POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Automated Optimization of DNN Models for People Counting with Infrared Sensors Using NAS

Supervisors

Prof. Daniele JAHIER PAGLIARI

Dr. Chen XIE

Dr. Matteo RISSO

Dr. Francesco DAGHERO

Dr. Alessio BURRELLO

Candidate

Seyed Morteza MOLLAEI

Academic Year 2022-2023

October 2023

Summary

The growing popularity of Neural Architecture Search (NAS) is changing optimization strategies in Deep Learning (DL). While NAS has typically been utilized for tasks such as image classification and object detection, this study investigates its application in improving DL models for people counting applications based on ultra-low-resolution infrared (IR) array sensors. These sensors are well-known for their low cost, energy efficiency, and privacy protection, given by the fact that they only gather low-resolution thermal maps, not revealing private information such as facial details of people. This makes them ideal for applications such as people counting in public spaces.

In our work, we perform a thorough examination of several DL architectures, using them as seed models for an efficient NAS tool, with the goal of finding several tradeoff points between accuracy and model size, both critical for low-power devices. The core of this study is the development of a new and purpose-driven NAS paradigm targeted at optimizing architectural characteristics directly related to worldwide usage of ultra-low-resolution IR array sensors.

Thanks to NAS technique, our findings show that DL models not only improve accuracy but also efficiency, even with simple model architectures. In particular, we are able to achieve up to 85.55% balanced accuracy on an open-source infrared dataset, improving by +2.85% with respect to a previous work that applied manually designed DL models on the same dataset, while also reducing the model memory by 44%. Importantly, these enhanced DL models can function in real time on low-power Internet of Things (IoT) nodes.

Acknowledgements

I would like to express my deepest appreciation to everyone who helped me make this research project feasible. My heartfelt gratitude goes to all my supervisors specially Prof. Pagliari for their important advice, unshakable support, and insightful feedback that lighted the way for me along my trip.

I am grateful to the volunteers who willingly offered their time and skills, allowing me to collect the necessary data and insights for this project. Their contributions constitute the foundation for my study.

Last but not least, I want to thank my family, friends, and loved ones for their unfailing encouragement, patience, and understanding, which have served as pillars of strength during this difficult yet gratifying journey.

Table of Contents

List of Tables

List of Figures

Acronyms

DL
Deep Learning

IoT
Internet of Things

MCU
Microcontroller

NAS
Neural Architecture Search

LNAS
Lightweight NAS

CNN
Convolutional Neural Network

PIT
Pruning In Time

ML
Machine Learning

AI
Artificial Intelligent

FPS
Frames Per Second

LSTM
Long-Short Term Memory

GRU
Gated Recurrent Unit

IR
Infrared

KNN
K-Nearest Neighbors

SVM
Support Vector Machines

RF
Random Forest

FNN
Feedforward Neural Network

CPU
Central Processing Unit

TCN
Temporal Convolutional Network

FC
Fully Connected

FLOP
Floating Point Operation

DNAS
Differentiable Neural Architecture Search

LINAIGE
Low-resolution INfrared-array data for AI on the edGE

MSE
Mean Square Error

MAE
Mean Absolute Error

Chapter 1
Introduction

Recently, there has been a spike in demand for the use of Deep Learning (DL) in the Internet of Things (IoT) world due to its remarkable performances in a wide range of IoT applications, such as embedded computer vision and time series forecasting [1], [2], [3], [4], [5]. Although, cloud-based computing centralizes data processing in remote data centers, allowing for scalability but occasionally resulting in latency issue, edge computing moves processing closer to data sources, performing the inference of the DL model directly on the IoT device [1], [6], [7].

Dropping the need for a constant connection or simply reducing the amount of data transmitted can bring several benefits. First, increased security, as no sensitive data is transmitted over a possibly insecure connection [1], [6]. Second, predictable and possibly lower latencies, leading to real-time applications being available on IoT end-nodes. Finally, lower energy footprints, as the data transmission is generally less optimized than performing computations locally. This last point is crucial for battery-operated IoT devices, that have to be operational for as long as possible.

However, while running the inference on IoT nodes can lead to several benefits, DL models are generally designed to run on powerful high-end hardware. These algorithms consume a lot of energy and need a lot of processing power, making them unsuitable for deployment on IoT nodes. These nodes, which are often powered by batteries and have limited memory and resources, rely on Microcontrollers (MCUs) to work properly. It is critical to choose appropriate models and hyper-parameters for successful DL deployment in energy-constrained IoT edge devices, a complex and time-consuming challenge due to the constrained hardware of MCUs.

Neural Architecture Search (NAS) has become increasingly popular, performing in an automatic process of the search for the optimal architectures and determining ideal network layouts. Several of these NAS approaches, in particular, have been specially optimized for edge devices, where resource restrictions are much more severe [8], [9], [10], [11], [12]. The importance of

using NAS tools derives from their capacity to efficiently explore a large search space of potential network configurations which enables researchers and engineers to focus on other areas of the machine learning pipeline. Existing NAS tools for 2D CNNs frequently employ a coarse-grained approach, making multiple copies of network layers for different designs, resulting in excessive memory and time consumption [12]. Lightweight NAS (LNAS) solutions, on the other hand, provide more efficient exploration of architectural space by focusing on certain model properties, such as channel numbers in layers, which are critical for computer vision [9], [10].

In this work, we leverage a lightweight NAS approach named Pruning In Time (PIT) [13], utilizing a single seed model as a basis to generate a diverse array of Pareto optimal architectures, achieving a finest balance between the number of operations/parameters and accuracy. The approach employs use of structural weight pruning, which involves increasing weights with trainable masks that reflect architectural settings. During training, these masks are improved using regularizers to reduce model complexity while retaining accuracy. Two inner regularizers collaborate: one decreases parameters and the other performs inference methods. This facilitates Neural Architecture Search (NAS), successfully revealing various Pareto architectures.

In particular, we apply PIT for an increasingly popular task of IoT applications, that is, people counting. The importance of people counting arises from its vast range of uses in public safety, urban planning, and commercial situations [14]. The practical applications of this technology range from monitoring the occupancy levels of indoor workspaces, museums, and hospitals to conducting in-depth assessments of people flow data at the doors of stores, supermarkets, and other public locations. Furthermore, in the context of the COVID-19 pandemic, people counting technology has played a critical role in monitoring and implementing social distancing standards and safety norms [15], [16], [17].

In the realm of Internet of Things (IoT), there is a vast variety of technical solutions for people counting, which may be generally classified into two primary types: instrumented and un-instrumented techniques [18]. The former makes use of transceivers that are already present in devices that users own or are given, such as smartphones, smartwatches, or tags [19]. However, since they rely on voluntary contributions and specialized technology, these techniques have substantial constraints that make them difficult to deploy in many real-world settings, particularly in public spaces. Un-instrumented solutions, on the other

hand, do not rely on person active participation and instead rely on external sensors such as proximity sensors, optical cameras, and infrared arrays [18], [20], [21], [22]. In particular, Infrared beam sensors and passive infrared sensors are the least expensive and easiest to deploy, however they have significant limitations. They rely on object motion and may fail to discriminate between many people in close proximity [23]. However, thanks to the significant improvement in computer vision and video analysis brought by Deep Learning (DL) models, these technologies are becoming increasingly appealing. Many of today vision-based techniques use optical cameras and Machine Learning (ML) algorithms to detect and locate people inside each frame [24], [25], [26]. While these technologies have shown to be efficient, they create serious privacy issues since they acquire and process sensitive information about individuals, such as face features and physical characteristics.

In this case, the use of low-resolution infrared (IR) array sensors is a viable choice that offers various benefits. These sensors are capable of capturing small thermal pictures (often 8x8 or 16x16 pixels) at around 10 frames per second (FPS) [27]. While they can detect basic forms, they do not record details such as facial characteristics, clothes, or hairstyles, allowing them to be used in a private setting. Furthermore, their low power consumption, low-cost, and low-resolution make them ideal in an IoT setting, where battery-powered devices need to be operational for long periods of time [13], [28], [29], [30], [31].

In this work, we benchmark several DL architectures on an infrared dataset with a resolution of 8x8. In particular, we select 8 state-of-the-art models on the task, using them as seed model for the NAS to generate a wide set of models, spawning accuracies between 81.38% and 92.74% and number of parameters between 575 and 18124. We show that NAS approaches are suitable also for IR data and can outperform hand-crafted models by up to 2.85%, while reducing the memory by 44%.

Chapter 2
Related Works

In this chapter, we present a review of the literature on people counting with multi-pixel infrared arrays. Table 2.1 highlights several studies, including sensor model, resolution, location, target dataset, counting methods, and IoT devices utilized for deployment. Previous works integrating low-resolution infrared sensors with Machine Learning (ML) have mostly focused on human activity identification [32], [33], [34], [35], [36], [37], [38], [39], [40], [41]. Some of these studies use traditional algorithms [32] - [35] while others offer Deep Learning (DL) methodologies such as Convolutional Neural Networks (CNNs), Long-Short Term Memory (LSTMs), Gated Recurrent Units (GRUs), or a mix of these (e.g., CNN-LSTM) [36] - [41]. The number and location of IR sensors utilized, the preprocessing methods performed on thermal pictures before feeding them to ML models, and the identified activities change depending on the work. These activities vary from everyday behaviors like walking, sitting, and standing identification [32], [34], [36], [38], [40] to more particular scenarios like detecting falls in older people [33], [35] epilepsy-induced convulsions [41], and even yoga position recognition [37]. As far as we know, the only work available for counting people using low-resolution IR sensors (8x8 pixels) is the method put forward in reference [42], [43].

Some studies used deterministic algorithms [17], [42], [44], [45], [46], [47] traditional ML models [48], or deep learning [30], [31], [49], [50]. For instance, in [17], the authors described a real-time pattern recognition method that uses data from low-resolution IR array sensors placed on entrances to detect the number of persons in a room. Similarly, [44] used a doorway sensor in conjunction with body extraction and localization algorithms, as well as the background assessment for people counting. [45] suggested a lightweight deterministic approach that uses a single array sensor mounted on a door to record item trajectories entering and departing a room, allowing calculation of the number of people. These techniques are intriguing because they make use of a single, low-resolution sensor but are confined to counting individuals entering and departing a room through a doorway, solving a simplified version of the generic people counting problem.

Table 2.1
Advance People Counting Methods Based on Infrared Arrays

Work	Sensor	Positioning	Dataset	Algorithm	Deployment Target
Perra et al. [17]	Grid EYE (8x8)	Door	Private	Deterministic	Z-Uno
Mohammadmoradi et al. [44]	Grid EYE (8x8)	Door	Private	Deterministic	Raspberry Pi Zero
Wang et al. [45]	MLX90641 (12x16)	Door	Private	Deterministic	ESP8266
Rabiee et al. [46]	Grid EYE (8x8)	Ceiling	Private/Nagoya-OMRON Dataset [29]	Deterministic	-
Singh et al. [47]	MLX90621 (16x4)	Ceiling/Side Wall	Private	Deterministic	Arduino Uno
Chidurala et al. [48]	Grid EYE (8x8) MLX90640 (32x24) Lepton (80x60)	Ceiling	Private	Naive Bayes KNN SVM RF	Raspberry Pi 3
Bouazizi et al. [49]	MLX90640 (32x24)	Ceiling	Private	CNN	Raspberry Pi 3
Gomez et al. [30]	Lepton (80x60)	Wall	Private	CNN	NXP LPC54102
Metwaly et al. [31]	MLX90640 (32x24)	Ceiling	Private	FNN CNN GRU	STM32F4/F7
Kraft et al. [50]	MLX90640 (32x24)	Ceiling	Thermo Presence [24]	CNN	Raspberry Pi 4
Xie et al. [16]	Grid EYE (8x8)	Ceiling	LINAIGE [31]	CNN (2 variants)	STM32L4
Xie et al. [51]	Grid EYE (8x8)	Ceiling	LINAIGE [31]	Wake-up Trigger + CNN	STM32L4
Xie et al. [43]	Grid EYE (8x8)	Ceiling	LINAIGE [31]	CNN (4 variants) CNN-LSTM CNN-TCN	STM32L4
This work	Grid EYE (8x8)	Ceiling	LINAIGE [31]	CNN (3 variants) CNN-TCN	-

(*) All of these entries relate to the deployment of the approach described in [42] in [43].

In [42], the authors present a more general deterministic technique with a ceiling-mounted sensor. Smoothing, linear interpolation, and hot area labeling and clustering are used to identify moving thermal objects from the background in this approach. Following that, each recognized thermal item is subjected to a threshold-based human detection to determine if it represents a person. To preserve accuracy, the reference backdrop image is regularly updated to continually filter out stationary heated objects.

Furthermore, researchers have looked into multi-sensor deterministic techniques. A people counting technique, for example, was described in [46] for effectively tracking population in smart constructions. Several low-resolution sensors strategically positioned at connecting points between different regions of the building are used by the algorithm to count the number of persons travelling between neighboring zones. Similarly, [47] presents a system for indoor people counting based on two deterministic algorithms. This approach entails the placement of three 16x4 thermal sensors at various sites, each sensor is orientated in the x, y, and z axes, respectively. The combination of these sensors provides precise and dependable indoor people counting.

The authors of [48] investigates people counting using three infrared (IR) arrays attached to the ceiling, each with a different resolution (8x8, 32x24, and 80x60). Various preprocessing and feature extraction approaches are used in the study, including active pixel and active frame detection, connected components analysis, and statistical features. To compare the performance, they use several Machine Learning models, such as Naive Bayes, K-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Random Forests (RFs). The RFs produces the greatest score for the 8x8-resolution array based on their results from a private dataset.

Various deep learning (DL) algorithms for indoor people location and counting utilizing infrared (IR) sensors have been developed recently. In [49], for example, the authors propose a method based on a Convolutional Neural Network (CNN) with 9 convolutional layers and 1 dense layer. This CNN analyzes data from a 32x24 pixel IR sensor positioned on the ceiling. Notably, they propose the option of collecting lower-resolution samples (down to 8x6 pixels) with the use of a separate 8-layer CNN for frame upscaling to decrease sensor costs.

In [30], another people counting algorithm is introduced. This method relies on wall-mounted sensors and leverages a small-sized CNN model designed particularly for a low-memory, low-power platform. In this scenario, the IR array has a greater resolution of 80x60 pixels. Similarly, in reference [31], the authors investigate several DL approaches for indoor occupancy estimates, such as Feedforward Neural Networks (FNNs), CNNs, and Gated Recurrent Units (GRU). Their investigations are based on a ceiling-mounted infrared array with a resolution of 24x32 pixels. In contrast, [50] employs a ceiling-mounted IR array with the same resolution (24x32 pixels) and a U-Net-inspired encoder-decoder

CNN architecture. This method effectively recreates people positions in the recorded frame.

In [16], another DL algorithm based on an ultra-low-resolution (8x8) array is proposed. It should be noted, however, this study focused on a reduced form of the people counting problem. The major goal was to determine whether the sensor region contained two or more persons, especially in the context of social distance monitoring to limit COVID-19 spread. A similar method was used in a prior study [51], where an additional deterministic wake-up-trigger was included to prevent extra CNN invocations when no people were present in the frame. This change considerably lowered the overall energy usage of the system.

Existing data-driven (ML or DL) research for IoT end-nodes using low-resolution infrared arrays are scarce. For example, studies [48] and [49] focus primarily on person counting deployment on high-end mobile Central Processing Units (CPUs), but fail to give extensive information on critical deployment features such as memory utilization, inference delay, and model utilization of energy. Second, [30] and [31] emphasize the use of expensive and power-intensive high-resolution arrays, which may compromise user privacy. Furthermore, [49] supports low-resolution sensors but depends on an additional CNN model for frame upscaling, increasing total inference complexity. In addition, publications such as [30], [31] and [48] use unreasonable data splitting approaches, such as arbitrarily choosing individual frames or sliding windows, which improperly simplifies the process. Only [49] uses a reasonable data separation at the session level, on the other hand, both [16] and [51] concentrate on a simpler task variation. In [43], the authors study the new application of Deep Learning (DL) approaches to a people counting task utilizing a single, ceiling-mounted, ultra-low-resolution IR array with just 8x8 pixels. The results of an exhaustive investigation of six families of efficient DL models and various hyper-parameter settings show that DL not only surpasses deterministic algorithms in regards to counting accuracy, but also in terms of energy consumption and latency.

In this work, we also utilized a single, ceiling-mounted, ultra-low resolution IR array with just 8x8 pixels data, [43], as the sensor configuration in the investigation. Our main objective is to study several deep learning (DL) models for people counting applying Neural Architecture Search (NAS) approach presented in [5] with an extensive search of hyper-parameter space.

Chapter 3

Background

3.1 Overview

Deep Learning has become increasingly popular due to its outstanding performances in a wide range of fields, such as Computer Vision and Speech Recognition. As opposed to traditional Machine Learning it does not rely on complex hand-crafted features, but instead extracts them automatically directly from the data. An advantage of DL lies in its capability to learn non-linear functions, allowing it to possibly approximate any function. In contrast, however, it usually requires large amounts of data, often underperforming for datasets of limited size. In order to further improve the performance of DL models, several architectures have been proposed, with the most popular being:

- *Feed-Forward Neural Networks (FNNs)*: also known as Multilayer Perceptrons (MLPs), are a basic type of artificial neural network. Data in FNNs flows unidirectionally from input to output layers, with no feedback loops. They are adaptable and commonly used for classification and regression applications.
- *Convolutional Neural Networks (CNNs)*: they are designed to analyze grid-like data, such as photos and movies. They use convolutional layers to learn spatial feature hierarchies automatically, making them crucial for tasks like image classification and object recognition.
- *Recurrent Neural Networks (RNNs)*: are specialized for sequential data, such as time series and natural language processing, where the sequence of items is important. RNNs use recurrent connections to allow information to propagate across time steps. They are crucial in applications such as language modeling and speech recognition since they excel at capturing temporal relationships.

While these three neural network architectures are the fundamental building blocks of deep learning, it is worth noting that several other specialized architectures and hybrid models have arisen to address specific tasks.

In the context of the learning process, Machine Learning (and thus DL as well) can be divided into three main categories: *supervised*, *unsupervised*, and *reinforcement* learning.

Supervised learning, that is used in this work, is a fundamental machine learning approach that is widely employed in a variety of applications. An algorithm is trained on a labeled dataset, with each data point consisting of input characteristics and associated output labels or target values. The basic purpose of the algorithm is to learn a mapping from the input characteristics to the output labels, allowing it to make predictions or classifications on new, previously unseen data.

Unsupervised learning is another type of a machine learning paradigm in which the algorithm learns patterns and structures in data without explicit supervision or labeled outputs. Unsupervised learning algorithms, rather than producing predictions, seek to find intrinsic patterns, groups, or correlations in data. Unsupervised learning is commonly used for clustering and dimensionality reduction.

Finally, *Reinforcement* learning is a sort of machine learning in which an agent learns to make decisions in a given environment in order to maximize a cumulative reward. In this context, there is no explicit dataset of labeled samples. Instead, the agent interacts with the environment, performs actions, receives feedback in the form of incentives or penalties, and gradually learns to optimize its activities. Reinforcement learning is frequently employed in applications such as game play and autonomous robots.

3.2 Neural Network Layers

Deep Neural Networks (DNNs) are composed of various layers, each playing an important role in data processing and transformation. Before we delve into the details of these layers, it is important to comprehend two key concepts: overfitting and underfitting.

- *Overfitting* happens when a model learns the training data, including noise and random fluctuations, exceedingly well. As a result, the model becomes highly specialized to the training data while failing to generalize successfully on unseen or new data. Overfitting occurs when a model is overly complicated in comparison to the data on which it is trained.
- *Underfitting* occurs when a model is too simplistic to capture the underlying patterns in the data. It does not sufficiently learn the training data, leading in low performance on both the training and test data. Underfitting occurs when a model is too shallow or lacks the ability to describe the intricacies in the data.

With these definitions in mind, we can now explore the various layers of neural networks, shown in Figure 3.1, understanding their roles and how they can impact overfitting and underfitting.

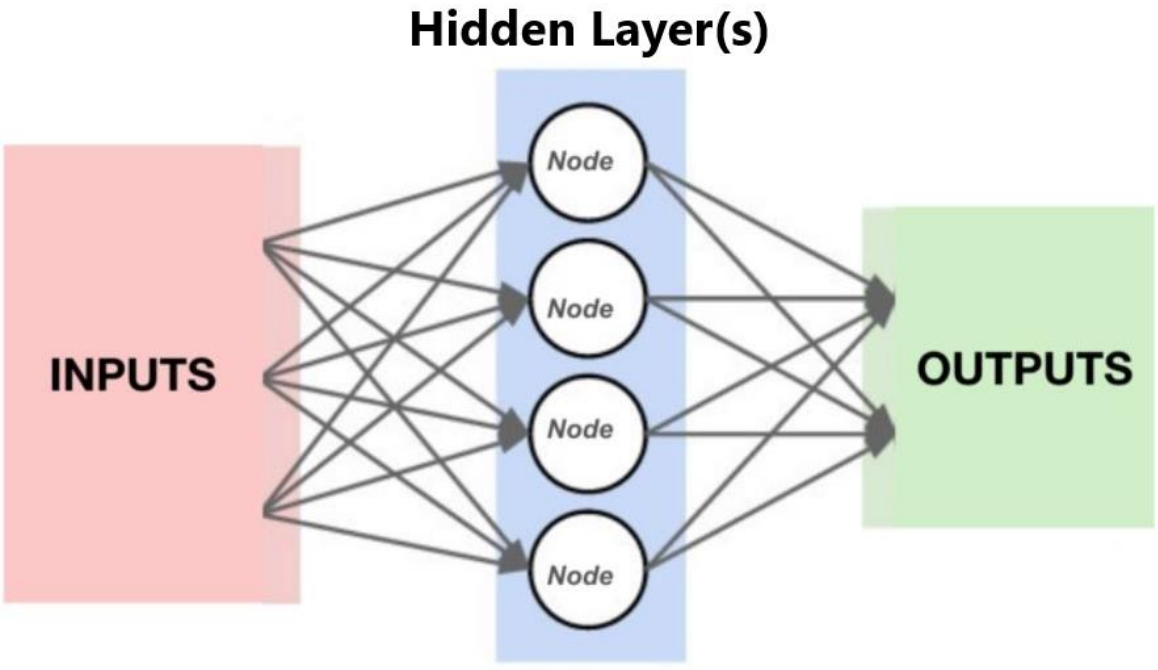


Fig. 3.1. Three main layers in deep learning algorithms.

1. *Input Layer*: The input layer serves as the first point of contact between the neural network and the input data. It is made up of neurons that correlate to the dimensions of the incoming data. This layer just receives input data and passes it on to the first intermediate layer. The dimensionality of the input data determines the size of the input layer.

2. *Intermediate (Hidden) Layer*: Intermediate layers are in charge of performing complex data transformations and extracting essential characteristics and patterns. Each hidden layer receives input from the previous layer and sends its output to the next layer. A node is a computational unit that contains one or more weighted input connections, a transfer function that combines the inputs in a certain manner, and an output connection. The nodes are then grouped into layers to form a network. The number and size of hidden layers can have a big impact on whether the network is vulnerable to overfitting or underfitting. Underfitting can occur when there are too few or too small layers, and overfitting can occur when there are too many or overly large layers.

3. *Output Layer*: The ultimate outcome of the neural network's calculations is produced by the output layer. The output layer size is normally defined by the distinctive problem requirements.

3.3 Neural Network Training

The process of teaching a neural network to produce correct predictions or classifications based on input data is known as training. This process includes several critical steps, including as the neural network design, the selection of a loss function, the employment of an optimization technique (often gradient descent), and the tuning of hyperparameters such as the learning rate. Backpropagation is also an important approach for updating the network weights and biases during training. In the following, we provide brief explanation of the most important steps:

Neural Network Architecture: The architecture of neural networks can vary depending on the scenario at hand, with different types of layers and activation functions utilized. We already discussed common designs, which are made up of layers of interconnected nodes. Weighted interconnections connect each node in a layer to every node in the subsequent layers. The equation of a neuron is the following:

$$z_i = \sum_j w_{ij} \times a_j + b \qquad (3.1)$$

where:

- z_i is the output to neuron i.
- w_{ij} is the weight of the connection between neuron j in the previous layer and neuron i in the current layer.
- a_j is the output (activation) of neuron j in the previous layer.
- b is the bias term.

Loss Function: A loss function (also known as a cost or objective function) calculates the difference between the neural network's predicted output and the actual target values. The nature of the problem influences the choice of a loss function. During training, the aim is to reduce the value of the loss function, which indicates that the network predictions are approaching closer to the real goals. The general formula of a loss function can be expressed as follows:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i; \theta), y_i) \qquad (3.2)$$

where,

- $L(\theta)$ denotes the overall loss or cost as a function of the model parameters (θ). The aim of training a neural network is to minimize this loss by modifying the model parameters.
- N is the number of training examples in the dataset.
- $\ell(f(x_i;\theta), y_i)$ is the specific loss for a single training example. This term measures the error or difference between the predicted output $f(x_i;\theta)$ of the neural network for input x_i and the actual target output y_i. The precise loss function (ℓ) chosen depends on the task.

Activation Function: In a neural network layer, activation functions are applied to the output of each neuron. They add non-linearity to the model, which allows it to learn complicated patterns. Sigmoid, Hyperbolic Tangent (tanh), Rectified Linear Unit (ReLU), Leaky ReLU, and Exponential Linear Unit (ELU) are examples of common activation functions. The activation function adopted can have a considerable influence on network performance and training speed, however, their choice is often problem-dependent.

Forward Pass: During the forward pass, input data is fed into the neural network, and layer-by-layer computations are performed to create an output prediction. Each neuron output is decided by applying an activation function to the weighted sum of its inputs as $a_i = \sigma(z_i)$ in which $\sigma(z_i)$ is the activation function applied to the input z_i of neuron i.

Backpropagation: Backpropagation is a supervised learning method that is used to update the neural network weights and biases in order to minimize loss. It consists of computing the gradient of the loss for each layer, starting from the output one. Weights are then updated by a small multiple of the partial derivative, repeating the process until a specific stopping criterion is met. The small multiplier is called learning rate, playing a major role on the final accuracy of the model. A high learning rate can cause overshooting of the ideal weights, whereas a slow learning rate can cause slow convergence. To increase training stability, techniques such as learning rate schedules and adaptive learning rates (e.g., Adam optimizer) can be applied. The general formula for updating a neural network weight (W) during backpropagation, which incorporates the learning rate (α), is as follows:

$$W_{new} = W_{old} - \alpha . \nabla L(W_{old}) \qquad (3.3)$$

Where:

- W_{new} and W_{old} are the new and old values for a given weight parameter.
- α is the learning rate controling the step size of the weight update.
- $\nabla L(W_{old})$ is the gradient of the loss function regarding the weights at their current values. This gradient is computed through backpropagation.

3.4 Neural Network Architectures

3.4.1 Feed-Forward Networks

The most basic type of neural network, feed-forward neural networks, have a single objective: to approximate a target function indicated by f^*. These networks create a connection between an input x and an output y, denoted as $y - f(x; \theta)$, with the role of the fully connected layers being pivotal in this process. These layers, also known as dense layers, form the backbone of the network, facilitating the transformation of input data through weighted connections and activation functions. Adjusting the parameter θ to achieve the best accurate approximation of the target function is part of the learning process. These models are referred as "feed-forward" because they lack feedback links that allow outputs from various stages to be sent back into the network. They are instead referred to as "networks" since they are often built by merging various functions in a certain way. This structure can be represented as a directed acyclic graph, which shows how various functions are interrelated and combined.

3.4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural network that is designed to handle grid-structured input, as seen in Figure 3.2. Due to its intrinsic capacity to capture both spatial and temporal dependencies within grid-structured data, CNNs have shown to be incredibly successful in a variety of applications, including image classification, voice analysis, and video processing. CNNs have evolved into an essential tool in current machine learning as a result of their capacity to automatically learn and discover meaningful patterns from raw data. As research and development in this field continues, CNNs are anticipated to make even more astounding advances, allowing them to address increasingly difficult and real-world problems.

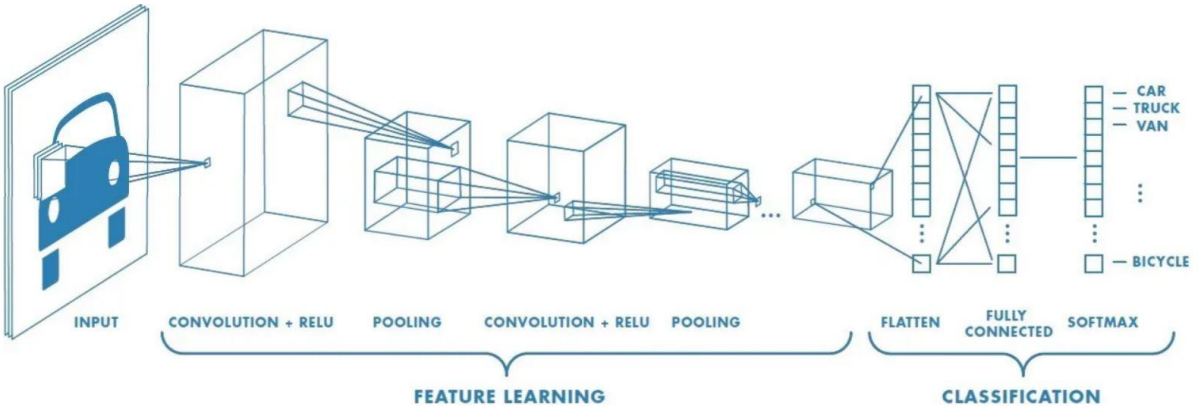


Fig. 3.2. A depiction of a Convolutional Neural Network (CNN) with multiple layers for extracting distinctive characteristics from an input image. (Image from [52])

At the heart of CNNs lies the *convolutional* layer, which employs learnable parametric filters to transform the input into multiple small receptive fields [53]. During the forward pass, a dot product operation is performed between the input data and the filters, resulting in activation maps for each filter that represent distinct features or patterns [53]. This convolution operation facilitates the extraction of diverse features or activation maps through parameter sharing, granting CNNs the desirable quality of translation invariance. In Eq. 3.4, the i and j represent input data positions, whereas m and n indicate filter positions. The formula implements the convolution process, computing each element in the output feature map y as the sum of element-wise products of the filter h and corresponding parts of the input data X. Convolution provides various benefits to machine learning models, including lower memory usage, reduced computational effort, and the ability to handle inputs of various sizes.

$$feature\ map = y[i,j] = input \otimes kernel = \sum\sum X[i-m, j-n].h[m,n] \qquad (3.4)$$

Concat layers, also known as concatenation layers, play an important role in CNNs by concatenating feature maps. This is critical for a number of reasons, including combating the vanishing gradient problem by enabling information to flow from earlier layers to later ones, allowing for the simultaneous learning of low-level and high-level characteristics. Concat layers are especially useful in multi-modal networks for fusing outputs from several input modalities, which improves the network capacity to handle complicated data interactions and produce thorough predictions.

CNNs use pooling layers to reduce the dimensionality of the feature maps, grouping values together and performing a reduction process. Among the several reduction functions, the most common are the maximum and the average [53], average pooling is shown in Figure 3.3. The goal of pooling is to condense the information while limiting the loss of information.

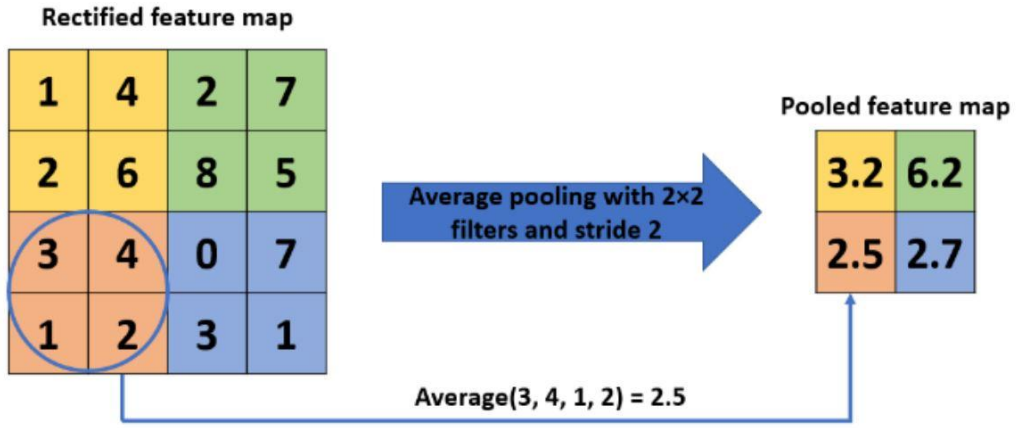


Fig. 3.3. An example of average pooling method. (Image from [54])

The *batch normalization* layer is used to improve the stability of input data distributions by altering their mean and variance. This technique improves the predictiveness of gradients utilized during training, increasing accuracy while decreasing training time. The *dropout* layer, on the other hand, is used during training to prevent overfitting of the network, but it is inactive during inference. It enables the termination of certain neurons to prevent co-adaptation, thus boosting the network's generalization capacity.

At the final stages of the CNN, *fully-connected* (FC) layers come into play. In contrast to the convolution and pooling layers, which act locally, the FC layers implement a global operation. These layers, which are typically positioned at the end of the network, connect every neuron in one layer to every neuron in the next layer [53].

3.4.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of neural network that is designed to process sequential data. They can handle sequences of values indicated as x(1),..., x(τ), and these sequences can be larger and more flexible in length than other non-specialized networks. RNNs, similar to feed-forward networks, assess the input sequence one element at a time. RNNs, on the other hand, have a "memory" vector that stores information about all previous items in the sequence. It is crucial to notice that the index t in the series of items x(t), shown in Figure 3.4, does not always correlate to real-world time. Instead, it represents the place in the sequence. As long as the complete sequence is accessible for the network to execute, this location might travel backward in real-world time or have no direct link to it.

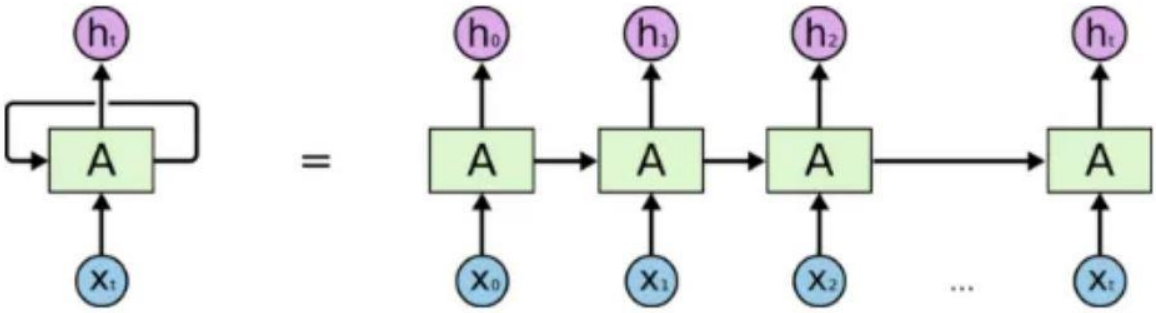
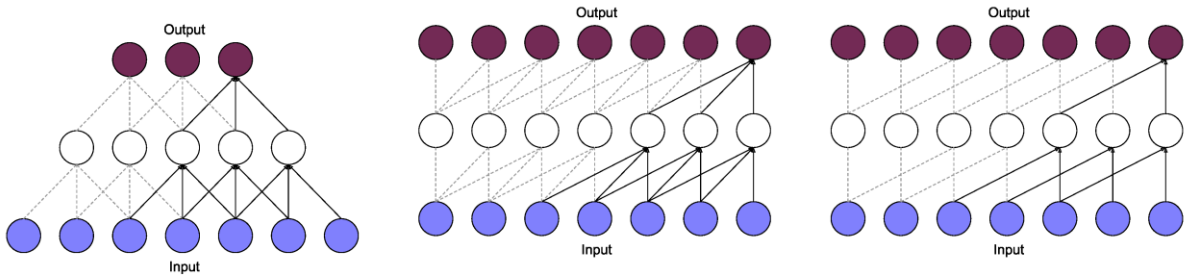


Fig. 3.4. An example of unrolled RNN. (Image from [55])

3.4.4 Temporal Convolutional Networks

Temporal Convolutional Networks (TCNs) have emerged as a prominent variation of 1-dimensional (1D) Convolutional Neural Networks (CNNs), outperforming classical RNNs, LSTM, and GRU models in different time-series processing applications [29], [56], [57]. TCNs have a significant benefit over RNN-based models in that they are less susceptible to training-time difficulties such as disappearing or raising gradients, which might hinder learning. TCNs also require less training memory when dealing with extended input sequences, which is a typical difficulty for RNN-based designs. They also provide computational advantages during inference, making them more efficient in terms of latency and energy usage. This benefit arises from their ability to use traditional CNNs data localization and arithmetic intensity properties [58]. TCNs and regular CNNs have the same essential building pieces, notably Convolutional, Pooling, and

Fully Connected layers. TCN convolutional layers, on the other hand, have two separate properties: causality and dilation, shown in Figure 3.5, which make them suited for processing temporal inputs. This innovative design enables TCNs to attain cutting-edge performance while resolving training-time issues and delivering computational efficiency during inference



(**a**) Standard convolution block with two layers with kernel size 3.

(**b**) Causal convolution block with two layers with kernel size 3.

(**c**) Dilated causal convolution block with two layers with kernel size 2, dilation rate 2.

Fig. 3.5. Comparisons between a conventional convolutional network (a), a causal convolutional network (b), and a dilated causal convolutional network (c). (Image from [59])

To retain the natural sequence of cause and effect in occurrences, TCN convolutional layers impose causality. In practice, this implies that the output at time step t (denoted as y_t) in a TCN convolution is only dependent on a finite number of previous inputs ($x[t-F,t]$). However, dilation is a strategy used in TCNs to extend the receptive field of convolutions along the time axis without introducing extra trainable parameters or increasing the number of operations necessary for inference. This is accomplished by introducing a fixed step (d) between the input samples processed by each convolutional filter. Eq. 3.5 represents the 1D dilated convolution operation in TCN layers, where x and y are input and output activations, T is the output sequence length, W is the array of filter weights, C_{in} and C_{out} are the number of input and output channels, K is the filter size, s is the stride, and F is the layer receptive field, which is determined by the dilation factor (d) and the filter size (K) as $F = d * (K-1) + 1$.

$$y_t{}^m = \sum_{i=0}^{K-1} \sum_{i=0}^{C_{in}-1} x_{ts-di}^{l} . W_i^{l,m}, \forall m \in [0, C_{out}-1], \forall t \in [0, T-1] \qquad (3.5)$$

3.5 Neural Architecture Search (NAS)

The development of efficient and compact convolutional neural network architectures suitable for edge devices has recently accelerated. For instance, early MobileNets [60], ShuffleNets [61], EfficientNet [62], and SqueezeNet [63] are some famous cases. In terms of efficiency, these models have shown considerable potential, making them perfect alternatives for edge computing applications. However, their development required challenging hand modifications of hyperparameters, which took a substantial amount of time and effort. Furthermore, each time a new target job or deployment platform was considered, the tuning procedure had to be restarted from the beginning.

To solve this issue, researchers have developed a number of automated or semi-automated approaches for optimizing neural network topologies. These approaches are known as Neural Architecture Search (NAS) algorithms. The major purpose of NAS algorithms is to relieve designer workloads by automating the process of determining the best architecture for a specific job or deployment target. NAS techniques generally operate as shown in Figure 3.6. The NAS explores the architecture search space, selecting a subset of the candidates. An evaluation strategy will then determine the "goodness" of the found architectures and a control mechanism will decide whether the process can stop or should continue. NAS algorithms can be categorized mainly on three factors:

a) Search space in which the search is undertaken,
b) How the search is performed, including the controller strategy and the evaluation of possible applicants
c) How this candidate performance is assessed.

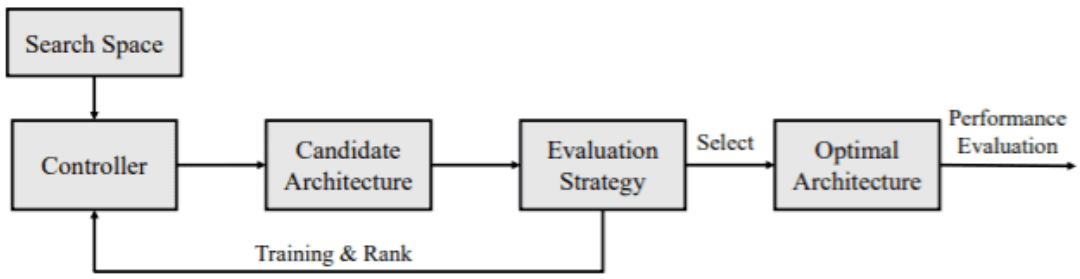


Fig. 3.6. Major components of the landscape of NAS algorithms. (Image from [64])

NAS methods operate by exploring a large design space that includes various combinations of network layers and hyperparameter values. During this investigation, they evaluate the performance of each design using a cost metric. This cost measure often takes into account both the network accuracy and its computational efficiency, which may be defined by factors such as the number of parameters or inference operations.

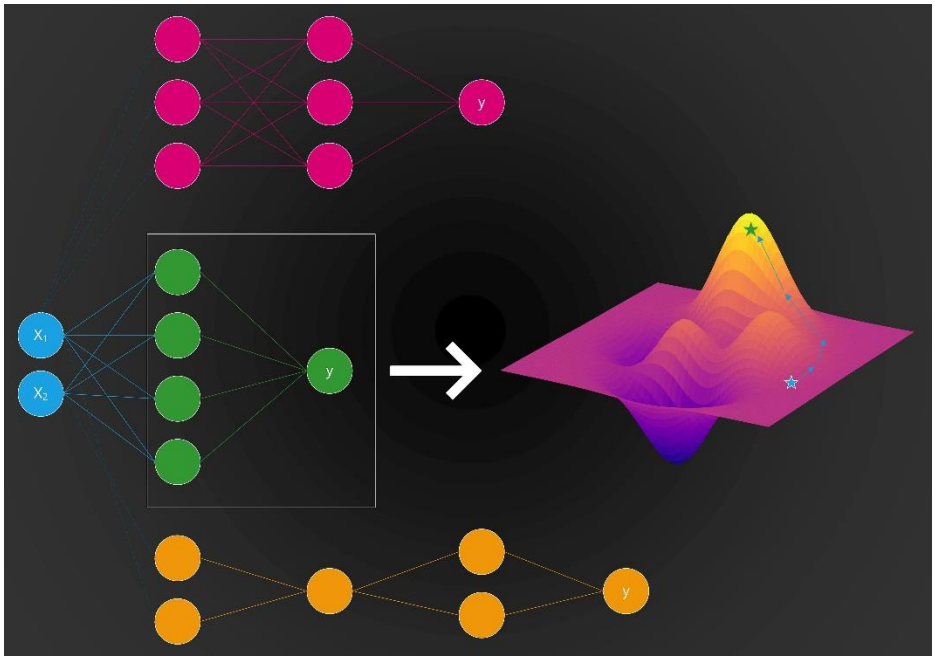


Fig. 3.7. Selecting suitable architecture as optimum network architecture. (Image from [65])

Table 3.1 presents a qualitative comparison of key works in this field, concentrating on search time, memory needs during training (Mem.), search space size, and the possibility to alter the resultant neural network architecture (number and type of layers). Smaller numbers are preferred for Time and Memory, but bigger values are desired for Search Space. Early NAS tools relied on Reinforcement Learning (RL) [8], [11], [66], [67], or Evolutionary Algorithms (EA) [68]. These approaches entail sampling structures from the search space and training them to convergence in order to evaluate their accuracy (and optionally cost), which then drives the next sample iteration. The repeated training in each iteration, on the other hand, is a substantial disadvantage, resulting in considerable GPU hours (thousands) even for relatively basic jobs, resulting in longer search times, as seen in Table 3.1.

Table 3.1
Advanced NAS (Values: ↑ = large, ↗ = medium, ↓ = small)

	Time	Mem.	Search Space	Topology
Reinforcement Learning				
Zoph et al. [8]	↑	↓	↗	Variable*
MNASNET [11]	↑	↓	↑	Variable
NASNET [66]	↑	↓	↗	Variable
MetaQNN [67]	↑	↓	↑	Variable
Evolutionary				
Real et al. [68]	↑	↓	↑	Variable
DifferentiableNAS				
DARTS [69]	↗	↑	↓	Variable
ProxylessNAS [12]	↗	↗	↗	Variable
DmaskingNAS				
FBNetV2 [9]	↓	↓	↑	Fixed
MorphNet [10]	↓	↓	↗	Fixed
S.-Path NAS [70]	↓	↓	↗	Fixed
PIT [5]	↓	↓	↑	Fixed

*Deploy only

To solve the time-consuming search problem of Reinforcement Learning (RL) and Evolutionary Algorithms (EA) techniques, more modern approaches, such as Differentiable Neural Architecture Search (DNAS), Fig. 3.8, have added supernets [69]. Supernets are Deep Neural Networks (DNNs) that include all conceivable alternative layers to be examined during the optimization process. For example, a single supernet layer may comprise many Convolutional layers with differing kernel sizes running in parallel. The challenge of choosing a certain architecture is therefore changed into the task of selecting a path inside the supernet [69]. The choice between multiple paths is represented by binary variables that are jointly learned with the conventional network weights using gradient-based learning. DNAS tools enhance the standard training loss function with an extra differentiable regularization term that captures the network cost to successfully search for correct and efficient designs. The number of parameters and the number of Floating Point Operations (FLOPs) per inference are two common cost metrics [10].

DNAS use a unique optimization approach in mathematics that includes looking for the minimum of a function written as follows:

$$\min_{W,\theta} = \mathcal{L}(W;\theta) + \lambda\mathcal{R}(\theta) \qquad (3.6)$$

The function in this case is made up of two major components. The conventional loss function, indicated by $\mathcal{L}$, evaluates the model performance using regular trainable weights W (such as convolutional filters). Second, there is

a set of extra NAS-specific trainable parameters that are in charge of encoding distinct supernet pathways.

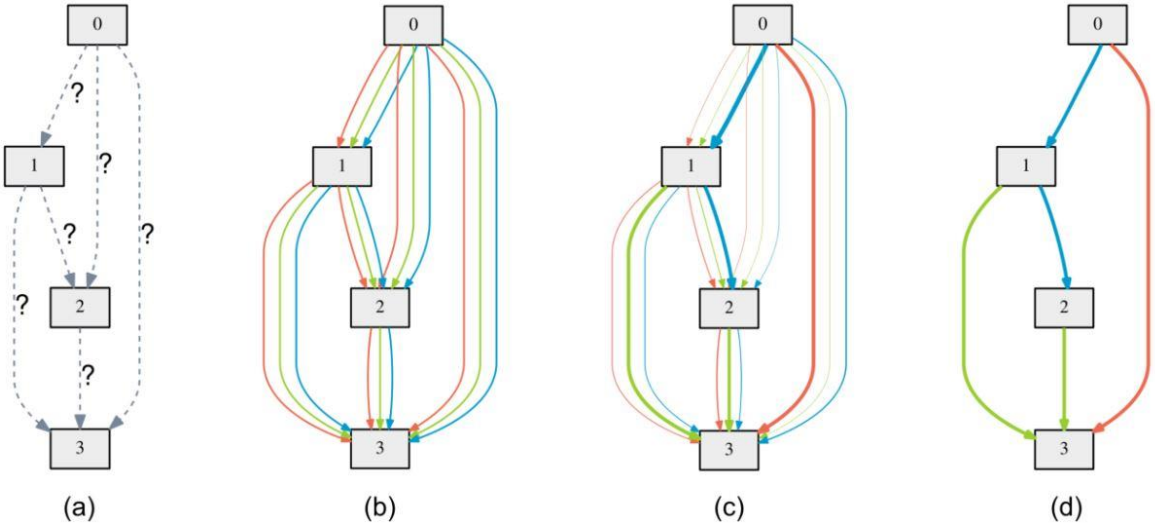


Fig. 3.8. DNAS overview: (a) At first, the operations applied to the neural network edges are not preset or known. (b) By adding a mix of candidate operations on each edge, the search space is continually extended. (c) The technique entails solving a bilevel optimization problem while concurrently optimizing the mixing probabilities (weights) of these candidate operations and the weights of the neural network. (d) The architecture is established using the learnt mixing probabilities, which show the significance of certain processes. (Image from [69])

A regularization loss R, which measures the total cost of the network, is also included to reach an ideal solution. To manage the effects of this regularization term, a hand-tuned parameter is applied, successfully balancing its impact against the principal loss term. The trade-off between model performance and regularization may be fine-tuned to obtain the best configuration.

Despite the improved efficiency of DNAS algorithms over earlier RL/EA-based methods, training the complete supernet still necessitates a significant amount of computing resources in terms of both training time and memory utilization. As a result of this constraint, the investigated search space for practical DNASs like the one indicated in reference [69] is reduced. To keep memory limitations manageable, these algorithms are limited to exploring only a few choices per layer during the search. However, the authors of reference [12] offer a more complex DNAS called ProxylessNAS, which tackles this issue by lowering memory needs. It provides this by maintaining no more than two supernet pathways in memory for each batch of inputs.

ProxylessNAS has a two-phase training strategy. The path parameters are fixed in the first phase, and one sub-architecture of the supernet is randomly sampled depending on their present values. Following that, the weights of the sampling architecture are updated using the training set. In the second phase, the normal weights are frozen, and the architectural parameters are learned using the validation set. This second step updates two pathways at the same time, sampling them from a multinomial distribution. ProxylessNAS may successfully explore a substantially broader search field than other DNAS tools due to this clever method, as seen in Figure 3.9.

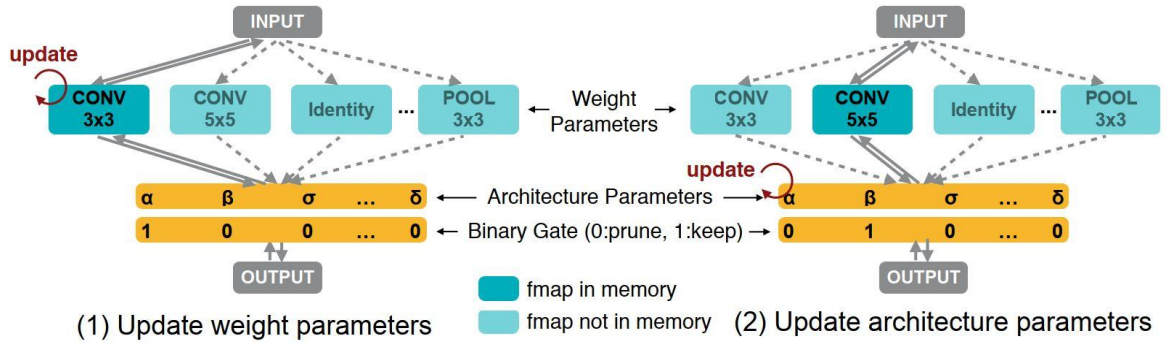


Fig. 3.9. The Proxyless NAS approach learning process for both weight parameters and architectural parameters. (Image from [12])

Advances in techniques such as DMaskingNAS [9], fine-grain NAS [10], and Single-Path NAS [70] have contributed to the progress of lightweight Neural Architecture Search (NAS). These approaches replace the old SuperNet with a single large, architecture with one route. Finding optimal architectures entails altering this initial seed model through hyper-parameter tweaking, namely the number of channels in each layer [10]. Trainable masks are used to prune sections of the network to perform this tweaking effortlessly during regular training. The goal of DMaskingNAS tools is the same as that of DNAS (as stated in [71]), with now representing the set of trainable masks.

FBNet-V2 [9], for example, employs a collection of specialized masks, each storing a distinct number of output channels or spatial resolutions and weighted with trainable parameters. The mask associated with the biggest parameter is picked after the search to establish the final architectural setting. MorphNet [10]

similarly use the pre-existing multiplicative terms of batch normalization layers as masking parameters [72]. If these parameters drop below a particular level, the preceding related channels of Convolutional layer or feature maps are removed. When compared to SuperNet-based techniques, these methods are more limited in terms of neural network structure. They do not, for example, allow you to select between different layers, such as a regular convolution and a depthwise plus point-wise convolution.

These restricted techniques, however, offer two key advantages. First, they are substantially more efficient in terms of memory utilization and search time while still identifying high-quality designs. Notably, the search time of a DMaskingNAS is comparable to the training time of a normal network. Second, some DMaskingNAS variations have the ability to explore the search space with considerably finer granularity. MorphNet [10], for instance, may simply choose between 1 and 32 output channels in a Convolutional layer with a precision of 1 by starting with a 32-channel seed layer and deleting channels with the smallest batch normalization multiplicative parameters. To achieve the same level of fine-grained selection with a standard DNAS, an extremely large supernet consisting of 32 parallel convolutional layers would be required. Additionally, the masking and supernet techniques may be used to overcome the constraints of DMaskingNAS [73].

The mentioned literature on NAS focuses mostly on 2D-CNNs for computer vision problems. Surprisingly, few of these techniques have been adapted for time-series data processing, despite the fact that many real-world activities include one-dimensional time-dependent signals, such as bio-signals, audio, energy-traces, and sensor readings from industrial equipment. In [5], they introduce a unique technique called PIT, which is particularly developed to optimize 1D networks. The method ideas are adaptable and might serve as the foundation for a more thorough NAS that investigates temporal hyperparameters of N-dimensional Convolutional layers, including 3D-CNNs, for spatial-temporal data processing.

3.6 Metrics

3.6.1 Accuracy

Accuracy Score is a measure used to assess a classification model performance. It calculates the fraction of correctly predicted occurrences in the dataset as a percentage of the total number of instances. In other words, it measures the model's ability to predict outcomes correctly across all classes. The accuracy score is determined mathematically using the formula:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions} \quad (3.7)$$

Where:

- "Number of Correct Predictions" is the number of times the model predictions match the actual class labels.
- "Total Number of Predictions" is the total number of predictions produced by the model.

It is a popular statistic for evaluating the overall performance of classification models, particularly when the classes in the dataset are balanced (equally frequent in the dataset). In the presence of imbalanced datasets, when one class outnumbers the others greatly, accuracy may not offer a whole picture of a model performance, and alternative measures like as precision, recall, F1 score, or balanced accuracy may be more helpful.

3.6.2 Balanced Accuracy

Balanced Accuracy (Bal. Acc.) is a performance indicator used in classification model assessment, particularly when dealing with datasets that are imbalanced. It is defined as the average of the per-class recall, where recall (also known as sensitivity or true positive rate) for each class is a measure of the model's ability to correctly identify instances of that class among all occurrences of that class. In such scenarios, since a model that merely predicts the majority class for all occurrences might obtain high accuracy while performing poorly on minority classes, Balanced Accuracy provides a fair assessment of a model performance across all classes.

The formula for Balanced Accuracy can be expressed as:

$$Bal.Acc = \frac{1}{N}\sum_{i=1}^{N}\frac{TP_i}{TP_i+FN_i} \quad (3.8)$$

Where:

- N is the total number of classes.
- TP_i represents the number of true positives (correctly predicted instances) for class i.
- FN_i represents the number of false negatives (instances of class i incorrectly predicted as a different class or not predicted as class i at all) for class i.

3.6.3 F1 Score

The harmonic mean of precision and recall is used to calculate the F1 Score. The following describes the way precision and recall are calculated:

- Precision is the ratio of true positive predictions to the total number of positive predictions generated by the model (also known as Positive Predictive Value). It determines how many of the model's positive predictions are accurate.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad (3.9)$$

- Recall (also known as Sensitivity or True Positive Rate) is the ratio of true positive predictions to total positive outcomes in the dataset. It assesses the model's ability to recognize all positive events properly.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ \text{Negatives}} \quad (3.10)$$

The F1 Score is then derived through calculating the harmonic mean of precision and recall by following formula which spans a range of 0 to 1, with higher values signifying greater model performance:

$$F1\ Score = 2 \times \frac{Precision * Recall}{Precision + Recall} \quad (3.11)$$

3.6.4 Mean Square Error

Mean Squared Error (MSE) indicates the average of the squared differences between predicted outcomes (commonly written as y) and actual observed values (denoted as $\hat{y}$) for an n-point dataset. It is determined as follows:

$$MSE = \frac{1}{n}\sum(\hat{y_i} - y_i)^2 \qquad (3.12)$$

Where:

- MSE: Mean Squared Error
- n: The total number of data points in the dataset
- $\hat{y_i}$: The predicted value for the i-th data point
- y_i: The actual observed value for the i-th data point
- $\sum$: The summation symbol, indicating that you calculate the squared difference for each data point and then sum them all up.

Squaring the differences serves several goals, including making all errors positive (negative errors are not cancelled out by squaring) and penalizing larger errors more strongly than smaller ones.

3.6.5 Mean Absolute Error

The absolute difference between each estimated value (denoted as $\hat{y}$) and its corresponding actual or observed value (denoted as y) in a dataset is used to compute the Mean Absolute Error. The MAE is calculated by averaging these absolute differences across all data points. The MAE formula is stated mathematically as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |\hat{y_i} - y_i| \quad (3.13)$$

Where:

- MAE represents the Mean Absolute Error.
- n is the total number of data points or observations in the dataset.
- $\hat{y_i}$ is the predicted value for the i-th data point.
- y_i is the actual or observed value for the i-th data point.
- $|\hat{y_i} - y_i|$ calculates the absolute difference between the predicted and actual values for each data point.

- The summation symbol ∑ represents the sum of these absolute differences across all data points.

MAE measures the average disparity between model predictions and actual values, with lower MAE indicating more predictive accuracy and equal weighting of errors regardless of direction.

3.7 Infrared Sensors and Machine Learning

In the realm of Infrared (IR) sensor arrays, a "pixel" is the essential building unit of an Infrared (IR) sensor array, similar to a pixel in a normal digital picture but specialized for recording infrared radiation. These pixels are arranged in a grid or array to create an IR sensor array. Their principal duty is to detect and quantify infrared radiation produced by objects in their field of view. As each pixel detects the intensity of heat radiation at a given position, they work collaboratively to create a picture or data set that varies in resolution depending on the size of the array. Heatmaps are frequently generated using this data to display the distribution of temperatures throughout the scene. Colors or shades are used to depict thermal fluctuations in heatmaps, with warmer colors representing greater temperatures and colder hues suggesting lower temperatures. These heatmaps are useful tools for thermal imaging, industrial monitoring, building diagnostics, and other applications that need an intuitive knowledge of temperature patterns.

Let us represent the most recent Infrared (IR) frame as x_t at any given time, instance t, which effectively acts as an "image". This single frame $X_t = x_t$ is sent into the identification model, as is a window of subsequent frames $X = \{x_{t-W+1}, \ldots, x_t\}$, where W denotes the window size (W = 3 in Fig. 3.10). The ultimate aim is to estimate the number of persons $y_t = \hat{f}(X_t)$. This projected number can be produced as a continuous scalar (regression formulation) or as a categorical value matching to one of many potential counts (classification formulation).

The procedure of calculating the input/output connection $\hat{f}(X)$ can be accomplished in two ways. The first method includes using deterministic rule-based algorithms to create a link between the input data and the related people count. Alternatively, using Machine Learning (ML) or Deep Learning (DL) methodologies, the connection can be discovered from a properly chosen training dataset. These ML/DL approaches allow the model to recognize complex patterns and variances in visual input, resulting in more accurate people counting results.

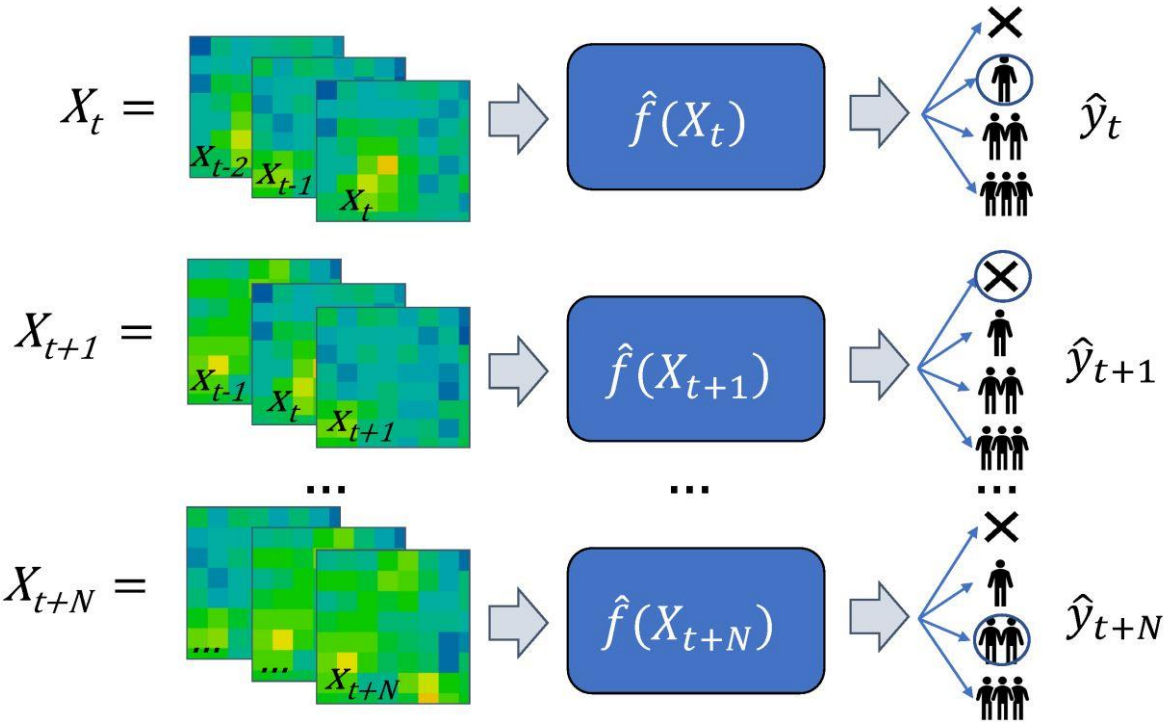


Fig. 3.10. Problem formulation for counting people using IR array sensors. Depending on the task, the prediction function $\hat{f}(X)$ can be produced using a rule-based deterministic approach or learnt from data using ML/DL, and the predicted person count y_t can be either a scalar or a class label. (Image from [43])

Chapter 4
Materials and Methods

4.1 Motivation

The goal of this study is to explore the applications of NAS to a novel field, that is, ultra-low-resolution infrared sensor arrays. The advantage of NAS techniques is the ability to explore a wide set of architectures in negligible time, as opposed to hand-crafted models, requiring considerable efforts and expertise from the developers. Starting from a single seed architecture, we are able to spawn several models, representing different trade-offs between memory and accuracy.

In this work, we use 8x8 IR sensors as they have several advantages w.r.t higher resolution ones. These benefits include improved privacy preservation, cheaper system costs, and decreased power consumption, which is especially important for systems that rely on battery power and must function over extended periods of time. The fundamental motivation for this work stems from the lack of a comprehensive comparison of deep learning models specialized to this type of data in the literature. As a result, our activity serves two interrelated functions. To begin, it provides useful information to system designers who desire to use such sensors, supporting them in selecting a suitable family of deep learning models depending on their desired accuracy and hardware memory limits. Second, it provides as a practical instance of how deep learning may attain higher accuracy while simultaneously being more efficient than traditional approaches.

4.2 Dataset

In this work, we utilize a new dataset called LINAIGE (Low-resolution INfrared-array data for AI on the edGE). The major focus of LINAIGE dataset is on tasks related to counting individuals and recognizing their presence in indoor situations. The first version of this dataset was introduced in a prior study [16]. This collection consists of infrared (IR) samples collected with a Panasonic Grid-EYE (AMG8833) sensor [42]. At a rate of 10 frames per second (FPS), the sensor generates an 8 by 8 array of data. Each of these frames is tagged with the number of individuals present.

The sensor was put on the ceiling during the data gathering procedure and it was used in a variety of interior settings, including offices, labs, and hallways. A lens with a viewing angle of 60° was employed on the sensor. During repeated data collecting sessions, volunteers moved inside the sensor range of vision, doing actions such as walking, standing, and running. Figure 4.1 shows examples of gathered frames and their corresponding persons counts. In diverse contexts, the maximum distance between humans within the sensor vision varied (between 1.53 and 2.04 meters), as detailed in [16]. The space for counting persons was roughly 2 square meters in size. People counting is possible in bigger regions by combining the results of many sensors that are strategically placed.

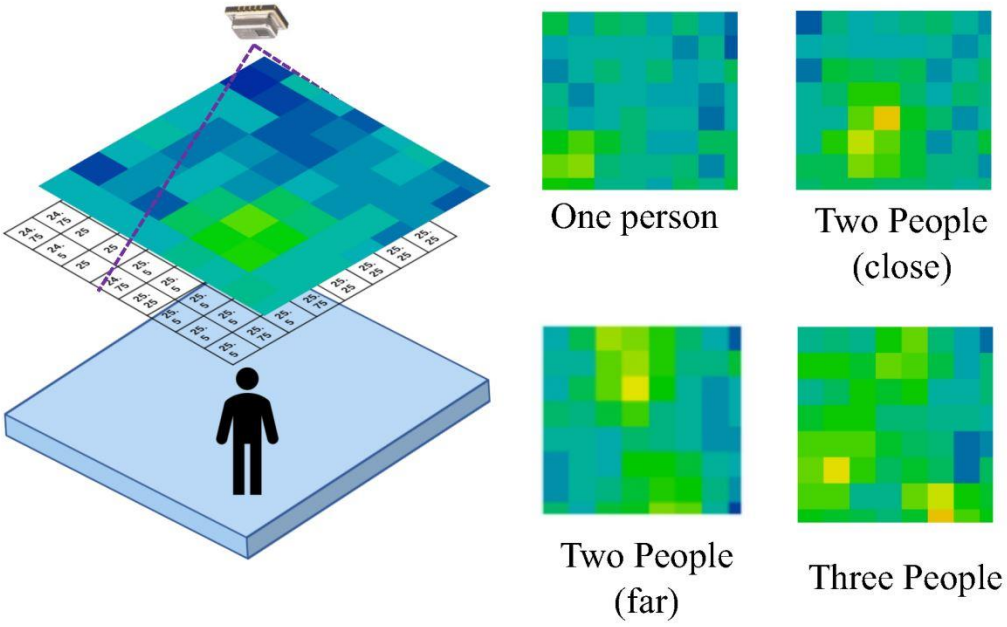


Fig. 4.1. Sensor attachment with an example of an infrared frame. (Image from [43])

To label IR frames, a semi-automatic technique was applied. This entailed configuring a data gathering system with a Raspberry Pi 3B single-board computer that featured both an infrared sensor and an optical camera facing in the same direction. The synchronized frames obtained from this configuration were processed. To begin, the optical frames were analyzed using a pre-trained object identification model (particularly, Mask R-CNN [74]) to automatically count the number of persons and this count was then applied to the associated IR frame. Human verification was employed to fix any errors produced by the object detection model in order to assure accuracy. Each frame was additionally assigned a confidence level (binary) by the human annotator. This level identifies frames where determining the precise persons count was difficult due to a small mismatch between the IR sensor and the optical camera angles.

In this work, we follow the pre-processing described in [43]. Notably, frames with more than three persons are removed, as they were exceedingly infrequent (constituting 0.66% of total data) and were only seen in one data collecting session. These frames hamper machine learning and deep learning model training and cross-validation. In addition, the smallest session (session 4 in [16]) with only 196 frames (approximately 20 seconds of data) was eliminated. This measure was done to ensure that performance metrics for recognition were reasonable. Following these changes, the updated dataset now contains 25,110 samples spread over 5 sessions. To each session is assigned a timestamp, the name of the environment, and the temperature of the room.

As done in [43], we purposefully eliminated difficult-to-label frames from both the training and testing stages. We apply the same per-session Cross-Validation (CV) technique detailed in [43], and Table 4.1 outlines its specifics. Notably, due to its large size compared to the other sessions (17958 frames vs a maximum of 2202 frames for other sessions, accounting for 71% of total data), we kept Session 1 in the training set across all rounds. Sessions 2, 3, 4, and 5 were utilized as the test set in various folds, while the remaining data was used for training. This approach resulted in four unique CV folds. The main benefit of this "leave-one-session-out" CV method is that it provides impartiality when evaluating model performance. This is accomplished by ensuring that the test frames are generated from environments, date-time combinations, and room temperature settings that differ from those seen during training. This simulation, in essence, closely resembles a real-world scenario in which the system is evaluated under conditions different from its training environment. A strictly random allocation of frames to

training and testing, on the other hand, would risk compromising the integrity of the evaluation process by enabling information to leak between the two phases, resulting in an oversimplified problem representation.

Table 4.1
Dataset Statistics and Cross Validation Strategy

Train Fold						**Test Fold**					
Session	**Sample N.**	**People Counts Statistics [%]**				**Session**	**Sample N.**	**People Counts Statistics [%]**			
		0	**1**	**2**	**3**			**0**	**1**	**2**	**3**
1,3,4,5	23529	26.07	43.49	23.61	6.83	2	1581	14.86	30.68	54.46	0
1,2,4,5	23591	22.37	44.03	26.84	6.77	3	1519	71.89	21.72	5.66	0.72
1,2,3,5	22908	25.3	41.85	26.17	6.67	4	2202	26.02	51.27	19.16	3.54
1,2,3,4	23260	24.69	43.02	26.08	6.20	5	1850	33.78	38.38	18.92	8.92

4.3 Pruning in Time

Pruning in Time (PIT), is intended for networks that handle time-series data, focusing mainly on convolutional and fully-connected layers, as they are the most memory and computationally intensive. PIT explores a wide range of sub-architectures produced from a seed model by modifying three critical hyperparameters. PIT can choose to reduce the number of output channels (C_{out}) or the filter size (F) while raising the dilation factor (d) in comparison to the default setup. These adjustments lead to reduced layer complexity and memory usage. Each convolutional/fully connected (FC) layer in the fundamental CNN is changed into a function $L_n(W^{(n)}, \theta^{(n)})$ that depends on its original weight tensor $W^{(n)}$ and a new set of architectural parameters $\theta^{(n)}$ to achieve this goal. The search space of PIT for a CNN with N layers is therefore described as the set:

$$S = \{L_n(W^{(n)}, \theta^{(n)})\}_{n=0}^{N-1} \quad (4.1)$$

The elements of $\theta^{(n)}$are merged in an appropriate manner throughout the search phase to generate a binary mask $\Theta^{(n)}$. This mask is then used to remove portions of the layer weights. In each iteration of the search, an architecture $\hat{S}$ is randomly picked from S using the Hadamard product of $W^{(n)}$ and $\Theta^{(n)}$, denoted as $\hat{S} = \{L_n(W^{(n)} \odot \theta^{(n)})\}_{n=0}^{N-1}$. The sections of $W^{(n)}$ that correspond to 0-valued mask elements are removed by this procedure. As a result, the seed layer can provide the same output as if it had fewer channels, a smaller receptive field, or even a bigger dilation. When working with slices of $W^{(n)}$, binary masks are required for the Hadamard product procedure. During architectural sampling, these masks decide whether a slice is fully eliminated (set to 0) or maintained as is (set to 1). In practice, this assures that only acceptable designs with whole-number C_{out}, F, and d values are examined. To do this, during the forward pass of search or training, $\Theta^{(n)}$ is transformed into binary form. This is accomplished through the use of a Heaviside step function with a constant threshold of 0.5.

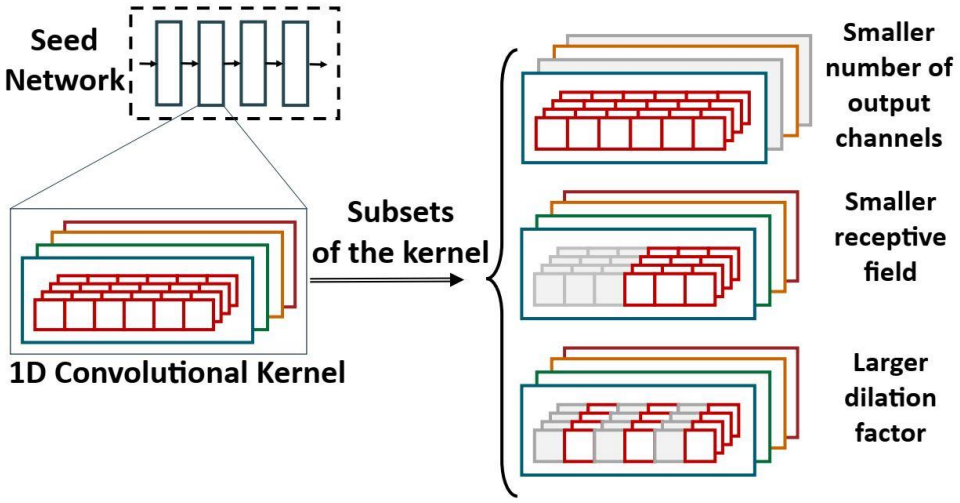


Fig. 4.2. Illustration of PIT search space. (Image from [5])

A differentiable technique is required to seamlessly incorporate the $\theta^{(n)} \rightarrow \Theta^{(n)}$ transformation into the gradient-based training of the network, allowing simultaneous learning of both the weights $W^{(n)}$ and architectural parameters $\theta^{(n)}$. However, when dealing with the Heaviside function, which has difficulties with its derivative being practically everywhere equal to 0 and non-existent in δ, the Straight-Through Estimator (STE) approach provided in BinaryConnect [75] is used. During the backward pass, the step function is replaced by a simple identity function.

The authors divide the $\theta^{(n)}$ parameters in their model into three different sets: $\alpha^{(n)}$ for altering the number of channels, $\beta^{(n)}$ for fine-tuning the receptive field, and $\gamma^{(n)}$ for affecting the dilation factor.

4.3.1 Channel search

The authors of [5] take inspiration from prior research referred to as [10] in order to examine the best number of channels in each convolutional layer. Binary masks were constructed using batch normalization (BN) layer settings in that study [72], allowing for the pruning of whole output channels. This method allows for the examination of numerous sub-layers with a $C_{out} < C_{out,seed}$ condition. The necessity for a BN layer after each convolutional layer limits the application of the approach suggested in [10], while ubiquitous in recent 2DCNNs.

To alleviate this shortcoming, the PIT technique detaches channel search from batch normalization. Instead, they employ a specific set of trainable parameters labeled as α. These parameters are used to zero-out complete convolutional layer filters from the W tensor. They can effectively explore the space of channel configurations without being bound by the presence of BN layers after each convolution since they handle each output channel individually.

In practical implementation, the procedure is taking a binary mask and multiplying each element of the mask by all of the weights of a certain convolutional filter. This affects the whole slice of the W tensor along the output channels axis. When a filter is multiplied by a 0-mask element, the associated output channel from that layer is effectively removed. Figure 4.3 shows how the Θ_A parameters are applied to a simple layer with $C_{out,seed} = 4$.

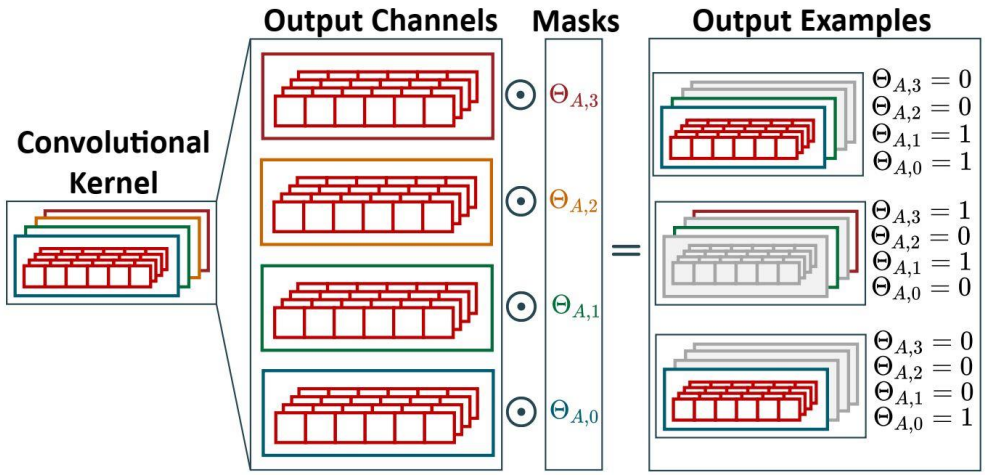


Fig. 4.3. $\Theta_{A,m} = 0$ causes the m-th convolutional filter to be deactivated, thereby setting a section with dimensions K × Cin inside the weights tensor W to zeros. (Image from [5])

Additionally, thanks to the binary masks used by PIT, any combination of channel can be removed, while previous approaches were limited to the last ones.

4.3.2 Receptive Field Search

The receptive field F, which describes the range of input time-steps involved in a convolution, is the second crucial hyperparameter. F is equivalent to the filter size in ordinary convolutions (F = K). For CNNs with dilation factors (d) higher than one, the connection changes, and the general formula turns into: $F = (K - 1) * d + 1$. PIT additionally considers the dilation factor and investigate both F and d, which indirectly enhances the filter dimension K.

PIT employs an array of extra trainable parameters β with a length of F_{seed} to explore the receptive field. In contrast to the output channels, the parameters β must be united further to define the matching binary differentiable masks. This extra step is required because just masking any collection of time-slices in the weights tensor will not effectively imitate the impact of a narrower receptive field in a causal CNN convolution. The receptive field should exclusively extend into the past, therefore, they eliminate the "oldest" time-slices, those multiplied with input time-steps that are the uttermost in the past, to achieve the desired effect of a reduced receptive field.

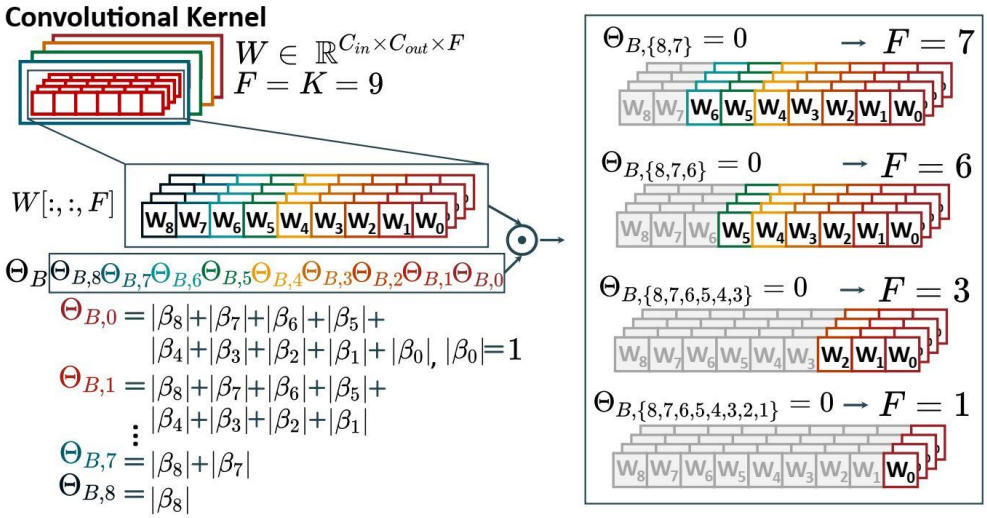


Fig. 4.4. A receptive field search is demonstrated by setting each value $\Theta_{B,i} = 0$ resulting in the elimination of one input time-step from the convolution output. This is accomplished by canceling out a particular portion of the weight tensor W with dimensions Cout × Cin, eliminating its influence on the output. (Image from [5])

4.3.3 Dilation Search

The PIT algorithm additionally examines the dilation factor in the same way as it studies the receptive field. Certain constraints are set on the weight tensor portions that NAS should trim by adding the search for dilatation. They must specifically ensure the creation of only normal dilation factors, which means that the time-step intervals between consecutive convolution inputs are all uniform for a given layer.

They begin with an array of adjustable parameters denoted as γ, which are used to generate differentiable binary masks. The technique is restricted to supporting dilation factors with powers of two. These unique elements are not only extensively used in numerous applications, but they also help in the simple

creation of masks. As a consequence, $\text{len}(\gamma) = [log_2(\text{F}_{\text{seed}})]$ determines the number of parameters in the array. Figure 4.5 depicts an example of how the tensor is created and its influence on dilation.

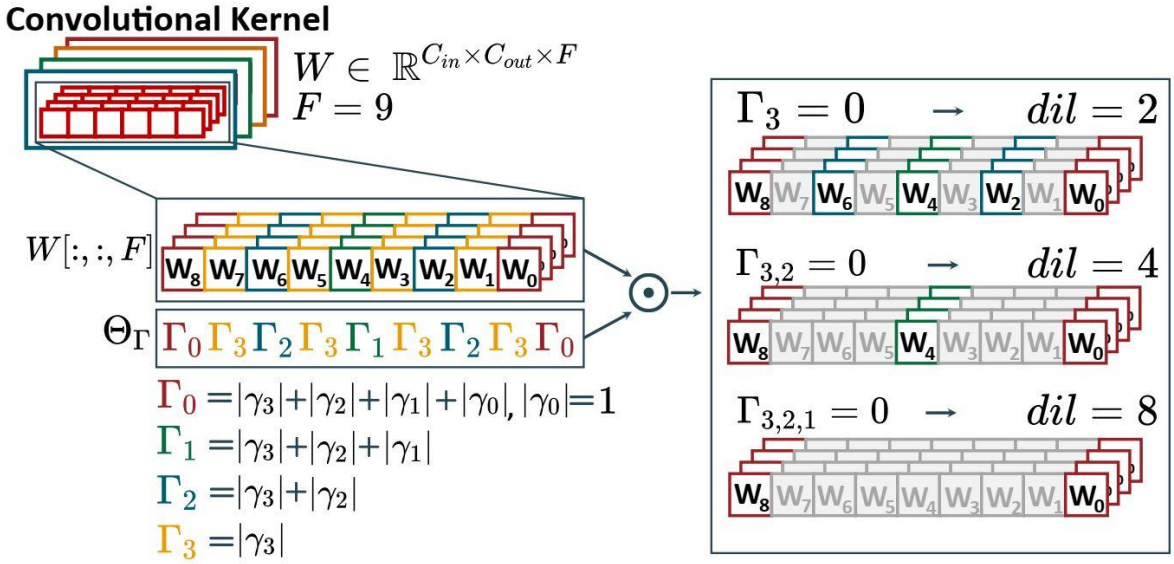


Fig. 4.5. Dilation search works by doubling the value of parameter 'd' for each case when Γ_i is set to 0. (Image from [5])

4.3.4 Joint Search

They apply their respective masks on the weight tensor of a layer to maximize the three hyperparameters described above. When compared to sequential hyperparameter optimization, this collaborative technique yields superior results. As a result, the PIT is able to examine complicated interconnections, notably between F and d, resulting in better performance.

4.3.5 Regularization

The PIT method seeks designs that are both accurate and simple, which it is required by combining the task-specific loss function (L) with a regularization term (R), Eq. 4.2.

$$\min_{W,\theta} = \mathcal{L}(W;\theta) + \lambda\mathcal{R}(\theta) \qquad (4.2)$$

This regularization term offers a distinct component that leverages past knowledge of the loss landscape, encouraging the optimization process to prefer solutions with lower computational costs.

The number of parameters (or model size) and the number of operations (OPs) required for inference are two cost metrics studied. R_{size} and R_{ops}, two separate regularizers, are employed to manage them. These regularizers are differentiable functions whose values are determined by the pre-binarization masks without Heaviside binarization. These masks are associated with the trainable architectural parameters $\alpha, \beta \; and \; \gamma$, and the use of pre-binarization masks, as in prior research [10], produces a smoother loss landscape, which enhances optimization convergence.

4.4 SuperNet

An approach to further expand the search space of the NAS is to chain different methods in cascade. In particular, in this work, we use first a SuperNet and then apply PIT on the output of this method. In this way, with SuperNet, we can choose between multiple convolutional types, ranging from basic convolutions to Depthwise-Separable convolutions, or even exclude some layers entirely. Following that, with PIT, we can explore a wide range of sub-architectures produced from previous stage.

A vital step in clarifying the process of building and deploying a SuperNet utilizing the pit_supernet technique is the development of a pivotal object known as the PITSuperNetModule. This module is distinguished by a collection of layers that strictly adhere to the PIT-SuperNet criteria. A SuperNet, for example, might have a variety of possibilities, such as 2D convolutions with various kernel sizes, Depthwise-Separable convolutions, and even the exciting potential of altogether skipping a layer by using nn.Identity. The selections for each convolutional layer are the main focus of this project.

It is critical to follow a methodical sequence of actions before embarking on the adventure of creating a network suitable to search, conducting the search itself, and eventually exporting the detected neural architecture:

1. The first stage entails utilizing the capabilities of the PITSuperNet conversion module, which effortlessly changes the model into an optimized format. This conversion procedure necessitates three main components: the model itself, the form of the input tensor (excluding batch size), and the selection of a chosen regularizer, which serves as a guiding principle for the optimization measure.
2. An important computation occurs as the training loop progresses, in which the regularization loss is computed and harmoniously coupled with the task loss. This collaboration strives to maximize both performance dimensions sustainably. A scaling factor is wisely used to the regularization loss to finely manage the equilibrium between these two losses.
3. The precisely optimized model is then exported as the last stage. Following the conversion, a prudent method would require submitting the exported model to an additional step of fine-tuning. This additional layer of refinement improves the model overall performance and resilience.

4.5 Model Architectures

We take the architectures from [43] as they represent the state-of-the-art models on the LINAIGE dataset. Afterward, we use PIT on each architecture, systematically assessing various levels of regularization strengths to identify distinct trade-off points. In the subsequent sections, we provide a comprehensive breakdown of each architectural variant.

4.5.1 Single-frame CNN

Single-frame CNNs, as suggested by their name, use a single frame as input, thus having no knowledge of the past number of people in the view of the IR sensor. This CNN design is crucial to our analysis, and Figure 4.6 depicts its basic structure. This architecture comprises Convolutional layers with ReLU activation, optional Max Pooling, and Fully Connected layers.

An extensive and methodical examination of various architectural modifications was carried out using this core design template as a starting point. This investigation entails the selective retention or destruction of certain levels within the design, shown in Figure 4.6 by the dashed boxes. Some of the architectural arrangements that have been considered are as follows:

1. One or two initial convolutional layers, each followed by Batch Normalization.
2. An optional additional Fully Connected layer before the output one.
3. Including or excluding a single Max Pooling layer.

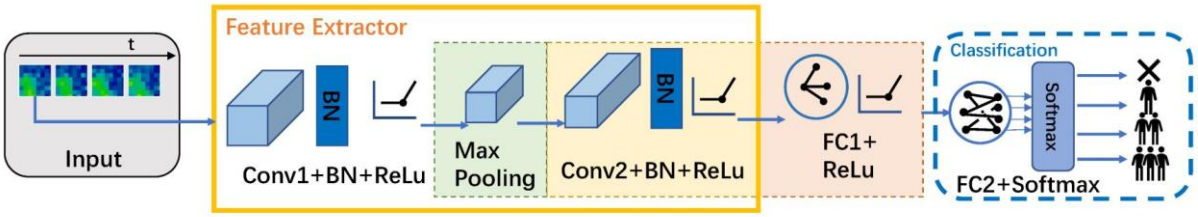


Fig. 4.6. Single-frame CNN. (Image from [43])

In order to broaden our architectural explorations, we performed experiments with changing the amount of feature maps (also known as channels) within each Convolutional layer. This entails a methodical investigation of various channel

counts from the set 8, 16, 32, 64. It is also worth noting that the Convolutional and Max Pooling kernel dimensions stay constant at 3x3 and 2x2, respectively. A single frame generated from an infrared (IR) array is processed as input data for the CNN model.

4.5.2 Multi-frame CNN

The previous model technique relied on a single infrared (IR) frame as input to address the problem of people counting. Multi-frame technique, on the other hand, tries to improve the precision of this job by using the temporal refinements contained in a series of sequential frames. The key concept involves using a sliding window setup of infrared frames as input. This approach has the potential to uncover important information about how individual movement patterns work. This intentional integration of temporal information becomes highly effective in improving prediction accuracy, especially in complex and diverse settings.

Having a look at Figure 4.7 to get a better understanding of this topic, the final picture draws attention to a unique location generating higher thermal signals, represented by a noticeable purple box. The multi-frame approach advantage resides in its ability to accurately recognize not just one person in this hot zone, but rather the presence of two closely positioned persons. This differentiation is made possible by studying the trajectory of motion displayed by these two people, as indicated by the emphasized trajectories in prior frames, captured by the red and orange cycles.

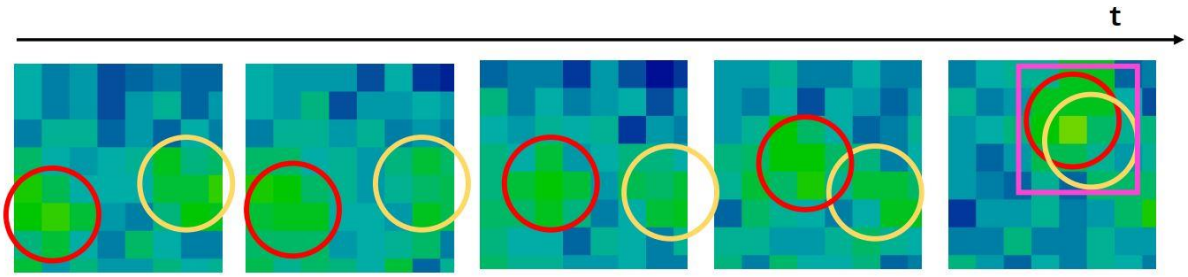


Fig. 4.7. An example of an IR frame sequence related to two persons moving in close proximity to each other. (Image from [43])

In basic terms, each frame inside a W-length sliding window is handled as an independent input channel. To elaborate, suppose the sliding window has a length given by W. Under this premise, a stacking method is used to assemble IR frames

along the channel dimension $X_t = \{x_{t-W+1},...,x_t\}$. This fusion leads in the formation of a tensor of dimensions (W, 8, 8). This tensor is then inextricably coupled with the people count label corresponding to the last frame, designated as y_t, and used for both training and testing.

Considering the architecture seen in Figure 4.8, fittingly named the Multi-Frame CNN model. During the design of this model, an equivalent was drawn with hyper-parameter configurations similar to those used in Single-frame CNNs. This translates into characteristics like the number of layers and the number of convolutional channels.

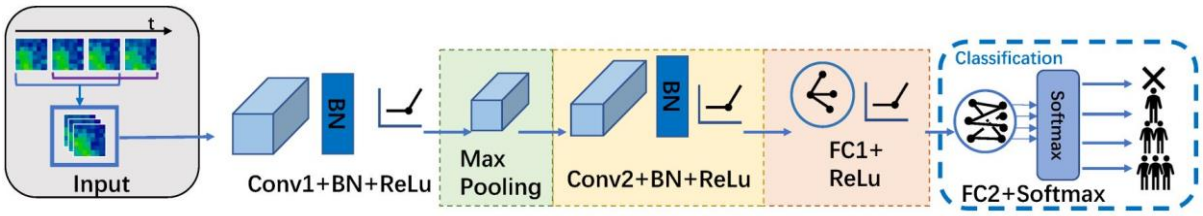


Fig. 4.8. Multi-frame CNN. (Image from [43])

Furthermore, an in-depth investigation focuses on the dimensions of the sliding window, designated as W. The experiments cover a range of numbers including 3, 5, 7, and 9. The inherent value here is in recognizing that an excessively shortened window may insufficiently leverage the reservoir of previous frame information. In contrast, an overly large window may mistakenly include insignificant previous data into the study. Finding an ideal balance in terms of window size emerges as a critical concern, since this element has a significant influence on both the computational demands and the memory complexities of the first Convolutional layer.

4.5.3 Majority Voting CNN

Majority voting, a technique used in ensemble learning, stands out as a straightforward and fast strategy that uses several categorization outputs to provide decisive predictions, lowering variability. There exist multiple possibilities for applying this method, including the use of different classifiers, the deployment of several versions of the same model with different training

approaches, or even the use of a single trained model supplied with various inputs. The latter technique is used in our present investigation, [76].

Our method includes applying majority voting, commonly known as mode inference, to predictions generated by applying a Single-frame Convolutional Neural Network (CNN) to individual frames in a sliding window. Figure 4.9 depicts this procedure schematically. This approach has a significant benefit, especially in the context of edge inference. It has a memory requirement that is identical to that of a single-frame CNN, and voting process of W predictions might lead to better prediction accuracy by efficiently filtering out rare mispredictions. It should be noted, however, this gain comes at the cost of increased inference latency and energy consumption, which is about W times more than that of a single-frame model. In our particular example, we set W to 5.

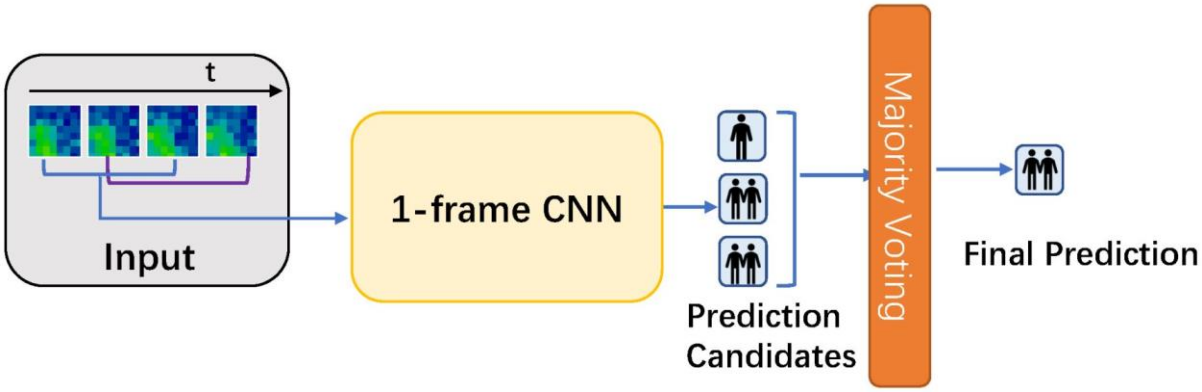


Fig. 4.9. Majority voting CNN. (Image from [43])

In this work, we benchmark the majority voting approach on all the Pareto optimal models on the accuracy versus parameters front of single-frame CNNs. By taking this strategy, we leverage the potential of ensemble learning while remaining mindful of the trade-offs between the accuracy and model complexity and computing efficiency.

4.5.4 CNN-TCN

Although majority voting has the benefit of being simple since it does not require any additional trainable parameters, it has a significant drawback: the inability to assign different weight to each individual infrared (IR) frame inside the sliding window. It is accepted to assume that more recent frames should have

played a more significant part in calculating the persons count, especially when dealing with a larger sliding window (denoted as W). While weighted voting might be used to alleviate this problem, it brings the obstacle of manually fine-tuning the weights provided to each frame, which can be a complex and time-consuming job. With this goal in mind, we use a combination of CNNs and TCNs, as they dynamically incorporate the relevance of recent frames.

In light of these benefits, our architectural design consists of a complex merging of outputs from normal 2D CNN feature extraction processes, each independently applied to a frame of infrared (IR) data. This collection of feature extractors is paired with a single TCN layer to produce a full representation of the data temporal and spatial properties, Figure 4.10 shows an example of this design.

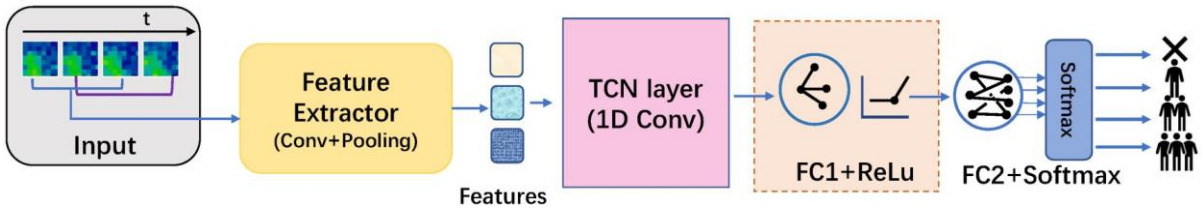


Fig. 4.10. CNN-TCN. (Image from [43])

Specifically, a window of frames is fed to feature extractor block, the produced output is flattened, changing it into a one-dimensional array that captures the essence of the data progression through time. This flattened representation then passes through one or two fully connected (FC) layers, each of which contributes to the complex process of creating predictions and capturing detailed patterns within the data.

We keep the 1D Convolutional kernel size at a small 3x1 to ensure consistency and control within the architectural framework, while using a dilation factor of 1 to guide the convolutional strides. We also increased the TCN layer output channel count to 32, a systematic and thorough search can produce a variety of architectural configurations that combine the strengths of 2D CNN feature extraction and TCN processing, allowing us to successfully describe the complicated dynamics of the data, if the output channel is different.

4.6 Training Procedure

The models are trained using the leave-one-session-out cross-validation technique described in Section 4.2. First, we train basic floating-point models in PyTorch. This training can last up to 500 epochs per fold. We use the ADAM optimizer with a categorical cross-entropy loss function for our optimization, and we start with a learning rate of 10e-3. After finding no progress for 5 consecutive epochs, we utilize a learning rate reduction of 0.3 to solve training loss stagnation. Early stopping is applied if there is no improvement for 10 consecutive epochs. We integrate class-dependent weights into the loss during training due to the considerable class imbalance in the LINAIGE dataset (refer to Table 4.1). The inverses of the class frequencies are used to determine these weights. In this stage, we collect the first seed model results in order to start the NAS procedure.

The following algorithm describes the basic phases of a PIT architecture search. The first step entails warming up for a set number of Steps_{wu} repetitions. All parameters (α, β and γ) are set to 1 and remain constant during this phase that is the same for binary masks Θ. As a result, warming step is equivalent to regular training of the base network with the primary goal of decreasing the task loss function L, and the number of warmup iterations is set by the user. It is worth noting that we choose session 1 for this step and also search loop phase to generalize the process and avoid impacting of other sessions on optimization procedure or possible data leakage.

Algorithm 4.1

1: for $i \leftarrow 1, \ldots, \text{Steps}_{\text{wu}}$ **do** #warmup loop

2: Update W based on $\nabla_W L(W)$

3: end for

4: while not converge **do** #search loop

5: Update W and θ based on $\nabla_{W,\theta} \left(L(W;\theta) + \lambda R(\theta)\right)$

6: end while

7: for $i \leftarrow 1, \ldots, \text{Steps}_{\text{ft}}$ **do** #fine-tune loop

8: Update W based on $\nabla_W L(W)$

9: end for

The second stage comprises the application of NAS. During the search loop, both the model weights (W) and the architectural parameters (θ) are tuned. The major goal of this phase is to minimize the total value of the task-specific loss (L) and one of the two regularization losses (R) outlined in Section 4.3. This is weighted by a regularization strength parameter (λ). The duration of the search phase is governed by an early-stop mechanism. If there is no improvement in this performance after 10 consecutive epochs, the search phase is terminated.

The parameters θ and their associated binary masks Θ are reset to their most recent values and stay unaltered in the final step. This entails picking from the search space the architecture that PIT recognized as the best in the previous step. Following that, the chosen network weights W are either fine-tuned or totally retrained, with the sole goal of reducing the L loss. During this phase, we perform leave-one-out cross-validation while keeping session 1 always in the training set to compute the balanced accuracy of the selected model.

It is sufficient to execute Algorithm 4.1 numerous times while modifying the regularization strength parameter (λ) to obtain unique Pareto points on the graph that balances accuracy versus cost (measured by the number of parameters)

Chapter 5
Experiments and Results

5.1 Setup

To evaluate the performance of our model we use the metrics reported in section 3.6, mainly focusing on the balanced accuracy. When performing cross-validation, we report the mean and the standard deviation of the metrics across each fold. The contribution of each fold is weighted depending on the number of test samples it contributes in relation to the overall number of test samples. We utilize the number of parameters as a proxy for model size to assess the computational complexity of each model in a hardware-independent manner. The code is written in Python (v3.8) based on Pytorch (v2.0), and all experiments are performed on a server equipped with a 32-cores CPU.

5.2 Seed Model Selection

As the seed model selection plays a crucial role on PIT, we refer to the hand-crafted state-of-the-art architectures reported in [43], re-training them using the procedure detailed in Section 4. This selection covers various architectural families on purpose, a deliberate choice to avoid the suboptimal restriction of focusing solely on a single architectural template. In our visual representation, marker shapes represent the family architecture with their corresponding window size, while colors represent the specific architecture. In particular, as shown in Table 5.1, "cross" shape indicates that the model belongs to normal CNN, on the other hand, "square" and "diamond" shapes are used for Majority Voting-CNN and CNN-TCN models, respectively. Concerning the networks, we describe them following the notation introduced in Section 3, denoting convolutional, pooling and fully-connected layers respectively with C, P and FC. For CNN-TCN, the number 32 after TCN indicates the output channels of TCN layer. For instance, the model labeled as [×3]C64-P-C64-FC-FC represents a CNN using 3 frames as input and composed by one convolutional layer with 64 output channels, one pooling layer, one convolutional layer with 64 channels and two Fully-Connected layers. Note that in this work, pooling layers always use square kernel and stride

of dimension 2, while FC have either 64 or 4 (in case of the last layer) output neurons.

Table 5.1
Summary of visual representation notations.

Symbol	Definition
×	CNN Architecture
♦	CNN-TCN Architecture
■	CNN-Majority Voting Architecture
C	Convolutional Layer
P	Pooling Layer
FC	Fully Connected Layer

Table 5.2 reports the scores of each model that we re-train in this work. Thanks to the proposed training approach, we outperform the baseline in most cases, with increases in terms of balanced accuracy ranging from 2.38% to 11.96%. The only exception is represented by the TCN-based architecture, where we obtain a drop in terms of balanced accuracy by 0.54%. It is imperative to underscore a couple of critical insights from our analysis. First, the reduction of the window size leads to a corresponding decrease in performance, and conversely, enlarging the window size creates an overfitting challenge, particularly when coupled with an increased number of convolutional layer channels. For instance, the balanced accuracy of [×3]C8-P-C16-FC-FC is 82.77% that is higher the one of [×3]C64-P-C64-FC-FC with 76.49%, however, the latter has more channel numbers.

When we shift our focus to cost-effective models, with parameters as a key criterion, "Majority Voting" models (represented by squares) emerge as strong competitors. This result is consistent with expectations, as multi-channel Convolutional Neural Networks (CNNs) frequently require a greater number of parameters, particularly in the first Convolutional layer, when the window size (W) exceeds 1. These models have a much greater computational cost, but they provide balanced accuracy advancements, especially when dealing with larger W values. In contrast, "Majority Voting" CNNs attain comparable performance levels while incurring a relatively minor increase in model size.

Table 5.2

Results of different seed models: C, P and FC denote as Convolutional Layer, Pooling Layer and Fully Connected Layer, respectively, and each value after them represents the number of channels.

Model		Bal. Acc.	Acc.	F1	MSE	MAE	No. of Parameters
[×3]C8-P-C16-FC	[43]	0.7762 ± 0.598	0.7804 ± 0.818	0.80 ± 0.07	0.27 ± 0.11	0.24 ± 0.09	1484
	This work	**0.8027** ± 0.03	0.8487 ± 0.02	0.8583 ± 0.02	0.1707 ± 0.02	0.1576 ± 0.02	
[×3]C8-P-C16-FC-FC	[43]	0.7778 ± 0.0898	0.8008± 0.0707	0.8173 ± 0.0540	0.2802 ± 0.1973	0.2259 ± 0.1117	2764
	This work	**0.8277** ± 0.05	0.8616 ± 0.04	0.8665 ±0.04	0.1587 ± 0.03	0.1450 ± 0.04	
[×3]C64-P-C64-FC-FC	[43]	0.6948 ± 0.152	0.6903 ± 0.2412	0.6919 ± 0.2530	0.4609 ± 0.4306	0.3566 ± 0.298	43268
	This work	**0.7649** ± 0.05	0.8168 ± 0.05	0.82 ± 0.04	0.2106 ± 0.05	0.1921 ± 0.05	
[♦3]C8-P-C16-TCN32-FC	[43]	**0.8018** ± 0.05	0.8448 ± 0.01	0.8503 ± 0.015	0.2005 ± 0.06	0.17 ± 0.02	3036
	This work	0.7964 ± 0.21	0.8207 ± 0.03	0.8272 ± 0.03	0.2005 ± 0.038	0.1862 ± 0.035	
[■5]C8-P-C8-FC-FC-Majority	[43]	0.7797 ± 0.098	0.7976 ± 0.0703	0.8117 ± 0.0563	0.3268 ± 0.2707	0.2388 ± 0.1233	1516
	This work	**0.8190** ± 0.0646	0.8686 ± 0.0362	0.8699 ± 0.0359	0.1445 ± 0.033	0.1357 ± 0.0348	
[×1]C8-P-C8-FC-FC	[43]	0.7413 ± 0.0967	0.7309 ± 0.0948	0.7559 ± 0.0719	0.5233 ± 0.4744	0.3407 ± 0.1954	1516
	This work	**0.7707** ± 0.64	0.7815 ± 0.04	0.7983 ± 0.04	0.2938 ± 0.078	0.2818 ± 0.056	
[×5]C8-P-C16-FC	[43]	0.6747 ± 0.0875	0.7708 ± 0.1456	0.7804 ± 0.1269	0.3043 ± 0.2071	0.2525 ± 0.1646	1628
	This work	**0.7942** ± 0.0621	0.7945 ± 0.0807	0.8076 ± 0.0729	0.2773 ± 0.132	0.2294 ± 0.0959	
[×5]C8-P-C16-FC-FC	[43]	0.7773 ± 0.0723	0.8332 ± 0.0642	0.8424 ± 0.0578	0.2019 ± 0.0834	0.1777 ± 0.0687	2908
	This work	**0.8011** ± 0.0494	0.8407 ± 0.0425	0.8467 ± 0.0393	0.1902 ± 0.0588	0.1691 ± 0.0458	
[×5]C64-P-C64-FC-FC	[43]	0.6820 ± 0.1507	0.6009 ± 0.2854	0.6093 ± 0.2891	0.6979 ± 0.8350	0.4978 ± 0.4663	44420
	This work	**0.7555** ± 0.0758	0.8389 ± 0.0408	0.8367 ± 0.0341	0.1775 ± 0.0381	0.1664 ± 0.0397	

In order to enhance NAS search, we chose blue and orange models due to their highest performances, and the green model was selected for its substantial architecture size. We also included the CNN-TCN and CNN-Majority models to ensure the diversity in architectures while maintaining strong performance. Table 5.3 displays these models alongside their associated performance metrics.

Table 5.3
Selected seed models to apply PIT.

Seed Model	Name	Balanced Accuracy [%]	No. of Parameters
[×3]C8-P-C16-FC	CNN-small	80.27	1484
[×3]C8-P-C16-FC-FC	CNN-medium	82.77	2764
[×3]C64-P-C64-FC-FC	CNN-large	76.49	43268
[♦3]C8-P-C16-TCN32-FC	TCN	79.64	3036
[■5]C8-P-C8-FC-FC-Majority	Majority-Voting	81.90	1516

As a further contribution, we select the top three most accurate models of Table 5.3 and we use them as seed for the SuperNet NAS, detailed in Section 4.4. Specifically, the SuperNet algorithm will select among normal convolutions with kernel 3 or 5 and depthwise separable convolutions with kernel 3. In this regard, we use the model_size_regularizer, which represents the total number of parameters in the layer as a function of α parameters. For each seed architecture, we repeat the experiments with alpha (α) in the interval [1e-10, 9e-01].

As shown in Figure 5.1, across all scenarios, we succeed to exceed the maximum balanced accuracy score achieved by the CNN seed models, which was 82.77%. Notably, this improvement comes with spanning more than one order of magnitude in terms of number of parameters. The most accurate result is obtained with the CNN-medium configuration, achieving a stunning 92.74% balanced accuracy with 18,124 parameters and the α value of 3e-06.

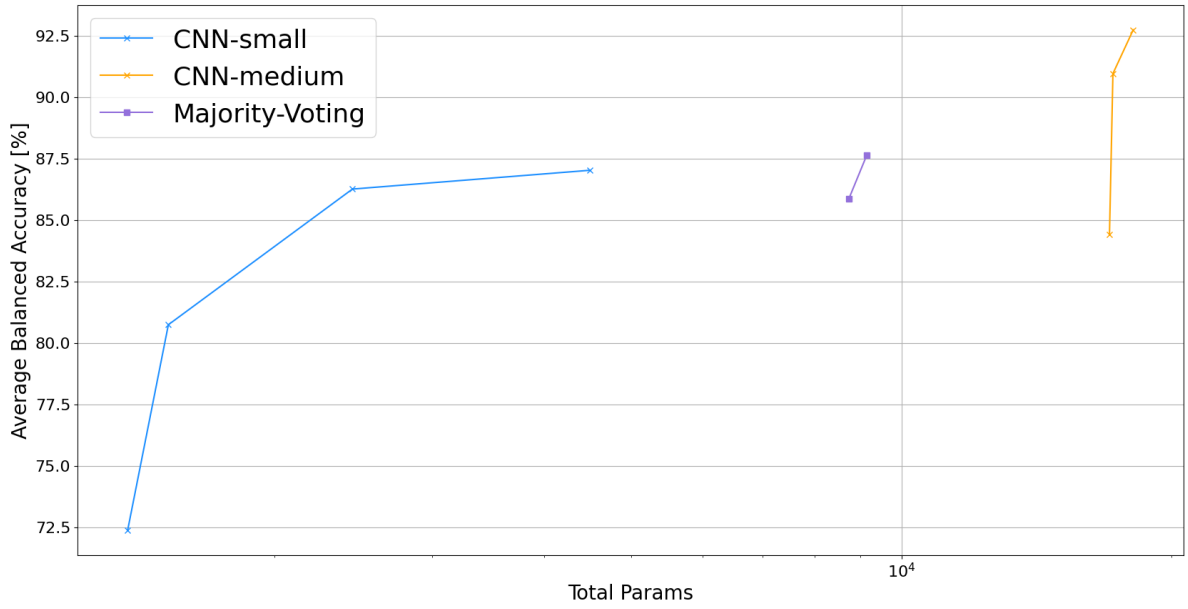


Fig. 5.1. Pareto-Optimal curves of three different seed models obtained by applying SuperNet with different α.

Therefore, we have opted to include three SuperNet architectures from its spectrum for the purpose of conducting PIT. Table 5.4 displays these models alongside their associated performance metrics. Our rationale for this choice lies in the fact that the top two configurations of CNN-small exhibit nearly identical accuracy when compared to Majority-Voting, but with a reduced number of parameters. Additionally, when comparing these two top-performing CNN-small variants with the two lowest-performing configurations of CNN-medium, we observe only a marginal difference in accuracy despite a significant reduction in number of parameters, approximately 75% reduction in size. As a result, we have exclusively selected the top-performing CNN-medium configuration due to its superior accuracy.

Table 5.4
Selected SuperNet models to apply PIT.

Seed Model	Name	Balanced Accuracy [%]	No. of Parameters	α
CNN-small	SuperNet-small-α1	86.26	2444	5e-05
CNN-small	SuperNet-small-α2	87.02	4492	1e-07
CNN-medium	SuperNet-medium	92.74	18124	3e-06

5.3 PIT Architectures

In this section we report the detailed results obtained when applying the PIT-NAS on the architectures reported in Tables 5.3 and 5.4. In particular, we applied a wide range of regularization strength parameters (λ) for all seed models from 1e-10 to 9e-01, with step of 0.1. Regarding following graphs in this sub-section, the blue curves are the Pareto-Optimal of PIT networks and the red cross is the single initial seed model.

5.3.1 Multi-Frame Convolutional Neural Networks

Figure 5.2 reports the Pareto-optimal models for Convolutional Neural Network seed models. Concerning the models found using CNN-small as seed, we achieve up to 80.2% accuracy, the same as the seed model, but with a reduction in size of 41%. If we allow a drop of 4.85% accuracy w.r.t the seed, the reduction reaches 57%, with an accuracy of 75.42%. Regarding CNN-medium, while the spawned architectures do not reach iso-accuracy with the seed model,

we find two points that introduce only slight drops (1.7% and 2.37%), while reducing the size by 5% and 6.3%, respectively. Allowing smaller reduction of 2.7% of accuracy w.r.t seed model, we obtain a size reduction of 41.4% (fourth most-accurate point). As far as CNN-large is concerned, the first interesting point is related to the knee point where we achieved a balanced accuracy of 78.3%, which is 1.8% higher than that of initial seed model. This improvement came alongside a remarkable reduction in size, with only 550 parameters, constituting 98.7% reduction in size. Furthermore, there are several points beyond this knee point that surpass the seed model performance, where the top point reached to 80.1% of balanced accuracy with 15% decrease in size.

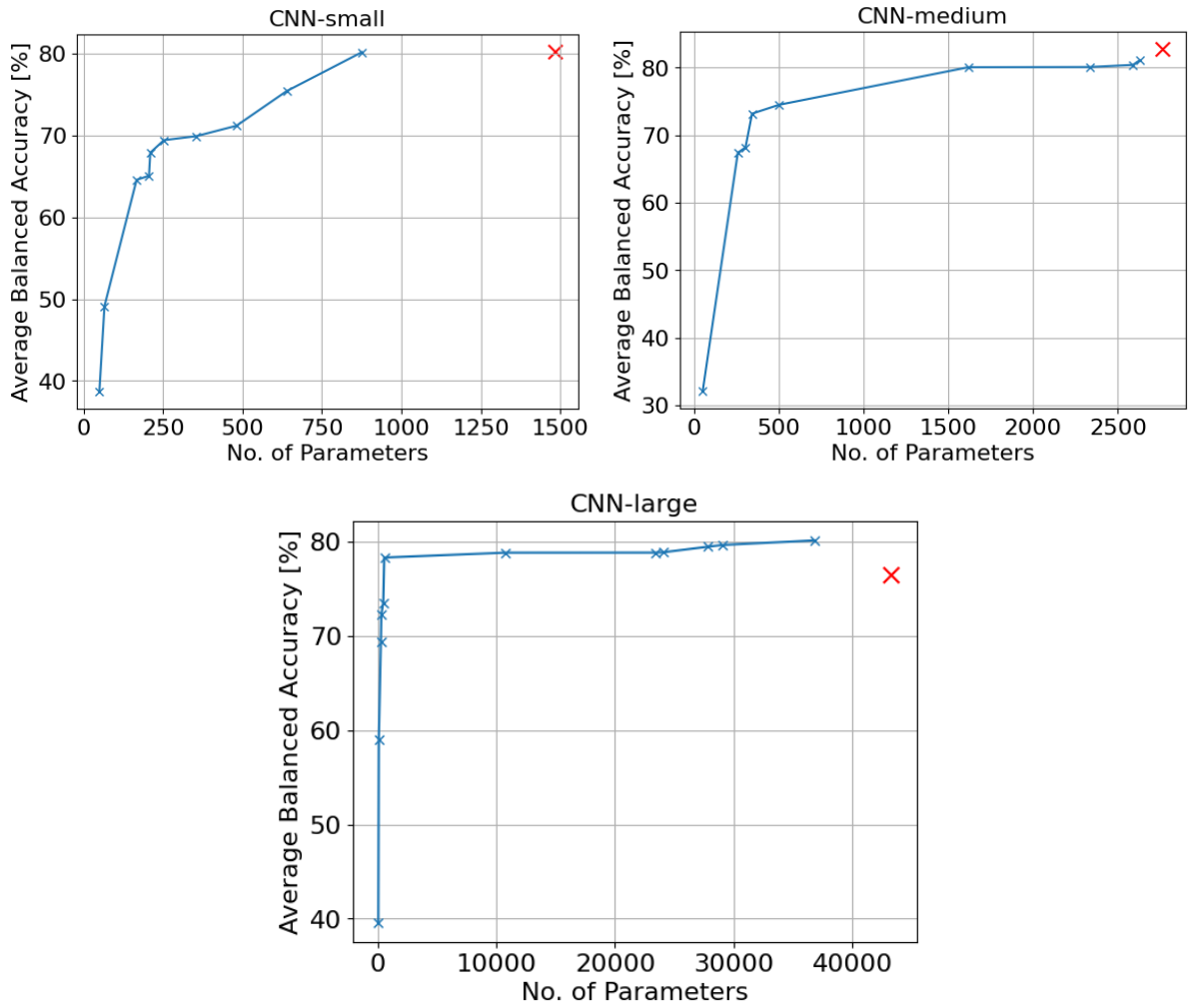


Fig. 5.2. Pareto-Optimal curve of CNN models applying PIT with different λ.

5.3.2 Temporal Convolutional Network

In this particular scenario, PIT did not manage to reach the same balanced accuracy as that of the seed model. Regarding the highest point of the curve in Figure 5.3, this size reduction of 42% came at a cost of 3.2% decrease in balanced accuracy (76.43%) w.r.t the seed. However, if we allow an accuracy drop of 3.6% at the second highest point, we achieve 50% reduction in size. These findings imply that further investigation or alternative optimization approaches may be needed to better understand and enhance the inherent architecture or setup of CNN-TCN.

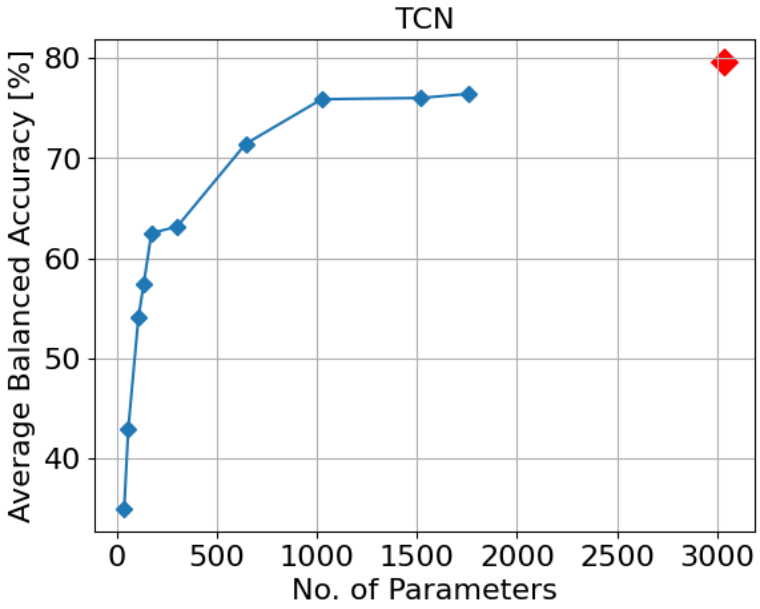


Fig. 5.3. Pareto-Optimal curve of TCN applying PIT with different λ.

5.3.3 Majority-Voting

The fifth highest point of Majority Voting design obtains iso-accuracy with the seed model while introducing a reduction of nearly 62% in size. Another interesting achievement by this model is to exceed the seed model balanced accuracy by 3.65% with 14.4% decrease in size (highest point). Figure 5.4 shows that there are 4 Pareto-Optimal points with higher balanced accuracy than the seed, with a reduction in terms of size ranging from 14.4% to 41.2%.

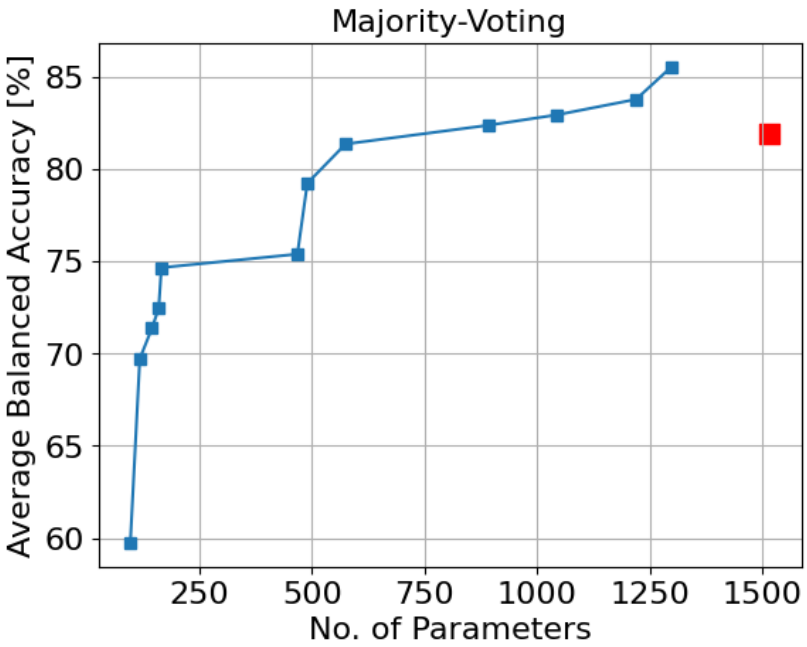


Fig. 5.4. Pareto-Optimal curve of Majority-Voting applying PIT with different λ.

5.3.4 SuperNet Models

Using SuperNet-small-α1 as the seed model, PIT does not achieve results as accurate as the seed models, as shown in Figure 5.5. The most favorable configuration reached the balanced accuracy of 77.52% which is 8.74% lower than that of seed model. Furthermore, the model size was almost identical, with 2329 parameters compared to the 2444 parameters. Regarding SuperNet-small-α2, however, we observe a noticeable disparity between the highest point on the Pareto curve and the performance of the seed model. The best result we attained was a balanced accuracy of 78.9%, which is 8.1% lower than that of the seed model, but it came with a 53% reduction in size. The final architecture is SuperNet-medium configuration that exhibits the highest balanced accuracy among all the seed models. According to Figure 5.5, while we did not quite reach the performance of the seed model (82.63% compared to 92.74%), we managed to achieve a 90.5% reduction in the number of parameters at the highest point on the Pareto curve.

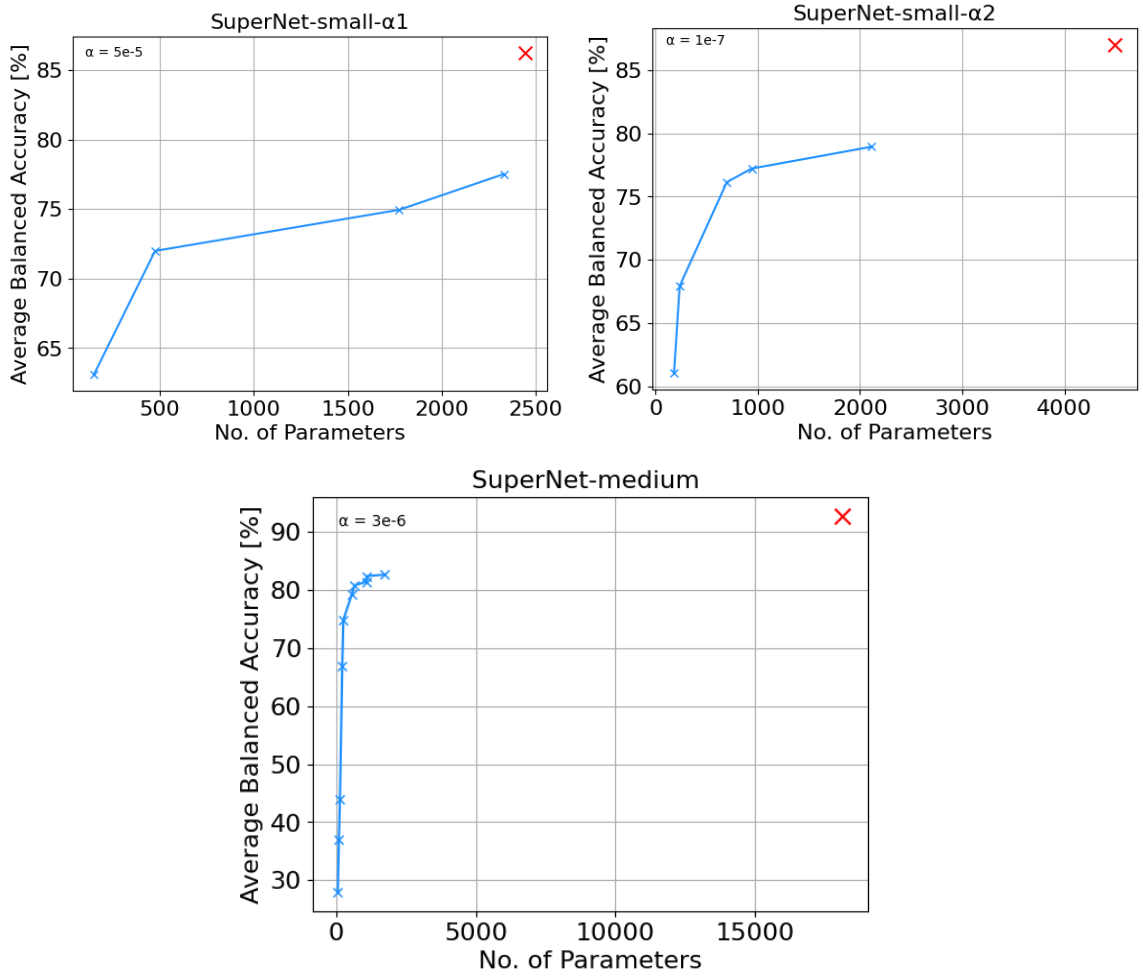


Fig. 5.5. Pareto-Optimal curve of CNN-SuperNet models applying PIT with different λ.

5.4 Architecture Comparison

Figure 5.6 illustrates the outcomes of PIT applied to our initial seed models. Notably, the points spawned for the Majority-Voting outperform the others, with the most accurate point achieving 85.55% balanced accuracy with 1298 parameters. TCNs instead underperform w.r.t the other seeds, with the top point achieving 3.2% less accuracy w.r.t its seed model with size reduction of 42% and 9.1% less accuracy w.r.t the Majority-Voting top point while introducing 26% increase in model size.

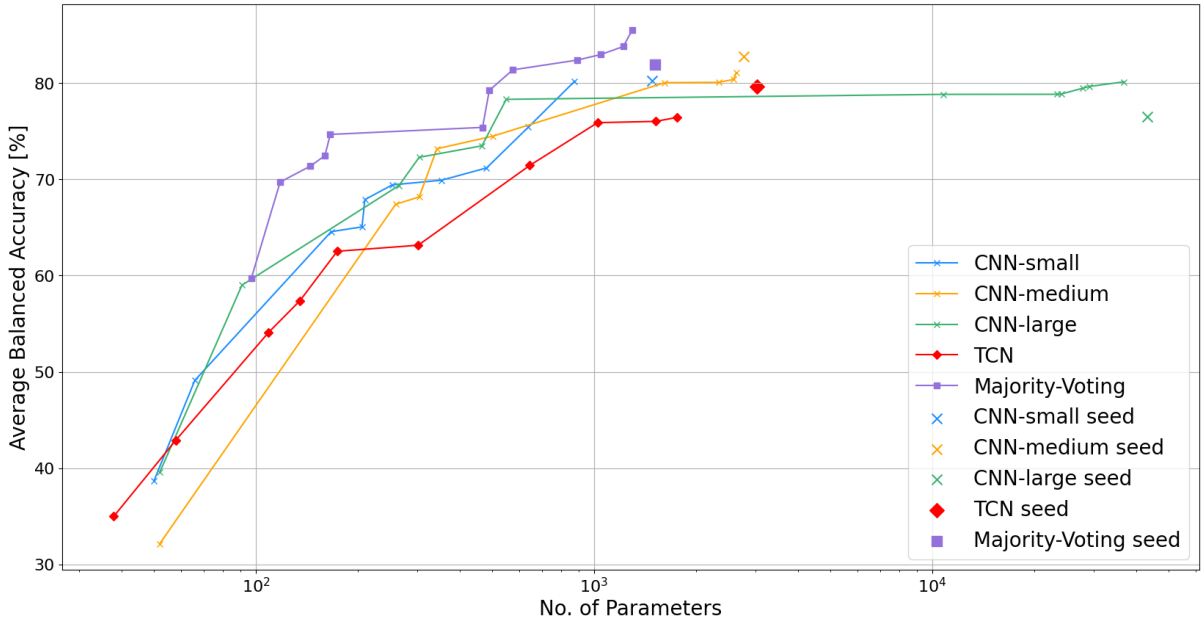


Fig. 5.6. Pareto-Optimal curves of different CNN seed models obtained by applying PIT.

In the context of SuperNet seeds, as shown in figure 5.7, we achieve the best results with SuperNet-medium, achieving a balanced accuracy of 82.63% with 1720 parameters. However, we are unable to reach similar accuracy w.r.t the seeds, we introduce a reduction of up to 90.5% in the number of parameters. In addition, the top point of the lowest curve reaches almost 5% less accuracy w.r.t the highest point of top curve requiring 26% more parameters.

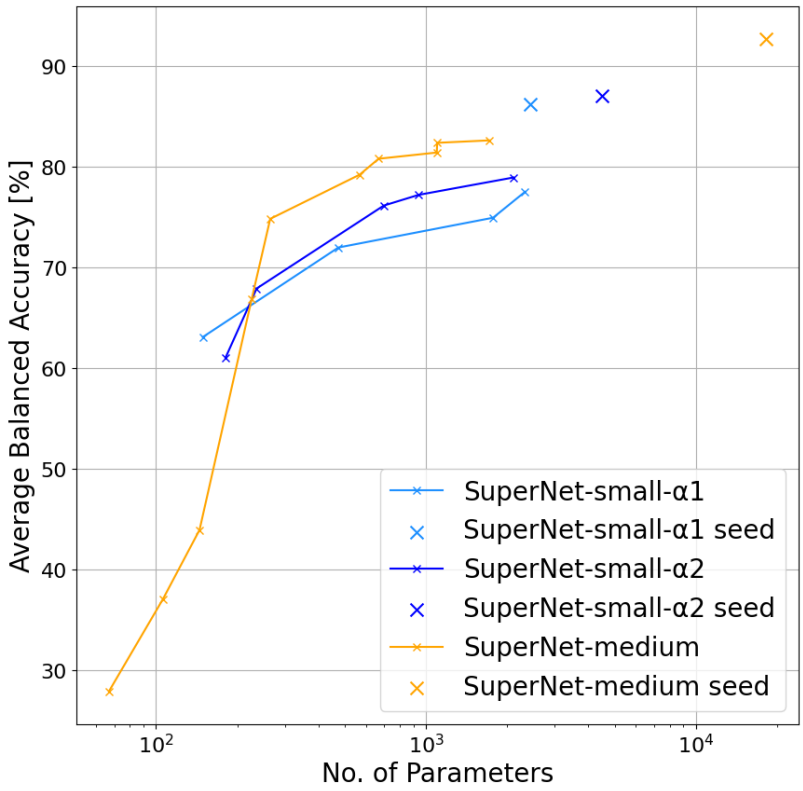


Fig. 5.7. Pareto-Optimal curves of different SuperNet seed models obtained by applying PIT.

To enhance our comparison analysis, we extracted the best-performing Pareto-Optimal front from Figure 5.6 including one point of CNN-small (light blue point), two points of CNN-large (green curve) and all the points of Majority-Voting (purple curve). We next compared these points to the best SuperNet curve from Figure 5.7. As shown in Figure 5.8, the Pareto fronts spawned from the hand-crafted models clearly outperform those generated by the SuperNet technique, except for one outlier with a balanced accuracy of 75.4% and 468 parameters. This could be attributed to the necessity for a broader range of options in terms of layer alternatives and α selection provided by the SuperNet approach.

We also used other state-of-the-art DL approaches [43] for people counting as our baseline for comparison, where top balanced accuracy achieved is 82.7% with 2320 parameters. As shown in Figure 5.8, when comparing our results with mentioned baseline, we achieved up to 2.85% improvement in balanced accuracy by Majority-Voting, while reducing the total number of parameters by up to 44%. Besides, the same balanced accuracy as the baseline is obtained only with 891 parameters (reduction of 61.6%). Although models spawned from SuperNet perform worse at iso-accuracy w.r.t their seeds, the most accurate of orange curve obtains the same accuracy as the baseline requiring 25.8% less parameters. Furthermore, Figure 5.8 highlights the presence of several data points with balanced accuracy greater than that of baseline (black circle) spanning diverse number of parameters. This variety provides valuable possibilities for selecting an architecture tailored to specific task requirements.

Precise inspection of these graphs reveals that PIT provides a diverse spectrum of Pareto-optimal structures. Across all studied scenarios, PIT-generated networks display great effectiveness in both exceeding and matching the accuracy levels achieved by the seed models while concurrently requiring fewer parameters for model deployment. These findings illustrate PIT adaptability and efficacy in obtaining higher performance while optimizing model complexity, demonstrating its potential for improving neural architecture design and implementation.

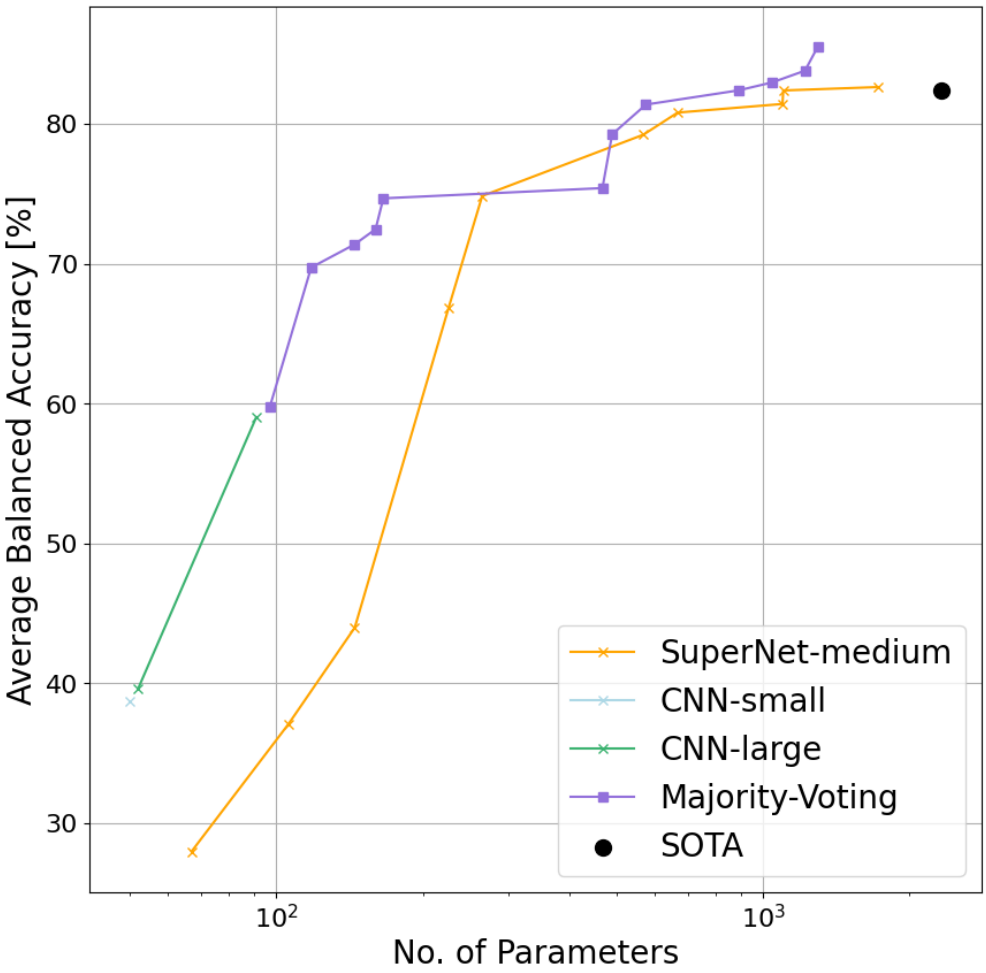


Fig. 5.8. Comparison between best-performing Pareto-Optimal front of the seed models with and without applying SuperNet.

Conclusion

In this work, we explore the effectiveness of neural architectural search (NAS) on ultra-low-resolution infrared (IR) data, with the goal of finding accurate yet small models that can be deployed on the IR sensors collecting the data.

Specifically, we apply two NAS algorithms, Pruning-in-Time (PIT) and SuperNet, on an open-source IR dataset for people counting. We use as seed models 8 state-of-the-art hand-crafted models, showing that we can achieve an improved balanced accuracy of up to 2.85%, while also reducing the size of the model by 44%. At iso-accuracy, we are instead able to reduce the memory footprint of the model by 61.6%, effectively showing the validity of NAS techniques on non-common kind of data such as IR frames.

References

[1] J. Chen et al., , "Deep learning with edge computing: A review," *Proceedings of the IEEE,* pp. vol. 107, no. 8, pp. 1655–1674, 2019.

[2] B. Jiang et al., "Wearable vision assistance system based on binocular sensors for visually impaired users," *IEEE Internet of Things Journal,* pp. vol. 6, no. 2, pp. 1375–1383, 2019.

[3] K. Muhammad et al., "Cost-effective video summarization using deep cnn with hierarchical weighted fusion for iot surveillance networks," *IEEE Internet of Things Journal,* pp. vol. 7, no. 5, pp. 4455–4463, 2020.

[4] A. Burrello et al., "Bioformers: Embedding transformers for ultra-low power semg-based gesture recognition," *in 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE),* p. pp. 1443–1448, 2022.

[5] M. Risso et al., "“Lightweight neural architecture search for temporal convolutional networks at the edge," *IEEE Transactions on Computers,* p. pp. 1–1, 2022.

[6] Z. Zhou et al., "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," *Proceedings of the IEEE,* pp. vol. 107,no. 8, pp. 1738–1762, 2019.

[7] W. Shi et al., "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal,* pp. vol. 3, no. 5, pp. 637–646, 2016.

[8] B. Zoph et al., "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578,* 2016.

[9] A. Wan et al., "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," *Proc. IEEE/CVF CVPR,* p. pp. 12 965–12 974, 2020.

[10] A. Gordon et al., "Morphnet: Fast & simple resource-constrained structure learning of deep networks," *Proc. of the IEEE CVPR,* p. pp. 1586–1595, 2018.

[11] M. Tan et al., "Mnasnet: Platform-aware neural architecture search for mobile," *Proc. IEEE CVPR,* p. pp. 2820–2828, 2019.

[12] H. Cai et al., "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332,* 2018.

[13] M. Risso et al., "Pruning in time (pit): A light-weight network architecture optimizer for temporal convolutional networks," *Proc. 58th DAC,* p. pp. 1–6, 2021.

[14] Y.-L. Hou et al., "People counting and human detection in a challenging situation," *IEEE transactions on systems, man, and cybernetics-part a: systems and humans,* pp. vol. 41, no. 1, pp. 24–33, 2010.

[15] P.-R. Tsou et al., "Counting people by using convolutional neural network and a pir array," *2020 21st IEEE International Conference on Mobile Data Management (MDM),* p. pp. 342–347, 2020.

[16] C. Xie et al., "Privacy-preserving social distance monitoring on micro-controllers with low-resolution infrared sensors and cnns," *Proceedings of the 2022 IEEE International Symposium on Circuits and Systems (ISCAS), ser. ISCAS 2022,* 2022.

[17] C. Perra et al., "Monitoring indoor people presence in buildings using low-cost infrared sensor array in doorways," *Sensors,* pp. vol. 21, no. 12, p.4062, 2021.

[18] C. Raghavachari et al., "A comparative study of vision based human detection techniques in people counting applications," *Procedia Computer Science,* pp. vol. 58, pp. 461–469, 2015.

[19] W. Xi et al., "Electronic frog eye: Counting crowd using wifi," *IEEE INFOCOM 2014-IEEE Conference on Computer Communications,* p. pp. 361–369, 2014.

[20] K. Hashimoto et al., "People count system using multi-sensing applicaapplication," *Proceedings of International Solid State Sensors and Actuators Conference (Transducers'97),* pp. vol. 2. IEEE, pp. 1291–1294, 1997.

[21] I. Udrea et al., "New research on people counting and human detection," *2021 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI),* p. pp. 1–6, 2021.

[22] M. B. Shami et al., "People counting in dense crowd images using sparse head detections," *IEEE Transactions on Circuits and Systems for Video Technology,* pp. vol. 29, no. 9, pp. 2627–2636, 2018.

[23] A. D. Shetty et al., "Detection and tracking of a human using the infrared thermopile array sensor—"grid-eye"," *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT),* p. pp. 1490–1495, IEEE, 2017.

[24] S. Basalamah et al., "Scale driven convolutional neural network model for people counting and localization in crowd scenes," *IEEE Access,* pp. vol. 7, pp. 71 576–71 584, 2019.

[25] V. Nogueira et al., "Retailnet: A deep learning approach for people counting and hot spots detection in retail stores," *2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI),* p. pp. 155–162, IEEE, 2019.

[26] S. D. Khan et al., "Person head detection based deep model for people counting in sports videos," *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS),* p. pp. 1–8, IEEE, 2019.

[27] Panasonic, "Panasonic High Performance Grid-EYE Sensors Reference Specification," *2019.*

[28] F. Daghero et al., "Ultra-compact binary neural networks for human activity recognition on risc-v processors," *ACM CF'21,* p. p. 3–11, 2021.

[29] M. Risso et al., "Robust and energy-efficient ppg-based heart-rate monitoring," *IEEE ISCAS,* p. pp. 1–5, 2021.

[30] A. Gomez et al., "Thermal image-based cnn's for ultra-low power people recognition," *ACM CF'18,* p. p. 326–331, 2018.

[31] A. Metwaly et al., "Edge computing with embedded ai: Thermal image analysis for occupancy estimation in intelligent buildings," *ACM INTESA,* pp. pp. 1015-–1020, 2019.

[32] S. Mashiyama et al., "Activity recognition using low resolution infraredarray sensor," *IEEE ICC,* p. pp. 495–500, 2015.

[33] A. Hayashida et al., "The use of thermal ir array sensor for indoor fall detection," *IEEE SMC,* p. pp. 594–599, 2017.

[34] L. Tao et al., "Home activity monitoring using low resolution infrared sensor," *arXiv preprint arXiv:1811.05416,* 2018.

[35] Z. Liu et al., "Fall detection and personnel tracking system using infrared array sensors," *IEEE Sensors Journal,* pp. vol. 20, no. 16, pp. 9558–9566, 2020.

[36] T. Kawashima et al., "Action recognition from extremely low-resolution thermal image sequence," *14th IEEE AVSS,* p. pp. 1–6, 2017.

[37] M. Gochoo et al., "Novel IoT-based privacy-preserving yoga posture recognition system using low-resolution infrared sensors and deep learning," *IEEE Internet of Things Journal,* pp. vol. 6, no. 4, pp. 7192–7200, 2019.

[38] L. Tao et al., "3d convolutional neural network for home monitoring using low resolution thermal-sensor array," *3rd IET TechAAL,* p. pp. 1–6, 2019.

[39] C.-S. Shih et al., "Multiple-image super-resolution for networked extremely low-resolution thermal sensor array," *IEEE SenSys-ML,* p. pp. 1–6, 2020.

[40] I. M. a. W.-N. Lie, "Two-stream deep learning architecture for action recognition by using extremely low-resolution infrared thermopile arrays," *SPIE IWAIT 2020,* p. pp. 164–169, 2020.

[41] O. Hanosh et al., "Convulsive movement detection using low-resolution thermopile sensor array," *IEEE/CVF CVPR Workshops,* p. pp. 300–301, 2020.

[42] P. Industry, "Grid-eye application note on social distancing. people detection and tracking with ceiling mounted sensors," 2020.

[43] C. Xie et al., "Efficient Deep Learning Models for Privacy-Preserving People Counting on Low-Resolution Infrared Arrays," *IEEE Internet of Things Journal,* pp. vol. 10, no. 15, pp. 13895-13907, 2023.

[44] H. Mohammadmoradi et al., "Measuring people-flow through doorways using easy-to-install ir array sensors," *in 2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS),* p. pp. 35–43, IEEE, 2017.

[45] H. Wang et al., "A lightweight people counting approach for smart buildings," in 2021 13th International Conference on Wireless Communications and Signal Processing (WCSP)," p. pp. 1–5, IEEE, 2021.

[46] R. Rabiee et al., "Multi-bernoulli tracking approach for occupancy monitoring of smart buildings using low-resolution infrared sensor array," *Remote Sensing,* pp. vol. 13, no. 16, p. 3127, 2021.

[47] S. Singh et al., "Non-intrusive presence detection and position tracking for multiple people using low-resolution thermal sensors," *Journal of Sensor and Actuator Networks,* pp. vol. 8, no. 3, p. 40, 2019.

[48] V. Chidurala et al., "Occupancy estimation using thermal imaging sensors and machine learning algorithms," *IEEE Sensors Journal,* pp. vol. 21,no. 6, pp. 8627–8638, 2021.

[49] M. Bouazizi et al., "Low-resolution infrared array sensor for counting and localizing people indoors: When low end technology meets cutting edge deep learning techniques," *Information,* pp. vol. 13, no. 3, p. 132, 2022.

[50] M. Kraft et al., "Low-cost thermal camera-based counting occupancy meter facilitating energy saving in smart buildings," *Energies,* pp. vol. 14,no. 15, 2021.

[51] C. Xie et al., "Energy-efficient and Privacy-aware Social Distance Monitoring with Low-resolution Infrared Sensors and Adaptive Inference," *2022 17th Conference on Ph.D Research in Microelectronics and Electronics (PRIME),* p. pp. 181–184, Jun. 2022.

[52] Khalifa et al., "Imperceptible Image Steganography Using Symmetry-Adapted Deep Learning Techniques," *Symmetry 14,* p. no. 7: 1325, 2022.

[53] I. Goodfellow et al., "Deep learning," *MIT press Cambridge,* p. Vol. 1, 2016.

[54] Gholamalinezhad et al., "Pooling methods in deep neural networks, a review," *arXiv preprint arXiv:2009.07485,* 2020.

[55] A. a. T. S. Saxena, "Predicting bitcoin price using lstm And Compare its predictability with arima model," 2018.

[56] S. Choi et al., "Temporal convolution for real-time keyword spotting on mobile devices," *arXiv preprint arXiv:1904.03814,* 2019.

[57] P. Tsinganos et al., "Improved gesture recognition based on semg signals and tcn," *Proc. IEEE ICASSP,* p. pp. 1169–1173, IEEE, 2019.

[58] S. Bai et al., "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv:1803.01271,* 2018.

[59] Lara-Benítez et al., "Temporal Convolutional Networks Applied to Energy-Related Time Series Forecasting," *Applied Sciences 10,* p. no. 7: 2322, 2020.

[60] M. Sandler et al., "Mobilenetv2: Inverted residuals and linear bottlenecks," *Proc. IEEE CVPR,* p. pp. 4510–4520, 2018.

[61] N. Ma et al., "Shufflenet v2: Practical guidelines for efficient cnn architecture design," *Proc. ECCV,* p. pp. 116–131, 2018.

[62] M. Tan et al., "Efficientnet: Rethinking model scaling for convolutional neural networks," *ArXiv, vol. abs/1905.11946,* 2019.

[63] F. N. Iandola et al., "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 1mb model size," *ArXiv, vol. abs/1602.07360,* 2016.

[64] S. Karagiannakos, "Neural Architecture Search (NAS): basic principles and different approaches," *https://theaisummer.com/,* 2021.

[65] A. Adam, "Neural Architecture Search — Limitations and Extensions," *Towards Data Science,* 2019.

[66] B. Zoph et al., "Learning transferable architectures for scalable image recognition," *Proc. IEEE/CVF CVPR,* p. pp. 8697–8710, 2018.

[67] B. Baker et al., "Designing neural network architectures using reinforcement learning," *ArXiv, vol. abs/1611.02167,* 2017.

[68] E. Real et al., "Large-scale evolution of image classifiers," *Proc. ICML. PMLR,* p. pp. 2902–2911, 2017.

[69] H. Liu et al., "Darts: Differentiable architecture search," *arXiv:1806.09055,* 2019.

[70] D. Stamoulis et al., "Single-path mobile automl: Efficient convnet design and nas hyperparameter optimization," *IEEE J. Sel. Topics Signal Process,* pp. vol. 14, no. 4, pp. 609–622, 2020.

[71] M. Zanghieri et al., "Robust real-time embedded emg recognition framework using temporal convolutional networks on a multicore iot processor," *IEEE Trans. Biomed. Circuits Syst,* 2019.

[72] S. Ioffe et al., "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Proc. ICML.PMLR,* p. pp. 448–456, 2015.

[73] S. R. Chaudhuri et al., "Fine-Grained Stochastic Architecture Search," *arXiv:2006.09581,* 2020.

[74] K. He et al., "Mask r-cnn,," p. [Online]. Available: https://arxiv.org/abs/1703.06870, 2017.

[75] M. Courbariaux et al., "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1or-1," *arXiv preprint arXiv:1602.02830,* 2016.

[76] A. Yazdizadeh et al., "Ensemble convolutional neural networks for mode inference in smartphone travel survey," *IEEE Transactions on Intelligent Transportation Systems,* pp. vol. 21, no. 6, pp. 2232–2239, 2020.