Universidade Estadual de Campinas
Instituto de Computação

Cícera Vanessa Marques Sampaio Sidrim

Neural Architecture Search Analysis for Object Detection

Análise de *Neural Architecture Search* para Detecção de Objetos

CAMPINAS
2022

Cícera Vanessa Marques Sampaio Sidrim

Neural Architecture Search Analysis for Object Detection

Análise de *Neural Architecture Search* para Detecção de Objetos

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestra em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientadora: Profa. Dra. Sandra Eliza Fontes de Avila

Este exemplar corresponde à versão final da Dissertação defendida por Cícera Vanessa Marques Sampaio Sidrim e orientada pela Profa. Dra. Sandra Eliza Fontes de Avila.

CAMPINAS
2022

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Si14n

Sidrim, Cícera Vanessa Marques Sampaio, 1996-
Neural architecture search analysis for object detection / Cícera Vanessa Marques Sampaio Sidrim. – Campinas, SP : [s.n.], 2022.

Orientador: Sandra Eliza Fontes de Avila.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Detecção de objetos. 2. Aprendizado de máquina. 3. Busca de arquitetura neural (Aprendizado de máquina). I. Avila, Sandra Eliza Fontes de, 1982-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações Complementares

Título em outro idioma: Análise de neural architecture search para detecção de objetos
Palavras-chave em inglês:
Object detection
Machine learning
Neural architecture search (Machine learning)
Área de concentração: Ciência da Computação
Titulação: Mestra em Ciência da Computação
Banca examinadora:
Sandra Eliza Fontes de Avila [Orientador]
Levy Boccato
Aurea Rossy Soriano Vargas
Data de defesa: 25-07-2022
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0001-9016-8402
- Currículo Lattes do autor: http://lattes.cnpq.br/6915865213064806

Universidade Estadual de Campinas
Instituto de Computação

Cícera Vanessa Marques Sampaio Sidrim

Neural Architecture Search Analysis for Object Detection

Análise de *Neural Architecture Search* para Detecção de Objetos

Banca Examinadora:

- Profa. Dra. Sandra Eliza Fontes de Avila (Orientadora)
  Universidade Estadual de Campinas
- Prof. Dr. Levy Boccato
  Universidade Estadual de Campinas
- Dra. Aurea Rossy Soriano Vargas
  Universidade Estadual de Campinas

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 25 de julho de 2022

Acknowledgements

First of all, I would like to thank Prof. Sandra Avila for all the support and encouragement during these years. Not only contributing to my training as a researcher but also as a person, always in an empathic, fair and inspiring way.

I thank the Foundation for Technological Innovations (FITec) for the financial incentive for the development of this research. This study was also funded in part by the Coordination for the Improvement of Higher Education Personnel - Brazil (CAPES) - Finance Code 001.

I thank UNICAMP, especially the Instituto de Computação (IC) and RECOD Lab (Reasoning for Complex Data) for the opportunities and for providing all the necessary structure to carry out this research.

I want to thank my family (Francisca, Roque and Larissa) for their encouragement throughout my training. In particular, Raíssa and Flávia, who welcomed me in Campinas and shared the daily routine with me, providing all the support necessary for me to continue and strong support in the most difficult moments.

Finally, I also thank the friends who were present on this journey with words and actions of support and encouragement, especially Dandara, Ed, Andressa and Bruno. Also to Chanely and Laysa for their support in the final stage.

Resumo

Uma das etapas do processo de automação do aprendizado de máquina (AutoML, do inglês *Automated Machine Learning*) é a busca da arquiteturas de redes neurais (NAS, do inglês *Neural Architecture Search*), que consiste na construção de arquiteturas de redes neurais profundas de forma automatizada. O objetivo principal dessa tecnologia é reduzir os esforços manuais e o tempo gasto nessa etapa, gerando automaticamente uma arquitetura propícia para um determinado conjunto de dados. A primeira rede do tipo NAS proposta surgiu em 2017 e exigia um alto poder computacional para ser executada. Desde então, várias melhorias foram propostas com o intuito de reduzir o custo computacional das NAS, tais como novas estratégias de busca baseadas em algoritmos evolutivos e em descida do gradiente, além de novas formas de avaliar uma rede gerada por uma NAS. Apesar dos avanços, ainda não vivemos a realidade onde os esforços e custos são reduzidos, e os resultados são surpreendentes. Nesta dissertação, investigamos a literatura de modo a identificar soluções baseadas em NAS para a tarefa de detecção de objetos. Consideramos um cenário de execução com poucos dados e uma infraestrutura menos robusta em relação aos experimentos relatados na literatura, apresentando assim uma visão mais realista do uso da NAS, trazendo à tona dificuldades enfrentadas e justificando o porquê de ainda não estarmos vivendo esta realidade. Nossas investigações foram realizadas em quatro redes NAS (DetNAS, SP-NAS, PC-DARTS + SSD e NAS *without training*) e duas ferramentas open-source de AutoML (H2O.ai e UFOD). Executamos a rede YOLOv5 em quatro diferentes datasets (Isoladores, *Blood Cell Count and Detection*, *Aquarium* e *Oxford Pets*). Os resultados da YOLOv5 foram comparados com os resultados da SP-NAS, única rede NAS para detecção de objetos cuja execução foi factível considerando as limitações existentes (poder computacional, disponibilidade e funcionamento do código) durante a etapa de experimentos. Para todas as bases de dados, os resultados da SP-NAS não superaram a YOLOv5.

Abstract

One of the steps in the machine learning automation process (AutoML) is the *Neural Architecture Search* (NAS), which consists of building deep neural network architectures in an automated way. The main objective of this technology is to reduce manual efforts and the time spent in this step, automatically generating an appropriate architecture for a given dataset. The pioneering NAS network appeared in 2017 and required high computational power. Since then, several improvements have been proposed to reduce the computational cost of NAS, such as new search strategies based on evolutionary algorithms and gradient descent and, as well as to evaluate a network generated by a NAS. Despite advances, we still do not live in a reality where efforts and costs are reduced, and the results are surprising. In this Master thesis, we investigate the literature to identify NAS-based solutions for object detection tasks, taking into account a scenario with few data and a less robust infrastructure concerning the experiments reported in the literature, thus presenting a more realistic view of its use, bringing to light the difficulties faced and justifying why we are not yet living this reality. We investigated four NAS networks (DetNAS, SP-NAS, PC-DARTS + SSD, NAS without training) and two open-source AutoML tools (H2O.ai and UFOD). We executed the YOLOv5 network in 4 different datasets (Insulators, BCCD, Aquarium, and Oxford Pets). The results of YOLOv5 were compared with the results of SP-NAS, the only NAS network for object detection whose execution was feasible given the existing limitations(computational power, availability, and code functioning) during the experiments stage. For all datasets, the SP-NAS results did not outperform YOLOv5.

List of Figures

List of Tables

List of Abbreviations and Acronyms

Auto-FPN	Automatic Feature Pyramid Network
CNN	Convolutional Neural Network
DARTS	Differentiable Architecture Search
DL	Deep Learning
FCN	Fully Convolutional Network
FITec	*Fundação para Inovações Tecnológicas*
FPN	Feature Pyramid Network
GPU	Graphics Processing Unit
MAE-DET	Maximum-Entropy Detection
ML	Machine Learning
MS-COCO	Microsoft Common Objects in Context
NAS	Neural Architecture Search
NAS-FCOS	Neural Architecture Search Fully Convolutional One-Stage
NAS-FPN	Network Architecture Search Feature Pyramid Network
OPANAS	One-Shot Path Aggregation Network Architecture Search
R-CNN	Region-based Convolutional Neural Network
RL	Reinforcement Learning
RoI	Region of Interest
RPN	Region Proposal Network
SM-NAS	Structural-to-Modular Neural Architecture Search
SP-NAS	Serial-to-Parallel Network Architecture Search
SVM	Support Vector Machine
YOLO	You Only Look Once

Contents

Chapter 1

Introduction

Artificial Intelligence (AI) is increasingly present in many businesses' strategic processes and decision-making. According to the Global AI Adoption Index 2021[1], 21% of Information Technology professionals interviewed in Latin America said they use AI in their organizations, and 40% of Brazilians reported using AI in the companies. The report also revealed a strong trend toward companies making significant investments in AI over the next years. For this to occur in a "healthy" way, we must consider the democratization of the use of AI.

When we talk about democratizing access to Artificial Intelligence, we must consider the use of solutions based on this technology in an operable way by people without specialized knowledge of the technology to add value to the scope of work, avoiding the creation of skill gaps that reproduce exclusion and preventing the restriction of that technology to only a largely homogeneous group. When it comes to in-depth learning and the steps involved in this process, we currently have a technology that is not accessible due to the need for specialists in the entire process of technology applicability, in addition to the need for a robust, high-cost infrastructure.

We can consider the availability of pre-trained models as a first step to making the use of Machine Learning (ML) and Deep Learning (DL) more accessible, saving the architecture construction steps and significantly reducing the high computational cost. However, transfer learning requires specialized knowledge to make adjustments to the model in the data set to be used, and despite the cost reduction, there still is a need for a slightly more elaborate infrastructure.

One way to make this technology more achievable would be to automate the process, allowing it to be used more user-friendly and closer to the reality of people without specific knowledge of Machine Learning. In this scenario, Automated Machine Learning [28], for short AutoML, has a significant role in the democratization of ML, DL, and AI.

AutoML is the process of automating the tasks of applying machine learning to real-world problems, reducing the demands on data scientists, and providing the possibility for domain experts to develop their ML process with a limited computational budget [28].

Among the steps of an ML/DL process, the definition of the neural network architecture certainly is one of the most relevant. The automation of this step is known

[1] https://newsroom.ibm.com/image/IBM%27s+Global+AI+Adoption+Index+2021_Executive-Summary.pdf

as **Neural Architecture Search (NAS)**. This concept received much attention from academia and industry after the excellent performance achieved by a neural network built from a NAS [75]. Zoph and Le used 800 GPUs for 28 days to generate the neural network, an unrealistic infrastructure for most people interested in this strategy.

Since then, new solutions based on NAS have been emerging. The literature has advanced by proposing spaces and search strategies [11, 34, 75, 76] and network evaluation techniques [62, 63] that accelerate the execution of these networks and reduce computational power. However, despite the improvements, we are still far from having a NAS-based solution that can be used with low computational power and encompasses the most diverse types of visual recognition tasks in the most diverse domains.

In this Master's thesis, our primary focus is to develop a NAS-based solution for object detection tasks, considering low computational power (less than 8 GPUS) and a dataset with few images (less than 10,000 images).

1.1 Motivation and Challenges

As we live in the data age, where machine learning techniques have become indispensable for our interaction with software and electronic devices, we involuntarily produce various information daily. However, developing this type of technology requires specialists and investment in resources that enable the execution of the techniques.

AutoML is a possible alternative to make the use of ML reachable to people with only business domain knowledge. Furthermore, we can achieve better results in less time by automating processes in costly steps such as building and selecting models (NAS-based solutions). Allied to this, this work, partially supported by *Fundação Para Inovações Tecnológicas*[2] (FITec), had as its initial purpose to develop a NAS-based solution for fault detection in electrical network insulators. Our goal to provide democracy of access to machine learning techniques matched the idea of building a solution with the limited infrastructure and the target audience that would maintain it.

By analyzing the NAS architectures, we noticed that large datasets are used to train the generated architectures, such as ImageNet (1.2 million images) [7] and MS-COCO (300 thousand images) [33]. However, this amount of data is not always available in our daily life. For our problem, we have a small set of image data (up to 10,000) provided by FiTec, since there is no public data for insulator images.

Another reality described in the NAS-based works concerns the computational power explored to execute these networks. It is well-known that NAS networks require many GPUs, but as advances in the area took place, some solutions executing on simpler infrastructures also emerged. However, not all of these solutions encompass the task of object detection (the subject of this Master's thesis) — the first detection-oriented networks were published in 2019: NAS-FPN [14], Auto-FPN [66], and DetNAS [4].

[2] `http://www.fitec.org.br`

1.2 Goals

The main objectives of this Master's thesis are:

- To investigate the literature on AutoML, specifically the works related to Neural Architecture Search, identifying possible NAS-based solutions for the object detection task[3].
- To analyze the viability of applying NAS networks given an infrastructure with low computational power (less than 8 GPUS) and a limited data set (less than 10,000 images).

1.3 Research Question

The main research question that this dissertation aims to answer is:

> Considering an infrastructure with low computational power and a limited set of images, how can an architecture based on Neural Architecture Search be used in an object detection process that generates results comparable to those of handcraft networks?

1.4 Contributions

We summarize our main contributions as follows:

- We provide a comprehensive NAS state-of-the-art review for the object detection task. We grouped the NAS according to the components of the detectors, including the backbone, neck, head, and region proposal network.
- From a realistic point of view of those who have restrictions to execute these methods, we report the failed attempts of using Neural Architecture Search to detect objects.

1.5 Outline

We organized this Master's thesis as follows:

Chapter 2 – Fundamental Background: We review the fundamental concepts to understand this Master's thesis, including 1) Object Detection: We overview the generic object detection task, emphasizing the detector's architectural pattern and the state-of-the-art detection networks; 2) Automatic Machine Learning: We describe the steps of the Automated Machine Learning pipeline process; and 3) Neural Architecture Search: We detail the search step of the NAS, structuring the concepts according to the phases of the search process.

[3]In March 2019, when we began this Master's thesis, there were no NAS proposals for object detection.

Chapter 3 – Literature Review: We present a NAS literature review focusing on architectures for object detection tasks, classifying them according to the searched detector component.

Chapter 4 – Methodology: We describe the methodology, datasets, and selection of the NAS networks process.

Chapter 5 – Experiments: We report the process and the results obtained by hand-crafted and NAS-based networks, describing all the attempts and difficulties.

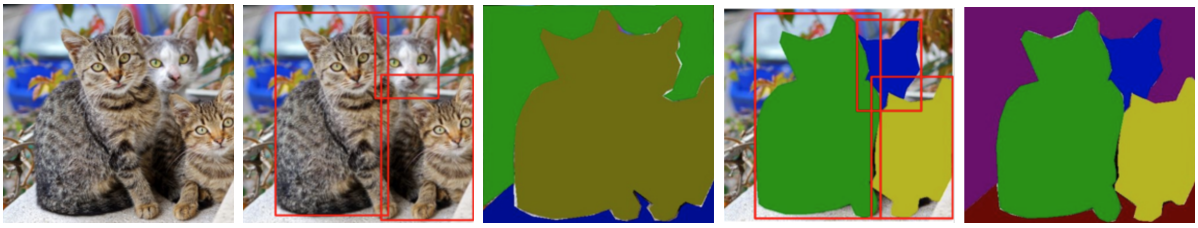
Chapter 6 – Conclusion: We raise a reflection on the democratization provided by AutoML and NAS based on the experience lived during the research development.

Chapter 2

Fundamental Background

2.1 Object Detection

There are several fundamental tasks related to visual recognition, such as **image classification**, which aims to recognize categories of objects in a given image [4] (Figure 2.1(a)), **object detection**, which, in addition to identifying the category to which the object belongs, also predicts its location through a bounding box (Figure 2.1(b)), **semantic segmentation**, which aims to assign each pixel of an image to a semantic class label (Figure 2.1(c)), **instance segmentation**, which distinguishes instances of different objects through pixel-level segmentation masks [36] (Figure 2.1(d)), and **panoptic segmentation** (Figure 2.1(e)) which is a combination of semantic and instance segmentation, where in addition to classifying all pixels in the image as belonging to a class label, we also identify which instance of that class they belong to.



(a) (b) (c) (d) (e)

Figure 2.1: Comparison of the different challenges of visual recognition. (a) Classification, (b) Object detection, (c) Semantic segmentation, (d) Instance segmentation, (e) Panoptic segmentation. Figure reproduced from Daniel Mechea (`https://medium.com/@danielmechea/what-is-panoptic-segmentation-and-why-you-should-care-7f6c953d2a6a`)(Accessed: 12 July 2022).

In this section, we focus on the object detection task, presenting the challenges and the main convolutional neural network architectures developed in recent years for this task (Section 2.1.1). Also, we detail the elements of an object detector (Section 2.1.2).

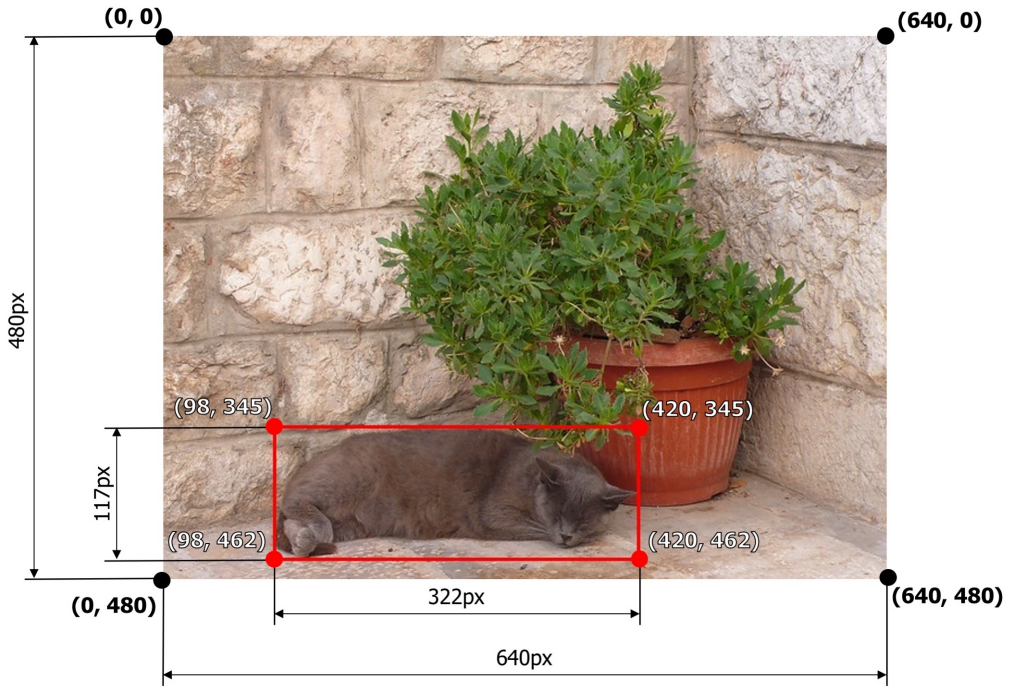


Figure 2.2: An example image with a bounding box (in red color) from the MS-COCO dataset [33]. Figure reproduced from Albumentation (`https://albumentations.ai/docs/getting_started/bounding_boxes_augmentation`).

2.1.1 Generic Object Detection

Object detection aims to develop computational models and techniques to answer one of the most fundamental questions of computer vision: *Which objects are where?* [77], that is, we are concerned with knowing the category of each object within the image as well as the corresponding locations.

To describe the spatial location of an object, we usually use a bounding box that is nothing more than a rectangle with coordinates (x, y) — originating from the top corner of the image — indicating the central axis of the bounding box, its height (h) and width (w) (Figure 2.2). In a data annotation process, we indicate the object's location through the measurements of a bounding box, and we use a class to reference which object is present in that bounding box [70].

Typically, there are two types of detection [36]: **specific object detection**, that is, an example of a match that aims to identify particular objects, places, or people (e.g., Grace Hooper's face, Christ the Redeemer), and **generic object detection**, which seeks to recognize different instances of objects of a given generic category (e.g., dogs, cups) [17].

The purpose of a generic object detection algorithm is to achieve two concurrent goals: 1) high quality/accuracy encompassing good distinction, i.e., the ability to differentiate and recognize the wide variety of real-world objects concisely, and 2) robustness in identifying the many variations in appearance of instance objects of the same category, achieving high efficiency in the sense of performing all real-time detection with acceptable processing and storage demands [72].

To achieve these goals, some challenges need to be overcome, such as a wide range of

variations between classes, many object categories, and how changes in appearance (e.g., different angles, illumination variation, complex backgrounds) [72, 77] are determined by the conditions under which the images were captured. When we think about the context of detecting isolates in electricity distribution networks, the diversity of images can be pretty broad, with significant variations in the background (urban, rural, type of soil), the type of insulator, the angle capture, lighting), as illustrated in Figure 2.3.

(a) Point of view variations (b) Illumination variation (c) Complex background

Figure 2.3: Challenges in insulator detection. Images from our dataset.

To overcome these difficulties and carry out the detection task, in the last years, extensive progress has been witnessed in the field of object detection that we can divide into two historical periods (Figure 2.4): The traditional period of object detection (before 2014) is represented by using handcraft features to detect objects due to the lack of good image representations and exclusive computational resources, and the period of deep learning-based approaches [77]. The latter, marked by the use of convolutional neural networks (CNNs), stands as the state-of-the-art in the object detection field.

CNN-based methods for object detection can be divided into two-stage and one-stage object detectors. The first CNNs proposed for object detection followed the two-stage architectural pattern that includes a first stage called the Region Proposal Network (RPN), where the candidate objects are proposed before the second stage is carried out, where the classification and regression of the bounding boxes take place. These models have higher accuracy rates but are generally slower. On the other hand, one-stage detectors do not have this step, and, despite being faster, they can generate less accurate results [23].

Two-stage Object Detector

R-CNN (Region-based Convolutional Neural Network) architecture introduced the two-stage approach. Girshick et al. [15] proposed using a selective search algorithm to extract 2000 proposed regions from an input image. These regions are scaled to a fixed size and then fed to the CNN model (e.g., AlexNet [27]) pre-trained on ImageNet [7], which extracts a vector of 4096 fixed-size feature vectors used to feed a linear Support Vector

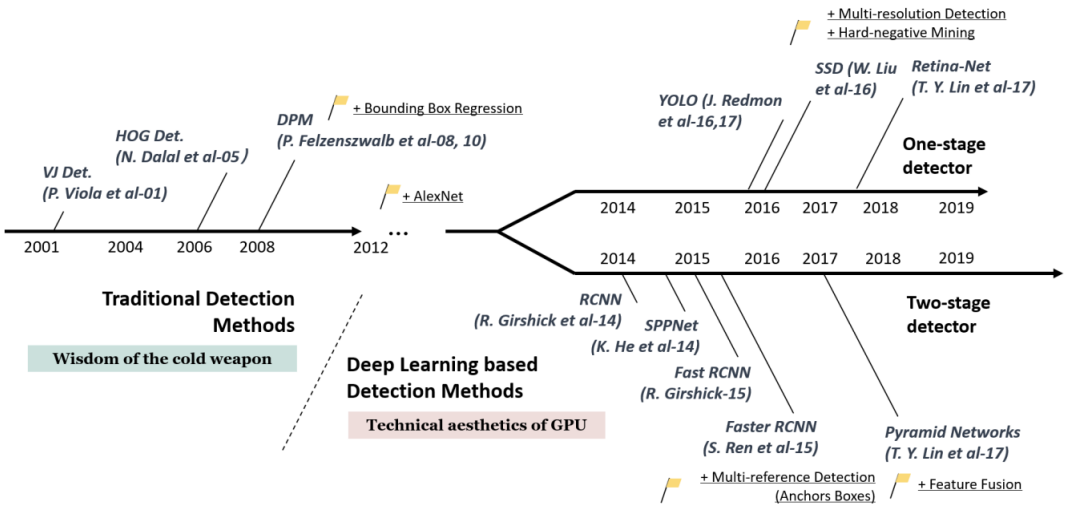


Figure 2.4: Object detection milestones. Figure reproduced from Zou et al. [77].

Machine (SVM) for verifying the existence of an object within each of these regions and classifying them.

However, this method presents some problems, such as the need for a considerably long time to sort the large number of region proposals generated by the image (∼2,000 regions per image) and the fact that the selective search algorithm is fixed and ends up not allowing learning at this stage of the process, thus possibly leading to the proposal of poor regions.

Similar to R-CNN, **Fast R-CNN** [15], proposed by the same authors, improved the detection speed obtained through the execution of the feature extraction process before the generation of the region proposals, thus reducing the number of CNN executions from 2000 to 1. The replacement of the SVM by a softmax layer also contributed to the speed gain of the network.

Despite the speed improvements, one of the major bottlenecks in the networks of the R-CNN family was not solved until the introduction of the **Faster R-CNN** network. To replace the use of the selective search algorithm, Ren et al. [48] suggested the use of the Region Proposal Network (RPN), a type of Fully Convolutional Network (FCN) to predict region proposals with a Region of Interest (RoI) Pooling layer, which is used to classify the image in the proposed region and predict the displacement values for the bounding boxes.

The two-stage detectors that emerged after those already mentioned maintained the same principle of speed increase and maximized computational sharing. For example, **R-FCN** [6] is a fully convolutional network with almost all calculations shared in the entire image, unlike detectors based on regions such as Fast-RCNN and Faster R-CNN, which apply an expensive subnet per region hundreds of times. **Mask R-CNN** [19], in addition to detecting objects in a computationally efficient way, has a branch to predict segmentation masks in each RoI pixel by pixel.

One-stage Object Detector

Approaches based on two-stage detection are computationally expensive, especially when considering the use in mobile/portable devices due to the limited storage and low processing capacity of these devices [36]. Based on this concept, researchers have developed unified detection strategies where the detection process is performed in a single step, eliminating the generation of proposals by regions [72].

YOLO (You Only Look Once) [45] is an example of a one-stage detection network and the forerunner of the local region-based detection technique that uses a single CNN network to classify and locate the object [77]. Through this operation, this network presents speed improvements compared to two-stage detectors, but in terms of accuracy, its loss is noticeable mainly in detecting small objects.

Later, **YOLOv2** [46] added a new feature extractor to its architecture called DarkNet-19 network, applied batch normalization, removed fully connected layers, and performed training at various scales. Available with a YOLOv2, **YOLO9000** [46] is a version of the network used for real-time object detection with the ability to identify more than 9,000 objects per class through hierarchical classification.

The latest version of the YOLO architecture developed by Redmon et al. was **YOLOv3** [47], an underlying 106-layer fully convolutional architecture that is more accurate and faster than previous versions. Unlike YOLOv2, which uses DarkNet-19 as a backbone, YOLOv3 uses DarkNet-53, making the feature extraction process more robust since, instead of 19 convolutional layers, the new backbone has a 53 convolutional layer structure.

Another improvement presented by YOLOv3 refers to detecting small-scale objects, a problem in the previous version. To increase performance in this regard, the network was designed so that layers with convolutions of different sizes could deal with specific object scales. For example, layer 52×52 detects smaller objects, while layer 26×26 detects medium-scale objects.

Another difference refers to the classification of detected objects. Instead of using a softmax function to define class scores assigning the highest score as object classification, YOLOv3 uses logistic regression to predict multiple labels for one object, avoiding situations in which one class includes another. For example, in a dataset, we can have categories such as animal and dog; in an image of a dog, it would not be wrong to define the object as both categories.

Alternative versions based on the YOLO family of architectures were released later. **YOLOv4** [1], for example, provides a high-performance object detector capable of executing on machines with little computational power through the implementation of *bag of freebies* techniques and *bag of specials*.

The bag of freebies consists of a set of methods (for example, cutmix and increased data in mosaic) to change training strategies by increasing data, expanding the variability of input images, and, consequently, designing greater robustness for the detector when dealing with specific images in different environments.

On the other hand, the bag of specials technique acts in post-processing, improving the attributes in a model, such as increasing the receptive field, introducing the attention mechanism, or strengthening the ability to integrate features.

Two months after the release of YOLOv4, **YOLOv5** was released [58][1]. The replacement of the DarkNet framework with the PyTorch framework is the most significant change compared to the last two versions of YOLO. Although DarkNet provides greater control over network operations due to its writing in C, the PyTorch framework allows new network insights to be implemented faster by community contributors.

2.1.2 Object Detector Architecture

An object detector consists of a series of components (Figure 2.5). Regarding one-stage object detectors, three components are standard: backbone, neck, and function head. The two-stage detectors also have the Region Proposal Network (RPN) [68]. All these components play a role in the object detection pipeline, and they are combined in the following sequence:

Backbone: Important for feature extraction. From this component comes the greatest proportion of architectural parameters. Classification networks with backbones are used in many detectors; however, this usage is not ideal as the task objectives are different [4]. In the classification task, the focus is on identifying only the object class, while in the detection task, the objective is to discover each instance of an object and where it is in the image. Some examples of detection backbones are ResNet [20] and ResNeXt [65].

Feature Fusion Neck: It merges features of different scales extracted from the backbone, helping to identify objects better [66]. Feature Pyramid Network (FPN) [31] is a detection feature fusion neck.

Region Proposal Network (RPN) [60]: Only present in two-stage detectors, it generates the region proposals. Composed of a convolution layer followed by two fully connected layers, it performs the classification of the proposed region and the regression of the bounding box [18].

Head: It refines the location of objects and predicts the final classification results [68]. Detectors have widely used two head structures: the fully connected head and the convolution head [64].

2.2 Automated Machine Learning

Whenever we have a dataset and want to collect information using machine learning techniques, we need to go through a series of steps until we reach our goal, such as data pre-processing, domain-oriented feature engineering definitions, and choosing improved models to lead to high predictive power [74]. Building this pipeline requires a highly trained team of human experts with in-depth knowledge of algorithms and statistics.

[1]During the process of presentation and submission of the final version of this research, new versions of the YOLO network were made available (YOLOv6 and YOLOv7).

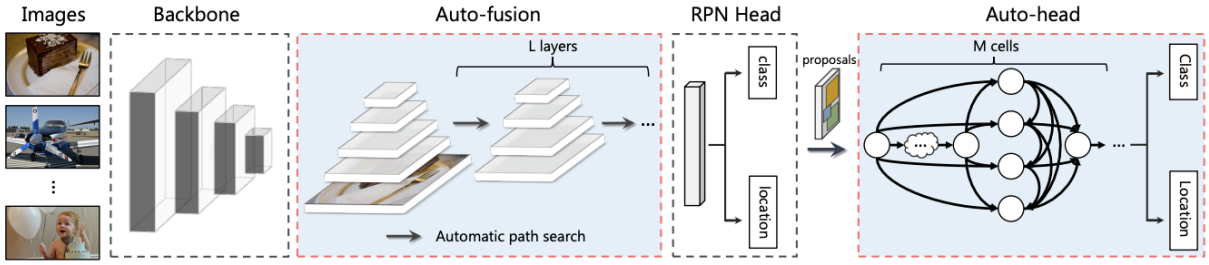


Figure 2.5: Overview of object detector architecture. Figure reproduced from Xu et al. [66].

This entire process is complex and requires interactive trial and error execution, making this task time-consuming and expensive.

To automate as many of these processes as possible without compromising the accuracy of the results, AutoML (Figure 2.6) proposes the development of a system capable of automatically executing all the configuration steps of the machine learning process and eventually generates a prediction model capable of performing any task we define [21].

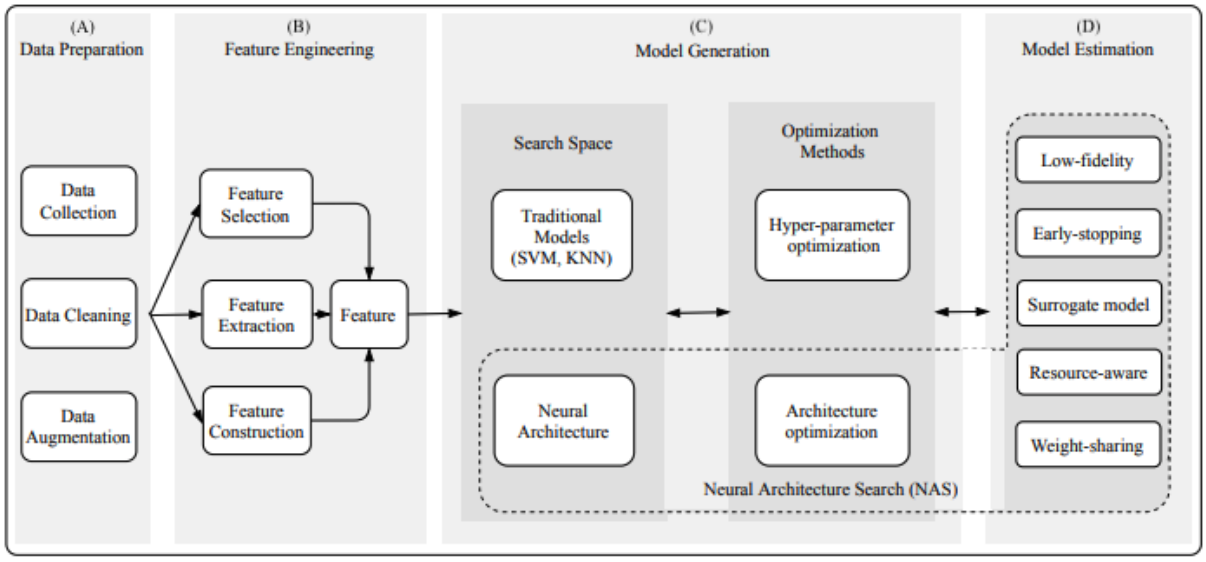


Figure 2.6: Overview of the AutoML pipeline, covering the steps of data preparation, resource engineering, generation, and model evaluation. Figure reproduced from Kaiyong et al. [21].

By automating these steps, we enable faster deployment of ML-based solutions, making them more affordable and taking applications to new levels of custom competence. The impact can be significant [69].

In this section, we describe the main steps of an AutoML-based solution: data preparation, feature engineering, model generation, and model estimation.

2.2.1 Data Preparation

The first step of the AutoML pipeline (Figure 2.6(a)) is the data preparation, which consists of three tasks (Figure 2.7):

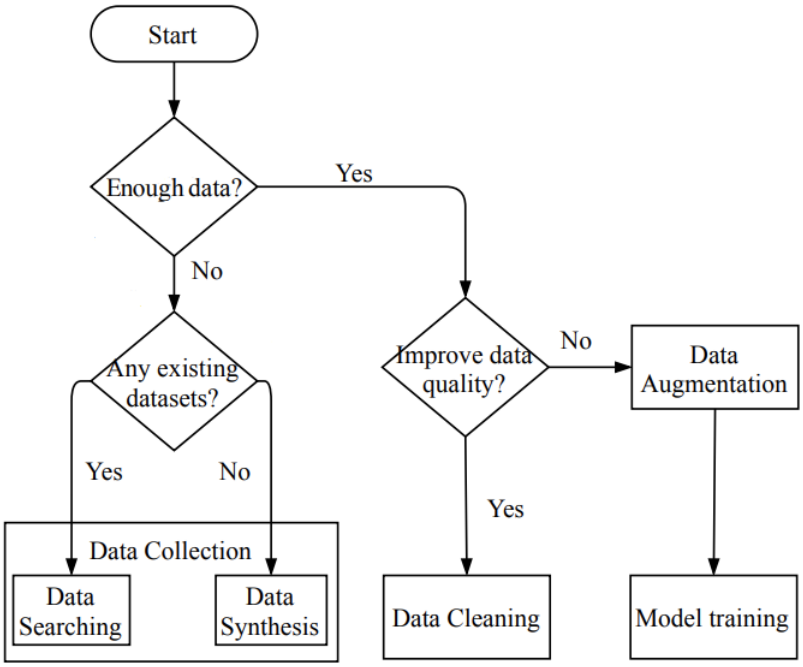


Figure 2.7: Pipeline of the data preparation process. Figure reproduced from Kaiyong et al. [21].

Data collection: This step builds a new dataset or expands an existing one.

Although we have well-structured and robust image datasets, such as CIFAR [26], MS-COCO [33] and ImageNet [7], we need to access visual data with slightly more specific domains through data search tools like Kaggle[2] and Google Dataset Search[3]. However, not all contexts have data available, making it necessary for specific domains to build a new dataset.

When building a new dataset, we can perform data collection by searching data previously captured and obtained in the web domain (Data Searching) or by generating synthetic data (Data Synthesis) using techniques such as Generative Adversarial Networks (GANs) [16].

In addition to the data collection process, it is necessary to carry out the data annotation process for supervised models. To automate this task, we have some tools like Amazon SageMaker Ground Truth[4] or Google AI Platform[5], which allows data labeling through the use of an ML model. These models label data, and humans only label cases where the model cannot predict labels reliably. The low-reliability data in the automatic labeling process is then used for training this ML model to increase its accuracy and reduce the need for manual labeling.

Data cleaning: It identifies and treats noisy data to avoid compromising the model's next steps.

[2] `https://www.kaggle.com`
[3] `https://datasetsearch.research.google.com`
[4] `https://aws.amazon.com/en/sagemaker/groundtruth`
[5] `https://cloud.google.com/ai`

Real-world data is commonly integrated from multiple sources, and the integration process can lead to various errors [9]. Data noise can negatively influence the model training process, affecting the model's performance and accuracy.

Much time is spent in this data preparation phase. To speed up this task, tools such as AlphaClean [25] have been proposed to make data cleaning a hyperparameter optimization problem, and BoostClean [24] automates this process through a solution based on the combination of boosting (a general method for improving the accuracy of any learning algorithm, mainly reducing bias and also reducing variance [51]) and feature selection.

Data augmentation: It expands the existing data set by incorporating new samples created from the original ones. Thus, it is possible to reduce overfitting in models by increasing the training data using information only in this data portion, regularizing the model.

When dealing with image data, there are several transformation techniques [21], such as affine transformations (e.g., rotation, scaling, random clipping, and reflection), elastic transformations (e.g., blur and change contrast, brightness, and channel), and advanced transformations (e.g., random erasure, image blending, clipping [8], and mixing [71]).

AutoAugmentation [5] is a data augmentation tool that uses reinforcement learning (RL) to find optimal image transformation policies from the data itself. Due to the need for powerful computational resources for the reinforcement learning module, an alternative version of this tool is DeepAugmentation[6], which replaces RL with Bayesian optimization, making the process cheaper and faster.

Other data augmentation methods are related to data synthesis. Adversarial noise [40], neural style transfer [41], and GANs [16] are promising but also complicated techniques that can end up diverging from realistic examples.

2.2.2 Feature Engineering

Feature engineering is the second step of an AutoML pipeline (Figure 2.6(b)). In this step, we seek to extract features from raw data to maximize the performance of ML models. In recent years, many automated feature engineering methods have been proposed [3, 73] and modern AutoML frameworks such as H2O[7] and Auto-Sklearn[8] have integrated this step into their functionality set [74]. At this stage, the actions taken are:

Feature selection: Choosing important features within the original feature set removes redundancies to simplify the model, avoids overfitting, and increases performance. Algorithms such as univariate selection, variance limit, feature importance, and correlation matrices are examples of automated strategies integrated into AutoML frameworks [74].

[6] https://github.com/barisozmen/deepaugment
[7] `https://www.h2o.ai`
[8] `https://github.com/automl/auto-sklearn`

Feature extractor: Through mapping functions, the dimensionality of the data is reduced [21], extracting the non-redundant features identified from a set of metrics. Unlike feature selection, feature extraction changes the original features. In this step, AutoML frameworks [21] typically apply Principal Component Analysis (PCA), Independent Component Analysis (ICA), and Linear Discriminant Analysis (LDA), for example.

Feature constructor: In this process, new features are generated from basic features or raw data [21], improving the robustness of the model and increasing the representativeness of the original features, making the model more generalized. Some methods of constructing existing auto features are integrated into the AutoML frameworks [50, 55].

2.2.3 Model Generation

The next step is the generation of models (Figure 2.6(c)), which is divided into:

Search space: In this step, we define the structures to design the models, for example, traditional models (e.g., Support Vector Machines (SVM), k-Nearest Neighbors (kNN), Decision Trees), or Deep Neural Networks (DNNs).

When working with the automatic selection of traditional models, the search space comprises candidate models and their respective hyperparameters. The main objective is the automatic search for an optimization algorithm so that efficiency and performance can be balanced [69].

Considering DNN architectures, Neural Architecture Search (NAS) techniques must be used to generate the models that achieve the best possible performance, in an automated way, with minimal human intervention.

Optimization methods: Alongside the choice of traditional models and the search for architectures, we have optimization methods divided into hyperparameter and architecture optimization.

Optimization in traditional structures involves the search for the best combination of hyperparameters. In NAS methods, the architecture optimization process works differently since, in most cases, the same set of hyperparameters is used during the architecture search phase. After finding the most promising architecture, the set of hyperparameters is redesigned and used to train or refine the final architecture.

As the core of this Master's thesis is Neural Architecture Search (NAS), we detail it in Section 2.3.

2.2.4 Model Estimation

The last step of the AutoML pipeline consists of evaluating the performance of the generated model (Figure 2.6(d)), a process directly included in the model generation stage in cases of architectural search. The simplest way to perform this assessment is by training

the model on a training dataset and estimating the performance on a test set. However, as this method consumes a lot of time and resources, more advanced methods have emerged to speed up this process, but the vast majority are likely to generate a loss in process fidelity. Thus, it is crucial to balance efficiency and effectiveness in an assessment. We detail this step in Section 2.3.3.

2.3 Neural Architecture Search

Neural Architecture Search (NAS) aims to automate network architecture engineering, i.e., to automatically build a neural network architecture from a search space for a given dataset without human experimentation. Although experts have manually designed the most popular existing models with good performance, such manual construction represents a more onerous process that may explore a limited number of possible architectures. Hence, by employing NAS, there is the potential to evaluate a more diverse set of candidate architectures during the search so that it may identify a more suitable neural network.

Elsken et al. [11] characterize the NAS as an approach with three main components (Figure 2.8):

- **Search space:** It defines the design principles that the generated neural architecture will use, providing a set of operations (e.g., convolution, pooling, fully connected) and defining how these operations can be connected to generate an appropriate neural network architecture.
- **Search strategy:** It details exploring the search space (often exponentially large or unlimited) to identify the best architecture, balancing the need to find suitable performing architectures quickly, and considering that premature convergence to a region of optimal sub-architectures should be avoided.
- **Performance estimation strategy:** It estimates the performance of the proposed neural network architectures.

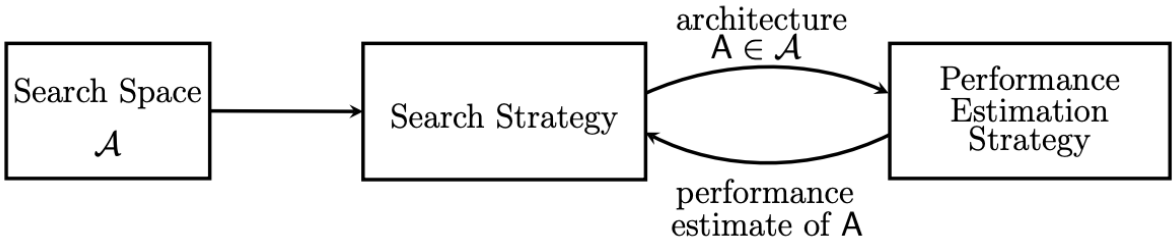


Figure 2.8: Main components of the NAS approach. Figure reproduced from Kaiyong et al. [21].

In this section, we describe each component that constitutes a NAS network, providing architectural details that will facilitate understanding of the architectures investigated in the following chapters of this Master's thesis.

2.3.1 Search Space

The search space defines which neural architectures a NAS approach can discover in principle [11]. The idea is to define a set of basic operations that constitute the search space and how these operations can be connected to build appropriate neural network architectures.

At this stage of the NAS approach, human intervention is necessary to design the specific search space for a given application. For example, if our final goal is to build an architecture for a computer vision task, it would be more appropriate to define the search space so that convolutional networks are explored. If we want a network for language modeling, we will define a space to create recurrent network architectures.

We detail the search spaces commonly used and reported in NAS studies in the following.

Chain-structured Search Space

Considered one of the simplest search spaces, in this projection category[9], we describe network topologies through a sequence of ordered nodes, where each node represents a layer responsible for performing a certain operation. In this set of n layers, layer i receives its input from layer $i-1$ and its output serves as input to layer $i+1$. A more complex version of this search space will allow the inclusion of skip connections between nodes (Figure 2.9).

Elsken et al. [11] state that each of the layers in the search space in a chain structure is associated with one or more specific parameters. Therefore, it is possible to parameterize the search space by:

- Number of layers (possibly unlimited);
- Type of operation that each layer performs (e.g., convolution, pooling, fully connected);
- Hyperparameters associated with the operation (e.g., filters, kernel size, steps to a convolutional layer).

Despite being an easy-to-implement structure with enormous representation capacity, supporting a wide range of operations that can be performed, this representation has disadvantages such as the high consumption of computational power to perform the search for deep networks (when deeper the model, the better is its generalization [21]) and to cover the entire search space exhaustively.

Cell-based Representation

In this case, the search space is based on the repetition of small graphics called cells that contain a set of operations. This structure was defined based on known models in computer vision tasks designed from the repetitions of stacked structures (e.g., ResNet [20],

[9]It is also referred to by the terms Entire-structure Search Space or Sequential-Layer Search Space.

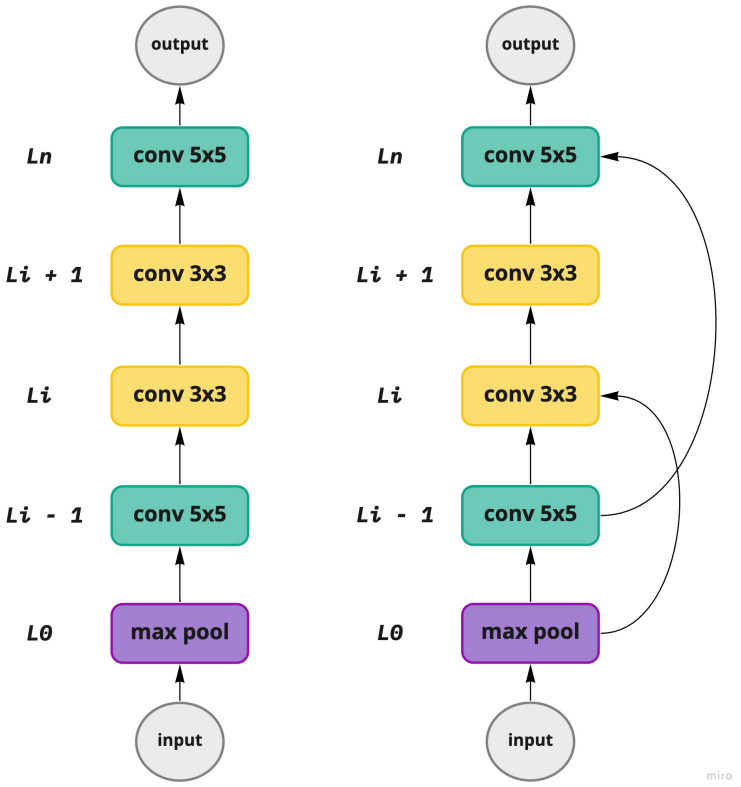


Figure 2.9: Representation of the chain-structured of a deep neural network. Representation without skip connections (in the left) and with skip connections (in the right).

and its variants, like ResNet101 and ResNet152 stack more bottlenecks modules in the ResNet architecture generating new networks). This search space makes it possible to easily reduce or increase the architecture's size by adjusting the number of cell repetitions. Another advantage of this implementation is the speed gain due to the reduced search space.

Zoph et al. [76] implemented an example of the applicability of this search space structure, where they proposed the NASNet network that uses two categories of cells in the construction of the network architecture (Figure 2.10):

- **Normal cell:** Input and output feature maps with the same dimension.
- **Reduce cell:** Output feature maps are reduced by a factor of two in width and height.

Some advantages of using this structure are:

- The size of the search space is drastically reduced due to the possibility of having small cells. Zoph et al. [76] report a seven-fold reduction and increase in performance compared to experiments performed in their previous work [75];
- Reason-based architecture can be more easily transferred to different datasets. In the NASNet [76] experiments, the cells used in the CIFAR-10 dataset are transferred to ImageNet, achieving good performance.

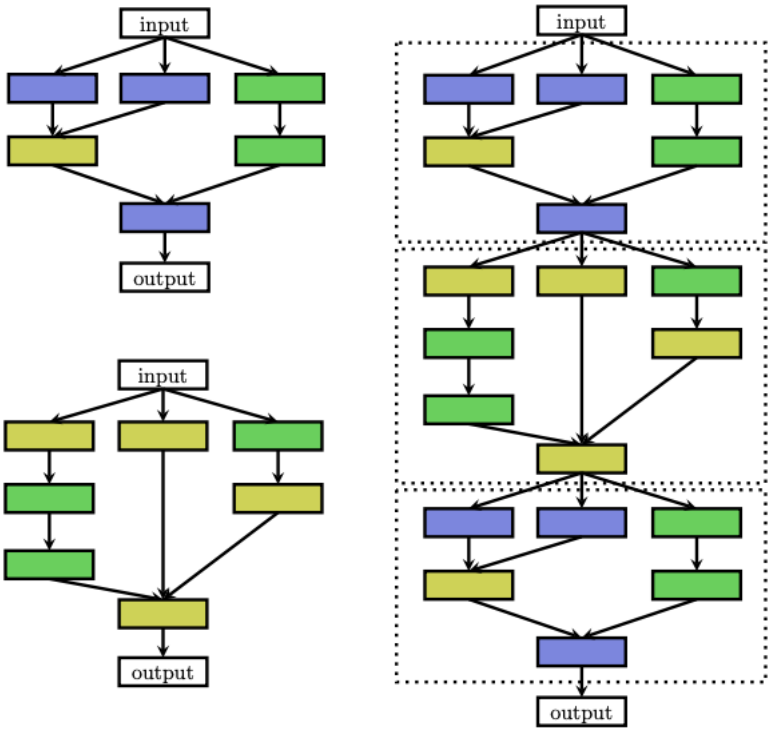


Figure 2.10: On the left, there are two types of cells: Normal cells (top) and reduction cells (bottom). On the right is an architecture built by stacking the cells sequentially. Figure reproduced from Elsken et al. [11].

- Zoph et al. [76] also demonstrated strong proof of a useful design pattern of repeatedly stacking modules in architectural engineering. For example, we can build strong models by stacking residual blocks on CNN or multi-headed attention blocks on Transformers.

Hierarchical Structure

The search space can also be represented as a two-level hierarchical structure: Cell level or the internal, which selects the operation and connection for each node in the cell, and network level or the external, which controls resolution changes space.

In some cell-based methods, it is possible to observe this hierarchical influence [21]. However, this approach focuses only on the cellular level and ignores the network level, which can cause, for example, the reduction of feature maps by stacking normal cells and then adding a reduction cell.

Liu et al. [34] published one of the first works using the hierarchical structure at the network level. Initially, there is a set of primitives (individual operations such as convolution, pooling, identity), which form small sub-graphs, also called "motifs", used recursively to form higher-level computation graphics [63] (Figure 2.11). Some advantages

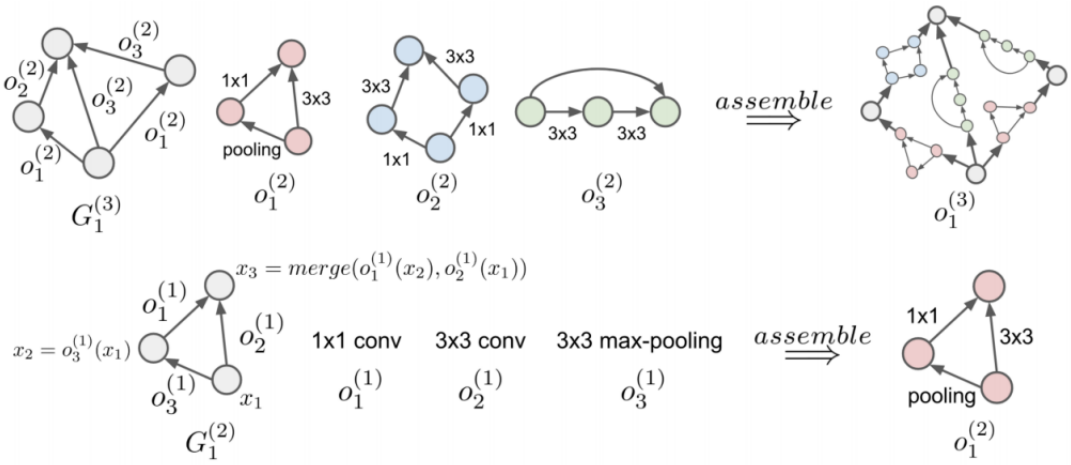


Figure 2.11: The representations at the bottom of the image show primitive operations $o_1^{(1)}$, $o_2(1)$, $o_3(1)$ forming a level-2 motif $o_1^{(2)}$. The representations at the top of the image represent the junction of level-2 motif $o_1^{(2)}$, $o_2(2)$, $o_3(2)$ forming level-3 motifs $o_1^{(3)}$. Figure reproduced from Liu et al. [34].

of using the technique are:

- These architectures are coded as a series of adjacency matrices, that is, matrices composed of 0 to the left and right along a diagonal. When performing the random filling of the right side of these adjacency matrices with the different operations available at each level, it is possible to notice a good operation even when compared to the more sophisticated research tested in the article.
- Another point is the dispersion of the resulting architectures. This dispersion is very desirable, for example, for counting parameters.

2.3.2 Search Strategy

After we specify a search space, we need to define the search strategy for identifying the best-performing architectures with the performance estimation strategies.

Reinforcement Learning

The pioneering publications on NAS, such as Zoph and Le [75]'s project, employed Reinforcement Learning (RL) to search for suitable networks.

This technique comprises an agent that usually consists of a Recurrent Neural Network (RNN) that acts as a controller and is responsible for generating the different proposals for convolutional neural network architectures. At each step t, a new architecture is tried, returning from the environment (training and evaluation of the generated network) observation of the state S_t and a reward scale R_t that will be used to update the agent's sampling strategy (Figure 2.12). In this way, the reward function is maximized, and it is expected that with the update of this parameter, the controller improves its decision-making in constructing new architectures.

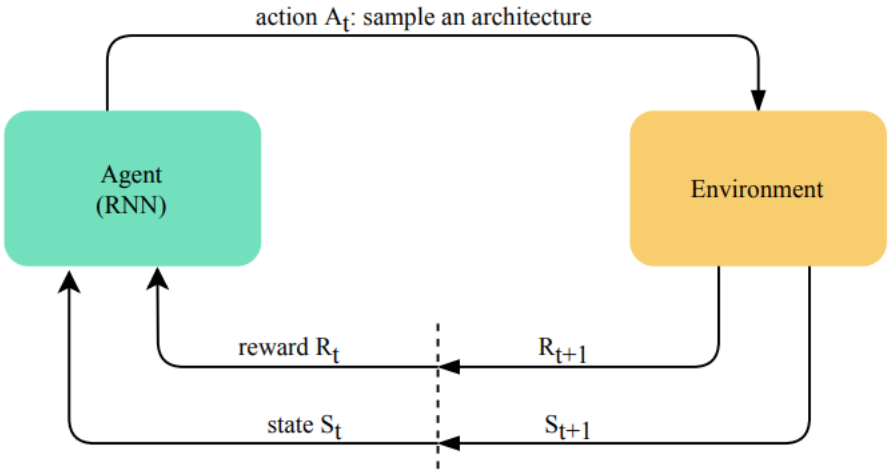


Figure 2.12: A high-level overview of NAS using reinforcement learning. Figure reproduced from Kayoing et al. [21].

A disadvantage of using reinforcement learning algorithms in NAS is the vast computational resources required to achieve satisfactory results. In experiments reported by Zoph and Le [75], 800 GPUs were used for 28 days for a neural network architecture search in the CIFAR-10 dataset [26], which consists of 60,000 32×32 images in 10 classes, with 6,000 images per class.

Evolutionary Algorithms

This strategy allows the creation of search algorithms based on a population meta-heuristic inspired by the natural evolutionary process to solve optimization problems. We can define its operation based on a set of steps: initialization, selection, crossover, mutation, update, and termination (Figure 2.13).

Initialization defines the generation of the first population, and then the other steps are performed repeatedly until the stopping criterion. The looped steps are [21]:

- **Selection:** This step consists of choosing a part of the generated networks to be used in the crossover process, keeping the neural architectures of good performance while eliminating those with low quality (or fitness, according to the terminology of evolutionary algorithms).
- **Crossover:** The networks chosen in the selection phase are grouped in pairs that generate a new network composed of half of the genetic information of each one of the original networks.
- **Mutation:** After copying the genetic information from the parent network, the mutation occurs. A small part of the networks generated in the crossover is modified so that they no longer perfectly reflect subsets of the genes of the origin networks.
- **Update:** After completing the previous steps, we have many new networks generated, and due to the limitations of computational resources, some of them will need to be removed in this step.

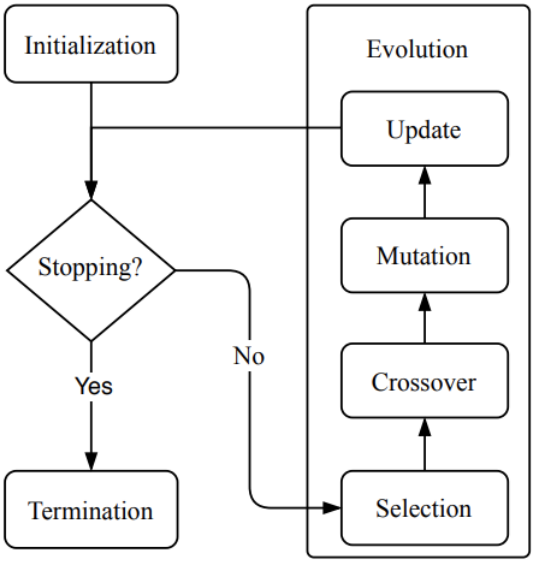


Figure 2.13: Operating steps of an evolutionary algorithm. Figure reproduced from Kayoing et al. [21].

Despite being a solution with computational cost and execution time lower than the experiments that use RL, many of the researches based on evolutionary algorithms ignore the budget limitations of the GPU, proposing solutions with long executions or the need for high computational power.

Gradient Descent

Unlike the aforementioned strategies that require a discrete search space, we need to make the search space differential to use the descending gradient as a search strategy. Differentiable Architecture Search (DARTS) [35] algorithm, one of the pioneers in the use of this strategy in NAS methods, transformed discrete search spaces into continuous and differential search spaces through space relaxation using a softmax function.

This strategy combines learning architecture parameters and network weights in a single model. We define this approach where a single neural network is trained as a one-shot architecture search during the search process[10].

Comparing the cost and size of the search space (Table 2.1) for each of the mentioned search algorithms, we observe that the larger the search space, the higher the corresponding search cost.

2.3.3 Performance Estimation Strategy

As the search algorithms aim to identify an architecture that maximizes some performance measures, it is necessary to define a strategy to estimate the performance of a child architecture, generating feedback to optimize the search algorithm. The main criterion

[10]It is also referred to by the term Differentiable Search.

Table 2.1: Comparison between search strategies.

Search Strategy	**Computation Cost**	**Search Space**
Reinforcement Learning	High	Large
Evolutionary Algorithms	High	Large
Gradient Descent	Low	Restrict

evaluated is the accuracy in a validation set, but recent works have also adopted other criteria in evaluating a model, such as size and latency.

Training from Scratch

The most obvious approach is to train each candidate child network from scratch on a training dataset and then estimate its accuracy on a validation set. However, this task is costly concerning the consumption of computational resources. Therefore, several new strategies have been proposed to accelerate this process, such as proxy task performance and parameter sharing.

Proxy Task Performance

This approach is faster and has a lower cost. We can summarize its operation in the following steps [63]:

- To train on a smaller dataset, often called a proxy data set.
- To train for fewer epochs, thus reducing the training time.
- To train and evaluate a down-scaled model in the research stage.
- To predict the learning curve.

Despite being faster and reducing computational cost, this strategy can introduce biases in the estimate because performance will usually be noisier.

Parameter Sharing

This technique proposes reusing parameters between the candidate networks, accelerating the process since training from scratch is unnecessary. Some applied solutions are the sharing of parameters between the child networks [44] based on morphism, where we inherit the weights of previous architectures [62].

All the possibilities presented to be adopted for each step of the neural architecture search process impact the performance of the generating network, determining the time and computational power needed to generate the candidate networks. The existing NAS in the literature defines their strategies for each stage based on their purpose, whether they are faster executions or with a broader range of candidate networks and better results in terms of accuracy.

Chapter 3

Literature Review

In 2017, Neural Architecture Search (NAS) became a hot research topic in machine learning after Zoph and Le achieved competitive performance on the CIFAR-10 [11] using reinforcement learning for generating architectures [75]. They used 800 GPUs for 3-4 weeks to achieve the results. It was the first significant step for this field of research, instigating a series of publications concerned with reducing computational costs and increasing performance after perceptions about the need to develop optimized methods for faster and more economical executions.

In this chapter, we review the NAS networks proposed in the literature for the object detection task. We describe how we selected the works (Section 3.1), and we explain each of the networks identified and analyzed in developing this Master's thesis. We grouped the NAS models according to the components of the detectors that the search is proposed for: backbone (Section 3.2), neck (Section 3.3), head (Section 3.4), and region proposal network (Section 3.5).

3.1 Selection of NAS-related Works

Although this literature review does not intend to be a formal meta-analysis, we took inspiration from Preferred Reporting Items for Systematic Reviews and Meta-Analysis (PRISMA) [42] to gather a representative sample of existing art.

We selected the NAS-related works using the *Semantic Scholar*[1] with the query "neural architecture search and object detection -video -textual", which reported 3470 articles. Taking into account that the beginning of research on NAS took place in 2017, after the publication of Zoph and Le [75], we defined the date range as 2017 to 2022, resulting in 2580 articles. We selected only conference papers and journal articles (resulting in 1740 articles) and excluded those published only on Arxiv; we got 198 articles.

From an initial analysis of the articles based on their titles and abstracts, 17 articles remained. After a more profound analysis from a brief reading of the articles, we ended up with 10 articles.

In recent years, research on the use of NAS for detection tasks has focused on the automated search of one or more components of an object detector. Therefore, we chose

[1] `https://www.semanticscholar.org`

to group the works based on the components of the architecture (Figure 3.1): backbone, neck, head, and region proposal network.

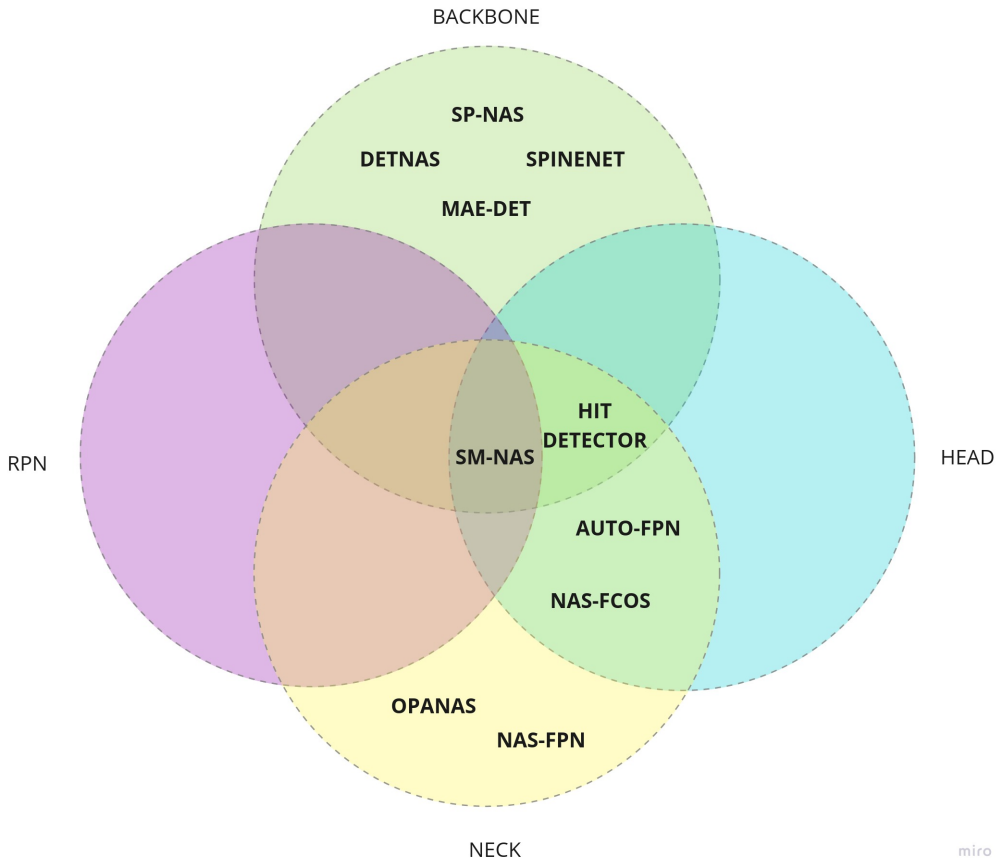


Figure 3.1: Venn diagram of latest NAS strategies for object detection with their main components.

3.2 Backbone

Object detectors rely heavily on the features extracted by the backbone. Many detectors use neural networks designed for image classification tasks as a backbone [29].

The design of this component for object detection must be different from the one designed for classification. In the object detection task, it is necessary to locate and classify different objects at different scales. The classification task needs only a label for the principal element of the image. Therefore, designing a backbone for detection is more challenging and requires more effort.

Despite the success of handcrafted backbones for detection, they are usually designed for datasets consolidated in this task, such as MS-COCO [33] and PASCAL VOC [12], which does not guarantee an adaptation to other datasets, as opposed to those built by a NAS network that relies on data to define the architecture, generating its solutions for your dataset.

Another challenge in constructing a detector for a NAS network is related to the pre-training stage of the backbones in the ImageNet dataset [7], which causes difficulties in the optimization process. Accuracy in the target task indicates a reward for the performance of a candidate network. However, the accuracy obtained during a pre-training step cannot be qualified as a requirement in this situation. The pre-training is only related to the identification of labels, in the case of the object classification step.

The backbone search starts in this stage of pre-training the candidate networks, which will adjust their performance as it undergoes training on the detection dataset, making the process very expensive. Even choosing to perform the training from scratch only on the detection set, this process requires more iterations to compensate for the pre-training step and ends up not presenting a computational gain.

In 2019, the first NAS-based approach was proposed for designing the backbone for object detection tasks. The network entitled **DetNAS** [4] is based on the one-shot super-network technique that contains all possible candidate networks in a search space based on the ShuffleNetv2 [38] block, a convolution architecture that involves the operation of channel splitting and shuffling.

DetNAS has a search process divided into three stages (Figure 3.2):

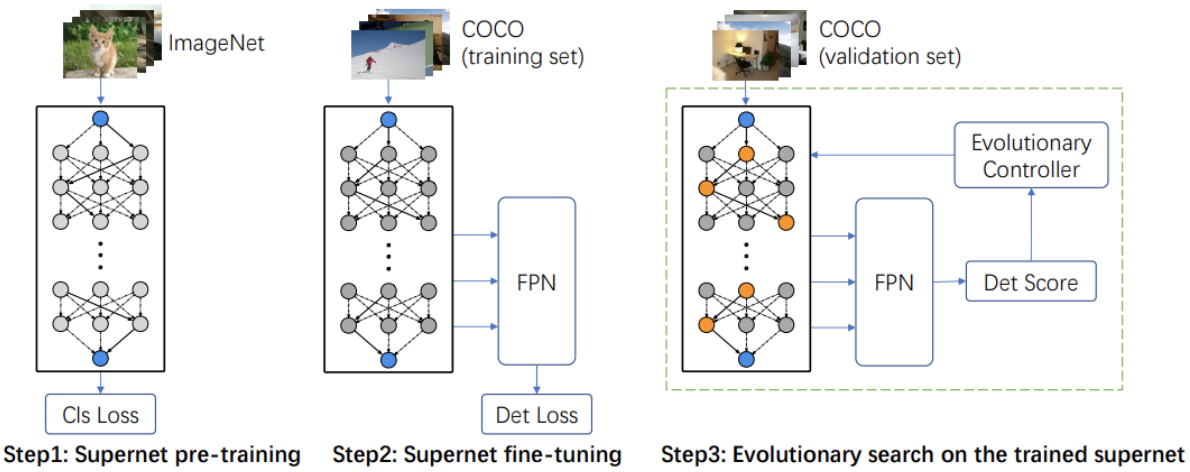


Figure 3.2: Pipeline of the DetNAS. Figure reproduced from Chen et al. [4].

- **Step 1 – Pre-training the one-shot supernet in ImageNet:** Different one-shot methods where the search space is relaxed, transforming from discrete to continuous and making the net weights deeply coupled candidates. In DetNAS, a technique was adopted to guarantee that in the pre-training of the supernet, it is possible to reflect the relative performance of the candidate networks through the adoption of a single possible path for feedforward and backward propagation. So no gradient or weight updates act on other paths or nodes in the supernet graph.
- **Step 2 – Fine-tune the one-shot super grid on detection datasets:** In this step, conventional batch normalization was replaced by synchronized batch normalization (SyncBN) during super grid training. In addition to performing batch statistics calculations on multiple GPUs, it increases the batch size. It solves the

problem of small batches resulting from pre-training on ImageNet, as classification instead of object detection does not use high-resolution images.

- **Step 3 – Architecture search in the supernet with an evolutionary algorithm:** Using the evolutionary algorithm search strategy, different candidate networks are generated according to the supernet path. The problem is that the batch statistics on one path must be independent of the others. Therefore, the batch statistics for each child network are recalculated before each evaluation on a small subset of data extracted from the training data.

After the publication of DetNAS, several failures and improvements were identified. For example, the proposal to incorporate pre-training more economically by dividing the weight and architecture optimization process into two stages ended up creating a large gap between the performance obtained in the validation of the child networks and the result of the complete training of the network, not reflecting the actual performance of the architectures. Another point is the pre-defined and fixed super-network, which considerably limits the backbone search space, providing only the kernel size change and ignoring the adjustment of other factors such as the number of blocks, downsampling rate, and channel size.

Other NAS networks to search backbones for object detection emerged later to overcome the issues identified in DetNAS, such as Serial-to-Parallel NAS (**SP-NAS**) [22], a framework with a serial-parallel architecture (Figure 3.3).

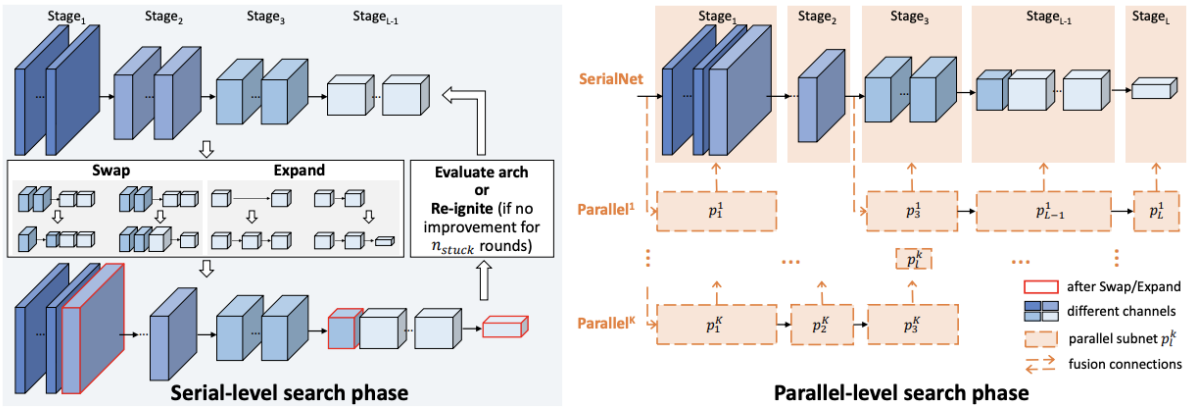


Figure 3.3: Overview of the SP-NAS. Figure reproduced from Jiang et al. [22].

During the serial search, the "swap-expand-reignite" strategy is adopted to avoid repetitive training of candidate networks, maintaining the effect of "ImageNet pre-training". At each research stage, network modifications are applied, including "swap" and "expand" the network.

For the "expansion" operation, a new block is added between two blocks but keeps the same number of input/output channels, where the convolution weight in the new block is always initialized as an identity matrix. In the "change" operation, only the pass of the neighboring block is changed, keeping the weight unchanged. Both modifications to the current network will keep the output unchanged as much as possible on boot, which retains the "ImageNet pre-training" effect.

The algorithm can crash without additional performance improvements when the transformed architecture deviates too far from the original ImageNet's pre-trained backbone. Thus, a "reignite" strategy is used, where training takes place on ImageNet for the locked architecture, "reigniting" the search process, requiring only 1 or 2 pre-training rounds on ImageNet.

After the serial search, during the parallel level search, the SerialNet (network obtained in the serial level search) is first used as the base backbone, and an RPN and an R-CNN head are added to the backbone, thus building a two-stage detector.

Another network launched to identify the best backbone for object detection was **SpineNet** [10]. It is based on encoder-decoder architectures, such as FPN [31] and DeepLabv3 [2], which are commonly used for the detection task.

The encoder, commonly referred to as a backbone, is a scaled-down network, and the decoder is a network applied to the backbone to retrieve spatial information. This type of architecture uses scaled-down backbones and loses much information since most data in the detection task have high resolution.

Therefore, to avoid the loss of spatial information, SpineNet consists of a meta-architecture called a scale-permuted model that allows for two significant improvements in the backbone architecture design. First, the spatial resolution of intermediate feature maps must increase or decrease at any time so that the model can retain spatial information as it goes deep. Second, the connections between feature maps must traverse scales, making it easy to merge those features at different scales. Thus, neural search architecture based on the reinforcement learning strategy uses a new search space design that includes these features to discover a permuted model effectively.

A NAS method was recently proposed for searching backbones, called MAximum-Entropy DETection (**MAE-DET**). Based on the Maximum Entropy Principle, a detection network is formulated as an information processing system. Its capacity is maximized when its entropy reaches the maximum under the given inference budgets leading to a better feature extractor for object detection. MAE-DET maximizes the differential entropy of sensing backbones, looking for the optimal configuration of network depth and width without training network parameters, thus reducing the cost of architecture design to almost zero.

3.3 Neck

Due to the massive number of parameters from the backbone, many researchers have focused on other components, such as the neck or pyramid features. This component helps the detector to better locate by identifying objects in different locations and size scales.

Pyramid features represent an image with different scales. They require multi-scale processing and convolutional networks to generate the pyramid representations but demand a large computation load. To solve this problem, cross-scale connections in CNNs were introduced to connect layers of features at different scales, improving the representations of these features and not only making them semantically strong, but also containing high-resolution information.

One of the challenges when designing a feature pyramid architecture is the high number of possible connections to combine different scales, exponentially increasing the number of layers in the network.

Neural Architecture Search Feature Pyramid Network (**NAS-FPN**) [14] chose to be based on the RetinaNet [32] framework, whose structure is composed of two main components: a backbone network (usually a next-generation image classification network) and a feature pyramid network (FPN [31]) to create a NAS capable of identifying a better FPN architecture for the RetinaNet backbone.

To discover the best FPN, a solution based on Zoph et al. [75] uses a controller to select the best model architectures in a given search space using the reinforcement learning strategy.

A significant advantage of employing NAS-FPN is the use of stacked pyramid networks, i.e., feature pyramid representations can be obtained at the output of any given pyramid network, enabling detection at any time and generating early-out detection results.

Classifiers and regression box heads can be attached after all intermediate pyramid networks have been trained with in-depth supervision. During inference, the model does not have to finish the forward pass to all pyramid networks and instead lands on the output from any pyramidal network and generates detection results. Thinking of hypotheses where theory or latency resource is a concern is one way of dynamically deciding how much sector resource should be allocated to generate detections.

One-Shot Path Aggregation Network Architecture Search (**OPANAS**) [30] introduced the idea of increasing search efficiency and detection accuracy by identifying a better architecture for the object detector neck.

Initially, it was carefully designed with four parameterized information paths (top to bottom, bottom to top, scale equalization, and merge division) and two parameterless paths (ignore-connect and none) to build the search space (Figure 3.4). These six modules introduce different information flows, different connections between the backbone and the detection head and lead to complementary and highly interpretable aggregation modules.

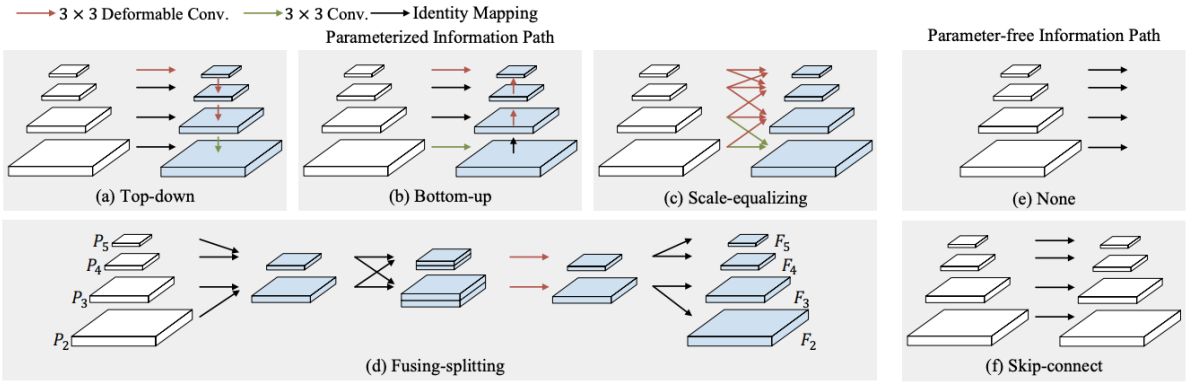


Figure 3.4: The six heterogeneous information paths proposed by OPANAS. Figure reproduced from Liang et al. [30].

OPANAS uses a new FPN search space, in which each candidate FPN is represented by a densely connected acyclic directed graph (each node is a pyramid of features, and

each edge is one of six information paths). In addition to an efficient single search method to find the optimal path aggregation architecture, the supernet is first trained to identify the ideal candidate through an evolutionary algorithm.

To solve the problem of weak adaptability (searched architectures for particular detectors) present in other FPN networks based on Neural Architecture Search. The functioning of OPANAS as a plug-and-play module was proposed, where the searched architecture can be easily adapted to main flow detectors, including, for example, RetinaNet [32] and Faster R-CNN [49].

3.4 Head

In the literature, we could not find a NAS approach specifically designed for the head of an object detector. Instead, some NAS approaches include searching for the head structure together with other components: For example, the Auto-FPN [66] and the NAS-FCOS [61], which proposes the search for both head and neck, and the Hit-Detector [18], which, in addition to the two components, also seeks a better backbone.

As mentioned before, **Auto-FPN** [66] proposes two automatic search modules for detection: auto-fusion, which automatically searches for a better fusion of resources at various levels (neck), and auto-head, which searches for a better structure to perform the classification and bounding box regression.

The auto-fusion module designed a fully connected search space with several expanded convolution operations between the resource levels to allow a flexible fusion of resources with different receptive fields. To make possible the usage of the network, a joint computational restriction was added that handles the size of the FLOPs and MAC parameters to regularize the search.

Afterward, Neural Architecture Search Fully Convolutional One-Stage (**NAS-FCOS**) [61] was proposed with the same objectives as Auto-FPN, but to identify components for one-stage detection architectures, i.e., an anchor-free one-stage detection structure. The main empirical demonstration provided in this research is that the FCOS detector performance of an anchorless stage can be enhanced with the FPN obtained by an automatic discovery.

The NAS-FCOS decoder consists of two subnets, an FPN and a set of prediction heads with shared structures. Unlike other FPN-based one-stage detectors, the heads have partially shared weights, and the number of layers to be shared is automatically decided by the search algorithm with a search strategy based on reinforcement learning.

Hit-Detector [18] presented a more ambitious proposal by combining the search for the backbone, neck, and head based on the hypothesis that exploring only one part of the detector at a time cannot fulfill the potential of each component, in addition to asserting that the backbone searched separately from the neck could result in a mismatch between the components.

In their experiments, Guo et al. [18] showed that different detector parts are sensitive to different operations. Thus, for each component, a specific sub-demand space is needed. For example, the backbone and head prefer operations with a higher expansion rate, as

an inverted residual block with more intermediate channels to increase feature expression. The neck prefers a higher dilation rate for larger receptive fields.

Furthermore, the backbone and head can increase performance better than the neck based on the assumption that object detection emphasizes more on the perception of the location of each object in an image. Therefore, the backbone designed for detection may have performed better than that designed for classification. Moreover, the head aims to identify and refine the location of the bounding boxes, so searching for more adequate convolution layers in the head can bring more benefits to the detection task.

The simultaneous end-to-end search introduced by Hit-Detector is performed hierarchically, consisting of two main steps (Figure 3.5):

- A custom and suitable sub-search space for each component is filtered from a large search space containing all candidate networks for the operation.
- In the sub-search spaces, a differential search of the corresponding component is performed.

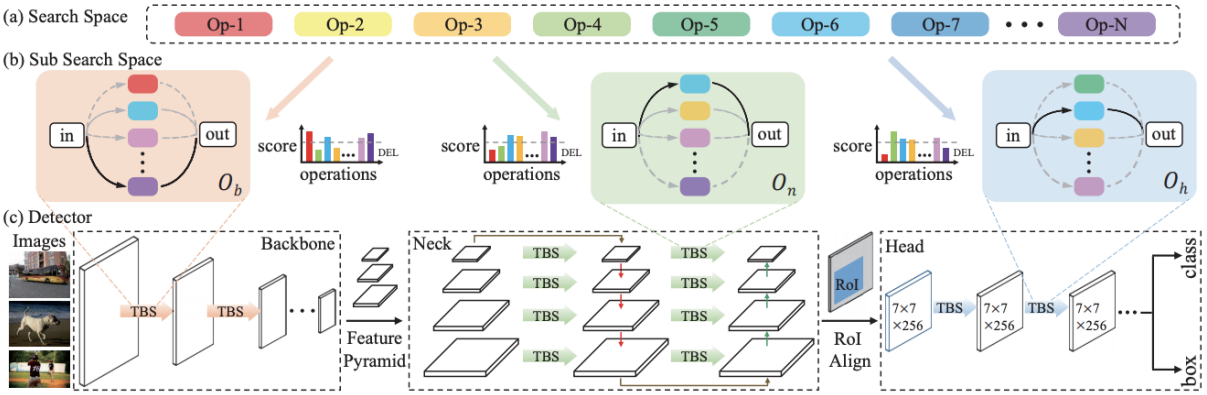


Figure 3.5: Overview of Hit-Detector architecture search framework. Figure reproduced from Guo et al. [18].

3.5 Region Proposal Network

The Region Proposal Network (RPN) is commonly used for generating proposals in a two-stage detector, consisting only of a convolution layer followed by two fully-connected layers that yield region proposal classification and bounding box regression. RPN has not been largely investigated because it is already a typically light and efficient element.

Structural-to-Modular Neural Architecture Search (**SM-NAS**) [68] is the only NAS network for object detection that still handles the use of RPN. Like Hit-Detector, it proposes the complete search of a detector's architecture. By introducing the automated search for both one-stage and two-stage detectors, it adapts the search space to be: without RPN (one-stage detectors) or with RPN (with guided anchoring).

Proposing the ideal combination between the modules through the use of evolutionary algorithms and the partial order pruning technique for a fast and parallel search, SM-NAS brought two important considerations:

- The one-stage detector is not always faster than the two-stage detector.
- A suitable combination of modules and input resolution can lead to an efficient detection system.

Chapter 4

Methodology

One of the main objectives of this Master's thesis is to analyze the viability of applying NAS networks for the object detection task in the context of low computational power (less than 8 GPUS) and a limited dataset (less than 10,000 images). In this chapter, we detail the methodology explored for analyzing the studied NAS networks, describing all the relevant steps in the process to facilitate this research's reproducibility. We also present the chosen datasets and the procedure adopted to select the NAS strategies used in the experiments.

4.1 Pipeline

Figure 4.1 illustrates the proposed pipeline, composed of four steps: 1) Dataset selection, 2) NAS selection, 3) NAS execution, and 4) Analysis of results. It is designed to ensure a better understanding of the steps in this Master's thesis, facilitating the reproduction of the work.

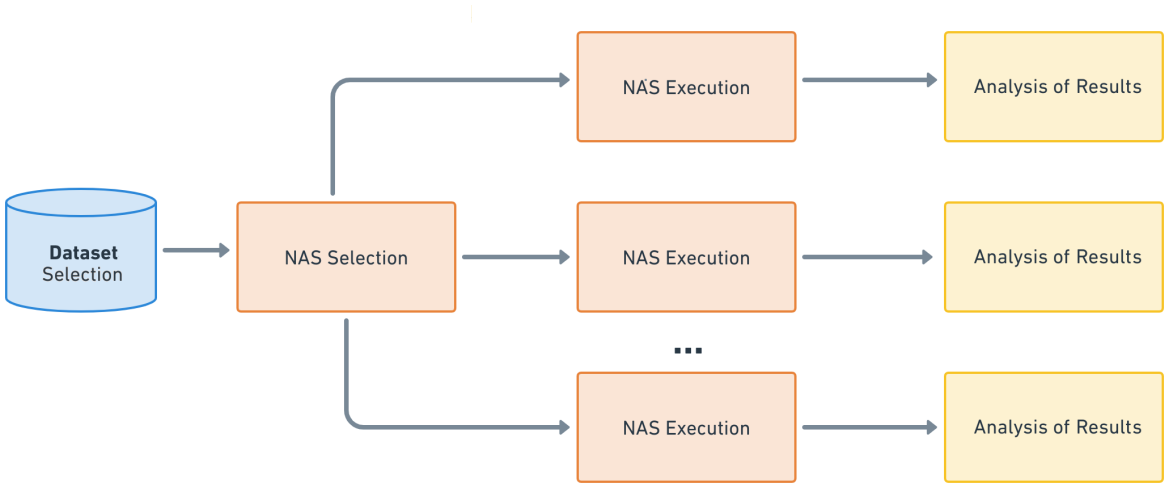


Figure 4.1: Our proposed pipeline.

1. **Dataset Selection:** To select a relatively small object detection dataset (less than 10,000 images), the data must be annotated in an annotation format required by the

network (e.g., PASCAL VOC, MS-COCO) and, preferably, distributed in training, validation, and test sets.

2. **NAS Selection:** To choose the NAS networks to be used in the experiments, criteria were defined to determine the possibility of executing the network within our structural conditions, such as the availability of the generator network code and the number of GPUs required for execution.

3. **NAS Execution:** After selecting the networks, we first replicate the original experiments (with the data and configurations reported in the article) to validate the implementation and the consistency of the authors' results. Next, we execute each of the selected NAS strategies considering the datasets selected in this Master's thesis.

4. **Analysis of Results:** At the end of the executions, we analyze the results and identify possible adjustments (e.g., hyperparameter optimization) to improve the performance. The object detection performance is evaluated using the standard metric, the mean Average Precision (mAP). Precision and recall can also be used to report in more detail the performance.

4.2 Datasets

4.2.1 Insulator Dataset

As previously mentioned, this Master's thesis was partially supported by *Fundação Para Inovações Tecnológicas*[1] (FITec), and its target application was the detection of electrical and network insulators classification according to their material, as well as the identification and classification of faults (e.g., polluted, rusted, or broken insulators).

To carry out the training and validation steps of the proposed solutions, FITec assured to provide 20 datasets containing images of the area of insulators of São Luís, Maranhão (MA), distributed by insulator material and types of faults (Table 4.1), captured by drones and annotated by experts.

However, only two (out of 20) datasets were provided (Table 4.2) containing images from the field captured by a drone camera and annotated only with the location of the insulators. The first dataset consists of 1538 images of 4000×3000 pixels, while the second consists of 760 images of 1100×820 pixels. Figure 4.2 illustrates a few samples from the Insulator dataset.

Therefore, the dataset used in the experiments contained 2298 images. In the first experiments, we used the first dataset as a training set (approximately 67% of the total images) and the second as a validation and test set, with 380 images each (approximately 33% of the total images).

Due to the lack of annotated data with all the information, the experiments performed on this set of insulator images only covered the location of the objects since we only had the insulator class.

[1] `http://www.fitec.org.br`

Table 4.1: Dataset proposed by *Fundação Para Inovações Tecnológicas*.

Insulator type	**Fault type**
All types	Lightly polluted Very polluted Rusty F1/F2 Rusty F3 Rusty F4/F5 Broken – Power Line 69 Kv (only 1) Broken – Power Line 138 Kv (more than 1) Broken – Power Line 138 Kv (2 or less than 2) Broken – Power Line 138 Kv (more than 2) Broken – Power Line 230 Kv (more than 2)
Line Post	Leaky Very leaky
Polimeric	Leaky Very leaky
Glass	Leaky Very leaky

Table 4.2: Final dataset provided by *Fundação Para Inovações Tecnológicas*.

Dataset	**#Images**	**Resolution**	**Labels**
Dataset 1	1538	4000×3000 pixels	*Insulator*
Dataset 2	760	1100×820 pixels	*Insulator*

Figure 4.2: Examples from the Insulator dataset. Images from our dataset.

The remaining datasets were obtained from Roboflow company[2], which hosts free public computer vision datasets in many popular formats (including COCO JSON, PASCAL VOC XML, and Tensorflow TFRecords). Launched in January 2020, Roboflow can be used to: annotate images or upload existing annotations, convert existing VOC XML annotations to COCO JSON annotations, obtain and share public datasets, preprocess images, and train models. We selected three datasets from Roboflow (BCCD, Aquarium, and Oxford Pets).

[2]`https://roboflow.com`

4.2.2 BCCD Dataset

The Blood Cell Count and Detection (BCCD) dataset consists of blood cell photos and is open-sourced on Github [52]. It contains 364 images of 640×640 pixels, totaling 4888 labels distributed in three classes: WBC (white blood cells, 372 occurrences), RBC (red blood cells, 4155 occurrences), and platelets (361 occurrences). The dataset distribution is described in Table 4.3.

Figure 4.3 exhibits some samples from the BCCD dataset.

Table 4.3: BCCD dataset distribution [52].

Dataset	#Images	Percentage
Training	255	70%
Validation	73	20%
Test	36	10%

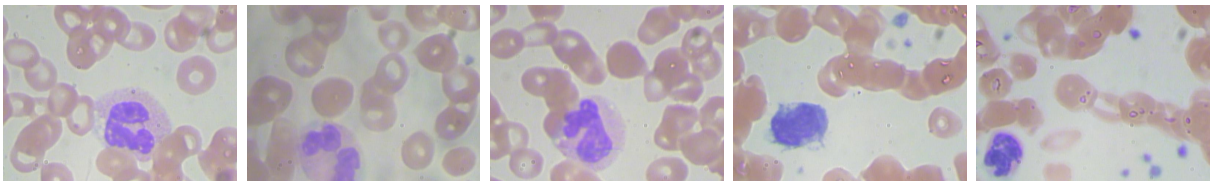


Figure 4.3: Examples from the BCCD dataset [52].

4.2.3 Aquarium Dataset

The Aquarium dataset[3] consists of 638 images of 1024×1024 pixels collected by Roboflow from two aquariums in the United States: the Henry Doorly Zoo in Omaha (October 16, 2020) and the National Aquarium in Baltimore (November 14, 2020). The images have labels for object detection distributed in 7 categories: fish (2669), jellyfish (694), penguin (516), shark (354), puffin (284), stingray (184), and starfish (116 examples). Figure 4.4 displays a few samples from the Aquarium dataset.

The dataset distribution is described in Table 4.4.

Table 4.4: Aquarium dataset distribution (by Roboflow).

Dataset	#Images	Percentage
Training	448	70%
Validation	127	20%
Test	63	10%

[3] `https://public.roboflow.com/object-detection/aquarium`

Figure 4.4: Sample from the Aquarium dataset (by Roboflow).

4.2.4 Oxford Pets Dataset

The Oxford Pets dataset[4] contains 3680 images distributed in 2 classes (cats and dogs). This dataset contains the object detection portion of the original dataset with bounding boxes around the animals' heads. The data provided by Roboflow is a part of the Oxford-IIIT Pet Dataset created by the Visual Geometry Group in Oxford [43] that originally contains 37 categories (based on race) with approximately 200 images for each class (cat and dog). Figure 4.2.4 shows some examples from the Oxford Pets dataset.

The dataset distribution is described in Table 4.5.

Table 4.5: Oxford Pets dataset distribution (by Roboflow).

Dataset	**#Images**	**Percentage**
Training	2576	70%
Validation	736	20%
Test	368	10%

Figure 4.5: Sample from the Oxford Pets dataset (by Roboflow).

[4] `https://public.roboflow.com/object-detection/oxford-pets`

4.3 Neural Architecture Search

We selected 10 networks for the NAS experiments (Section 3.1): DetNAS [4], NAS-FPN [14], SM-NAS [68], NAS-FCOS [61], Hit-Detector [18], SP-NAS [22], SpineNet [10], OPANAS [30], and MAE-DET [57]. We analyzed the actual feasibility of executing all those networks, considering mainly the availability of the code and the number of GPUs needed to execute each.

Considering the code available, only four possibilities remained: DetNAS, NAS-FPN, SP-NAS, and SpineNet. Moreover, after discarding the models whose execution proved impracticable (or unfeasible) due to the high computational cost, we ended up with only two approaches: DetNAS and SP-NAS. Table 4.6 summarizes the networks.

Table 4.6: Selection of NAS networks to execute the experiments.

Network Conference	Conference **Year**	**Code**	**GPUs/TPUs**	**Dataset**	**AP (%)**
DetNAS [4]	NeurIPS'19	Available	8 GPUs for 1.5 days for supernet pre-training, 8 GPUs for 1.5 days for supernet fine-tuning, 20 GPUs for 1 day for search on the supernet (Small Search Space: GTX 1080Ti | Large Search Space: Tesla V100)	ImageNet in pre-training, MS-COCO and PASCAL VOC in other executions	COCO: 36.6 (FPN), 33.3 (RetinaNet), PASCAL VOC: 81.5 (FPN), 80.1 (RetinaNet)[a] |
NAS-FPN [14]	CVPR'19	Available	100 TPUs	MS-COCO	48.3
Auto-FPN [66]	ICCV'19	Not available	8 V100 GPUs	MS-COCO, PASCAL VOC, BDD	MS-COCO: 40.5, PASCAL VOC: 38.9, BDD: 39.2
SM-NAS [68]	AAAI'20	Not available	8 V100 GPUs to perform the search	MS-COCO	45.9
NAS-FCOS [61]	CVPR'20	The search network is not available; only the generated network	8 V100 GPUs for 4 days to perform the search and 4 V100 GPUs to perform complete training of the resulting network	PASCAL VOC for the search stage and MS-COCO for the final network training	46.1
Hit-Detector [18]	CVPR'20	The search network is not available, only the generated network	8 V100 GPUs	ImageNet in pre-training, MS-COCO in other executions	41.4
SP-NAS [22]	CVPR'20	Not officially available, alternative implementation in AutoML tool	4 nodes with 8 V100 GPUs each	MS-COCO, PASCAL VOC, BDD, ECP	MS-COCO: 49.1, PASCAL VOC: 84.1 (AP50), BDD: 38.7, ECP: 89.6 (mAP)
SpineNet [10]	CVPR'20	Available	100 TPUs to perform the search	MS-COCO	52.1
OPANAS [30]	CVPR'21	The search network is not available, only the generated network	1 V100 GPU for 4 days	MS-COCO	OPANAS + FPN: 41.6
MAE-DET [57]	ICML'22	The search network is not available, only the generated network	1 NVIDIA V100 GPU	MS-COCO	48.0

[a]The results are separated into FPN and RetinaNet because during the experiments the identified backbone was coupled to one of these two detection structures.

Chapter 5

Experiments

In this chapter, we present the experimental results obtained with YOLOv5, the state-of-the-art handcrafted network for object detection (Section 5.1), and with NAS-based networks (Section 5.2). We also detail the challenges and failed attempts to use NAS networks for object detection tasks.

We report the results as adopted in MS-COCO Detection Challenge, using the mean Average Precision (mAP). AP is the average over multiple IoU (the minimum IoU to consider a positive match). AP@[.5:.95] corresponds to the mean AP for IoU from 0.5 (coarse localization) to 0.95 (near-perfect localization) with a step size of 0.05. IoU (intersection over union) measures the overlap between two boundaries, i.e., it measures how much our predicted boundary overlaps with the ground truth (the real object boundary).

All experiments were conducted on a single NVIDIA Quadro RTX 8000 GPU, with 48 GB of GDDR6 (Graphics Double Data Rate 6) synchronous dynamic RAM.

5.1 Handcrafted Network: YOLOv5

To compare and verify the effectiveness of using NAS, we first evaluate the object detection performance of the handcrafted network in the selected datasets: Insulator, BCCD, Aquarium, and Oxford Pets. Therefore, we chose to execute the state-of-the-art object detection network, the YOLO [1, 47] model (Section 2.1.1).

YOLOv5 emerged a few months after YOLOv4 and has gained popularity by presenting good results and developing the evolutions of the YOLO network using the PyTorch framework as a basis. Due to the framework's popularity, the network has gained more users and contributors to implement improvements. The code of YOLOv5 is publicly available on GitHub [58].

Training and Validation

For all experiments, we used the YOLOv5 pretrained on MS-COCO [33] for object detection. Following the YOLOv5 training, we applied the same setup as suggested on the YOLOv5 repository[1], including a learning rate of 0.01, stochastic gradient descent with

[1] `https://github.com/ultralytics/yolov5/blob/ed887b5976d94dc61fa3f7e8e07170623dc7d6ee/data/hyps/hyp.scratch-low.yaml`

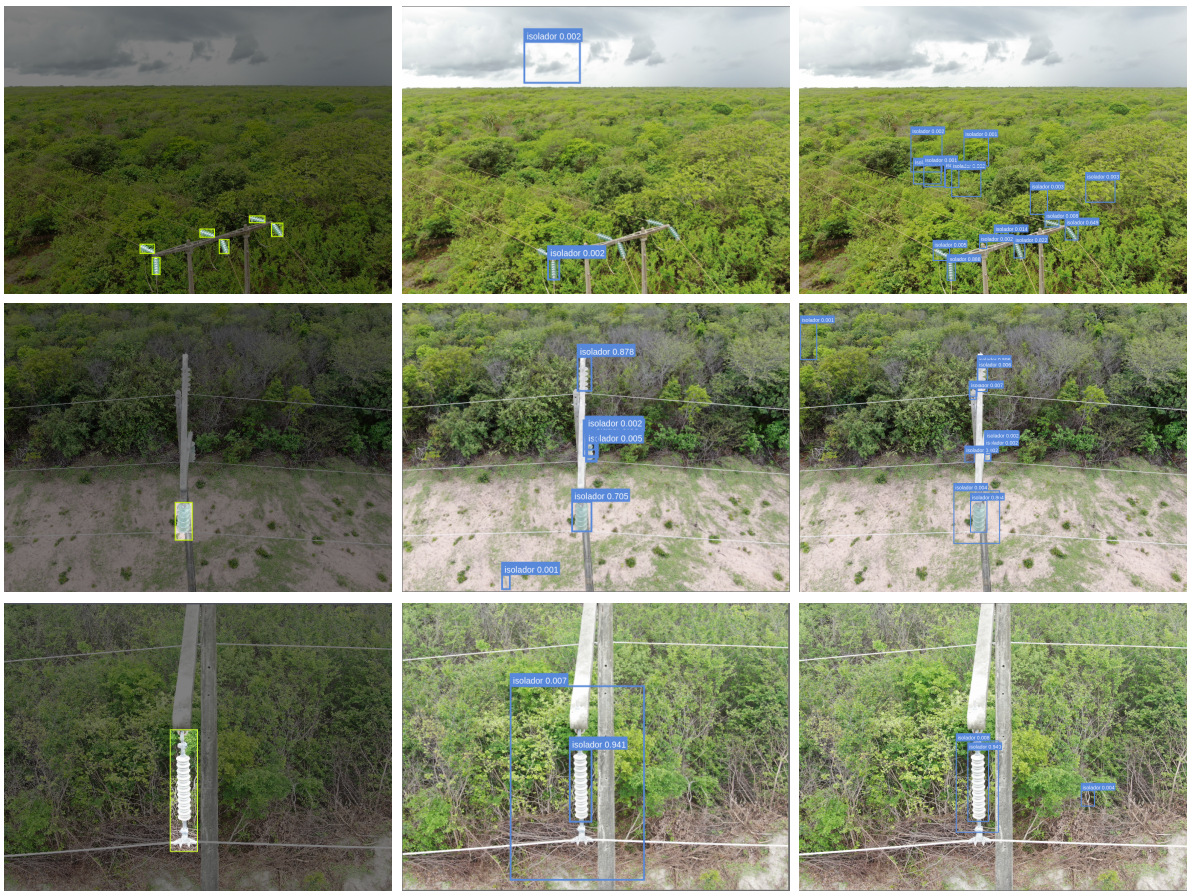
a momentum of 0.937, weight decay of 0.0005, batch size of 32, and 100 epochs.

For the experiments conducted on the Insulator dataset, due to GPU memory limitations, we first resized all images to 1080×1080 pixels and, then, to 640×640 (the YOLOv5 default). For the remaining datasets, we resized all images to 640×640 pixels (except for the BCCD dataset, which already contains images with these dimensions).

In Tables 5.1, 5.2, 5.3 and 5.4, we report the results on validation sets for each dataset. Figures 5.1, 5.2, 5.3, and 5.4 show samples of ground-truth and detection results. We observed, given the results, many false negatives, possibly generated by the erroneous understanding of the very repetitive backgrounds of the images. Also, several objects were detected for the same insulator (or other objects, in Figures 5.2, 5.3, and 5.4).

Table 5.1: Result of the YOLOv5 network on the Insulator dataset (validation set).

Precision	**Recall**	**mAP**	**mAP@.5:.95**	**Resolution**
0.807	0.837	0.838	0.366	640
0.817	0.790	0.785	0.353	1080

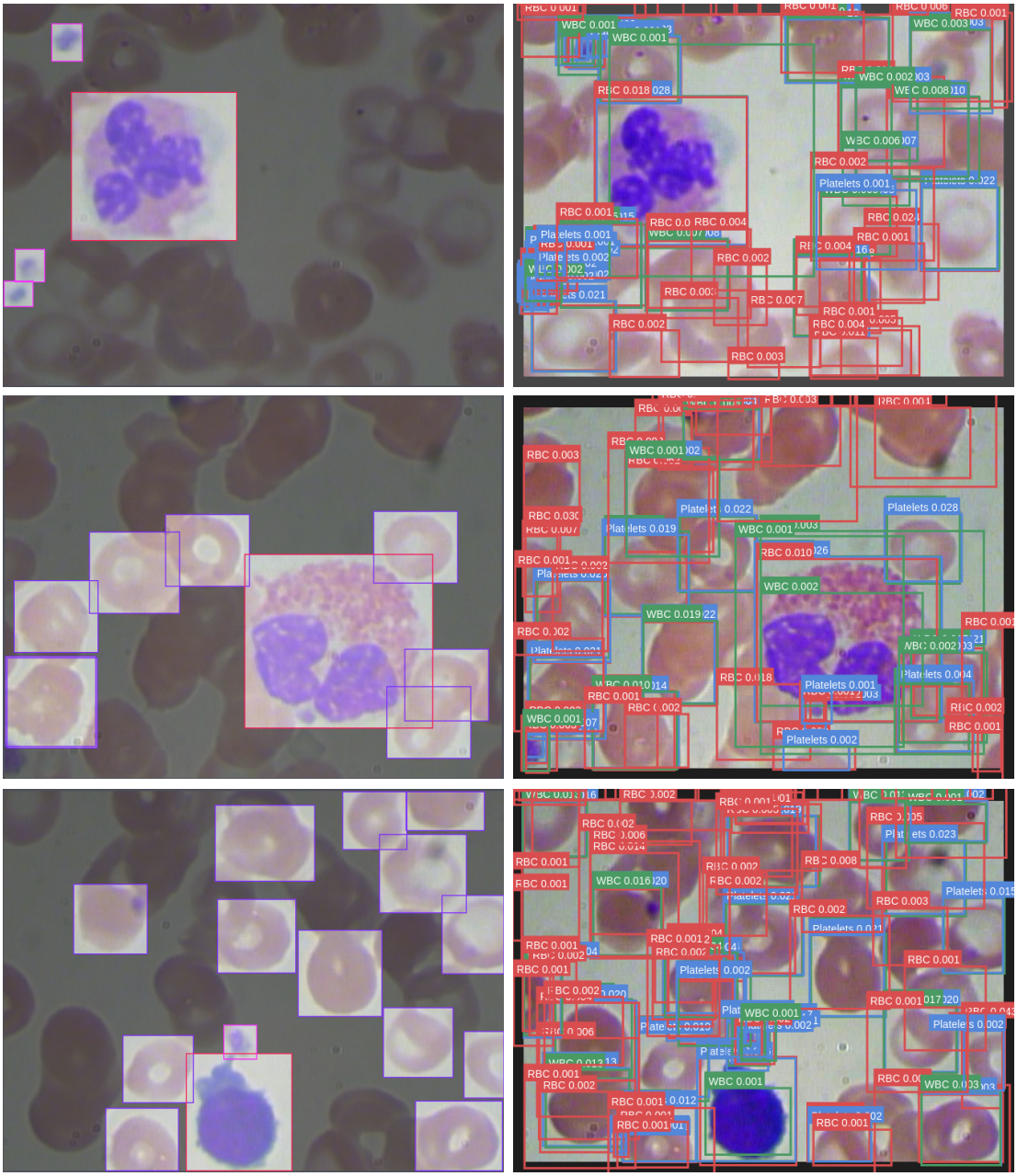


(a) Ground-truth (b) YOLOv5 (640 pixels) (c) YOLOv5 (1080 pixels)

Figure 5.1: Results of the YOLOv5 network on the Insulator dataset (validation set).

Table 5.2: Result of the YOLOv5 network on the BCCD dataset (validation set).

Class	Precision	Recall	mAP	mAP@.5:.95
All	0.846	0.858	0.884	0.584
Platelets	0.806	0.815	0.841	0.418
RBC	0.781	0.763	0.821	0.569
WBC	0.951	0.996	0.990	0.764



(a) Ground-truth (b) YOLOv5

Figure 5.2: Results of the YOLOv5 network on the BCCD dataset (validation set).

Table 5.3: Result of the YOLOv5 network on the Aquarium dataset (validation set).

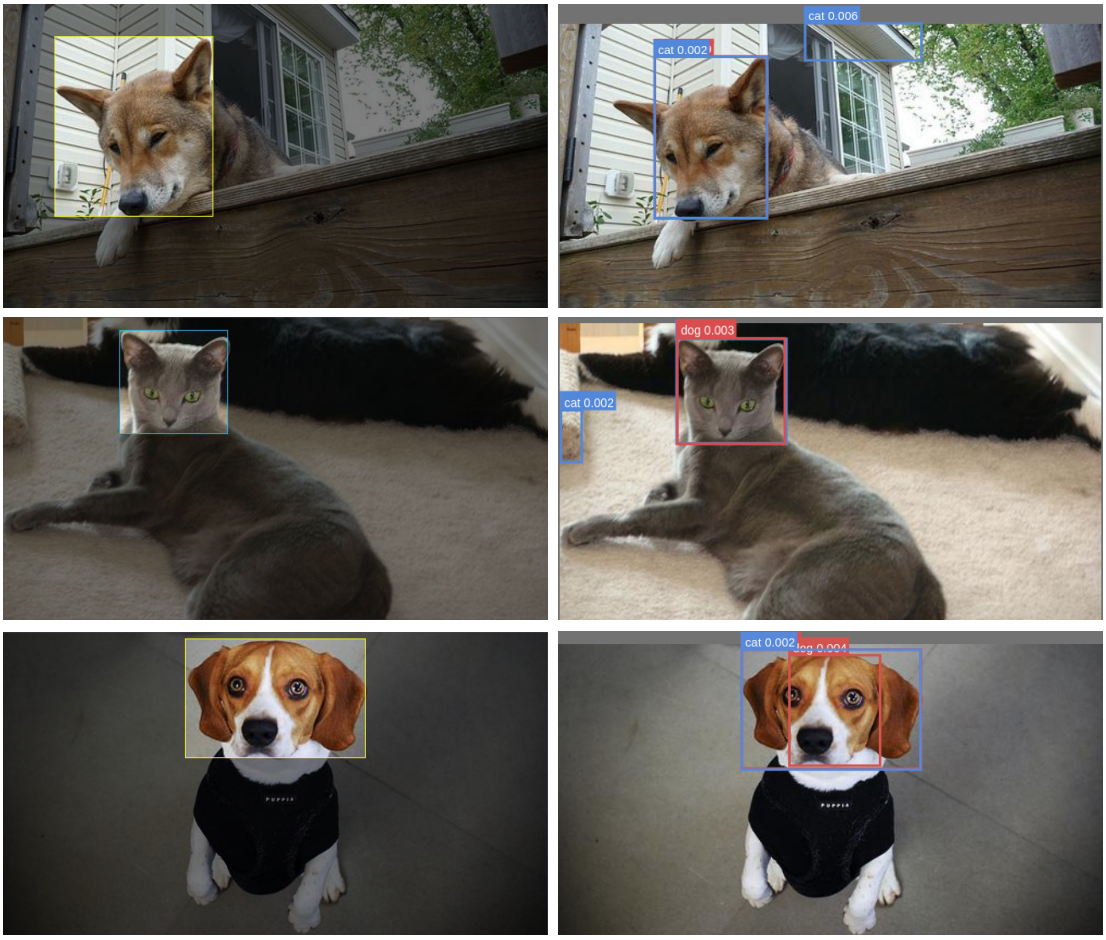
Class	Precision	Recall	mAP	mAP@.5:.95
All	0.735	0.615	0.652	0.378
Fish	0.742	0.588	0.642	0.335
Jellyfish	0.848	0.814	0.854	0.515
Penguin	0.705	0.642	0.654	0.284
Puffin	0.707	0.554	0.564	0.278
Shark	0.599	0.525	0.548	0.320
Starfish	0.824	0.602	0.654	0.474
Stingray	0.722	0.581	0.646	0.443

(a) Ground-truth (b) YOLOv5

Figure 5.3: Results of the YOLOv5 network on the Aquarium dataset (validation set).

Table 5.4: Result of the YOLOv5 network on the Oxford Pets dataset (validation set).

Class	Labels	Precision	Recall	mAP	mAP@.5:.95
All	2579	0.978	0.966	0.990	0.816
Cat	817	0.987	0.966	0.992	0.860
Dog	1762	0.968	0.967	0.988	0.772



(a) Ground-truth (b) YOLOv5

Figure 5.4: Results of the YOLOv5 network on the Oxford Pets dataset (validation set).

Test

For all tests, we assigned the best training result as weight.

In Tables 5.5, 5.6, 5.7, and 5.8, we present the results on test sets for each dataset using different IoU values.

Table 5.5: Result of the YOLOv5 network on the Insulator dataset (test set).

640×640 pixels				**1080×1080 pixels**			
IoU	**Precision**	**Recall**	**mAP**	**IoU**	**Precision**	**Recall**	**mAP**
0.4	0.898	0.933	0.949	0.4	0.803	0.901	0.895
0.5	0.896	0.928	0.949	0.5	0.802	0.901	0.895
0.6	0.907	0.911	0.950	0.6	0.783	0.887	0.881
0.7	0.879	0.878	0.934	0.7	0.737	0.767	0.791
0.8	0.772	0.792	0.855	0.8	0.540	0.705	0.576

Table 5.6: Result of the YOLOv5 network on the BCCD dataset (test set).

IoU	**Precision**	**Recall**	**mAP**
0.4	0.830	0.894	0.891
0.5	0.830	0.894	0.890
0.6	0.830	0.893	0.886
0.7	0.846	0.833	0.877
0.8	0.810	0.777	0.825

Table 5.7: Result of the YOLOv5 network on the Aquarium dataset (test set).

IoU	**Precision**	**Recall**	**mAP**
0.4	0.868	0.573	0.651
0.5	0.852	0.576	0.657
0.6	0.853	0.575	0.655
0.7	0.841	0.565	0.645
0.8	0.765	0.559	0.618

Table 5.8: Result of the YOLOv5 network on the Oxford Pets dataset (test set).

IoU	**Precision**	**Recall**	**mAP**
0.4	0.982	0.966	0.983
0.5	0.982	0.966	0.983
0.6	0.980	0.966	0.983
0.7	0.978	0.965	0.982
0.8	0.975	0.963	0.980

5.2 Neural Architecture Search

The "overall picture" from the experiments with the selected NAS networks can be summarized as follows. The articles listed in Table 4.6 are not described in a level of detail that allows reimplementation. We contacted all authors associated with the works listed in Table 4.6 and mostly got no answer at all, or on rare occasions, got an explanation that they could share neither data nor code, and, sometimes, not even details about their methods. We acknowledge the exception: VEGA/SP-NAS [59] shared their data and code. Also, VEGA's authors have answered all of our questions, which makes SP-NAS the only reproducible NAS for object detection.

5.2.1 SP-NAS

To reproduce the SP-NAS experiments [22], we used an open-source AutoML tool called VEGA[2] [59]. The original code of the SP-NAS network is not available by the authors.

During the SP-NAS environment configuration process, we only experienced difficulty related to executing on multiple GPUs. We used the `dask` library[3] for distributing the processes. However, in the SP-NAS configuration process, the authors did not include the installation of this library, so we implemented it ourselves.

After the `dask` installation, we still had some problems in the parallel execution. The consumption of the GPUs was too low and, consequently, the execution was relatively slow. For example, we distributed the execution into three NVIDIA Quadro RTX 8000 GPUs: one achieved the maximum consumption of 28% of its processing capacity, while the other two did not reach 2% of consumption. Thus, we made some changes to the code regarding the `dask` configuration. We could not reach the maximum processing rate, but, in the previous example, the processing percentage after the adjustments oscillated between 15% and 30% on all three GPUs.

With the proper environment for execution, we replicated the SP-NAS experiments using the hyperparameters and dataset reported in the paper [22]. However, we achieved results inferior to those reported in the paper, as shown in Table 5.9. We reported our results to the VEGA authors through GitHub issues. Unfortunately, they informed that the algorithm available (version 1.4) does not perform well because (not surprisingly) it needs the pre-trained backbone in the ImageNet. According to the authors, the code will be refactored to improve search efficiency and will be available in future versions.

Following the VEGA training, we applied the same setup as suggested on the VEGA repository[4], including a learning rate of 0.02, stochastic gradient descent with a momentum of 0.9, weight decay of 0.0001, batch size of 4, and 25 epochs.

The input data must be formatted according to the MS-COCO dataset to perform the SP-NAS execution through the VEGA framework. For the Insulator dataset, it was necessary to convert the data to the MS-COCO input format.

The first conversion attempt was made using the tool FiftyOne[5], as recommended by

[2] `https://www.noahlab.com.hk/opensource/vega`
[3] `https://dask.org`
[4] `https://github.com/huawei-noah/vega/blob/master/examples/nas/sp_nas/spnas.yml`
[5] `https://voxel51.com/docs/fiftyone`

Table 5.9: Comparison between the results of the replicated experiment performed on VEGA and the results reported in the SP-NAS paper on MS-COCO 2017.

Experiments	**mAP**
Ours	0.168
Original	0.491

MS-COCO's authors. When performing the conversion, due to the absence of information in the original annotation files of the Insulator dataset, a field such as "isCrowd" and "segmentation" were not filled in, producing errors in the data reading step.

In another attempt, we used the voc2coco[6] tool, which created a more suitable tool for replacing the missing values.

With the data adequately organized, we could finally execute the model. However, the obtained results were very poor. Hence, we decided to proceed to the other datasets with similar characteristics to our Insulator dataset (few classes and relatively few samples). Table 5.10 displays the results.

Table 5.10: Result of the SP-NAS (by VEGA) on Insulator dataset, BCCD, Aquarium and Oxford Pets.

	mAP	
Dataset	**SP-NAS**	**YOLOv5**
Insulator	0.005	0.949
BCCD	0.593	0.890
Aquarium	0.362	0.657
Oxford Pets	0.800	0.983

Comparing the results obtained with YOLO5, we observed that the SP-NAS did not perform better in any execution.

5.2.2 DetNAS

We initially tried to replicate the DetNAS experiments by configuring the environment's experiment according to the repository requirements. We faced some outdated libraries and dependencies. After many discussions in the GitHub forum, we have configured the environment and started the DetNAS experiment replication.

DetNAS consists of three phases. The first one is the *dataset configuration*. The authors provided, in .json files, annotated data split into training and validation sets. They used the MS-COCO 2014 dataset (*coco_2014_train* + *coco_2014_valminusminival*), in which 5000 images are selected for the validation set, and the remaining are used in the super-network training process.

[6] `https://github.com/yukkyo/voc2coco`

The second step is the *supernet training*. First, the supernet needs to be pre-trained on the ImageNet dataset; as reported in the paper, it is a mandatory step for object detection backbones. The authors provided the pre-trained model in their repository, and it is possible to pre-train the model on our own. As our objective was to replicate the original experiments and avoid efforts with this process, we chose to use the model provided by the authors.

Thus, we proceeded to the super-network training in the MS-COCO dataset. During this execution, we also faced a few problems. Errors were triggered, making it impossible to complete the process. Many of the errors were related to the external library `maskrcnn-benchmark`[7] used in the project. After a few days of dealing with the error correction, it was possible to perform the supernet training using 4 NVIDIA RTX 8000 GPUs.

The third step is the *server configuration* to distribute the search for candidate networks. For this, we used the `RabbitMQ`[8] library; the installation and configuration of this library required efforts due to the Recod.ai laboratory[9] cluster's access restrictions. Fortunately, the configuration was successfully accomplished, and we started searching for candidate networks.

Simultaneously with the previous step, we evaluated the candidate networks. As illustrated in Figure 5.5, a process is responsible for generating the codes of the candidate architectures and distributing the networks in other parallel processes to be evaluated individually. It also returns the networks.

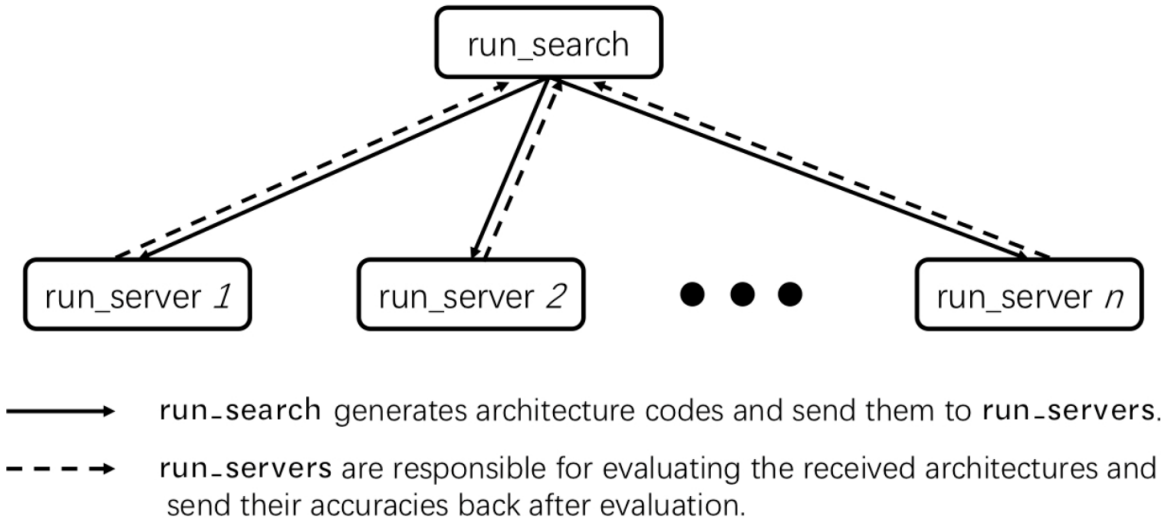


Figure 5.5: Generation and evaluation process of candidate networks. Figure reproduced from `https://github.com/megvii-model/DetNAS`.

As a result, a "cryptic" value was returned (Figure 5.6). After (re)reading the paper and analyzing the result (revisiting the code), we hypothesized that the result is numerical

[7] `https://github.com/facebookresearch/maskrcnn-benchmark`
[8] `https://www.rabbitmq.com`
[9] https://recod.ai

lists where each is a network block, and each number represents a layer. However, we were not able to use this result. It was not clear which operation each number represented.

```
> (2, 3, 2, 0, 2, 3, 3, 3, 2, 0, 3, 1, 0, 3, 3, 3, 3, 3, 2), (2, 3, 1, 1, 2, 0, 3, 3, 2, 0, 3, 1, 0,
3, 3, 3, 3, 3, 3, 2), (1, 3, 2, 3, 2, 2, 3, 3, 2, 0, 3, 1, 0, 3, 3, 3, 3, 3, 2), (1, 3, 2, 3, 2, 2,
3, 3, 2, 0, 3, 1, 2, 3, 3, 3, 3, 3, 2), (1, 3, 2, 3, 2, 2, 3, 3, 2, 0, 3, 1, 2, 3, 3, 3, 3, 3, 3,
2), (2, 3, 2, 0, 2, 0, 3, 3, 2, 0, 3, 1, 0, 3, 3, 3, 3, 3, 2), (2, 3, 2, 3, 2, 2, 3, 3, 2, 0, 3, 1,
0, 3, 3, 3, 3, 3, 2), (2, 3, 2, 3, 2, 2, 3, 3, 2, 0, 3, 1, 0, 3, 3, 3, 3, 3, 2), (2, 3, 2, 3, 2,
2, 3, 3, 2, 0, 3, 1, 0, 3, 3, 3, 3, 3, 2), (2, 3, 2, 3, 2, 0, 3, 3, 2, 0, 3, 1, 0, 3, 3, 3, 3, 3, 3,
2)
```

Figure 5.6: Result from executing DetNAS.

Unfortunately, the turnaround time when contacting the project's authors through GitHub was too long (a month). Therefore, considering the efforts we had already dedicated to solving network problems, we decided to stop the experiments with DetNAS.

5.2.3 PC-DARTS + SSD

Due to the scarcity of NAS networks suitable for object detection, we looked for alternative methods.

PC-DARTS [22] is based on the research of differentiable architecture (DARTS) [35] to present a solution with a more efficient search and with reduced use of computational power compared to DARTS. By identifying that the best supernet does not necessarily produce the best sub-net, the solution was designed to randomly select a portion of the supernet channels, streamlining the search process and addressing specific optimization gaps.

For the classification task, PC-DARTS obtained better accuracy than DARTS in the CIFAR-10 dataset (2.57% vs. 2.76%). In the experiments, the feasibility of using the network for object detection was also verified, using the network found as a backbone for a Single Shot Detector (SSD) [37], obtaining a good result when compared to the use of the SSD300 and SSD512 with the backbone of the state-of-the-art of the time, the VGG-16 [53].

PC-DARTS has in its code structure two essential definitions: *primitives*, which consists of the list of possible operations of the network (e.g., 5×5 convolution, 3×3 max pooling, 3×3 average pooling), and the *genotype*, that indicates the sequence according which these operations are connected. When performing the architecture search process, it returns a genotype, which establishes the best combination of the primitive operations.

Following the authors, we applied the same genotype for CIFAR-10, obtaining similar results to the PC-DARTS paper (see Table 5.11) using 1 GPU NVIDIA RTX 8000.

The authors also reported results for the ImageNet dataset. In this scenario, we parallel the process, allowing its execution in multiple GPUs. Following the PC-DARTS experiments, we needed to perform the ImageNet search to obtain the network's backbone to plug the SSD detector. We faced some problems concerning the ImageNet dataset,

Table 5.11: Comparison between the PC-DARTS results replicated (ours) with the original paper.

Experiments	Dataset	Test Error (%)
Ours	CIFAR-10	3.58
Original	CIFAR-10	2.57
Ours	Tiny ImageNet	38.19
Original	ImageNet	24.20

and, as an alternative, we chose to use the Tiny ImageNet dataset[10], which consists of 200 training classes, and each class has 500 images. The test set contains 10,000 images, and all images are 64×64 pixels. The results can be seen in Table 5.11. We highlight that the results on Tiny ImageNet and ImageNet are not comparable.

Next, we attempted to couple the generated network to the object detector but were unsuccessful. Although this process was reported in the paper, the implementation is not available. We encountered difficulty dealing with the backbone's output and converting it into a valid input for the SSD detector.

5.2.4 NAS without Training

NAS without training [39] was designed for the classification task, but following the same proposal as PC-DARTS [67], we decided to try to couple the network generated by this NAS as a backbone to an object detector.

Furthermore, and most relevant, NAS without training presents a proposal consistent with one of the realities we want to evaluate: the use of NAS in infrastructure with limited computational power. This solution partially predicts the trained accuracy of a network from its initial state, making it possible to search for robust networks without *any* training in a matter of seconds and on a single GPU.

We execute the network on CIFAR-10 and CIFAR-100 datasets to reproduce the experiments. We successfully obtained the same accurate results reported by the authors. So, we decided to perform the experiments in other datasets. It was a somewhat laborious process since usage in specific datasets is not reported. Also, the project[11] has a specific dependence on some NAS benchmarks. However, the first author confirmed the possibility of using another dataset and provided some instructions for implementing this proposal.

We modified the code and used it for the classification task on skin lesion data from ISIC Challenge 2020[12], obtaining an AUC of 0.82 in the validation set. However, like PC-DARTS, it was not possible to couple an object detector to the classification network for the same reasons.

[10] `https://www.kaggle.com/c/tiny-imagenet/data`
[11] `https://github.com/BayesWatch/nas-without-training/releases/tag/v1.0`
[12] `https://challenge2020.isic-archive.com`

5.2.5 H2O.ai

We analyzed the feasibility of using the AutoML open-source tool H2O[13], which includes automatic training and tuning of many models (such as fully-connected multi-layer neural networks) within a user-specified time limit.

Despite showing reasonable solutions for tabular data, the documentation did not report resources for object detection. We contacted the team from H2O, and we were informed that the solutions for computer vision tasks would be available in the H2O Driverless AI[14] platform, but this functionality is not free.

5.2.6 UFOD

We also tried to use the Unified Framework for Object Detection (UFOD)[15] [13], an open-source AutoML framework for constructing object detection models and ensembling them. UFOD is a set of implementations of object detection networks known in the literature (e.g., YOLO, Faster-RCNN, RPN) with the possibility of simultaneous execution to identify which one generates a better result for a given dataset.

Some environment configuration problems arose during the experiments with UFOD and made it impossible to evaluate this solution. We chose not to insist on this as it automatically diverged from our proposal to build a deep neural network architecture.

[13] `https://www.h2o.ai`
[14] `https://www.h2o.ai/products/h2o-driverless-ai`
[15] `https://github.com/ManuGar/UFOD`

Chapter 6

Conclusion

For the past couple of years, researchers and companies have tried making deep learning more accessible to non-experts by providing access to pre-trained computer vision models, for example. Using a pre-trained model for another task is known as transfer learning, but it requires sufficient expertise to fine-tune the model on another dataset. Fully automating this procedure allows even more users to benefit from the significant progress that has been made in machine learning to date.

During the development of this Master's thesis, we noticed that despite the advances in the field of Automated Machine Learning (or AutoML), we are still far from democratizing access to Machine Learning techniques and the complete automation of the process (if, in fact, possible).

Even when faced with more robust AutoML tools, such as the Google AutoML platform, we are dealing with accessibility to enable people without much technical knowledge to use that framework of tools. However, we are limiting it, on the other hand, due to the need for a robust machine to execute the processes and the requirement to use google machines that need to be paid to be accessed.

Tools with open-source versions, such as H2O, already break this paradigm. Maybe they are closer to this reality of democratization in terms of availability of execution, but in terms of ease of execution, it is a problem since more technical knowledge is required to enjoy the free functionality of the tool. However, in paid versions, the structuring of the complete ML pipeline is coupled to a single tool with more didactic use.

Focusing only on the neural network architecture construction stage makes us realize that NAS techniques reduce efforts and optimize neural network construction time. Notwithstanding, they require robust machines to train thousands of models before converging. In addition, exploring specific search spaces for each of the domains limits the transfer of knowledge.

Other difficulties identified during this Master's thesis reinforce the thought that we may be advancing, but we still remain pretty distant from the perspective of providing open and easy access for many people to ML tools. We experienced a frustrating experience when faced with non-available codes, poorly structured codes, networks with extremely robust infrastructure needs, and processes encompassing only one type of problem. In our data in a more straightforward database like an Oxford Pets experiments (only two classes, centralized objects, detection area only the face of the animal), we

obtained satisfactory results (mAP: 0.800) with the SP-NAS Compared to the obtained result by the YOLO network however for the same database (mAP: 0.983) difference when an accuracy difference is significant. All these difficulties led us to question whether this is the best solution to democratize access to machine learning.

However, if AutoML is not the best solution, how can we remove the obstacles using machine learning? Rachel Thomas, founding director of the Center for Ethics in Applied Data at the University of San Francisco and one of the founders of the fast.ai organization, discussed in a blog post[1] about the AutoML hype and some ways to solve this problem. We summarized those ideas in the following:

1. **Make deep learning easier to use through techniques that make training a network simpler and faster.** Some findings that make this possible are techniques such as dropout [56] which allows training on smaller datasets without excessive tuning or superconvergence [54], speeding up the training process, and requiring fewer computational resources. **Unmasking myths** reinforce the thought that it is only possible to propose something if we have massive datasets and compelling computers or that these techniques can only be performed by a specialist with the highest degree of training in the area. **Increase access for people who do not have the money or credit cards needed to use a cloud GPU**. While the cost of cloud GPUs is within the budget of some, many people still cannot afford it. To offer open notebooks such as Google Colab[2] that provide an environment for using Jupyter Notebooks (.ipynb) files that do not require configuration to use and can be executed entirely in the cloud, giving a user access to a free GPU (currently using long-lived GPU in this tool is not usable for free) can promote more democratization than a robust tool like AutoML.

Therefore, we conclude that the techniques proposed by AutoML can be a great tool to assist and speed up some processes performed by a specialist in the area and even provide superior results in specific steps. Even some paid technology, such as Google AutoML, proves to be a great solution for a start to using machine learning for those who can afford it. However, we cannot fit this solution as democratic if it is not easily accessible for those interested in using it.

[1] `https://www.fast.ai/2018/07/23/auto-ml-3`
[2] `https://colab.research.google.com`

Bibliography

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *ArXiv*, abs/2004.10934, 2020. 21, 51

[2] Liang-Chieh Chen, G. Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 39

[3] Xiangning Chen, Bo Qiao, W. Zhang, W. Wu, Murali Chintalapati, Dongmei Zhang, Qingwei Lin, Chuan Luo, X. Li, H. Zhang, Y. Xu, Yingnong Dang, Kaixin Sui, and X. Zhang. Neural feature search: A neural architecture for automated feature engineering. In *IEEE International Conference on Data Mining*, pages 71–80, 2019. 25

[4] Yukang Chen, T. Yang, X. Zhang, Gaofeng Meng, X. Xiao, and Jian Sun. DetNAS: Backbone search for object detection. In *Conference on Neural Information Processing Systems*, 2019. 14, 17, 22, 37, 49, 50

[5] Ekin Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc Le. Autoaugment: Learning augmentation strategies from data. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019. 25

[6] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Conference on Neural Information Processing Systems*, page 379–387, 2016. 20

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009. 14, 19, 24, 37

[8] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *ArXiv*, abs/1708.04552, 2017. 25

[9] Xin Luna Dong and Divesh Srivastava. Big data integration. In *IEEE International Conference on Data Engineering*, pages 1245–1248, 2013. 25

[10] Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V. Le, and Xiaodan Song. Spinenet: Learning scale-permuted backbone for

recognition and localization. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 11589–11598, 2020. 39, 49, 50

[11] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(1):1997–2017, 2019. 14, 27, 28, 30, 35

[12] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. 36

[13] Manuel García-Domínguez, César Domínguez, Jónathan Heras, Eloy Mata, and Vico Pascual. UFOD: An AutoML framework for the construction, comparison, and combination of object detection models. *Pattern Recognition Letters*, 145:135–140, 2021. 62

[14] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7029–7038, 2019. 14, 40, 49, 50

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. 19, 20

[16] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Conference on Neural Information Processing Systems*, page 2672–2680, 2014. 24, 25

[17] Kristen Grauman and Bastian Leibe. *Visual Object Recognition*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2011. 18

[18] Jianyuan Guo, Kai Han, Yunhe Wang, Chao Zhang, Zhaohui Yang, Han Wu, Xinghao Chen, and Chang Xu. Hit-detector: Hierarchical trinity architecture search for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 11402–11411, 2020. 22, 41, 42, 49, 50

[19] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *IEEE International Conference on Computer Vision*, pages 2980–2988, 2017. 20

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 22, 28

[21] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021. 23, 24, 25, 26, 27, 28, 30, 32, 33

[22] Chenhan Jiang, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Sp-nas: Serial-to-parallel backbone search for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 11860–11869, 2020. 38, 49, 50, 57, 60

[23] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868, 2019. 19

[24] Sanjay Krishnan, Michael J Franklin, Ken Goldberg, and Eugene Wu. Boostclean: Automated error detection and repair for machine learning. *ArXiv*, abs/1711.01299, 2017. 25

[25] Sanjay Krishnan and Eugene Wu. Alphaclean: Automatic generation of data cleaning pipelines. *ArXiv*, abs/1904.11827, 2019. 25

[26] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 24, 32

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 19

[28] Erin LeDell. H2o automl: Scalable automatic machine learning. In *7th ICML Workshop on Automated Machine Learning (AutoML)*, 2020. 13

[29] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. DetNet: Design backbone for object detection. In *European Conference in Computer Vision*, pages 339–354, 2018. 36

[30] Tingting Liang, Yongtao Wang, Guosheng Hu, Zhi Tang, and Haibin Ling. Opanas: One-shot path aggregation network architecture search for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021. 40, 49, 50

[31] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 936–944, 2017. 22, 39, 40

[32] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42:318–327, 2020. 40, 41

[33] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference in Computer Vision*, 2014. 14, 18, 24, 36, 51

[34] Hanxiao Liu, K. Simonyan, Oriol Vinyals, Chrisantha Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018. 14, 30, 31

[35] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *International Conference on Learning Representations*, 2019. 33, 60

[36] Li Liu, Wanli Ouyang, X. Wang, P. Fieguth, J. Chen, Xinwang Liu, and M. Pietikäinen. Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, 128:261–318, 2019. 17, 18, 21

[37] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. In *European Conference in Computer Vision*, pages 21–37, 2016. 60

[38] Ningning Ma, X. Zhang, Haitao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *European Conference in Computer Vision*, 2018. 37

[39] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. Neural architecture search without training. In *International Conference on Machine Learning*, 2021. 61

[40] Agnieszka Mikołajczyk and Michał Grochowski. Style transfer-based image synthesis as an efficient regularization technique in deep learning. In *International Conference on Methods and Models in Automation and Robotics*, pages 42–47, 2019. 25

[41] Agnieszka Mikołajczyk and M. Grochowski. Data augmentation for improving deep learning in image classification problem. In *International Interdisciplinary PhD Workshop*, pages 117–122, 2018. 25

[42] David Moher, Alessandro Liberati, Jennifer Tetzlaff, Douglas G Altman, and Prisma Group. Preferred reporting items for systematic reviews and meta-analyses: the prisma statement. *PLoS medicine*, 6(7):e1000097, 2009. 35

[43] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505, 2012. 48

[44] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, 2018. 34

[45] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2015. 21

[46] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 6517–6525, 2017. 21

[47] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *ArXiv*, abs/1804.02767, 2018. 21, 51

[48] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys*, 54(4), 2022. 20

[49] Shaoqing Ren, Kaiming He, Ross B. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015. 41

[50] Dan Roth and Kevin Small. Interactive feature space construction using semantic information. In *Conference on Computational Natural Language Learning*, 2009. 26

[51] Robert E. Schapire. *The Boosting Approach to Machine Learning: An Overview*, pages 149–171. Springer New York, New York, NY, 2003. 25

[52] Shenggan. BCCD Dataset. `https://github.com/Shenggan/BCCD_Dataset`, 2017. 8, 47

[53] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 60

[54] Leslie N. Smith and Nicholay Topin. Super-convergence: very fast training of neural networks using large learning rates. In *Defense + Commercial Sensing*, 2019. 64

[55] Parikshit Sondhi. Feature construction methods : A survey. Technical report, Univeristy of Illinois, 2009. 26

[56] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. 64

[57] Zhenhong Sun, Ming Lin, Xiuyu Sun, Zhiyu Tan, Hao Li, and Rong Jin. Mae-det: Revisiting maximum entropy principle in zero-shot nas for efficient object detection. In *International Conference on Machine Learning*. PMLR, 2022. 49, 50

[58] Inc. Ultralytics. YOLOv5. `https://github.com/ultralytics/yolov5`, 2020. 22, 51

[59] Bochao Wang, Hang Xu, Jiajin Zhang, Chen Chen, Xiaozhi Fang, Ning Kang, Lanqing Hong, Wei Zhang, Yong Li, Zhicheng Liu, Zhenguo Li, Wenzhi Liu, and Tong Zhang. VEGA: Towards an End-to-End Configurable AutoML Pipeline, 2020. 57

[60] Jiaqi Wang, Kai Chen, Shuo Yang, Chen Change Loy, and Dahua Lin. Region proposal by guided anchoring. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2960–2969, 2019. 22

[61] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, and Chunhua Shen. Nas-fcos: Fast neural architecture search for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 11940–11948, 2020. 41, 49, 50

[62] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *International Conference on Machine Learning*, 2016. 14, 34

[63] Lilian Weng. Neural architecture search. *lilianweng.github.io/lil-log*, 2020. 14, 30, 34

[64] Yue Wu, Yinpeng Chen, Lu Yuan, Zicheng Liu, Lijuan Wang, Hongzhi Li, and Yun Fu. Rethinking classification and localization for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 10183–10192, 2020. 22

[65] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5987–5995, 2017. 22

[66] Hang Xu, Lewei Yao, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In *IEEE International Conference on Computer Vision*, pages 6648–6657, 2019. 14, 22, 23, 41, 50

[67] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020. 61

[68] Lewei Yao, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. SM-NAS: structural-to-modular neural architecture search for object detection. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 12661–12668, 2020. 22, 42, 49, 50

[69] Quanming Yao, Mengshuo Wang, Hugo Jair Escalante, Isabelle Guyon, Yi-Qi Hu, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *CoRR*, abs/1810.13306, 2018. 23, 26

[70] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2019. `http://www.d2l.ai`. 18

[71] Hongyi Zhang, Moustapha Cissé, Yann Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. 25

[72] Xin Zhang, Yee-Hong Yang, Zhiguang Han, Hui Wang, and Chao Gao. Object class detection: A survey. *ACM Computing Surveys*, 46, 10 2013. 18, 19, 21

[73] Guanghui Zhu, Zhuoer Xu, Xu Guo, C. Yuan, and Yihua Huang. DIFER: Differentiable Automated Feature Engineering. In *International Conference on Automated Machine Learning*, 2022. 25

[74] Marc-André Zöller and Marco F Huber. Benchmark and survey of automated machine learning frameworks. *Journal of Artificial Intelligence Research*, 70:409–472, 2021. 22, 25

[75] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. 14, 29, 31, 32, 35, 40

[76] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018. 14, 29, 30

[77] Zhengxia Zou, Z. Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *ArXiv*, abs/1905.05055, 2019. 18, 19, 20, 21