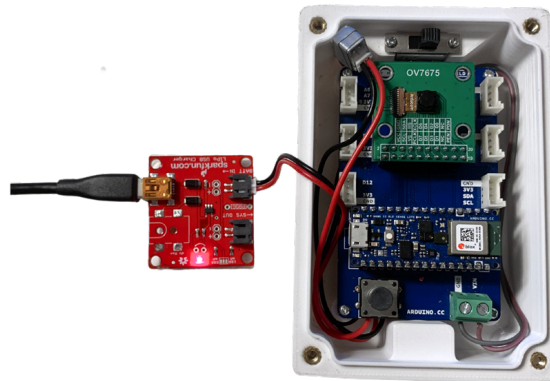


Figure 4.6: Charging process

Chapter 5

Tests & Results

This chapter describes the evaluation of various approaches for generating the training dataset and the testing of different pre-trained models.

Initially, multiple methodologies for constructing the training dataset were tested to determine the impact of balanced categories and slight augmentation of under-represented categories on model accuracy and stability. Subsequently, various models and image sizes were evaluated to identify the configurations that achieve the best performance while meeting the hardware constraints. Finally, the optimizations made on the chosen model are described. These details are presented in this chapter.

5.1 Testing Different Approaches for Generating the Training Dataset

In order to determine which of the 3 approaches described in Section 3.3.2 obtains the best results for generating the training dataset, all these approaches were tested.

Firstly, a testing and a training datasets were generated utilizing each approach, then a model was trained, tested and evaluated using those datasets. This was done multiple times for varying numbers of images per category used in the training dataset.

The model used for this purpose was MobilenetV2. This model was selected for its frequent and well-documented usage in the literature for this type of application: embedded systems with low memory. The methodology implemented for the model's training and testing are described in Section 3.4.2 and Section 3.4.3, respectively. The hyperparameters used for the training were:

- Training/Validation split: 80/20%
- Dropout Rate: 0.1
- Number of Neurons: 8
- Learning Rate 0.001
- Batch Size: 32

While these might not be the best parameters for each dataset size individually, it is necessary to have these parameters constant for every model training, so that there can be a meaningful comparison. By trial-and-error these were found to be the best overall, considering maximising learning with few images and minimising overfitting at every dataset size.

All the training datasets for these tests were generated on Windows, and while using the UI of the *RSM* tool described in Section 3.2 makes this process much faster, this is still a very intensive and time-consuming endeavour. To automate this process for generating tens of datasets, the CLI feature of *RSM* was used instead of the UI. Many text files with the arguments to be passed to the program were created, along with a batch file that calls `randomsamplemaker.py` multiple times with the arguments specified in the text files. Running the batch file simply by clicking on it twice creates all the datasets specified by the arguments. The developed batch file is presented in Appendix A.1.

There were 72 tests done in total: 24 for each approach and of these 24, 3 for each number of images per category. The following number of images per category were chosen for the tests: 10, 20, 50, 100, 150, 200, 250, 389 (389 is the maximum number of images in any category). These values are the number of images used per category for the non-wildfire class, for the wildfire class the values are half of these ones because the wildfire class has one more layer of categories than the non-wildfire class as it distinguishes between fire and smoke. For example, 10 images per category in the non-wildfire class correspond to 5 images per category in the wildfire class (5 images of the fire category + 5 images of the smoke category = 10 images).

The objectives of these tests are evaluating how the accuracy and variance changes with the increase of images per category for each approach and how these approaches compare to each other. To this effect, 3 graphs were plotted for each approach with the accuracy and variance values plotted as dots and the best approximation used for the curves, mostly polynomial approximations. Another graph was done for comparing the approaches by plotting the averages of the accuracies of each test and plotting the curves for each approach with simple smooth lines connecting the dots for representation.

5.1.1 An approach with balanced categories

This approach for generating the Training Dataset is described in Section 3.3.2: "1. An approach with balanced categories". The Testing Dataset is generated in the general use case described in Section 3.3.1. The results of the tests conducted using this approach are presented in Figure 5.1.

From these tests some important results and conclusions can be drawn:

- **Highest Accuracy:** The highest accuracy obtained was 88%, achieved once with both 250 and 389 images per category. However, these instances had higher variance, indicating that while larger datasets can achieve peak accuracies, they may also lead to less consistent results. This highlights the potential trade-off between accuracy and stability when increasing the dataset size.

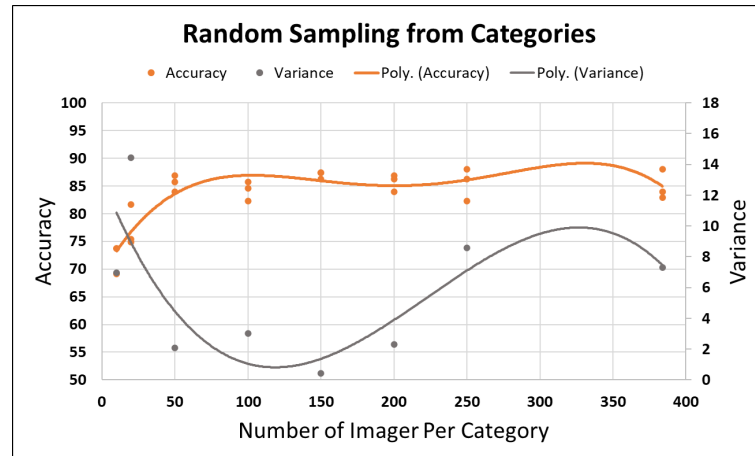


Figure 5.1: Accuracies obtained by random sampling from categories and corresponding variance. The curves are polynomial approximations of degree 3 and degree 4, respectively.

- Lowest Accuracy:** The lowest accuracy observed was 69.14% on a test done with 10 images per category. However, excluding tests with fewer than 100 images per category, the lowest accuracy obtained was 82.29%, recorded twice, once for 100 and another for 250 images per category. It should be noted that one of the tests done with 250 images per category was the one that achieved the highest accuracy. This variance within the same category size underscores the importance of dataset balance and suggests that simply increasing the number of images does not guarantee improved performance.
- Variance Trends:** The variance curve shows that variance decreases with the increase in images up to 150, after which it increases again. This behavior supports the theory that as the number of images increases while maintaining balance the variance should decrease and as the number of images increases by disrupting balance, above a certain threshold of non-balance, the variance should increase again. The lowest variance was obtained with 150 images per category, coinciding with the highest average accuracy of 87.05
- Impact of Insufficient Data:** Tests with fewer than 50 images per category exhibited significant fluctuations in both accuracy and variance, indicating insufficient data for reliable training outcomes. This reinforces the necessity of having a minimum number of images to achieve stable performance.
- Diminishing Returns Beyond 200 Images:** For image counts exceeding 200 per category, accuracy remains high, but increased variance suggests that additional images introduce noise or overfitting to some categories rather than improving model performance. This indicates diminishing returns in accuracy improvements with further increases in dataset size.
- Polynomial Approximations:** Polynomial approximations indicate that accuracy plateaus around 150 images per category, further supporting the notion of diminishing returns with

larger datasets. This plateau suggests an optimal point where further increases in images do not significantly enhance model accuracy.

- **Optimal Balance:** The observed patterns suggest that maintaining a balanced dataset with an optimal number of images per category is crucial for minimizing variance and achieving consistent high accuracy. The combination of low variance and high average accuracy at 150 images per category makes it the most effective configuration for this approach.

In summary, using 150 images per category provides an optimal balance of high accuracy and low variance, making it the most effective training dataset configuration for this approach. Further details on the averages of the tests and comparisons with other datasets are presented in Section 5.1.4.

5.1.2 An approach with balanced categories and slight augmentation

This approach for generating the Training Dataset is described in Section 3.3.2 in "2. An approach with balanced categories and slight augmentation". The Testing Dataset is generated in the general use case described in Section 3.3.1.

The results of the tests conducted using this approach are presented in Figure 5.2.

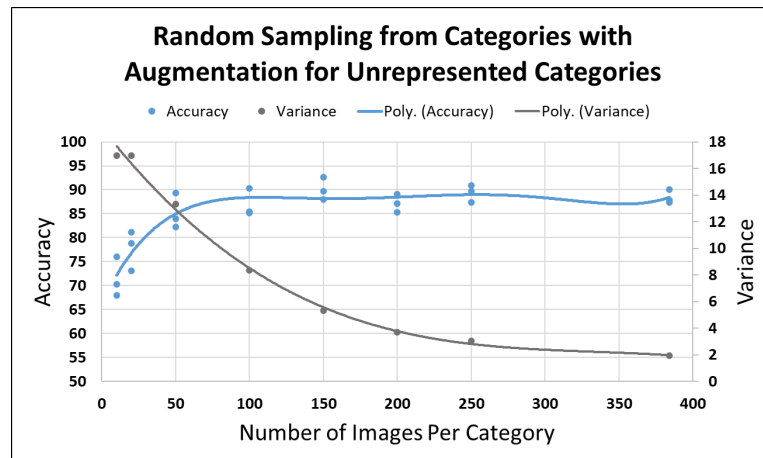


Figure 5.2: Accuracies obtained by random sampling from categories with augmentation for under-represented categories and corresponding variance. The curves are polynomial approximations of degree 5 and degree 3, respectively.

From these tests some important results and conclusions can be drawn:

- **Highest Accuracy:** The highest accuracy obtained was 92.57%, achieved when training the model on 150 images per category.
- **Lowest Accuracy:** The lowest accuracy observed was 68% on a test done with 10 images per category. However, excluding tests with fewer than 100 images per category, the lowest accuracy obtained was 85.3%, observed with 200 images per category.

- **Variance Trends:** The variance curve shows a clear decrease in variance with an increase in images per category. This aligns with the theoretical expectations that increasing the number of training images while maintaining category balance reduces variance. Additionally, it confirms that most of the variance is due to the training and not the testing dataset.
- **Impact of Insufficient Data:** Tests with fewer than 50 images per category exhibited significant fluctuations in accuracy, highlighting the inadequacy of small datasets for reliable training outcomes. This reinforces the necessity of having a minimum number of images to achieve stable and reliable model performance.
- **Polynomial Approximations:** Polynomial approximations indicate that accuracy plateaus around 100 images per category. This plateau suggests that on average further increases in the number of images do not significantly enhance model accuracy. However, the continuous decrease in variance with increasing images is a positive outcome, suggesting that model stability continues to improve.
- **Optimal Balance:** The observed patterns suggest that maintaining a balanced dataset with an optimal number of images per category is crucial for minimising variance and achieving consistent high accuracies. The combination of low variance and high average accuracy at 150 images per category makes it the most effective configuration for this approach. Moreover, while the variance is higher when using more images, the lowest accuracy obtained at 150 images per category is higher than the lowest value of accuracy of any other group of tests. Additionally, the highest value of accuracy is also higher than every other test. The lowest accuracy obtained at 150 images per category was 88% and the highest was 92.57%.

In summary, using 150 images per category with slight augmentation for under-represented categories provides an optimal balance of high accuracy and low variance, making it the most effective dataset configuration for this approach. Further details on the averages of the tests and comparisons with other datasets are presented in Section 5.1.4.

5.1.3 An approach without considering balance for training

This approach for generating the Training Dataset is described in Section 3.3.2 in "3. An approach without considering balance for training". The Testing Dataset is generated in the general use case described in Section 3.3.1.

The number of images used for each test was taken from the number of images used in each corresponding test in "1. An approach with balanced categories" (5.1.1). This allows for consistency when comparing the graphs of the different approaches.

The results of the tests conducted using this approach are presented in Figure 5.2.

From these tests some important results and conclusions can be drawn:

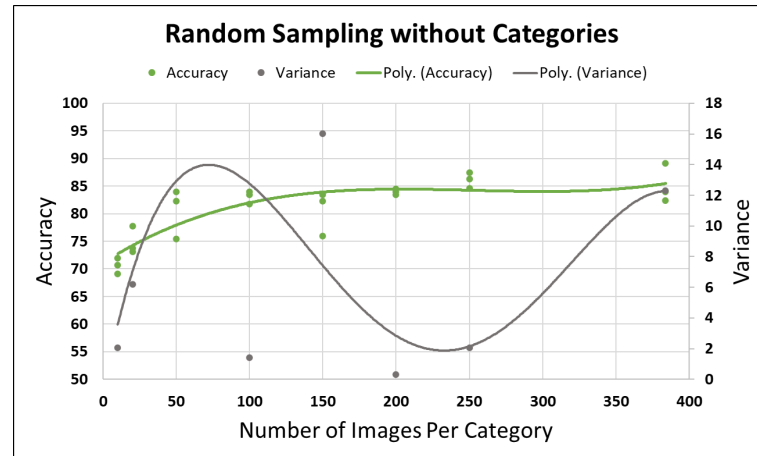


Figure 5.3: Accuracies obtained by random sampling without categories and corresponding variance. The curves are polynomial approximations of degree 4 and degree 3, respectively.

- Highest Accuracy:** The highest accuracy obtained was 89.14%, achieved with 384 images. This demonstrates the potential for high accuracy even without explicitly balancing categories, however this potential does not come with consistency by low variance, as another test using 384 images per category obtained an accuracy of 82.43%.
- Lowest Accuracy:** The lowest accuracy observed was 69.14%, recorded with 10 images. However, excluding tests with fewer than 50 images, the lowest accuracy obtained was 75.43%, recorded with 50 images. This indicates the variability in performance at lower image counts.
- Variance Trends:** The variance curve shows significant fluctuations across different image counts. While some points, such as 10, 100 and 200 images, show very low variance, others, like 50, 150 and 389 images, exhibit extremely high variance. This inconsistency presents a challenge in drawing reliable conclusions from these results. A potential explanation lies in the random nature of the selection process employed by this approach. Since it randomly samples from the entire dataset, there's a possibility of selecting a balanced set of images on some occasions, whereas on others, the selection might be unbalanced. This potential imbalance in the sampled data could contribute to the observed fluctuations in variance.
- Impact of Insufficient Data:** Tests with fewer than 50 images per category exhibited significant fluctuations in accuracy and higher variance, highlighting the inadequacy of small datasets for reliable training outcomes. This reinforces the necessity of having a minimum number of images to achieve stable performance.
- Polynomial Approximations:** Polynomial approximations indicate that accuracy tends to plateau around 100-150 images per category. This plateau suggests that, on average, further increases in images do not significantly enhance model accuracy beyond this point. However, the variance shows more fluctuation, with a notable dip at around 200 images before rising again.

- **Optimal Balance:** The observed patterns suggest that while maintaining a balanced dataset is important, even without explicitly balancing categories, a sufficient number of images per category can achieve high accuracy. For example, 200 images per category provides a good balance of low variance and high accuracy. However as mentioned earlier these results of variance might not be significant, further testing should be done if additional conclusions need to be drawn.

In summary, using random sampling without considering categories can still achieve high accuracy. However, due to the significant fluctuations in variance observed, no solid conclusions can be drawn regarding the optimal number of images per category. Ensuring dataset diversity and exploring more refined augmentation techniques could help in achieving more consistent performance. Further details on the averages of the tests and comparisons with other approaches are presented in Section 5.1.4.

5.1.4 Comparison between approaches and choice of approach for generating the training dataset

The averages of every group of accuracies obtained for every approach presented in the previous sections were calculated and plotted in Figure 5.4, the curves are simple smooth lines connecting the dots for representation.

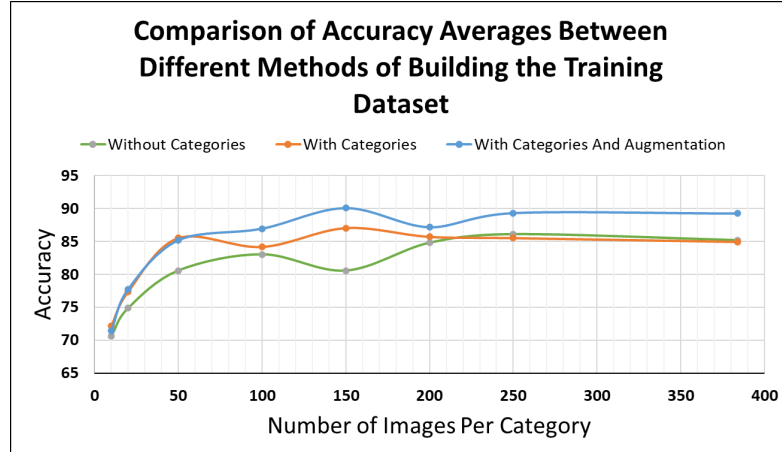


Figure 5.4: Average accuracies obtained by random sampling using 3 different methods. The curves are simple smooth lines connecting the dots for representation.

From the graphs presented in Figures 5.1 - 5.4 and from the results of each approach, including Table 5.1 several conclusions can be drawn:

- **Highest Accuracy:** The method with balanced categories and slight augmentation had all the highest accuracy scores for the tests done over 50 number of images per category. It achieved the highest individual accuracy of 92.57%, the highest maximum average accuracy of 90.09%, the highest minimum individual accuracy of 82.29%, the highest minimum

Table 5.1: Accuracy and average accuracy comparison

	Accuracy	Average Accuracy
Min		
Without Categories	75.43	80.57
With Categories	82.29	84.19
With Categories + Augmentation	82.29	85.19
Max		
Without Categories	89.14	86.09
With Categories	88	87.05
With Categories + Augmentation	92.57	90.09
Average		
Without Categories		80.62
With Categories		85.49
With Categories + Augmentation		87.86

average accuracy of 85.19%, and the highest average accuracy of 87.86%. This demonstrated that using the approach with categories and slight augmentation is advantageous for training the model.

- **Variance Trends:** Variance trends varied significantly across the different methods. The approach without considering categories showed the most inconsistency, with variance fluctuating widely and making it difficult to draw reliable conclusions. The approach using balanced categories without slight augmentation showed decreasing variance with an increase in images up to 150, after which variance increased again. The method with balanced categories and slight augmentation had the most consistent variance, steadily decreasing as the number of images increased, indicating improved stability.
- **Impact of Insufficient Data:** All methods demonstrated significant fluctuations in accuracy when fewer than 50 images per category were used, highlighting the inadequacy of small datasets for reliable training outcomes and reinforcing the necessity of having a minimum number of images to achieve stable performance. Notably, the approach that does not consider categories exhibited the slowest rate of improvement in accuracy for datasets with fewer than 50 images per category, compared to the other two methods, where the increase in accuracy was more significant and similar between each other, this similarity was already expected as there are almost no under-represented categories when considering under 50 images per category.

5.1.4.1 Conclusion and choice of approach

Based on the analysis of the results, the method with balanced categories and slight augmentation is the most effective approach for building the training dataset. This approach provides the highest accuracy and most stable performance. On average, it saw a 7.24% increase in global accuracy compared to the method without the usage of categories or slight

augmentation, and a 2.37% increase in global accuracy compared the the method only using balanced categories. Therefore, it was the one selected for training the model. Optimal results were achieved using 150 images per category, and consequently, this value was used for generating the training dataset.

5.2 Testing Various Models and Image Sizes

This section describes the tests conducted for the different options of pre-trained models described in Section 3.5. The main purpose is determining which can be used, considering the hardware constraints defined in Section 4.1. Firstly, it describes how the estimated values of inferencing time, RAM usage and flash usage change with respect to MobileNet's alpha values and image size. Additionally, it describes the testing of the models for each alpha value and image size and selects the best ones considering the hardware constraints. All of these tests were done in Edge Impulse Studio.

5.2.1 Estimated Inferencing Time, RAM Usage and Flash Usage

MobileNetV1

Edge Impulse has the option for using the pre-trained MobileNetV1 for TL on image classification tasks. The alpha values that can be used for this model are 0.1, 0.2, 0.25, 0.5 and 0.75.

This models was tested for every alpha value for an image input size of 32x32p. The results were plotted in the graph presented in Figure 5.5.

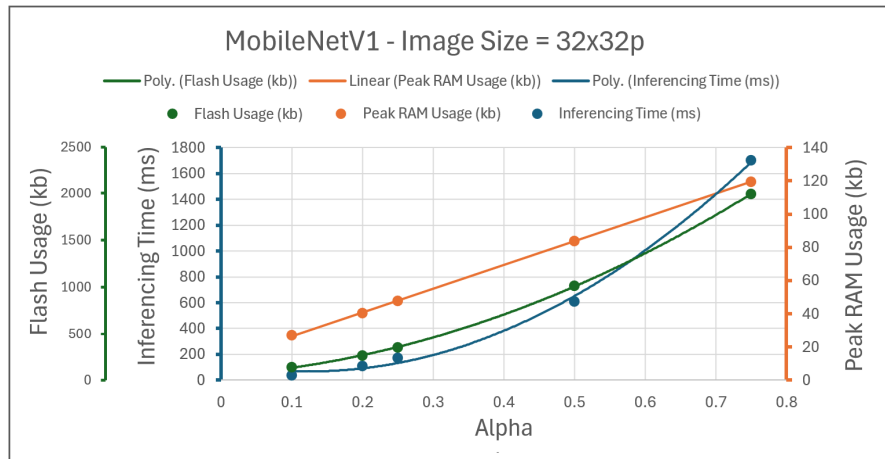


Figure 5.5: Change in inferencing time, RAM usage and flash usage with respect to MobileNetV1's alpha value

The following conclusions can be derived from these results:

- **Inferencing Time:** As the alpha value increases, the inferencing time increases exponentially. This is due to an increase in the alpha value leading to an increase in the number of

model parameters, which increases the computational load and hence the time required for inferencing.

- **Flash Usage:** Similar to inferencing time, flash usage increases exponentially with the increase of alpha values. This is because a higher alpha value corresponds to a larger model with more parameters, requiring more flash memory to store the model weights.
- **Peak RAM Usage:** Peak RAM usage also rises with increasing alpha values, although this increase is almost perfectly linear. This trend is due to the larger model size necessitating more memory for loading the model during inference.

This models was also tested for every image size for an alpha value of 0.25. The results were plotted in the graph presented in Figure 5.6

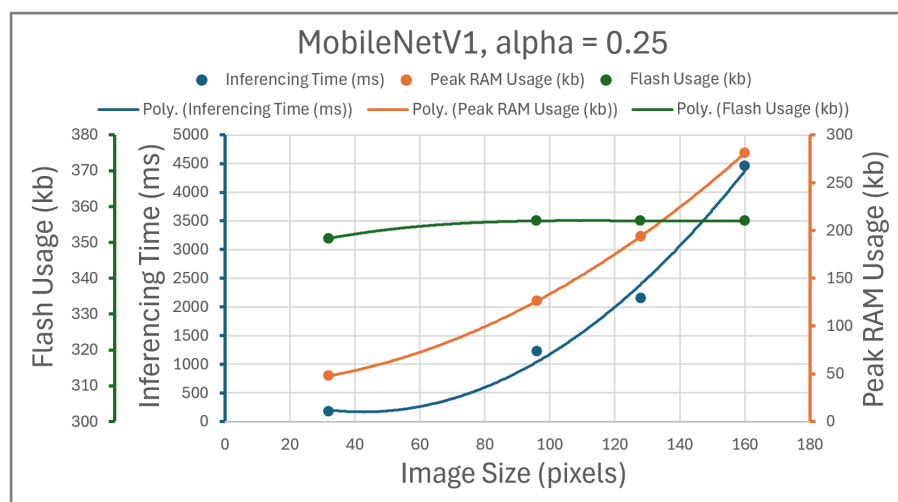


Figure 5.6: Change in inferencing time, RAM usage and flash usage with respect to MobileNetV1's input image size

The following conclusions can be derived from these results:

- **Inferencing Time:** As the image size grows, the amount of data processed by the model increases, leading to longer processing times. This trend is evident from the curve representing inferencing time, which shows an exponential rise as the image size increases.
- **Flash Usage:** Flash usage also increases with image size from its value with 32x32 pixel images to its value with 96x96 pixel images. This initial increase occurs because larger images require more parameters and memory to store the model weights and intermediate computations. However, beyond 96x96 pixels, the flash usage remains constant at 356 KB. This plateau indicates that the model has reached its maximum storage capacity for weights and computations, suggesting that additional image size increases do not require further flash memory.

- **Peak RAM Usage:** Peak RAM usage increases exponentially with image size. As image size increases from 32x32 pixels to 160x160 pixels, the RAM usage rises due to larger images requiring more memory for processing. This trend indicates that handling larger images significantly increases memory consumption due to the need for more intermediate data storage and computations.

MobileNetV2

Edge Impulse has the option for using the pre-trained MobileNetV2 for TL on image classification tasks. The alpha values that can be used for this model are 0.05, 0.1, 0.35, 0.5, 0.75 and 1.

This models was tested for every alpha value for an image input size of 32x32p. The results were plotted in the graph presented in Figure 5.7.

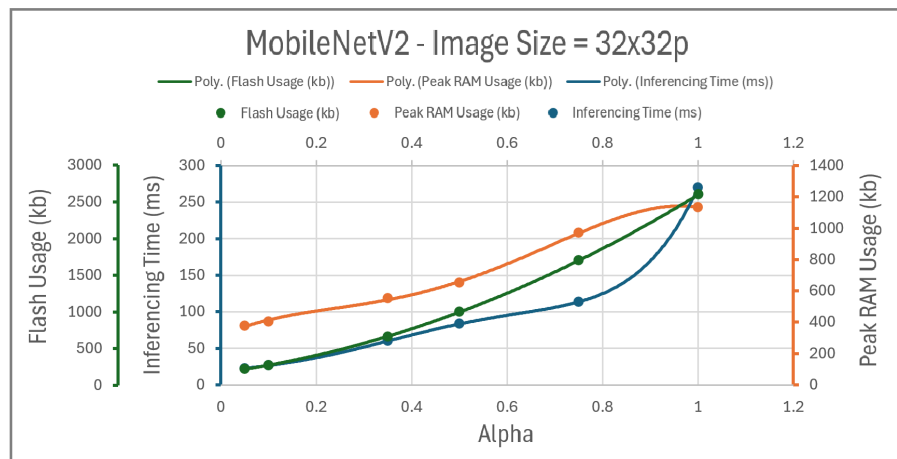


Figure 5.7: Change in inferencing time, RAM usage and flash usage with respect to MobileNetV2's alpha value

The following conclusions can be derived from these results:

- **Inferencing Time:** The inferencing time increases almost linearly with the increase of alpha from 0.05 to 0.75. From 0.75 to 1, the inferencing time increases at a higher rate, indicating that larger models require significantly more time to make predictions.
- **Flash Usage:** Flash usage also increases with the alpha value. Higher alpha values result in a more complex model with more parameters, which increases the storage requirements for model weights and computations. The graph illustrates a substantial increase in flash usage as alpha increases, peaking at the highest alpha value tested.
- **Peak RAM Usage:** Peak RAM usage also rises with increasing alpha values, although this increase is almost perfectly linear. This trend is due to the larger model size necessitating more memory for loading the model during inference.

This models was also tested for every image size for an alpha value of 0.25. The results were plotted in the graph presented in Figure 5.8

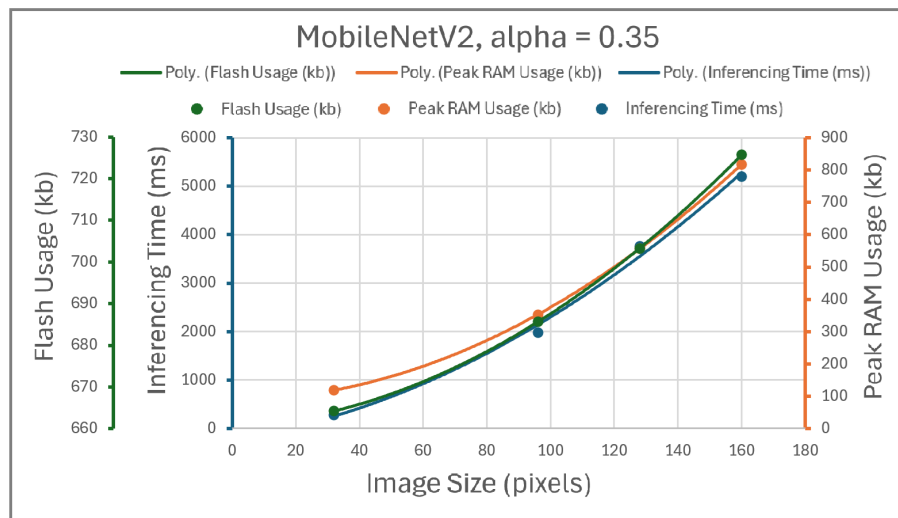


Figure 5.8: Change in inferencing time, RAM usage and flash usage with respect to MobileNetV2's input image size

The following conclusions can be derived from these results:

- **Inferencing Time:** Inferencing time increases polynomially with image size. As the image size grows from 32x32 pixels to 160x160 pixels, the processing time rises significantly. This is due to the increased computational complexity and the larger amount of data that needs to be processed by the model.
- **Flash Usage:** Flash usage also increases polynomially with image size. Larger images require more parameters and memory for storing the model weights and intermediate computations. The graph shows that flash usage rises steadily with increasing image size, reflecting the additional storage needs for larger input data.
- **Peak RAM Usage:** Peak RAM usage increases linearly with image size. As image size grows, more memory is needed to handle the larger input data and the associated computations during inference. The graph indicates a consistent linear increase in RAM usage, showing that larger images proportionally increase memory consumption.

Conclusions

As expected, in both models, an increase in alpha or image size leads to an increase in all performance metrics: Inferencing Time, Peak RAM Usage and Flash Usage.

Comparing the models without considering accuracy is not appropriate, as one model with a lower value of alpha and a smaller image size might achieve a higher accuracy than the other that has a different architecture. The comparisons made between specific configurations are presented in Section 5.2.2.

5.2.2 Best Models Complying with the Hardware Constraints

The available pre-trained models that can be used in Edge Impulse are MobileNetV1 and MobileNetV2 with multiple values of alpha. Each time a model is trained, Edge Impulse estimates the Inferencing Time, Peak RAM Usage and Flash Usage for a specific hardware environment, in this case, the Arduino Nano 33 BLE Sense. These estimates are made for the model being deployed using the TFLite model directly, using the EON Compiler or using the EON Compiler RAM Optimized.

The tests were done in increasing order of alpha, and for each alpha in increasing order of image size: 32x32, 96x96, 128x128 and 160x160. These tests were conducted for each scenario until the configuration did not comply with the hardware limitations. This process is presented in Figure 3.14.

The method used for training the models is described in Section 3.4.2 and for testing, the one described in Section 3.4.3.

The training dataset was generated by the approach described in Section 3.3.2 in "2. An approach with balanced categories and slight augmentation". This decision was based on the comparative analysis conducted in Section 5.1.4. The testing dataset was generated using the general use case described in Section 3.3.1. Every training and testing phases were done using the same images from the datasets generated, for different image sizes the original dataset was resized.

All the models used had no dense layer for TL to ensure the lowest memory consumption and a valid comparison between models.

The various models were tested, and the results were obtained and placed in table A.1. The models with the highest combination of alpha and image size are highlighted in yellow in the table and were chosen for further testing. The models chosen were:

- MobileNetV1 with 0.25 as the value for alpha using images of 128 by 128 pixels compiled using the EON Compiler (RAM Optimized).
- MobileNetV2 with 0.1 as the value for alpha using images of 96 by 96 pixels compiled with the EON Compiler (RAM Optimized).
- MobileNetV2 with 0.35 as the value for alpha using images of 32 by 32 pixels compiled using the TFLite model directly.

5.2.3 MobileNetV1, 128x128p, alpha=0.25

The model MobileNetV1, with an alpha value of 0.25 and using 128 by 128 pixel images was tested with 0, 2, 4, 8, 16, 32 and 64 neurons at the dense layer following the base model. The hyperparameters used for each test can be found in Figure A.2 (a).

From the results obtained, the models using 2 and 4 neurons were the ones that achieved the highest global accuracy of 88.57%. However, the one using 2 neurons had the highest AUC-ROC,

Table 5.2: MobileNetV1, alpha=0.125, image size = 128x128p - Performance metrics for different numbers of neurons in one dense layer prior to the classification layer.

#Neurons	Global Accuracy (%)	Non-Wildfire Accuracy (%)	Wildfire Accuracy (%)	Area under ROC Curve
0	86.29	87.8	84.7	0.89
2	88.57	90	87.1	0.91
4	88.57	92.2	84.7	0.9
8	84.57	83.3	85.9	0.88
16	88	92.2	83.5	0.91
32	86.86	87.8	85.9	0.91
64	85.71	91.1	80	0.88

of 0.91, and also the highest percentage of true positives (Wildfire Accuracy), of 87.1%. Thus, the MobileNetV1 with an alpha of 0.25 using 128 by 128 pixel images and using 2 neurons at the dense layer following the base model was chosen as the best model for this configuration.

5.2.4 MobileNetV2, 96x96 alpha=0.1

The model MobileNetV2, with an alpha value of 0.1 and using 96 by 96 pixel images was tested with 0, 2, 4, 8, 16, 32 and 64 neurons at the dense layer following the base model. The hyperparameters used for each test can be found in Figure A.2 (b).

Table 5.3: MobileNetV2, alpha = 0.1, image size = 96x96 - Performance metrics for different numbers of neurons in one dense layer prior to the classification layer.

#Neurons	Global Accuracy (%)	Non-Wildfire Accuracy (%)	Wildfire Accuracy (%)	Area under ROC Curve
0	87.43	88.9	85.9	0.9
2	86.86	90	83.5	0.9
4	88.57	88.9	88.2	0.91
8	86.29	87.8	84.7	0.87
16	86.86	86.7	87.1	0.91
32	86.29	85.6	87.1	0.91
64	84.57	83.3	85.9	0.89

From the results obtained, the model using 4 neurons achieved the highest global accuracy of 88.57%. It also achieved the highest AUC-ROC, of 0.91, and the highest Wildfire Accuracy of 88.2%. Thus, the MobileNetV2 with an alpha of 0.1, using 96 by 96 pixel images and 4 neurons at the dense layer following the base model, was chosen as the best model for this configuration.

5.2.5 MobileNetV2, 32x32, alpha=0.35

The model MobileNetV2, with an alpha value of 0.35 and using 32 by 32 pixel images was tested with 0, 2, 4, 8, 16, 32, 64 and 128 neurons at the dense layer following the base model. The hyperparameters used for each test can be found in Figure A.2 (c).

From the results obtained, the model using 64 neurons on the dense layer achieved the highest global accuracy of 72%. It also achieved the highest true positive accuracy (Wildfire Accuracy) and had the third-highest AUC-ROC. Thus, the MobileNetV2 with an alpha of 0.35, using 32 by

Table 5.4: MobileNetV2, alpha = 0.35, image size = 32x32 - Performance metrics for different numbers of neurons in one dense layer prior to the classification layer.

#Neurons	Global Accuracy (%)	Non-Wildfire Accuracy (%)	Wildfire Accuracy (%)	Area under ROC Curve
0	65.14	91.1	37.6	0.7
2	68.57	86.7	49.4	0.76
4	69.14	94.4	42.4	0.75
8	66.86	90	42.4	0.71
16	68	92	42.4	0.72
32	68.57	88.9	47.1	0.73
64	72	91.1	51.8	0.73
128	69.71	92.2	45.9	0.73

32 pixel images and 4 neurons at the dense layer following the base model, was chosen as the best model for this configuration.

5.2.6 Best Performing Model for Each Configuration

A summary of the best performing models for each configuration is presented in Table 5.5.

Table 5.5: Summary of performance metrics for best configurations

Model	Image Size	Alpha	#Neurons	Global Accuracy (%)	Area under ROC Curve	Inferencing Time (ms)
MobileNetV1	128x128	0.25	2	88.57	0.91	2193
MobileNetV2	96x96	0.1	4	88.57	0.91	1404
MobileNetV2	32x32	0.35	64	72	0.73	319

From the results obtained, it can be seen that the MobilenetV1 with a 128x128 image size and alpha of 0.25, and the MobileNetV2 with 96x96 image size and alpha of 0.1 both achieved the highest global accuracy and AUC-ROC. The distinguishing factor is the inferencing speed, where the MobileNetV2 configuration saw a 1.56% increase over the MobileNetV1's.

This increase in inferencing speed is an important factor to consider in embedded systems applications where battery capacity is limited, as a faster inferencing speed increases the system's operational time without recharging.

Lastly, MobileNetV2 with a 32x32 image size and alpha of 0.35 was the model with the lowest global accuracy. Although it obtained highest non-wildfire accuracy of 91.1%, its wildfire accuracy was only 51.8%, which makes this configuration unsuitable for wildfire detection applications.

Choice of Model

Considering the comparative analysis, and given the importance of accurately detecting wildfires along with the need for efficient inferencing speed to reduce battery usage, the MobileNetV2 with a 96x96 image size, alpha of 0.1 and 4 neurons in the final dense layer prior to the classification layer was chosen for further optimization and deployment. The methodology for this further optimization is described in Section 3.6 and the tests and results are described in Section 5.3.

5.3 Final Model Optimization and Tunning

This section presents the results of the optimization and tuning processes for the final model, including the performance metrics for different dropout rates, the effect of batch normalization, and the learning rate optimization.

Dropout Rate Adjustment Results

Table 5.6 shows the performance metrics for various dropout rates. The metrics considered are global accuracy, non-wildfire accuracy, wildfire accuracy, and the area under the ROC curve.

Table 5.6: Performance metrics for different dropout rates.

Dropout	Global Accuracy (%)	Non-Wildfire Accuracy (%)	Wildfire Accuracy (%)	Area under ROC Curve
0.1	88.57	88.9	88.2	0.91
0.2	87.43	84.4	90.6	0.9
0.3	88.57	84.4	92.9	0.9
0.4	88	83.3	92.9	0.89
0.5	89.14	87.8	90.6	0.9
0.6	88.57	87.3	89.4	0.9
0.7	88.57	85.6	91.8	0.9

From Table 5.6, it is observed that a dropout rate of 0.5 yielded the highest global accuracy at 89.14%, indicating its potential to effectively mitigate overfitting while maintaining high performance across different metrics.

Batch Normalization Results

Following the determination of the optimal dropout rate for the original configuration, batch normalization was introduced. Table 5.7 presents the performance metrics with batch normalization applied across the same range of dropout rates.

Table 5.7: Performance metrics for different dropout rates with batch normalization.

Dropout	Global Accuracy (%)	Non-Wildfire Accuracy (%)	Wildfire Accuracy (%)	Area under ROC Curve
0.1	87.43	86.7	88.2	0.92
0.2	88.57	86.7	90.6	0.91
0.3	89.71	88.9	90.6	0.91
0.4	88.57	85.6	91.8	0.9
0.5	88	86.7	89.4	0.91
0.6	89.14	91.1	87.1	0.91
0.7	87.43	84.4	90.6	0.92

Table 5.7 indicates that adding batch normalization improves performance across various dropout rates. The optimal configuration was identified at a dropout rate of 0.3, achieving a global accuracy of 89.71%.

An additional improvement from using batch normalization is in inferencing speed: 992ms, an improvement of 41.5% from the previous 1404ms without batch normalization. While this seems counterintuitive, as batch normalization should add computational overhead, the compression and

optimization algorithms that Edge Impulse employs on the Arduino Nano 33 BLE Sense might make the inference faster when using batch normalization in this case. This inferencing time was tested on the actual Arduino Nano 33 BLE Sense to ensure this was not an error with Edge Impulse's estimation, and indeed, the inferencing time was close to the expected, at around 1.1 seconds.

Learning Rate Tuning Results

The learning rate tuning involved evaluating several learning rates to identify the one that provided the best performance. Table 5.8 presents the performance metrics for different learning rates, including global accuracy, non-wildfire accuracy, wildfire accuracy, and AUC-ROC. Figure 5.9 displays the effect of learning rate on global accuracy.

Table 5.9 illustrates the performance metrics obtained from tuning the learning rate for the selected model configuration (dropout rate of 0.3 with batch normalization).

Table 5.8: Performance metrics for different learning rates.

Learning Rate	Global Accuracy (%)	Non-Wildfire Accuracy (%)	Wildfire Accuracy (%)	Area under ROC Curve
0.0001	86.86	83.3	90.6	0.89
0.0003	86.86	84.4	89.4	0.90
0.0005	89.71	88.9	90.6	0.91
0.001	88.00	86.7	89.4	0.89
0.005	90.29	88.9	91.8	0.91
0.01	91.43	88.9	94.1	0.93
0.02	85.14	76.7	94.1	0.91
0.05	48.57	0.00	100	0.50

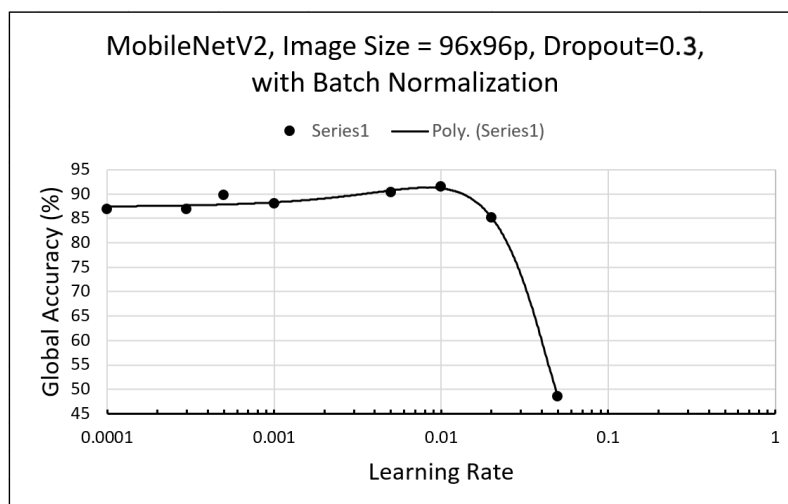


Figure 5.9: Learning rate optimization

From the experiments shown in Figure 5.9, the optimal learning rate was 0.01, which yielded the highest global accuracy of 91.4%.

Final Model Configuration

Considering all the previous tests, the final model configuration employed was MobileNetV2 with a 96x96 image size input, 4 neurons in the final dense layer prior to the activation layer, batch normalization after this dense layer, followed by 0.3 dropout rate.

The hyperparameters used for training the model were:

- Learning Rate: 0.01
- Batch Size: 32
- Training/Validation split: 80/20

5.4 Testing Final Model on Available Datasets

The final model obtained in Section 5.3 was evaluated on three distinct datasets: DeepFire, Corsican Fire Database and BowFire. Table 5.9 presents a comparative analysis of the performance of our proposed approach (PA) with existing image classification models reported in the literature.

It should be noted that while some wildfire detection research papers tested their models on the BoWFire dataset, inconsistencies within those papers prevented their results from being used for comparison, as they might be unreliable.

- **DeepFire dataset:** The final model achieved a global accuracy comparable to VGG19, a model with approximately 100 times more parameters than MobileNetV2 ($\alpha=0.1$) [27, 36]. With a difference in accuracy of only 0.79%.
- **Corsican Fire Database:** The Corsican Fire Database was tested using two approaches PA1 (Proposed Approach 1) and PA2 (Proposed Approach 2). This is because this dataset contains two types of images, random and diverse images from different settings, and sequences of images from the same settings. It is not clear if the test done on InceptionV3 model used the PA1 or PA2.

PA1 tested the proposed model on all images, while PA2 tested only on the diverse images, excluding the sequences of images in the same scenarios.

PA1 had a significant reduction in accuracy compared to the InceptionV3 model. On the other hand, the PA2, where those sequenced images were not used for testing, achieved a 96.13% accuracy, comparable to the 20 times larger, in terms of parameters, InceptionV3 model [82]. The final model saw a 2.47% decrease in global accuracy, that represents only the true positive rate, as the Corsican Fire Database is composed of only fire images.

- **BoWFire dataset:** Direct comparisons to other models' performance could not be made for this dataset, as the information obtained from papers found in the literature regarding models tested on this dataset could not be verified and had multiple inconsistencies. However, several conclusions could be made from testing the final model on this dataset.

Firstly, the global accuracy was significantly lower than on the other datasets. This is to be expected as this dataset contains many images that are not pertinent to the primary focus of this work.

Secondly, there were 25 images of firefighters or people that resemble firefighters in the dataset in non-wildfire contexts. Of these 25 images, 20 were misclassified as wildfire. This suggests an unintended association learned by the model between firefighters and wildfire events. Incorporating additional images of firefighters in non-wildfire contexts into the training data is a proposed mitigation strategy and is included in Future Work (Section 7.2).

Table 5.9: Comparison of performance metrics with the literature

Metrics (%)	DeepFire dataset		Corsican Fire Database			BoWFire dataset
	PA	Khan et al. - VGG19 [78]	Sousa et al. - InceptionV3 [83]	PA1	PA2	PA
Accuracy	94.9	95	98.6	88.73	96.13	72.44
TPR	94.6	94.21	-	-	-	93.4
TNR	95.2	95.79	-	-	-	48.1
FPR	4.8	4.21	-	-	-	6.6
FNR	5.4	5.79	-	-	-	51.9

In summary, the final model demonstrated comparable performance to more parameter-heavy models on the DeepFire and the Corsican Fire Database, with a slight drop in accuracy. The model's performance on the BoWFire dataset highlighted areas for improvement, particularly in handling images with firefighters in non-wildfire contexts, suggesting the need for more diverse training data.

Chapter 6

Discussion

The objective of this project was to develop a functional, low-power wildfire detection system using CNNs, suitable for deployment in remote environments, particularly the WUI. The system involved creating a balanced dataset, training and deploying CNN models on resource-constrained hardware, and designing a protective enclosure for real-world testing. While the project achieved several successes, there are areas where alternative approaches could have led to even better results. This discussion reflects on both the project's achievements and limitations, critically analyzing potential improvements.

A key strength of the project was the development of a well-balanced dataset. By combining 17 datasets selected from the internet and hand-captured images, the model was trained on a variety of wildfire scenarios, including different times of day, distances, and environmental conditions. This diversity was essential for the model's ability to generalize effectively across various test datasets. Data augmentation further enhanced the dataset, allowing for more stable model performance by artificially increasing data diversity and balance. This strategy contributed to a global accuracy of 91.4%, which is impressive given the resource constraints of the application.

The creation of this balanced dataset is a significant contribution to the field of wildfire detection. By providing a diverse and representative collection of images, the project has laid a foundation for future research and development in this area. This dataset can be used by other researchers to develop and evaluate new models, potentially leading to even more accurate and efficient wildfire detection systems.

The RSM tool played a vital role in automating the dataset balancing process. By allowing the random selection of images across categories, the RSM ensured that each dataset had a representative balance of images. This reduced human bias in data selection and made the model training process more efficient. Given its utility in this project, the RSM tool could be employed in further research and machine learning tasks to enhance dataset creation and model performance. Additionally, the tool could be developed further to support advanced data augmentation techniques, improving the variety and richness of training datasets.

Although the dataset was generally well-balanced, there were some limitations that became evident during testing. For instance, the model misclassified firefighters as wildfire events in

the BoWFire dataset, indicating a need for more images of firefighters in non-wildfire situations. While the dataset could have been further enhanced with additional images, the model's overall performance was still commendable, demonstrating the effectiveness of the dataset and training process.

Regarding model selection, MobileNetV2 (alpha=0.1, 96x96p) was a solid choice given its lightweight architecture and the need for efficient performance on the Arduino Nano 33 BLE Sense. The model effectively balanced accuracy and computational demands, achieving competitive results compared to more complex models such as InceptionV3 or VGG19.

Nevertheless, the project could have explored more alternative lightweight architectures. Models such as SqueezeNet or super-lightweight variants of EfficientNet might have offered better performance, especially in terms of inference speed and resource usage. Given the importance of optimizing for low-power hardware, these alternative architectures could have yielded even better results. However, utilizing alternative models would have required custom optimization techniques, as Edge Impulse, at the time of writing, did not support other models for deployment using the EON-Compiler, a tool that effectively reduces Flash and Peak RAM usage, enabling the deployment of models that would otherwise exceed the hardware's limitations.

Another aspect of the project that worked well was the integration of the system on the Arduino Nano 33 BLE Sense, demonstrating the feasibility of real-time wildfire detection on embedded, low-power hardware. Edge Impulse Studio simplified the deployment process, allowing for a seamless transition from model training to hardware integration. The system performed in real-time, showcasing the practicality of machine learning at the edge. While the protective enclosure was a proof-of-concept design, it served its purpose during field testing. Future iterations could benefit from more robust and professionally manufactured enclosures for long-term field deployment.

When comparing the project's results to existing literature, it is clear that the system performed competitively, especially given the hardware constraints. The MobileNetV2 model, despite being lightweight, achieved accuracy levels comparable to much larger models such as VGG19 and InceptionV3. On the Corsican Fire Database, for example, the model achieved an accuracy of 96.13%, which is remarkable considering that InceptionV3, with around 20 times more parameters, produced similar results.

In conclusion, the project successfully demonstrated the potential for a low-power, embedded wildfire detection system using lightweight CNN models. The system's deployment on edge hardware, its protective enclosure, and the diversity of the dataset were significant achievements. While there were areas for improvement, the project's overall performance and potential for future development are promising. By addressing the identified limitations and exploring alternative approaches, future iterations of the system could achieve even higher accuracy and efficiency, making wildfire detection more reliable and scalable for deployment in high-risk areas.

Chapter 7

Conclusion and Future Work

This study aimed to develop and evaluate embedded wildfire detection algorithms tailored for the WUI. By creating a comprehensive and balanced dataset, training efficient CNN models, and deploying these models on a resource-constrained hardware platform, the Arduino Nano 33 BLE Sense in this work, the conducted research addressed the critical need for early and accurate wildfire detection. The conclusions drawn from this work encompass the dataset creation process, model training and evaluation, deployment considerations, and future directions.

7.1 Objectives Achieved

The research successfully met its primary objectives:

- **Dataset Creation:** A comprehensive and balanced dataset was developed, incorporating images from various sources to represent diverse wildfire scenarios, including a tool for managing images and balancing the dataset by upsampling.
- **Model Training:** Multiple CNN models were trained and evaluated, focusing on comparing their performance to select the best model.
- **Deployment:** The best-performing model was successfully deployed on resource-constrained hardware, the Arduino Nano 33 BLE Sense, demonstrating real-time wildfire detection capabilities.

7.1.1 Dataset Created

The creation of a new dataset was a vital component of this research. A diverse dataset was established by integrating images from multiple sources, including hand-captured photographs. This dataset was categorized to ensure a balanced representation of various wildfire scenarios, distinguishing between different times of the day, distances from the wildfire and between fire and smoke. This dataset forms a solid foundation for training accurate and generalizable models for wildfire detection.

7.1.2 Random Sample Maker (RSM)

To manage the images and balance the dataset, RSM was developed using Python. RSM automates the process of creating a balanced dataset by allowing users to generate a random subset of the dataset, extracting a fixed number of images from each category. This tool is essential for maintaining balance in the dataset, as manual selection could introduce bias. RSM supports cross-platform compatibility and includes functionalities for slightly augmenting images in under-represented classes. This technique includes horizontal flipping and slight rotation, which help increase the diversity and robustness of the dataset without manual intervention. These features ensure that the dataset remains balanced, which was one of the main focuses of the dataset creation process.

7.1.3 Models Trained

The training of CNN models was conducted on the Edge Impulse Studio platform, utilizing transfer learning principles. MobileNetV1 and MobileNetV2 were chosen as they were the only available on the platform for image classification tasks for the hardware constraints of the Arduino Nano 33 BLE Sense. Hyperparameters were finely tuned through iterative testing, focusing on optimizing accuracy while complying with hardware constraints. The primary focus of testing was to compare the performance of different models, assessing different metrics, mainly the global accuracy, true positive accuracy and AUC-ROC. Through this comparative analysis, the MobileNetV2 with an alpha of 0.1 using 96x96 pixel images emerged as the best-performing model, achieving a global accuracy of 91.4% and an AUC-ROC of 0.93.

7.1.4 Deployment of Models

Deployment of the trained models was carried out on the Arduino Nano 33 BLE Sense using the Edge Impulse platform. This involved compiling models with TFLite, EON-Compiler and a specific EON-Compiler optimized for RAM usage. The deployment process was seamless, facilitated by Edge Impulse's tools for exporting models as Arduino libraries, which were then integrated into the Arduino IDE for final deployment. The best-performing model was compiled using the EON-Compiler optimized for RAM usage.

7.1.5 Enclosure Build

An enclosure was designed and built using SolidWorks to protect the hardware from environmental factors and ensure reliable operation in the field. The design process focused on creating a robust, water-resistant, and easy-to-deploy solution.

7.2 Future Works

Future work will focus on several key aspects to enhance the efficacy and application scope of the wildfire detection models:

- **Increasing Dataset Size and Maintaining Balance:** Expanding the dataset with more diverse images, including those captured in various environmental conditions. Collaborating with organizations that have private collections of wildfire images can help achieve this. Additionally, separating the dataset into more specific categories such as urban, forest, and mountain scenarios could enhance the model's ability to generalize across different environments. Finally, increasing the number of images that include firefighters in non-wildfire scenarios could decrease the number of false positives seen during testing in those settings.
- **Expanding the Capabilities of the Random Sample Maker Tool:** Enhancing the RSM tool to automatically categorize images into different categories using color and line patterns, as well as machine learning techniques. This automation could eliminate the need for manual category assignment, making the dataset creation process more efficient and reducing human bias.
- **On-Site Training and Fine-Tuning:** Performing on-site training of the model with images captured over a full day could help the model adapt to specific locations and conditions, reducing false detections, such as mistaking sunsets for wildfires. Additionally, adjusting detection thresholds based on environmental conditions could further enhance accuracy.
- **Integration with IoT:** Integrating the detection system with IoT networks for real-time data collection and analysis.
- **Battery Solutions:** Using solar panels in combination with batteries could extend the continuous operational time without recharging.
- **Testing in Real-World Scenarios:** Conducting extensive field tests to validate model performance in real-world scenarios, ensuring that the models can effectively detect wildfires under various conditions.
- **Exploration of Other Architectures:** Investigating the use of other lightweight CNN architectures and advanced machine learning techniques such as SqueezeNet variations, which are not available through the Edge Impulse platform and may potentially improve detection accuracy.

In conclusion, the project successfully constructed a novel dataset and trained models for wildfire detection. Results indicate promising performance in terms of accuracy and deployment feasibility. Continued refinement and real-world evaluation are essential to optimize model effectiveness and reliability in wildfire mitigation.

Bibliography

- [1] United Nations. Sustainable development goals. <https://www.un.org/sustainabledevelopment/sustainable-development-goals/>, n.d.
- [2] António Bento-Gonçalves and António Vieira. Wildfires in the wildland-urban interface: Key concepts and evaluation methodologies. *Science of the total environment*, 707:135592, 2020.
- [3] Ankita Mohapatra and Timothy Trinh. Early wildfire detection technologies in practice—a review. *Sustainability*, 14(19):12270, 2022.
- [4] Robert S Allison, Joshua M Johnston, Gregory Craig, and Sion Jennings. Airborne optical and thermal remote sensing for wildfire detection and monitoring. *Sensors*, 16(8):1310, 2016.
- [5] Chiu Chun Chan, Sheeraz A Alvi, Xiangyun Zhou, Salman Durrani, Nicholas Wilson, and Marta Yebra. Iot ground sensing systems for early wildfire detection: Technologies, challenges and opportunities. *arXiv preprint arXiv:2312.10919*, 2023.
- [6] Ely Intorf Grace, Asplund and Droege. The osborne firefinder, 1955. Accessed: 2024-07-30.
- [7] Thou-Ho Chen, Ping-Hsueh Wu, and Yung-Chuen Chiou. An early fire-detection method based on image processing. In *2004 International Conference on Image Processing, 2004. ICIP'04.*, volume 3, pages 1707–1710. IEEE, 2004.
- [8] Wilfrid Schroeder. Nasa active fire product, 2024. Available at <https://viirsland.gsfc.nasa.gov/Products/NASA/FireESDR.html>, last accessed in May 2024.
- [9] Panagiotis Barmpoutis, Periklis Papaioannou, Kosmas Dimitropoulos, and Nikos Grammalidis. A review on early forest fire detection systems using optical remote sensing. *Sensors*, 20(22):6442, 2020.
- [10] Francesco Carta, Chiara Zidda, Martina Putzu, Daniele Loru, Matteo Anedda, and Daniele Giusto. Advancements in forest fire prevention: A comprehensive survey. *Sensors*, 23(14):6635, 2023.

- [11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.
- [12] Chunling Chen, Ziyue Zheng, Tongyu Xu, Shuang Guo, Shuai Feng, Weixiang Yao, and Yubin Lan. Yolo-based uav technology: A review of the research and its applications. *Drones*, 7(3):190, 2023.
- [13] Luís Miguel Ferreira, A Paulo Coimbra, and Aníbal T de Almeida. Autonomous system for wildfire and forest fire early detection and control. *Inventions*, 5(3):41, 2020.
- [14] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.
- [15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. Learning Rate: pp. 292-304.
- [16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. Overfitting and Underfitting: pp. 108-127.
- [17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. Convolutional Operation: pp. 332.
- [19] Abdelmalek Bouguettaya, Ahmed Kechida, and Amine Mohammed Taberkit. A survey on lightweight cnn-based object detection algorithms for platforms with limited computational resources. *International Journal of Informatics and Applied Mathematics*, 2(2):28–44, 2019.
- [20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.
- [21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [22] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018.
- [23] Keiron O’shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

- [24] Hao Wu, Qi Liu, and Xiaodong Liu. A review on deep learning approaches to image classification and object segmentation. *Computers, Materials & Continua*, 60(2), 2019.
- [25] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.
- [26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.
- [27] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.
- [28] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of field robotics*, 37(3):362–386, 2020.
- [29] GSDMA Sreenu and Saleem Durai. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *Journal of Big Data*, 6(1):1–27, 2019.
- [30] Harry A Pierson and Michael S Gashler. Deep learning in robotics: a review of recent research. *Advanced Robotics*, 31(16):821–835, 2017.
- [31] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.
- [32] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- [33] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
- [34] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.
- [35] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

- [36] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.
- [37] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.
- [38] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.
- [39] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3:1–40, 2016.
- [40] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [41] Pete Warden and Daniel Situnayake. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O’Reilly Media, 2019.
- [42] Youssef Abadade, Anas Temouden, Hatim Bamoumen, Nabil Benamar, Yousra Chtouki, and Abdelhakim Senhaji Hafid. A comprehensive survey on tinyml. *IEEE Access*, 2023.
- [43] Swapnil Sayan Saha, Sandeep Singh Sandha, and Mani Srivastava. Machine learning for microcontroller-class hardware: A review. *IEEE Sensors Journal*, 22(22):21362–21390, 2022.
- [44] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems*, 3:800–811, 2021.
- [45] Anshul Gaur, Abhishek Singh, Anuj Kumar, Ashok Kumar, and Kamal Kapoor. Video flame and smoke based fire detection algorithms: A literature review. *Fire technology*, 56:1943–1980, 2020.
- [46] Olga Dipuo Molaudzi and Samuel Adewale Adelabu. Review of the use of remote sensing for monitoring wildfire risk conditions to support fire risk assessment in protected areas. *South African Journal of Geomatics*, 7(3):222–242, 2018.
- [47] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

- [48] Jonathan S Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. A constructive prediction of the generalization error across scales. *arXiv preprint arXiv:1909.12673*, 2019.
- [49] BS Larsen. Synthetic minority over-sampling technique (smote). *GitHub* (https://github.com/dkbsl/matlab_smote/releases/tag/1.0), 2022.
- [50] Chris Drummond, Robert C Holte, et al. C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on learning from imbalanced datasets II*, volume 11, 2003.
- [51] Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pages 973–978. Lawrence Erlbaum Associates Ltd, 2001.
- [52] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.
- [53] Kiran Maharana, Surajit Mondal, and Bhushankumar Nemade. A review: Data pre-processing and data augmentation techniques. *Global Transitions Proceedings*, 3(1):91–99, 2022.
- [54] Thomas Fontanari, Tiago Comassetto Fróes, and Mariana Recamonde-Mendoza. Cross-validation strategies for balanced and imbalanced datasets. In João Carlos Xavier-Junior and Ricardo Araújo Rios, editors, *Intelligent Systems*, pages 626–640, Cham, 2022. Springer International Publishing.
- [55] Shawn Hymel, Colby Banbury, Daniel Situnayake, Alex Elum, Carl Ward, Mat Kelcey, Mathijs Baaijens, Mateusz Majchrzycki, Jenny Plunkett, David Tischler, et al. Edge impulse: An mlops platform for tiny machine learning. *arXiv preprint arXiv:2212.03332*, 2022.
- [56] Edge Impulse. Edge impulse eon tuner, 2024. Available at <https://docs.edgeimpulse.com/docs/edge-impulse-studio/eon-tuner>, last accessed in May 2024.
- [57] Edge AI and Vision Alliance. 2021 edge ai and vision product of the year award winner showcase: Edge impulse (edge ai developer tools), 2021. Available at <https://www.edge-ai-vision.com/2021/05/2021-edge-ai-and-vision-product-of-the-year-award-winner-showcase-edge-imp> last accessed in May 2024.
- [58] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE Access*, 8:85714–85728, 2020.
- [59] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

- [60] Franklin Oliveira, Daniel G Costa, Flávio Assis, and Ivanovitch Silva. Internet of intelligent things: A convergence of embedded systems, edge computing and machine learning. *Internet of Things*, page 101153, 2024.
- [61] Partha Pratim Ray. A review on tinyml: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1595–1623, 2022.
- [62] Pedro Vinícius AB de Venâncio, Adriano C Lisboa, and Adriano V Barbosa. An automatic fire detection system based on deep convolutional neural networks for low-power, resource-constrained devices. *Neural Computing and Applications*, 34(18):15349–15368, 2022.
- [63] Pedro Vinícius AB de Venâncio, Adriano C Lisboa, and Adriano V Barbosa. D-fire dataset. <https://github.com/gaiasd/DFireDataset>. visited on 2024-07-25.
- [64] ai. ff dataset. <https://universe.roboflow.com/ai-ldmcz/ff-zmcwi>, oct 2023. visited on 2024-07-05.
- [65] Stocksnap, 2024. Available at <https://stocksnap.io/>, last accessed in May 2024.
- [66] Picryl: Public domain search similarity engine, 2024. Available at <https://picryl.com/>, last accessed in May 2024.
- [67] Pexels: Stock photos, 2024. Available at <https://www.pexels.com/>, last accessed in May 2024.
- [68] Duplicate photo cleaner, 2024. Available at <https://www.duplicatephotocleaner.com/>, last accessed in May 2024.
- [69] Jeffrey A. Clark. Python imaging library (fork by jeffrey a. clark), 2024. Available at <https://pypi.org/project/pillow/>, last accessed in May 2024.
- [70] Charlie Clark. A python library to read/write excel 2010 xlsx/xlsm files, 2024. Available at <https://pypi.org/project/openpyxl/>, last accessed in May 2024.
- [71] standard python interface to the tcl/tk gui toolkit, 2024. Available at <https://docs.python.org/3/library/tkinter.html>, last accessed in May 2024.
- [72] Johannes Buchner. Image hashing library for python, 2024. Available at <https://pypi.org/project/ImageHash/>, last accessed in July 2024.
- [73] Opencv, 2024. Available at <https://pypi.org/project/opencv-python/>, last accessed in May 2024.
- [74] Rodrigo S. Comba. Github: Random sample maker, 2024. Available at <https://github.com/rodrigocomba/Random-Sample-Maker/tree/main>, last accessed in July 2024.

- [75] Edmundo Casas, Leo Ramos, Eduardo Bendek, and Francklin Rivas-Echeverría. Assessing the effectiveness of yolo architectures for smoke and wildfire detection. *IEEE Access*, 11:96554–96583, 2023.
- [76] Yazan Al-Smadi, Mohammad Alauthman, Ahmad Al-Qerem, Amjad Aldweesh, Ruzayn Quaddoura, Faisal Aburub, Khalid Mansour, and Tareq Alhmiedat. Early wildfire smoke detection using different yolo models. *Machines*, 11(2):246, 2023.
- [77] T. Toulouse, L. Rossi, A. Campana, T. Celik, and M. Akhloufi. Computer vision for wild-fire research: an evolving image dataset for processing and analysis. *Fire Safety Journal*, 92:188–194, 2017.
- [78] Ali Khan, Bilal Hassan, Somaiya Khan, Ramsha Ahmed, and Adnan Abuassba. Deepfire: A novel dataset and deep transfer learning benchmark for forest fire detection. *Mobile Information Systems*, 2022(1):5358359, 2022.
- [79] Daniel YT Chino, Letricia PS Avalhais, Jose F Rodrigues, and Agma JM Traina. Bowfire: detection of fire in still images by integrating pixel color and texture analysis. In *2015 28th SIBGRAPI conference on graphics, patterns and images*, pages 95–102. IEEE, 2015.
- [80] Wamiq Raza, Anas Osman, Francesco Ferrini, and Francesco De Natale. Energy-efficient inference on the edge exploiting tinyml capabilities for uavs. *Drones*, 5(4):127, 2021.
- [81] Arduino. Arduino tiny machine learning kit. <https://store.arduino.cc/products/arduino-tiny-machine-learning-kit>, 2024. Accessed: 2024-07-30.
- [82] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.
- [83] Maria Joao Sousa, Alexandra Moutinho, and Miguel Almeida. Wildfire detection using transfer learning on augmented datasets. *Expert Systems with Applications*, 142:112975, 2020.
- [84] Eunchanws. Firesmoke object detection dataset. <https://universe.roboflow.com/eunchanws/fire-smoke-object-detection>, dec 2022. visited on 2024-07-25.
- [85] Fire Detection. Fire data annotations dataset. <https://universe.roboflow.com/fire-detection/fire-data-annotations>, jul 2022. visited on 2024-07-25.
- [86] GFO. fire and smoke detector dataset. <https://universe.roboflow.com/gfo-fclde/fire-and-smoke-detector-yykya>, jul 2024. visited on 2024-07-25.
- [87] Heba Saker Ahmed Shaheen Ahmed Gamaleldin, Ahmed Atef. Fire dataset. <https://www.kaggle.com/datasets/phylake1337/fire-dataset/data>. visited on 2024-07-25.

- [88] S. Khan R. Ahmed A. Khan, B. Hassan and A. Adnan. Forest fire dataset. <https://www.kaggle.com/datasets/alik05/forest-fire-dataset>. visited on 2024-07-25.
- [89] Subhransu Maji Rashmi Tonge and C.V. Jawahar. Parsing world's skylines with shape constrained mrfs. In *Computer Vision and Pattern Recognition*, 2014.
- [90] Ivan Skorokhodov, Grigorii Sotnikov, and Mohamed Elhoseiny. Aligning latent and image spaces to connect the unconnectable. *arXiv preprint arXiv:2104.06954*, 2021.
- [91] Landscapes dataset lhq 1024x1024. <https://www.kaggle.com/datasets/dimensi0n/lhq-1024>. visited on 2024-07-25.
- [92] Yolo for tree detection. Tree detection by season / type dataset. <https://universe.roboflow.com/yolo-for-tree-detection/tree-detection-by-season-type>, mar 2024. visited on 2024-07-25.
- [93] mytest. Fire smoke detection dataset. <https://universe.roboflow.com/mytest-blult/fire-smoke-detection-eozii>, sep 2023. visited on 2024-07-26.
- [94] sdsd. big dataset. <https://universe.roboflow.com/sdsd-1wvfn/big-qhege>, apr 2022. visited on 2024-07-25.
- [95] . Lights dataset. <https://universe.roboflow.com/-vkjhe/lights-euqp0>, nov 2021. visited on 2024-07-25.
- [96] Universitas Widya Dharma pontianak. Exdark dataset(custom image) dataset. <https://universe.roboflow.com/universitas-widya-dharma-pontianak-xqoqm/exdark-dataset-custom-image>, nov 2022. visited on 2024-07-25.
- [97] APM. Day-or-night-classfication dataset. <https://universe.roboflow.com/apm/day-or-night-classfication>, oct 2023. visited on 2024-07-25.
- [98] Jeffrey A. Clark. Wildfire observers and smoke recognition image database), 2024. Available at http://wildfire.fesb.hr/index.php?option=com_content&view=article&id=49&Itemid=54, last accessed in May 2024.
- [99] Andy Dunnings. Fire image data set for dunnings 2018 study - png still image set. <https://collections.durham.ac.uk/files/r2d217qp536>, 2018. Accessed: May, 2024.
- [100] Burak Kizilkaya, Enver Ever, Hakan Yekta Yatbaz, and Adnan Yazici. An effective forest fire detection framework using heterogeneous wireless multimedia sensor networks. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 18(2):1–21, 2022.

Appendix A

A.1 Batch File

```
1 @echo off
2 setlocal enabledelayedexpansion
3
4 cd /d "%~dp0"
5
6 echo Current directory: %cd%
7 echo Listing files in the current directory:
8 dir
9
10 for %%f in (arguments_*.txt) do (
11     set args=
12     if not exist "%%~f" (
13         echo The file %%~f cannot be found.
14         pause
15         exit /b
16     )
17
18     for /F "usebackq delims=" %%i in ("%%~f") do (
19         echo Argument: %%i
20         set args=!args! %%i
21     )
22
23     echo Final constructed arguments: !args!
24     echo Running: python randomnessmaker.py !args!
25     call python randomnessmaker.py !args!
26 )
27
28 pause
29 endlocal
```

Listing A.1: Batch File (.bat) used with *RSM*'s CLI feature

A.2 Dataset Folder Structure

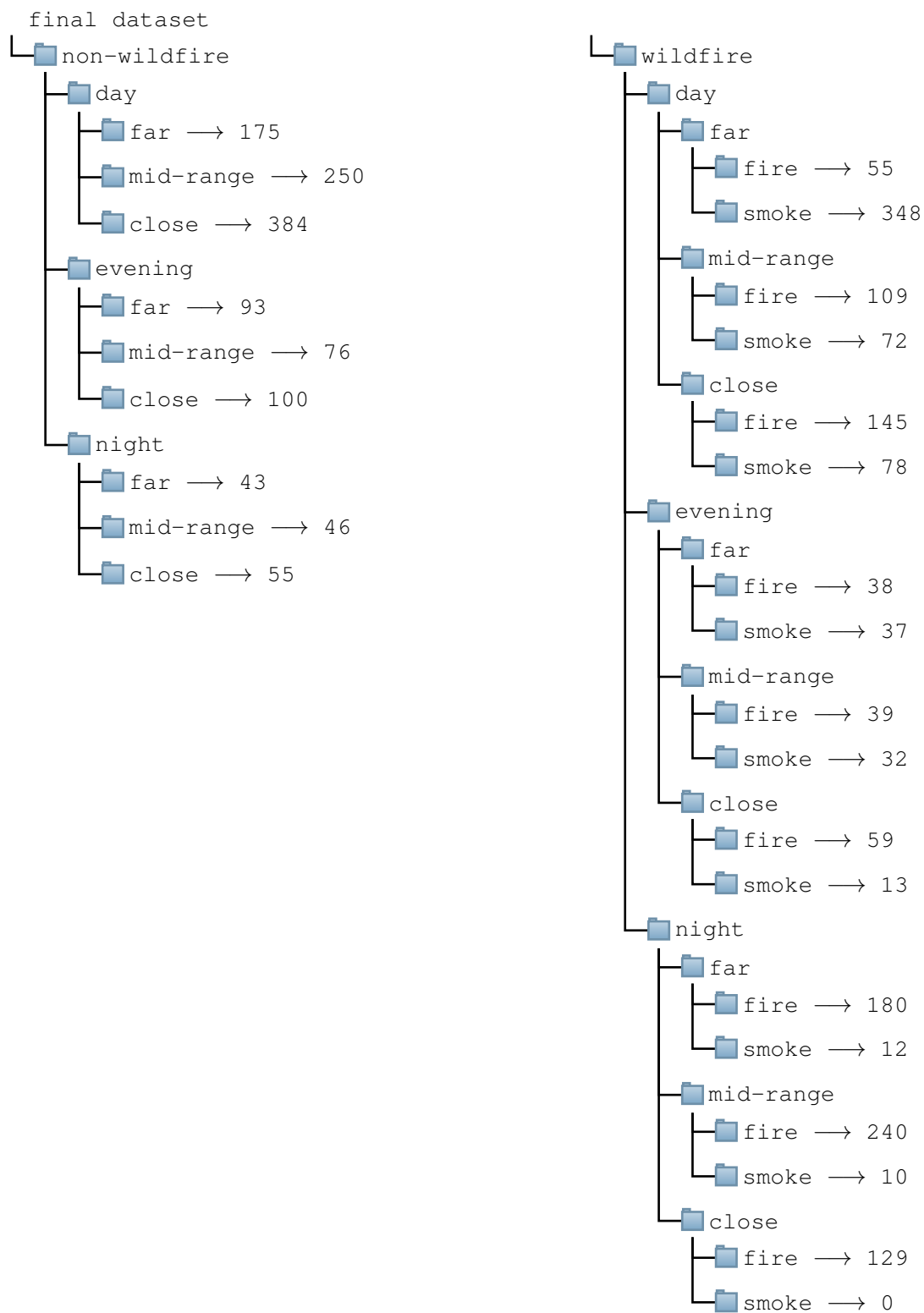


Figure A.1: Final dataset: wildfire and non-wildfire folder structures

A.3 On-Device Performance Metrics

Table A.1: Testing different models on different sized images (NxN pixels) using different methods for compiling the model: TFlite (TFL), EON-Compiler (EC) and EON-Compiler RAM Optimised (ECR), to obtain Inferencing Time (IT), Peak RAM Usage (PRU) and Flash Usage (FU). Models selected for testing are displayed in yellow background colour.

		MobileNetV1			MobileNetV2				
		alpha = 0.25	alpha = 0.5	alpha = 0.75	alpha=0.05	alpha = 0.1	alpha = 0.35	alpha = 0.5	
	IT	173ms	610ms	1701ms	105ms	124ms	280ms	455ms	32
	PRU	47.8k	83.7k	119.6k	80.9k	86.7k	118.6k	207.8k	x
	FU	351.1k	1012.4k	2.0M	220.4k	275.1k	664.2k	906.0k	32
TFL	IT	1222ms	4180ms	-	902ms	983ms	1979ms	-	96
	PRU	126.1k	239.9k	-	315.2k	320.9k	352.8k	-	x
	FU	356.0k	1017.3k	-	241.8k	296.5k	685.7k	-	96
	IT	2152ms	-	-	1588ms	1731ms	-	-	128
	PRU	193.9k	-	-	518.6k	524.3k	-	-	x
	FU	356.0k	-	-	259.3k	314.0k	-	-	128
	IT	4463ms	-	-	-	-	-	-	160
	PRU	280.9k	-	-	-	-	-	-	x
	FU	356.0k	-	-	-	-	-	-	160
EC	IT	173ms	610ms	1701ms	105ms	124ms	280ms	-	32
	PRU	58.6k	110.3k	162.0k	75.7k	85.6k	139.4k	-	x
	FU	299.1k	939.0k	1.9M	153.0k	203.0k	565.9k	-	32
	IT	1222ms	4180ms	-	902ms	983ms	1979ms	-	96
	PRU	124.8k	242.4k	-	270.8k	280.7k	334.5k	-	x
	FU	303.6k	943.5k	-	174.1k	224.0k	586.9k	-	96
	IT	2152ms	-	-	1588ms	1731ms	-	-	128
	PRU	181.3k	-	-	440.3k	450.2k	-	-	x
	FU	303.6k	-	-	181.6k	241.5k	-	-	128
EC-R	IT	4463ms	-	-	-	-	-	-	160
	PRU	253.8k	-	-	-	-	-	-	x
	FU	303.6k	-	-	-	-	-	-	160
	IT	1222ms	4180ms	-	902ms	983ms	1979ms	-	96
	PRU	134.2k	225.3k	-	159.1k	169.0k	232.7k	-	x
	FU	317.9k	957.7	-	186.3k	236.3k	598.8k	-	96
	IT	2152ms	-	-	1588ms	1731ms	-	-	128
	PRU	178.9k	-	-	232.5k	242.5k	-	-	x
	FU	317.9k	-	-	203.8k	255.8k	-	-	128
EC-R	IT	4463ms	-	-	-	-	-	-	160
	PRU	235.0k	-	-	-	-	-	-	x
	FU	317.9k	-	-	-	-	-	-	160

A.3.1 Dataset References

Table A.2: List of datasets and sources used and respective references

Dataset or Source Name	Reference
D-Fire Dataset	[62, 63]
Fire&Smoke Object detection	[84]
Fire Data Annotations	[85]
fire and smoke detector	[86]
fire and smoke	[86]
FIRE Dataset	[87]
DeepFire Dataset	[78, 88]
Skylines 12	[89]
Landscapes dataset	[90], [91]
Tree detection by season	[92]
ff Dataset	[64]
Fire & Smoke Detection	[93]
big Dataset	[94]
Lights	[95]
ExDark Dataset	[96]
Day-Or-Night-Classification	[97]
Wildfire Observers and Smoke Recognition Image Database	[98]
Fire Image Data Set for Dunnings 2018 study	[99]
Effective Forest Fire Detection Dataset for Heterogeneous Wireless Multimedia Sensor Networks	[100]
Picryl	[66]
Pexels	[67]
StockSnap	[65]

A.3.2 Output Excel File from *RSM* tool

Table A.3: Number of images in each folder

	A	B	C	D	E
1	Non-Wildfire	0			
2		Day	0		
3			Far	175	
4			Medium	250	
5			Close	384	
6		Evening	0		
7			Far	93	
8			Medium	76	
9			Close	100	
10		Night	0		
11			Far	43	
12			Medium	46	
13			Close	55	
14	Wildfire	0			
15		Day	0		
16			Far	0	
17				Fire	55
18				Smoke	348
19			Medium	0	
20				Fire	109
21				Smoke	72
22			Close	0	
23				Fire	145
24				Smoke	78
25		Evening	0		
26			Far	0	
27				Fire	38
28				Smoke	37
29			Medium	0	
30				Fire	39
31				Smoke	32
32			Close	0	
33				Fire	59
34				Smoke	13
35		Night	0		
36			Far	0	
37				Fire	180
38				Smoke	12
39			Medium	0	
40				Fire	240
41				Smoke	10
42			Close	0	
43				Fire	129
44				Smoke	0

A.4 Training Hyperparameters for Different Numbers of Neurons

(b)	MobileNetV1 128x128 Image Size 0.1 Dropout Rate 32 Batch Size 80/20% Training/Validation Split		
	#Neurons	#Training Cycles	Learning Rate
	0	10	0.001
	2	20	0.001
	4	10	0.001
	8	10	0.002
	16	10	0.002
	32	10	0.002
	64	10	0.001
(c)	MobileNetV2 96x96 Image Size 0.1 Dropout Rate 32 Batch Size 80/20% Training/Validation Split		
	#Neurons	#Training Cycles	Learning Rate
	0	50	0.0001
	2	50	0.0005
	4	50	0.0005
	8	10	0.001
	16	50	0.0001
	32	8	0.0003
	64	3	0.001
(c)	MobileNetV2 32x32 Image Size 0.1 Dropout Rate 32 Batch Size 80/20% Training/Validation Split		
	#Neurons	#Training Cycles	Learning Rate
	0	25	0.0005
	2	10	0.0005
	4	10	0.0005
	8	20	0.001
	16	20	0.001
	32	20	0.001
	64	20	0.001
	128	20	0.0005

Figure A.2: Tuned training hyperparameters for different numbers of neurons