Segmentation multi-capteurs de déchets métalliques pour tri robotisé

Auteur : Raimondi, Rémi
Promoteur(s) : Louppe, Gilles
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil électricien, à finalité spécialisée en "signal processing and intelligent robotics"
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/11426

University of Liège
Faculty of Applied Sciences

Multi-sensor instance segmentation of scraps for robotic sorting

Master's thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering

by

Rémi Raimondi

Supervisor: Prof. Gilles Louppe
Co-supervisor: Pierre Barnabé

Academic Year 2020-2021

Abstract

During the last decades, the intensive exploitation of natural resources has known a significant growth due to the increasing need for raw materials. As a consequence, the environment has been subjected to a lot of damage. Modestly, by improving recycling and waste recovery, this master's thesis contributes to a partial solution that could slow down the exploitation of natural resources. Indeed, this thesis is carried out as part of the **Multipick** project which is aiming at creating an industrial demonstrator capable of sorting over 20,000 tonnes of metals per year at a rate of 16 scraps per second thanks to the use of robots and artificial intelligence. More precisely, the objective of this master's thesis is to study and assess the potential of deep neural networks to increase the performance of the actual waste characterization system operating inside a prototype version of the demonstrator called **Pick-it**. The innovation of this work lies in the type of input processed by the deep neural network: instead of using classical RGB images, the deep neural network is fed with multi-feature tensors composed of eleven 2D images staked together and containing diverse characteristics extracted from a dual X-Ray transmission sensor, a 3D ranging camera, and a hyperspectral camera. Leveraging these multi-feature tensors, a fully functional and optimized Mask R-CNN deep neural network is successfully integrated inside the pioneer Pick-it prototype. This model has been shown to achieve excellent results in terms of localization but reaches too low classification accuracy with the prototype to outperform the current waste characterization method. Nevertheless, the achieved results show great potential and unveil various directions of future work that could lead to the creation of a deep learning model outperforming the current characterization method and reaching state-of-the-art performance in the task of multi-sensor instance segmentation.

Acknowledgments

First, I would like to thanks my supervisor and co-supervisor, Professor Gilles Louppe and Mister Pierre Barnabé to let me work on this really innovative and pioneer project.

More precisely, I would like to express my thanks to Professor Gilles Louppe for giving me precious feedback, advice throughout the whole master's thesis, and for giving me access to the Alan cluster to carry out all my experiments.

I also would like to express my special thanks to Pierre Barnabé for its great technical support, advice, and collaboration on the data acquisition and integration of the model inside the prototype.

I am also thankful to my family and close friends for their unconditional support.

Finally, special thanks go to Clara Daga, Nicolas Mahiat, and Maxime Vandegar who accepted to proofread parts of this work.

Contents

Chapter 1

Introduction

1.1 Context

This master thesis is carried out as part of the Multipick project which aims at building an industrial demonstrator composed of 16 robots that should eventually sort nearly 20,000 tonnes of metals per year at a rate of 16 pieces per second: a world premiere. The building of the Multipick demonstrator is based on a prototype version built previously in the context of another project called Pick-it (this project ended in 2019). This previous project fitted itself into the "Reverse Metallurgy" consortium formed in the Walloon Region. This consortium started in 2015 with the aim of promoting recycling in Wallonia. More precisely, its main goals are to create a pole of expertise in this field, find new valorization streams to boost the economy, and favor circular economy and autonomy regarding primary and secondary materials. Pick-it was one component of this broad project which is dedicated to physical separation of waste (mainly metallic scraps) in several high grades concentrates using robots. To this aim, three sensors (a dual X-Ray transmission sensor, a 3D ranging camera, and a hyperspectral camera) have been placed above a conveyor belt to extract data about the scraps. Then, taking these data as input, an algorithm detects and classifies them. Finally, from these detections and classifications, precise commands are sent to the robots to sort the wastes into the right container (specific to each category of material).

This Multipick project is conducted by the GeMMe research team for the Comet group which is a Belgian industrial group, specialized in trading ferrous and non-ferrous metals and their derivatives (Group, 2021). Briefly, GeMMe is a research group of the University of Liège that actively contributes to the development of innovative processes for a more efficient management of mineral and metallic resources (GeMMe, 2021).

1.2 Problem statement

Thanks to the use of classical machine learning approaches, the Pickit team has already obtained great results with their prototype. This has led their industrial partner (Comet Group) to start, in 2020, the upscaling of the Pick-it technology at their recycling plant through the Multipick project.

However, their current characterization method is only using the spectral information of the scraps to classify them and does not exploit any spatial information such as shape and texture. Because of that, the Pick-it team is convinced that their current solution is very likely to be suboptimal.

Moreover, due to the non-exploitation of the spatial information, the scope of application of their current method is limited: for instance, they are unable to distinguish laminated zinc from zamak, to locate welds, and to recognize batteries.

Therefore, they decided to leverage the deep learning field to increase the performance of their characterization system by using advanced deep learning architectures that are able to handle simultaneously both spectral and spatial information provided by the different sensors.

This objective is exactly what this master's thesis is aiming for: implementing a high-performing instance segmentation deep learning method end-to-end (from data acquisition to its integration into the real Pick-it prototype). Due to the physical implementation of the prototype, the instance segmentation method must be able to make its predictions in less than 500 ms.

1.3 Outline

The main content of this master's thesis truly begins in Chapter 2 by describing the current equipment setup and the characterization method currently implemented in the Pick-it prototype. Then, it proceeds in Chapter 3 with a recall of some key concepts from the deep learning theory and a detail description of the Mask R-CNN deep neural network. Chapter 4 briefly reviews the current state-of-the-art deep neural networks in instance segmentation and highlights the contribution of this master's thesis. Chapter 5 presents how to build a convenient instance segmentation dataset from the data acquired on the conveyor belt. Chapter 6 is dedicated to the detail description of the specific Mask R-CNN architecture that is optimized in Chapter 7 and evaluated in Chapter 8. Chapter 9 concludes with main directions for further researches and improvements. Finally, Chapter 10 contains the appendices with some relevant auxiliary data.

Chapter 2

Current equipment and method

Outline
In this short chapter, we introduce the current equipment setup and characterization method implemented in the Pick-it prototype. More precisely, in Section 2.1, we describe the combination of sensors currently available and the underlying data generated. In section 2.2, we briefly explain the current characterization method.

2.1 Equipment setup and available data

To be able to classify the scraps, the Pick-it team has currently installed three types of sensors. Each of those sensors measures a specific line of the conveyor belt and is continuously triggered by an encoder. More precisely, this encoder follows the conveyor belt in such a way that it enables at the end to build multi-feature images.

The first sensor is a 3D profiling camera which captures a millimeter estimation of the top height profile of the samples. Consequently, this sensor is really useful to segment the scraps.

The second one is a Visible Near-InfraRed (VNIR) hyper-spectral camera from which eight reflectance intensities are sampled in the range 400 to 1000 nm. The reflectance corresponds to the percentage of energy reflected in a specific wavelength range. Digitally, it is stored inside an UINT16 variable: 100 % reflectance is equal to 65535. Roughly, these data enable to discriminate objects based on their colors.

The third and final sensor is a dual-energy X-Ray Transmission (XRT) detector, which enables the computation of the X-ray transmission induced by the scraps passing between the X-ray source

Sensor	Spatial resolution (mm)	Number of features
3D profile camera	0.5	1
VNIR camera	0.35	8
XRT detector	0.7	2

Table 2.1: Spatial resolution and number of features associated to each sensor operating in the Pick-it prototype.

and the detector. Thanks to the dual X-ray energy, two transmission energies are measured. Indirectly, these measures give valuable data about the density of the different scraps.

Table 2.1 describes the spatial resolution and the number of features associated to each sensor. Figure 2.1 shows a simplify model of the current Pick-it prototype where the Laser-Induced Breakdown Spectroscopy (LIBS) scanning system is the most recent addition to the prototype but is still currently being calibrated and is therefore not used in the context of this master's thesis.

Regarding their current dataset, it is composed of approximately 2000 samples which have been manually sorted using a portable X-Ray fluorescence probe among the following 6 classes (categories):

- Aluminum
- Copper
- Brass
- Zinc
- PCB
- Nickel

However, in the context of this master's thesis, the zinc class has been split into the laminated zinc and zamak classes to perform a finer sorting. The paint and rubber classes have been added by the Pick-it team after observing that the red painted objects were systematically classified as copper and the rubber objects as aluminum. Therefore, in this master's thesis, 9 classes of objects are considered in total.

Appendix I shows visually several physical samples for each class.

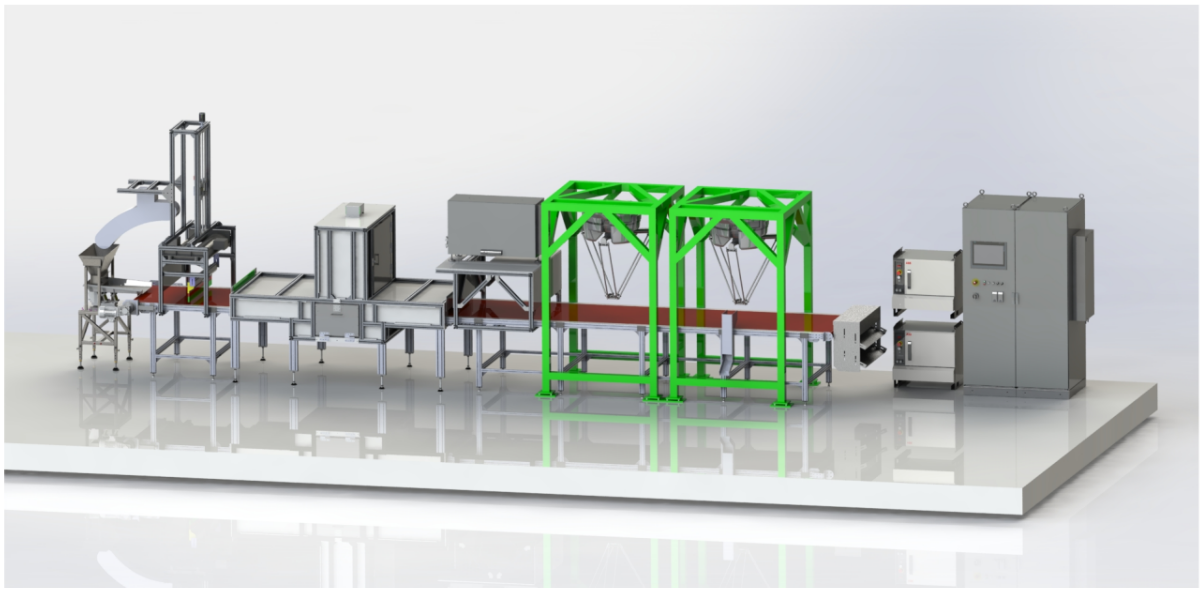


Figure 2.1: Model of the current Pick-it prototype with respectively from left to right: the waste feeder, 3D camera, VNIR camera, XRT module, LIBS scanning system, delta-robots and electrical boxes.

2.2 Current characterization method

In summary, the current method operates in four main steps.

Firstly, the objects are segmented using the 3D profile obtained via the 3D profiling camera. More precisely, the method filters out of the 3D profile detections which are smaller than 1 mm. Then, using a classical blob detection technique, it finds and extracts the different objects. To avoid false object detections, a blob of pixels is considered as an object if and only if its area is greater than 300 pixels.

Secondly, the data acquired via the X-Ray module and VNIR camera are mapped onto the referential area defined by the 3D profile to create for each pixel of each detected object a vector of features. Concretely, each feature vector is composed of ten elements: the first eight correspond to the frequency data given by the VNIR camera, while the last two are the dual-energy X-Ray transmission.

Thirdly, all these vectors are fed into a machine learning algorithm called Support Vector Machine (Wang, 2005) to classify each pixel.

Finally, for each segmented object, statistics on the corresponding pixel predictions are computed and then fed into a second Support Vector Machine model to finally predict a class at the object level.

Chapter 3

Background

Outline
In this chapter, we recall some key concepts from the deep learning theory and describe in detail the Mask R-CNN deep neural network. In Section 3.1, we introduce fundamental concepts on which deep neural networks build upon. In Section 3.2, we describe what were the important milestones and concepts that led to the creation of Mask R-CNN, and we explain its working principle in detail.

3.1 Deep learning basics

The purpose of this section is to recall some key concepts from the deep learning theory upon which the Mask R-CNN model is based on. Therefore, this section is not a complete overview of the deep learning theory but an oriented tour of the main concepts that are required to understand the Mask R-CNN method.

3.1.1 Link between deep and machine learning

Deep learning (DL) is a subset of machine learning (ML) which itself is a subset of the artificial intelligence (AI) field. Figure 3.1 illustrates these connections. Artificial intelligence is the discipline of making machines or programs which are able to reproduce a human reasoning and/or to exhibit intelligent behavior (Latombe, 2011), while machine learning is concerned with the design and study of algorithms which are able to learn the parameters of a model from observed data. Deep learning is focused on the creation and the analysis of machine learning algorithms that employ (deep) neural networks.

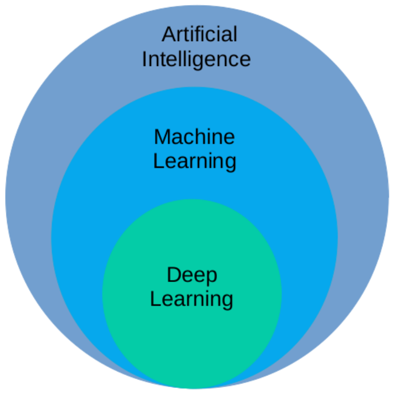


Figure 3.1: Link between Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL): DL $\subset$ ML $\subset$ AI.

3.1.2 Supervised learning

As deep learning is a subset of machine learning, it also requires data to operate. More precisely, in the context of this master's thesis, we focus on a supervised learning task. This supervised setting means that the algorithms learn by exploiting data composed of input-output pairs obtained by observing (or simulating) the system of interest. That is,

$$(\mathbf{x}_i, y_i) \sim P(X, Y) \tag{3.1}$$

with $\mathbf{x}_i \in \mathcal{X}$ (the input vector), $y_i \in \mathcal{Y}$ (the output), $i = 1, ..., N$ and $P(X, Y)$ an unknown joint probability distribution.

Using these data, one goal of machine learning is to solve the inference problem which consists in finding the conditional probability given by

$$P(Y = y | X = \mathbf{x}) \tag{3.2}$$

for any new $(\mathbf{x}, y)$.

In practice, this inference problem can take several forms: if the output y is a class ($\mathcal{Y} = \Delta^C$), it is called a classification problem, whereas if the output y is a scalar ($\mathcal{Y} = \mathbb{R}$), it is called a regression problem.

3.1.3 Neural network primitive

Before defining the learning process, let us recall one of the most basic deep neural networks called multi-layer perceptron.

The main primitive of all neural networks is the neuron

$$n = \sigma(\mathbf{w}^T\mathbf{x} + b) \tag{3.3}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ (sigmoid function), $n \in \mathbb{R}, \mathbf{x} \in \mathbb{R}^P, \mathbf{w} \in \mathbb{R}^P$ and $b \in \mathbb{R}$.

Placing q of these units in parallel allows to form a *layer* $\mathbf{l}$ with q outputs:

$$\mathbf{l} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b}) \tag{3.4}$$

where $\mathbf{l} \in \mathbb{R}^q$, $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{W} \in \mathbb{R}^{p \times q}$, $b \in \mathbb{R}^q$ and where σ is extended to the element-wise sigmoid function.

From these layers, a first family of neural networks can be built by composing them in series:

$$\begin{aligned} \mathbf{l}_0 &= \mathbf{x} \\ \mathbf{l}_1 &= \sigma(\mathbf{W}_1^T\mathbf{l}_0 + \mathbf{b}_1) \\ &... \\ \mathbf{l}_L &= \sigma(\mathbf{W}_L^T\mathbf{l}_{L-1} + \mathbf{b}_L) \\ f(\mathbf{x};\theta) &= \mathbf{l}_L \end{aligned} \tag{3.5}$$

where θ corresponds to the model parameters $\{\mathbf{W}_k, \mathbf{b}_k, ...|k = 1, ..., L\}$.

This first family of neural networks are known in the literature as fully connected (feedforward) networks, fully connected layers or multi-layer perceptron. In practice, this family of networks is not limited to use the sigmoid as activation function: there exists many other activation functions.

3.1.4 Training a deep neural network

A preliminary step to train and optimize a deep neural network is to split the training set into two subsets: a training set and a validation set. Another essential preliminary step is to set the number $N_\mathbf{p}$ of model and algorithm hyperparameter vectors $\mathbf{p}$ to explore and their respective values. In machine learning, hyperparameters are parameters whose values directly impact the learning process. For example, very common model and algorithm hyperparameters are the topology of the neural network and the learning rate respectively. Thanks to that, until the $N_\mathbf{p}$ hyperparameter vectors have been processed or until a sufficiently satisfying score has been obtained on the validation set, the training of a deep neural network can be summarized as the repetition of the three following steps:

1. Pick a hyperparameter vector $\mathbf{p}$ (which has not been processed yet).
2. Train the deep learning network with these specific hyperparameters on the training set.
3. Evaluate the resulting model using the validation set.

After that, the final model parameters θ are obtained by training on the whole training set the neural network using the best hyperparameters identified through the previous sequence of steps. Finally, to measure the performance of the final model on unseen (new) data, this final model is usually evaluated on another dataset which exclusively contains data that have never been processed by the network. This extra set is called the testing set or test set.

More precisely, to train any deep neural network (step n°2 of the optimization sequence above), a *loss* function l has to be defined. A loss function gives a scalar measurement of how far the model prediction $f(\mathbf{x};\theta)$ is from the ground truth output y given by the corresponding input-output pair $(\mathbf{x}, y)$ in the dataset. Usually, in a regression setting, the loss is the squared difference between the output predicted by the model $f(\mathbf{x};\theta)$ tuned by the parameters θ and the true output y. In the literature, it is referred as the ℓ_2 loss. That is:

$$l(y, f(\mathbf{x};\theta)) = (y - f(\mathbf{x};\theta))^2 \tag{3.6}$$

In the case of a classification setting and considering that the number of outputs of the neural network is equal to the number of categories U (also called classes) to classify, the most used loss is the *cross entropy*. It is formally defined as follows:

$$l(y, f(\mathbf{x};\theta)) = -\sum_{i}^{U} b_i \ \log\left(\frac{\exp(f(\mathbf{x};\theta)_i)}{\sum_{j=1}^{U}\exp(f(\mathbf{x};\theta)_j)}\right) \tag{3.7}$$

where b_i is the i^{th} element of the one-hot vector $\mathbf{b}$ and is only equal to one if i corresponds to the class number assigned to y. $f(\mathbf{x};\theta)_i$ is the i^{th} output of the model f having the parameters θ and taking the vector $\mathbf{x}$ in input.

The argument of the log function is called a softmax operation and enables to produce from the outputs of the model an estimated probability vector $P(Y|\mathbf{x})$. The cross entropy loss increases as the predicted probability for the ground truth class is far from one, and is exactly equal to zero if the predicted probability for that class is one.

Using this concept of loss, the total training error $\mathcal{L}(\theta)$ (also called empirical risk in the literature) committed by the model $f(\mathbf{x};\theta)$ can be directly computed by the following averaging of the loss over all the N input-output pairs of the training dataset

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} l(y_n, f(\mathbf{x}_n;\theta)) \tag{3.8}$$

with θ corresponding to the model parameters and f representing the model itself.

The main objective of the deep learning training process is to learn the deep neural network parameters (also called weights) θ which minimize the error on unseen data (estimated via the testing set) using the available training data. As the training and unseen data come from the same joint probability distribution $P(X,Y)$, the minimum of the training error is likely to be close to the one of the test error: the inference problem has been turned into a minimization problem. Consequently, this problem can be solved using numerical optimization techniques such as gradient descent.

Gradient descent uses local linear information to iteratively go towards a minimum (hopefully the global one but this is not guaranteed) of the loss function $\mathcal{L}(\theta)$. More precisely, the model parameters θ are updated iteratively as follows:

$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t) \quad \forall t \tag{3.9}$$

with t being an index representing the current iteration, θ_0 the initial parameters of the model and γ a positive scalar called the learning rate.

In the case of a deep neural network model, one may wonder if it is really possible to compute the gradient of the loss with respect to all the model parameters θ. In practice, it is totally possible thanks to the fact that the neural network is just a composition of differentiable functions. Indeed, the total derivatives of the loss required to compute the gradient can be evaluated using the reverse automatic differentiation procedure. Briefly, this procedure consists to first feed the network with the input to compute and store all the intermediate results from inputs to outputs. Then, it evaluates the required derivatives backward by applying the chain rule recursively and using the intermediate values computed previously.

However, in practice, the update rule expressed in Equation 3.9 is almost never used in deep learning because it directly depends on the size of the dataset. Indeed, as the size of the datasets in deep learning is most of the time quite large, the computational cost of using gradient descent is too high to be used inside a practical application.

Instead, to speed up the computation and reduce the variance of each update, the learning error expressed in Equation 3.8 is not computed for each input-output pairs inside the training set but for subsets of pairs called mini-batches. Even with this extension (called mini-batch gradient descent), the gradient descent optimizer has a considerable drawback: it computes the gradient exactly. In others words, it minimizes the empirical risk, although this risk is only an empirical estimation of the error on unseen data (called generalization error). Consequently, (batch) gradient descent is not used in practice. Instead, another optimizer called Stochastic Gradient Descent (SGD) with the following update rule is applied:

$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta l(y_{i(t+1)}, f(\mathbf{x}_{i(t+1)}; \theta_t)). \tag{3.10}$$

where the index $i(t+1)$ denotes the input-output pair picked randomly at iteration $t+1$.

Conversely to the previous update rule, as it always computes the loss for a unique random input-output pair, this new rule is independent of the size of the dataset considered. This SGD optimizer might seem to lead to bad generalization but it actually achieves very good generalization performance despite showing bad optimization performance for minimizing the empirical risk.

Over the years, multiple more advanced optimizers have been proposed on top of SGD to increase the efficiency of the optimization in terms of quality, speed and parameter tuning: nowadays, the two most used optimizers in deep learning are Adam (Kingma & Ba, 2014) and RMSProp (Hinton, 2014).

3.2 Deep learning and computer vision

For several years, deep learning has been revolutionizing the field of computer vision by achieving outstanding results in several of its main tasks.

More precisely, among many tasks, deep learning models have reached state-of-the-art performance in the tasks of classification, classification plus localization, object detection, semantic segmentation, and instance segmentation. Figure 3.2 describes graphically the aims of each of these five tasks.

Briefly, the purpose of the classification task is to assign (among a set of classes) a class to an object in an input picture, whereas the purpose of the semantic segmentation task is to give a class to each pixel of the input image without taking into account the possibility of having the presence of multiple objects of the same class.

The aim of the classification plus localization task is to output in addition of the class, a tight bounding box around the object pointed by the class. Object detection is the generalization of the classification plus localization task to multiple objects. In other words, the goal of the object detection task is to detect every object belonging to a set of classes by delimiting a tight bounding box and assigning a class to each of them, while the instance segmentation task can be seen as

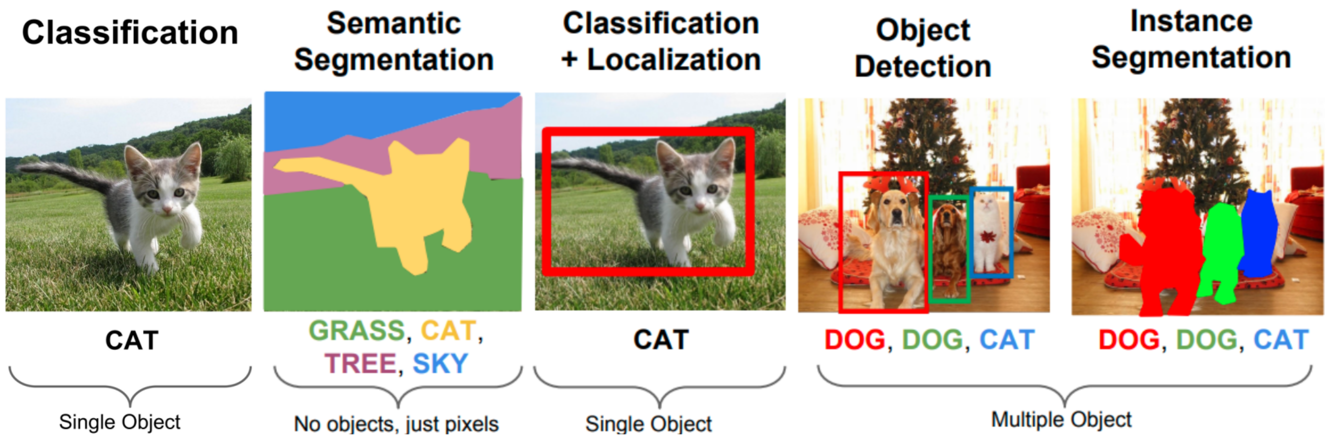


Figure 3.2: Computer vision main tasks (Li et al., 2017) with respectively from left to right: the classification that aims at assigning a class to the picture, the semantic segmentation that assigns a class to every pixel of the image, the classification plus localization that classifies and locates through a bounding box the detected instance, the object detection that extends classification plus localization to multiple instances and finally, the instance segmentation that merges the goals of semantic segmentation and object detection by providing different semantic masks for each detected instance.

the merging of the semantic segmentation task and the object detection one. Indeed, it consists in giving a class to each pixel of each instance (i.e. an object belonging to the set of classes considered) in the picture. Therefore, one could say that this task is the most complex one among the five cited.

In practice, researchers have solved theses tasks incrementally: they first solved the simplest one (classification) and then reused their breakthroughs and tricks to solve increasingly complex tasks to finally solve the instance segmentation task. Therefore, to deeply understand the state-of-the-art instance segmentation neural networks, it is required to know how they solved these simpler tasks and especially what their thinking process was.

3.2.1 Classification

The classification task has been tackled very effectively using Convolutional Neural Networks (CNNs). As their names suggest, these networks are made of convolutional layers. These layers have been introduced to solve the inability of the multi-layer perceptron to take advantage of the structure existing in some large signals. Examples of large structured signals are images and sounds samples. In the case of images, the key idea behind this new layer is to leverage the prior knowledge that close pixels are typically correlated to one another to create neural networks that efficiently learn from these large spatially structured signals.

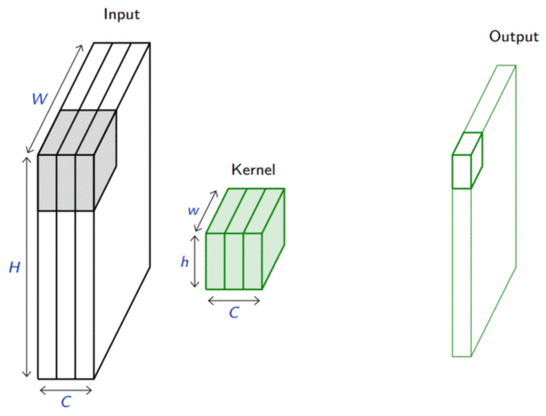


Figure 3.3: Visual representation of the convolutional layer principle (Fleuret, 2021): A kernel $\mathbf{k} \in \mathbb{R}^{C\times h\times w}$ slides across the input $\mathbf{x}$ along its width W and height H, such that at each possible overlapping of $\mathbf{k}$ and $\mathbf{x}$, an element wise product between their elements is computed, then the results are summed up and finally stored inside a 2D output tensor $\mathbf{o} \in \mathbb{R}^{(H-h+1)\times(W-w+1)}$. Note that the number of channels (C) is the same for the input and the kernel.

A convolutional layer implements a convolution operation on an input tensor (representing an image for instance). Figure 3.3 shows more precisely how it is performed in the case of a 3D tensor $\mathbf{x} \in \mathbb{R}^{C\times H\times W}$. In summary, a 3D tensor $\mathbf{k} \in \mathbb{R}^{C\times h\times w}$ (called *kernel*) slides across the input along its width W and height H, such that at each possible overlapping of $\mathbf{k}$ and $\mathbf{x}$, an element wise product between their elements is computed, then the results are summed up and finally stored inside an output tensor $\mathbf{o} \in \mathbb{R}^{(H-h+1)\times(W-w+1)}$. More formally, this $\mathbf{o}$ (also called the output feature map) is given by

$$\mathbf{o}_{j,i} = \mathbf{b}_{j,i} + \sum_{c=0}^{C-1}(\mathbf{x}_c \circledast \mathbf{k}_c)[j,i] = \mathbf{b}_{j,i} + \sum_{c=0}^{C-1}\sum_{n=0}^{h-1}\sum_{m=0}^{w-1}\mathbf{x}_{c,n+j,m+i}\mathbf{k}_{c,n,m} \tag{3.11}$$

where $\mathbf{k}$ and $\mathbf{b}$ are shared parameters to learn.

As mentioned earlier convolutional neural networks (CNNs) are composed of convolutional layers. However, they also contain other types of layers. Indeed, a convolutional layer is usually followed by a Rectified Linear Unit (ReLU) activation function, a pooling layer or some Fully Connected layers (FC).

The ReLU activation is described graphically in Figure 3.4 and is mathematically defined as follows:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.12}$$

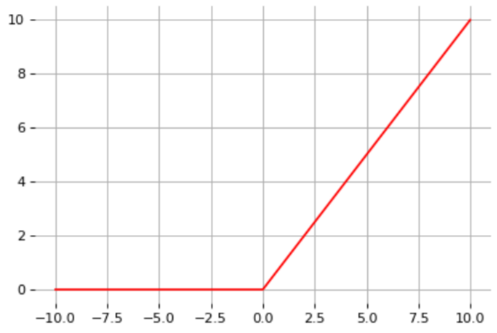


Figure 3.4: Visual representation of the ReLU(x) activation function (Louppe, 2021a): If $x \leq 0$, this function is equal to zero. Otherwise, it corresponds to the identity function.

ReLU (Glorot, Bordes, & Bengio, 2011) are mainly used jointly with a smart weight initialization to overcome the vanishing gradients phenomenon that occurs in deep neural networks.

In addition to exploiting the correlation between neighbor pixels inside spatially structured large-dimensional signals (such as images) to reduce the required computational power, a pooling layer can be used to decrease the dimensionality of the input signal while preserving its general structure.

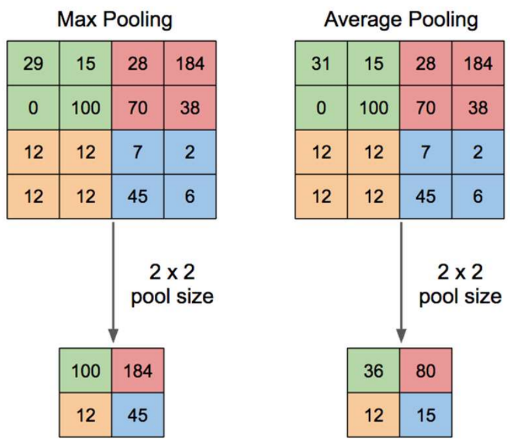


Figure 3.5: Illustration of the max and average pooling operation (Yani & Si.M.T.BudhiIrawan, 2019). Briefly, these operators consist of a window that slides over all regions in the input (according to a step size), computing a single output value for each traversed location.

As its name suggests, a pooling layer implements a pooling operation. In practice, two pooling operations are widely used: max and average pooling. Assuming a pool area of size $h \times w$ and an input tensor $\mathbf{x} \in \mathbb{R}^{C\times(rh)\times(sw)}$, the max and average pooling operators

$$\mathbf{o}_{c,j,i} = \max_{n<h,m<w} \mathbf{x}_{c,rj+n,si+m} \tag{3.13}$$

$$\mathbf{o}_{c,j,i} = \frac{1}{hw} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{x}_{c,rj+n,si+m} \tag{3.14}$$

respectively produce both an output tensor $\mathbf{o} \in \mathbb{R}^{C\times r\times s}$. Figure 3.5 also shows these two operations in a particular case where $h = w = r = s = 2$ and $C = 1$. One can notice that the mathematical expression of the pooling operation is very similar to the one of the convolution. Actually, this is not a coincidence: a convolution layer can be used to implement a pooling operation by applying a step size (called a stride) greater than one when moving the kernel across the input signal.

Finally, in the specific case of the classification task, the whole CNN is usually followed by a softmax activation layer

$$\text{softmax}(\mathbf{o})_{\text{i}} = \frac{\exp(\mathbf{o}_i)}{\sum_{j=1}^{U} \exp(\mathbf{o}_j)}, \quad \text{for } i = 1, \ldots, U. \tag{3.15}$$

which has the purpose of producing from the outputs $\mathbf{o}$ of the CNN a vector of U probability estimates $P(Y = i|\mathbf{o})$ (where U corresponds to the number of classes considered).

Historically, one of the first successful CNN was the AlexNet which won the 2012 ImageNet Large Scale Visual Recognition Challenge by outperforming with a large margin all the other competitors (ImageNet, 2012). This deep neural network is composed of 11 layers: the first 8 are convolutional and pooling layers and the remaining three are fully connected layers. The output of this last layer is a vector of 1000 features which is finally fed to a softmax activation function to produce the probability distribution over the 1000 classes of the ImageNet dataset (Krizhevsky, Sutskever, & Hinton, 2012).

3.2.2 Semantic segmentation

Contrary to the classification task, the aim of semantic segmentation is not to output one class (or a probability distribution over the classes) for the entire input image but a class for each pixel of the input image. Therefore, semantic segmentation does not differentiate instances: it classifies each pixel independently.

A first straightforward solution to solve this task could be to chop the whole input image into small patches and apply the CNN used to solve the classification tasks to each of the patches and thus get a class for each. However, this naive solution is extremely inefficient as it would require

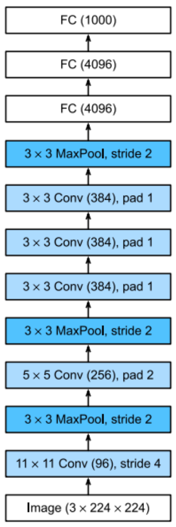


Figure 3.6: Illustration of the AlexNet deep neural network (Zhang et al., 2021) where FC, MaxPool and Conv stand for Fully Connected, Max pooling and Convolutional layer, respectively. This first very famous CNN is only composed of 11 layers: the first 8 are convolutional and pooling layers and the remaining three are fully connected layers.

for an input image of size $w \times h$ to chop the image into $w \times h$ patches of 1 pixel size and thus run the CNN $w \times h$ times.

A smarter idea is to not consider the classification of each pixel independently but as whole. It is what the state-of-the-art techniques in semantic segmentation do by using Fully Convolutional Networks (FCNs). FCNs are deep neural networks composed of convolutional layers with downsampling and upsampling operations inside the networks. More precisely, a FCN takes as input the whole image $C_1 \times H_1 \times W_1$ and outputs an $C_2 \times H_1 \times W_1$ image where the number of output channels C_2 is equal to the number of classes considered: each channel of the output corresponds to the probability distribution of each pixel belonging to a specific class associated to this channel. This stack of channels can be finally transformed into a $H_1 \times W_1$ semantic segmentation map using an $\arg\max$ operation on the C_2 channels to get the semantic segmentation mapping of the input image. In practice, as doing convolutional operation at the original input image resolution is too computationally expensive, FCN usually downsamples the original resolution using pooling layers and upsamples it to come back to the original resolution at the end of the network using upsampling layers. A very popular upsampling layer is the transposed convolution layer.

To understand this new layer, one has to notice that the convolution operation ($\circledast$) can be expressed as a matrix multiplication between the convolutional kernel $\mathbf{k}$ rearranged as a sparse Toeplitz circulant matrix $\mathbf{K}$ (called the convolution matrix) and the input vector $\mathbf{x}$ with the appropriate reshaping v, that is

$$\mathbf{h_1} = \mathbf{x} \circledast \mathbf{k} \iff v(\mathbf{h_1}) = \mathbf{K}v(\mathbf{x}) \tag{3.16}$$

More precisely, in the particular case of 1D convolution with a kernel of size 3 (and a stride equals to one), the convolution operation can be expressed as follows:

$$\mathbf{h_1} = \mathbf{x} \circledast \mathbf{k} \iff \begin{bmatrix} k_2x_1 + k_3x_2 \\ k_1x_1 + k_2x_2 + k_3x_3 \\ k_1x_2 + k_2x_3 + k_3x_4 \\ k_1x_3 + k_2x_4 \end{bmatrix} = \begin{bmatrix} k_1 & k_2 & k_3 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 & 0 \\ 0 & 0 & k_1 & k_2 & k_3 & 0 \\ 0 & 0 & 0 & k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} 0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ 0 \end{bmatrix} \tag{3.17}$$

In this example, as the stride is equal to one, the convolution operation results in an output with the same resolution as the input $\mathbf{x}$.

Knowing this property, one can now define mathematically the transposed convolution ($\circledast^T$) by the following equation

$$\mathbf{h_2} = \mathbf{x} \circledast^T \mathbf{k} \iff v(\mathbf{h_2}) = \mathbf{K}^T v(\mathbf{x}) \tag{3.18}$$

Similarly, in the particular case of 1D transposed convolution with a kernel of size 3, the transposed convolution is expressed by the following Equation:

$$\mathbf{h_2} = \mathbf{x} \circledast^T \mathbf{k} \iff \begin{bmatrix} k_1x_1 \\ k_2x_1 + k_1x_2 \\ k_3x_1 + k_2x_2 + k_1x_3 \\ k_3x_2 + k_2x_3 + k_1x_4 \\ k_3x_3 + k_2x_4 \\ k_3x_4 \end{bmatrix} = \begin{bmatrix} k_1 & 0 & 0 & 0 \\ k_2 & k_1 & 0 & 0 \\ k_3 & k_2 & k_1 & 0 \\ 0 & k_3 & k_2 & k_1 \\ 0 & 0 & k_3 & k_2 \\ 0 & 0 & 0 & k_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{3.19}$$

In this 1D example, one can see that the resolution (dimension) of the output of this new operation is indeed higher than one of the input $\mathbf{x}$: the transposed convolution is an upsampling operator.

Figure 3.7 illustrates the output of a particular FCN applied to a RGB picture of a dog. From this figure, one can clearly see that the first half of this network is downsampling the resolution while the second half is upsampling it to retrieve the original spatial resolution. Thanks to the downsampling operation, it is possible to use very deep neural networks. Indeed, it enables to work at a lower spatial resolution for many of the layers inside the network and therefore cuts off

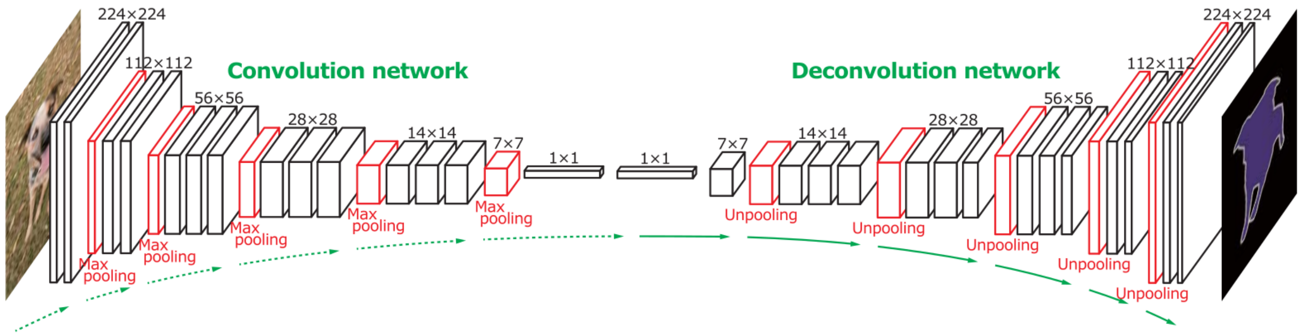


Figure 3.7: Example of Fully Convolutional neural Network (FCN) composed of a downsampling and an upsampling part (Noh et al., 2015): the FCN is fed with a RGB picture and outputs the corresponding segmentation map.

the required computational power. A big advantage of FCN is that the dimension of its output directly depends on the size of the picture sent in its input. Therefore, as this one is arbitrary, contrary to fully connected networks, FCN can handle any input dimensions.

3.2.3 Classification plus localization

Sometimes, in addition to classify an object present inside an image (classification task), one also wants to know where this object is located inside that image. This new task is called classification plus localization. More precisely, in addition of predicting a class, this task aims at drawing exactly one bounding box around the region where the object is located inside the input image. The third picture of Figure 3.2 shows an example of what the classification plus localization task is aiming for.

To tackle this problem, the community has reused the same machinery as the one learned from image classification and has treated the localization subtask as a regression problem: the basic architecture to tackle this problem is to first feed the input image through a CNN to produce a tractable vector of features and then to use these features as input for two sets of fully connected layers to output respectively the class of the object present in the picture (or a probability distribution over all the considered classes) and four scalars which uniquely define the bounding box around the classified object.

As this new network has two outputs, its training requires two losses. Furthermore, as the two outputs have not the same type (one is categorical and the other continuous), it requires two different loss functions: a classification loss and a regression loss. The classification loss allows to compute the error between the ground truth class and the predicted one. Commonly, the classification loss used is the cross entropy loss (also called log loss) defined in Equation 3.7. To lighten the notations, this loss can be rewritten as follows:

$$\ell_{cls}(\mathbf{p}, u) = -\log \mathbf{p}_u \tag{3.20}$$

where $\mathbf{p}_u$ is the probability value corresponding to the class u in the estimated probability vector output by the neural network.

Regarding the regression loss, a common one is the ℓ_2 loss that measures the dissimilarity (the error) between the predicted bounding box coordinates $\mathbf{t}$ and the actual ground truth coordinates of the bounding box $\mathbf{t}^*$. Formally, this regression loss can be defined as:

$$\ell_{reg}(\mathbf{t}, \mathbf{t}^*) = \sum_{i \in \{x,y,w,h\}} (t_i - t_i^*)^2 \tag{3.21}$$

where t_x^*, t_y^*, t_w^*, t_h^* denote the ground truth bounding box's center coordinate and its width and height. t_x is for the predicted bounding box (likewise for y, w and h).

Notice that, as one assumes a fully supervised setting with this new network, in addition to the ground truth class for each of the training images inside the dataset, one also needs to store the actual coordinates of the bounding box around the labeled object inside each of those images.

Usually, these two losses are combined into a multi-task loss ℓ_{multi} to jointly train the neural network for classification and bounding box regression:

$$\ell_{multi}(\mathbf{p}, u, \mathbf{t}, \mathbf{t}^*) = \ell_{cls}(\mathbf{p}, u) + \lambda\ \ell_{reg}(\mathbf{t}, \mathbf{t}^*) \tag{3.22}$$

with λ being an hyperparameter that controls the balance between the ℓ_{cls} and ℓ_{reg} losses.

Figure 3.8 describes schematically this new kind of deep neural network.

3.2.4 Object detection

As for the previous tasks, in object detection, one starts with some fixed set of classes that one wants to detect. However, the goal of object detection is to detect every instance of these classes present in the input image without knowing ahead of time how many of those are present. More precisely, each detection should be described by a bounding box around the instance detected and an associated class (category). Therefore, object detection is clearly the extension of the classification plus localization task to a multiple and varying number of instances in the input image.

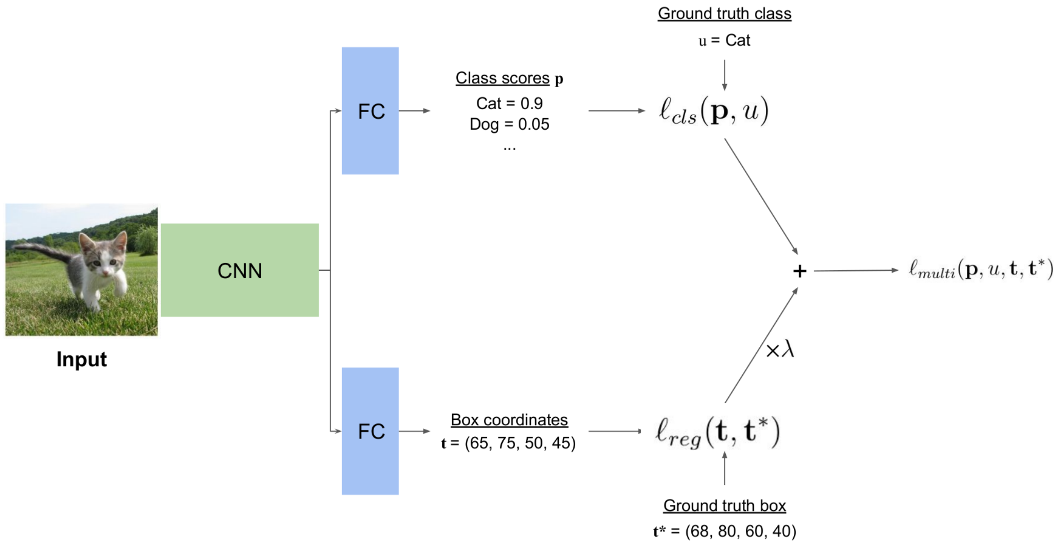


Figure 3.8: Example of an architecture to tackle the classification plus localization task with its associated multi-task loss computation. FC stands for Fully Connected layers, while CNN corresponds to Convolutional Neural Network (CNN). This network is trained jointly for the classification and bounding box regression thanks to the used of the multi-task loss ℓ_{multi}. This loss is obtained by combining the regression loss ℓ_{reg} and the categorical loss ℓ_{cls}.

As one does not know in advance the number of instances, their aspect ratios and their scales in the input image, using a sliding window approach is intractable. Indeed, a sliding window approach is equivalent to the process of applying a CNN to many different crops of the input image. Through this process, the CNN could theoretically classify each of theses crops as a specific object class or as background. However, one does not know how many crops to take and what should be their size. Thus, to work properly, this method requires to apply the CNN to a large amount of crops at various scales, which is extremely computationally expensive.

To solve this computational efficiency issue the community has created two types of deep neural networks: the one-stage and two-stage neural networks.

One-stage architectures

The one-stage architecture consists in treating the object detection problem as a large regression problem to make all the predictions all at once (bounding box coordinates and classes) with a large CNN. For example, a pioneer neural network using this approach is the You Only Look Once (YOLO) object detector (Redmon, Divvala, Girshick, & Farhadi, 2016). In summary, it divides the input image into a $S \times S$ grid and for each grid cell predicts B bounding boxes with their associated confidence score, and U class probabilities. All these predictions are output all at once

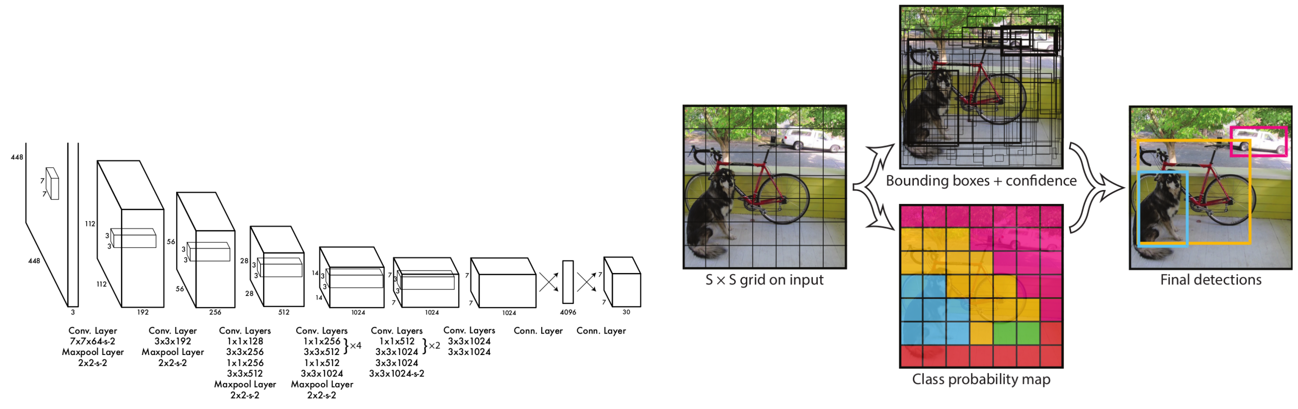

Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer
S × S grid on input
Bounding boxes + confidence
Class probability map
Final detections

Figure 3.9: Architecture of YOLO (left graph): it is composed of 24 convolutionnal layers followed by 2 fully connected layers. YOLO system overview (right graph): it divides the input image into a $S \times S$ grid and for each grid cell predicts thanks to the neural network B bounding boxes with their associated confidence score, and U class probabilities (Redmon et al., 2016).

as a $S \times S \times (B * 5 + U)$ tensor. Figure 3.9 shows the neural network architecture and the model of YOLO.

The main advantage of the one-stage detectors is their fast inference time. However, their accuracies usually do not compete with the two-stage detectors.

Two-stage architectures

The main family of two-stage detectors relies on region proposals. A region proposal is a part of the image which is likely to contain instances. Back in the day, these region proposals were computed using more traditional signal processing approaches: for instance, by exploiting edges (Zitnick & Dollár, 2014). A popular method was Selective Search (Uijlings, Van De Sande, Gevers, & Smeulders, 2013) which gives approximately two thousands region proposals in a few seconds on a single CPU. Figure 3.10 shows an example of region proposals obtained using such computer vision techniques.

Thanks to these region proposals, the use of CNN in object detection becomes possible. Indeed, applying the CNN only to these finite and well defined regions instead of all possible locations and scales allows to make it computationally tractable. These ideas have been concretely put together and implemented in 2014 through a complete deep neural network called R-CNN (Girshick, Donahue, Darrell, & Malik, 2014). The authors of this paper called it like that because their network combines region proposals with a CNN.

In summary, R-CNN works in four steps. Firstly, a region proposal algorithm is fed with the

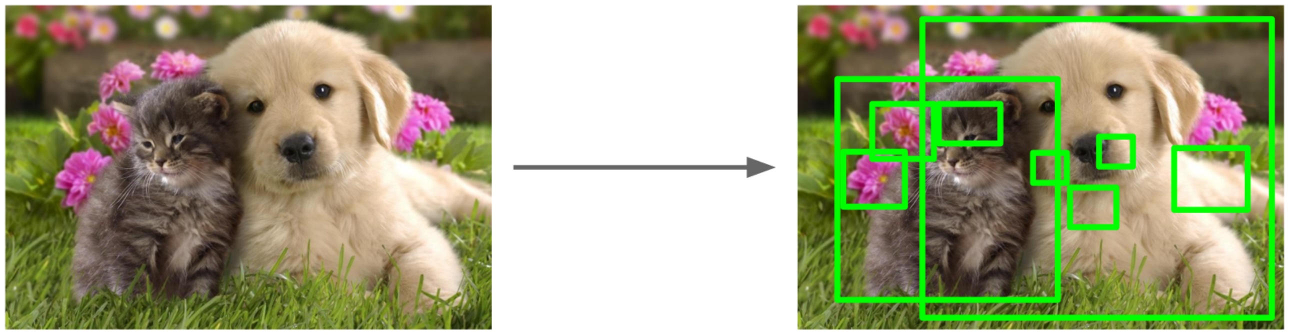


Figure 3.10: Example of region proposals (green boxes) obtained using "classical" computer vision techniques (Li et al., 2017).

input image and produces around two thousand region proposals. Secondly, these regions are warped into fixed size images to be fed into the CNN. Thirdly, the CNN extracts fixed-length feature vectors from each of these warped regions. And finally, for each of these vectors, class-specific linear Support Vector Machines (SVMs) (Noble, 2006) give the associated predicted class score and a class-specific regressor module outputs the corresponding bounding box coordinates. Figure 3.11 summarizes all these steps.

R-CNN achieved the best results on the object detection Dataset VOC2012 (Pascal2, 2012) in 2014. However, this network has several important drawbacks. The main one is that it is still very computationally expensive due to the independent application of the CNN to each of the two thousands region proposals: the training and evaluation of this model is very slow (several days and more than thirty seconds respectively). Moreover, the region proposal locations and qualities depend on the region proposal algorithm used: this step is not learned and therefore not optimized during training. This is why the authors of R-CNN created a first improved version called Fast R-CNN (Girshick, 2015) in 2015.

With Fast R-CNN, instead of processing each region proposal independently, the first step is to pass the whole input image through a FCN to obtain a feature map of the input image. Then, the region proposals (also called regions of interests) computed beforehand with the input image are cropped from the feature map: it allows to do the heavy convolution computation only once instead of two thousands times. The next step of Fast R-CNN is to reshape the Regions of Interest (RoIs) cropped from the feature map to be able to feed them afterwards into fully connected layers (which expect a specific input size). This operation is done (in a differential manner) using a layer called RoIPool. Finally, for each of these reshaped crops, fully connected layers followed by a categorical classifier and a bounding box regressor process the extracted features. The categorical classifier outputs a discrete probability distribution, $\mathbf{p} = (p_0, ..., p_U)$ over the $U + 1$ classes considered (the additional class corresponds to the background class), whereas the bounding box regressor outputs the bounding box regression offsets (associated to the RoI being processed), $\mathbf{b^u} = (b_x^u, b_y^k, b_w^u, b_h^u)$, for each of the U classes. A complete visual overview of Fast R-CNN is shown in Figure 3.12.

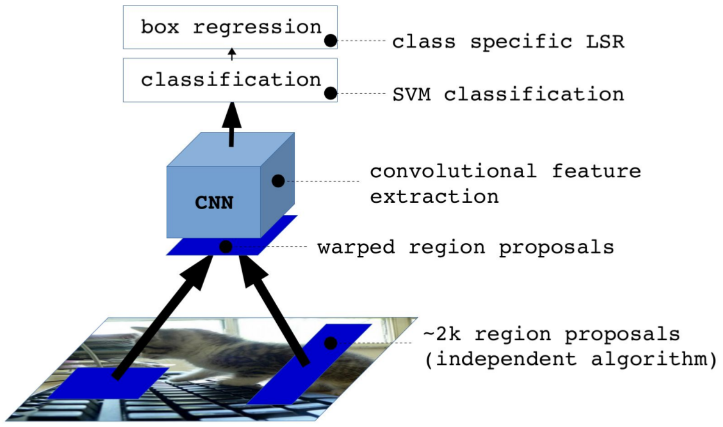


Figure 3.11: R-CNN system overview (Kaiming et al., 2017). Firstly, an independent algorithm extracts from the input image around 2000 region proposals. Secondly, these proposals are warped into fixed size regions. Thirdly, a Convolutional Neural Network (CNN) computes features for each warped proposal. Fourthly, a class-specific Support Vector Machine (SVM) classifies each region. Finally, for each region, a class-specific bounding box regressor predicts a bounding box by performing a Least Squares Regression (LSR) on the features computed by the CNN.

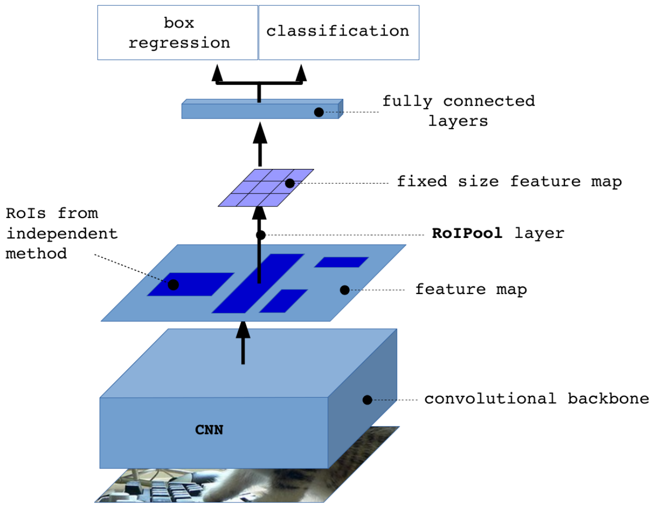


Figure 3.12: Fast R-CNN system overview (Kaiming et al., 2017). A region proposal algorithm computes Regions of Interest (RoIs) from the input image. Then, the whole input image is passed through a Fully Convolutional neural Network (called backbone) to obtain a feature map of the input image. Following this computation, the RoIs are cropped from this feature map and reshaped to a fixed feature map to be able to feed them afterwards into fully connected layers. This operation is done (in a differential manner) using a layer called RoIPool. Finally, for each of the reshaped crops, fully connected layers followed by a categorical classifier and a bounding box regressor output the class confidence scores and the corresponding bounding box regression offsets.

Thanks to this new architecture, Fast R-CNN achieves fast training and testing time while improving the quality of the detections. More precisely, using VGG16 as CNN, Fast R-CNN evaluates images 213 times faster than R-CNN and reduces the training time by 9 (Girshick, 2015). Actually, Fast R-CNN is so fast that during evaluation, the region proposal algorithm dominates the overall run-time. In other words, it is the non-deep learning part of the network that bottlenecks the entire method. Moreover, as previously mentioned in R-CNN, the quality of the region proposals is completely tied up by the region proposal algorithm.

Acknowledging this main drawback, the authors of Fast R-CNN reacted few months later with a new version called Faster R-CNN (Ren, He, Girshick, & Sun, 2015). This new version replaces the region proposal algorithm with a deep neural network called Region Proposal Network (RPN). Figure 3.13 describes its working principle.

To be computationally efficient, this network shares a set of convolution layers with the FCN used to extract features from the input image. Concretely, the region proposals are generated by sliding a small window over the feature map output by the last shared layer. The output of this windowing operation is then mapped into a fix feature vector. After that, this feature vector is fed to two sets of fully connected layers to predict k bounding box coordinates and their $2k$ scores that give the probability for each proposal of containing an object or the background. The first set of fully connected layers predicting the bounding boxes has k independent heads for the k different considered anchors such that each head is responsible for one aspect ratio and one scale (they do not share their weights). Anchors are predefined (rectangle) bounding boxes centered at the current sliding window location and are used as initial object location guesses. The right part of Figure 3.13 shows some of these k anchors. Doing so, thanks to the anchors, RPN is able to predict bounding boxes of various sizes even though the features cropped from the feature map are of fixed spatial size.

In practice, the RPN is implemented as a fully convolutional network by using an $n \times n$ convolutional layer to emulate the sliding window process, and the two sets of fully connected layers are both built by using convolutional layers.

Using shared layers of the FCN (used to extract features from the input image) and implementing the RPN in a fully convolutional way make this approach translation invariant, both in terms of the anchors and the functions that compute the proposals relative to the anchors: if an object is translated in the input image, the corresponding proposal will be translated exactly in the same way and the same function is able to predict the proposal in either location.

In order to be able to train the RPN, a binary class label (i.e. being an object or not) is assigned to each anchor. This assignement depends on the Intersection over Union (IoU) between each anchor and the ground truth bounding boxes. IoU is a standard performance metric in object detection and is defined as follows:

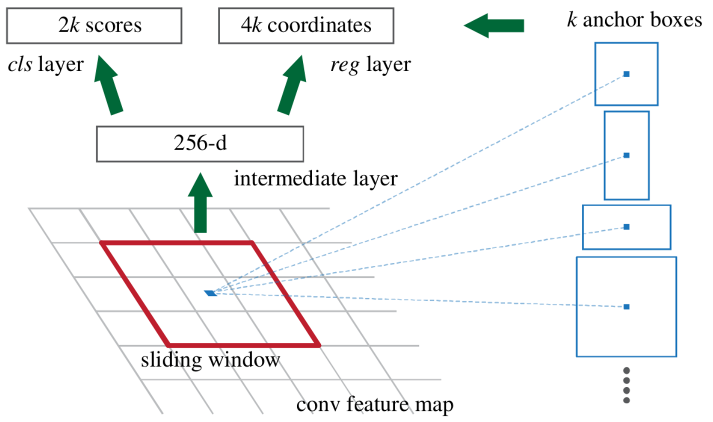


Figure 3.13: Illustration of the Region proposal Network (RPN) (Ren et al., 2015). Region proposals are generated by sliding a small window (emulated by an $n \times n$ convolutional layer) over the feature map output by the last shared layer with the FCN used to extract features from the input image. The output of this sliding window is then mapped into a fix feature vector. After that, this feature vector is fed to two sets of fully connected layers (implemented in practice by two convolutional layers) to predict k bounding box coordinates and their $2k$ scores that give the probability for each proposal of containing an object or the background.

$$IoU(B, \hat{B}) = \frac{\text{area}(B \cap \hat{B})}{\text{area}(B \cup \hat{B})} \tag{3.23}$$

where $\hat{B}$ and B are the predicted and ground truth bounding boxes, respectively. Figure 3.14 expresses graphically the two terms of this equation.

Coming back to the RPN, if the anchor has the highest IoU or an IoU greater than 0.7 with any ground truth bounding box, a positive label is assigned to this specific anchor, whereas if the IoU ratio is lower than 0.3 for all ground truths, a negative label is assigned to this anchor. Anchors which are neither positive nor negative are just not considered and do not contribute to the loss computation. Hence, Faster R-CNN minimizes the following multitask loss function to train the RPN:

$$\ell(\{p_i\}, \{t_i\})_{RPN} = \frac{1}{N_{cls}} \sum_i \ell_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* \ell_{reg}(t_i, t_i^*) \tag{3.24}$$

where i is the index of an anchor. p_i and p_i^* are respectively the predicted probability of being an object for the anchor i and the ground truth label. This ground truth label is equal to 1 if the anchor is positive and is equal to 0 if the anchor is negative. t_i and t_i^* correspond to the vector containing the 4 parametrized coordinates of the predicted bounding box and that of the ground truth bounding box associated to a positive anchor, respectively. More precisely, the following

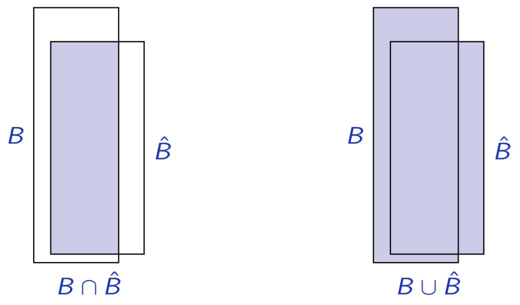


Figure 3.14: Visualization of the intersection (left graph) and the union (rigth graph) between the ground truth bounding box and the predicted one, respectively. (Fleuret, 2020).

parameterization is applied:

$$\begin{array}{ll} t_x = (x - x_a)/w_a, & t_y = (y - y_a)/h_a \\ t_w = \log(w/w_a), & t_h = \log(h/h_a) \\ t^*_x = (x^* - x_a)/w_a, & t^*_y = (y^* - y_a)/h_a \\ t^*_w = \log(w^*/w_a), & t^*_h = \log(h^*/h_a) \end{array} \tag{3.25}$$

where x, y, w an h correspond to the box's center coordinates, its width and height. x, x_a, x^* are for the predicted box, anchor box and ground truth box respectively (likewise for y, w and h).

The regression loss ℓ_{reg} is a smooth ℓ_1, that is:

$$\ell_{reg}(t_i, t^*_i) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{\ell_1}(t_i - t^*_i) \tag{3.26}$$

in which

$$\text{smooth}_{\ell_1}(z) = \begin{cases} 0.5z^2 & \text{if } |z| < 1 \\ |z| - 0.5 & \text{otherwise} \end{cases} \tag{3.27}$$

is a loss less sensitive to outliers than the ℓ_2 used in R-CNN.

The classification loss ℓ_{cls} is a simple cross entropy loss over the object and non-object classes. Finally, N_{cls} and N_{reg} are simply two normalizing terms and λ is a hyperparameter which balances the importance of the regression and classification loss in the global multitask loss computation.

By merging RPN and Fast R-CNN into a single network, Faster R-CNN is a complete deep neural network that can be trained end-to-end. Figure 3.15 describes visually this complete two-stage detector.

For each region of interest (RoI), the bounding box regressor and the classifier at the end of the Faster R-CNN are trained jointly thanks to the following multi-task loss:

$$\ell_{faster}(\mathbf{q}, u, \mathbf{b}^u, \mathbf{b}^*) = \ell_{cls}(\mathbf{q}, u) + \lambda(u)\ \ell_{box}(\mathbf{b}^u, \mathbf{b}^*) \tag{3.28}$$

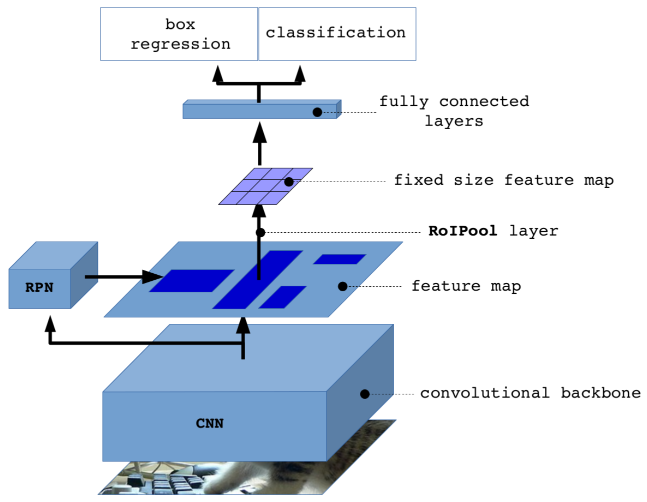


Figure 3.15: Faster R-CNN overview (Kaiming et al., 2017). The whole input image is passed through a FCN (called backbone) to obtain a feature map of the input image. Then, a RPN computes region proposals from the feature map. Following this computation, the proposals are cropped from this feature map and reshaped to a fixed feature map to be able to feed them afterwards into fully connected layers. This operation is done using a layer called RoIPool. Finally, for each of the reshaped crops, fully connected layers followed by a classifier and a bounding box regressor output class confidence scores and the corresponding bounding box coordinates.

with λ being a function equal to a hyperparameter that controls the balance between the ℓ_{cls} and ℓ_{box} losses if u does not correspond to the background class. If the value of u corresponds to the background class, $\lambda = 0$. $\mathbf{q}$ corresponds to the estimated probability vector output by the network classifier and $\mathbf{b}^u$ represents the predicted bounding box offset coordinates (relative to the considered RoI) for the ground truth class u. $\mathbf{b}^*$ is the true bounding box regression target: it corresponds to the ground truth bounding box coordinates specified as offsets relative to the RoI considered.

In practice, the bounding box regressor and the classifier are implemented by two set of fully connected layers. More precisely, the set corresponding to the bounding box regressor is composed of U output heads (one for each class), which output each four offset bounding box coordinates relative to the considered RoI. When Faster R-CNN is not being trained, the predicted bounding box coordinates relative to the input image are computed by combining the four offset coordinates corresponding to the most probable class and the coordinates of the considered RoI.

The ℓ_{cls} and ℓ_{box} are identical to the losses previously defined in Equation 3.20 and 3.26, respectively.

As the RPN shares some convolutional layers with the Fast R-CNN part of the Faster R-CNN, they must not be trained independently. The solution used in practice consists in performing an alternating training: first the RPN is trained, and the region proposals obtained are directly used to train Fast R-CNN. The FCN network used by Fast R-CNN is then used to initialize RPN, and the process is repeated many times.

In summary, the RPN part of Faster R-CNN enables it to learn by itself how to quickly produce highly qualitative regions of interest from the input image. Thanks to these high quality RoIs, the Fast R-CNN part of the complete network can consider and process much less RoIs while ensuring accurate object detections: in 2016, Faster R-CNN achieved state-of-the-art object detection performance on PASCAL VOC 2007, 2012 and MS COCO datasets with only 300 regions of interest considered per image.

3.2.5 Instance segmentation

As described in the introduction, the goal of this master's thesis is to obtain a high instance segmentation performance by building and training a deep neural network.

Concretely, the instance segmentation task can be seen as the merging of the object detection and the semantic segmentation tasks: given an input image, we want to predict the location and the class of all the instances (among a predefined list of classes) inside this image (object detection task) but rather than just predicting a bounding box to locate each of the detected instances, we want to predict a semantic segmentation mask for each of these instances (semantic segmentation task). In other words, we want to be able to know which pixels of the input image correspond to which detected instances. The rightmost picture in Figure 3.2 visually shows the expected output (mapped onto the input image) of an instance segmentation method.

Similarly, to the other tasks, there exists many different networks to tackle this hybrid problem. Nonetheless, as mentioned at the beginning of this chapter, in this master's thesis, Mask R-CNN (He, Gkioxari, Dollár, & Girshick, 2017) is the one chosen to tackle it. The reasons of this choice are explained in Section 6.1. This deep neural network extends Faster R-CNN by adding a branch for predicting object mask in parallel with the existing bounding box detection branch. Figure 3.16 shows the resulting network.

More precisely, this new branch (called mask branch) is implemented using a FCN and outputs a $D \times D \times U$ mask from each RoI, which encodes U masks of resolution $D \times D$: one for each of the U classes considered. A detailed example of a Mask R-CNN head architecture with $D = 28$ is shown in Figure 3.17.

Moreover, the authors of Mask R-CNN replaced the RoIPool layer by a new RoIAlign layer that fixes the misalignment issue present in the RoIPool layer. Indeed, in Faster R-CNN, the RoIPool layer allows to extract a small fixed size feature map (e.g., $N \times N$) from each RoI output by the RPN. As the RoI coordinates predicted by the RPN are floating numbers, they do not match the discrete granularity of the $MN \times MN$ feature map given by the last shared layer of

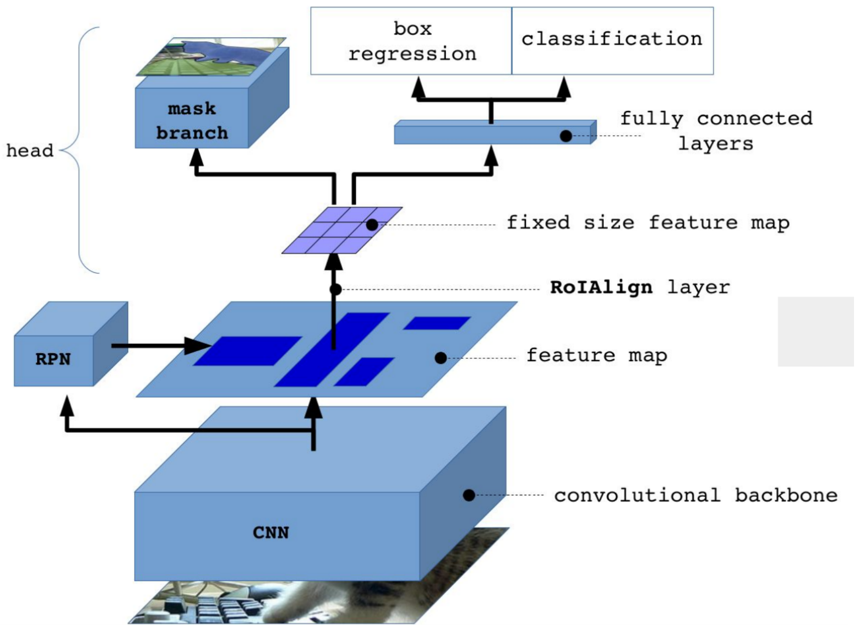


Figure 3.16: Mask R-CNN overview (Kaiming et al., 2017). This network is identical to Faster R-CNN (Figure 3.15) except that it also outputs a semantic mask for each RoI thanks to a mask branch (implemented by a FCN) that directly takes the fixed size feature map in input.

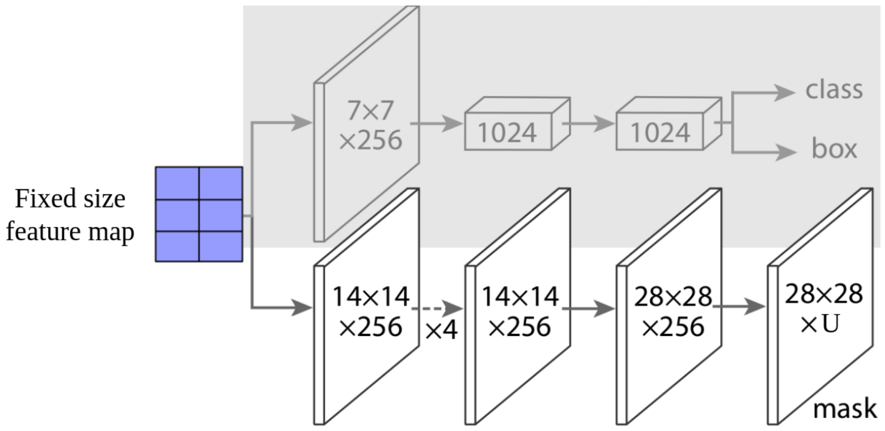


Figure 3.17: Example of a Mask R-CNN head architecture (Kaiming et al., 2017). A mask branch is added to the two existing heads in Faster R-CNN. U is the number of classes considered. Numbers correspond to spatial resolution and channels. Arrows denote either convolutional, transposed convolution or fully connected layers and can be inferred from context. '$\times 4$' denotes a stack of four consecutive convolutional layers. All convolution layers have a kernel of 3×3 except the output convolutional layer which has a 1×1 kernel. Transposed convolution layer has a stride of 2 with a 2×2 kernel. ReLU is used in hidden layers.

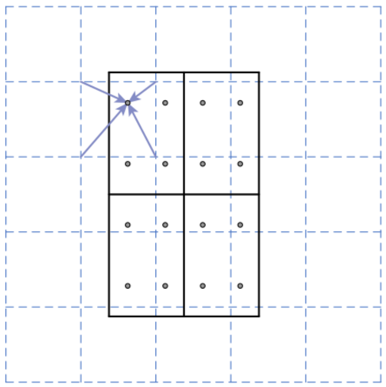


Figure 3.18: Illustration of the bilinear interpolation conducted by the RoIAlign layer. The solid back lines represent an RoI (with 2×2 bins in this example), the dashed blue grid a feature map, and the black dots the 4 sampling points in each bin. The value of each point is computed by performing a bilinear interpolation with the four nearby grid points on the feature map (highlighted by the blue arrows) (He et al., 2017).

the convolutional backbone. Therefore, a quantization operation is performed on each floating-number x by computing $[x/M]$, where M is a feature map stride and [.] is the rounding operator. This quantized RoI is then subdivided into spatial bins that are themselves quantized. Finally, the feature values covered by each bin are aggregated using max pooling. All these quantizations directly introduce misalignments between the RoI and the extracted features such that if not removed, these misalignments will directly prevent a pixel-accurate mask prediction. This is why this layer has been replaced by a RoIAlign layer which avoids doing any quantization. The $[x/M]$ is replaced by a x/M, and a bilinear interpolation is used to compute the values of the features at 4 regularly sampled locations in each RoI bin. Then, for each bin, the 4 associated values are aggregated by performing a max or average pooling. Figure 3.18 visually describes the bilinear interpolation conducted by the RoIAlign layer.

During training, the following multi-task loss is computed on each sampled RoI:

$$\ell_{head} = \ell_{faster} + \ell_{mask} \tag{3.29}$$

where ℓ_{faster} is the loss defined for Faster R-CNN in Equation 3.28. ℓ_{mask} is the average binary cross entropy loss applied on the u-th mask (where u is the integer number corresponding to the ground truth class associated to the RoI being processed) on which a per-pixel sigmoid has been applied beforehand. Formally, ℓ_{mask} can be defined as follows:

$$\ell_{mask} = -\frac{1}{D^2} \sum_{i \leq i,j \leq D} \left(y_{ij} \log \hat{y}_{ij}^{u} + (1 - y_{ij}) \log(1 - \hat{y}_{ij}^{u}) \right) \tag{3.30}$$

where y_{ij} is the binary ground truth mask associated to the RoI being processed and $\hat{y}_{ij}^{u}$ is the u-th $D \times D$ mask output by the mask head of Mask R-CNN on which three operations have been performed: first, a per-pixel sigmoid is applied to transform the values of the mask into estimated probabilities. Then, all these values are binarized at a threshold of 0.5 to produce a binary mask. Finally, this $D \times D$ binary mask is resized using the predicted bounding box position corresponding to the location of the mask in the input image such that y_{ij} and $\hat{y}_{ij}^{u}$ have the same dimensions.

Contrary to what was initially done with Faster R-CNN in 2015, the RPN is completely merged into Mask R-CNN such that they form a single deep neural network. Thanks to that, an end-to-end training can be applied to train the overall network at each iteration of the optimizer. The forward pass first produces RoIs which are treated as fixed, precomputed RoIs. Then, the backward pass propagates the signals as in a classical deep neural network, except that for the shared layers the signals sent are computed based on a combination of the RPN loss ℓ_{RPN} (defined in Equation 3.24) and the Mask R-CNN head loss ℓ_{head} (defined in Equation 3.29).

At test time, only the u-th mask output by the mask branch is used, where u is the integer number associated the class having the highest confidence score with the classification branch. It is resized to the RoI size and binarized at a threshold of 0.5.

In 2017, Mask R-CNN achieved state-of-the-art results in instance segmentation by outperforming by a large margin the previous state-of-the-art networks (He et al., 2017).

Chapter 4

Related work

Outline
This chapter briefly describes the current state-of-the-art deep neural networks in instance segmentation and highlights the originality of this master's thesis subject.

Since several years, the deep learning community is actively trying to solve the semantic segmentation task with classical RGB images. Indeed, as shown in Figure 4.1, a new method is created almost every year and beat all previous ones in terms of performance (PapersWithCode, 2021).

The current state-of-the-art is a Cascade Mask R-CNN with EfficientNet-B7 as backbone and NAS-FPN as the feature pyramid. In summary, a Cascade Mask R-CNN is a multi-stage extension of the Mask R-CNN architecture (Cai & Vasconcelos, 2019). EfficientNet-B7 was the state-of-the-art CNN in 2019: it achieved state-of-the-art performance in the classification task (Tan & Le, 2019). This CNN architecture has been built by upscaling a baseline model (EfficientNet-B0) developed using a Neural Architecture Search (NAS) (Tan et al., 2019). Similarly, the NAS-FPN pyramid network has also been determined using NAS (Ghiasi, Lin, & Le, 2019). Very briefly, a Feature Pyramid Network (FPN) allows to do multi-scale feature fusion from the features produced by a CNN (more explanations regarding FPN are given in Section 6.3.3).

More recently, exploiting hyper-spectral images has gained more and more attention thanks to the ability of deep learning to handle very high-dimensional inputs. The most current application is the processing of satellite or medical images. In satellite imaging, the sensors usually capture a broad range of the light spectrum for each spatial position, habitually about 200 different reflectance intensities. All these intensities can be placed into a 3D volume to embed intuitively the spatial information. Using this large 3D volume, several methods have been proposed in the literature. For instance, in the paper "Segmenting Hyperspectral Images Using Spectral-Spatial Convolutional Neural Networks With Training-Time Data Augmentation", the authors have man-

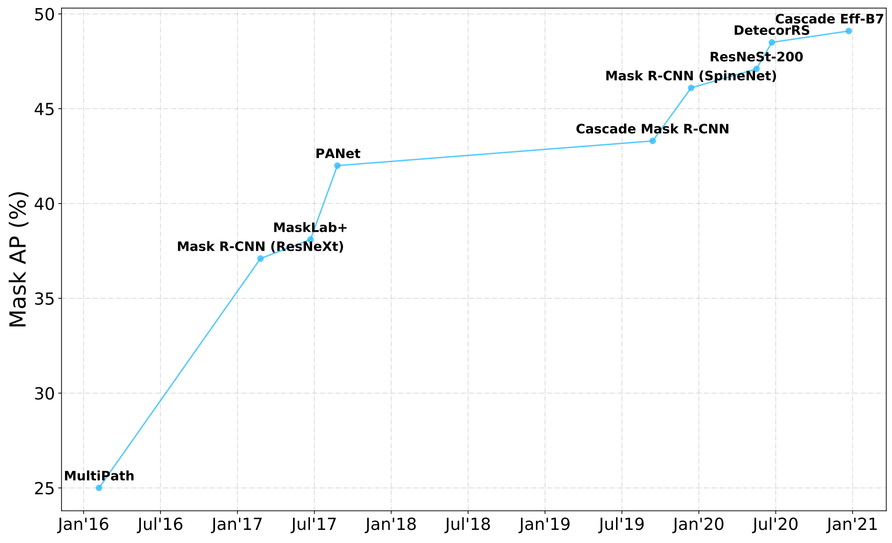


Figure 4.1: Highest mask Average Precision (see Section 7.1.2 for a complete description of this metric) since January 2016 obtained in the task of instance segmentation on the COCO test-dev dataset (Lin et al., 2014). This graph has been built using the data collected in the PapersWithCode website (PapersWithCode, 2021). The terms inside parentheses correspond to the deep neural networks used as backbones. One can observe that a lot of progress has been achieved since 2016 and that the discovery of new state-of-the-art models keeps accelerating.

aged to classify in real time each spatial location of hyper-spectral satellite images using 3D convolutional layers (Nalepa, Tulczyjew, Myller, & Kawulok, 2019).

However, the Pickit multi-feature images are actually neither spectral nor RGB images. Indeed, thanks to the three sensors, the Pickit multi-feature images embedded depth, spectral and atomic density data for each spatial location through 11 channels. Nowadays, aside from RGB and large hyper-spectral images, there are very few deep learning researches about multi-feature images containing completely different spatially correlated data and having more than 3 channels.

Therefore, we can clearly state that this master's thesis is a completely original work and that in addition to contributing to the Multipick project, it directly contributes to the deep learning community.

Chapter 5

Dataset

Outline
In this chapter, we describe the creation of the instance segmentation dataset from the multi-feature images acquired on the conveyor belt (Section 5.1) and how its size has been augmented thanks to several data augmentation techniques (Section 5.2).

5.1 Dataset creation

The first concrete step of this master's thesis is to create an instance segmentation dataset. Indeed, to train a supervised deep neural network, it is required to have the ground truth masks, bounding boxes, and classes associated with all inputs.

As mentioned in Chapter 1, the inputs of the deep neural network are multi-feature images. More precisely, each of those inputs can be seen as a 3D tensor $\boldsymbol{x} \in \mathbb{R}^{C \times H \times W}$, where C (Channel), H (Height), and W (Width) are respectively equal to 11, 1000 and 1096. In fact, each input can be seen as a stack of 11 2D images, where each 2D image contains a different type of features. Table 5.1 precisely describes the type of features associated with each channel (2D image) of the input. Figure 5.1 shows visually three channels of one multi-feature image drawn from the dataset. The whole 11 channels of this multi-feature image are displayed in Appendix II.

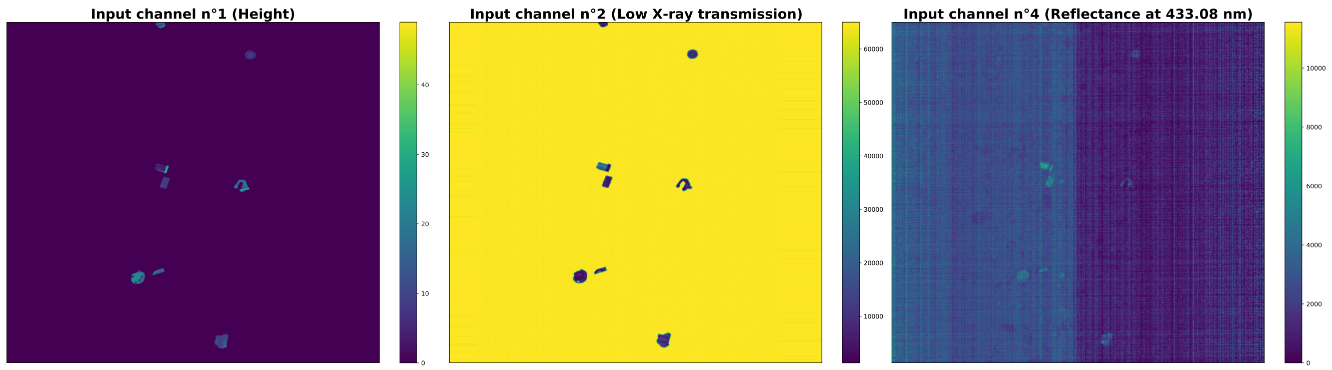


Figure 5.1: Overview of the first, second, and fourth channels of one dataset sample: those 3 channels contain completely different features. The first one gives the height of the sample in millimeters, while the second one encodes the low energy X-Ray transmission from 65535 (full transmission) to 0 (full absorption). The last depicted channel corresponds to the percentage of the energy which is reflected (called reflectance) at a bandwidth wavelength close to 433 nm: 100 % reflectance is associated to the number 65535 and 0 % to 0.

Channel n°	Feature type
1	Height
2	Low energy X-ray (40-80 keV) transmission
3	High energy X-ray (80-110 keV) transmission
4	Reflectance at $\lambda = 433$ nm
5	Reflectance at $\lambda = 497$ nm
6	Reflectance at $\lambda = 561$ nm
7	Reflectance at $\lambda = 626$ nm
8	Reflectance at $\lambda = 691$ nm
9	Reflectance at $\lambda = 778$ nm
10	Reflectance at $\lambda = 866$ nm
11	Reflectance at $\lambda = 955$ nm

Table 5.1: Description of the input in terms of its 11 channels and the associated features (λ corresponds to the wavelength): each channel contains a different type of features.

As the first channel of each input gives the heights in millimeters of each sample above the conveyor belt, we can use it to build the ground truth bounding boxes and masks. More precisely, in this master's thesis, the ground truths have been computed for each input by applying the following procedure:

1. We extract the first channel from the 11 channels forming the input.
2. We transform the resulting image into a binary one by assigning 1 to every pixel having a

height greater than 1 millimeter and 0 otherwise. This threshold enables to not consider the roughness of the conveyor belt as scraps.

3. We apply the Difference of Gaussian (DoG) method using the *blob_dog* function of the Skimage library (Scikit-image, 2021) to find blobs in the binary image.
4. Finally, for each detected blob, we use its approximate position and size obtained with DoG to compute a precise bounding box and mask.

To ease the creation of the ground truth classes, the samples have been sent to the conveyor belt class by class such that each multi-feature image acquired contains scraps of only one specific class. Thanks to that, the generation of the ground truth classes is straightforward.

To retrieve easily all the ground truths corresponding to a specific multi-feature image, this work followed the COCO format applied in the COCO challenge (Yin, Tsung-Yi, Matteo, & Alexander, 2020) with a slight modification regarding the encoding of the masks.

In this format, all the ground truths (also called annotations) are stored into a JSON file (*Introducing JSON*, n.d.). This file is defined as a collection of "*info*", "*categories*", "images", "*annotations*" and "*licenses*" objects. That is:

```
{
"info": info,
"licenses": [license],
"categories": [category],
"images": [image],
"annotations": [annotation],
}
```

The "*info*" object is composed of high level data about the dataset:

```
info{
    "year": int,
    "version": str,
    "description": str,
    "contributor": str,
    "url": str,
    "date_created": datetime,
    }
```

The "*licenses*" object list contains the licenses (if any) that are applied to the images present inside the dataset:

```
license{
        "id": int,
        "name": str,
        "url": str,
        }
```

The "*categories*" object list encodes all the classes (called categories) considered inside the dataset. Its general form can be defined as follows:

```
categories[{
            "id": int,
            "name": str,
            "supercategory": str,
           }]
```

"*images*" object list contains all the image names considered in the dataset with some relevant information about each one. This object has the following data structure (*coco_url*, *flickr_url*, and *date_captured* are just for reference, they are not necessarily required):

```
image{
    "id": int,
    "width": int,
    "height": int,
    "file_name": str,
    "license": int,
    "flickr_url": str,
    "coco_url": str,
    "date_captured": datetime,
    }
```

Finally, the object list "*annotations*" is composed of all the annotations (ground truths) associated with the images present inside the dataset. Each object "annotation" (described below) is composed of a series of fields, including the category id (which corresponds to the class id), the image id of the image corresponding to this annotation and the segmentation mask of the instance. The format of this segmentation depends on whether the instance represents a single object ($iscrowd = 0$) or a collection of objects ($iscrowd = 1$). In the context of this master's thesis, each instance represents a single object. Therefore, the segmentation mask of each object instance should be encoded with a list of polygon vertices. However, in practice, this thesis deviated from this standard by providing the pixel coordinates of all non-zero elements present inside the segmentation mask. By doing this, the mask encoding and decoding are greatly facilitated. Regarding the bounding box, it is provided for each annotation through four scalars: the top-left coordinates of the mask, its width, and its height.

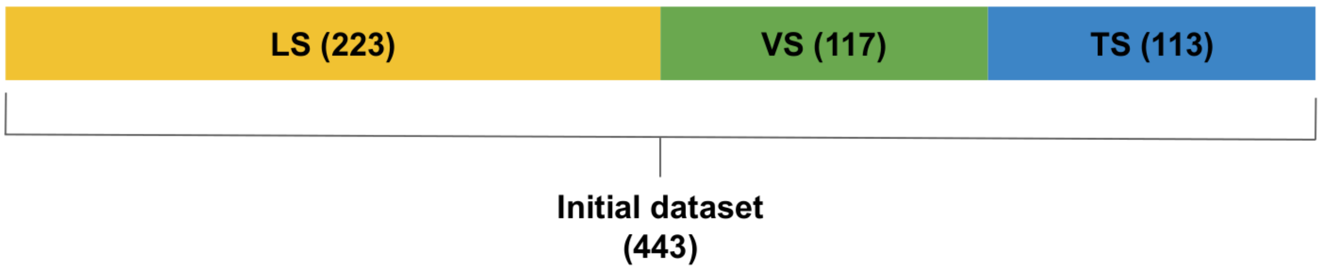


Figure 5.2: Illustration of the division of the initial dataset into 3 parts: the Learning Set (LS), the Validation Set (VS) and the Test Set (TS). The scalars in parentheses correspond to the number of multi-feature images contained inside each set.

```
annotation{
              "id": int,
              "image_id": int,
              "category_id": int,
              "segmentation": RLE or [polygon],
              "area": float,
              "bbox": [x,y,width,height],
              "iscrowd": 0 or 1,
              }
```

Creating the dataset using this format enabled us to use the COCO API (*COCO API*, 2021) to access and manipulate the annotations and the images very easily.

In practice, to provide a proper performance estimation of the final deep neural network, three distinct sets from the initial dataset (acquired on the conveyor belt) have been created. In total, this dataset contains 453 multi-feature images. More precisely, this initial dataset has been divided into 3 sub-datasets:

1. A Learning Set (LS) (223 images).
2. A Validation Set (VS) (117 images).
3. A Test Set (TS) (113 images)

This division is shown graphically in Figure 5.2 and the resulting class distribution within each set is described in Table 5.2. Building these 3 sub-datasets enabled us to apply the well-known "test set method" (Geurts & Wehenkel, 2020). This method allows to properly determine a good model and estimate its performance by performing the following procedure:

1. Fit the models to compare on the learning set, using different neural network architectures or different hyperparameters.

		Classes									Total
		Copper	Laminated zinc	Paint	Brass	Aluminum	PCB	Nickel	Rubber	Zamak	
Sets	LS	293	80	14	212	469	40	89	17	352	1566
	VS	171	66	6	176	194	33	46	7	135	834
	TS	112	60	17	112	262	13	45	9	137	767

Table 5.2: Class distribution inside the original Learning, Validation, and Test Sets: the three sets are imbalanced in terms of class.

2. Based on its performance on the validation set, select the best one.
3. Retrain this model on a new dataset built by merging the learning set and the validation set together.
4. Compute an estimation of the performance of this resulting model on the test set.
5. Retrain this model on the whole initial dataset (LS + VS + TS) to obtain the final model.

5.2 Dataset augmentation

To improve the overall performance, we can apply data augmentation to increase the size of the Learning Set (LS). Indeed, it is been proven in the deep learning literature that suitable data augmentation increases the model performance (Mikołajczyk & Grochowski, 2018).

The data augmentation performed in this work has been implemented using the *transform* class of the TorchVision library. More specifically, 4 new datasets have been created by applying 4 different processes on each image of the original LS.

To create the first and second new datasets, all images of the initial LS have been flipped horizontally and vertically, respectively.

The third new dataset has been obtained by applying an affine transformation to all images of the original LS. More precisely, each multi-feature image is rotated randomly with an angle between -180° and +180° and is then translated with random horizontal and vertical shifts between 0% and 40% of the width and the height of the multi-feature image, respectively.

The last new dataset is composed of patchwork multi-feature images. Each of those patchwork images has been generated by performing the following process:

1. An artificial multi-feature image is created by assigning to each channel a specific constant value that corresponds to the feature value acquired when there is no object on the conveyor belt.

2. The number of objects to integrate to the initial multi-feature image is drawn randomly in the range $[2, 10]$.

3. While this amount of objects has not been integrated in the artificial multi-feature image, an object is randomly extracted from the initial LS and is patched in the multi-feature image (if it does not overlap with any already patched object, otherwise another object is randomly picked).

Thanks to this new dataset, the initial LS is extended with multi-feature images containing objects of different classes.

It is important to note that we only got new images that are meaningful and possible to encounter in practice in the Multipick application.

Once these four new datasets have been built, they have simply been added to the original LS to form an augmented LS. Figure 5.3 highlights the three augmentations performed to build the first three new datasets. Figure 5.4 visually shows the resulting augmented dataset.

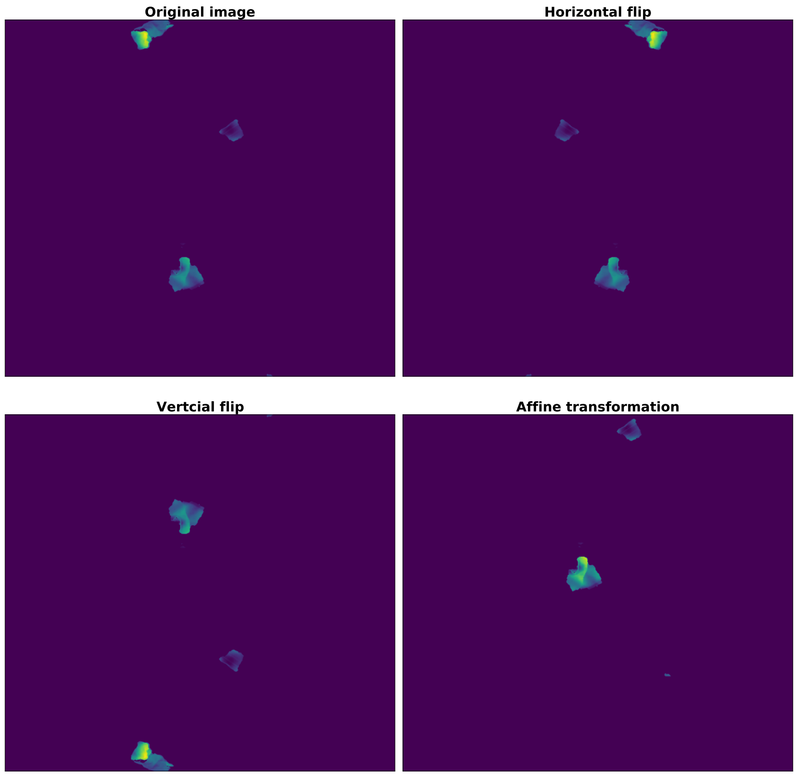


Figure 5.3: Visualization of three data augmentations performed on the same original image (upper left image). The depicted pictures correspond to the channel n°1 of the multi-feature images.

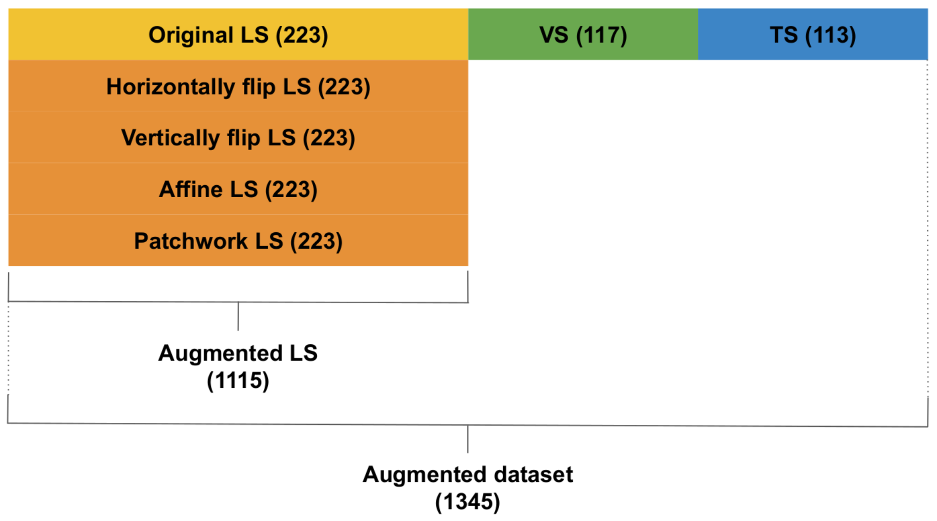


Figure 5.4: Illustration of the augmented dataset. The scalars in parentheses correspond to the number of multi-feature images contained inside each set. Compare to the original dataset (Figure 5.2), the Learning Set (LS) size has been multiplied by 5 thanks to the addition of 4 new sets (in orange) obtained by performing 4 different data augmentation techniques.

Chapter 6

Method

Outline
In this chapter, we describe in detail the specific Mask R-CNN architecture that will be tested in the next chapter. In Section 6.1, we motivate the choice of using Mask R-CNN in the context of this master's thesis. In Section 6.2, the required data preprocessing is briefly explained. Finally, in Section 6.3, the different Mask R-CNN backbones used in the next chapter are described.

6.1 Method selection

The use of Mask R-CNN to tackle this work is mainly motivated by the three following reasons:

1. Mask R-CNN achieves great instance segmentation performance. Even the current state-of-the-art deep neural network in instance segmentation (Cascade Eff-B7) is based on Mask R-CNN (Ghiasi et al., 2020) (see Figure 4.1).

2. Mask R-CNN is highly flexible and modular. Indeed, the backbone of Mask R-CNN can be any CNN and the head of the network can also be replaced very easily by another architecture. This great advantage of Mask R-CNN is directly visible in Figure 4.1: Mask R-CNN achieved state-of-the-art result multiple times thanks to the used of more and more efficient backbones. Furthermore, Mask R-CNN can be extended in cascade to form a larger deep neural network called Cascade Mask R-CNN (Cai & Vasconcelos, 2019).

3. Mask R-CNN architecture and working principle are described in details in the literature: it is possible to get a deep understanding of it.

	Channel n°										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Mean	0.16	64593	64404	2014	1911	2502	2479	2318	2281	2200	2187
Standard deviation	1.73	3390	3003	1335	858	474	417	404	414	453	554

Table 6.1: Means and standard deviations of the 11 channels computed on the original learning set.

6.2 Preprocessing

To have an effective training, the input data $\mathbf{x}$ is normalized channel-wise. That is, each channel of the input is normalized according to the corresponding channel mean and standard deviation of the original learning set. The channel means and standard deviations of the original learning set are shown in Table 6.1.

6.3 Mask R-CNN backbone

As mentioned in Section 3.2.5, the first part of Mask R-CNN is a CNN backbone. In the context of this master's thesis, two families of advanced CNN architecture have been evaluated inside Mask R-CNN: ResNet-FPN and ResNeXt-FPN. These two families correspond to the popular ResNet and ResNeXt architectures on which a Feature Pyramidal Network (FPN) has been applied on top.

6.3.1 ResNet

ResNet stands for Residual Network because this CNN is composed of a sequence of building blocks called residual blocks. A residual block is a stack of convolutional layers with an identity shortcut connection. Hence, instead of directly fitting the desired function H(x), the stacked layers fit another function $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$ that has been empirically proven to be much easier to optimize. The intuition behind is that if the desired function is the identity one, it would be easier to push the residual $\mathcal{F}(\mathbf{x})$ to zero than to fit the identity function by a stack of nonlinear layers. A generic residual block is shown in Figure 6.1.

Figure 6.2 describes the different ResNet architectures when used as feature extractor.

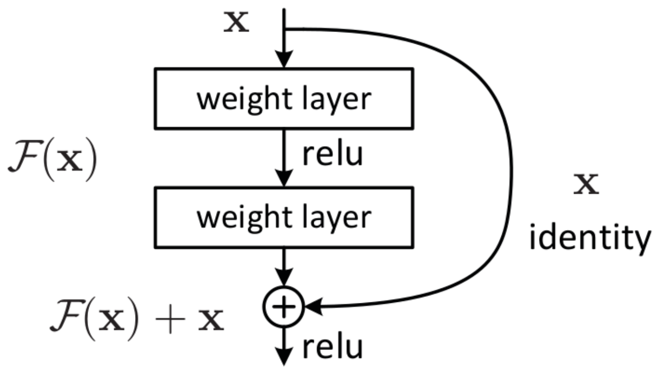
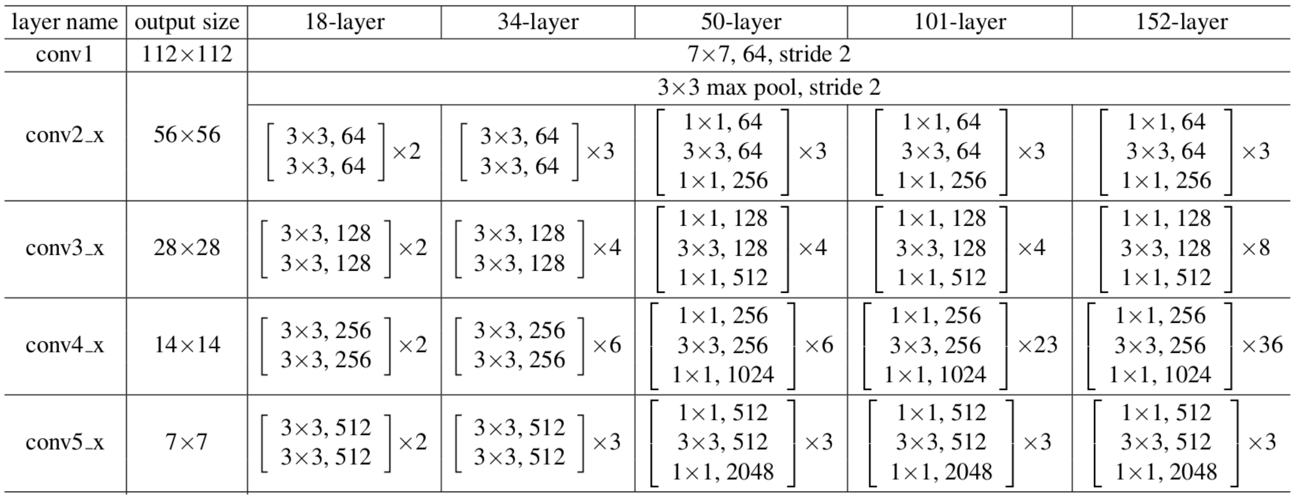


Figure 6.1: Residual block used inside ResNet. It is a stack of convolutional layers with an identity shortcut connection (He et al., 2016).



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$

Figure 6.2: Architecture of different versions of ResNet when used as feature extractor (He et al., 2016). Residual blocks are shown in brackets with the numbers of blocks stacked. In practice, down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

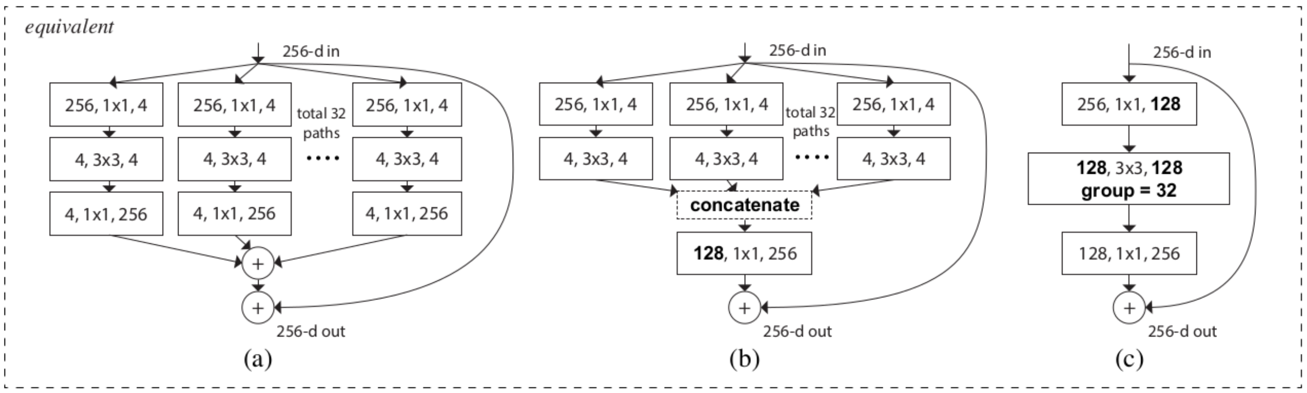


Figure 6.3: Equivalent ResNeXt building blocks with a cardinality equal to 32. **(a)** Aggregated residual transformation formulation. **(b)** Early concatenation formulation. **(c)** Grouped convolutions formulation. A layer is denoted as (# input channels, filter size, # output channels) (Xie et al., 2017).

6.3.2 ResNeXt

ResNeXt is a variant of ResNet. The difference lies in the building block used. As in ResNet, the ResNeXt building block has a shortcut connection. But, instead of just stacking convolutional layers in series, it aggregates a set of independent stacked convolutional layers with the same topology. Figure 6.3 visually illustrates this building block with three equivalent formulations. The last formulation is the one used in practice. In this reformulation, all the first 1×1 layers are replaced by a single wider layer, and the splitting is performed using a grouped convolutional layer. Briefly, a grouped convolutional layer is a layer that divides its input channels into groups and performs an independent convolution on each of those groups. Its output is formed by concatenating the results of all convolutions.

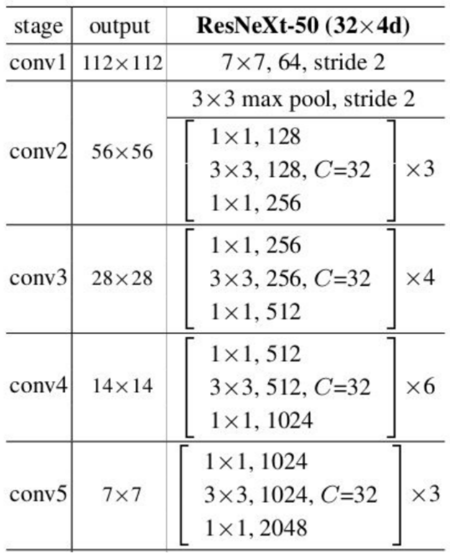
This new building block exposes a new dimension, which is called "cardinality" . This dimension corresponds to the number of stacked layers used in parallel.

Thanks to this new building block, ResNeXt outperforms ResNet while maintaining the same complexity. Figure 6.4 describes a ResNeXt architecture when used as feature extractor.

6.3.3 Feature Pyramid Networks

Principle

In summary, the goal of a Feature Pyramid Network (FPN) is to build high-level semantic feature maps at all scales using a CNN backbone. It takes an image of an arbitrary size as input and outputs multiple feature maps of proportional size. The performed process is independent of the



stage	output	**ResNeXt-50 (32×4d)**	
conv1	112×112	7×7, 64, stride 2	
conv2	56×56	3×3 max pool, stride 2	
		$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix}$	×3
conv3	28×28	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix}$	×4
conv4	14×14	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix}$	×6
conv5	7×7	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix}$	×3

Figure 6.4: ResNeXt with 50 layers and using the building block formulation depicted in Figure 6.3 (c) (Xie et al., 2017). ResNeXt building blocks are shown in brackets with the numbers of blocks stacked. $C = 32$ refers to a grouped convolution operation with 32 groups, while $4d$ means that each of the 32 convolutionnal layers composing the grouped convolution operation has a width of 4.

CNN backbone used and can be decomposed into 2 phases: the bottom-up phase and the top-down phase with lateral connections.

Bottom-up phase This phase corresponds to the feed-forward pass of the CNN backbone. Thanks to this pass, several feature maps at various scales are computed. Making the hypothesis that the backbone can be decomposed into successive blocks (called stages) such that the layers inside each stage generate feature maps of the same size. The FPN uses the last feature map of each stage as its initial set of feature maps. In the specific case of ResNet (and ResNeXt), it corresponds to the output of the last residual blocks of conv2, conv3, conv4 and conv5 (see Figure 6.2). In the literature, these outputs are commonly denoted as $\{C2, C3, C4, C5\}$.

Top-down phase with lateral connections. This second and final phase starts with $C5$ on which a 1×1 convolution is performed to produce the coarsest feature map. Then, the spatial resolution of this feature map is upsampled by a factor 2 and merged with the corresponding bottom-up map (which has been fed to a 1×1 convolution to reduce channel dimensions, beforehand) by element-wise addition. This process is iterated until the feature map with the finest resolution is created. Figure 6.5 visually illustrates this procedure. Finally, the enhanced set of feature maps (called $\{P2, P3, P4, P5\}$) is obtained by applying a 3×3 convolution on each merged feature map.

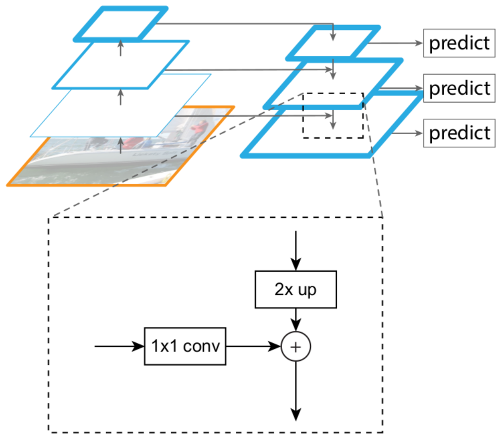


Figure 6.5: Illustration of the upsampling and merging operations taking place during the top-dwon phase of a Feature Pyramidal Network (Lin et al., 2017).

FPN for RPN

In the original RPN design (described in Figure 3.13), a small FCN with two heads is evaluated on 3×3 windows over a single scale feature map to perform binary classification and bounding box regression with respect to a set of anchors. The anchors enable to detect object with various shapes and scales even if the RPN is using a single scale feature map.

However, thanks to FPN, the RPN performance can be improved: it is performed by replacing the single feature map by the $\{P2, P3, P4, P5\}$ feature maps produced by FPN. The small FCN is now attached to each level of the feature pyramid and each anchor of a single scale is assigned to the feature map corresponding to that scale: anchor areas of $\{32^2, 64^2, 128^2, 256^2\}$ pixels are defined on $\{P2, P3, P4, P5\}$, respectively. As in the original implementation, anchors of multiple aspect ratios are used at each level. This enhanced RPN can be naturally trained and evaluated in the same fashion as in the original implementation described in Section 3.2.4.

FPN for Mask R-CNN head

Most commonly, the head of Mask R-CNN makes its predictions by extracting features from a RoI (given by the RPN) defined on a single-scale feature map.

Thanks to the FPN, it is possible to map RoIs of different scales to the set of feature map of different scales generated by the FPN. More precisely, a RoI of width w and height h is assigned to the level P_k of the FPN by evaluating the following empirical equation:

$$k = \lfloor k_0 + \log_2(\sqrt{wh}/224) \rfloor \tag{6.1}$$

where 224 is a scalar that has been proven to provide good results, k_0 is the target level on which a RoI with $w \times h = 224^2$ should be mapped into. In practice, k_0 is set to 4.

Using this mapping, features from the P_k feature map are pooled to a fixed size feature map and processed in the same way as in the original implementation.

Chapter 7

Mask R-CNN optimization

Outline
In this chapter, we carry out the optimization of Mask R-CNN. More precisely, in Section 7.1, we describe the different metrics that have been used to evaluate the performance of Mask R-CNN. In Section 7.2, we describe the baseline Mask R-CNN model used as a reference and the evaluation of its performance. Finally, in Section 7.3, we express the optimization procedure that has been followed to determine the best Mask R-CNN model and the underlying experiments.

7.1 Evaluation metrics

In this master's thesis, the quantitative evaluation of the Mask R-CNN models has been performed using the concept of contingency table and mean average precision.

7.1.1 Confusion matrix and contingency table

A very popular and useful tool to evaluate the performance of a binary classification algorithm is the confusion matrix C. Each line of the matrix corresponds to the instances in an actual class, while each column represents the instances in a predicted class. Figure 7.1 illustrates such a table with a positive and negative class. By definition, each element C_i^j of the table is equal to the number of inputs known to be of class i and predicted to be of class j. Therefore in the case of a binary classification, the count of true positives (TP) is C_0^0, false negatives C_0^1 (FN), false positives C_1^0 (FP) and true negatives C_1^1 (TN). These four quantities are extremely useful because it is possible to compute several performance metrics by combining them. For instance, the recall and precision can be computed as follows:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{7.1}$$

and

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{7.2}$$

This concept of confusion matrix trivially extends itself to more than two classes. Such a matrix is then called a contingency table. In that case, if the elements of such table T are denoted by t_{ij}. Then, the precision and recall for the class k are defined by

$$\text{recall} = \frac{t_{kk}}{\sum_j t_{kj}} \tag{7.3}$$

and

$$\text{precision} = \frac{t_{kk}}{\sum_i t_{ik}}. \tag{7.4}$$

respectively.

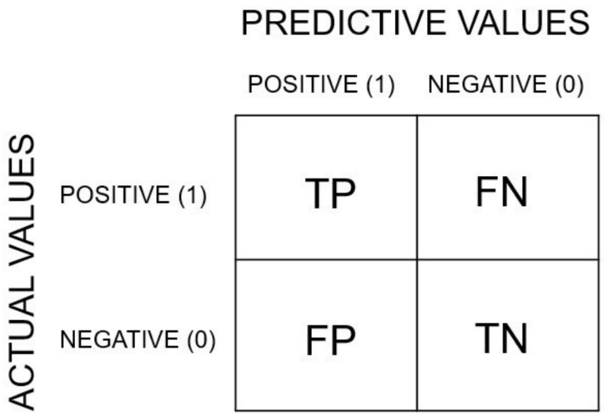


Figure 7.1: Illustration of a confusion matrix where TP, TN, FP and FN correspond to the true positive, true negative, false positive and false negative counts, respectively (Stardat, 2021).

And the accuracy A is defined as

$$\text{A} = \frac{\sum_k t_{kk}}{\sum_{ij} t_{ij}}. \tag{7.5}$$

Likewise, the error rate ERR is trivially defined as

$$\text{ERR} = 1 - A. \tag{7.6}$$

7.1.2 Average precision and mean average precision

A popular metric in objection detection to compare the performance of different algorithms is the Average Precision (AP). This class metric is the area under the precision-recall curve of a given class and is computed for each class in 5 steps (COCO, 2021a):

1. Each prediction on the test (or validation) set is classified as true positive or false positive. Concretely, a prediction (composed of the bounding box coordinates, and a class with its confidence) is classified as true positive (TP) if the IoU between the bounding box defined by the prediction and the one of the corresponding ground truth is greater than a specific threshold. Otherwise, the prediction is consider as false positive (FP). Furthermore, in the case where there are more than one prediction overlapping a ground truth, only the prediction with the highest IoU is considered as TP: all the others are considered as FP.
2. All predictions are sorted in descending order by their class confidence score.
3. After that, the accumulated TP and FP can be computed for each prediction.
4. From these accumulated values, the precision and recall for each prediction can be calculated. Plotting all these precision and recall values gives the precision-recall curve.
5. The final step is to compute the area under the obtained precision-recall curve. In practice, to be fast, this area is approximated in the COCO challenge by averaging the precision at a set of 101 equally spaced recall levels (from zero to one). Each precision value at a specific recall level is interpolated by taking the maximum precision value whose recall value is greater or equal than the next recall level. Figure 7.3 shows an example of a possible precision-recall curve obtained by applying such interpolation to a set of 11 recall levels and considering an IoU threshold of 0.75 on a toy set of cat detections summarized in Figure 7.2 (Padilla, Passos, Dias, Netto, & da Silva, 2021).

Bounding Box	Confidence (τ)	IOU	IOU > 0.75?	$\sum$TP(τ)	$\sum$FP(τ)	$Pr(\tau)$	$Rc(\tau)$
D	99%	0.91	Yes	1	0	1.0000	0.0833
K	98%	0.70	No	1	1	0.5000	0.0833
C	95%	0.86	Yes	2	1	0.6667	0.1667
H	95%	0.72	No	2	2	0.5000	0.1667
L	94%	0.91	Yes	3	2	0.6000	0.2500
I	92%	0.86	Yes	4	2	0.6667	0.3333
A	89%	0.92	Yes	5	2	0.7143	0.4167
F	86%	0.87	Yes	6	2	0.7500	0.5000
J	85%	-	No	6	3	0.6667	0.5000
B	82%	0.84	Yes	7	3	0.7000	0.5833
E	81%	0.74	No	7	4	0.6364	0.5833
G	76%	0.76	Yes	8	4	0.6667	0.6667

Figure 7.2: Precision and recall computations for a set of bounding box detections, considering a total of 12 ground truths and an IoU threshold of 0.75 (Padilla et al., 2021). $\Sigma TP(\tau)$ and $\Sigma FP(\tau)$ are the accumulated TPs and FPs, respectively, whose corresponding confidence levels are larger than or equal to the confidence τ. Knowing that $TP + FN = P$ (the number of ground truths), Precision $Pr(\tau)$ and recall $Rc(\tau)$ values are calculated based on Equations 7.2 and 7.1 respectively .

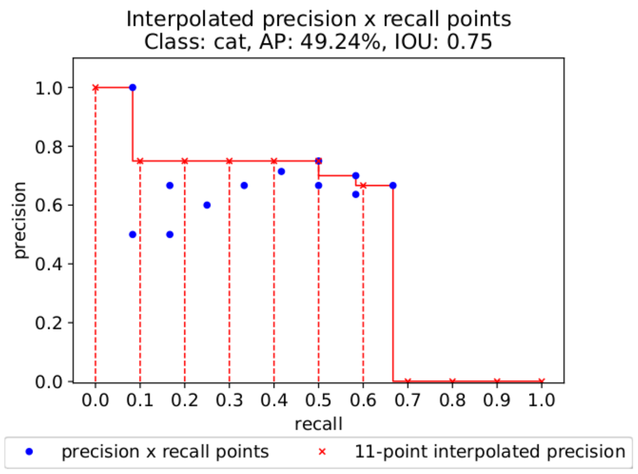


Figure 7.3: Average precision metric computation using 11-point interpolation with the precision and recall values of Figure 7.2 (IoU threshold of 0.75) (Padilla et al., 2021).

Another very popular scalar metric to evaluate object detection neural networks is the mean average precision (mAP). However, even if this metric is very popular, its definition varies through the deep learning literature. In the context of this work, I decided to use the definition given in the COCO challenge (COCO, 2021b): the mAP is the AP averaged over all classes and 10 IoU thresholds (from 0.5 to 0.95 with a step size of 0.05). This metric allows to compare very easily the global performance of different object detection algorithms.

Hyperparameter	Value
Training	
Optimizer	SGD with a momentum of 0.9 (Pytorch, 2021)
Learning rate	0.01
Number of epochs	100
Batch size	4
Weight decay	10^{-4}
Training dataset	Original learning set (no data augmentation)
Anchors	
Sizes of each anchor (in pixels)	{8, 16, 32, 64, 128}
Aspect ratios for each anchor size	{0.5, 1, 2}
Neural networks	
Backbone	ResNet-18 with FPN on top (Lin et al., 2017)
Head	Same as Figure 3.17
RPN	Same as Figure 3.13
Hardware	
Number of GPU used	2
GPU type	NVIDIA RTX 2080 Ti
Amount of CPU memory used	8 GB of memory

Table 7.1: Main Mask R-CNN baseline hyperparameters and neural networks.

In the context of instance segmentation, the computation of the AP and mAP metrics are unchanged except for the IoU evaluation, which is performed over masks instead of bounding boxes.

7.2 Baseline model

Concretely, ResNet-18 has been used as Mask R-CNN backbone to study and determine its best hyperparameters. Table 7.1 describes the main baseline hyperparameters used.

Using these baseline hyperparameters and neural networks, the loss and mean Average Precision (mAP) curves shown in Figure 7.4 have been observed. These loss curves ℓ have been obtained by computing after each epoch e the following expression:

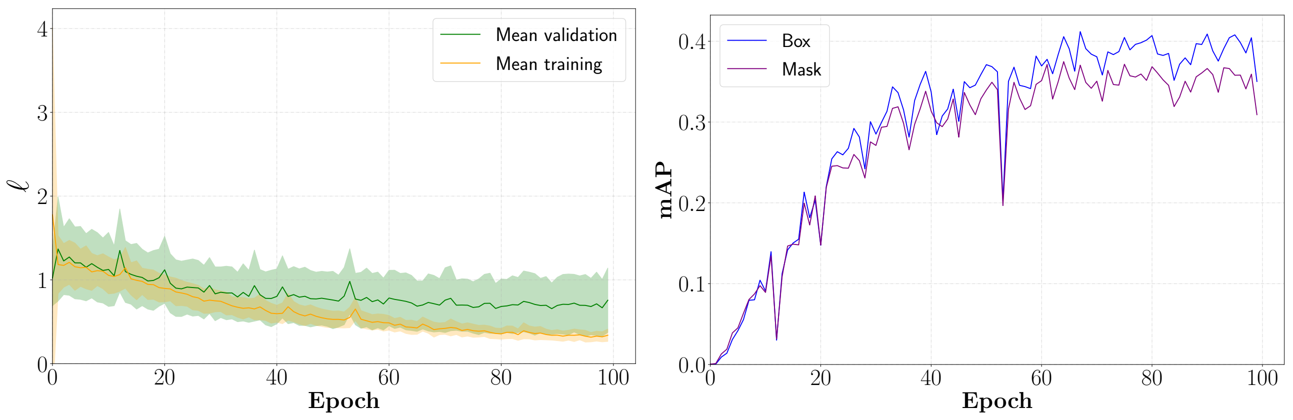


Figure 7.4: Loss curves (left graph) and mean average precision (right graph) obtained by Mask R-CNN using the baseline hyperparameters and neural networks described in Table 7.1. The validation loss and the mean average precision have been computed using the validation set: it reached a maximal mAP of 0.404 and 0.359 for its bounding box and mask predictions, respectively. In the left graph, the dark curves correspond to the average of the sum of the ℓ_{head} losses (see Equation 7.7) for each epoch.

$$\ell_e = \frac{1}{|D|} \sum_{i \in D} \sum_{j \in image_i} \ell_{head}^{ij} \tag{7.7}$$

where j corresponds to the j^{th} RoI detected in the i^{th} input image of the dataset D considered ($|D|$ is the size of the dataset D). ℓ_{head}^{ij} is the loss previously defined in Equation 3.29.

The validation loss and the mAP have been computed using the Validation Set (VS): the baseline deep neural network reached a maximal mAP of 0.404 and 0.359 for its bounding box and semantic mask predictions, respectively. From Figure 7.4, we can also observe that both training and validation loss curves are never increasing (in global trend) with the number of epochs: this points out that the model has not overfitted the training data.

7.3 Optimization procedure

To build a Mask R-CNN reaching very high performance, its optimization has been divided into two successive steps.

First, the best Mask R-CNN hyperparameter values have been determined with a fixed and relatively small CNN backbone. Using a small CNN backbone reduces the time required to train Mask R-CNN, and therefore enables the testing of several hyperparameters values in a reasonable amount of time.

Aspect ratios	mAP$_{box}$ (%)	mAP$_{mask}$ (%)
{0.5, 1, 2}	40.4	35.9
{1, 2, 3}	33	29.7
{0.5, 1, 1.5, 2, 2.5, 3}	35.6	31.8
{0.5, 0.75, 1, 1.25, 1.5, 1.75, 2}	**42.3**	**39.3**
{0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2}	40	35

Table 7.2: Performance result of Mask R-CNN on the validation set using different settings of anchor and the others baseline parameters defined in Table 7.1. mAP$_{box}$ and mAP$_{mask}$ are the mean average precision computed by using the bounding boxes and the masks, respectively. The set of aspect ratios in bold font achieved the best mAPs.

Then, using these Mask R-CNN hyperparameter values, a set of CNN backbones have been evaluated to determine which one is leading Mask R-CNN to achieve its best performance.

7.3.1 Hyperparameter optimization

To determine the best Mask R-CNN hyperparameter values with ResNet-18 as CNN backbone, the following hyperparameters have been sequentially tuned and evaluated on the VS : the aspect ratios for each anchor, the optimizer, and the data augmentation. By sequentially, it means that once a suitable value for an hyperparameter has been determined, this hyperparameter value has been used and left unchanged during the tuning of the other hyperparameters.

Anchor aspect ratios As shown in Table 7.2, the set of aspect ratios $\{0.5, 0.75, 1, 1.25, 1.5, 1.75, 2\}$ achieved the best mAPs: this set of aspect ratios has been used in all the next experiments of this master's thesis.

Optimizer In addition to the SGD optimizer, the Adam (Kingma & Ba, 2014) and AdamW (Loshchilov & Hutter, 2017) optimizers have also been tested. Table 7.3 summarizes the results and highlights that the Adam optimizer led to the best performance in terms of mAPs. Therefore, the Adam optimizer is the optimizer that has been used in all the next experiments of this master's thesis.

Data augmentation As explained in Section 5.2, we performed data augmentation to increase the size of the learning set. To see the impact in terms of performance on the validation set, ResNet-18 with the best hyperparameters identified so far has been trained on an incrementally growing learning set. The results of this experiment are displayed in Table 7.4: the learning

Optimizer	mAP$_{box}$ (%)	mAP$_{mask}$ (%)
SGD	42.3	39.3
Adam	**43.3**	**39.7**
AdamW	41.9	38

Table 7.3: Performance result of Mask R-CNN on the validation set using different optimizers. To work properly with Adam(W), the learning rate had to be decreased down to 0.0001. mAP$_{box}$ and mAP$_{mask}$ are the mean average precision computed by using the bounding boxes and the masks, respectively. The optimizer in bold font achieved the best mAPs.

Learning Set					mAP	
Original	Horizontally flip	Vertically flip	Affine	Patchwork	Box (%)	Mask (%)
✓	×	×	×	×	43.3	39.7
✓	✓	×	×	×	43	39.1
✓	✓	✓	×	×	42.5	38.2
✓	✓	✓	✓	×	**43.3**	**40.3**
✓	✓	✓	✓	✓	41.7	38.4

Table 7.4: Performance result of Mask R-CNN on the validation set with the best hyperparameters identified in the previous experiments using different (augmented) learning sets. The different augmentations are described in Section 5.2. The best mAPs are highlighted in bold font.

set composed of the original, horizontally flip, vertically flip, and affine datasets achieved the best performance. However, even if the performance of the complete learning set (containing all datasets) did not reach the best performance, it has still been used in all the next experiments of this master's thesis. Because, as the performance measurement is done on the validation set, the obtained mAps are just rough estimations of the real performance. In practice, the complete learning set could reach higher performance than the best one obtained in this experiment. This scenario is even highly probable as the Patchwork dataset is the only dataset having multi-feature images that are very likely to be encounter in reality. Indeed, its multi-feature images contain objects of different classes.

7.3.2 Backbone optimization

Once the best Mask R-CNN hyperparameter values have been determined with ResNet-18-FPN as CNN backbone, several other CNN backbones have been tested to determine which one is leading Mask R-CNN to achieve its best performance. More precisely, upscale versions of ResNet and another family of CNNs called ResNeXt have been tried. Sections 6.3.1, 6.3.2, and 6.3.3 describe ResNet, ResNext, and FPN respectively.

Backbone	mAP$_{box}$ (%)	mAP$_{mask}$ (%)
ResNet-18-FPN	41.7	38.4
ResNet-50-FPN	51.2	45.6
ResNet-101-FPN	19.8	19.9
ResNeXt-50-32×4d-FPN	51	48
ResNeXt-101-32×8d-FPN	**53.9**	**48.7**

Table 7.5: Box and mask mAPs on the validation set. ResNeXt-101-32×8d with FPN on top achieved the best performance with a mAP$_{box}$ and mAP$_{mask}$ of 53.9 % and 48.7 %, respectively. ResNet-50-FPN, ResNeXt-50-32×4d-FPN and ResNeXt-101-32×8d-FPN have been trained with a learning rate of 10^{-4}, while a learning rate of $9 \cdot 10^{-5}$ has been applied for ResNet-101-FPN.

Hyperparameter	Value
Training	
Optimizer	Adam
Learning rate	10^{-4}
Number of epochs	100
Batch size	4
Weight decay	10^{-4}
Training dataset	Augmented learning set
Anchors	
Sizes of each anchor (in pixels)	{8, 16, 32, 64, 128}
Aspect ratios for each anchor size	{0.5, 0.75, 1, 1.25, 1.5, 1.75, 2}
Neural networks	
Backbone	ResNeXt-101-32×8d with FPN on top
Head	Same as Figure 3.17
RPN	Same as Figure 3.13
Hardware	
Number of GPU used	2
GPU type	NVIDIA RTX 2080 Ti
Amount of CPU memory used	16 GB of memory

Table 7.6: Main characteristics of the efficient Mask R-CNN model determined throughout Section 7.3. The augmented learning set is the set depicted in Figure 5.4.

Table 7.5 shows the results obtained testing 5 different backbones: ResNeXt-101-32×8d with FPN on top reached the best performance with a mAP$_{box}$ and mAP$_{mask}$ of 53.9 % and 48.7 %, respectively.

7.3.3 Optimized model

Thanks to the optimization procedure described and conducted in Section 7.3, a very efficient Mask R-CNN model has been determined. Table 7.6 summarizes its main characteristics.

Chapter 8

Performance evaluation

Outline
In this chapter, we carry out an evaluation of the performance of the Mask R-CNN model designed in Chapter 7 and we discuss the results. More concretely, in Section 8.1, we present how the estimation of the theoretical performance has been computed and describe the obtained values. In Section 8.2, we proceed with the computation of the real performance estimation using a real set of scraps and we discuss the results.

8.1 Test set evaluation

Once we have determined the best hyperparameters and the CNN backbone (see Chapter 7), we can train a new Mask R-CNN model with this fine-tuned setup on a new dataset obtained by merging the complete augmented learning set with the validation set: this new dataset contains 1232 multi-feature images. Thanks to the test set previously defined in Section 5.1, we have a proper way to estimate the performance of this new model.

Figure 8.1 shows the loss and mean average precision curves. From this figure, we can observe that the loss curves both decrease up to the 40^{th} epoch. This points out that the model has stopped learning after this 40^{th} epoch. As the validation loss (green curve) computed on the test set is never increasing (in average), we can also state that the model has not overfitted the training data. Finally, the right graph shows that the model has reached a maximal mAP on the test set of 54 % and 47.1 % for its bounding box and mask predictions, respectively.

Using the test set, a contingency table can be computed. Table 8.1 shows the complete contingency table. Thanks to this table, we can evaluate the theoretical accuracy and the precision/recall

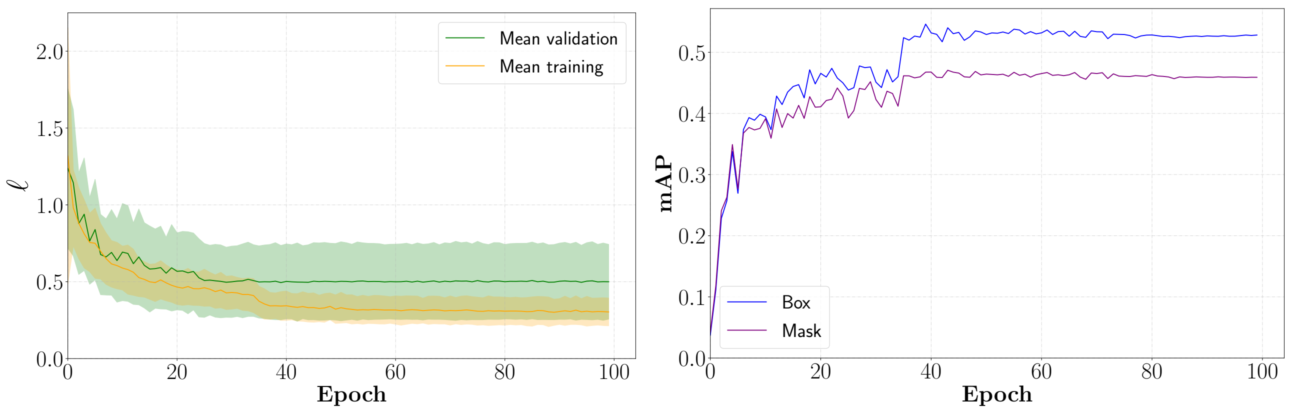


Figure 8.1: Loss curves (left graph) and mean average precision (right graph) obtained by the optimized Mask R-CNN model using the hyperparameters and neural networks described in Table 7.6. The test loss and the mean average precision have been computed using the test set: it reached a maximal mAP of 54 % and 47.1 % for its bounding box and mask predictions, respectively. In the left graph, the dark curves correspond to the average of the sum of the ℓ_{head} losses (see Equation 7.7) for each epoch.

per class. Table 8.2 shows the obtained values with the percentage of missed object and the average inference time (i.e. the time taken by the model to produce the box, class and mask predictions from a multi-feature image given in input): the Mask R-CNN model achieved a really good estimated accuracy of 89.5 %. More precisely, the copper and aluminum classes obtained excellent results by reaching a precision and a recall greater than 90 %, whereas the paint and rubber classes got a recall score smaller than 60 %. These low recall values are a direct consequence of the under representation of these two classes inside the acquired dataset. In terms of detection, the Mask R-CNN model reaches excellent performance by detecting 95.2 % of the objects present inside the test set.

		Predicted classes								
		Copper	Laminated zinc	Paint	Brass	Aluminum	PCB	Nickel	Rubber	Zamak
Actual classes	Copper	102	0	0	5	0	0	0	0	0
	Laminated zinc	0	46	0	0	0	0	0	0	12
	Paint	1	0	9	0	4	2	0	0	0
	Brass	6	0	0	92	3	0	4	1	2
	Aluminum	0	0	1	3	242	1	1	0	3
	PCB	0	0	0	1	1	10	0	0	0
	Nickel	0	0	0	3	2	0	33	0	4
	Rubber	0	1	0	1	2	0	0	5	0
	Zamak	2	5	0	1	2	0	3	0	114

Table 8.1: Contingency table computed using the optimized Mask R-CNN model on the test set defined in Section 5.1.

		Precision (%)	**Recall** (%)
Classes	Copper	91.9	95.3
	Laminated zinc	88.5	79.3
	Paint	90	56.3
	Brass	86.8	85.2
	Aluminum	94.5	96.4
	PCB	76.9	83.3
	Nickel	80.5	78.6
	Rubber	83.3	55.6
	Zamak	84.4	89.8

Accuracy (%)	89.5
Missed objects (%)	4.8
Inference time (ms)	149

Table 8.2: Theoretical accuracy estimation, precision/recall per class, percentage of missed object, and average inference time using the optimized Mask R-CNN model on the test set: the model achieved a really good estimated accuracy of 89.5 %. The precision, recall, and accuracy values have been obtained by only taking into account the detections of the model having a classification confidence score higher than 75 %. The average inference time has been measured with a batch size of 1 on a NVIDIA Quadro RTX 6000 GPU.

Some visual Mask R-CNN results on the test set are shown in Figure 8.2. As the test set is exclusively composed of multi-feature images containing objects of the same class, we can observe that the predictions are excellent but not perfect. For instance, the model has detected an object of the brass class inside the upper-left picture, while in practice the corresponding multi-feature image only contains copper scraps. Moreover, we can also see that some parts of several objects have not been detected.

8.2 Deployment and evaluation

Once the test set evaluation of the optimized Mask R-CNN model has been performed, we can build the final Mask R-CNN model with the fine-tuned setup determined in Chapter 7 on a new dataset obtained by merging the complete augmented learning set with the validation and test sets: this new dataset contains 1345 multi-feature images and corresponds to the "augmented dataset" shown in Figure 5.4.

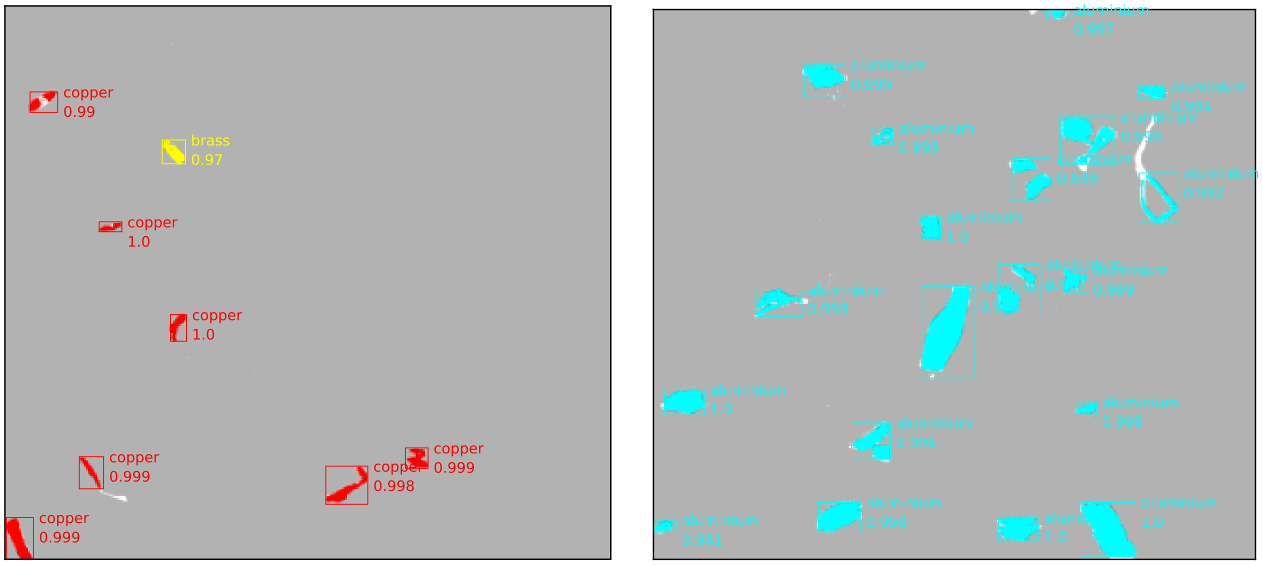


Figure 8.2: Mask R-CNN results on the test set showing detected scraps on the conveyor belt. These results are based on the optimized Mask R-CNN model, achieving a mask mAP of 47.1 % and running at 6 fps. Masks are shown in color on top of the scraps displayed in white, and bounding boxes, classes, and confidence scores are also displayed. Each color corresponds to a specific class: some parts of several objects have not been detected and one object has been misclassified.

8.2.1 Model integration

The current solution (from the acquisition to the grasping of the detected objects) used in the Multipick project is fully implemented in C++. In contrast, the final Mask R-CNN model is fully implemented in Python. Therefore, with the Multipick team, we had to find a way to integrate the Python model into the existing C++ pipeline.

We first tried to convert the Python model into a format that can be imported and interpreted directly in C++. In theory, the ONNX (ONNX, 2021) and TorchScript (PyTorch, 2021) formats are well suited to perform this operation. The Pytorch Python library enables to convert easily the model to these formats and there exist C++ libraries to import the model in these formats and interpret it. However, in practice, the final Mask R-CNN Python model uses recent functions and modules that are not yet supported in the Windows C++ libraries. Therefore, we had to find another way to integrate the Python model.

The solution that we finally found consists in making the Python model and C++ pipeline coexist through a TCP/IP client and server. More precisely, the computer inside the Pick-it prototype is creating a client in C++ and a server in Python, which communicate with each other. The communication between the client and server operates as follows:

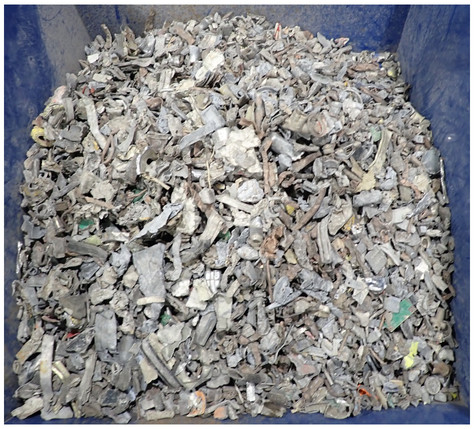


Figure 8.3: Illustration of the mix of scraps (called real test set) used to evaluate an estimation of the real performance of the final Mask R-CNN model: the objects have various shapes and the class distribution is completely unknown.

1. The C++ client sends to the Python server the multi-feature image acquired on the conveyor belt.
2. The Python server receives the multi-feature image, preprocesses it, and feeds it to the Mask R-CNN Python model.
3. The outputs of the model are postprocessed by the Python server and sent to the C++ client.
4. The C++ client decodes the data sent by the server and computes some useful information to properly grasp the objects such as the center of gravity of each detected object.
5. Finally, the C++ client sends the corresponding commands to the delta-robots to grasp and sort in the associated containers the detected scraps.

8.2.2 Real performance estimation

After successfully integrating the final Mask R-CNN model inside the Pick-it prototype, we assessed its real performance on the prototype by feeding the waste feeder (see Figure 2.1) with a real mix of scraps having an unknown class distribution. Figure 8.3 shows a picture of this mix.

Figure 8.4 illustrates the sorting of some of these scraps by the delta-robots thanks to the predictions of the final Mask R-CNN model.

By counting by hand the resulting number of scraps in each container, a contingency table can be built. The complete contingency table is shown in Table 8.3. In the current real prototype, the

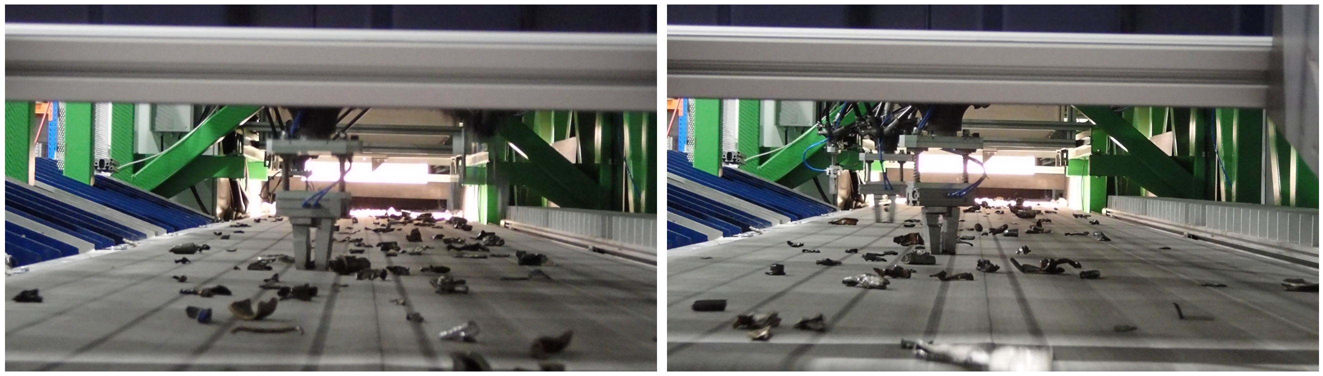


Figure 8.4: Illustration of the delta-robots grasping some scraps on the moving conveyor belt during the real assessment. We can observe that the conveyor belt is saturated with scraps.

		Predicted classes				
		Copper	Zinc	Brass	Aluminum	Nickel
Actual classes	Copper	72	21	60	15	0
	Laminated Zinc	2	185	4	3	1
	Paint	0	0	1	0	0
	Brass	5	32	132	7	3
	Aluminum	2	2	6	299	5
	PCB	0	0	1	0	0
	Nickel	0	1	16	4	32
	Rubber	0	0	1	1	0
	Zamak	0	0	22	0	0

Table 8.3: Contingency table obtained using the final Mask R-CNN model on the real test set with the Pick-it prototype. In the current real prototype, the laminated zinc and zamak objects are sorted inside the same container (called zinc). The PCB, rubber and paint objects are ignored.

laminated zinc and zamak objects are sorted inside the same container (called zinc), and the PCB, rubber and paint objects are ignored. This why this contingency table has only five columns.

From this table, the same metrics as the ones calculated in the theoretical assessment (Section 8.1) are computed, but only for the copper, zinc, brass, aluminum, and nickel classes. Table 8.4 summarizes the results: the final Mask R-CNN model achieved a good estimated accuracy of 77.1 %. To maximize the activity of the delta-robots, the conveyor belt is saturated with scraps: the percentage of missed (not detected) objects is therefore not meaningful anymore in terms of detections. From Table 8.4, we can notice that the inference time is different from the one obtained in Table 8.2: it is perfectly normal as the GPU used in practice is different from the one used for the test set evaluation in Section 8.1. This inference time being under 500 ms, the final Mask R-CNN model remains sufficiently fast to run properly on the Pick-it prototype.

		Precision (%)	**Recall** (%)
Classes	Copper	88.9	42.9
	Zinc	77.1	94.9
	Brass	54.3	73.7
	Aluminum	90.9	95.2
	Nickel	78	61.5

Accuracy (%)	77.1
Inference time (ms)	160

Table 8.4: Real accuracy estimation, precision/recall per class, and average inference time using the final Mask R-CNN model on the real test set. The precision, recall, and accuracy values have been obtained by only taking into account the objects sorted by the robots using the detected objects having a classification confidence score higher than 75 %. The average inference time has been measured with a batch size of 1 on a NVIDIA Leadtek RTX2080 GPU.

		Precision (%)	**Recall** (%)
Classes	Copper	> 93	> 60
	Zinc	> 90	> 60
	Brass	> 70	> 60
	Aluminum	> 85	> 60

Table 8.5: Description of the different requirement values for the copper, aluminum, brass, and zinc classes currently set by the Comet Group. Each class precision and recall requirement expresses the thresholds after which the sorting of that particular class becomes profitable. These values have been computed by Comet Group through a complete market analysis.

To be able to assess whether the obtained precision and recall values are sufficiently high or not, let us compare their values with the precision and recall requirements currently set by the Comet Group. Each class precision and recall requirement expresses the thresholds after which the sorting of that particular class becomes profitable. Table 8.5 shows the existing requirement values for the copper, zinc, brass, and aluminum classes. All the values are in % in mass, while all the precision and recall values computed in this master's thesis are in % of classified objects. Therefore, in theory, we can not do a direct comparison between these values. However, in practice, we will do an approximation and compare these different values: by comparing Tables 8.4 and 8.5, we can observe that the final Mask R-CNN model only reaches the requirements for the aluminum class. Therefore, we can state that the achieved results are good but not yet sufficient to be profitable.

8.3 Discussion

8.3.1 Real versus theoretical

From Tables 8.2 and 8.4, we observe a gap of 12.4 % between the theoretical and the real accuracy estimations. This gap can be mainly explained by the following reasons:

- **Distribution shift and overestimation of the theoretical performance.** The theoretical performance has been evaluated on the test set which contains only 767 objects drawn from the same distribution as the one on which the Mask R-CNN model has been trained. In contrast, the real test set has been drawn from the true distribution of objects. This distribution is very likely to be quite different from the distribution on which the model has been trained on because the data are extremely diversified and the training dataset used is small. Indeed, as shown in Figure 8.3, the objects have a large number of different shapes. Moreover, as they come from different sources (automobile, pipes, ...), the objects also have a large range of atomic concentrations. Therefore, during the real assessment, the Mask R-CNN model has encountered a large amount of objects with new shapes and atomic concentrations that differ by a significant margin from the ones on which it has been trained on: this directly leads to many wrong classifications and thus a drop in terms of accuracy. Furthermore, as the theoretical assessment has been performed on a test set drawn from the same distribution as the one on which the model has been trained on and as the size of this test set is small, the theoretical performance estimations are overestimated.

- **Class overlapping.** The brass being an alloy of copper and zinc, there is an overlapping between the copper, zinc and brass classes. Similarly, zamak is an alloy with a base metal of zinc and alloying elements of aluminum, magnesium, and copper: there is a second overlap. These overlaps are directly visible in Table 8.3 where all zamak and many copper scraps and have been wrongly classified in the brass container. Having such overlapping increases significantly the complexity of the classification task.

- **Heterogeneous object classes.** The acquired dataset used to train and test theoretically the deep learning model is only composed of multi-feature images containing objects of the same class. However, most of the time in practice, the multi-feature images sent to the model contain objects of multiple classes. The "pacthwork" augmentation described in Section 5.2 directly tackles this issue but there is no guarantee that this augmentation solves it completely: the model might still experience some difficulties to handle multi-feature images with heterogeneous object classes.

- **Change of the acquisition conditions.** The acquisition conditions of the multi-features images might have changed between when the dataset was acquired in October 2020 and

		Precision (%)	**Recall** (%)
Classes	Copper	90.8	68
	Zinc	82.6	97.4
	Brass	68.6	78.7
	Aluminum	93.6	93.9
	Nickel	77.6	24
Accuracy (%)		81	

Table 8.6: Real accuracy estimation, precision/recall per class of the current characterization method used in the Pick-it and Multipick projects. These values have been computed by the Comet Group on a real test set having a total weight of 99.415 kg.

when the real assessment was performed in May 2021. This change could have been due to multiple factors: change of some parameters in the X-ray module, in the VNIR camera, in the illumination of the conveyor belt under the VNIR camera, ... If this acquisition change has occurred, it impacts directly the quality and the values of the features embedded inside the multi-feature images, it can thus mislead the model.

8.3.2 Current ML model versus final DL model

The performance of the current characterization method (based on a Support Vector Machine method) used in the Multipick project is shown in Table 8.6.

By comparing Tables 8.4 and 8.6, we can observe that except for the nickel class, the deep learning Mask R-CNN model has not outperformed the current characterization method in the classification task. Nevertheless, in terms of accuracy, the two approaches differ only by 3.9 %.

However, as mentioned in Section 1.2, CNN based models (such as Mask R-CNN) have intrinsically a great potential to outperform the current characterization method thanks to their exploitation of the spatial information existing between pixels. Indeed, conversely to the CNN based models, the current characterization method classifies each pixel individually and it does not leverage the spatial information existing between pixels to make its pixel predictions.

Another advantage of the Mask R-CNN model over the current characterization method is its completeness. The Mask R-CNN model detects and predicts the class, position, and precise profile of each detected object all by itself. On the contrary, the current characterization method requires an upstream segmentation method to transform its pixel predictions into object level classifications.

Chapter 9

Conclusion

Due to the non-exploitation of the spatial information provided by the sensors, the current characterization method of the Multipick demonstrator is very likely to be suboptimal and has a limited scope of application. However, reaching the highest performance possible is of great importance as it directly translates into a higher waste recovery.

In this master's thesis, we have investigated the potential of deep neural networks to outperform the current characterization method by exploiting simultaneously both spectral and spatial information provided by three sensors (a dual X-Ray transmission sensor, a 3D ranging camera and a hyperspectral camera).

At the end of this thesis, we have finally succeeded to show the great potential of deep learning to perform multi-sensor instance segmentation by integrating a fully functional and optimized deep neural network inside the real and pioneer Pick-it prototype.

While the results obtained with the deep learning model in terms of localization are excellent, the classification accuracy obtained with the Pick-it prototype is too low to be profitable and to outperform the current waste characterization method.

Nevertheless, this preliminary work shows the great potential of the exploitation of the spatial information existing between pixels and unveils various directions of future work that could lead to the creation of a deep learning model outperforming the current characterization method. Concretely, here are five possible avenues (sorted by decreasing order of potential in terms of performance improvement) that could be implemented to achieve this goal:

1. **Train on more data.** The most effective and simple way of improving the performance of the Mask R-CNN model is to train it on a much larger dataset. Doing so, the deep learning model will encounter a larger variety of object shapes with diverse atomic concentrations,

from which it will be able to fit a much more accurate model. As all the codes required to go from the acquired dataset to the training and the integration of the trained model into the Pick-it prototype have already been implemented, this improvement is really straightforward to put in place and could by itself achieve the goal.

2. **Use a more complex classification head.** As shown in Figure 3.17, the classification head of the Mask R-CNN model used in this master thesis is quite simple: replacing it by a more complex set of layers could significantly improve the accuracy of the predictions. As the Mask R-CNN model is fast (160 ms to produce the box, class and mask predictions from a multi-feature image given in input), we could replace the current classification head by a fast and effective CNN such as MobileNetV3 (Howard et al., 2019).

3. **Use a more effective backbone.** Instead of using the ResNeXt-101-FPN as Mask R-CNN backbone, we could use the brand new EfficientNetV2-M (Tan & Le, 2021) CNN that achieves state-of-the-art results in the classification task (with a FPN on top), while having an acceptable inference time for the Pick-it application. As the feature maps produced by the backbone are the main components of the Mask R-CNN model, a backbone producing more valuable features will directly improve the Mask R-CNN predictions.

4. **Extend Mask R-CNN.** Instead of using the Mask R-CNN with FPN architecture presented in this master's thesis, we could use its extension called Cascade Mask R-CNN (Cai & Vasconcelos, 2019). As shown in Figure 4.1, this deep neural network significantly outperformed Mask R-CNN and achieved state-of-the-art results in 2020. However, this extension leads to an increase of the inference time. But, in practice, this increase might be sufficiently small to be acceptable in the Multipick application.

5. **Train on heterogeneous images.** As mentioned in Section 8.3.1, training a Mask R-CNN model on a dataset exclusively composed of multi-feature images containing objects from the same class could have a negative impact on the performance (even though a "patchwork" data augmentation is performed). Therefore, a direct improvement would be to acquire multi-feature images containing heterogeneous object classes. It has not been performed in this master's thesis because this makes the creation of the ground truths quite difficult. However, in practice, this could be done and automated thanks to the use of a laser-induced breakdown spectroscopy scanning system that would enable to precisely determine the ground truth class of each object during the acquisition of multi-feature images on the conveyor belt.

Chapter 10

Appendices

I Overview of the 9 classes

The 9 classes of scraps to detect and sort are illustrated with several physical samples for each class in the following figure:

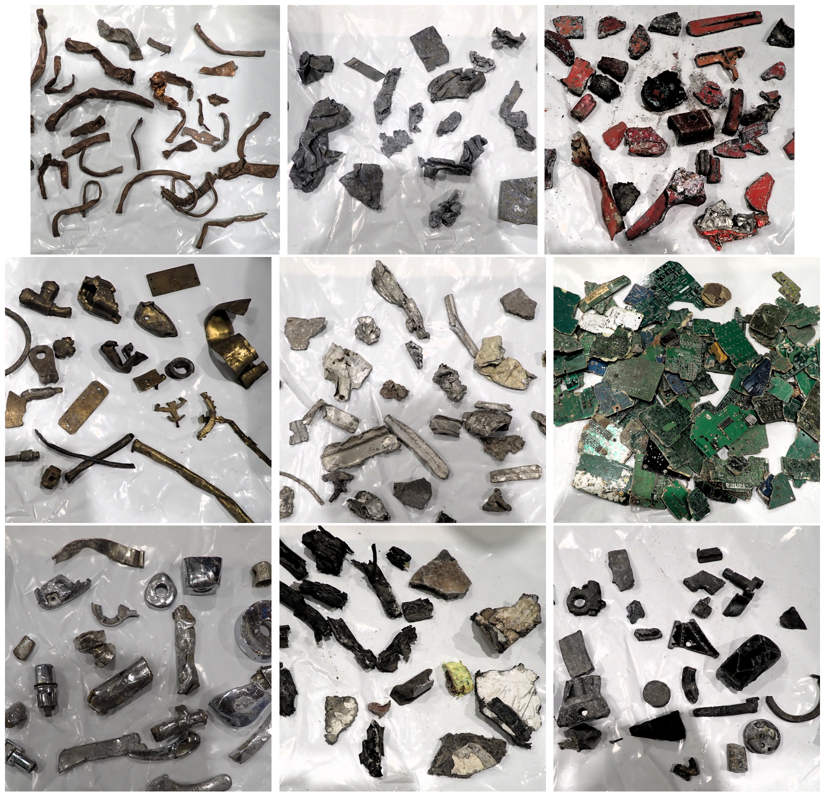


Figure 10.1: Overview of the 9 classes of scraps considered in this master's thesis. First row: copper, laminated zinc and paint samples. Second row: brass, aluminum, and PCB scraps. Third row: nickel, rubber and zamak objects.

II Input channels overview

The following figure represents the 11 channels of one of the multi-feature images present inside the dataset:

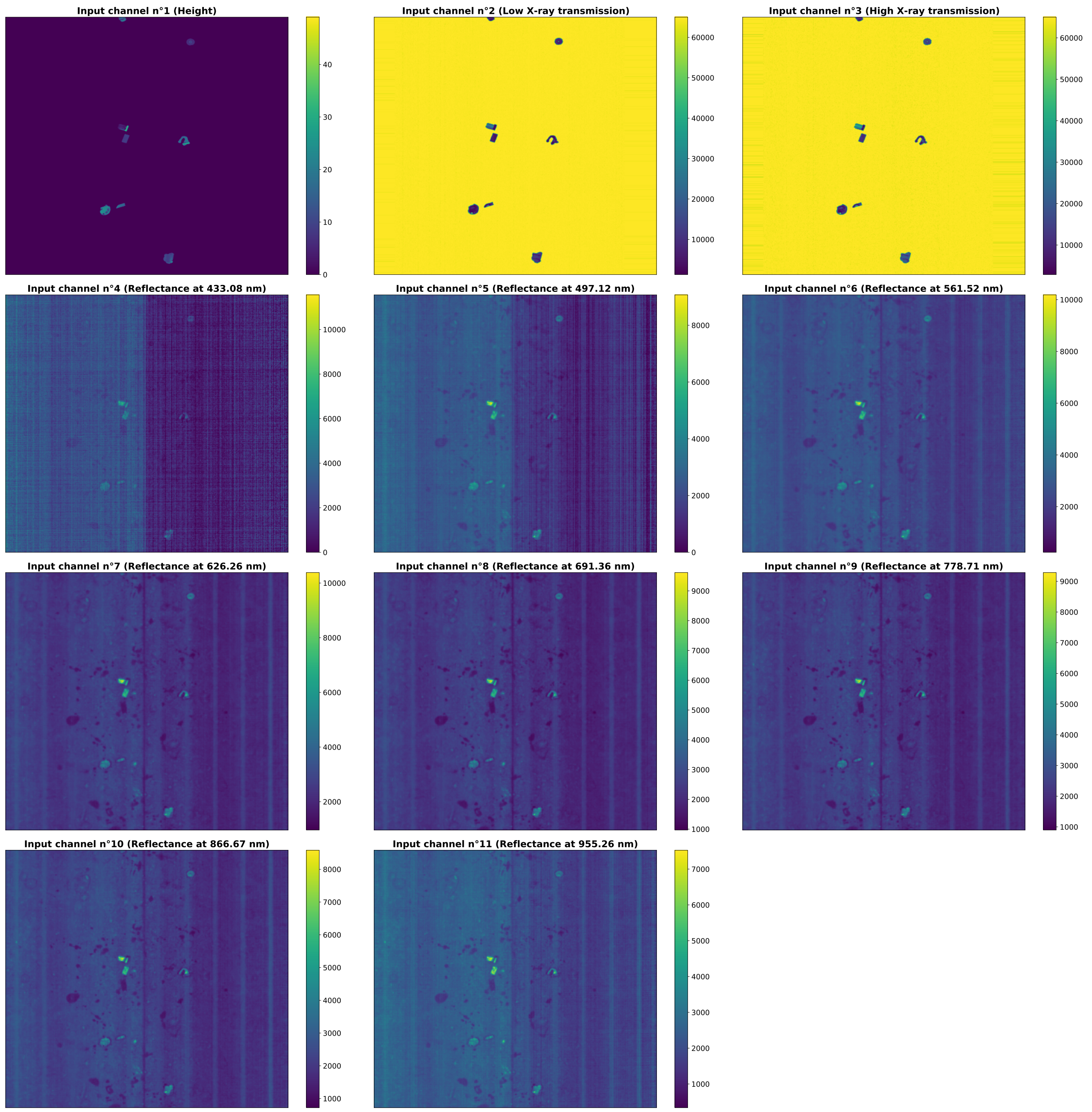

Input channel n°1 (Height)
Input channel n°2 (Low X-ray transmission)
Input channel n°3 (High X-ray transmission)
Input channel n°4 (Reflectance at 433.08 nm)
Input channel n°5 (Reflectance at 497.12 nm)
Input channel n°6 (Reflectance at 561.52 nm)
Input channel n°7 (Reflectance at 626.26 nm)
Input channel n°8 (Reflectance at 691.36 nm)
Input channel n°9 (Reflectance at 778.71 nm)
Input channel n°10 (Reflectance at 866.67 nm)
Input channel n°11 (Reflectance at 955.26 nm)

Figure 10.2: Overview of the 11 channels of one sample of the dataset.

References

Cai, Z., & Vasconcelos, N. (2019). Cascade r-cnn: high quality object detection and instance segmentation. *IEEE transactions on pattern analysis and machine intelligence*.

COCO. (2021a). *Coco evaluation script.* Retrieved 2021-03-26, from `https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/cocoeval.py`

COCO. (2021b). *Detection evaluation.* Retrieved 2021-03-25, from `https://cocodataset.org/#detection-eval`

Coco api. (2021). Retrieved 2021-05-05, from `https://github.com/cocodataset/cocoapi`

Fleuret, F. (2020). *Computer vision tasks.* Retrieved 2021-03-16, from `https://fleuret.org/dlc/materials/dlc-slides-8-1-CV-tasks.pdf` (slide 12)

Fleuret, F. (2021). *Deep learning 4.4 convolutions.* Retrieved 2021-02-22, from `https://fleuret.org/dlc/materials/dlc-slides-4-4-convolutions.pdf` (slide 2)

GeMMe. (2021). *Gemme: Construction materials, process mineralogy, mineral processing, extractive metallurgy recycling.* Retrieved 2021-03-30, from `http://www.gemme.ulg.ac.be/`

Geurts, P., & Wehenkel, L. (2020). *Bias/variance trade-off, model assessment and model selection.* Retrieved 2021-05-05, from `https://people.montefiore.uliege.be/lwh/AIA/IML___Model_evaluation.pdf`

Ghiasi, G., Cui, Y., Srinivas, A., Qian, R., Lin, T.-Y., Cubuk, E. D., … Zoph, B. (2020). Simple copy-paste is a strong data augmentation method for instance segmentation. *arXiv preprint arXiv:2012.07177*.

Ghiasi, G., Lin, T.-Y., & Le, Q. V. (2019). Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 7036–7045).

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the ieee international conference on computer vision* (pp. 1440–1448).

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 580–587).

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 315–323).

Group, C. (2021). *Comet group.* Retrieved 2021-03-30, from `https://www.cometgroup.be/`

?lang=en

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. In *Proceedings of the ieee international conference on computer vision* (pp. 2961–2969).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Hinton, G. (2014). *Overview of mini-batch gradient descent.* Retrieved 2014-03-16, from `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., ... others (2019). Searching for mobilenetv3. In *Proceedings of the ieee/cvf international conference on computer vision* (pp. 1314–1324).

ImageNet. (2012). *Large scale visual recognition challenge 2012 (ilsvrc2012).* Retrieved 2021-02-23, from `http://www.image-net.org/challenges/LSVRC/2012/results.html`

Introducing json. (n.d.). Retrieved 2021-05-05, from `https://www.json.org/json-en.html`

Kaiming, H., Georgia, G., Piotr, D., & Ross, G. (2017). *Mask r-cnn iccv 2017(oral).* Retrieved 2021-04-25, from `https://www.slideshare.net/windmdk/mask-rcnn`

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*, 1097–1105.

Latombe, J.-C. (2011). *Cs 121: Introduction to ai.* Retrieved 2021-03-22, from `http://ai.stanford.edu/~latombe/cs121/2011/slides/A-introduction.pdf`

Li, F.-F., Johnson, J., & Yeung, S. (2017). *Lecture 11: Detection and segmentation.* Retrieved 2021-02-22, from `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf` (slide 17, 62)

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2117–2125).

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740–755).

Loshchilov, I., & Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Louppe, G. (2021a). *Lecture 2: Multi-layer perceptron.* Retrieved 2021-02-23, from `https://glouppe.github.io/info8010-deep-learning/pdf/lec2.pdf` (slides 33, 50)

Louppe, G. (2021b). *Lecture 4: Training neural networks.* Retrieved 2021-03-22, from `https://glouppe.github.io/info8010-deep-learning/?p=lecture4.md#21` (slides 21, 25, 28)

Louppe, G. (2021c). *Lecture 5: Convolutional neural networks.* Retrieved 2021-02-23, from `https://github.com/glouppe/info8010-deep-learning/blob/master/lecture5.md` (slides 35, 40)

Mikołajczyk, A., & Grochowski, M. (2018). Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary phd workshop (iiphdw)* (pp. 117–122).

Nalepa, J., Tulczyjew, L., Myller, M., & Kawulok, M. (2019). Segmenting hyperspectral images using spectral-spatial convolutional neural networks with training-time data augmentation. *arXiv preprint arXiv:1907.11935*.

Noble, W. S. (2006). What is a support vector machine? *Nature biotechnology*, *24*(12), 1565–1567.

Noh, H., Hong, S., & Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of the ieee international conference on computer vision* (pp. 1520–1528).

ONNX. (2021). *Open neural network exchange.* Retrieved from `https://onnx.ai/`

Padilla, R., Passos, W. L., Dias, T. L. B., Netto, S. L., & da Silva, E. A. B. (2021). A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, *10*(3). Retrieved from `https://www.mdpi.com/2079-9292/10/3/279` doi: 10.3390/electronics10030279

PapersWithCode. (2021). *Instance segmentation.* Retrieved 2021-04-05, from `https://paperswithcode.com/task/instance-segmentation`

Pascal2. (2012). *The pascal visual object classes homepage.* Retrieved 2021-03-09, from `http://host.robots.ox.ac.uk/pascal/VOC/index.html`

Pytorch. (2021). *Source code for torch.optim.sgd.* Retrieved 2021-05-09, from `https://pytorch.org/docs/stable/_modules/torch/optim/sgd.html#SGD`

PyTorch. (2021). *Torchscript.* Retrieved from `https://pytorch.org/docs/stable/jit.html`

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 779–788).

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*.

Scikit-image. (2021). *blob_dog.* Retrieved 2021-07-05, from `https://scikit-image.org/`

Stardat. (2021). *Confusion matrix.* Retrieved from `https://www.stardat.net/post/confusion-matrix`

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 2820–2828).

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105–6114).

Tan, M., & Le, Q. V. (2021). Efficientnetv2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298*.

Uijlings, J. R., Van De Sande, K. E., Gevers, T., & Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, *104*(2), 154–171.

Wang, L. (2005). *Support vector machines: theory and applications* (Vol. 177). Springer Science & Business Media.

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1492–1500).

Yani, M., & Si.M.T.BudhiIrawan, S. (2019). Application of transfer learning using convolutional neural network method for early detection of terry's nail. In (p. 3).

Yin, C., Tsung-Yi, L., Matteo, R. R., & Alexander, K. (2020). *Coco 2020 object detection task.* Retrieved 2021-05-05, from `https://cocodataset.org/#detection-2020`

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). *Dive into deep learning.* Retrieved 2021-03-01, from `https://d2l.ai/d2l-en.pdf` (pp. 259, 614, 615)

Zitnick, C. L., & Dollár, P. (2014). Edge boxes: Locating object proposals from edges. In *European conference on computer vision* (pp. 391–405).