Deep Learning for the Classification and Detection of Animals in Artworks

Auteur : Claes, Yann
Promoteur(s) : Geurts, Pierre
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en science des données, à finalité spécialisée
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/11511

University of Liège
Faculty of Applied Sciences

Deep Learning for the Classification and Detection of Animals in Artworks

Master's thesis carried out to obtain the degree of Master of Science in Data Science and Engineering.

Author
Yann Claes

Supervisor
Pierre Geurts

Academic year 2020-2021

Contents

Abstract

Digitization has established itself as an essential process in our daily modern life, as more and more business applications rely on this information processing tool in order to improve their customers' experience, for instance with online shopping. Aside of commercial uses, digitization is also applied to the domain of cultural heritage, for example to provide high-resolution representations of various kinds of artworks. In this thesis, we apply deep learning techniques to the art world with the end goal of pushing forward the digitization of artistic collections, by developing automatic techniques for their annotation process. In particular, we assess the performance of standard classification architectures when coping with a domain transfer from natural images to artworks. We then evaluate the impact of different learning settings and provide insightful observations about model predictions. Then, we move to the object detection problem, which represents the end task of this work. We investigate state-of-the-art object detection models and try to enhance their performance using several transfer learning strategies. Additionally, we present various error patterns encountered with the models and conclude with propositions of other learning approaches and perspectives to further tackle and improve on this problem.

Acknowledgements

First, I would like to thank my supervisor, Pierre Geurts, for providing me with this thesis proposal, which helped me to discover my interest in academic research. I am grateful for his precious advice, guidance and thorough proofreading of this work.

Second, I would also like to thank Matthia and Mike Kestemont, our partner at the University of Antwerp, for their constant presence and their help throughout this entire academic year.

Finally, I would like to thank my family and friends for their warm support during this work, but mostly for the great moments we lived during these last five years.

Chapter 1

Introduction

Digitization, defined as

> *The process of changing data into a digital form that can be easily read and processed by a computer* [1]

has grown in coverage of our everyday applications since its origination in the 1950s [2]. The development of computers yielded a series of innovative technologies that characterize the evolution of this new process. From the very first commercialized computer equipped with magnetic disk storage, to the latest growth of online transactions (facilitated by the introduction of the Internet), digitization paved its way to this day such that it now appears inconceivable to even imagine a life without these technologies.

Naturally, digitization does not only impact the world of online transactions and online interactions between companies and their customers. Digitization can also be extended to culture and cultural heritage [3], where it allows the creation of a new life form for this ancient but important field which provides an everlasting memory of our history. With the advent of virtual museum tours, people now have the possibility to contemplate artworks while staying home, a great opportunity in recent pandemic times which reshaped our ways of living. For example, the European Union has created several funding instruments that support the digitization and analysis of cultural heritage, under which *Europeana*. Europeana [4] is an initiative of the European Union whose aim is to share cultural heritage from thousands of European cultural entities through its digital transformation.

In an originally separated perspective, the domain of computer vision emerged [5] in the 1960s through research about visual perception, whose goal was to study how neurons react in response to various visual stimuli. From this point, the hierarchical neural structure behind computer vision was established, and researchers concentrated on reproducing human neurological structures, leading to the very first implementations of neural networks [6], named perceptrons.

Despite the extreme optimism about the field of Artificial Intelligence (AI) (which led to the AI winter starting in the 1970s[1]), it still managed to survive throughout time with the implementation of convolutional networks [7]. In addition, the introduction of the Internet made access to plenty of data much easier. Twenty years later followed the

[1] AI winter resulted from the large gap between the computing resources at the time and the complexity of the expectations and promises coming from researchers.

advent of deep learning algorithms, among which deep convolutional neural networks, which modified once and for all the landscape of computer vision.

To this day, even though a variety of methods exists (linear and non-linear filters, region growing, etc.) for a variety of tasks (classification, detection, segmentation), computer vision is prominently characterized by deep learning techniques [8] which take advantage of today's thirst for *Big Data*.

Although initially separately defined, computer vision can be readily applied to support and improve current cultural heritage digitization techniques. This union of domains can be associated to *Digital Humanities* [9], a field of research located at the intersection of the humanities and digital (computing) tools.

The present work is part of the *INSIGHT* project [10]. INSIGHT is a research project whose end goal is to make use of AI (specifically Natural Language Processing and Computer Vision) to supplement cultural digitization with descriptive metadata, which will allow the easy integration of enriched digitized collections into Europeana.

This work is anchored in the computer vision part of the project, whose goal is to apply deep learning techniques in order to automatically annotate large collections of artworks. For this intent, digital collections of the Royal Museums of Fine Arts of Belgium and the Royal Museums of Art and History are made available. The essence of this work is thus to study object detection techniques from machine learning (in particular, deep learning) applied to the artistic domain and to try and compare the results obtained to those established for natural images, as long as the comparison makes sense.

In this thesis, we investigate the application of standard deep learning architectures designed for classification and object detection to the artistic domain. In particular, we will focus on depictions of animals represented in various styles, as will be illustrated in Chapter 3. More specifically, object detection will be studied in two subsequent stages: firstly, classification of animal crops will be thoroughly analyzed before diving into the complete object detection pipeline, considering full-size paintings and no longer crops.

In this work, the following research questions will be considered:

- How do state-of-the-art classification architectures perform when facing a domain transfer from natural images to artworks?
- How, and to what extent, can we improve the classification performance of these architectures to account for this domain shift?
- Can we gain any insights about the predictions produced by these classifiers, i.e. can we characterize some important features considered by the models?
- How do standard object detection approaches compete while coping with this domain shift?
- How can we derive different training settings to deal with such domain transfer?

In order to address these research questions, we will begin both stages by a review of the performance of state-of-the-art architectures, pre-trained on large datasets of photo-realistic images, when being applied to crops or paintings respectively. The models that will be considered are presented in Chapter 2. Then, different training settings will be considered to determine how each relates to final performance. Details for these settings

are provided in Chapter 4 and 5 respectively and results for each architecture will be compared to see whether some settings are more beneficial when applied to specific models. In particular, a transfer learning setting will be compared with other approaches, along with variations of usual transfer learning.

The structure of this report is as follows. Chapter 2 will review related research about classification and object detection techniques in general and applied to artworks, and notably the research conducted in [11] (as part of the INSIGHT project) that deals with musical instruments detection in paintings. It will also provide a broad definition of transfer learning and how it can be accomplished. Chapter 3 will define in further details the scope of this thesis along with a description of the datasets used and the data retrieving process, to finish with the division in subsequent tasks. Chapter 4 will cover the classification task together with the results obtained while Chapter 5 covers the detection task.

Chapter 2

Background and related work

This chapter will review some theoretical background related to convolutional neural networks, which represent the state-of-the-art models for all machine learning tasks tackled during this thesis. Subsequently, it will cover state-of-the-art architectures considered in this thesis. Finally, it will present analogous research conducted in the field of digital humanities.

It will be split into two main sections, following the line of work presented in Chapter 1. Section 2.1 will review theoretical background and related work carried out in the scope of classification and object detection tasks. It will also present the different state-of-the-art classifiers and detectors that will be studied throughout these tasks, highlighting their differences and respective improvements. Finally, a definition of deep transfer learning will be provided as well as techniques showing how transfer learning can be applied. Section 2.2 will cover related work conducted in the field of digital humanities, concerning classification and detection tasks accomplished in the domain of artworks.

2.1 Deep learning for Computer Vision

2.1.1 Classification

In machine learning, and in particular in computer vision, a classification task is a task whose aim consists in training a model in order to correctly associate input images to their corresponding class, picked among a finite set of potential classes. An example batch of animal samples and their corresponding class is displayed on Figure 2.1.

Related work

There are numerous image classification techniques: k-nearest neighbor classifiers, support vector machines [12], etc. However deep convolutional neural networks (DCNN) have become a reference for this task, as they enable learning increasingly complex features and textures by processing the input image across their convolutional layers and by leveraging local receptive fields [13] from previous layers. This allows to extract spatial information more efficiently and accurately than what would be achieved if only fully-connected neural networks were used. For readers unfamiliar with convolutional neural networks, definitions of their main concepts are provided in Appendix A.

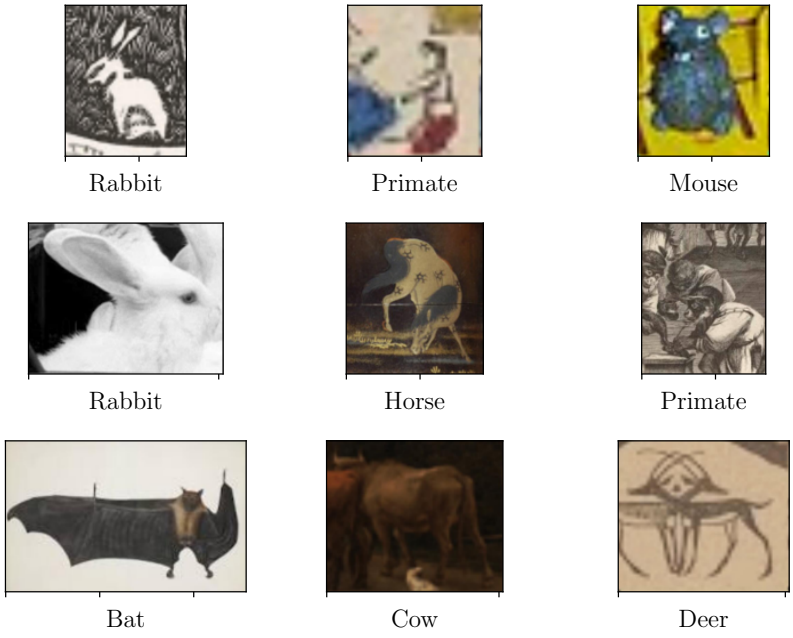


Figure 2.1: Random batch of animal crops with their classes.

Following this line of work, several architectures were elaborated, trained and tested on the ImageNet dataset [14], which contains around one million photorealistic pictures spread over a thousand classes and which represents the reference testbed for classification systems. The very first DCNN taking advantage of high-performance computing resources such as GPUs was AlexNet [15], which outperformed its previous state-of-the art by 10% on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) of 2012, thereby paving the way of DCNNs into the image classification field. Other architectures relied on similar principles but developed additional modifications to improve upon previous work: for instance, VGG networks [16] studied the influence of the network's depth, where adding more and more convolutional layers helped to increase the downstream accuracy, whereas Inception networks [17] are rather characterized by carefully crafted structures. Residual networks (*ResNets*) [18] built upon deeper networks whilst adding residual connections from layer inputs to layer outputs, and ResNeXt models [19] combine principles from both ResNet and Inception network architectures.

In total, three classifiers will be studied throughout the classification task, to eventually appraise the impact of transfer learning and fine-tuning. These classifiers are the following:

- ResNet-101
- ResNeXt-101 32×8d
- VGG-16

They were selected because, among all existing classifiers, they represent three particular classes of architectures, as will be underlined in the following theoretical reviews, and each class achieves state-of-the-art results on natural images. Pre-trained versions of these three classifiers on ImageNet are also directly available in PyTorch, which will allow us to explore transfer learning approaches. Note that more classifiers could of course be

added, but it would only increase the comparison load, while the main purpose is to study potential gains from transfer learning with a domain shift.

Each of the three architectures will now be reviewed to highlight the main contributions and differences that each of them brings.

VGG-16

VGG frameworks [16] build upon the initial architecture of AlexNet [15], with the main purpose of improving accuracy. To that end, authors decided to study network depth as a hyperparameter, fixing other parameters of the architecture. Only small convolutional filters will be considered, i.e. (3×3) and (1×1) filters. This allows to increase network depth safely without risking to end up with empty input tensors.

These frameworks are based on the following generic convolutional neural network configuration: input images will be 224×224 RGB crops, which will be processed through a stack of convolutional layers with small kernel sizes, as was stated in the above paragraph, and a spatial padding is added to ensure that spatial resolution is maintained after convolutions. Spatial downsampling will only be implemented by five max-pooling layers with a 2×2 window and stride 2. Therefore, the output of these layers will be a tensor of dimensions $\frac{H}{2} \times \frac{W}{2}$.

After this stack of convolutional layers come three fully-connected layers, the last one having 1 000 output neurons in the case of the ImageNet ILSVRC classification task. Class probabilities are eventually computed using a final softmax layer. All layers rely on the ReLU activation function.

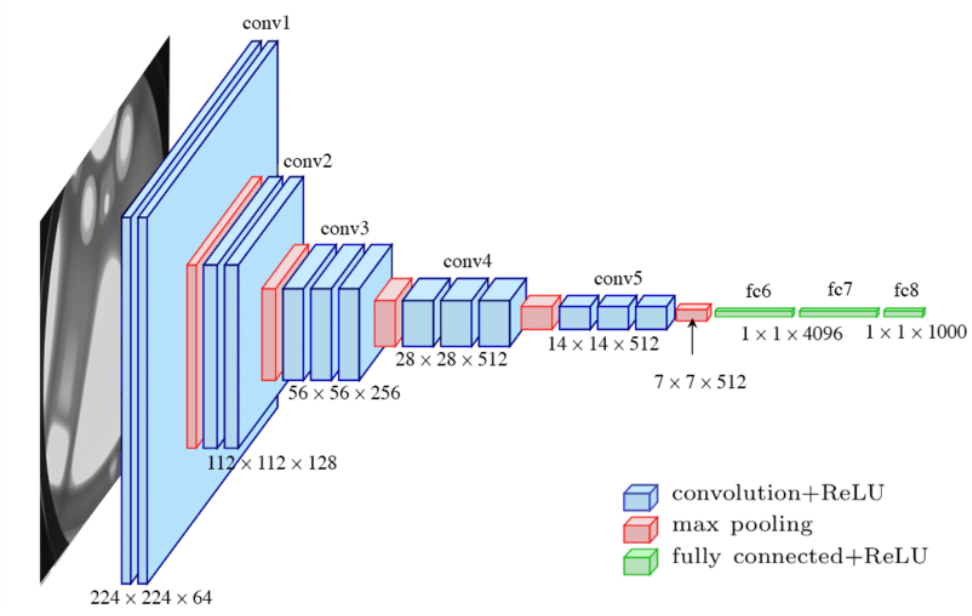


Figure 2.2: Architecture of a VGG-16 classifier. (Source: [20])

Differences with previous configurations Contrary to previous classifier implementations like in [15] and [21] where larger convolutional filters (11×11 and 7×7) are used

in the first layers, VGG frameworks rely on 3×3 filters, which allow to increase the effective receptive field. Indeed, a stack of three 3×3 convolutional layers has an effective receptive field of dimensions 7×7 but with only $3 \times 9 = 27$ parameters (omitting bias and assuming single-channel inputs and outputs) compared to 49 parameters that would need to be optimized with a single 7×7 convolutional filter.

Another advantage of this approach is that it integrates more non-linear rectification layers than previous approaches, which helps the decision surface region to be more discriminative.

In [16], authors show that deeper CNNs eventually lead to lower classification error on the ILSVRC dataset, whilst underlining that such models generalize well to other tasks and datasets, where they are pre-trained on ILSVRC and fine-tuned on task-specific datasets to avoid issues of overfitting. The architecture of VGG-16, used in this work, is displayed in Figure 2.2.

ResNet-101

In [18], authors build upon deeper neural networks introduced by VGG frameworks. However, training deeper networks is harder because of two issues:

- Vanishing/Exploding gradients
- Network degradation

The first issue has been coped with thanks to normalized initialization of weights [22, 23] along with the introduction of batch normalization [24].

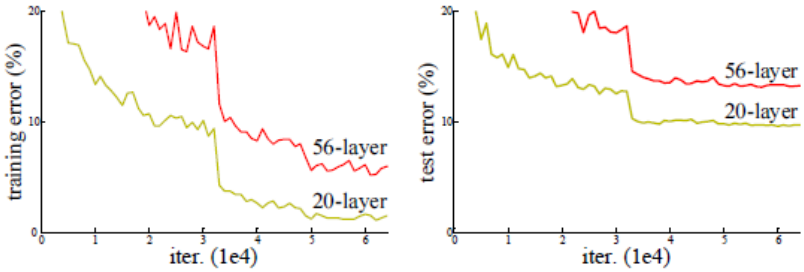


Figure 2.3: An illustration of degradation. (Source: [18])

Degradation is defined as the counter-intuitive phenomenon where training and test errors increase for deeper plain networks, while it could be expected that going deeper and deeper would only enhance the learning capacity, thereby reducing those errors. An illustration of degradation is shown in Figure 2.3.

Residual networks (ResNets) are thus defined to deal with this degradation issue, where layers are reframed to learn residual functions with respect to layer inputs, hence a residual mapping will be learnt by the network layers. The residual mapping to be learnt is

$$\mathcal{F}(\mathbf{x}) + \mathbf{x}$$

as can be seen from Figure 2.4. This reformulation is simply obtained using shortcut connections. The advantage of identity shortcut connections is that they do no add any

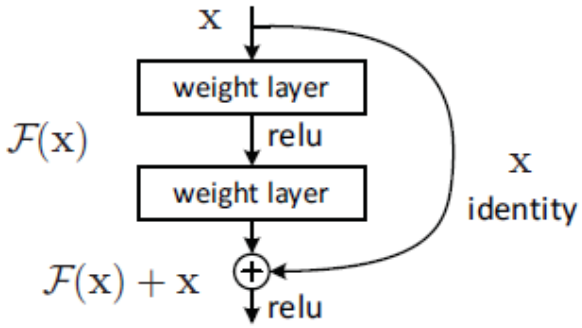


Figure 2.4: The residual learning block used in ResNets. (Source: [18])

parameter nor computational complexity. Multiple convolutional layers can be represented through the function $\mathcal{F}(\mathbf{x})$. Thus, a plain CNN is turned into its residual counterpart by simply introducing shortcut connections between certain module inputs and outputs.

Authors show that relying on such residual learning blocks allows to increase depth whilst alleviating the degradation issue, thanks to experiments conducted on ImageNet validation data where they compared values obtained for top-1 error with plain and residual networks. Therefore, deeper networks now perform better and converge faster. In addition, such frameworks can be considerably deeper than previous VGG models but with a lower complexity. ResNets also show fine generalization performance on tasks like object detection on PASCAL VOC or COCO benchmarks, reaching a 27.2% validation mAP @[.5, .95] when used as backbone of a Faster R-CNN model (introduced in a later section) against 21.2% for the same detection model but relying on a VGG-16 backbone.

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				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.5: Architectures for different ResNet models. Residual blocks are shown in brackets. (Source: [18])

The specific architecture selected in this work is ResNet-101, which indicates that the residual network has 101 convolutional layers, and is detailed in Figure 2.5. Building blocks consist of three layers made of two 1×1 and a single 3×3 convolutions, where the 1×1 convolutions are responsible for reducing and restoring the input tensor dimensions.

Thus, the middle 3×3 convolutional layer is left with input and output tensors of smaller dimensions, which names this building block a *bottleneck* residual block. Such a block allows to reduce training time. Since the framework was trained on ILSVRC, the final layers are a 1 000-way linear layer followed by a softmax layer.

ResNeXt-101 32×8**d**

As their name indicates, ResNeXt models [19] are built on top of aforementioned ResNet frameworks, since the straightforward rule of stacking blocks of the same shape allows to reduce the choices of hyperparameters but also to decrease the chances of overfitting to particular datasets. However, instead of simply stacking residual blocks composed of three convolutional layers, as can be seen from Figure 2.5, ResNeXt architectures repeat a residual block that aggregates a set of transformations. A comparison of both building blocks is provided in Figure 2.6. Transformations in this new block share the same topology, such that the cardinality, i.e. the size of the set of transformations, becomes a hyperparameter to investigate.

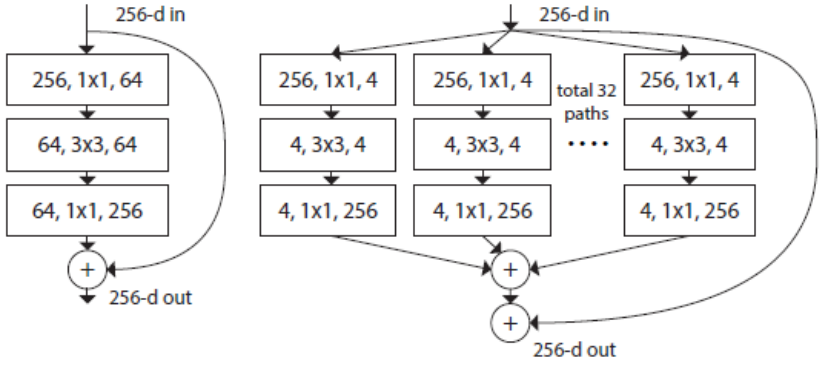


Figure 2.6: Comparison of building blocks used in ResNet (left) and in ResNeXt (right). (Source: [19])

ResNeXt frameworks combine principles defined in ResNets with principles defined in Inception [17] architectures. The latter rely on a "split-transform-merge" scheme where the input is first split into smaller pieces, then transformed using convolutional filters and eventually merged by concatenation. However, such a strategy requires customization for each transformation and can be too difficult to adjust if a dataset or task shift is to be considered.

For these reasons, ResNeXt frameworks consider transformations of the same topology within a building block, that will be aggregated by summation. This allows for a design that is general enough compared to Inception models, but it also leads to much simpler designs. Authors also highlight that a simple neuron in a neural network already performs a split-transform-merge operation, as the input vector is divided into individual components which are then scaled by corresponding weights and eventually aggregated by summation, which corresponds to a simple dot product operation.

There are two equivalent formulations to the building block defined on the right of Figure 2.6: these are displayed in Figure 2.7. Authors provide intuitive proofs of equivalence in [19], thus those will not be covered here and left to the reader's interest. Authors decide to rely on the third formulation for easier implementation.

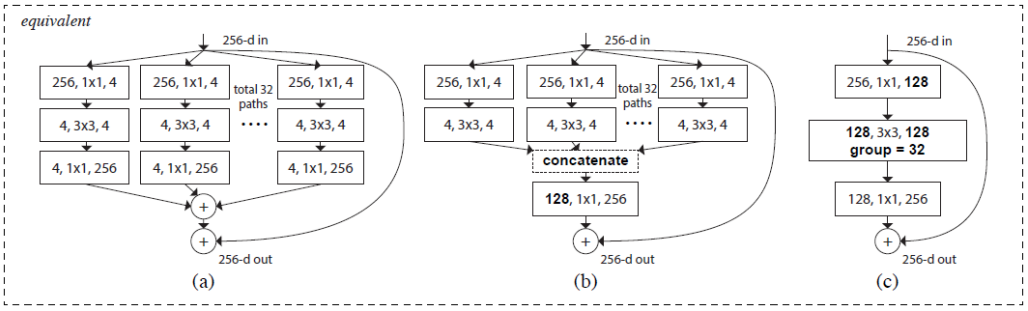


Figure 2.7: Illustration of equivalent building blocks for ResNeXt architectures. (Source: [19])

Aggregated transformations can be defined as

$$\mathcal{F}(\mathbf{x}) = \sum_{i=1}^{C} \mathcal{T}_i(\mathbf{x})$$

with $\mathcal{T}_i(\mathbf{x})$ an arbitrary function, e.g. the neuron dot product defined above, and with C referring to the aforementioned cardinality. To respect the split-transform principle, $\mathcal{T}_i$ should project the input into an embedding and transform the latter. Furthermore, as was mentioned, all transformations share the same topology and correspond to the bottleneck block defined in [18], where the first 1×1 convolutional layer will be responsible for the *split* operation.

Since ResNeXt models learn residual mappings, the output of each module is defined as

$$\mathbf{y} = \mathbf{x} + \sum_{i=1}^{C} \mathcal{T}_i(\mathbf{x})$$

ResNeXt-101 32×8d consists of the same architecture as ResNet-101, in terms of number of blocks per stage, which can be observed in Figure 2.5, i.e. conv2 contains 3 blocks, conv3 contains 4 blocks, etc., however, the structure of these blocks is different since each outputs the aggregation of C transformations, in this case 32 identical transformations are performed. The width (number of input and output channels) of the bottleneck layer is 8.

In order to derive a honest assessment of the influence of cardinality on downstream accuracy, a control of complexity needs to be performed to ensure that ResNeXt frameworks have roughly the same amount of parameters (and FLOPs) as C increases. From their experiments, authors showed that increasing cardinality whilst decreasing width allowed to reduce top-1 validation error obtained on the ILSVRC much more than when keeping a fixed cardinality and increasing depth or width. Similarly to previous ResNet models, ResNeXts generalize well on other tasks, such as object detection (using a Faster R-CNN model) and perform even better than their ResNet counterparts. Thus, ResNeXt frameworks could be expected to show better performance on paintings as some features are shared between both domains, e.g. animal aspects, environments, etc.

Classification metrics

To assess the performance of a classifier, we usually compute its error (misclassification) rate, or equivalently its top-1 accuracy. The latter can be computed by the ratio of correct

predictions over the total amount of predictions, i.e. the amount of input crops.

We can also compute precision and recall metrics to characterize the classifier even further. As a reminder, precision and recall are defined as:

$$\text{Precision}(c) = p(y = c \mid \hat{y} = c) = \frac{TP}{TP + FP}$$
$$\text{Recall}(c) = p(\hat{y} = c \mid y = c) = \frac{TP}{TP + FN}$$

for a given class c, where y denotes the ground truth and $\hat{y}$ the predicted class. In other words, precision gives us a measure of how confident one can be about the predictions of the model, while recall gives us a measure of the detection abilities of the model, for the considered class.

There exists of course a trade-off between precision and recall. Indeed, to increase recall, one could simply classify each input crop as a particular class, which would guarantee to have no false negatives for this class. However, this would naturally increase the number of false positives for that class, thereby reducing its precision.

In addition to precision and recall metrics, we can also compute the F-1 score. This score is defined as:

$$\text{F-1}(c) = 2 \cdot \frac{\text{P}(c) \cdot \text{R}(c)}{\text{P}(c) + \text{R}(c)}$$

and summarizes the trade-off between precision and recall. A score of 1 indicates that the model has a perfect precision and recall for the considered class.

Grad-CAMs

Grad-CAMs [25] represent a visualization tool that helps to understand predictions made by CNNs. In this approach, localization maps can be produced for a particular target class c, which will highlight important regions for the model to predict that class.

For that purpose, Grad-CAM (which stands for Gradient-weighted Class Activation Mapping) leverages the gradient information of the target concept that flows into the last convolutional layer of the CNN. Indeed, deeper layers should embed deeper representations while keeping spatial information, which allows to derive semantic maps for each output class. After having backpropagated the gradients of the corresponding output score with respect to the the feature map activations A^k, it computes importance weights α_k^c for each feature map k. Finally, a localization map of dimensions equal to those of the final convolutional maps is produced as follows:

$$L_{\text{Grad-CAM}^c} = \text{ReLU}\left(\sum_k \alpha_k^c A^k\right)$$

The ReLU activation is useful to capture only features that are relevant to the target class c.

However, Grad-CAM cannot produce highly detailed gradient maps, as it produces coarse class-discriminative heatmaps. For that purpose, authors combined Guided Backpropagation (which helps to visualize gradients at the pixel level) with Grad-CAM visualizations. Indeed, guided backpropagation allows to represent (positive) gradient information at

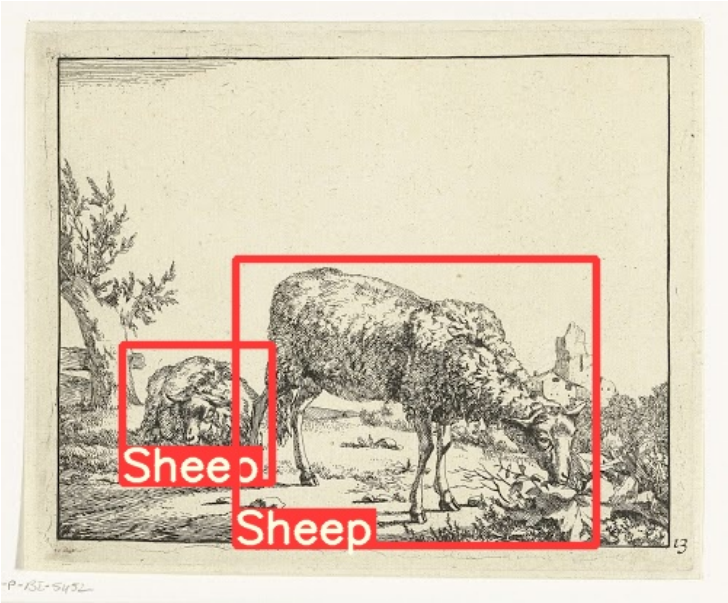


Figure 2.8: A sample of expected detection.

the pixel level, thus with respect to the input image, but it cannot discriminate classes. Therefore, combining both approaches yields as final representation a class-discriminative visualization, which is also of high-resolution.

2.1.2 Object detection

Object detection is usually defined as a combination of two different machine learning tasks:

- Object localization
- Object classification

Object localization consists in drawing bounding boxes corresponding to certain objects in an input image while object classification was defined above. Detection naturally combines both tasks, as object detection algorithms will output bounding boxes along with class labels and confidence scores for each class. An example of the output ideally expected from a detector can be observed in Figure 2.8, which represents a drawing of two sheep.

Related work

Similarly to what has been observed for the classification frameworks, detection algorithms have moved towards an intensive use of convolutional neural networks as it allows for an easy exploitation of spatially structured data.

While the ILSVRC also contains an object detection challenge, other detection evaluation benchmarks exist, such as the PASCAL VOC challenge [26] or the MS COCO challenge [27], both datasets containing respectively 20 and 80 classes of common objects annotated with bounding boxes.

Several models were trained on these task-specific datasets but some of them also lever-

aged other auxiliary datasets, like the ImageNet classification dataset [14]. For instance, R-CNN networks [28] rely on region-based object detection, using selective search to generate region proposals, and its CNN component is first pre-trained on the ImageNet classification dataset and then fine-tuned for detection either on PASCAL VOC or the detection dataset of ILSVRC. R-CNN received improvements which lead to Fast R-CNN [29] and Faster R-CNN [30] to cope mainly with efficiency issues, the latter relying on a separate region proposal network, allowing to reduce vastly the computation needed to generate regions of interest.

Other architectures like YOLO [31] drop region proposal components to rely only on a single network that will produce simultaneously bounding boxes and class probabilities for these boxes. Contrary to region-based detection methods, YOLO has the advantage of seeing the entire image, which allows the network to learn contextual information about objects, thereby reducing the number of false positives.

A choice also had to be made concerning the detection frameworks to evaluate for this task. Eventually, two object detectors will be covered throughout this task, and the impact of transfer learning and fine-tuning will also be assessed as much as it is feasible. These detection architectures are the following:

- Faster R-CNN
- YOLO

Similarly to the classification task, Faster R-CNN [30] and YOLO [31] were selected because they constitute state-of-the-art architectures to which new object detection models are often compared. Faster R-CNN is directly available in a PyTorch environment[1] while YOLO is not directly accessible from the same PyTorch environment but more details about the version used in the present work will be provided in a later section. Both architectures are interesting to compare because they belong to a particular type of approach for object detection.

As was done for the preceding classification task, a literature review for both models will be carried out to point out their main conceptual differences but also to provide a measure of their performance on the usual detection testbeds, i.e. benchmarks conducted on PASCAL VOC or COCO. Since they also rely on CNNs to solve their task, all building block definitions provided in Appendix A apply.

Faster R-CNN builds upon Fast R-CNN [29], which itself builds upon R-CNN [28]. It is thus reasonable to introduce the main features of both preceding architectures.

R-CNN

R-CNN uses a region proposal approach to solve downstream detection tasks. It relies on a selective search technique [32] that will output a set of 2 000 region proposals out of the original input image. This technique relies on an exhaustive search across the entire image combined with segmentation and similarity measures in order to yield various regions of different scales, in a bottom-up approach, i.e. from the pixel level to higher scales. Each region proposed by this technique is then warped and resized to deal with the fixed-size issue proper to CNNs combined with fully-connected layers.

[1] https://pytorch.org/vision/stable/models.html

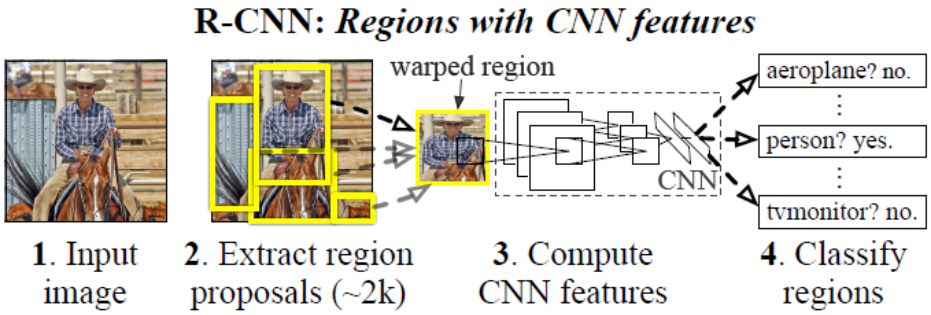


Figure 2.9: R-CNN architectural overview. (Source: [28])

Each resized region is fed to the CNN part of the system, which is responsible for extracting the main features and for translating them into a 4 096-dimensional feature vector. This final feature vector is eventually fed as input to class-specific SVM classifiers, which will score each region as belonging to the class the SVM was designed for. A bounding box regressor (specific to each class) is then applied after scoring in order to refine the region proposals into better-fitting localization boxes. The system overview of R-CNN is displayed in Figure 2.9.

The main issue of R-CNN is that it is computationally costly given that, for each proposal region, there corresponds a forward pass in the CNN; there are thus 2 000 forward passes for each input image. Furthermore, selective search works as an independent model, hence there is no learning potential to leverage in that component: if it produces bad initial candidates, it is likely that R-CNN will output bad refined boxes too. Finally, the R-CNN system flow is composed of three modules in total: the region proposal module, implemented by selective search, the feature extractor, implemented by a CNN, and the final classification/regression module. This combination of modules increases computation times.

Fast R-CNN

To alleviate the issue of processing each proposal region into a feature extraction module, Fast R-CNN extracts features for the whole input image once, using a CNN, and will rely on these feature maps for further steps.

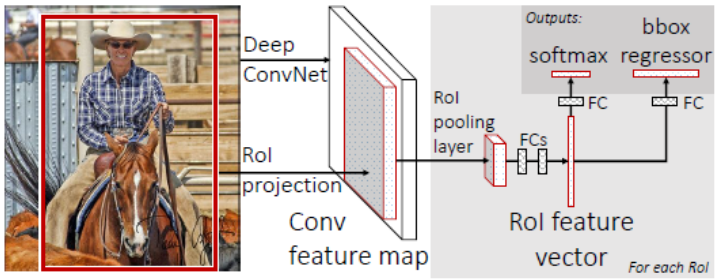


Figure 2.10: Fast R-CNN architectural overview. (Source: [29])

Fast R-CNN still uses a selective search technique to produce region proposals for the original input image. It will select the appropriate portion of the output feature map of the CNN module that corresponds to the particular region proposal. This portion is called

a *region of interest* (RoI). Note that, to obtain the RoI, region proposals naturally have to be scaled down by a certain factor, which represents the total spatial downsampling up to the present output feature map.

Since region proposals have different dimensions, Fast R-CNN cannot use fully-connected layers as simply as in R-CNN due to the issue of input dimensions. Thus, it will rely on a RoI pooling layer, which constitutes a particular case of an SPP block that uses only a single layer for max-pooling, characterized by a certain number of bins. This RoI pooling layer will produce fixed-size output vectors for each region, which will eventually be fed to fully-connected layers for classification (using a softmax layer) and for bounding box regression. The architecture of Fast R-CNN is illustrated in Figure 2.10.

Fast R-CNN allows to reduce computation times for both training and inference, since only a single CNN forward pass is required for a given input image. Furthermore, it improves accuracy as there is no loss of resolution since it can start from the original image. Nevertheless, region proposals generated by selective search are still not trainable and represent an efficiency bottleneck in the total computation as it takes around 2 seconds to produce region proposals in a CPU implementation.

Faster R-CNN

Faster R-CNN introduces a *region proposal network* (RPN), that will replace the selective search technique used in preceding models. This RPN consists of a fully convolutional neural network that will simultaneously predict object boundaries (coordinates) and object scores, at each position in the input feature map.

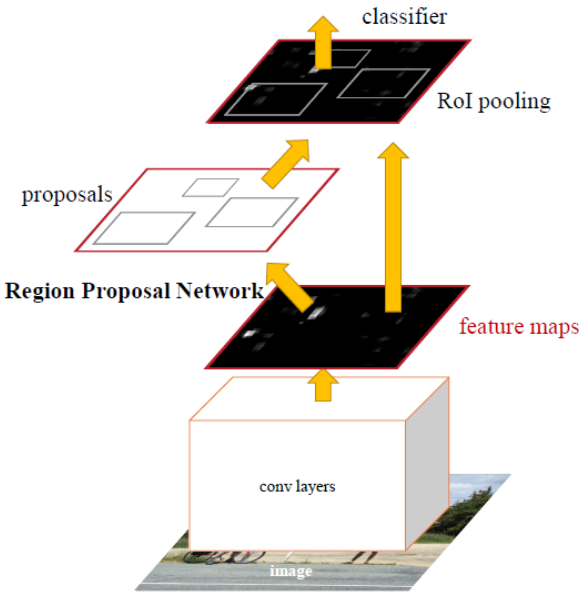


Figure 2.11: Faster R-CNN architectural overview. (Source: [30])

The goal of the RPN is to output a set of rectangular region proposals, each of which having a score of its probability of being an object against being part of the background. To generate such proposals, a small network is slid over the last convolutional feature map of the shared CNN. This small network has a sliding window of size $n \times n$, which means that it will take as input an $n \times n$ window from the output feature map. For each sliding

location, the RPN predicts a maximum of k region proposals which will be compared against k reference boxes, called anchors.

An anchor represents a box that is centered at the location of the sliding window and that is characterized by a particular size and aspect ratio. An anchor can either be positive, negative or insignificant for training. It will be considered as positive roughly if it has an IoU greater than 0.7 with any existing ground truth box, and will in that case be associated with that box. If its IoU with any ground truth box is lower than 0.3, then the anchor is a negative anchor as it represents the background. Anchors with an IoU that does not meet the above conditions will not influence training of the RPN. With these anchors defined, the RPN is trained in order to provide accurate region proposals, i.e. regions that encapsulate as precisely as possible ground truth objects while predicting high object scores (close to 1) if an object is indeed included in the proposal box.

On top of this RPN, a usual Fast R-CNN detector can be used and the process is the same as the one defined above. Thus, it will eventually output for each region proposal class probabilities and refined bounding boxes. Both modules can be combined in a unified network, as shown in Figure 2.11, and they can be trained jointly to optimize both detection and region proposal performance. Faster R-CNN thus alleviates the non-learnable issue that was inherent to the selective search technique and is able to produce region proposals efficiently.

YOLO

Unlike Faster R-CNN, YOLO [31] represents another class of detection frameworks, in particular regression-based detection frameworks, whereas Faster R-CNN belongs to region-proposal-based detection frameworks. This means that the general approach is completely different, as YOLO tries to combine all detection steps in one, i.e. it will not generate region proposals and then classify these regions but rather start from an input image and directly produce bounding box coordinates and class probabilities.

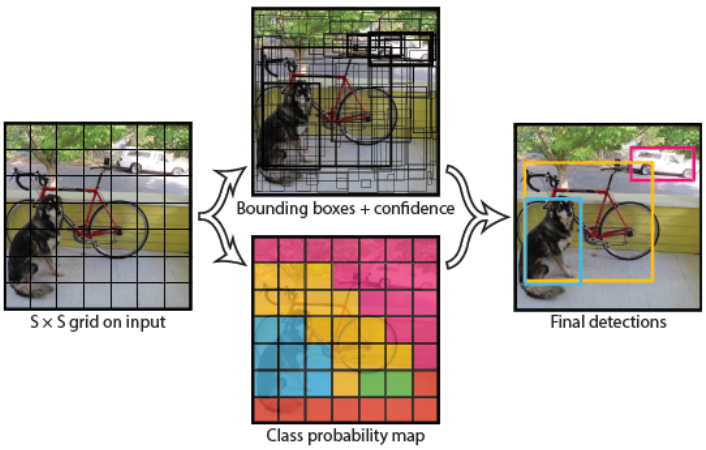


Figure 2.12: YOLO system overview. (Source: [31])

YOLO divides the input image into an $S \times S$ grid, as can be observed from Figure 2.12, where each grid cell is responsible for predicting the object whose center falls into it. Each cell will predict B bounding boxes along with their confidence score (one per box). A confidence score for a particular bounding box represents how confident the model is

that this box indeed contains an object but it also represents the accuracy of the box. In other words, confidence associated to a bounding box is defined as $P(\text{Object}) * \text{IoU}_{\text{pred}}^{\text{truth}}$; thus confidence should be zero if no object lives in that cell and otherwise it should be equal to the IoU between the predicted box and the ground truth box. Bounding box predictions are composed of 5 elements: the box center coordinates along with its width and height, plus a value representing the confidence associated with that box by the considered cell.

Each grid cell also predicts C conditional class probabilities, $P(C_i \mid \text{Object})$, i.e. the probability that the cell contains an object of class i given that the cell contains an object. At inference time, box confidence predictions are multiplied with conditional class probabilities in order to produce class-specific scores for each box. Such a score indicates the probability that the considered box contains the specific class along with the quality of the box.

YOLO is composed of convolutional layers combined with fully-connected layers, with no RoI pooling (since no region proposals are provided), hence a fixed-size image is required as input. The final fully-connected layers will yield bounding box coordinates along with class probabilities. YOLO has the advantage of reasoning globally about the input image when it has to make predictions, which is a main difference with region-based approaches. It is thus able to incorporate contextual information about the different classes during training.

The architecture used in this work is YOLOv5 [33], which brings slight improvements on YOLOv3 [34]. The latter namely adds merging of upsampled feature maps with feature maps obtained in previous layers in order to improve the quality of the resulting feature maps, which will contain more meaningful semantic information combined with finer-scale information. This merging process is used to predict bounding boxes at multiple scales. The overall idea of YOLO still remains the same, although each box will predict the classes it may contain. It also adds a new kind of data augmentation that should address the issue of detecting small objects by introducing mosaic augmentation, an augmentation technique that combines four images into four tiles scaled with a random ratio. In YOLOv5, anchor boxes are learnt automatically based on the provided training data, which is another difference with Faster R-CNN where anchor boxes have predefined scales and aspect ratios. This mechanism should help if the dataset contains objects with bounding boxes that are of very different aspects. Note that both Faster R-CNN and YOLO architectures rely on the use of a backbone as feature extracting module. Therefore, it will be possible to use previous classifiers as feature extractors.

Object detection metrics

An important quantity to evaluate object detection models is the Intersection over Union (IoU) threshold, which measures the quality of a predicted bounding box against the ground truth bounding box. In fact, it computes the ratio between the intersection of both bounding boxes over their union, and it therefore ranges in $[0, 1]$. An ideal object detector should thus only predict bounding boxes such that they have an IoU equal to 1 with the corresponding ground truth.

This threshold is then used to determine whether a predicted bounding box relates to any ground truth object in the input image. In general, a standard minimum value of 0.5 is considered to assign a bounding box to an object, but if the predicted box achieves

a higher IoU with another ground truth box, then it will rather be associated with the corresponding object.

For a given pair of ground truth and predicted bounding boxes, we can classify the prediction as a true positive or false positive. If classes for both bounding boxes match, then the prediction is considered as a true positive. Otherwise, it is considered as a false positive. Note that we can also have false positives in the case where the model predicted a bounding box but no ground truth box matched the prediction with an IoU higher than the minimum threshold value. Finally, we can have false negatives when ground truth objects have no associated predicted bounding box with a large enough IoU value.

We can thus compute usual precision and recall values and draw precision-recall curves for each class and for a particular IoU threshold. As was already explained, precision provides a measure of the confidence we can have in the model's predictions whilst recall characterizes the detection abilities of the model. With these metrics, we can compute an average precision metric (AP) for each class, which thus relates to the same IoU threshold as the one used to compute precision and recall curves. The average precision simply computes the area under the precision-recall curve, thus it computes the average precision value for recall values ranging in $[0, 1]$. Finally, we can compute a mean average precision metric (mAP), which is also defined for a specific (range of) IoU threshold(s). It consists of the mean of average precision scores across classes. Usual mAP metrics are computed for an IoU threshold equal to 0.5, denoted as mAP @.5, or averaged over several IoU thresholds, e.g. mAP @[.5, .95]. In the latter case, the mAP is computed for thresholds equal to $0.5, 0.55, \ldots, 0.95$ and these metrics are then averaged to yield mAP @[.5, .95].

2.1.3 Transfer learning

As was evoked in Chapter 1, a transfer learning approach will be considered for both tasks. Transfer learning can be defined [35], in all generality, with the help of two important notions: a domain $\mathcal{D}$ and a task $\mathcal{T}$. A domain $\mathcal{D}$ is defined through two characteristic components: a feature space $\mathcal{X}$ along with a marginal probability distribution $P(X)$ that is defined on $\mathcal{X}$. On the other hand, a task $\mathcal{T}$ is composed of a label set and of a function $f(\cdot)$ that predicts corresponding labels based on a given domain. This function $f(\cdot)$ is learnt from the training set. Given source and target domain-task pairs, denoted by $(\mathcal{D}_S, \mathcal{T}_S)$ and $(\mathcal{D}_T, \mathcal{T}_T)$ respectively, transfer learning is defined as a process that will boost the learning of the modeling function $f_T(\cdot)$ thanks to knowledge acquired in the source setting, where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$, i.e. the domains or the tasks are different.

In this work, the transfer learning process being used can be considered as an inductive transfer learning setting, where tasks are different and domains can be similar or different, and where labeled data is available for both the source and the target domains, although the amount of accessible data for the target domain can be much smaller than that of the source domain.

Applying transfer learning is desirable since usual machine learning methods make the assumption [35] that training and test data share the same feature space and distribution, which becomes untrue when facing such a task where there is a clear domain transfer, from the photo-realistic domain to the artistic domain. This approach is particularly helpful as it notably allows to train models on the target domain by reusing knowledge acquired in

the (larger) source domain. This therefore reduces the expensiveness of training and avoids training from scratch, for which it was demonstrated multiple times [36–38] that final performance was eventually worse than for models trained using transfer learning.

There exist different approaches to perform transfer learning for image classification and object detection using CNNs, among which fine-tuning, which has been applied in several related works [36–39] that highlighted the resulting increase in performance. Fine-tuning a CNN consists in reusing some part of the network, e.g. a block of convolutional layers that serve as feature extracting module, pre-trained on a source task/domain, and to continue training both the added blocks as well as this extracted module on a target task/domain. This approach will thus also be considered in the present work but other settings will also be studied for the detection task, such as freezing the reused component to capture potential stabilization effects.

2.2 Deep learning in Digital Humanities

In this section, we will cover the related work accomplished in the field of Digital Humanities. Firstly, we will review several classification tasks carried out in the art world. Then, we will cover multiple projects that tackle object detection in artworks in different angles.

2.2.1 Classification applied to artworks

In [40], the gap in domain shift between natural images and paintings is quantified by comparing the performance of standard classifier architectures when they are trained on either the first or the second type of pictures. Furthermore, they achieve higher classification performance when applying region-based classification on top of a Faster R-CNN detector (all trained on photo-realistic images) rather than simple image-level classification, which they attribute to the large amount of small depictions that are prevalent in paintings.

As concerns art style classification, authors in [41] investigated the use of standard classification architectures, such as ResNets, and showed that they also give good performance on artistic depictions, underlining the fact that models trained on natural images can be transferred between domains and still yield reasonable performance. However, they also demonstrate that network re-training is needed to achieve the best performance on this task, implying that features learnt from natural images are not optimal when it comes to artworks, which could have been expected. To support these results, authors relied on the Wikipaintings dataset [42], specifically designed for art style classification, which consists of 85 thousand paintings spread across 25 genres. In [43], authors proposed the augmentation of PASCAL VOC using style transfer as presented in [44] to improve classification performance on paintings, tackling the inherent lack of annotated artistic data. In addition, they experimented a fusion of two CNNs, one trained for object recognition on their augmented data and the other trained for style recognition, and achieved an improvement in mean average precision. Similar classification tasks were conducted in [36] using standard architectures either pre-trained on natural images or on large available paintings datasets.

Finally, in [11], a benchmark dataset called MINERVA is created for the detection of

musical instruments in artworks. A thorough analysis of state-of-the-art classification and detection models applied on this dataset is provided. More specifically, classifiers are pre-trained on the Rijksmuseum dataset [45] and fully fine-tuned to achieve the best downstream performance. It was shown [36] that starting from weights trained on this dataset could outperform models pre-trained on ImageNet only.

2.2.2 Detection applied to artworks

In [46], the author tested a simplified YOLO architecture to detect people, on a set of 218 cubist paintings [47] depicting people. The network was pre-trained on ImageNet and fine-tuned on PASCAL VOC. The author claims to achieve a 34% average precision on this task while on the PASCAL VOC dataset, YOLO reaches an average precision of 63.5% for the person class. However one could question the robustness of the obtained results since in [47] authors point out that the dataset is highly biased: indeed, paintings represent persons in portrait angles where the torso occupies most of the depiction area. Thus, precision rates will be positively biased because a random detection will more likely be treated as a true positive. Furthermore, this approach only detects a single class, hence it consists of a binary detection task. Still, this work underlines that reusing pre-trained detection frameworks while shifting domains can produce baseline results to be improved upon. Other works also tackle such a cross-depiction problem, as is conducted in [48] where they achieved 58% average precision on 41 different styles representing persons, by fine-tuning a Fast R-CNN architecture pre-trained on ImageNet. In addition, they show the influence of customizing the Intersection-Over-Union (IoU) lower bound for negative region proposals.

In [49], a weakly-supervised approach is proposed to tackle object detection in artworks. Using a multiple-instance learning (MIL) technique, they combine region proposals from a Faster R-CNN detector trained on natural images with a discrimination similar to what is done with SVM in order to determine which regions correspond to which object category. This allows the use of image-level annotations that are much more frequent than instance-level annotations, at least for the artistic domain. Furthermore, they demonstrate the applicability of their method to detect new classes, which are specific to artistic depictions and that cannot be found in photographs. Such a weakly-supervised approach represents an interesting path to follow for future work.

In [11], they demonstrate using their newly created MINERVA dataset the advantages of transfer learning from photographs to fine-art paintings when it comes to object detection, where several detection tasks of increasing complexity are carried out. They underline the natural drop in performance that can be observed for such tasks in this domain, due to the variety of depiction styles along with the inherent smaller amount of available samples. Indeed, for the detection problem, it appears that instruments with few depictions also achieve low average precision results. This piece of work will mainly serve as comparison tool for this work to relate results obtained on similar tasks but for different categories.

Chapter 3

Datasets and scope of research

This chapter will define in more details the line of work followed and will review the datasets that were initially considered. Each resulting task will be covered in subsequent chapters.

3.1 Methodology

This work is part of the INSIGHT project[1], whose end goal is to augment cultural heritage collections with more (refined) descriptive metadata. In particular, the aim is to apply transfer learning techniques to existing classification and detection architectures for fine-art paintings and other artworks, in order to study the impact on performance compared to what would be obtained with frameworks trained on photo-realistic images, and also to compare to existing work of similar philosophy [11].

The object detection problem will be split into two consecutive tasks:

1. Object classification in artworks
2. Object detection in artworks

Thus, the first part of this thesis, related to the application of AI to the field of digital humanities, will be devoted to classification, relying on standard DCNN architectures to solve the problem. As mentioned, transfer learning will be conducted to account for the domain shift. The second part will cover the full detection pipeline and will therefore also study the reuse of well-known frameworks for natural images.

3.2 Datasets

Before solving any of both tasks, it is needed to gather relevant datasets, which are provided through the INSIGHT project as an ensemble of digital assets coming from two museum clusters in Brussels: the Royal Museums of Fine Arts of Belgium and the Royal Museums of Art and History. Digitized artworks have been annotated within the Cytomine[2] [50] software, as was done for the MINERVA dataset in [11].

[1] https://hosting.uantwerpen.be/insight/index.php/about/
[2] https://cytomine.be/

3.2.1 Presentation of the datasets

Two different projects are created to deal with object detection: one dedicated to depictions of fruits and the other dedicated to depictions of animals. For each project, objects have been manually annotated with bounding boxes by volunteers and members of the project. A summary of the content of both projects is provided in Table 3.1. Note that

	Animals	Fruits
Number of paintings	8 234	4 685
Number of annotations	11 116	26 736
Number of classes	25	32

Table 3.1: Content summary for both datasets.

annotations are not uniformly spread across paintings: therefore, there exist paintings that have not been annotated.

Given that the annotations for the animals project were finished first, the focus was put on this project. However, all following methods should also be applicable for the fruits project.

3.2.2 Exploring the dataset

To highlight the discrepancy of artistic styles present in the dataset, a random subset of artworks is displayed in Figure 3.1. From this figure, we can discern a painting of a horse, but also a carving of a horse, drawings of other animals along with a coin depicting a chicken, although chickens do not appear in the list of animals to detect. This corroborates the previous remark stating that not all paintings were annotated: since they come from a mix of clusters, it is natural that this mix was not hand-curated, hence it will contain paintings that do not contain any animal or that contain non-related animals, and those will thus not be used. For example, some paintings will contain depictions of chickens, which do not belong to the 25 animal classes that should eventually be detected.

One can also draw a barplot representing frequencies of appearance for each animal class to get a first insight about the distribution of animals across paintings. This can give us an idea of which animals were regularly pictured and which others were rarely depicted. The corresponding barplot is represented in Figure 3.2. As can be observed from the barplot, there are three most occurring classes: domestic dogs, primates and horses. Other classes are less portrayed, some of them even having less than 200 appearances; the minimum being achieved by bats with a frequency of 146. The same behavior was observed in [11] where the distribution of musical instruments showed a long tail of rarely depicted objects, although the situation with animals is a little bit less severe.

As was underlined in Chapter 2, a major concern about artistic representations lies in the size of pictured objects. Indeed, small objects (and in this case, small animals) can be expected to be frequently sketched as part of the background, not necessarily intended for later computer vision reuse. We can arbitrarily decide with a patch threshold area whether an object is considered as small or not. Let us fix this threshold to 1 024, i.e. objects whose bounding box can be assimilated to a rectangle of dimensions 32×32 and let us rename these particular bounding boxes as *smallest bounding boxes*. This choice

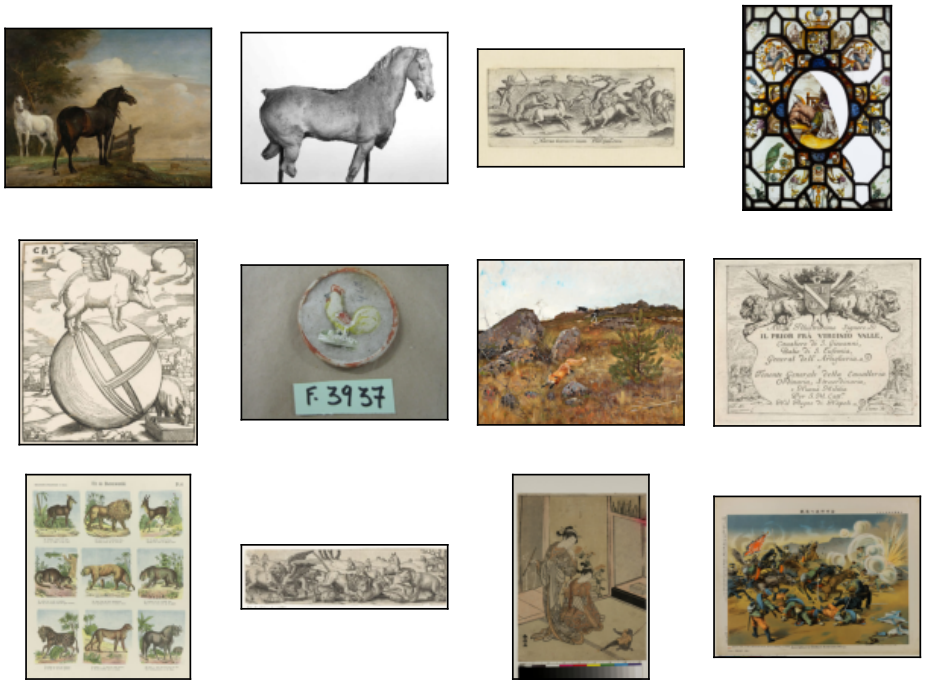


Figure 3.1: Random batch of paintings.

of threshold was motivated by the definition of small objects provided in the COCO evaluation guidelines[3]. For each animal, proportions (higher than 5%) of such small patches are reported in Table 3.2, along with proportions for patches of area smaller than 64×64. From this table, we can observe that some animals could be expected to have larger proportions of smallest bounding boxes than others: for instance, rats and mice are by definition small animals, hence it is natural to depict them in small and realistic proportions. The same remark holds for bats. Sheep, cows and squirrels have proportions around 20%, thus one can expect that every fifth occurrence of such animal will be rather small. For squirrels, this is most likely due to their size, greater than rodents but still smaller than most other animals. As concerns sheep and cows, this is probably due to paintings where they are pictured in the background of vast landscapes. If we now look at the proportions of boxes smaller than 64×64, it appears that animals with larger proportions of smallest boxes also have larger proportions of boxes whose area is in $]1\,024, 4\,096]$. Thus, we could intuitively expect the detection performance to be worse for those animals than for those that are not even reported in the table, e.g. tigers, leopards, lions, elephants, although this hypothesis should be checked when evaluating detection models. Some paintings corresponding to the smallest bounding boxes are displayed in Figure 3.3 and 3.4, for the six animals which have the highest proportions.

Paintings are downloaded in original dimensions. When retrieving paintings and annotations from the Cytomine platform, a general annotations file is constructed, a sample of which is shown in Table 3.3. Each row corresponds to a specific bounding box:

- `ID` represents the bounding box ID, provided by the platform, which also serves as file name to save the crop corresponding to that box,

[3] https://cocodataset.org/#detection-eval

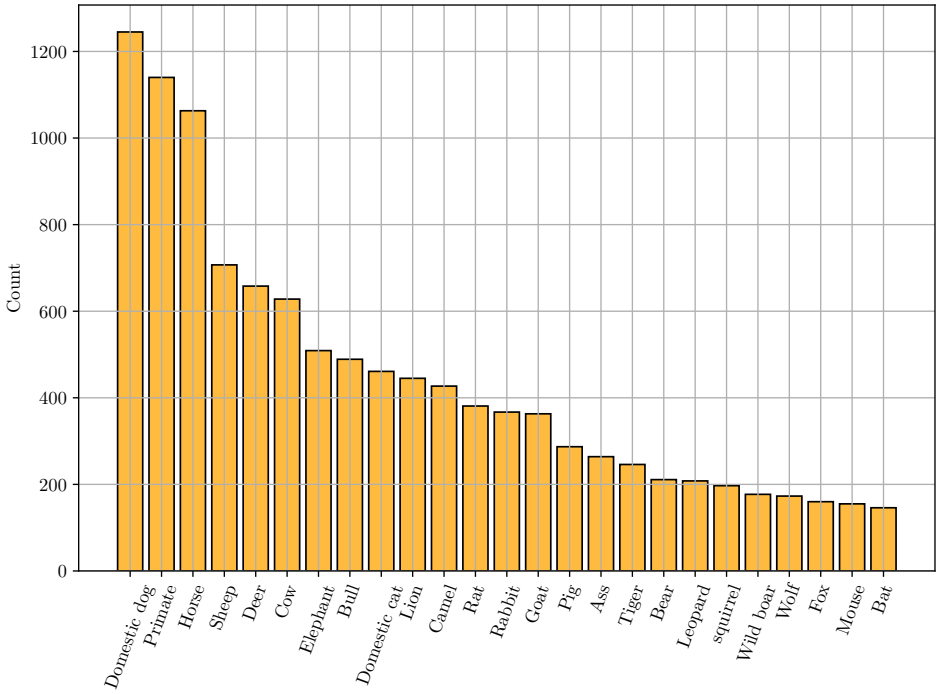


Figure 3.2: Barplot of the frequencies of appearance for the animals project.

- `Image` corresponds to the relative file name of the considered painting,
- `Project` represents the project ID on the platform (in this case the project ID will be identical for all boxes),
- `Term` is the animal label corresponding to the bounding box,
- `Area` and `Perimeter` correspond to the area and perimeter of the bounding box,
- (`xleft`, `ybottom`) and (`xright`, `ytop`) are the coordinates of the bottom-left and upper-right corners of the bounding box, relatively to the image. The origin of the coordinate system is defined as the bottom left corner of the image, however coordinates conventions might have to be transformed for later reuse in detection models.

For each subsequent task, this file will be post-processed to generate training, validation and testing sets. For the classification problem, only `ID` and `Term` are needed, while for the detection problem, `Image`, `Term` and all four coordinates are required.

	Area $\leq 1\,024$	Area $\leq 4\,096$
Rat	30.71%	65.09%
Mouse	28.39%	55.48%
Bat	25.34%	45.89%
Sheep	23.48%	41.58%
Cow	16.72%	44.90%
Squirrel	16.24%	56.34%
Domestic dog	15.26%	36.14%
Horse	9.41%	23.33%
Deer	9.27%	22.49%
Wild boar	8.47%	27.68%
Goat	7.71%	22.31%
Primate	6.75%	35.17%
Camel	6.32%	29.04%
Pig	6.27%	33.10%
Ass	6.06%	16.29%
Rabbit	5.99%	26.97%

Table 3.2: Proportions of bounding boxes whose area is smaller than a certain threshold, for each animal. Only animals whose proportions of smallest boxes is above 5% are reported.

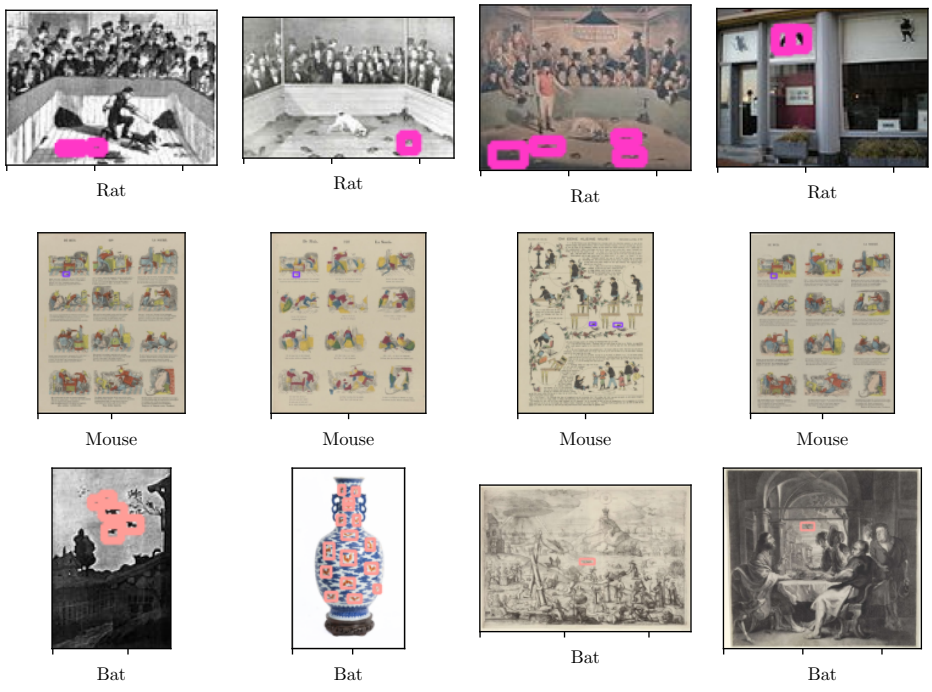


Figure 3.3: Paintings with the smallest bounding boxes, for rats, mice and bats.

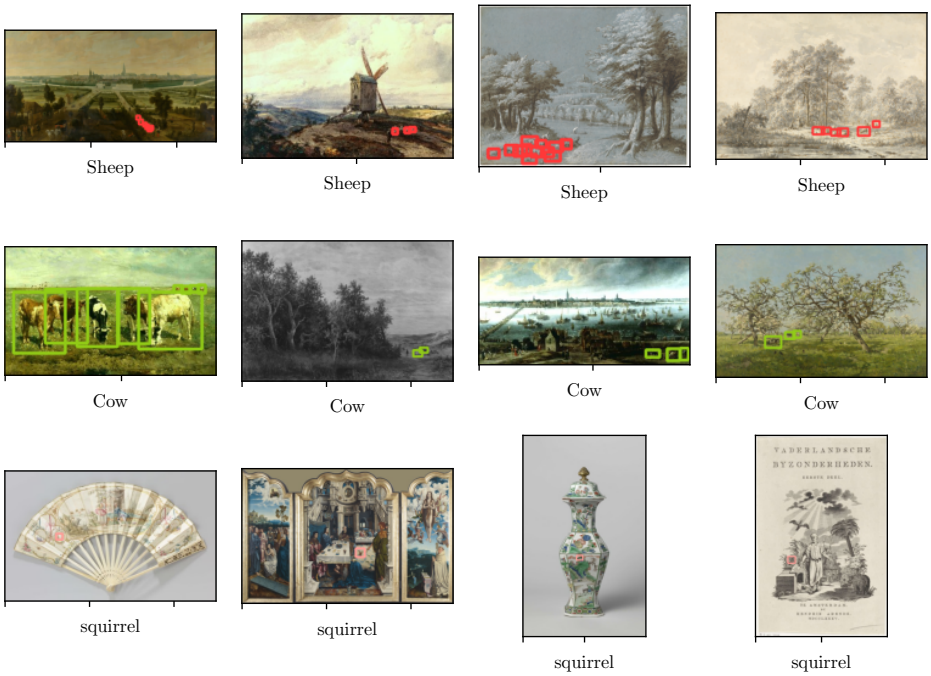


Figure 3.4: Paintings with the smallest bounding boxes, for sheep, cows and squirrels.

ID	Image	Project	Term	Area	Perimeter	xleft	ybottom	xright	ytop
7842721	7135974.jpg	5927070	Horse	33288.0	742.0	327.5	30.5	479.5	249.5
7842703	7135974.jpg	5927070	Horse	64052.0	1014.0	143.5	9.5	382.5	277.5
7842684	7135980.jpg	5927070	Domestic cat	77348.0	1122.0	77.5	17.5	394.5	261.5
7842664	7135989.jpg	5927070	Sheep	2904.0	224.0	131.24	41.07	202.46	81.84
7842647	7135989.jpg	5927070	Domestic dog	2375.0	195.0	73.8	58.3	125.5	104.24
7842628	7135989.jpg	5927070	Cow	7752.0	360.0	220.26	57.15	328.24	128.94
7842612	7135989.jpg	5927070	Cow	17046.0	528.0	275.97	49.68	428.18	161.68
7842590	7136169.jpg	5927070	Sheep	15160.0	531.0	9.51	180.63	191.5	263.93
7842575	7136169.jpg	5927070	Goat	18428.0	550.0	76.51	189.69	236.77	304.68
7842560	7136169.jpg	5927070	Cow	20810.0	578.0	11.32	233.15	148.94	384.35
7842541	7136169.jpg	5927070	Domestic dog	12986.0	473.0	398.84	118.16	485.76	267.55
7842526	7136169.jpg	5927070	Sheep	11776.0	436.0	234.05	49.35	354.47	147.13
7842506	7136169.jpg	5927070	Bull	50878.0	911.0	43.91	32.14	302.86	228.62
7842478	7136280.jpg	5927070	Horse	19964.0	566.0	31.56	132.27	166.96	279.72
7842461	7136280.jpg	5927070	Rabbit	2567.0	203.0	222.38	128.41	277.31	175.15
7842442	7136280.jpg	5927070	Lion	9433.0	390.0	341.39	140.94	448.36	229.12
7842424	7136280.jpg	5927070	Bull	16786.0	518.0	325.01	153.47	450.77	286.94
7842395	7136646.jpg	5927070	Goat	161320.0	1612.0	77.5	151.5	447.5	587.5
7842374	7136388.jpg	5927070	Domestic cat	116053.0	1388.0	72.5	11.5	485.5	292.5
7842351	7136220.jpg	5927070	Goat	6778.0	331.0	84.75	27.2	175.56	101.83

Table 3.3: A sample of annotations for the animals dataset.

Chapter 4

Classification

This chapter will detail the different steps that were conducted to deal with the first part of the work along with the results obtained for each of them. Section 4.1 details how the datasets are pre-processed to deal with the classification task. Section 4.2 covers the different steps carried out to test pre-trained architectures on painting crops as well as to improve their performance. A comparison with the results of [11] will also be provided. In Section 4.3, we will analyze and interpret the behavior of the classifier that achieved the best results. Finally, Section 4.4 will summarize and conclude the different experiments carried out to tackle the classification task.

4.1 Problem definition

As was defined in Chapter 2, the first task is the problem of classification, which in the scope of this work, consists in training a model to correctly associate bounding box patches with their appropriate animal. For that purpose, different standard classifier architectures and different training settings will be compared.

For each training setting, crops will be split into training, validation and testing sets according to the following proportions:

- Training set $\rightarrow$ 50%
- Validation set $\rightarrow$ 25%
- Testing set $\rightarrow$ 25%

Although it naturally decreases the amount of data on which the model can be trained, thereby reducing the learning potential, this split choice allows for a honest model tuning and a decent later performance assessment. Splits are performed using a stratified approach, i.e. all splits will keep similar proportions of each animal compared to the proportions in the whole dataset. In order to prevent information leakage from happening across splits, whenever a crop is included in a split, all other crops from the same painting will also be included in the same split. Thus, there might be small differences in the proportions of each animal across splits, although it should remain moderate thanks to the stratified approach. Note that this approach is inspired from [11], where an identical splitting strategy was adopted.

4.2 Applying pre-trained classifiers on paintings

In order to obtain an initial baseline, the classifiers defined above, pre-trained on ImageNet, can be directly evaluated on the paintings testing set to see whether or not they perform well with no further training. To some extent, it will give an idea of the gap between the photo-realistic domain and the artistic domain. Please note that, in this chapter, to simplify reading, crops extracted from paintings (and corresponding to specific animals) may be referred to as *paintings*, although they relate to sub-parts of the corresponding paintings.

4.2.1 Intersecting ImageNet labels with paintings classes

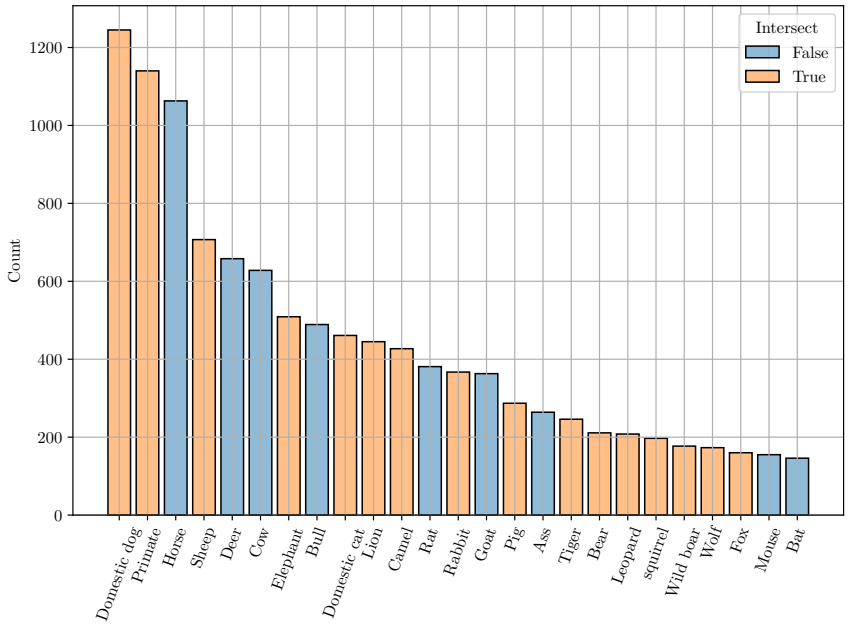


Figure 4.1: Barplot of the frequencies of appearance for the animals project, with relative information of label intersection.

Classifiers are trained for the 1 000-class ILSVRC classification sub-task, therefore the intersection between ImageNet labels and paintings has to be manually checked. The list of ImageNet labels can be easily found searching the web[1] and labels are checked one by one to see whether they correspond to any animal known in the 25 animal classes of the paintings. The resulting barplot is shown in Figure 4.1. As can be seen from the barplot, 9 animals have no intersecting pictures in the corresponding ImageNet dataset. Thus, two alternatives can be considered:

- Evaluate classifiers only on the intersection
- Construct a new dataset of natural images to span all 25 classes in order to train a new classification layer

While the first alternative is the simplest and most straightforward one, the second should not appear as impossible. Indeed, during the upstream annotation process, animals that

[1] https://image-net.org/challenges/LSVRC/2017/browse-synsets.php

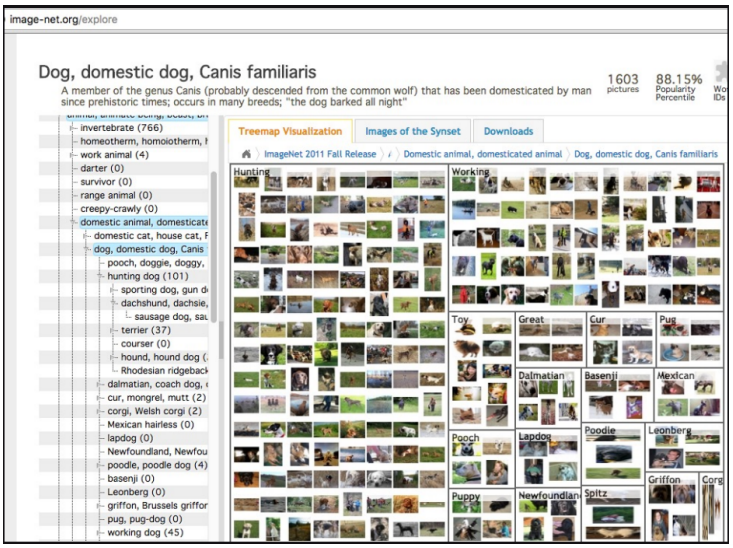


Figure 4.2: Sample tree-map structure for the class "Domestic dog". (Source: [51])

are eventually annotated were selected only if they had a direct correspondence with some node in the ImageNet tree hierarchy [52]. This hierarchy is constructed based on the WordNet structure [53], which is a massive English lexical database where words are gathered into concepts called *synsets*. A synset is a set of cognitive synonyms, and synsets are naturally organized in a hierarchical way based on their conceptual relationships with each other. For instance, "domestic dog" is a child node of "domestic animal", as can be seen from Figure 4.2. Note that not all synsets defined in WordNet have a correspondence with some ImageNet picture subset. Indeed, only nouns are currently indexed by ImageNet.

4.2.2 Construction of a custom ImageNet dataset

Each synset is identified by a specific WordNet ID, thus each animal class present in the paintings has a one-to-one mapping with such IDs. Therefore, it should be possible to download natural images related to each class *provided that* an access to ImageNet pictures is available. There exist Python softwares[2] that allow to download (a certain amount of) photographs from ImageNet when provided with WordNet IDs corresponding to the classes of interest. Correspondence between these IDs and plain English words is also given.

After listing all interesting WordNet IDs, the software is launched to retrieve around 1 000 images for each animal class. Coupled with data augmentation techniques like mirrorring, equalizing, brightness modification, etc., such an amount would give reasonable quantities of training images. Note that, for most synsets, less than 1 000 pictures could be eventually recovered. Therefore, those classes needed completion with child synsets. For example, "wild boar", identified by WordNet ID `n02396427`, was completed with "boar" (`n02396014`) and "warthog" (`n02397096`), although some noticeable visual differences exist between both species. Visual similarity between different species of a same animal was tried to be kept as much as possible during this image retrieving process.

When a sufficient amount of pictures is gathered for every animal out of the 25 different

[2] https://github.com/mf1024/ImageNet-datasets-downloader

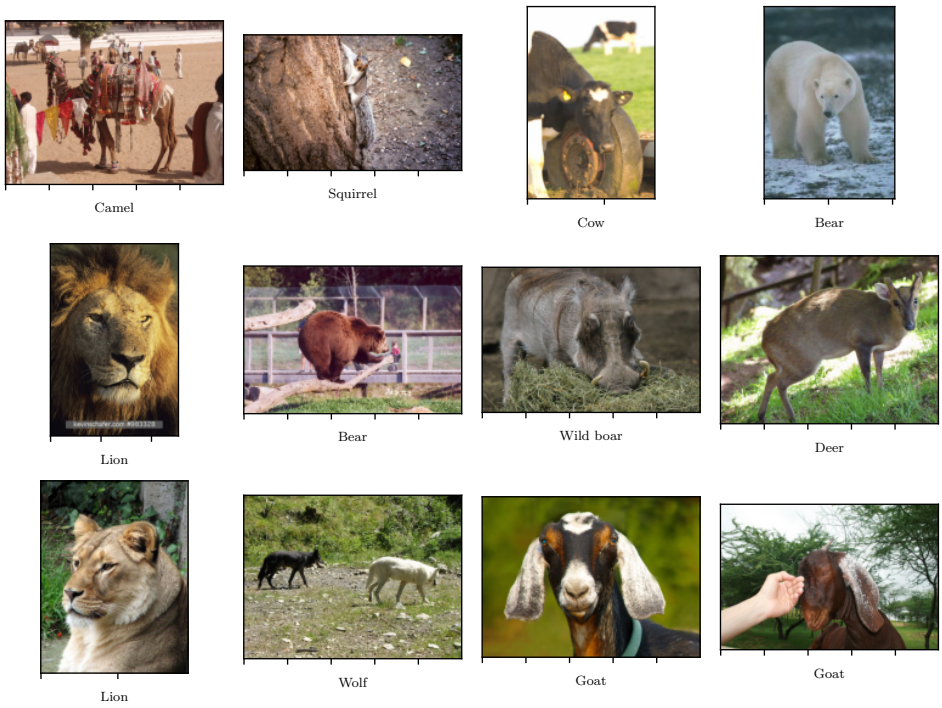


Figure 4.3: Random batch of ImageNet animal pictures.

classes, a data cleaning step is essential as there is a non zero probability that some hosted images no longer exist, thereby resulting in dead images, but not in dead URLs. In addition, there are also images of unrelated animals that have slipped in some other animal subsets that must be manually moved to the appropriate class or simply removed. Finally, the last step is to equalize the number of images for each animal. A random batch of examples is provided for visualization in Figure 4.3. As can be seen from this batch, there will be occurrences of multiple animals appearing in the same picture, which is the case for wolves but it will also be the case for other animals that live in a kind of pack, e.g. cows, sheep, etc.

4.2.3 Fine-tuning classifiers on ImageNet

With the previously constructed ImageNet animals-only dataset, all three classifiers can be trained to incorporate classes outside of the intersection. A similar dataset splitting approach is considered, but proportions are slightly modified: the testing set still represents 25% of the dataset but the validation set is now smaller with a proportion of 15% and the training set thus represents 65% of the data.

The final linear layer with 1 000 output neurons is replaced by a linear layer with 25 output neurons and the whole architecture is fine-tuned on this custom dataset. As can be observed from Figure 4.4, models easily converge on the training set and quickly saturate on the validation set. The early stopping criterion is based on validation loss, computed at each epoch, and training is halted whenever validation loss no longer improves after 10 consecutive epochs. The final (optimal) model is the one that achieves the lowest validation loss.

From this figure, we can also observe that all three models need very few epochs to

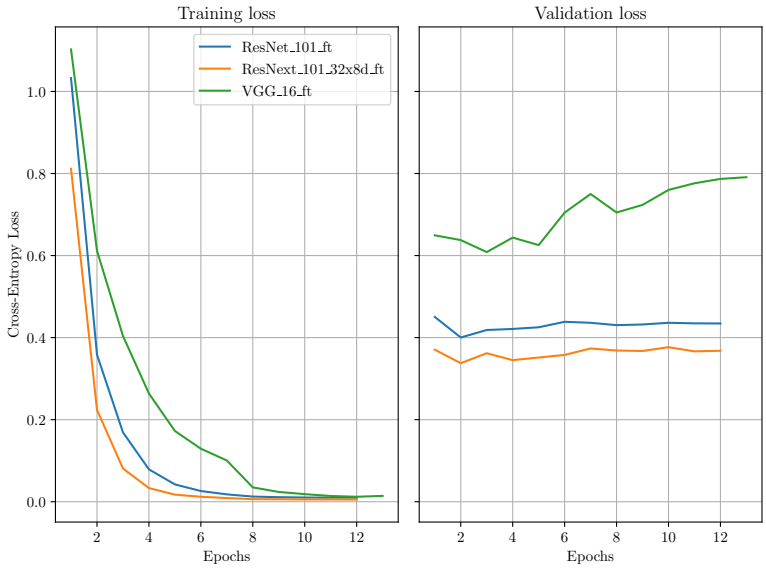


Figure 4.4: Training and validation losses for all three models, fine-tuned on ImageNet animal pictures.

	Top-1 accuracy (animals)	Top-1 accuracy (ImageNet)
VGG-16	81.95%	71.59%
ResNet-101	87.80%	77.37%
ResNeXt-101 32×8d	90.10%	79.31%

Table 4.1: Performance on the ImageNet animals-only test set and on the 1-K ImageNet test set.

integrate new classes: both ResNe(X)t architectures only need two while VGG-16 needs three epochs to reach its lowest validation loss.

Performance on the test set is reported in Table 4.1, along with results of each classifier on the 1-K ImageNet test set[3]. Although both quantities are not directly comparable since the second task is harder (25 classes against 1 000 classes), it is worth noting that fine-tuned classifiers do not perform worse on the simpler task. In addition, as could be expected due to the temporal ordering of architectures development, the ResNeXt framework achieves the highest accuracy with an improvement of 2.30%. Top-1 accuracy per class is displayed in Figure 4.5. From this figure, we can observe that ResNeXt achieves the highest performance for almost all classes. Bulls seem harder to classify and it could have an impact on later detection performance. This might be due to the resemblance between bulls and cows, the latter reaching an accuracy 20% higher than the former. A similar scenario can be imagined between mice and rats, as far as VGG-16 is concerned. We can also notice that the three classes for which ResNe(X)ts achieve highest performance are tigers, lions and leopards. This can be intuitively explained by the specific visual features that these animals present: for instance, tigers and leopards have very typical and recognizable coatings, while lions have a typical mane, also easily distinguishable.

[3] https://pytorch.org/vision/stable/models.html

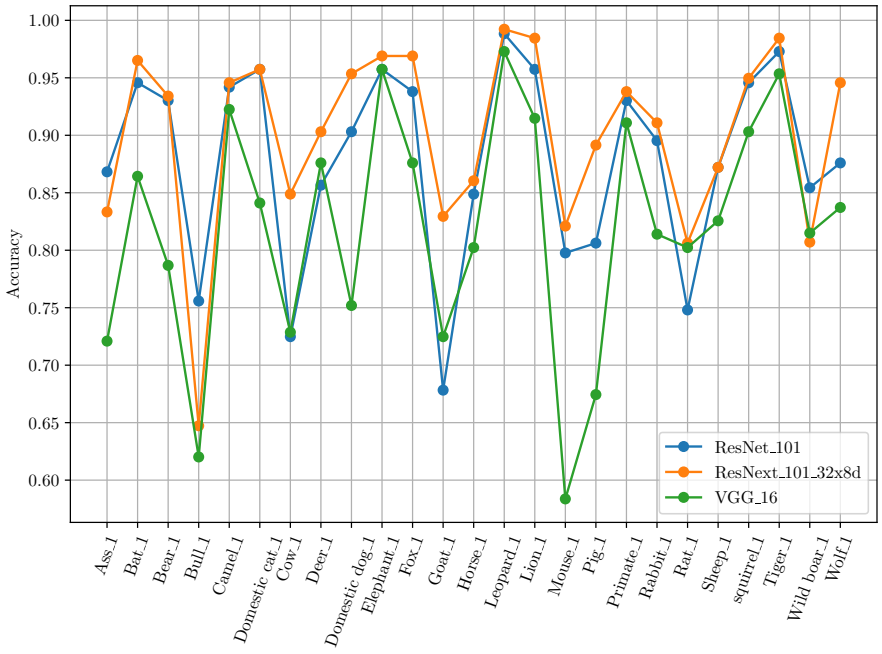


Figure 4.5: Top-1 per class accuracy on the ImageNet animals-only test set.

4.2.4 Evaluation of pre-trained classifiers

With all these steps done, classifiers can now be evaluated on animal crops extracted from the paintings, using the test set proportions defined in Section 4.1. Results for models used *as is*, i.e. with no further training, are available in Table 4.2.

	Top-1 accuracy
VGG-16	20.56%
ResNet-101	29.78%
ResNeXt-101 32 × 8d	37.24%

Table 4.2: Performance of classifiers pre-trained on ImageNet and fine-tuned on ImageNet (animals-only), on the paintings test set.

Similarly to the previous situation, ResNeXt achieves the highest test accuracy, outperforming its ResNet counterpart by 7.46%. VGG-16 shows a significantly worse performance with a little less than 17% accuracy difference. This gap of performance could be expected from the results obtained in Table 4.1 where VGG-16 achieves a top-1 accuracy gap of almost 9% with respect to ResNeXt. However, it should be noted that the gap has increased with domain transfer.

Per class accuracy metrics can be visualized in Figure 4.6 and can be compared to Figure 4.5, i.e. those obtained on natural images. As a first observation, it appears clear that the trend is completely reversed. Indeed, accuracies are now at the lower end rather than the higher end, except for three classes: bats, leopards and tigers, as all can be classified with a satisfying performance, whatever the domain. This could be expected given that leopards and tigers have very distinguishable and recognizable coats. As concerns bats, since they are the only flying animals in the dataset, it is possible that the model takes advantage

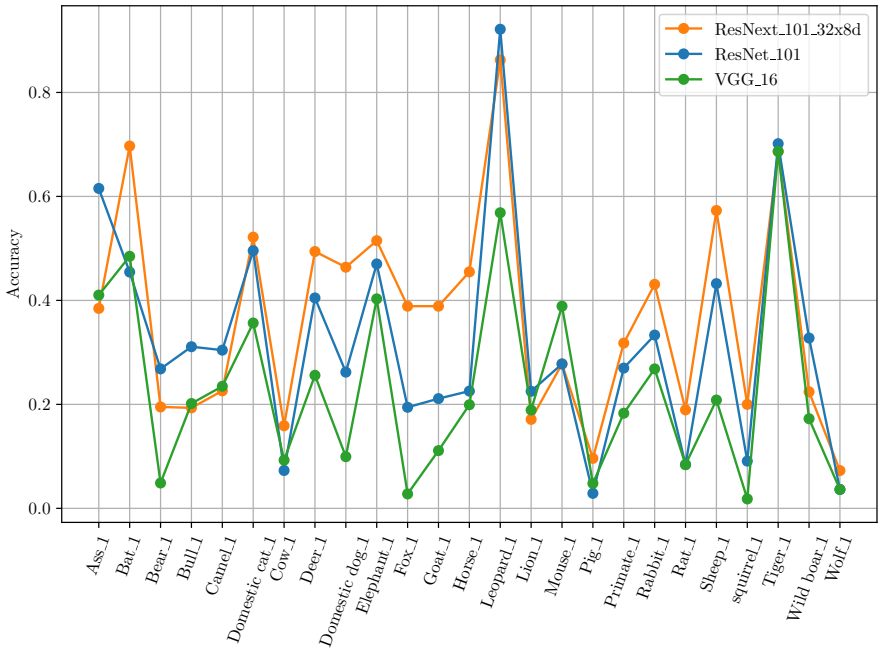


Figure 4.6: Top-1 per class accuracy on the paintings test set, for classifiers pre-trained on the 1-K ImageNet and ImageNet animals-only datasets.

from some background information related to the sky. In addition, bats generally have a common posture at rest but also when they are pictured while flying.

4.2.5 Fine-tuning classifiers on paintings

With the results obtained using pre-trained classifiers without additional training, one can only be disappointed but such a drop in performance could be expected due to the higher diversity of styles and shades spread across paintings, which is not the case for natural images. Therefore, all three models will be fully fine-tuned on paintings to account for domain transfer.

On the other hand, several baselines will be derived from the same architectures in order to compare this transfer learning technique with alternative standard techniques, such as:

- Re-training a new final classification layer, while keeping the rest of the pre-trained network unchanged,
- Training a classifier from scratch on paintings only,
- Training a classifier from scratch on ImageNet animal pictures only[4],
- Training a classifier from scratch on a mix of ImageNet animal pictures and paintings.

For all these scenarios, data augmentation will be considered in order to increase the amount of diverse data fed to the model during training. Among others, an input im-

[4]This setting differs from classifiers evaluated previously since the latter are trained on the 1-K ImageNet dataset and then fine-tuned on ImageNet animal pictures.

age could be augmented through mirroring (horizontal flipping), equalization, brightness increase/reduction, sharpness increase/reduction, etc. The input will be augmented with a probability of 0.5. Paintings crops will be resized to a standard image classification input dimension, i.e. 224×224, and will be further normalized before being processed into the convolutional neural networks, using mean and standard deviation vectors obtained for the general 1-K ImageNet training set, i.e. $\mu = [0.485, 0.456, 0.406]$ and $\sigma = [0.229, 0.224, 0.225]$. All architectures will be trained using stochastic gradient descent, with an initial learning rate $\eta = 0.001$ combined with a learning rate decay with a factor equal to 0.1, every 7 epochs. Using a learning rate scheduler has shown [54] improved optimization and generalization performance. A batch size of 32 crops will be considered for both training and validation.

For the mix of natural images and paintings, a ratio of 50% of paintings will be considered. Note that the two other *from scratch* baselines also represent a kind of mix, with ratios of 100% and 0% respectively. If this approach yields interesting and encouraging results, further exploration of the optimal mix might be considered. Re-training only the last linear layer on paintings could be expected to produce worse results than fully fine-tuning the networks but should at least produce improved results compared to classifiers pre-trained on 1-K ImageNet only. Note that the mix of pictures and paintings only happens for the training set. Indeed, all approaches share the same validation set, which contains only paintings. A more standard approach would have been to construct specific validation sets for each mix, i.e. a validation set with 50% of natural images and 50% of paintings for the 50%-mix and a validation set containing only photo-realistic pictures of animals for the mix based only on ImageNet. However, since the end goal is to generalize on paintings strictly, a validation set purely composed of paintings is considered, as it will allow to pick as final (optimal) model the model that performs best on paintings, and not on natural images/a mix of paintings and natural images. The test set is naturally only composed of paintings, as its goal is to assess the performance considering the artistic domain only, and it is also shared across all settings.

Last layer re-training vs full fine-tuning

Training and validation losses are provided for both fully fine-tuned and last layer re-training settings in Figure 4.7, and training loss along with top-1 validation accuracy are displayed in Figure 4.8.

From these figures, it appears clear that fully fine-tuning the architectures allows to bring training losses further down than what is achieved when re-training only the last layer on paintings. The same can be noted for validation losses although the drop is much more moderate but *optimal* fully fine-tuned architectures (i.e. architectures obtained at the epoch corresponding to lowest validation loss) still reach lower values of validation loss compared to their last layer setting counterpart. In addition, it is worth noting that both VGG networks achieve higher losses, whatever the considered setting, and that ResNeXt architectures achieve the lowest losses for both training situations.

As far as validation accuracy is concerned, similar observations hold. Indeed, VGG frameworks achieve the lowest accuracies for each setting while ResNeXt models still outperform the others, although it seems more modest for the last layer re-training setting and the gap between ResNets and ResNeXts is rather small. Furthermore, for the last layer setting, all three architectures achieve rather identical validation accuracy even though

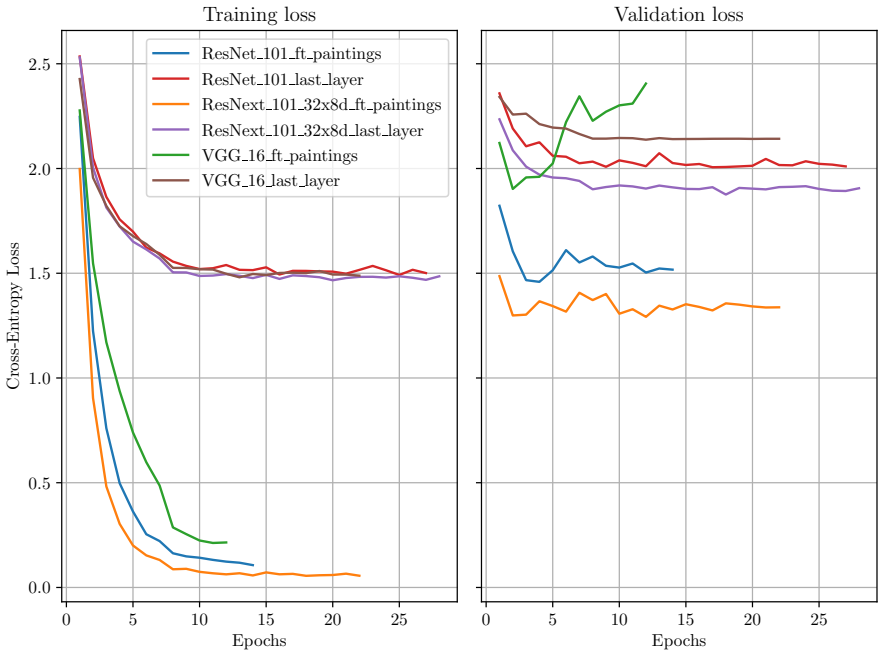


Figure 4.7: Training and validation losses comparison for full fine-tuning and last layer re-training.

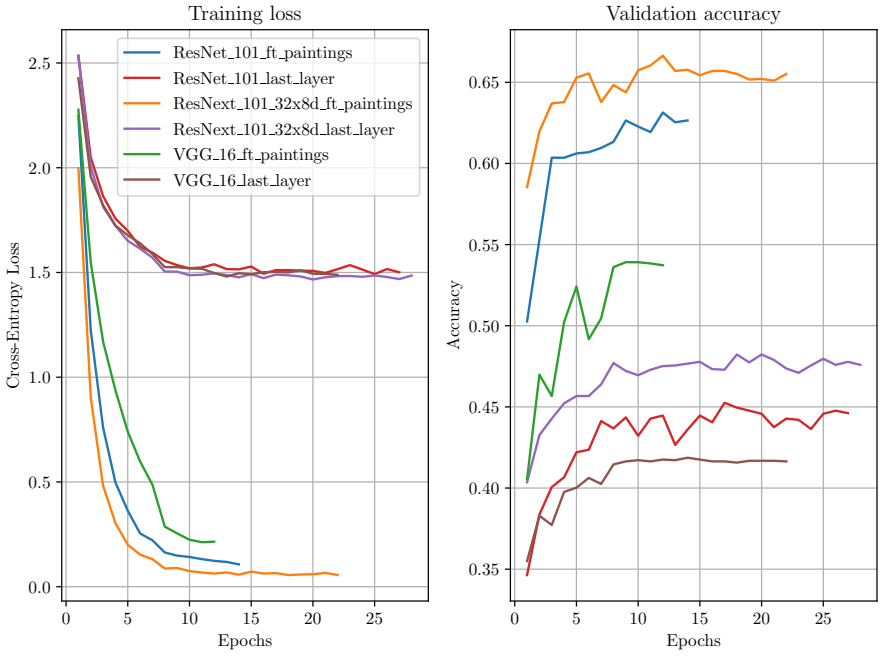


Figure 4.8: Training loss and validation accuracy comparison for full fine-tuning and last layer re-training.

ResNeXt outperforms both. It should be noted that the VGG model chosen as optimal for the fine-tuning setting does not achieve the highest validation accuracy, as the latter seems to keep increasing after the optimal epoch, even though an asymptotic behavior already becomes visible around the 10th epoch. Minimizing validation loss, in this case cross-entropy loss, does not imply the maximization of validation accuracy, even if both are related. Indeed, in a scenario with imbalanced data, accuracy could be improved by assigning the majority class to each input sample with the counter effect of worsening the underlying probability distribution learnt by the model. For this reason, optimizing with validation loss is preferred as the resulting model should be more robust.

These observations line up with previous results derived for each classifier. ResNeXt achieves (slightly) better performance compared to its ResNet counterpart, while VGG is lagging behind. However, these are results obtained on validation data. For honest and thorough assessment, the test set will be considered.

Training classifiers from scratch

The same learning curves can be derived for models trained from scratch on the three aforementioned mixes, i.e. only paintings (ratio of 1), only ImageNet animal pictures (ratio of 0) and a perfect mix of both (ratio of 0.5). Curves are displayed in Figure 4.9 and 4.10 for a comparison of losses and loss-accuracy respectively.

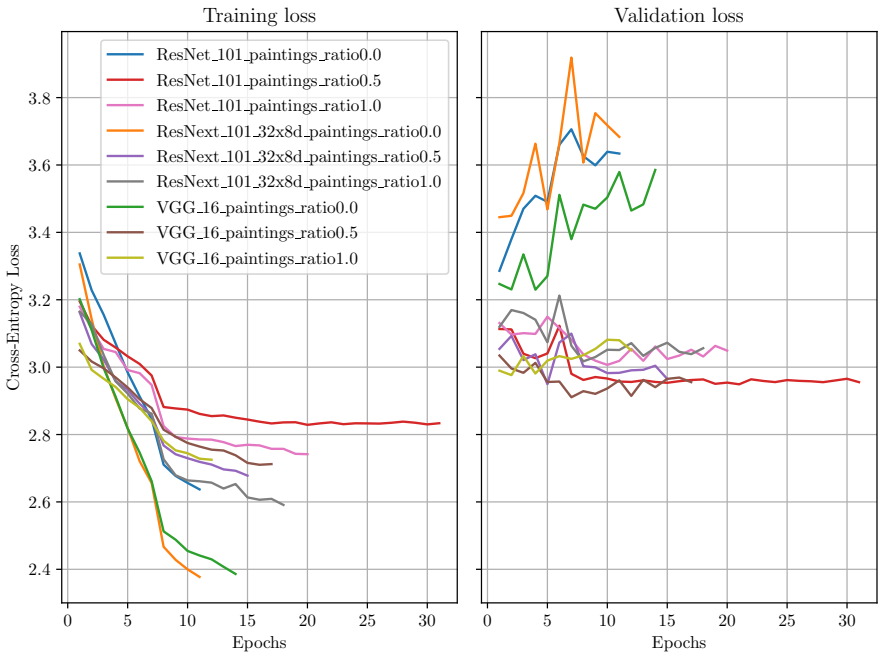


Figure 4.9: Training and validation losses comparison for three mixes.

With a first glance at these figures, it appears clear that training does not happen as smoothly and as good as in previous settings. Indeed, from Figure 4.9, we can notice that most training losses saturate above loss values of 2, while previous settings lead to losses tending towards 0 (for the fine-tuning setting) or towards 1.5 for last layer re-training. The same behavior can be observed for validation losses. In addition, when architectures are trained from scratch on ImageNet animal pictures only, training is stopped much

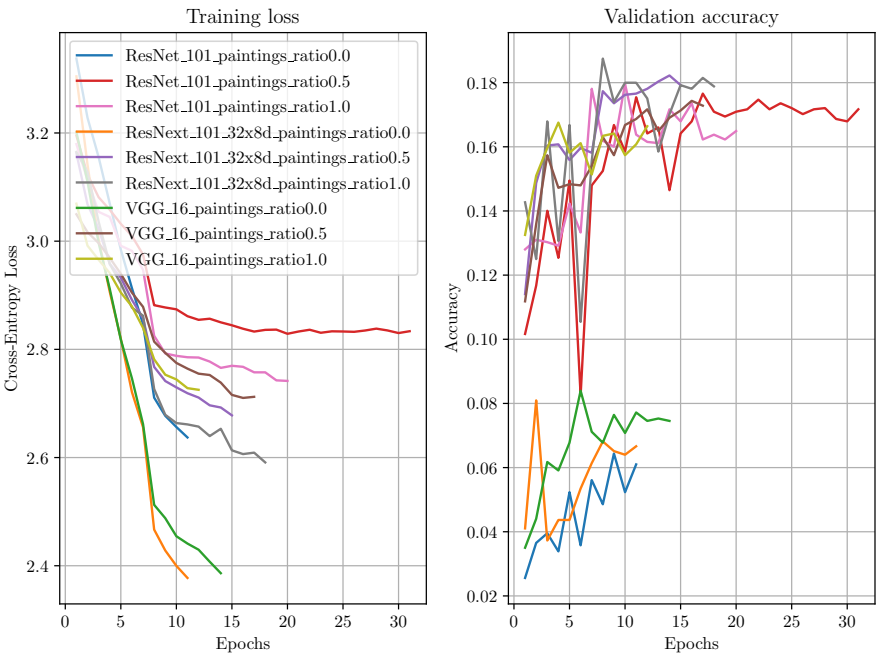


Figure 4.10: Training loss and validation accuracy comparison for three mixes.

earlier than for the two other mix scenarios because the corresponding validation losses diverge after a few epochs. When trained on paintings only, training converges after roughly 20 epochs (hence the best model is achieved around the 10th epoch) while the 50%-mix of natural images and paintings takes more epochs to converge as far as ResNet is concerned.

The scenario is similar for validation accuracy. Indeed, models trained exclusively on natural images of animals achieve a much smaller validation accuracy than when paintings are included in the training set. However, using 50% of paintings or strictly using paintings for training does not seem to have a visible impact on validation accuracy, as can be seen from Figure 4.10 where all six models reach comparable levels of accuracy. Considering the validation accuracy in Figure 4.8 and 4.10, it appears that training classifiers from scratch eventually yields worse performance, whatever the mix. Indeed, pre-trained models (fine-tuned or with their last layer re-trained) reach a validation accuracy above 50% while models trained from scratch seem to saturate below 20%.

4.2.6 Assessing final performance

All training settings can now be compared using the test set in order to see whether previous observations also apply in a more general sample. Results can be found in Table 4.3. For each scenario, finest results are highlighted in bold.

A first observation is that results obtained on the test set mostly align with those obtained on the validation set, as far as model ordering is concerned. In fact, ResNeXt outperforms other architectures for most settings, except when models are trained from scratch. It should be noted that generalization is acceptable, although a performance drop is remarkable for all settings. For instance, validation accuracy seemed to approach values near 70% for fully fine-tuned ResNe(X)ts, as can be observed from Figure 4.8, while test set

	ResNeXt-101 32 × 8d	ResNet-101	VGG-16
ImageNet animals only	2.72%	2.07%	**5.08**%
50% mix	14.25%	**20.38**%	16.58%
Paintings only	19.67%	19.55%	**19.96**%
Pre-trained on 1-K ImageNet	**37.24**%	29.78%	20.56%
Last layer	**45.94**%	44.03%	38.89%
Fine-tuning	**65.01**%	56.74%	45.94%

Table 4.3: Performance on the paintings test set for different settings..

accuracy reaches values closer to 60%, even though ResNeXt reaches a peak at 65%.

As was expected from the learning curves, architectures trained from scratch perform way worse than pre-trained architectures, fine-tuned or not. The best performance for such models is achieved by ResNet-101 with a top-1 accuracy of 20.38% when being trained on the 50% mix. Relying on paintings only yields models with similar performance, however models trained from scratch on the ImageNet animals-only dataset show very poor performance, with a top-1 accuracy sligthly above 5%, which is even lower than the performance achieved by architectures pre-trained on the 1-K ImageNet dataset (and fine-tuned on ImageNet animals). Still, models pre-trained on ImageNet perform better than models trained strictly on ImageNet animal pictures for a reason: the latter are trained on no more than 25 000 photographs of animals while the former are pre-trained on more than 1 000 000 natural images and then fine-tuned using the same animals-only training set. It is thus natural that these models achieve higher accuracy.

Re-training only the last layer of classifiers already yields significant improvements compared to results for standard pre-trained models, with increases ranging from 8% to 18% for VGG-16. Furthermore, given the relatively small amount of data, training such settings only takes a relatively small amount of time compared to trainings performed on the 1-K ImageNet.

Finally, fine-tuning completely the classifier framework yields the highest accuracy, whatever the considered model. The number of epochs required to fine-tune the models up to optimal performance is very small, which means that it only takes a small amount of time to improve accuracy by roughly 20%. ResNeXt achieves, as expected, the finest accuracy, with a value of 65% (corresponding to an improvement by 28% compared to its photo-realistic-pre-trained counterpart) and ResNet follows with a top-1 accuracy of 56.74%, which corresponds to an improvement of 27%[5]. VGG-16 eventually reaches an accuracy of 45.94%, which represents an increase of roughly 25%. It is worth noting that the gain in performance between the last-layer and fine-tuning settings is not as pronounced for VGG-16 as it is for ResNe(X)t models, where both improve by another 20% and 12% compared to 7% for VGG-16.

As a measure of comparison, in [11], classification accuracy for musical instruments extracted from paintings achieves a top-1 accuracy of 73.66% on the corresponding test set. However, this result is obtained for a classification task covering only 5 instruments and

[5]Please note that improvements are quantified through absolute differences in accuracy and are thus not expressed in terms of fractions of reference accuracy. For instance, ResNeXt improves its accuracy with fine-tuning by an amount of 27.77%, which corresponds to a gain of 75%.

is thereby easier than the current classification task which deals with 25 classes. Another experiment was run considering the top-20 instruments, which is thus quite comparable to the present task, although some of the considered instrument classes have frequencies of appearance which are relatively small with around 200 occurrences. Still, performance remains comparable. For this 20-instrument task, a top-1 accuracy of 36.51% is achieved by the best model (an Inception-V3 architecture), which lies behind present results obtained for animal classification using fully fine-tuned models. Note that, in their work, authors started from architectures trained on the Rijksmuseum collection instead of on the 1-K ImageNet dataset.

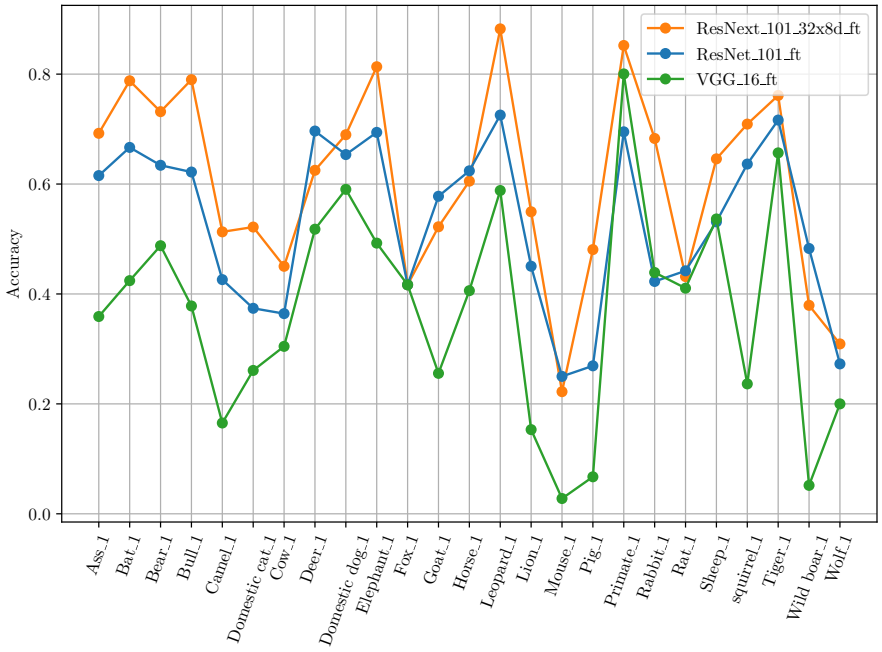


Figure 4.11: Top-1 per class accuracy on the paintings test set, for the full fine-tuning setting.

A general measure of per-class accuracy is provided in Figure 4.11. The performance gap between each of the three models is clearly visible from this figure, for almost all animals. This figure can be put in contrast with Figure 4.6 to visualize for which classes models improved the most, but also to see for which classes models could not be refined. To ease the comparison, a simple summary of the comparison between both figures is provided in Table 4.4, where the mean improvement is measured across all three classifiers. Please note that this represents a mean trend; it is therefore possible that a certain model improved its performance on a class while the mean improvement (across all three models) for that class is negative, or not as significantly represented. For instance, models improved their performance for foxes on average by 21.30% whereas ResNeXt only shows an improvement of 2.78% for that class. However, information from Table 4.4 should be combined with information displayed in Figure 4.6: there, it appears clear that ResNeXt's improvement for foxes is indeed very small, but it also appears that this particular classifier already performed much better with an accuracy around 40%, while the two others only reached at most 20%. In general, models agree for classes which show the highest improvements, such as primates, bears and squirrels. In addition, they also line up as far as performance decrease is considered since all three decreased (or only very slightly increased) for animals

	ResNext-101 32 × 8d	ResNet-101	VGG-16	Mean
Primate	53.42	42.51	61.74	52.56
Bear	53.66	36.59	43.90	44.72
Squirrel	50.91	54.55	21.82	42.43
Domestic dog	22.59	39.16	49.10	36.95
Bull	59.66	31.09	17.65	36.13
Rat	24.21	35.79	32.63	30.88
Cow	29.14	29.14	21.19	26.49
Horse	15.04	39.85	20.68	25.19
Deer	13.10	29.17	26.19	22.82
Goat	13.33	36.67	14.44	21.48
Pig	38.46	24.04	1.92	21.47
Fox	2.78	22.22	38.89	21.30
Wolf	23.64	23.64	16.36	21.21
Elephant	29.85	22.39	8.96	20.40
Lion	37.84	22.52	−3.60	18.92
Rabbit	25.20	8.94	17.07	17.07
Sheep	7.29	9.90	32.81	16.67
Camel	28.70	12.17	−6.96	11.30
Ass	30.77	0.00	−5.13	8.55
Bat	9.09	21.21	−6.06	8.08
Wild boar	15.52	15.52	−12.07	6.32
Tiger	7.46	1.49	−2.99	1.99
Leopard	1.96	−19.61	1.96	−5.23
Domestic cat	0.00	−12.17	−9.57	−7.25
Mouse	−5.56	−2.78	−36.11	−14.82

Table 4.4: Individual and mean improvements in per-class accuracy between the full fine-tuning setting and the standard pre-trained models. The mean is computed across models.

like leopards, cats and mice, the latter showing the highest drop. A possible explanation for these declines might be linked to potential confusions between animal classes. For example, mice have the highest reductions of accuracy, but they could very likely be confused with related animals such as rats which show an accuracy growth.

Some animals should intuitively be more easily classified than others, e.g. leopards and tigers which have recognizable coats, either with black dots or black stripes. Nevertheless, this hypothesis cannot be validated for leopards, as ResNet shows a drop of performance. As far as tigers are concerned, moderate improvement is achieved by these two models, but it is important to notice in Figure 4.6 that tigers (and leopards) achieved the highest accuracy among all animals, therefore the improvement can only be moderate but it does not explain ResNet's drop for leopards. A reason for the performance drop of leopards could be that styles differ across the training/validation and test sets, thereby underlying the generalization problem of state-of-the-art classifiers. Leopard samples for both the

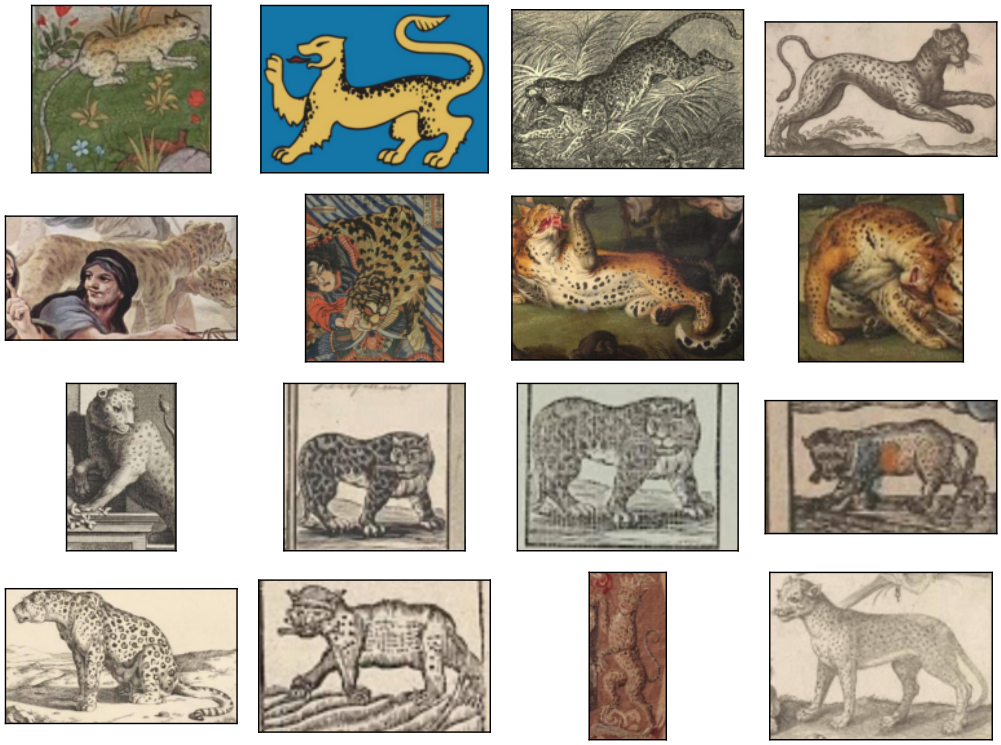


Figure 4.12: Training (first two rows) and test (last two rows) samples for leopards.

training and test splits are represented in Figure 4.12. From these samples, it seems that test samples mostly contain leopard depictions carried out with a pencil, with rather black and white shades, while training samples present a higher diversity of illustrations, among which pencil drawings but also colorful paintings.

A comparison of the per class performance of ResNeXt for the different transfer learning settings considered is depicted in Figure 4.13. From this figure, we can notice that the fine-tuning setting outperforms the other approaches for all classes, except for two of them: mice and rats. Mice could be expected since we highlighted its performance decrease in Table 4.4, which compares performance for fine-tuned classifiers against their pre-trained counterparts. As concerns rats, both last layer and fine-tuning settings outperform the pre-trained version of ResNeXt, but the last layer re-training performs slightly better than the fine-tuning one. If we now only compare the last layer setting with pre-trained classifier performance, we can see that the former improves on the latter but this enhancement is not uniform, as it performs worse than the latter for some animals. In any case, the fine-tuning setting should be preferred as is illustrated in Figure 4.13, since it allows to incorporate features specific to paintings into lower layers, which is not the case for the last layer setting as those layers remain frozen.

As a final step, the fine-tuned ResNeXt classifier is tested back on the ImageNet animals-only dataset to see how it compares to its pre-trained version. This comparison is represented in Figure 4.14. From this figure, we can observe that the pre-trained version consistently outperforms its fine-tuned counterpart. This could be expected since the former was only optimized with natural images, therefore it has only constructed feature representations for natural images. On the contrary, the latter has modified its feature representations to embed artistic features, which do not necessarily correspond to any

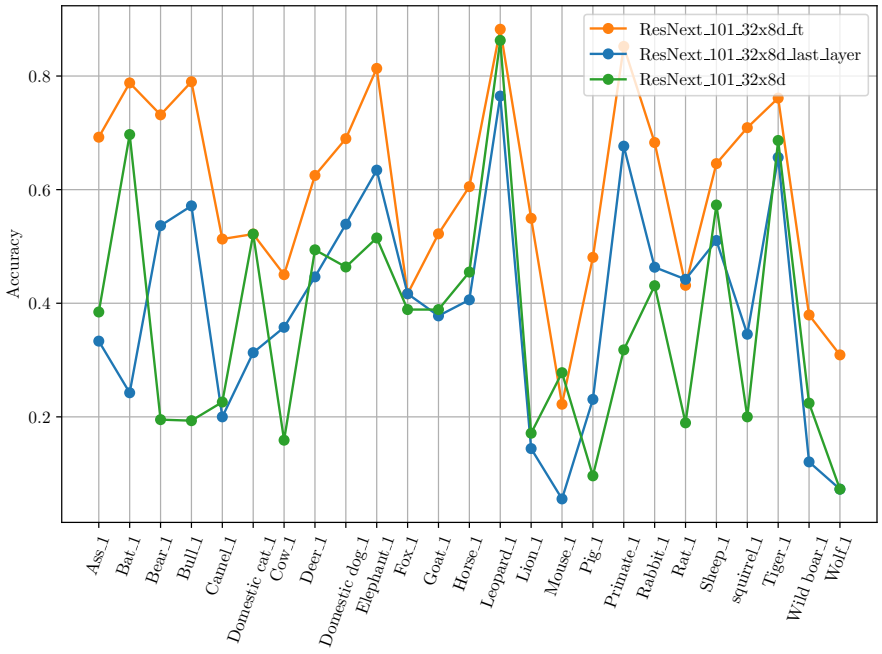


Figure 4.13: Top-1 per class accuracy on the paintings test set, comparing transfer learning applied to ResNeXt (pre-trained classifier, last layer re-training, fine-tuning).

features in the photo-realistic domain.

4.3 Analyzing the final classifier

4.3.1 Examining the confusion matrix

There are several possibilities to gain more insight about the behavior of classifiers when presented with unseen crops, among which a review of the confusion matrix that will allow to spot the model's strengths but also its weak points. For example, one may wonder how well the model performs for each class, but even more, how well it discriminates related animal classes, e.g. how well does the model distinguish asses from horses? How well does the model differentiate between rats and mice? Furthermore, (per-class) top-1 accuracy metrics are a first step towards measuring model performance and interpretation, but they have a tendency to smoothen results which prevents from conducting fine-grained analyses. Indeed, accuracy metrics underline the performance of the framework but it is impossible to determine where errors and confusions are mostly distributed.

All further analyses can be conducted for each classifier derived in the previous section, i.e. for each training setting. However, due to the high amount of resulting models and due to the (assumed-to-be) likely duplicate interpretations that would occur between all models, a relatively thorough analysis will only be carried out for the classifier which obtained the highest performance across all settings, i.e. the fully fine-tuned version of ResNeXt, pre-trained beforehand on the 1-K ImageNet and the animals-only ImageNet datasets.

The confusion matrix of this particular ResNeXt model is displayed in Figure 4.15, where columns represent ground truths while rows represent model predictions. Deeper shades

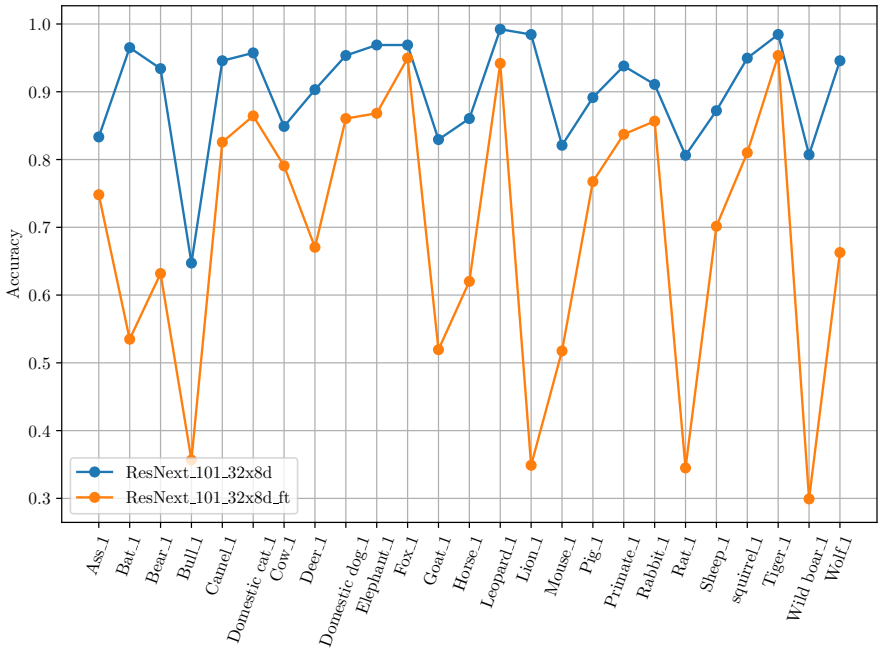


Figure 4.14: Top-1 per class accuracy on the ImageNet animals-only test set, for the pre-trained and fine-tuned versions of ResNeXt.

indicate more frequent predictions, for each ground truth animal.

From this matrix, some intuitive confusion hypotheses evoked earlier can be confirmed or contradicted. For instance, it was expected that rats and mice would very likely be confused with each other, with no preliminary hypothesis on the direction of confusion. It can be observed that mice are predominantly classified as rats rather than mice (12 predictions vs 8 only), while rats are mainly classified as rats even though the number of corresponding mice predictions is not negligible, with 17 such misclassifications. Another example of similar confusions includes cows and bulls, for which it can be noticed that the model makes non-negligible mistakes too. In addition, it can be noticed that wolves are confused with dogs and foxes, which can be easily understood as these animals share some visual features. On top of that, wild boars seem often misclassified as pigs, another natural confusion since both belong to the same family. Finally, the same confusing behavior can be observed for asses and horses. However, it appears clear that the confusion is this time only one-sided with mostly horses being confused with asses, and not the other way around.

We can look more closely at rows in order to see whether the model could make biased predictions towards certain animals at the expense of others. When presented with asses, the model correctly classifies them as asses but it is also worth noting that several other animals are confused with asses: horses, as underlined previously, but also some camels, cows, bulls and goats. We can also notice that goats and sheep are interchangeably confused. Thus, we could expect that the model leverages irrelevant features related to the environment in which these animals are depicted, e.g. the classifier might be biased towards goat (or sheep) predictions whenever a non-negligible amount of grass is visible in the crop, and the same idea applies for asses. Furthermore, camels are sometimes classified as horses and more surprisingly, dogs can also be confused with this animal or

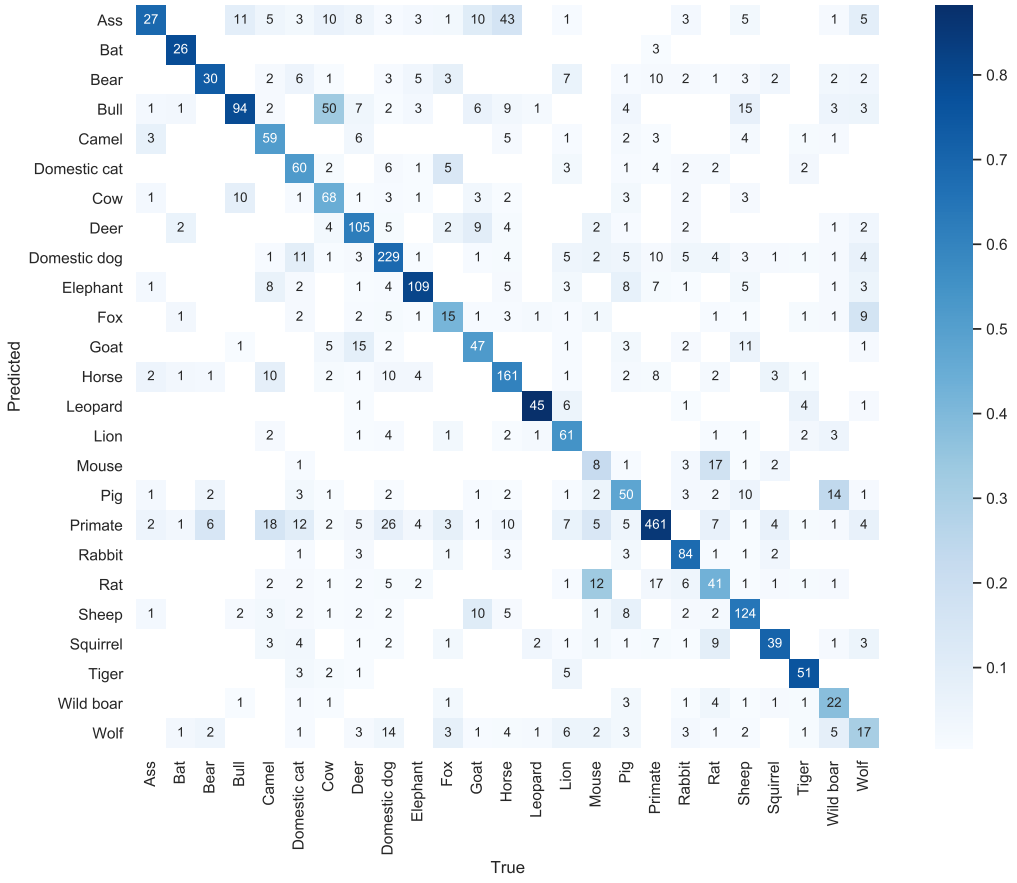


Figure 4.15: Confusion matrix for ResNeXt-101 32 × 8d fully fine-tuned on paintings. Rows represent predictions while columns correspond to ground truths.

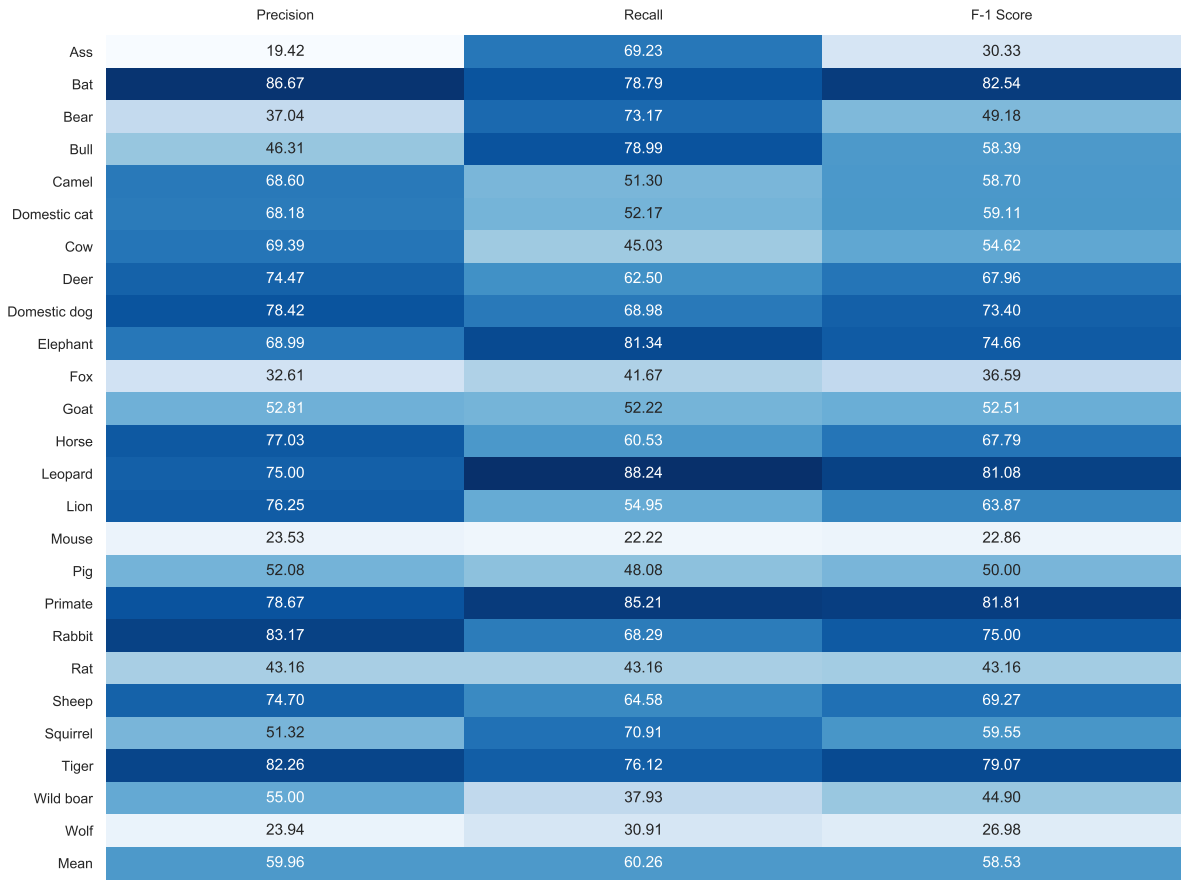
with primates.

4.3.2 Computing precision and recall

With the information presented in the confusion matrix, one could wonder about the precision-recall trade-off achieved by the model, for each animal. It is worth noting that, as more incorrect predictions are made for a given animal presented as input crop, then its corresponding recall decreases but the precision of the animals with which it was confused also decreases.

Intuitively, it is difficult to provide a general trend about precision and recall values for each class since the behavior of the model varies depending on the considered class. Still, one could expect that animals which are often predicted by the model, as is the case for asses, will have a higher recall, while animals mostly confused with asses will have a smaller recall due to this increase of false negatives.

Since rows of the confusion matrix correspond to model predictions and columns to ground truths, precision for a given class is evaluated on each row whilst recall is evaluated on each column. The corresponding precision and recall values are reported in Figure 4.16, along with their F-1 score, for each class. We can observe that the model has roughly equal mean precision and mean recall, indicating that, on average, we should be relatively confident about the predictions made by the model, with a mean precision of 59.96%,



	Precision	Recall	F-1 Score
Ass	19.42	69.23	30.33
Bat	86.67	78.79	82.54
Bear	37.04	73.17	49.18
Bull	46.31	78.99	58.39
Camel	68.60	51.30	58.70
Domestic cat	68.18	52.17	59.11
Cow	69.39	45.03	54.62
Deer	74.47	62.50	67.96
Domestic dog	78.42	68.98	73.40
Elephant	68.99	81.34	74.66
Fox	32.61	41.67	36.59
Goat	52.81	52.22	52.51
Horse	77.03	60.53	67.79
Leopard	75.00	88.24	81.08
Lion	76.25	54.95	63.87
Mouse	23.53	22.22	22.86
Pig	52.08	48.08	50.00
Primate	78.67	85.21	81.81
Rabbit	83.17	68.29	75.00
Rat	43.16	43.16	43.16
Sheep	74.70	64.58	69.27
Squirrel	51.32	70.91	59.55
Tiger	82.26	76.12	79.07
Wild boar	55.00	37.93	44.90
Wolf	23.94	30.91	26.98
Mean	59.96	60.26	58.53

Figure 4.16: Precision and recall values for each class, considering predictions carried out by ResNeXt-101 32×8d.

with satisfying detection capabilities, illustrated with a mean recall of 60.26%.

We can observe that the model has a very poor F-1 score for mice and wolves. For these animals, the model is characterized by low precision and recall values.

As regards asses, for which it was hypothesized that recall would be higher than precision (since many crops were classified as this animal), we can note that this assumption is indeed verified and that horses, for instance, have a lower recall but higher precision.

Finally, tigers, leopards, primates and bats achieve the highest F-1 scores, which means that most instances of each animal will be detected by the model but also correctly classified, for the majority of the time. Thus, we can be quite confident about the model predictions made for these classes, the highest precision being achieved for bats. Other animals also achieve relatively high precision values, such as deers and lions, which can be explained by the fact that they are animals easier to recognize. However, their recall values are relatively smaller, which indicates that the model has difficulties to identify such species.

4.3.3 Visualizing model predictions

In order to have a deeper insight about what could bring the classifier to such misclassifications, e.g. why the classifier predicts some sheep as cows, a Grad-CAM technique will

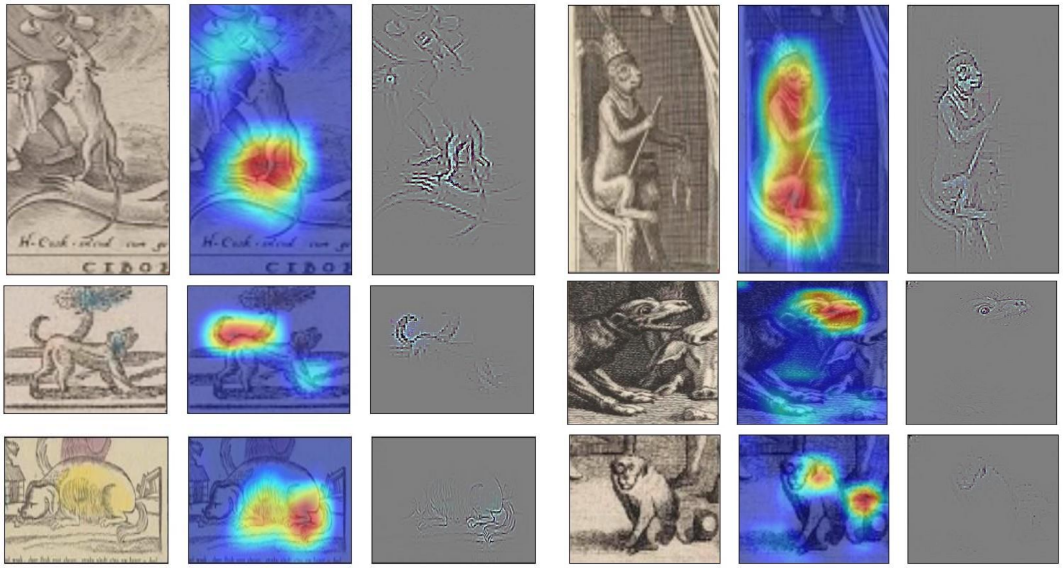


Figure 4.17: Dog samples being classified as primates (left) and that should be labeled as primates (right).

be considered, as was presented in Chapter 2.

Several confusions will be investigated, namely:

- Dogs confused with primates
- Goats confused with asses
- Goats confused with sheep
- Cows confused with asses

Please note that not all confusions present the same pattern for a given class. For instance, error interpretation performed for dogs confused with primates does not necessarily hold across all such confusions; some of them do not highlight any specific pattern or do not even seem to focus on the wrong parts of the input crop. It was also observed that, in general, some crops were hard to distinguish even with a human eye.

Dogs confused with primates

A sample of dog crops misclassified as primates is displayed on Figure 4.17, where both a Grad-CAM and a Guided Grad-CAM are represented. From these samples, it seems that the model primarily focuses on the animal tail to guide its final prediction, leaving a smaller importance to its posture, e.g. if it is lying down or walking. Furthermore, it seems that the action of standing up on back legs makes the model more likely to predict primates, for which it is a more natural attitude.

Nevertheless, it should be noted that some samples classified as primates indeed correspond to primates, and that it rather consists of a small annotation mistake, as can be observed on the right of the figure.

Goats confused with asses

Identically to the previous case, samples of goats misclassified as asses are illustrated on Figure 4.18. There, it seems that the classifier focuses mostly on the head and the neck of

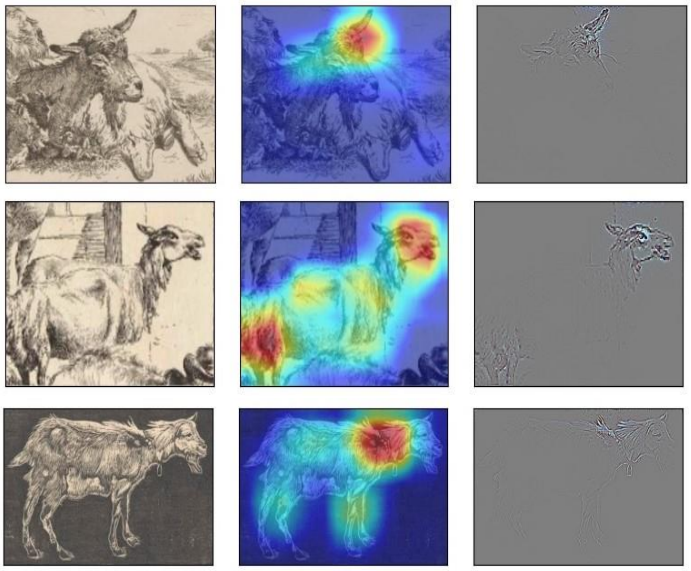


Figure 4.18: Goat samples being classified as asses.

the animal but leaves very little importance to its hair. Still, the distinction between goats and asses is not completely clear from these samples and such misclassifications could be expected. Furthermore, these observations infirm the previous intuitive hypothesis that the model confuses both species because of environmental factors such as grass or even a barn-like environment.

Goats confused with sheep

Samples of goats misclassified as sheep are depicted on Figure 4.19. From there, nothing can really be concluded about specific reasons as to why goats can be classified as sheep. Indeed, from all samples, it appears that the model leverages informative features, such as the goat's horns or even its coat, which should look different from the coat (wool) of a sheep. Nevertheless, it could be possible that some pictures depicting animals looking similarly to the first or last rows of Figure 4.19 have been annotated as sheep in the training set, therefore implying this confusion.

As was the case for primate samples labeled as dogs, there were also samples of sheep annotated as cows, or for which the difference between both animals was very difficult to establish, as can be visualized on Figure 4.20. From this figure, we can see that the first and last samples have been labeled as goats, although the confusion with sheep is reasonable. For the second sample, the model relied on features extracted from the dog to the right, the goat to the middle, but also from the sheep in the top-left corner, which could explain its prediction although the ideal case would have been a goat prediction as it represents the main animal in the crop.

Cows confused with asses

Some samples out of the few samples of cows being misclassified as asses can be visualized on Figure 4.21. It appears that the model extracts information mainly from the head of the cow along with its legs, although it also appears clear that each sample represents cows and not asses. For the first sample, it also extracts features from the person standing

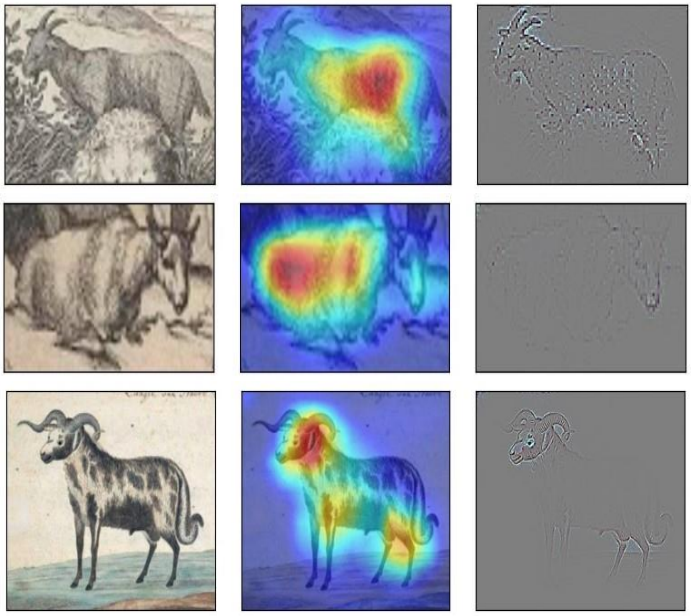


Figure 4.19: Goat samples being classified as sheep.

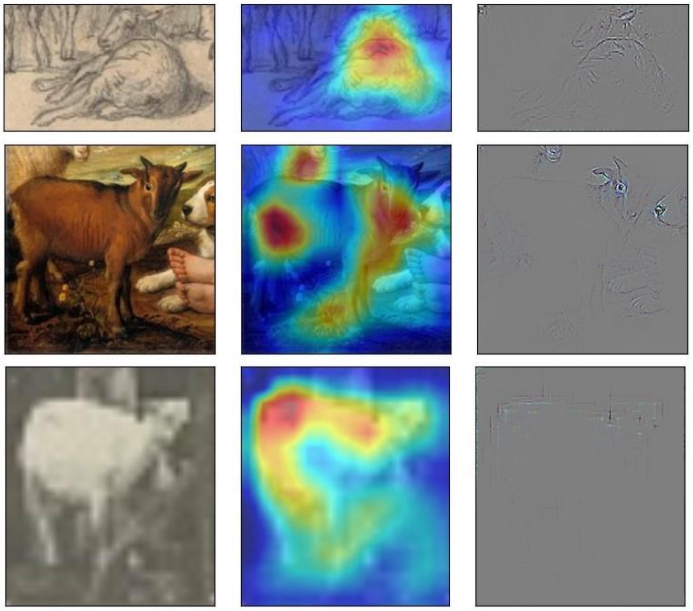


Figure 4.20: Goat ground truths that could be labeled as sheep or where confusion is admissible.

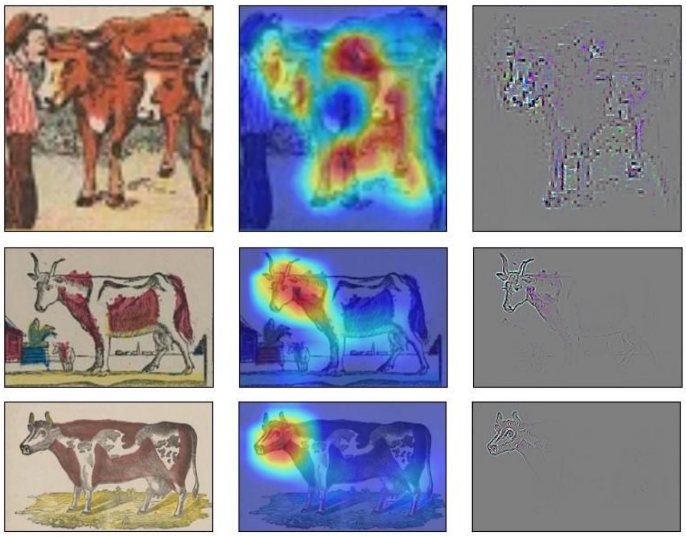


Figure 4.21: Wild boar samples being classified as cows.

next to the cows, so this might push the model towards predicting asses. For the two other samples, we could conclude that this specific shape of head tends to be associated to asses and also underline the fact that the model does not leverage any information about the difference in marks in the cow's fur, which is here brown and white but which could also be black and white.

4.4 Conclusions

In this chapter, we have tackled the classification of animal crops extracted from artworks of various styles using three classifier architectures of different kinds. In particular, we have studied the performance of VGG-16, ResNet-101 and ResNeXt-101 32×8d.

To serve as an initial baseline, we decided to assess their performance when applied *as is* on painted animal crops, all three of them being pre-trained on the 1-K ImageNet dataset. In order to have a measure of accuracy for each animal, we have constructed an animals-only version of ImageNet that contains roughly 1 000 photo-realistic pictures per class. Each classifier was thus fine-tuned completely on this new dataset, reaching top-1 accuracies between 81% and 90% for natural images. When applied to paintings, a performance drop could be observed, the highest accuracy dropping from 90.10% on photo-realistic images to 37.24% on paintings. This performance drop is inherent to the domain transfer being involved.

Afterwards, several approaches were explored to determine whether they could improve upon these baseline results. Classifiers were re-trained from scratch on different training sets: one containing only painting crops, one containing only the ImageNet animal pictures from the dataset previously constructed, and one containing an equal mix of paintings and natural images. In addition, we also investigated two specific transfer learning techniques. The first one consists in using pre-trained classifiers off-the-shelf, which means that their last linear layer is replaced by a new layer while previous layers remain frozen throughout training. The other approach consists in fully fine-tuning the classifiers, except for their final fully-connected layer which is replaced by a new one.

As could be expected, models trained completely from scratch show a much worse final top-1 accuracy. In fact, they even achieve results lower than when pre-trained classifiers are used as is. Training from scratch on the animals-only dataset reaches a significantly smaller accuracy than when paintings are included in the training set, which could also be expected since no artistic features would be incorporated in the models. Models do not seem to gain anything in downstream performance when a mix of paintings and photo-realistic pictures is used rather than only paintings. The highest accuracy reached overall in these settings is only equal to 20.38%, which illustrates the importance of starting from pre-trained weights, especially for tasks where a limited amount of data is available.

For models starting from pre-trained weights, we could observe a consistent gain in the off-the-shelf (last layer) setting as well as for the full fine-tuning approach. In addition, there is a considerable gap in performance between off-the-shelf and fine-tuned networks, particularly for ResNe(X)t models. The highest performance on the test set is achieved by ResNeXt-101 32×8d with a top-1 accuracy of 65.01%. This model improved for each class, except for mice, showing a negative transfer with a drop of 5.56% compared to its pre-trained equivalent.

As a final step, we decided to examine more deeply the behavior of this fine-tuned architecture. For that purpose, we have derived its confusion matrix on the test set along with precision-recall measures for each class. This allowed us to determine the animals that the model easily detects but also those for which we can trust more easily the model's predictions. It was observed that the model achieved the best precision-recall trade-off for bats, primates, leopards and tigers, meaning that we can be relatively confident when the model predicts such animals. We have also provided various visual representations that helped to understand how the model made mistakes for several misclassifications highlighted by the confusion matrix, by putting in contrast regions of the input image to which the model paid the most attention to make its final (erroneous) prediction.

Chapter 5

Detection

This chapter will describe the different steps and approaches that were used to deal with the second part of the work. Section 5.2.2 will define the dataset splitting strategy that was adopted. Section 5.2 will cover in more details how pre-trained object detectors were applied to paintings and determine whether standard transfer learning approaches enhanced their performance. Section 5.3 will detail the different experiments that were conducted on pre-trained object detectors to deal with different module freezing settings. Section 5.4 will review the object detector selected as final model to visually analyze its predictions. Finally, Section 5.5 will summarize and conclude the experiments carried out on the object detection models.

5.1 Problem definition

As was defined in Section 2, the second part relates to the problem of object detection, which in the scope of this work, consists in training an object detector to correctly locate bounding boxes around animals and then to correctly classify these boxes with the appropriate animal. To this end, a similar approach as for the problem of classification will be considered. Indeed, different state-of-the-art architectures will be put in contrast, for several training settings which will be detailed in the next section. Note that this task now considers full-sized paintings and no longer crops extracted from paintings.

For each setting, paintings will be split into training, validation and testing sets according to the following proportions:

- Training set $\rightarrow$ 50%
- Validation set $\rightarrow$ 25%
- Testing set $\rightarrow$ 25%

This split choice was carried out for the previous classification task and it is thus natural to consider the same proportions. As a reminder, it allows for a honest model tuning and an unbiased later performance assessment. Splits are once again performed using a stratified approach, i.e. all splits will keep similar proportions of each animal compared to the proportions present in the entire dataset and in fact they correspond to the same splits that were derived for the classification task.

5.2 Applying pre-trained object detectors on paintings

This section will detail the different experiments and assessments conducted with pre-trained object detectors applied for the detection of animals in paintings. This represents the end task of the present work and will produce benchmark results for this new problem.

5.2.1 Evaluation of pre-trained object detectors

As was done for the classification task, pre-trained object detection architectures described above can be tested directly on paintings to assess their performance with no fine-tuning at all. However, as was the case for the classification task, the classes that detectors have been trained on do not entirely match the classes found in the paintings dataset. Indeed, both frameworks have been pre-trained on COCO, which is composed of 80 classes among which only a restricted subset intersects with the animals depicted in the artworks. This subset consists of bears, cats, cows, dogs, elephants, horses and sheep, i.e. only 7 classes out of 80.

Contrary to the pre-processing steps carried out before evaluating the performance of pre-trained classifiers, as detailed in Section 4.2.2, no custom ImageNet nor COCO datasets will be constructed. This means that all following steps will build upon features learnt for the 80 corresponding classes, even if no labeled instances of the remaining 18 animals have been processed by these detectors. Note that this could also have been considered for the classification task; however constructing a custom dataset for natural image classification is less time-consuming than constructing a custom dataset for object detection in natural images, which would entail manually annotating thousands of images. In addition, this allowed for a thorough preliminary evaluation while preliminary evaluation for the detection task will only cover 7 classes.

In the scope of this work, given the novelty of the tackled problem as well as the non-negligible presence of small objects (i.e. small animals), the main measure of comparison considered for the evaluation of the different architectures will be the (mean) average precision computed at an IoU threshold equal to 0.5, as is also used in [11]. This represents a decent compromise between feasibility and severity, as we could expect detection performance to be on the lower end.

	Bear	Domestic cat	Cow	Domestic dog	Elephant	Horse	Sheep
Faster R-CNN	6.71	11.07	27.60	13.88	20.48	23.85	25.28
YOLOv5	9.58	21.30	35.30	25.20	36.40	35.50	41.70

Table 5.1: Performance of object detectors pre-trained on COCO, on the paintings test set.

These metrics have been computed for the intersecting classes and are available in Table 5.1. The pre-trained Faster R-CNN detector relies on a ResNet-50 backbone while YOLO relies on a CSP-Bottleneck backbone.

From these results, it appears clear that YOLO outperforms Faster R-CNN for each of the intersecting classes, thereby underlining the fact that the performance gap is smaller for

this network when shifting from the photo-realistic to the artistic domain. Unfortunately, average precision scores achieved by these detectors on natural images (COCO test-dev) cannot be obtained for each separate class. Indeed, COCO testbeds considers[1] only the mean average precision metric (mAP), which they equivalently (and confusingly) call average precision (AP). Thus, we will only be able to compare results derived on paintings. As a measure of comparison, a similar YOLO implementation achieves[2] an mAP @.5 equal to 73.40% on the COCO test set while Faster R-CNN (implemented with Local Importance-based pooling) reaches a value of 65.70%.

5.2.2 Fine-tuning object detectors on paintings

To follow the same line of work as for the classification task, both architectures will be fine-tuned on paintings using the training and validation sets defined in Section 5.1.

As was the case previously, data augmentation will be performed for both architectures, with the slight variation that YOLOv5 integrates autonomously data augmentation techniques, among which the aforedefined mosaic augmentation. Thus, augmentation techniques will differ between both architectures. As concerns Faster R-CNN, usual data augmentation techniques, such as those used in Chapter 4, are considered. In both networks, input paintings will be resized to meet sizes of 640px for YOLO and in $[800, 1333]$px for Faster R-CNN. Note that this was already the case for the evaluation of detectors pre-trained on COCO reported in Section 5.2.1. Learning rate decay is considered for both models along with a batch size of 1 and 2 paintings respectively, due to memory constraints, and an SGD optimizer.

Contrary to the classification task, early stopping is no longer based on validation loss, which for both networks consists of the combination of individual loss components characterizing the classification performance, the objectness performance, etc. Instead, a fitness measure involving mean average precision scores for different IoU settings is considered. This fitness criterion can be defined as:

$$\text{fitness} = 0.1 \times \text{mAP}\,@.5 + 0.9 \times \text{mAP}\,@[.5, .95]$$

Considering such a fitness criterion instead of validation loss will help to continue training if models continue learning, while losses could quickly saturate. Indeed, the end goal is to maximize mean average precision, which makes this fitness criterion a rather logical choice. Note that this criterion was originally defined in YOLOv5 and it seemed natural to extend its use to Faster R-CNN for a fairer comparison. Training will be stopped whenever the fitness measure no longer increases after 10 consecutive epochs.

Faster R-CNN: backbone influence

As was mentioned previously, Faster R-CNN is composed of a backbone module which serves as feature extracting component. For the previous evaluation of detectors, a ResNet-50 backbone pre-trained on COCO was used, but it is also possible to replace this module by another convolutional neural network. A natural choice is the ResNeXt-101 32×8d that was the best-performing fine-tuned classifier in the previous chapter. One can simply drop the final linear layer to keep only convolutional and pooling layers. Note

[1] https://cocodataset.org/#detection-eval
[2] https://paperswithcode.com/sota/object-detection-on-coco

that the idea is to compare the impact of backbones initialized with different settings, i.e. one that is pre-trained on photo-realistic images while the other one is pre-trained on natural images and fine-tuned on paintings. Hence, the architecture does not really matter as it is not the central study point, and some bias might thus be embedded inside the backbone architecture, as it is possible that one architecture fits better than the other for the artistic domain.

It is also worth noting that, for the Faster R-CNN model relying on the COCO-pre-trained backbone, the RPN is also initialized with COCO-pre-trained weights; only the pre-trained head, which will perform bounding box regression and classification, is replaced by a new Fast R-CNN predictor, designed for 25 output classes instead of 80. When the ResNeXt backbone is used inside Faster R-CNN, both the RPN and the Fast R-CNN predictor will be initialized with new weights.

Both settings will be fully fine-tuned on paintings and one could intuitively expect that performance achieved by the object detector whose backbone is ResNeXt-101 32×8d (or another CNN fine-tuned on paintings) will eventually be better than the performance achieved with a backbone where no paintings-like features are initially present. In addition, backbones fine-tuned on paintings also incorporate features for all animals depicted in various artistic styles and not only for animals of the COCO intersection depicted in a photo-realistic manner.

Faster R-CNN's loss is composed of four elements:

- `loss_box_reg`, which yields a measure of the accuracy of the predicted output bounding boxes
- `loss_classifier`, which yields a measure of the classification quality for the predicted output bounding boxes
- `loss_objectness`, which yields a measure of the quality of the confidence that the RPN has about the presence of objects in the different generated regions
- `loss_rpn_box_reg`, which yields a measure of the accuracy of region proposal boxes generated by the RPN

Results obtained on the training and validation sets can be visualized on Figure 5.1 and 5.2, which represent training losses and validation fitness respectively. From these figures, it appears that both settings reach similar loss values with a similar number of training epochs. However, the model with the ResNeXt backbone converges to a higher loss compared to its COCO counterpart. Looking at the individual loss components, we can see that this is not really due to the classification nor bounding box regression components in the prediction head, as both approach close values in any setting. In addition, both loss components start from lower values than those of their COCO equivalent, taking advantage of the feature representation built inside the fine-tuned ResNeXt backbone. Thus, the largest contribution to this loss increase is rather caused by the region proposal network, which has higher initial losses and eventually converges to higher final values. The impact of starting from pre-trained weights instead of new weights is clearly visible.

This higher total loss value also translates into the validation fitness measures, where it appears clear that the Faster R-CNN detector using the ResNeXt backbone reaches a substantially lower fitness score. In addition, it also starts from a lower fitness measure, very close to zero. Since both detectors rely on a pre-trained backbone and on a new

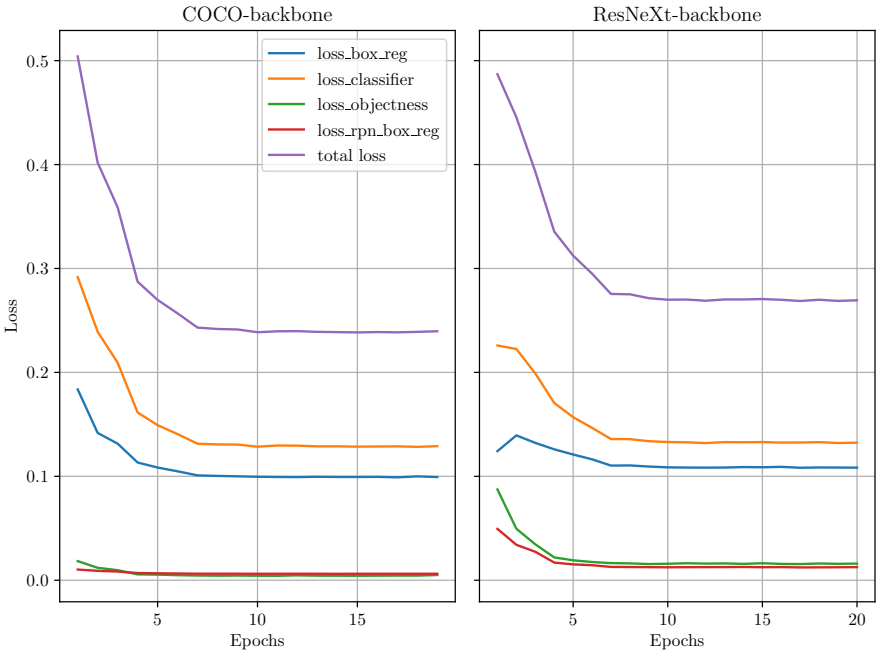


Figure 5.1: Training losses for the fine-tuning of Faster R-CNN, with a backbone pre-trained on COCO (left) and with a backbone already fine-tuned on paintings (right).

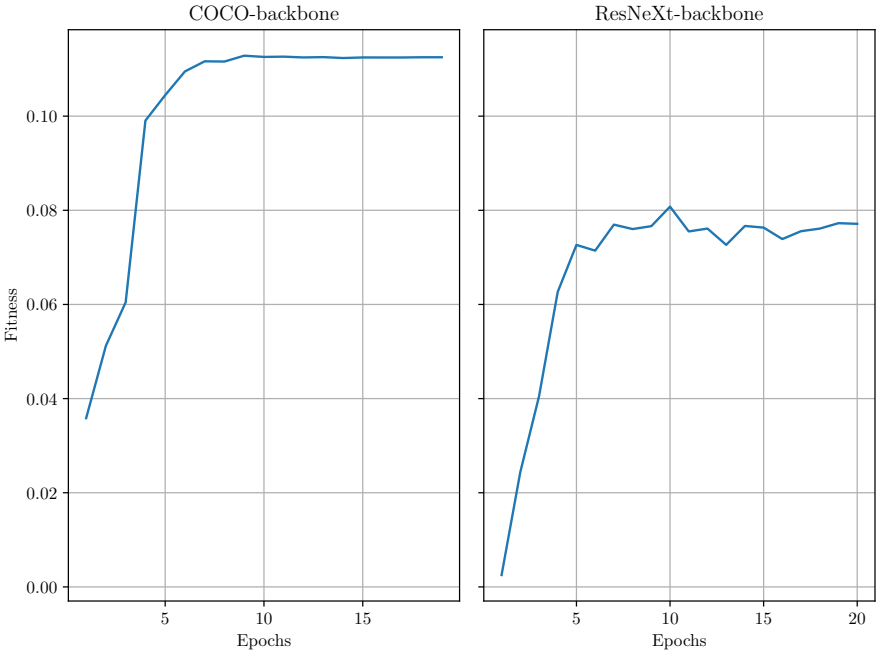


Figure 5.2: Validation fitness for the fine-tuning of Faster R-CNN, with a backbone pre-trained on COCO (left) and with a backbone already fine-tuned on paintings (right).

Fast R-CNN predictor head, we can argue that this difference in performance is mostly attributable to the RPN, which produces poor region proposals in the first epochs for the ResNeXt setting. Even though proposals improve, which can be deduced from the decrease of the RPN loss components and from the validation fitness increase, the final performance gap is still clearly visible. Nevertheless, final performance will be assessed with average precision measures computed for an IoU threshold of 0.5, which only contributes by a factor of 0.1 to the fitness score.

Both fine-tuned models can now be evaluated on the paintings test set, as defined in Section 5.1. For each detector, average precision values are computed, for each animal, such that a deeper understanding of the difficult classes can be gained. As a general measure, the mean average precision will also be computed, denoted as mAP @.5. Results are reported in Table 5.2.

	mAP @.5	Ass	Bat	Bear	Bull	Camel	Cat	Cow	Deer	Dog	Elephant	Fox	Goat
COCO-backbone	12.30	**14.95**	0.00	**13.65**	**30.46**	3.40	3.87	**21.90**	18.60	**33.66**	**45.22**	6.08	7.57
ResNeXt-backbone	**13.00**	8.09	0.00	8.16	25.70	**9.83**	**15.36**	14.68	**20.23**	31.54	32.06	**15.28**	**12.04**
	Horse	Leopard	Lion	Mouse	Pig	Primate	Rabbit	Rat	Sheep	Squirrel	Tiger	Boar	Wolf
COCO-backbone	21.82	4.31	0.57	0.00	**1.87**	18.16	12.20	1.99	10.54	**2.05**	**30.75**	**3.45**	**0.45**
ResNeXt-backbone	**30.84**	**5.10**	**5.78**	0.00	0.92	**18.30**	**27.58**	**4.05**	**19.10**	0.00	20.29	0.00	0.00

Table 5.2: Performance of a fine-tuned Faster R-CNN with two different backbones, on the paintings test set.

From these results, it can be surprisingly observed that the model relying on the fine-tuned ResNeXt backbone achieves a higher mAP @.5 than its COCO counterpart, with an improvement of 0.70%. This observation corroborates the initial hypothesis stating that a detector that includes a backbone already fine-tuned on paintings should achieve a higher performance than one that includes a backbone pre-trained on natural images. Still, the result is surprising since there was a visible performance gap between both models that was clearly underlined in Figure 5.2. Thus, it could mean that the model relying on the fine-tuned ResNeXt backbone outperforms its COCO equivalent for an IoU threshold of 0.5, but that its performance should get worse than that of the latter when the minimum IoU increases. This can be checked by running both models on the validation set and a comparison of validation-test results is provided in Table 5.3. Please note that these results will very likely be positively biased for both backbones but the most interesting part is the ordering of models rather than their validation performance. From validation results, we can observe that the situation is the opposite and rather corresponds to that observed in Figure 5.2, where the performance of the Faster R-CNN model fine-tuned from a COCO pre-trained backbone and RPN is higher than that of the model fine-tuned from a paintings fine-tuned backbone. Thus, it means that the ResNeXt-backbone Faster R-CNN outperformed its COCO counterpart on the test set by chance, benefiting from a more advantageous split. Still, it is worth noting that the difference between both validation mAP @.5 scores is only of 2% while it can be observed from Figure 5.2 that the difference between both fitness scores is roughly of 4%. Therefore, we can expect that, given the definition of the fitness criterion, the performance of the Faster R-CNN model with ResNeXt backbone will get even worse for higher IoU thresholds compared to that of the COCO backbone model.

The ResNeXt-backbone model is not a clear winner across all animal classes. Indeed, even if it performs better on average, there are several animals for which it shows more

	Test set	Validation set
COCO-backbone	12.30	18.45
ResNeXt-backbone	13.00	16.5

Table 5.3: Performance of Faster R-CNN on the paintings validation and test sets, for both backbones.

difficulties than the COCO-backbone detector, e.g. for asses, elephants, cows, etc. There are animals for which detection performance was expected to be decent, such as elephants and tigers, which should be easier to recognize than other (smaller) animals like bats, mice or rats. However, there are animals that should have been easy to localize and classify but for which both models achieve very poor performance, e.g. leopards that have a distinguishable coat but only have a 5.10% AP. Some animals even have 0.00% average precision, such as bats, mice, squirrels, wild boars and wolves although classification performance was decent for most of them.

One could wonder whether the bounding box size has a noticeable impact on the downstream performance on object detectors. For instance, do rats, mice, bats, sheep, cows and squirrels necessarily perform worse since they have the highest proportions of small bounding boxes across all animals, as was highlighted in Table 3.2? We could argue that this is true for the first three animals and squirrels but that it does not really hold as regards cows and sheep. In addition, there are some other animals that do not seem to have large proportions of small bounding boxes, such as pigs and wolves, but for which AP performance is worse than for small animals, say rats. To assess this potential influence more rigorously, we can compute a correlation coefficient between the mean area of bounding boxes for each animal and its corresponding AP score. In this case, a Spearman rank correlation coefficient will be considered and the p-value associated with the null hypothesis stating that the two series of values are uncorrelated will also be calculated.

	Correlation coefficient	p-value
COCO-backbone	0.465	0.019
ResNeXt-backbone	0.472	0.017

Table 5.4: Spearman rank correlation results for the correlation between mean bounding box area and AP, for both Faster-RCNN models.

From Table 5.4, we can observe that, whatever the backbone setting, there is a positive correlation between mean bounding box area and its corresponding AP, meaning that animals with larger bounding boxes are more likely to be correctly detected than animals with smaller bounding boxes. In both cases, this observation is supported by a p-value smaller than the standard 0.05 significance threshold, meaning that the null hypothesis should be rejected, i.e. bounding box area and average precision are not uncorrelated.

We can compare average precision scores obtained for the fine-tuned detector relying on a COCO-pre-trained backbone with those of the Faster R-CNN model pre-trained only on COCO, which are reported in Table 5.1, for animals belonging to the intersection between COCO classes and the animal classes from the paintings. This comparison is provided in Table 5.5. We can observe that fine-tuning yields strong improvements for bears, dogs and elephants where the resulting AP is twice as big as the AP obtained using

	Bear	Domestic cat	Cow	Domestic dog	Elephant	Horse	Sheep
COCO only	6.71	11.07	27.60	13.88	20.48	23.85	25.28
Fine-tuned	13.65	3.87	21.90	33.66	45.22	21.82	10.54

Table 5.5: Performance (paintings test set) of Faster R-CNN with the same backbone, pre-trained on COCO, without and with fine-tuning on paintings.

the detector as is. However, performance drops of various magnitudes can be observed for cats, cows, horses and sheep. Note that performance degradation could also be observed in Table 4.4 for a ResNet-101 classifier, where three classes (among which cats) suffered from fine-tuning.

Finally, these results can be compared with [11], where they notably tested the detection performance of a fine-tuned YOLO detector on the 10 most occurring musical instruments. There, only two instruments reach average precision scores above 25%, for an IoU of 0.5, and two others achieve an average precision around 5% while the six last ones have scores close to zero. It can be underlined from Table 5.2 that there are some classes that also reach such low scores, but more animals achieve a non-negligible average precision score. It is worth noting that, as was discussed in [11], the drop of performance with respect to standard results obtained for photo-realistic images could be expected due to the smaller size of the training data along with the higher variation of styles that is inherent to artworks. We can also try to quantify the influence of training data size on average precision scores, using as before a Spearman correlation test that will assess the correlation between the amount of training occurrences and the corresponding AP measures, for each animal.

	Correlation coefficient	p-value
COCO-backbone	0.642	1.2×10^{-5}
ResNeXt-backbone	0.756	5.4×10^{-4}

Table 5.6: Spearman rank correlation results for the correlation between training occurrences and AP, for both Faster-RCNN models.

For both Faster R-CNN models, it can be observed from Table 5.6 that the number of training occurrences has, as expected, a strong impact on the downstream detection performance.

YOLO: backbone influence

Identically to the previous case, fine-tuning will be carried out on YOLO for both types of backbone: the first one being pre-trained on COCO (CSP-Bottleneck backbone) and the second one being the fine-tuned ResNeXt-101 32×8d, as was done for the previous experiment on Faster R-CNN. Note that YOLO's head will be initialized with COCO pre-trained weights if the backbone is initialized with such weights too. Otherwise, it will be initialized with new weights.

YOLO's loss is composed of three elements:

- `loss_box`, which represents a measure of the accuracy of predicted bounding boxes

- `loss_obj`, which represents a measure of the quality of the confidence that the model has about the presence of objects in the different cells
- `loss_cls`, which represents a measure of the classification quality for the predicted bounding boxes

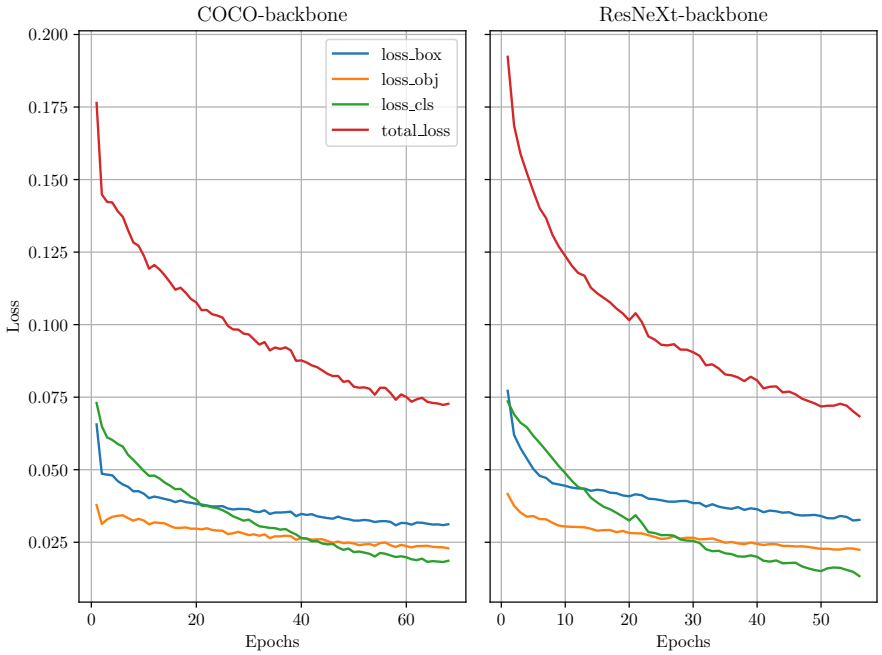


Figure 5.3: Training losses for the fine-tuning of YOLO, with a backbone pre-trained on COCO (left) and with a backbone already fine-tuned on paintings (right).

Training losses are displayed for both backbones in Figure 5.3 while validation fitness is depicted in Figure 5.4. As can be observed from these figures, both settings show a very similar behavior, even though starting from a backbone fine-tuned on paintings converges in less epochs (57 vs 69). Losses reach close values in both settings and their decrease looks much alike: at first, the classification module contributes the most to the total loss but it reduces more continuously than the two other components, such that in the end, the box accuracy component dominates. It is worth noting that starting from the ResNeXt backbone seems to increase the convergence of the classifier head. Indeed, values for `loss_cls` are smaller using a ResNeXt backbone than when using a COCO backbone, at identical epochs, although both start from close initial values (obtained after one epoch of training).

As far as validation fitness is concerned, the same observations as for training losses hold since both models present the same learning trend. However, it should be noted that the model that relies on a ResNeXt backbone starts with a lower validation fitness. Still, it easily catches up with the other model and even achieves higher values before it, probably taking advantage of the feature representations that its backbone has constructed during its preliminary fine-tuning. Nevertheless, the YOLO detector fine-tuned with a COCO-pre-trained backbone eventually reaches a higher validation fitness score, even though the difference between both peak performances is relatively small. This could contradict our initial and intuitive hypothesis that a model built with a backbone fine-tuned on paintings will eventually perform better. Please note however that, as was the case for

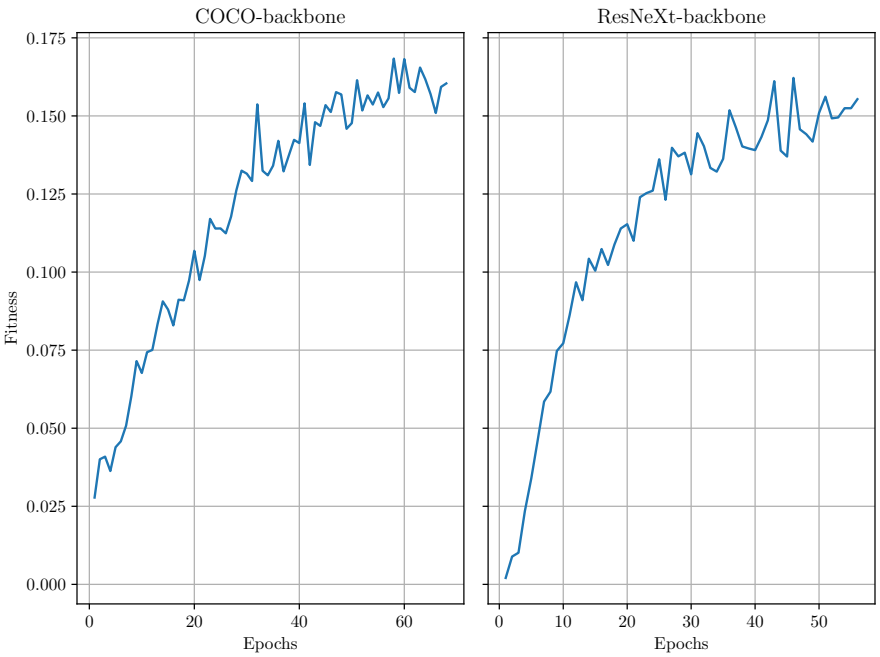


Figure 5.4: Validation fitness for the fine-tuning of YOLO, with a backbone pre-trained on COCO (left) and with a backbone already fine-tuned on paintings (right).

	mAP @.5	Ass	Bat	Bear	Bull	Camel	Cat	Cow	Deer	Dog	Elephant	Fox	Goat
COCO-backbone	18.30	**19.60**	0.88	9.01	**31.60**	5.31	9.88	22.30	**27.40**	34.80	**41.00**	13.30	19.20
ResNeXt-backbone	**19.30**	14.20	**2.91**	**14.60**	27.00	**8.48**	**10.60**	**30.10**	20.80	**37.50**	36.20	**19.90**	**21.40**

	Horse	Leopard	Lion	Mouse	Pig	Primate	Rabbit	Rat	Sheep	Squirrel	Tiger	Boar	Wolf
COCO-backbone	**32.10**	28.50	9.15	**1.13**	**7.41**	**31.00**	21.40	**7.07**	**21.90**	10.20	**35.20**	8.89	**9.65**
ResNeXt-backbone	28.40	**41.90**	**15.20**	0.77	5.74	28.40	**21.90**	5.11	**17.20**	**17.00**	28.80	**22.10**	6.31

Table 5.7: Performance of a fine-tuned YOLO with two different backbones, on the paintings test set.

Faster R-CNN too, these values are obtained on the validation set. Furthermore, they do not correspond to the metric that will be used to assess the performance of each model, the latter being the mAP @.5. In addition, since both settings achieve similar fitness scores, we could expect them to perform similarly on the test data, which could not have been expected for Faster R-CNN from Figure 5.2.

Comparing to Figure 5.2, we can see that YOLO achieves a higher validation fitness score with respect to Faster R-CNN, whatever the backbone. This should imply that YOLO's performance on the test set will also be better, thereby endorsing the general observation concluded from Table 5.1 on the COCO intersection.

Results for both backbones are reported in Table 5.7, where average precision scores are again displayed for each animal. We can notice the same behavior as for the previous Faster R-CNN settings, where the detector built upon the COCO-backbone reaches a higher validation fitness but a lower test mAP @.5, with a notable difference of 1%. Identically to what was done for Faster R-CNN, we can compute the mAP @.5 on the validation data, which is reported in Table 5.8. On the contrary to Faster R-CNN, validation and test results are ordered in the same manner, with similar differences in mAP magnitude. Thus, we can conclude that the YOLO model fine-tuned from a fine-tuned

ResNeXt backbone indeed outperforms its COCO equivalent.

	Test set	Validation set
COCO-backbone	18.30	27.70
ResNeXt-backbone	19.30	28.40

Table 5.8: Performance of YOLO on the paintings validation and test sets, for both backbones.

Similarly to Faster R-CNN, the ResNeXt-backbone model does not outperform its COCO counterpart for all classes. Animals that are easy to recognize have again a decent AP, such as elephants, tigers and even leopards, which was not the case for Faster R-CNN. In addition, leopards even have the highest average precision score with a ResNeXt backbone, while Faster R-CNN only achieved an AP equal to 5.10% for this setting. YOLO has no animals for which the AP is equal to 0.00%. Similarly to Faster R-CNN, YOLO's performance for small animals is poor, for say mice and rats, although non zero.

	Correlation coefficient	p-value
COCO-backbone	0.546	4.7×10^{-3}
ResNeXt-backbone	0.402	0.046

Table 5.9: Spearman rank correlation results for the correlation between mean bounding box area and AP, for both YOLO models.

	Correlation coefficient	p-value
COCO-backbone	0.572	2.8×10^{-3}
ResNeXt-backbone	0.445	0.026

Table 5.10: Spearman rank correlation results for the correlation between training occurrences and AP, for both YOLO models.

To confirm this observation, we can compute the same Spearman tests as were conducted for Faster R-CNN. Results are reported in Table 5.9 for the correlation between bounding box area and AP while results about the correlation between training occurrences and AP are displayed in Table 5.10. We can observe a similar behavior as for Faster R-CNN, although the impact of the amount of training occurrences on downstream performance seems less pronounced; still a positive correlation between both quantities is not negligible. However, it can be noted that the best YOLO model (ResNeXt backbone) seems less affected by bounding box size, and we can observe that its performance on the animals with the smallest bounding boxes (rats, mice, bats, sheep, cows, squirrels) has improved for most of them.

In order to study the impact of fine-tuning on the downstream performance of YOLO, we can compare average precision scores obtained for a YOLO model pre-trained only on COCO with those of the fine-tuned YOLO model with a COCO pre-trained backbone. This comparison is provided in Table 5.11. We can observe that YOLO improves with fine-tuning for only three classes out of seven: bears, dogs and elephants. Since this was already the case for Faster R-CNN, as reported in Table 5.5, we can argue that this

	Bear	Domestic cat	Cow	Domestic dog	Elephant	Horse	Sheep
COCO only	9.58	21.30	35.30	25.20	36.40	35.50	41.70
Fine-tuned	14.60	9.88	22.30	34.80	41.00	32.10	21.90

Table 5.11: Performance (paintings test set) of YOLO with the same backbone, pre-trained on COCO, without and with fine-tuning on paintings.

degradation of performance is most likely due to the variance of the artistic styles used to depict these particular animals.

Comparing Faster R-CNN with YOLO

A summary of the performance of each model, for both settings, is provided in Table 5.12. As was already underlined, YOLO with a ResNeXt backbone achieves the highest mAP @.5 with a value of 19.30%, outperforming its COCO equivalent by 1% and outperforming both Faster R-CNN models by 6%, which agrees with the conclusion derived for both models in Section 5.2.1.

	mAP @.5
Faster R-CNN - COCO backbone	12.30
Faster R-CNN - ResNeXt backbone	13.00
YOLO - COCO backbone	18.30
YOLO - ResNeXt backbone	**19.30**

Table 5.12: Fine-tuning performance comparison, for both models and both settings.

5.3 Transfer learning with frozen modules

Instead of fully fine-tuning object detectors on paintings, other approaches can be considered. For example, one could try an *off-the-shelf* method [55], as was done in Chapter 4 for the setting where only the last layer is re-trained. The idea is to freeze the feature extracting module, i.e. a convolutional neural network, and to train a single additional linear layer. This principle can be transposed for object detectors, in which case the backbone would be frozen and the classifying head would be trained from scratch. This approach has been demonstrated to be beneficial [56] for a domain gap between natural and synthetic pictures. In this work, we want to extend this technique and study the impact of sequentially freezing the feature extracting part of the network, training the rest of the detector in an off-the-shelf manner, unfreezing the feature extractor and finally fine-tuning the whole model. A motivation for this approach is that it could help stabilizing the object detector as, potentially, non-frozen weights would have started to converge and the domain shift influence would thus have a reduced impact. Although authors illustrated in [56] that this approach degraded final performance compared to the setting where only the feature extractor remains frozen, we can still explore this path as it may depend on the type of training data.

In order to make the rest of this section clearer, an overview of all the freezing settings that will be experimented next is displayed in Figure 5.5. All the following sections are organized in a similar fashion.

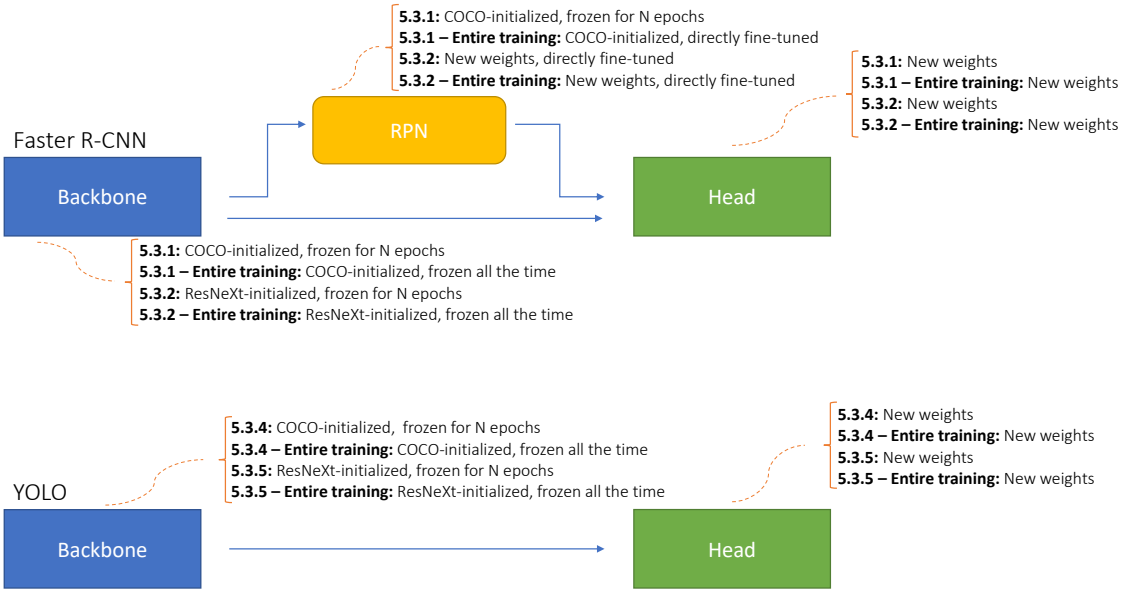


Figure 5.5: Overview of the different freezing settings that will be considered, for each architecture.

Section 5.3.1 will consider a Faster R-CNN detector whose backbone and RPN are initialized with COCO pre-trained weights, and whose predictor head is replaced by a new Fast R-CNN head. Firstly, it will study the impact of freezing both pre-trained modules (backbone and RPN) for N epochs while fine-tuning the head, before unfreezing both modules and fine-tuning them. Then, it will also study the impact of freezing only the backbone, during the entire training, while letting the pre-trained RPN and the new predictor head fine-tune.

Section 5.3.2 will consider a Faster R-CNN model whose backbone is initialized with the paintings fine-tuned ResNeXt model obtained in the previous chapter. In contrary to Section 5.3.1, the RPN has to be initialized with new weights. Once again, the predictor head is initialized with a new Fast R-CNN head. Similarly, it will first study the impact of freezing the pre-trained module (here, only the ResNeXt backbone) for N epochs while fine-tuning the RPN and the head, before unfreezing it to fine-tune the whole architecture. Afterwards, it will explore the influence of freezing the backbone during the entire training while both the new RPN and the new predictor head are being trained.

Section 5.3.4 will consider a YOLO detector whose backbone is initialized with COCO pre-trained weights and whose head is replaced by a new predictor head. It will study the impact of freezing the backbone for the first N epochs while the head is being trained, before unfreezing it and fine-tuning the entire model. Then, it will analyze the impact of freezing the backbone throughout training while the new head is being trained.

Finally, Section 5.3.5 will consider a YOLO model whose backbone is initialized with the paintings fine-tuned ResNeXt model obtained in the previous chapter. Once again, the head is initialized with new weights. Likewise, it will assess the influence of freezing this backbone for N epochs whilst training the head, before unfreezing the backbone and fine-tuning the model. Subsequently, it will study the impact of freezing the backbone during the entire training while training the predictor head.

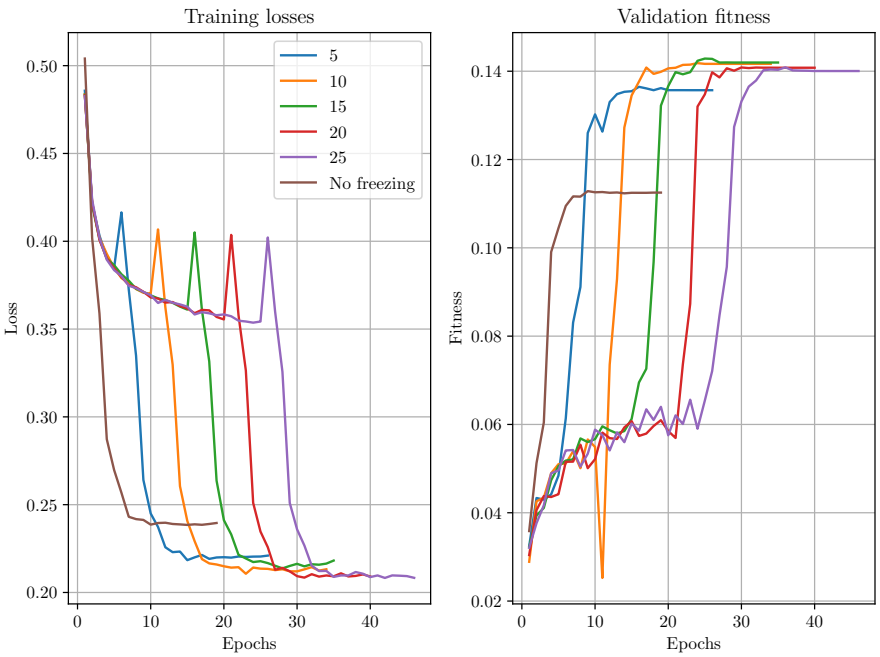


Figure 5.6: Training loss and validation fitness for Faster R-CNN initialized with COCO pre-trained weights, with the backbone and RPN being frozen for N epochs vs no freezing.

5.3.1 Faster R-CNN - COCO pre-trained weights

In this section, experiments will be conducted on a Faster R-CNN detector initialized with COCO pre-trained weights. As was the case before, the RPN will also be initialized with such weights and only the predictor head will be replaced by a new one. The goal is to study the impact on performance when using a pre-trained backbone and RPN as is (for N epochs) before fine-tuning them on paintings, as was done in the previous section. It is natural to start from pre-trained weights for the RPN, even though it is not responsible for feature extraction, since its influence was clearly visible on Figure 5.1. Even though we can expect that region proposals tuned for natural images will not be of the finest quality for paintings, we still arbitrarily decide to freeze the pre-trained RPN since it will be fine-tuned anyway after N epochs and by doing so, we will be able to assess through the fitness score the adequacy of these generated regions. Note that, moving away from the idea of freezing pre-trained components, we could have investigated similar experiments where both the backbone and RPN are initialized with COCO pre-trained weights but where, this time, only the backbone is frozen for N epochs and the RPN would thus be fine-tuned from the very beginning. Due to time and computational constraints, this could not have been explored.

Given the relatively small size of training data, freezing will be tested for $N = 5, 10, 15, 20$ and 25 consecutive epochs before unfreezing the affected layers. Compared to the previous scenario where full fine-tuning was carried out, here only total loss values will be reported for each epoch to compare the influence of N rather than the freezing influence for a particular value of N. Training loss and validation fitness are displayed on Figure 5.6. As can be observed, all models eventually converge and they all reach a final loss value lower than that of their equivalent obtained in the previous section. The same observation can be made for validation fitness scores that are all substantially higher. We can thus expect that

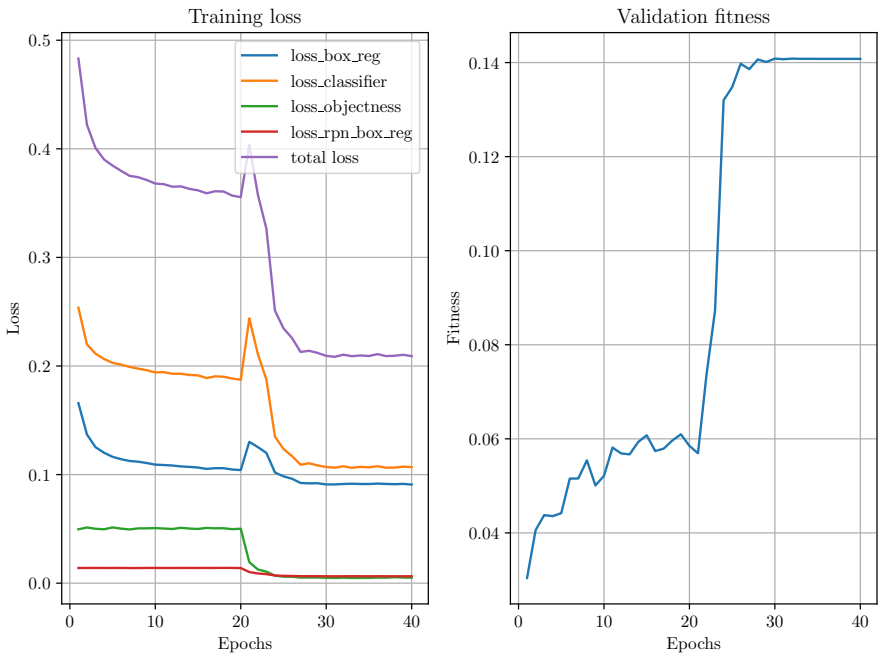


Figure 5.7: Training loss and validation fitness for Faster R-CNN initialized with COCO pre-trained weights, with the backbone and RPN being frozen during 20 epochs.

any such model will eventually perform better than the fully fine-tuned model, although this hypothesis has to be checked on the test data with our mAP @.5 reference metric since it is possible that the main difference in validation scores between both scenarios is attributable to the mAP @[.5, .95] instead, by definition of the fitness criterion.

It is worth noting that there is a visible peak in training losses at the epoch right after the backbone and RPN are unfrozen, and this peak increases as we wait longer before unfreezing the backbone and the RPN. This can be attributed to the sudden training of the unfrozen modules, i.e. the backbone and the RPN, which could be due to several reasons. For instance, it could be caused by the momentum in the optimizer, for which gradients before unfreezing are not cleanly reset, thereby irrelevant gradients are considered during the momentum (and weights) update. Another reason could be linked to batch normalization layers in the lower modules. For example, if their weights (β and γ) are updated for the second part of training, it could ruin the features learnt so far by the higher layers. This peak can be visualized on individual loss components, as is illustrated for $N = 20$ in Figure 5.7. From this figure, we can observe that the loss components corresponding to the RPN stagnate around their initial values during their frozen epochs and suddenly decrease as they are unfrozen. Thus, the spike in total loss is caused by the predictor head and is mostly attributable to the classification loss.

Nevertheless, the assumptions put forward remain speculations and, due to time constraints, the reason for this spike could not be investigated any further. Still, it can be noted that this behavior has already happened for a similar situation[3] where lower layers start frozen and are unfrozen after a certain amount of epochs.

However, it appears clear that unfreezing the backbone and the RPN to enable their

[3] https://forums.fast.ai/t/learn-unfreeze-causes-spike-in-loss-when-training-unet-on-image-segmentation-challenge/21460

fine-tuning eventually helped the model to reach losses that are significantly smaller than those that would be obtained if freezing was continued indefinitely. In a next step, we will consider an approach where the backbone and the RPN are initialized with COCO pre-trained weights but where only the backbone is frozen during the whole training, corresponding to the best performing setting in [56]. In this way, we will be able to assess the impact of (un)freezing the feature extracting module and we will also be in state to verify that there exists a substantial gap between losses and performance obtained in both settings. Note that the training loss for that particular approach will not necessarily line up exactly with the loss trend observed during backbone and RPN freezing, since only the backbone (and no longer the RPN) will be frozen for all epochs.

As was evoked, freezing the pre-trained RPN allows, to some extent, to assess the quality of the regions proposed during training. This quality is reflected directly into the corresponding training loss components (`loss_objectness` and `loss_rpn_box_reg`) but it also translates into the validation fitness at the corresponding epochs. We can observe from Figure 5.7 that the losses achieved by the RPN before unfreezing are considerably higher, mostly due to its objectness component. Thus, it means that the regions proposed fit quite well the ground truth objects but the RPN has a harder time differentiating positive boxes (objects) from negative boxes (background). Note that, since each network is initialized with the same COCO pre-trained weights, loss values for `loss_objectness` and `loss_rpn_box_reg` will be identical whatever the value of N as long as the RPN remains frozen. Furthermore, it can be observed from the fitness scores that freezing the RPN with pre-trained weights leads to proposals of sufficient quality in order to improve the entire detection pipeline, but fine-tuning it after unfreezing leads to a much higher fitness, significantly higher than when full fine-tuning is performed without any freezing. However, we cannot claim from this figure whether this improvement mainly comes from the fine-tuning of the RPN or from the fine-tuning of the backbone. This will be made possible when investigating the setting where the backbone remains frozen during the whole training with the RPN being fine-tuned already at the very first epoch. Nevertheless, we can still conclude that this specific freezing-unfreezing technique leads to a large increase in fitness, meaning that better regions are generated, which is visible from the losses in Figure 5.7, but also that the predictor head becomes finer. As was already debated in previous paragraphs, a larger validation fitness score does not necessarily imply larger mAP @.5; therefore this metric will be computed for each model on the test set, along with the average precision for each animal.

Test set results for these freezing experiments are reported in Table 5.13. As a general observation, we can see that the trend that emerged for the validation set is confirmed for the test set. Indeed, models for which the backbone and the RPN were frozen and then unfrozen consistently outperform their counterpart with no freezing involved, the best setting ($N = 25$) outperforming it by 2.56%. Looking at each class separately, we can notice huge improvements for two specific animals, whatever the number of freezing epochs: bats, moving away from an AP of 0.00% to one around 15.00% and leopards, previously with a 4.31% AP and now with an AP of 32.20% for $N = 25$. For the latter, it seems that the longer both components are frozen, the more its average precision increases. This could mean that, for these animals, the model first needs to learn to localize and classify them using features built from natural images before incorporating new features related to the artistic style. Still, we can observe a similar degradation of performance as regards animals intersecting with COCO classes, as was already underlined for previous

	mAP @.5	Ass	Bat	Bear	Bull	Camel	Cat	Cow	Deer	Dog	Elephant	Fox	Goat
No freezing	12.30	**14.95**	0.00	**13.65**	30.46	3.40	3.87	21.90	18.60	33.66	**45.22**	6.08	7.57
$N = 5$	13.90	8.18	14.39	13.2	31.78	2.82	4.22	**23.25**	17.52	33.04	40.63	13.23	**10.32**
$N = 10$	14.54	11.78	16.41	13.42	33.07	3.15	3.05	22.70	20.15	**35.31**	44.67	12.68	6.38
$N = 15$	14.13	10.49	**18.66**	8.89	29.73	**4.36**	3.15	22.59	18.48	34.41	40.78	11.14	7.72
$N = 20$	14.80	9.44	14.85	12.96	35.06	3.12	**4.95**	20.41	**22.21**	33.64	44.78	12.75	7.85
$N = 25$	**14.86**	12.68	14.29	12.88	**35.40**	4.22	4.23	22.21	19.89	34.95	44.78	**15.79**	8.65

	Horse	Leopard	Lion	Mouse	Pig	Primate	Rabbit	Rat	Sheep	Squirrel	Tiger	Boar	Wolf
No freezing	21.82	4.31	0.57	0.00	1.87	18.16	12.20	1.99	10.54	2.05	30.75	3.45	0.45
$N = 5$	22.34	23.53	**0.94**	0.28	2.50	19.91	9.68	2.71	**12.35**	5.26	30.12	2.87	2.42
$N = 10$	**23.74**	24.31	0.70	**0.93**	2.96	19.58	13.41	3.03	11.32	**5.48**	**33.16**	1.72	0.28
$N = 15$	23.47	28.07	0.68	0.00	3.44	18.29	13.42	3.12	11.17	4.57	33.12	2.11	1.45
$N = 20$	21.58	28.65	0.68	0.00	3.97	**20.51**	**13.92**	**4.03**	10.34	3.85	32.97	**5.17**	**2.24**
$N = 25$	20.49	**32.20**	0.50	0.00	**4.08**	20.34	13.43	2.28	10.29	2.42	32.74	1.72	1.10

Table 5.13: Performance of a fine-tuned Faster R-CNN with a COCO backbone frozen for the first N epochs, on the paintings test set.

fine-tuned versions of Faster R-CNN and YOLO.

Freezing during the entire training (backbone only)

We will now consider a similar Faster R-CNN detector, whose backbone and RPN are initialized with COCO pre-trained weights but where only the backbone will be frozen. Thus, the RPN will get fine-tuned from the very beginning and the backbone will remain frozen through the entire training stage. We will thereby be in state to judge whether using an object detector (almost) off-the-shelf indeed yields better performance than the previously considered freezing and unfreezing technique, as claimed in [56]. Although not explicitly mentioned in [56], we decide to initialize the RPN with COCO pre-trained weights instead of training it from scratch since we have observed from Figure 5.1 that this leads to lower training losses.

From Figure 5.8, we can observe that the training loss for the setting where only the backbone is frozen but for the entire training eventually converges to a significantly higher loss value than settings where the backbone is eventually fine-tuned, after being frozen or not. As was also expected, the loss trend does not exactly line up with the one observed for the N-epochs-frozen settings since the RPN gets fine-tuned from the beginnning of training, thereby vastly reducing its corresponding loss as could be seen from Figure 5.1 and 5.7, inducing a reduction of total loss.

As regards validation fitness, the model reaches a very low fitness score, close to those obtained during the freezing stage of the backbone and of the RPN, for a number of epochs greater than 15. However, we can notice that the model achieves its peak performance in a significantly smaller amount of epochs if only its backbone is frozen and not if both the backbone and the RPN are. This highlights the impact of fine-tuning the RPN from the start and not after $N \geq 15$ epochs, emphasizing the fact that studying the scenario with a COCO pre-trained backbone and RPN where only the backbone is frozen for N epochs is an interesting path to explore.

This approach is also assessed on the paintings test set, on which it achieves an mAP @.5 equal to 7.01%, thereby endorsing the fact that it produces worse predictions than any freezing-unfreezing setting.

We can thus conclude that freezing the backbone during the entire training is less bene-

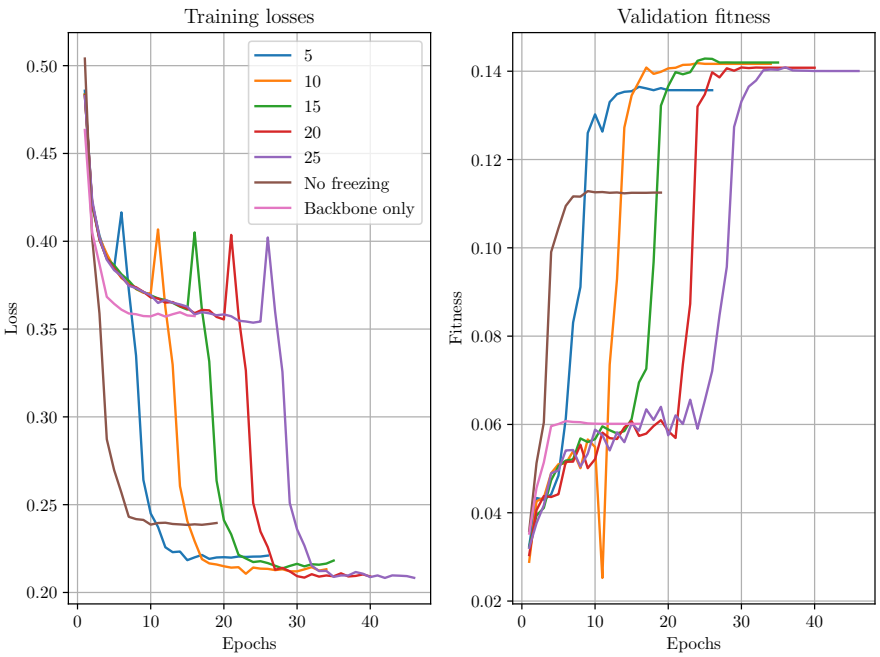


Figure 5.8: Training loss and validation fitness for Faster R-CNN initialized with COCO pre-trained weights, with the backbone and RPN being frozen for N epochs vs no freezing vs backbone freezing during the entire training. "Backbone only" represents the case where the backbone is frozen during the entire training.

ficial than freezing and unfreezing lower layers, contradicting what was observed in [56]. Freezing the backbone for all epochs also yields worse downstream performance than that of the fully fine-tuned detector, which could be expected since no paintings features would be incorporated in the backbone. Finally, given that the fitness score for the setting with only the backbone being frozen quickly stagnates while the RPN and predictor head are being fine-tuned, we can argue that unfreezing the backbone, as was done in the freezing-unfreezing setting along with the unfreezing of the RPN, contributes the most to the improvement that could be observed in Figure 5.6. This thus suggests to try the aforementioned approach of freezing and unfreezing only the backbone, and no longer the RPN.

5.3.2 Faster R-CNN - Fine-tuned ResNeXt backbone

In this section, similar freezing experiments will be carried out on a Faster R-CNN model but now considering as backbone the fine-tuned ResNeXt. In this case, the RPN has to be initialized with new weights and, therefore, only the backbone will be frozen for N epochs, which differs from the previous setting. Identically to the experiments ran considering COCO pre-trained weights, the backbone will be frozen for $N = 5, 10, 15, 20$ and 25 epochs before getting unfrozen. Since the RPN is initialized with new weights, it was decided not to freeze it, although it does not correspond to the setting considered for COCO pre-trained weights in Section 5.3.1, because the main idea was to study the influence of freezing the pre-trained modules.

Training loss and validation fitness can be visualized on Figure 5.9, where both quantities for their no freezing equivalent are also reported. A similar trend as with COCO pre-

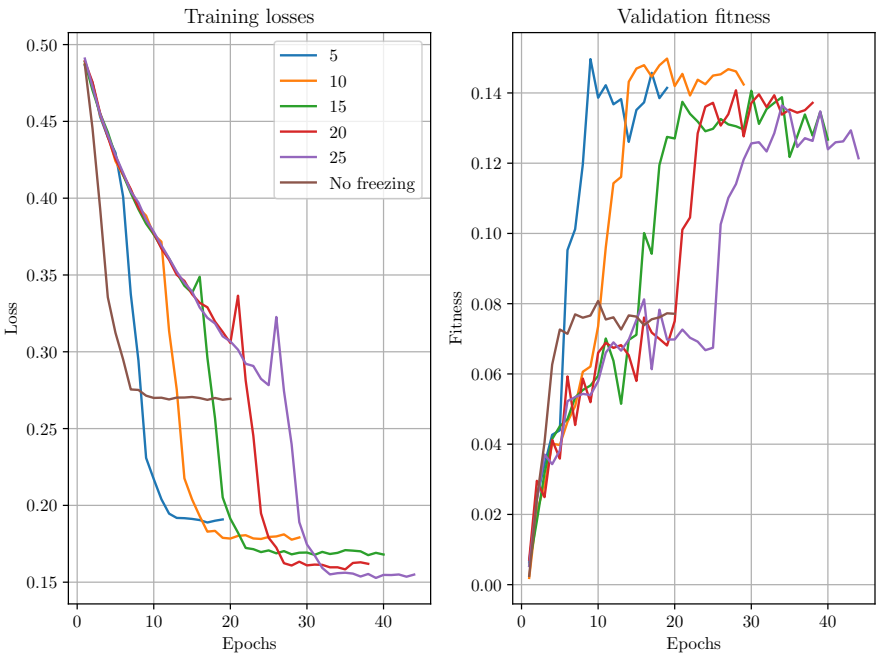


Figure 5.9: Training loss and validation fitness for Faster R-CNN with a paintings fine-tuned ResNeXt backbone, the latter being frozen for N epochs vs no freezing.

trained weights can be observed, where the freezing-unfreezing approach reaches lower loss values and higher fitness scores. However, there is now a much higher difference in loss (and fitness) values when comparing with the no-freezing setting than the difference obtained starting from COCO pre-trained weights. Thus, we could expect these models to perform much better than their no-freezing equivalent, although this will have to be checked on the test set. Note that, as was the case previously, peaks are visible in the training losses at the epoch following the unfreezing. Nevertheless, they now appear only for $N \geq 15$ while they already appeared for $N = 5$ with a COCO pre-trained backbone and RPN. In addition, the longer we wait before unfreezing the backbone, the higher the peak but also the lower the final loss value, which was also the case before. Furthermore, it appears clear that peaks obtained with a paintings-fine-tuned ResNeXt backbone are much smaller than those obtained, at equal epochs, with a COCO pre-trained backbone and RPN.

It seems that starting from COCO pre-trained weights or from a fine-tuned ResNeXt backbone eventually yields similar fitness values when freezing for N epochs, as can be visualized in Figure 5.10. However, as could be observed in both no-freezing settings, the test mAP @.5 for the ResNeXt backbone was higher than that of the COCO backbone, while the latter achieved a much higher validation fitness score than the former. There, metrics were also computed on the validation set to see whether this was due to chance, which turned out to be true. Therefore, both freezing-unfreezing approaches will be compared on the validation set as well to see if any difference in mAP @.5 performance on the test data also translates to the validation data.

Test set results for this ResNeXt freezing-unfreezing setting are reported in Table 5.14. Once again, we can observe that such an approach improves the downstream test set mAP @.5, this time even outperforming the no-freezing counterpart by 8.45%. We can

	mAP @.5	Ass	Bat	Bear	Bull	Camel	Cat	Cow	Deer	Dog	Elephant	Fox	Goat
No freezing	13.00	8.09	0.00	8.16	25.70	**9.83**	15.36	14.68	20.23	31.54	32.06	15.28	12.04
$N = 5$	**21.45**	**22.80**	9.76	16.25	**33.21**	9.35	**18.11**	23.88	26.18	34.35	45.80	33.11	**29.00**
$N = 10$	20.28	19.49	14.39	14.87	27.87	7.20	14.58	20.82	**27.79**	**35.02**	46.08	**33.35**	20.17
$N = 15$	20.61	17.37	**16.47**	**18.93**	31.22	8.72	10.57	**26.29**	24.98	29.59	**47.01**	30.19	21.96
$N = 20$	18.42	16.21	10.47	13.76	29.31	7.19	13.08	20.07	27.15	31.49	39.27	24.08	23.18
$N = 25$	19.52	17.29	3.70	14.77	30.02	9.59	11.21	24.02	20.93	31.55	43.48	33.27	23.64

	Horse	Leopard	Lion	Mouse	Pig	Primate	Rabbit	Rat	Sheep	Squirrel	Tiger	Boar	Wolf
No freezing	30.84	5.10	5.78	0.00	0.92	18.30	**27.58**	**4.05**	19.10	0.00	20.29	0.00	0.00
$N = 5$	29.02	**43.07**	**22.39**	1.73	5.06	19.42	22.17	3.03	**26.65**	9.92	**37.60**	4.25	**10.06**
$N = 10$	26.61	37.27	20.90	2.59	**7.80**	17.42	24.88	2.81	24.19	**13.73**	34.87	3.45	8.94
$N = 15$	**32.15**	47.87	13.27	**4.47**	6.29	**20.14**	20.37	1.44	22.86	13.12	35.05	6.32	8.57
$N = 20$	23.77	37.19	18.05	2.67	5.25	18.53	22.06	1.89	20.95	10.32	34.41	4.31	5.51
$N = 25$	27.41	43.10	19.80	2.14	6.70	15.08	23.81	1.45	24.28	13.34	33.77	**8.33**	5.31

Table 5.14: Performance of a fine-tuned Faster R-CNN with a fine-tuned-ResNeXt backbone frozen for the first N epochs, on the paintings test set.

thus conclude that freezing the backbone for a few epochs leads to significant improvements compared to a model fine-tuned from the very beginning. This also confirms the intuitive hypothesis that a model built upon a backbone fine-tuned on paintings should perform better than one built upon a backbone pre-trained on photo-realistic images, since it already incorporates relevant features.

	Validation		Test	
	COCO	ResNeXt	COCO	ResNeXt
$N = 5$	22.35	29.94	13.90	21.45
$N = 10$	22.91	29.88	14.54	20.28
$N = 15$	22.65	28.00	14.13	20.61
$N = 20$	22.62	28.54	14.80	18.42
$N = 25$	22.68	27.94	14.86	19.52

Table 5.15: Validation and test performance (mAP @.5) comparison for freezing-unfreezing Faster R-CNN with a COCO pre-trained/fine-tuned ResNeXt backbone.

These results can be put in contrast with those obtained for identical settings but starting from a COCO pre-trained backbone and RPN, and are reported in Table 5.15. From this table, it appears clear that the trend observed for the test set also holds for the validation set. Therefore, freezing-unfreezing should be considered for any backbone but a fine-tuned ResNeXt backbone should be preferred if we want to maximize the mAP @.5. Since both backbones achieve relatively similar fitness scores, as is illustrated in Figure 5.10, we can deduce that they only reach comparable performance for the mAP @[.5, .95], although it should be noted that both models could show different performance for each of these thresholds, as this represents an average over 10 IoU thresholds. Note that this performance gap can be due to the fact that the ResNeXt backbone already incorporates paintings-specific features, but it could also be due to the fact that the RPN was frozen for N epochs.

Freezing during the entire training

In a similar manner to what was carried out for the Faster R-CNN with COCO pre-trained weights, a setting where the backbone remains frozen throughout training is considered,

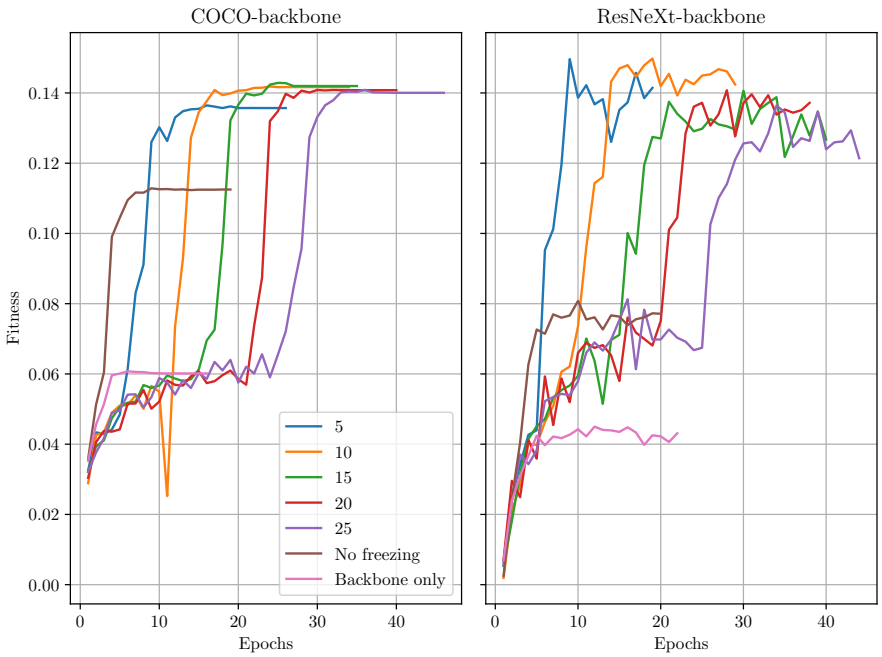


Figure 5.10: Validation fitness comparison for Faster R-CNN with two different backbones. "Backbone only" refers to the setting where the backbone remains frozen during the entire training, with all other components being fine-tuned.

which can be called off-the-shelf. This will allow us to compare what was highlighted with the COCO backbone equivalent to see whether it also performs worse than the freezing-unfreezing scenario.

As can be observed from Figure 5.11 and 5.10, freezing the backbone during the whole training yields significantly higher losses and significantly lower fitness scores, which was already observed with a COCO pre-trained backbone. We can thus expect its performance to be worse than that of any Faster R-CNN model encountered so far, since it achieves the lowest validation fitness.

When evaluated on the test set, this model achieves an mAP @.5 equal to 8.61%, which surprisingly outperforms its COCO equivalent by 1.60%. Since an identical scenario occurred between both backbones for the no-freezing setting, performance on the validation set will be assessed for both settings where freezing is carried out throughout training. On the validation set, the Faster R-CNN with COCO backbone achieves an mAP @.5 of 10.01% while its ResNeXt counterpart reaches 12.15%. Therefore, even though the validation fitness of the latter is lower than that of the former, it still performs better for an IoU threshold of 0.5. This also means that its performance with higher thresholds will necessarily be worse than that of the model with the COCO backbone.

Finally, we can underline the fact that using the object detector with a paintings fine-tuned ResNeXt backbone off-the-shelf does not lead to gains of performance compared to freezing-unfreezing settings.

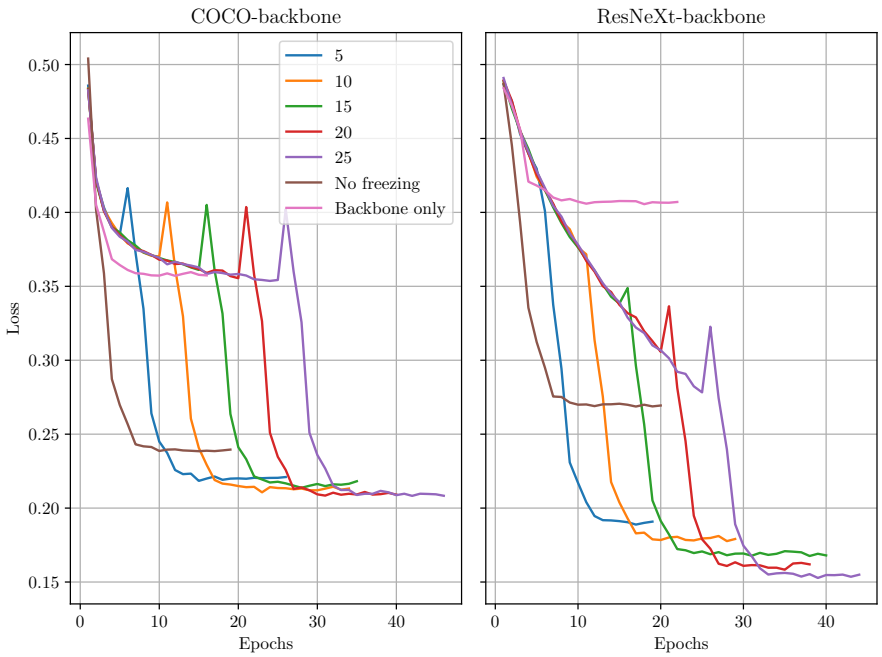


Figure 5.11: Training loss comparison for Faster R-CNN with two different backbones. "Backbone only" refers to the setting where the backbone remains frozen during the entire training, with all other components being fine-tuned.

5.3.3 Freezing Faster R-CNN: Conclusions

As could be observed through the different freezing experiments conducted on Faster R-CNN, results are of similar nature whatever the considered backbone, i.e. starting from a paintings fine-tuned ResNeXt backbone or from a COCO pre-trained backbone and RPN. In both cases, the highest improvements are achieved when the backbone (and RPN, for the COCO pre-trained Faster R-CNN) is frozen for N epochs and then unfrozen in order to let it fine-tune on paintings.

It can be noted that no common value of N leads to the finest enhancements across both backbone settings. Indeed, for the COCO pre-trained version, mAP @.5 performance seems to improve as we wait longer before unfreezing the backbone and RPN. On the other hand, for the ResNeXt version, it rather seems to decrease as N increases. There is also a large difference in the improvements across backbones since the setting with the ResNeXt backbone improves by 8.45% compared to its no-freezing equivalent while the setting with COCO pre-trained weights only improves by 2.56%.

Whatever the considered backbone, freezing it during the entire training leads to the worst results and to the highest losses, which could somehow be expected since no artistic features are incorporated during training.

Similar experiments where the COCO pre-trained RPN is replaced by a new RPN with randomly initialized weights (as is the case with the ResNeXt backbone) could be considered to investigate whether using a pre-trained RPN eventually results in a learning bottleneck compared to a "from scratch" RPN, since performance achieved with a pre-trained RPN is nowhere near that of the model relying on a ResNeXt backbone, which does not start from pre-trained weights for the RPN. However, as could be observed from

Figure 5.1, starting from pre-trained weights leads to lower losses for the RPN, hence it should not represent a bottleneck in the learning process. We can thus claim that the model that uses the ResNeXt backbone mainly benefits from the fact that its backbone has already embedded paintings-specific features compared to a COCO pre-trained backbone which only incorporates features from natural images. Another sanity check to verify this claim would be to run the same freezing-unfreezing experiments with a COCO pre-trained backbone and RPN where only the backbone would get frozen for N epochs. If we then observe that the final performance for the latter is even with that of the best ResNeXt-initialized freezing-unfreezing setting, it would mean that the main benefit does not necessarily come from the ResNeXt backbone but rather from the fact that the RPN is not frozen.

5.3.4 YOLO - COCO pre-trained weights

In a similar philosophy as for Faster R-CNN, we can initialize YOLO's backbone with COCO pre-trained weights and initialize its head with new weights. Note that this differs from the full fine-tuning setting considered previously since the head was also initialized with pre-trained weights at that moment. Here, the idea is to study the influence of freezing and unfreezing the backbone along with the influence of the backbone itself; therefore it made more sense to start from a new head, as was done for Faster R-CNN.

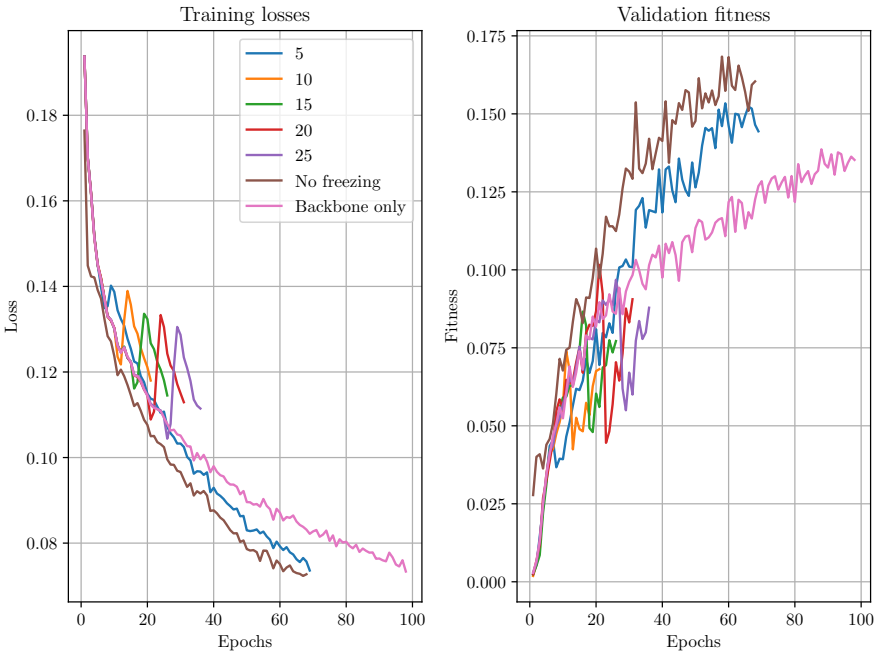


Figure 5.12: Training loss and validation fitness for YOLO, with a backbone initialized with COCO pre-trained weights and frozen for N epochs vs no freezing vs freezing during the entire training. "Backbone only" refers to the setting where the backbone is frozen for the whole training.

Once again, freezing will be considered for $N = 5, 10, 15, 20$ and 25 epochs before allowing the fine-tuning of the backbone. Training loss and validation fitness are displayed on Figure 5.12. From this figure, we can observe that only three settings achieve relatively comparable performance: no-freezing (full fine-tuning obtained in Section 5.2.2), freezing

for 5 epochs and backbone freezing throughout training, which will be discussed later. All the other freezing-unfreezing settings do not manage to surpass a validation fitness of 10%, therefore their performance can only be expected to degrade.

These poor performing settings also stop training quicker than the other ones. In fact, they all stop training after the first ten epochs that follow unfreezing since no improvement is observed based on the fitness criterion. We can observe that, for the epoch following the unfreezing, validation fitness increases but then suddenly drops and it is impossible to catch up for the next ten consecutive epochs. This also translates into the training losses, where we can see, e.g. for $N = 20$, that the loss decreases at the 21st epoch but then suddenly increases, reaching a peak that is too hard to recover in ten epochs. The longer we wait before unfreezing, the higher the peak, which explains why the setting with $N = 5$ could recover and continue its training, although never reaching a loss nor validation fitness better than those obtained for the full fine-tuning setting, which is different than what was observed for Faster R-CNN where the no-freezing setting performed the worst. These peaks were also observed for Faster R-CNN but the model always managed to catch up. Similar hypotheses can be evoked, i.e. relating to momentum and batch normalization layers trained after unfreezing, but we can add the possibility that the network started from a too high learning rate after unfreezing, which could have made it bifurcate from its original path derived during the freezing stage. One could argue that this effect is dampened for $N = 5$ because the learning rate at the fifth epoch did not have the time to decrease too much thanks to learning rate scheduling. This is not the case for Faster R-CNN where learning rate scheduling is only involved after unfreezing lower layers; the learning rate thus remains constant during the freezing stage.

	mAP @.5	Ass	Bat	Bear	Bull	Camel	Cat	Cow	Deer	Dog	Elephant	Fox	Goat
No freezing	18.30	**19.60**	0.88	9.01	31.60	5.31	9.88	22.30	**27.40**	**34.80**	**41.00**	13.30	**19.20**
$N = 5$	**18.40**	18.50	**4.88**	**12.20**	**35.50**	**5.97**	7.98	**28.80**	20.10	32.70	36.70	**21.30**	17.20
$N = 10$	10.10	7.92	0.23	3.53	27.60	1.28	9.08	20.90	13.80	32.10	27.60	1.16	9.09
$N = 15$	11.30	11.20	0.27	7.23	25.80	3.95	**10.00**	22.30	12.50	28.60	37.60	1.75	9.50
$N = 20$	12.20	9.58	0.24	13.10	25.80	1.35	8.87	27.00	18.60	32.50	32.60	4.75	8.70
$N = 25$	12.90	17.70	0.32	10.10	23.40	1.63	8.26	24.50	17.20	26.90	40.80	4.95	9.07

	Horse	Leopard	Lion	Mouse	Pig	Primate	Rabbit	Rat	Sheep	Squirrel	Tiger	Boar	Wolf
No freezing	**32.10**	**28.50**	**9.15**	1.13	7.41	**31.00**	**21.40**	7.07	21.90	10.20	**35.20**	**8.89**	**9.65**
$N = 5$	31.30	25.50	6.34	1.38	**8.29**	28.30	21.30	**7.48**	**23.70**	**18.00**	29.80	7.65	8.76
$N = 10$	29.60	8.26	1.67	0.48	2.77	18.70	4.38	1.82	8.60	0.67	17.10	2.76	1.49
$N = 15$	27.80	11.70	0.93	1.04	4.71	19.60	9.54	2.57	11.00	1.06	16.00	3.07	2.31
$N = 20$	28.80	14.10	1.07	**1.94**	3.46	16.20	12.70	2.21	15.20	3.05	14.80	4.91	4.62
$N = 25$	31.80	17.30	1.88	1.33	7.91	14.80	18.30	3.97	10.30	6.14	19.00	2.68	3.38

Table 5.16: Performance of a fine-tuned YOLO with a COCO backbone frozen for the first N epochs, on the paintings test set.

From the validation fitness, we could expect all freezing-unfreezing models to be outperformed by their fully fine-tuned counterpart. Results on the test set are reported in Table 5.16. From these, we can notice that, as was expected, settings with $N \geq 10$ perform very poorly compared to $N = 5$ or when no freezing is involved. We can however notice that, the longer we wait before unfreezing, the higher the mAP @.5. This can be explained by the fact that YOLO's head is composed of several layers compared to Faster R-CNN whose head, i.e. the Fast R-CNN predictor, is only composed of two linear layers. Therefore, it is natural that waiting longer before unfreezing (and, to some extent, before stopping training, since unfreezing results in model degradation) allows for a better tuning of the head's weights.

	Test set	Validation set
No freezing	18.30	27.70
$N = 5$	18.40	27.00
$N = 10$	10.10	14.50
$N = 15$	11.30	16.60
$N = 20$	12.20	19.20
$N = 25$	12.90	17.90

Table 5.17: Performance of YOLO on the paintings validation and test sets, for a backbone initialized with COCO pre-trained weights (fully fine-tuned vs frozen for N epochs).

We can notice that the best mean average precision is obtained for the setting with $N = 5$ and not for the fully fine-tuned model, as could be expected from Figure 5.12. As a sanity check, models will also be assessed on the validation set, as was already carried out multiple times for previous Faster R-CNN settings. Validation results are reported in Table 5.18. We can observe from these results that performance indeed deteriorates for $N \geq 10$ but that models improve if we wait longer before unfreezing. In addition, it also tells us that the fully fine-tuned YOLO model outperforms by 0.70% the setting with $N = 5$, which contradicts the observation made for the test set. In any case, both models perform similarly well, and models for which freezing is carried out for too long perform much worse. This can be put in contrast with the results obtained for Faster R-CNN considering a backbone and RPN initialized with COCO pre-trained weights, for which it was concluded that this freezing-unfreezing technique was successful enough to improve downstream mean average precision performance.

Freezing during the entire training

YOLO can also be trained off-the-shelf with COCO pre-trained weights to assess whether this approach yields better results than the previous freezing-unfreezing one. From Figure 5.12, we can see that this is indeed the case, at least comparing with settings for which N is not equal to 5. Comparing to the latter, its performance seems a little bit worse. In addition, the fully fine-tuned YOLO model seems to outperform all other considered settings.

It is also worth noting that this setting took the most time to finish training. However, we can see that it tends to converge to loss values slightly worse than those obtained for the full fine-tuning setting, while validation fitness seems much worse compared to its fine-tuning counterpart. Furthermore, we can see that, whatever the considered setting, training is stopped before losses have converged on the training set. This seems mainly due to YOLO's learning behavior as regards the validation fitness, which is quite messy.

Validation and test set performance for this setting is reported in Table 5.18 where it is compared with the performance obtained by its fully fine-tuned equivalent and the setting with $N = 5$. We can see that it achieves worse mean average precision scores whatever the assessment data set and should therefore not be preferred. However, it is worth noting that freezing the backbone during the entire training leads to a dampened degradation of model performance compared to settings where unfreezing is considered. Keeping in mind previous assumptions about reasons for the observed loss peaks, this behavior can be expected since we will never mess with the gradients history nor with

	Test set	Validation set
No freezing	18.30	27.70
$N = 5$	18.40	27.00
Entire training	15.00	23.80

Table 5.18: Performance of YOLO on the paintings validation and test sets, for a backbone initialized with COCO pre-trained weights, fully fine-tuned, frozen for 5 epochs or frozen during the entire training.

batch normalization layers present in the backbone, which remain frozen indefinitely. It thus has no peak to recover from and can continue training freely. In addition, the head trained from scratch will be trained longer and therefore be more performant.

In general, we can observe that, for all settings previously considered, training seems to not have completely converged. This is trivial to verify for the freezing-unfreezing approach, but for the other techniques, we can argue that the model could have achieved lower losses and higher validation fitness scores if it had the opportunity to continue training any further. This could be checked by running similar experiments and dropping the early stopping criterion which cuts off training whenever no validation fitness improvement is seen in ten consecutive epochs. Of course, we would still keep as final model the one that achieved the highest validation fitness. Please note that these experiments were not run due to time and computational constraints, as was already the case previously with the COCO pre-trained version of Faster R-CNN.

5.3.5 YOLO - Fine-tuned ResNeXt backbone

Instead of starting from COCO pre-trained weights, we will now begin our freezing-unfreezing experiments with weights obtained for the fine-tuned ResNeXt model, and we will also compare with results obtained for the (almost) fully fine-tuned version of this model, where only the head was trained from scratch. In these experiments, contrary to those conducted with COCO pre-trained weights, both the fine-tuned YOLO model and its freezing-unfreezing equivalents will start from a new head. In the previous scenario, only the fully fine-tuned version started from a pre-trained head too.

Freezing epochs will be the same as before. The corresponding training loss and validation fitness curves are represented in Figure 5.13. In opposition to the freezing-unfreezing approach carried out with COCO weights, we can see that the setting with $N = 10$ also manages to recover from the loss peak, ending at very close loss values to the fine-tuning setting, which also translates into validation fitness scores. As was the case previously, models with $N \geq 15$ stop training too early but manage to achieve higher fitness scores than their COCO counterparts. This means that using the paintings fine-tuned backbone as starting point is beneficial thanks to the particular features it embeds.

Similarly to Faster R-CNN, peaks in training losses are smaller when the ResNeXt backbone is used rather than COCO pre-trained weights, which explains why the setting with $N = 10$ could recover. Once again, training seems to be stopped too early and it could be expected that models would achieve higher performance with no stopping criterion. If time had permitted, we could have re-run these experiments with no stopping criterion to assess the performance of (the best of) these models after say 300 epochs.

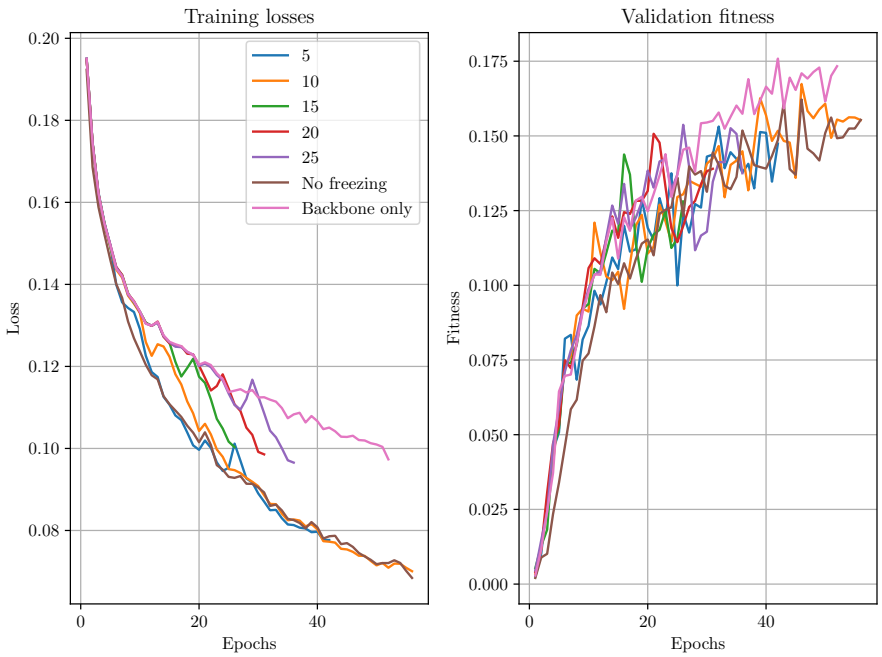


Figure 5.13: Training loss and validation fitness for YOLO, with a paintings fine-tuned ResNeXt backbone frozen for N epochs vs no freezing vs freezing during the entire training. "Backbone only" refers to the setting where the backbone is frozen for the whole training.

Test sets results are available in Table 5.19. Similarly to the previous scenario where COCO weights were used, we can see that the final performance degrades as backbone freezing and unfreezing is applied. Nevertheless, the degradation is less severe with this paintings fine-tuned backbone, emphasizing the fact that starting from such a backbone allows to quickly reach a performance that is comparable to that achieved with a fully fine-tuned YOLO starting from a backbone and head pre-trained on photo-realistic images, although it starts from a head with random weights.

Freezing during the entire training

Finally, the ResNeXt backbone will be frozen throughout training, while the head still starts from new initial weights. For a YOLO detector with a COCO initialized backbone, we observed that freezing the backbone during the entire training also led to a degradation of downstream performance, but it was less severe than when unfreezing was considered.

This final setting can be observed in Figure 5.13, where both training loss and fitness scores are displayed across epochs. Similarly to the same scenario with a COCO backbone, we can notice that it will reach final loss values higher than those obtained when fine-tuning the backbone, which could be expected. It also achieves loss values higher than what is obtained for the freezing-unfreezing setting but seems to achieve a higher final fitness score. This improvement in performance will have to be compared to other settings based on the test set.

From Table 5.20, we can observe that freezing the backbone during the entire training consistently outperforms previously obtained mAP @.5 on both the validation and test

	mAP @.5	Ass	Bat	Bear	Bull	Camel	Cat	Cow	Deer	Dog	Elephant	Fox	Goat
No freezing	**19.30**	14.20	2.91	14.60	27.00	**8.48**	10.60	**30.10**	20.80	37.50	36.20	19.90	21.40
$N = 5$	17.70	**17.00**	8.96	**16.30**	27.50	7.06	9.42	21.80	**25.40**	31.90	32.40	20.00	16.60
$N = 10$	18.70	14.70	**10.80**	14.30	**32.60**	6.42	10.10	26.30	24.00	34.60	37.00	18.30	17.50
$N = 15$	18.80	15.50	5.63	11.20	28.20	4.60	**16.70**	24.90	23.30	36.70	41.30	17.80	22.20
$N = 20$	19.10	15.70	7.67	10.10	31.20	4.84	12.10	23.20	19.80	40.00	37.10	**25.50**	**23.00**
$N = 25$	18.80	8.30	7.40	10.20	32.10	5.19	16.40	27.40	16.50	**37.70**	**44.00**	14.90	22.50

	Horse	Leopard	Lion	Mouse	Pig	Primate	Rabbit	Rat	Sheep	Squirrel	Tiger	Boar	Wolf
No freezing	28.40	**41.90**	**15.20**	0.77	5.74	**28.40**	21.90	5.11	17.20	**17.00**	28.80	**22.10**	6.31
$N = 5$	26.20	34.00	12.60	2.03	4.69	24.30	22.80	4.53	12.90	6.31	29.50	17.70	**9.44**
$N = 10$	31.90	31.50	7.25	**2.86**	3.70	22.60	25.50	6.54	18.20	12.30	33.60	19.30	5.71
$N = 15$	21.50	40.40	13.40	1.75	5.78	22.10	30.20	**9.55**	11.10	3.58	38.50	16.40	8.85
$N = 20$	**32.50**	35.00	14.30	1.96	5.40	17.90	29.10	6.40	**20.80**	3.84	**39.50**	13.80	6.85
$N = 25$	26.70	39.10	14.50	2.26	**7.75**	19.20	**33.70**	9.52	12.00	4.92	32.80	15.80	8.18

Table 5.19: Performance of a fine-tuned YOLO with a fine-tuned-ResNeXt backbone frozen for the first N epochs, on the paintings test set.

	Test set	Validation set
No freezing	19.30	28.40
$N = 20$	19.10	29.90
Entire training	20.10	33.40

Table 5.20: Performance of YOLO on the paintings validation and test sets, for a backbone initialized with COCO pre-trained weights, fully fine-tuned, frozen for 5 epochs or frozen during the entire training.

sets, with an improvement of 0.80% on the latter. This confirms the trend represented in Figure 5.13. Still, we can once again underline the fact that the model could have potentially achieved higher fitness scores, thus higher performance, if training had not stopped based on the stopping criterion. Note that opposite results can be observed for $N = 20$ and the fine-tuning setting between the validation and test results, which illustrates that both achieve similar performance.

5.3.6 Freezing YOLO: Conclusions

Throughout the experiments conducted on YOLO, we have mostly observed close behaviors for both backbones although the corresponding magnitudes are different. Indeed, freezing and unfreezing the backbone generally leads to a degradation of downstream performance. Nevertheless, this degradation is less severe when YOLO starts from a backbone that is already fine-tuned on paintings rather than a backbone pre-trained on natural images. This could seem intuitive since the former already integrates features related to the artistic domain while the latter does not, which greatly helps to overcome the peak happening after unfreezing is conducted.

As concerns the setting where the backbone remains frozen throughout training, it appears that it still deteriorates performance for the COCO initialized YOLO but allows to improve final performance with the ResNeXt backbone. An interesting experiment would be to remove the stopping criterion and freeze the backbone for a high number of epochs, say 50, then to unfreeze it and let training continue, but also to re-run the experiment where the backbone remains frozen during the entire training with no stopping criterion either. This way, we would be able to assess whether there are any gains to eventually

fine-tune the backbone on full size paintings (since the backbone is currently fine-tuned on crops) after the head has had enough time to stabilize. As was already mentioned, we could also run all previous experiments with no stopping criterion, which could not be carried out due to time and computational constraints.

The best resulting YOLO detector corresponds to the model whose backbone is frozen throughout training, as was illustrated before. It outperforms the second best detector (its fine-tuned equivalent) by 0.80% on the test set and by 5% on the validation set.

5.4 Analyzing the final object detector

In this section, we will try to provide some visual intuitions about the predictions made by the best model obtained in the previous experiments, i.e. the Faster R-CNN detector initialized with the ResNeXt backbone that was frozen for 5 epochs. For that purpose, we will provide several visual samples of correct and incorrect predictions and try to underline some interesting patterns.

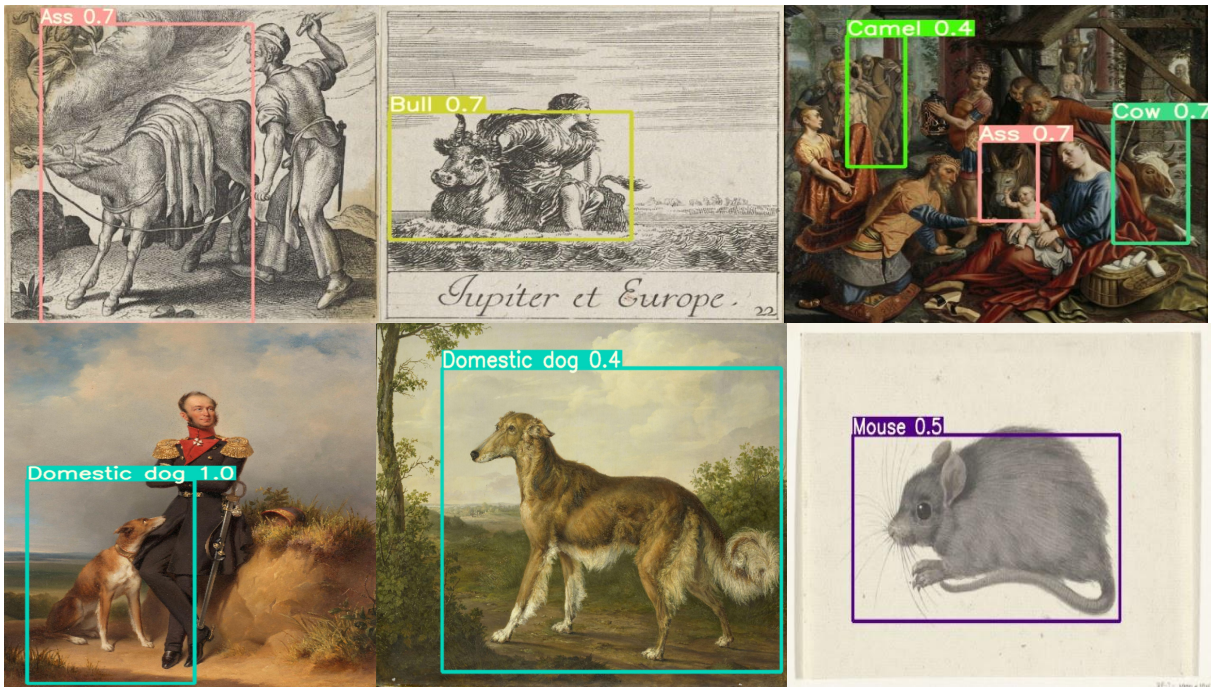


Figure 5.14: Examples of correct detections.

Some examples of correct detections are depicted in Figure 5.14. It is worth noting that the model incorrectly detected a cow instead of a bull in the last painting of the first row. This error points back to the intuitive misclassifications that we could observe for the ResNeXt model in the confusion matrix.

What could also be observed from the predictions made by the model is that it seems to frequently detect humans as primates, as is illustrated in Figure 5.15. Interestingly, they are all standing up, although the last one only focused on the head, similarly to the first one, where both heads are facing towards the observer. It could be insightful to run the detector on photo-realistic images of people standing up or sitting, but also facing the camera. This could be solved by adding a human category.

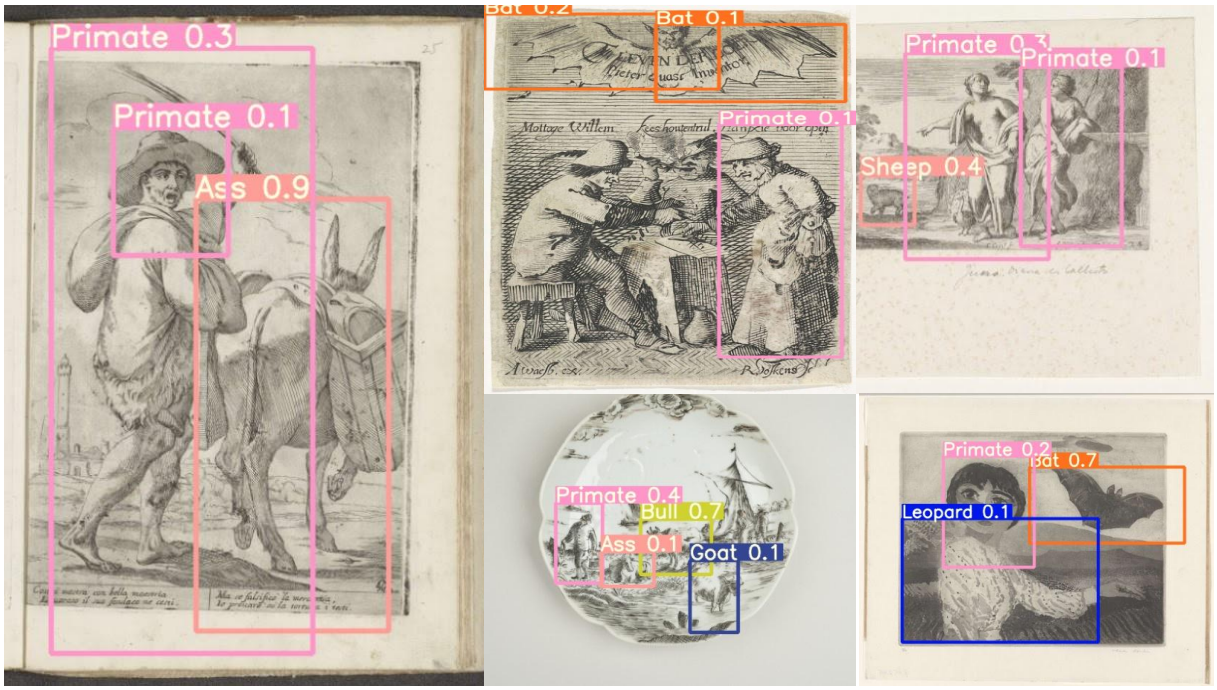


Figure 5.15: Examples of incorrect primate detections.

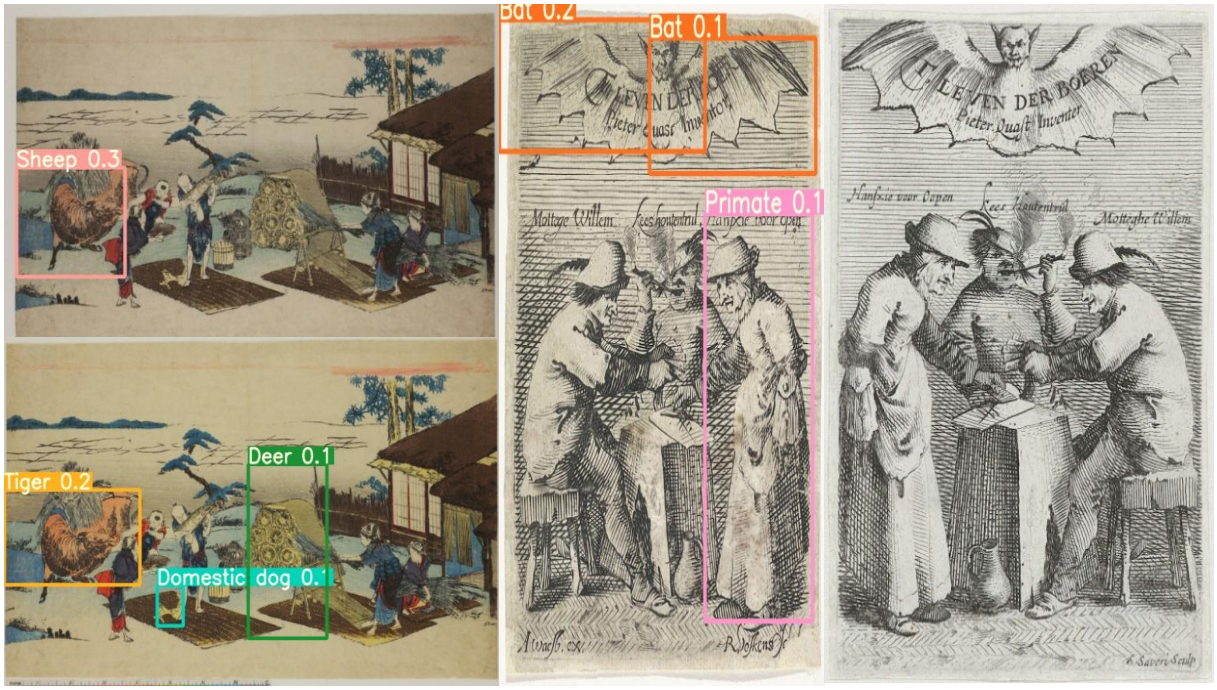


Figure 5.16: Influence of colorimetry on detections.

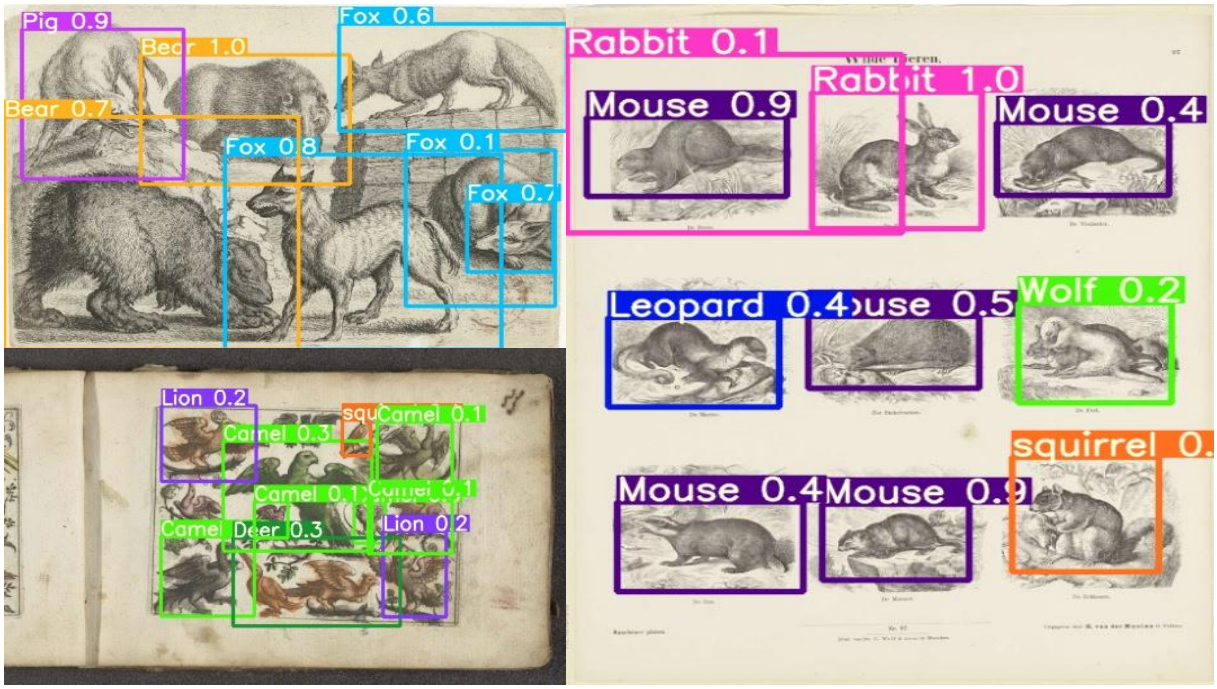


Figure 5.17: Examples of incorrect detections when facing animals that did not appear during training.

We could also observe that the model was sensitive to slight changes in colorimetry and lighting conditions. From Figure 5.16, it can be noticed that the model produces different detections as colorimetry varies. For the detections to the right of the figure, the model is unable to detect the bat depicted on top of the painting. Furthermore, these colorimetry changes also show that the model is not insensitive to symmetry. Indeed, both paintings are horizontally flipped but represent the same scene. However, no detections are produced for the last picture.

In Figure 5.17, we illustrate the behavior of the model when it is provided with artworks that mix animals present in the 25 output classes with animals that are not. The model tries to make sense out of their depictions although it has never seen instances of these animals. For example, in the drawings to the right of the figure, the model correctly detects the rabbit in the first row and the squirrel in the last row but it then detects marmots and beavers as mice, which are in some sense the most resembling animals that it has seen during training. An identical observation can be made for the bottom left depictions, where the model should only detect the bat on top. Instead, it detects all other flying animals except the latter, and classifies them as camels.

Finally, in Figure 5.18, we represent an error pattern that does not appear too frequently but which is still not negligible. This pattern consists in producing multiple bounding boxes for the same animal, resulting in more false positives for the redundant detections. Usually, a non-maximum suppression (NMS) technique is applied to remove redundant bounding boxes. This technique consists in removing lower scoring bounding boxes that have an IoU greater than a specified IoU threshold with a higher scoring box. However, in this work, we applied NMS with an IoU threshold of 0.3, which is already quite strict. We could decrease this threshold even more, but then it could lead to smaller animals not being detected if they are standing in front of another larger animal, since the smaller

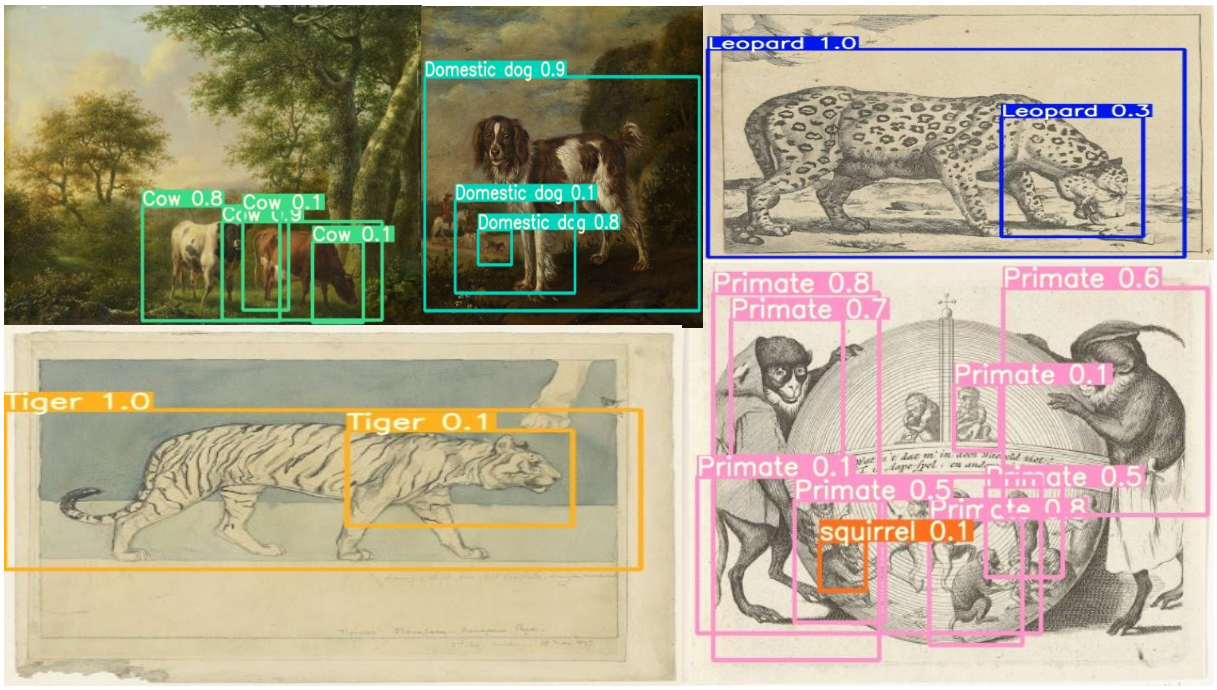


Figure 5.18: Examples of redundant bounding boxes.

bounding box will most likely be included in the larger box. It is worth noting that most duplicate bounding boxes correspond to the head or the legs of the animal, which could indicate that they are representative features to detect these animals.

5.5 Conclusions

In this chapter, we have tried to thoroughly compare two object detection approaches: a region-based object detection technique against a unified regression model. For that purpose, we have opted for Faster R-CNN and YOLO respectively.

As a very first step, we have have assessed their performance as detectors pre-trained on natural images, in particular on the COCO dataset. Only seven animals happened to lie in the intersection with COCO, therefore we could only evaluate them on seven classes. It appeared that YOLO was a clear winner against Faster R-CNN, outperforming it for every class.

Afterwards, we have studied, for each model, the influence of using a backbone (and RPN, in the case of Faster R-CNN) pre-trained on COCO in opposition to a backbone that was already fine-tuned on paintings. In this case, we relied on the convolutional layers of the fine-tuned ResNeXt model obtained in Chapter 4. Each model was then fine-tuned adequatly on paintings and performance was assessed to see whether improvements were indeed visible compared to detectors only pre-trained on COCO.

For Faster R-CNN, we have observed that, contrary to what could have been expected, fine-tuning a COCO pre-trained model yields a slightly higher mAP @.5 than fine-tuning a detector whose backbone already incorporates artistic features, even though we can underline that both approaches reached relatively similar performance. Indeed, the former performed better on the validation set while the latter performed better on the test set.

We have also compared results obtained with the COCO pre-trained version against those obtained for its fine-tuned counterpart to understand the impact of fine-tuning, at least for the animals present in the intersection. There, it was observed that the model improved on three animals but its performance degraded for the four others.

As regards YOLO, no hesitation exists between both backbones: the best mAP @.5 is achieved with the backbone already fine-tuned on paintings, whatever the evaluation set. Contrary to Faster R-CNN, none of both YOLO's fine-tuned models have 0.00% average precision for any animal. Nevertheless, an identical pattern was observed between the COCO pre-trained YOLO model and its fine-tuned equivalent, whose performance also deteriorated for the same animals as for Faster R-CNN. This most likely indicates a case of negative transfer due to the probable variance in artistic styles depicting these particular classes. For both Faster R-CNN and YOLO, Spearman correlation tests have pointed out that there exists a positive (non negligible) correlation between the amount of training occurrences for each animal and their corresponding average precision scores. In addition, they have also underlined that, for animals with smaller bounding boxes, models were more likely to achieve worse performance. Both observations were hypothesized in Chapter 3.

The performance gap that was highlighted between both models when evaluated as COCO pre-trained models, as shown in Table 5.1, persists for their fine-tuned versions, whatever the backbone. The best fine-tuned detector is YOLO with the paintings fine-tuned backbone, reaching an mAP @.5 equal to 19.30% on the test set.

In a slightly different approach, freezing was considered for both detectors, once again for both types of backbone. Concretely, the idea was to see whether freezing the backbone (and RPN, for the case of the COCO pre-trained Faster R-CNN model) for N epochs before unfreezing it could lead to improvements with respect to the settings where both the backbone and the head are directly fine-tuned from the very first epoch. There, a different behavior was observed between both Faster R-CNN and YOLO models.

Concerning Faster R-CNN, freezing for N epochs improved downstream performance compared to direct fine-tuning, whatever the considered backbone. However, a notable difference could be observed in the magnitude of these improvements, as starting from the ResNeXt backbone yields a gain of 8.45% while starting from the COCO backbone (and RPN) only produced a 2.56% improvement on the test set. This underlines the benefits of combining this freezing-unfreezing approach with a backbone already incorporating artistic features, which helps the predictor head to converge to a higher performance level when unfreezing is carried out.

For YOLO, the opposite effect can be observed. Indeed, this freezing-unfreezing technique deteriorates performance for both backbones. This can be attributed to the fact that YOLO's head takes more epochs to train from scratch compared to Faster R-CNN's head as it is composed of more layers. When unfreezing the backbone, similar loss peaks as for Faster R-CNN can be observed, which indicate that training is disturbed at some point. Faster R-CNN is able to recover in less than ten epochs and can then continue training and achieve higher performance. YOLO could unfortunately not recover in ten epochs for most freezing settings, which could also be endorsed by a too high initial learning rate after unfreezing. We can however notice that degradation is less pronounced when YOLO starts from the ResNeXt backbone, which underlines the fact that starting from a backbone already fine-tuned on paintings really helps the head when it needs to be trained

from scratch. As was already explained, removing the stopping criterion for YOLO models would very likely be beneficial, although this could not have been investigated.

Finally, a setting where the backbone remains frozen throughout training was considered for both models, and both types of backbone. For both detectors, it was observed that freezing the paintings fine-tuned backbone lead to better results than freezing the COCO backbone, which could be expected as the heads, trained from scratch, are able to take advantage of feature representations that are more specific to paintings. For Faster R-CNN, this approach leads to worse results than the freezing-unfreezing approach, which can be expected since no fine-tuning of the backbone is involved. For YOLO, we can observe that it consistently outperforms the freezing-unfreezing settings, which underlines that disturbed too much when unfreezing is performed. Indeed, freezing the ResNeXt backbone for the entire training even yields the best results across all other YOLO variations. We can thus expect that, if we let the backbone frozen for a sufficiently high number of epochs such that the head can stabilize, and that we eventually unfreeze the backbone in order to fine-tune the entire network with a small enough learning rate, then we could improve results for both backbones. Naturally, it would imply to target and to resolve the learning issues encountered when unfreezing.

Chapter 6

Conclusion

In this thesis, as part of the INSIGHT project, we have investigated the application of deep learning techniques to the field of digital humanities. In a perspective directed towards the automatic annotation of artworks to ease their digital integration, several convolutional neural network architectures have been studied to tackle the related tasks of classification and object detection. In particular, this work focused on the classification and detection of animal depictions although a collection of artworks illustrating fruits has also been annotated by project members for similar purposes.

For the classification task, we have assessed the performance of several state-of-the-art classifiers in multiple settings. The initial baseline consists of their performance as architectures pre-trained on ImageNet. There, we have been able to appraise how these architectures performed when facing such a domain transfer. It resulted that the best achieved top-1 accuracy was equal to 37.24% for the ResNeXt-101 32×8d model, clearly highlighting the gap implied by domain transfer.

From there, different approaches were explored to evaluate their impact on classification accuracy. Models trained from scratch were compared to off-the-shelf and completely fine-tuned models, the latter achieving the highest accuracy whatever the considered architecture. Once again, the best performance was reached by the ResNeXt architecture, with an accuracy of 65.01%, outperforming by almost 28% its pre-trained counterpart. In addition, we have also provided different interpretational means to gain a deeper understanding about model predictions and about its performance for each animal. For instance, we have presented several visual illustrations to characterize regions of the input images thought as important by the model to make its final prediction.

Regarding the detection task, we followed a similar path as for the classification problem. Two distinct object detection approaches were compared and their performance was first assessed using detectors pre-trained on COCO. This allowed us to get a first flavour of how well they could deal with the domain shift from photo-realistic pictures to artworks, although, unfortunately, a deep comparison with their performance on natural images when used as pre-trained detectors could not be obtained.

As a last step, we have studied the influence of several transfer learning approaches to deal with this domain transfer. For that purpose, a meta-comparison of backbone influence was provided to determine whether starting from a backbone already fine-tuned on paintings would necessarily yield better performance. Different training behaviors were obtained

depending on the considered setting but also on the considered model and backbone. From all experimented settings, the best object detector that we have derived is a Faster R-CNN model, using as backbone the fine-tuned ResNeXt classifier obtained during the classification task, for which its backbone was frozen for 5 epochs. This model achieved a final mAP @.5 equal to 21.45%. In addition, we have highlighted other experiments that could have been carried out to further assess the impact of the models' individual components and of the training setting. Along with those, some ideas and intuition to improve current results have been proposed.

All approaches explored in this work can be extended from the task of classifying/detecting animals to that of classifying/detecting fruits. Even more, it is not necessary to construct an ImageNet fruits-only dataset as we could directly fine-tune pre-trained classifiers on fruit crops, unless we want to assess per class fruit accuracy for pre-trained architectures. In addition, we could even reuse the ResNeXt classifier fine-tuned on animal crops and fine-tune it further on fruit crops. Then, we could either use this final classifier as backbone to an object detector and repeat the various experiments or simply apply the same technique with a detector that is fine-tuned on animal artworks, i.e. reuse the detector to fine-tune it further on fruit depictions.

To try to improve upon current results, we could investigate other learning approaches than strict supervised object detection where input pictures are matched with output bounding box annotations. For instance, we could explore techniques of few-shot learning applied to object detection [57, 58]. These approaches help detectors to better detect objects that have only a few annotated examples. This would potentially improve performance for the tail of less frequent animals. In [57], for example, they rely on a two-stage pipeline. In the first stage, the entire object detector is trained on objects that appear frequently. The second stage consists in freezing the feature extracting modules and only fine-tuning the predictor head on a balanced set of frequent and rare classes. Another possibility could be [59], where authors propose a novel deep learning framework to tackle the problem of zero-shot object detection by relying on textual descriptions to help the optimization of the network. An easy way to provide textual descriptions automatically (with no human involved) would be to retrieve the animal names as we retrieve their images and bounding boxes.

Appendix A

Building blocks of CNNs

To illustrate all the following concepts, an example of a convolutional neural network is provided in Figure A.1. It graphically and abstractly represents the usual stack of convolutional and pooling layers, followed by some fully-connected layers.

A.1 Convolutional layer

A convolutional layer takes as input a tensor of dimensions $C \times H \times W$ and outputs a tensor of dimensions $D \times H' \times W'$, where C and D denote the number of input and output channels, respectively, and where the other dimensions indicate the height and width of the tensor. For example, in Figure A.1, the number of output channels D after the first convolutional layer is equal to 32. Each output channel d corresponds to a particular kernel (filter) of dimensions $C \times h \times w$, which will capture specific features using spatial information thanks to the convolution operation: every component (i, j) in an output channel is calculated as the the sum of the elements of the component-wise product between its corresponding filter and a window of the same size as the filter centered at $(\cdot, i, j)$ in the input tensor. This window is called the receptive field of element (i, j), and the effective receptive field is defined as the receptive field with respect to some input tensor, e.g. with respect to the input image.

A stride is also defined, to reduce the spatial dimensions of the input tensor, which is called downsampling, by moving the filter multiple elements at a time (horizontally and vertically) instead of one element at a time.

Padding can be added to the input tensor to deal with spatial downsampling in the output tensor and it consists in adding dummy bordering elements to the input. Without padding, the output tensor will inevitably shrink if the spatial dimensions of the filter are greater than 1.

A.2 Pooling layers

The idea of a pooling layer is to reduce tensor dimensions while keeping as much information as possible, to decrease the amount of parameters needed as depth increases. The number of channels remains unimpaired. For this purpose, pooling layers apply a certain operator (e.g. the max operator) by sliding a $n \times m$ pooling window, with a stride of

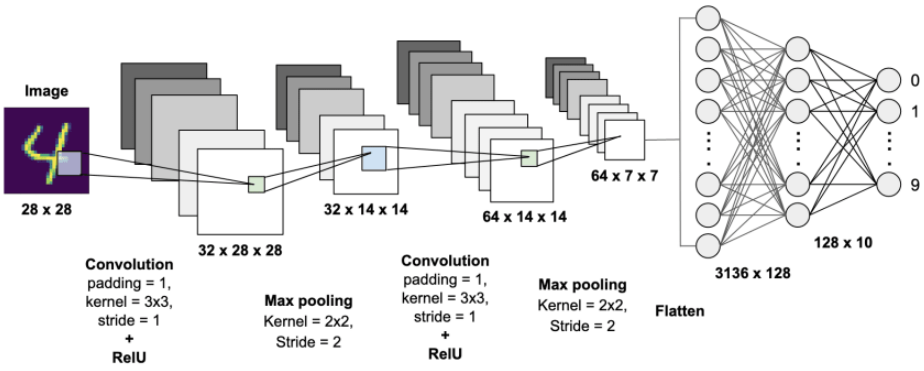


Figure A.1: Example of a CNN architecture for digit classification. (Source: [60])

(n, m). This means that, for an input tensor of dimensions $H \times W$, the output will be a tensor of dimensions $\frac{H}{n} \times \frac{W}{m}$ where each element corresponds to the value resulting from the corresponding $n \times m$ window in the input tensor on which the operator is applied. This principle is illustrated in Figure A.1 between the first two convolutional layers, where the input tensor is downscaled by a factor of 2.

A.3 Activation functions

Activation functions are mostly non-linear functions applied to increase the representation capacity and complexity of successive layers. It will eventually help the decision function to be more discriminative, however less interpretable.

Most widely used activation functions are namely the sigmoid function and the hyperbolic tangent function, defined as:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$
$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

which will output a value in $[0, 1]$ or $[-1, 1]$ respectively. However, such functions have a tendency to saturate (to a high or low response) for most values of their domain, and are firmly responsive only around 0-inputs. In addition, they also lead to vanishing gradient issues as network depth is increased, since the maximum derivative value that can be obtained from a sigmoid function when backpropagating is 0.25.

ReLU [15], which stands for rectified linear unit, accounts for these issues and is defined as follows:

$$\text{ReLU}(z) = \max(0, z)$$

A.4 Softmax layer

In general, a softmax layer applied to an input tensor $t \in \mathbb{R}^{C \times H \times W}$ is defined by the following function:

$$p_{chw} = \frac{\exp(t_{chw})}{\sum_{i=1}^{C} \exp(t_{ihw})}$$

with c, h, w indices respectively $\leq C$, $\leq H$ and $\leq W$, where C denotes the number of channels of the tensor while H and W correspond to its spatial dimensions. Thus, such a layer is applied to every element in a *channel-wise* manner. This function has the nice property that

$$\sum_{c=1}^{C} p_{chw} = 1$$

giving similar behavior as with probabilities. For example, one could imagine a softmax layer applied after the final fully-connected layer in Figure A.1. There, the input tensor would have $H = W = 1$ and $C = 10$.

A.5 Batch normalization

A batch normalization layer [24] consists most of the time of an added layer between a convolutional layer and an activation layer, which shifts and rescales the current mini-batch according to the mean and variance estimated during training. It yields better performance and higher stability, often permitting to achieve similar or finer performance with fewer training steps.

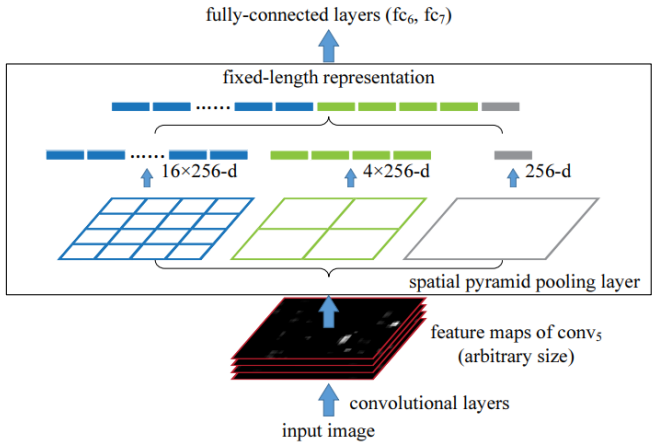


Figure A.2: Example of an SPP operation with three layers. (Source: [61])

A.6 Spatial pyramid pooling

Spatial pyramid pooling [61] alleviates the issue of using fixed-size images in CNNs. Indeed, for most CNN architectures that perform classification or detection, fully-connected layers are required in order to perform the final classification (for example). However, such layers require a fixed-size input vector, thereby implying that the last convolutional layer outputs a fixed-size vector. Since the number of feature maps is fixed beforehand and is independent of the image dimensions, the solution is to impose fixed dimensions to the input image, which can result in a loss of information.

With spatial pyramid pooling (SPP), flattening the output feature map of the last convolutional layer is no longer needed, as SPP breaks the latter down into a fixed-length 1-dimensional vector. In general, this is carried out by concatenating outputs of max-pooling layers applied with varying scales, as is illustrated in Figure A.2.

Bibliography

[1] "digitization". In: *Oxford advanced learner's dictionary of current English Online*. Oxford University Press. URL: https://www.oxfordlearnersdictionaries.com/definition/english/digitization (visited on 04/16/2021) (page 3).

[2] Matt Tarpey. "A Brief History of Digitization". URL: https://www.exelatech.com/blog/brief-history-digitization (visited on 04/16/2021) (page 3).

[3] European Commission. "Digital cultural heritage". URL: https://digital-strategy.ec.europa.eu/en/policies/cultural-heritage (visited on 04/17/2021) (page 3).

[4] Europeana. "About us". URL: https://www.europeana.eu/en/about-us (visited on 04/17/2021) (page 3).

[5] Motion Metrics. "How Artificial Intelligence Revolutionized Computer Vision: A Brief History". URL: https://www.motionmetrics.com/how-artificial-intelligence-revolutionized-computer-vision-a-brief-history (visited on 04/17/2021) (page 3).

[6] Frank Rosenblatt. "Principles of neurodynamics. perceptrons and the theory of brain mechanisms". Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY, 1961 (page 3).

[7] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. "Handwritten digit recognition with a back-propagation network". In: *Advances in neural information processing systems*. 1990, pp. 396–404 (page 3).

[8] Marc Van Droogenbroeck. "Computer Vision". 2015. URL: http://hdl.handle.net/2268/184667 (page 4).

[9] Wikipedia. "Digital humanities". URL: https://en.wikipedia.org/wiki/Digital_humanities (visited on 04/17/2021) (page 4).

[10] Insight. "About - Insight". URL: https://hosting.uantwerpen.be/insight/index.php/about/ (page 4).

[11] Matthia Sabatelli, Nikolay Banar, Marie Cocriamont, Eva Coudyzer, Karine Lasaracina, Walter Daelemans, Pierre Geurts, and Mike Kestemont. "Advances in Digital Music Iconography: Benchmarking the detection of musical instruments in unrestricted, non-photorealistic images from the artistic domain". In: *Digital Humanities Quarterly* 15.1 (2021) (pages 5, 21–24, 29, 40, 54, 60).

[12] Nello Cristianini, John Shawe-Taylor, et al. "An introduction to support vector machines and other kernel-based learning methods". Cambridge university press, 2000 (page 6).

[13] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. "Object recognition with gradient-based learning". In: *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345 (page 6).

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y (pages 7, 15).

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105 (pages 7, 8, 90).

[16] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014) (pages 7–9).

[17] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9 (pages 7, 11).

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (pages 7, 9, 10, 12).

[19] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. "Aggregated residual transformations for deep neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500 (pages 7, 11, 12).

[20] Max Ferguson, Ronay Ak, Yung-Tsun Tina Lee, and Kincho H Law. "Automatic localization of casting defects with convolutional neural networks". In: *2017 IEEE international conference on big data (big data)*. IEEE. 2017, pp. 1726–1735 (page 8).

[21] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833 (page 8).

[22] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48 (page 9).

[23] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256 (page 9).

[24] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456 (pages 9, 91).

[25] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. "Grad-cam: Visual explanations from deep networks via gradient-based localization". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626 (page 13).

[26] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338 (page 14).

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755 (page 14).

[28] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587 (pages 15, 16).

[29] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448 (pages 15, 16).

[30] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *arXiv preprint arXiv:1506.01497* (2015) (pages 15, 17).

[31] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788 (pages 15, 18).

[32] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. "Selective search for object recognition". In: *International journal of computer vision* 104.2 (2013), pp. 154–171 (page 15).

[33] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, Ayush Chaurasia, TaoXie, Liu Changyu, Abhiram V, Laughing, tkianai, yxNONG, Adam Hogan, lorenzomammana, AlexWang1900, Jan Hajek, Laurentiu Diaconu, Marc, Yonghye Kwon, oleg, wanghaoyang0106, Yann Defretin, Aditya Lohia, ml5ah, Ben Milanko, Benjamin Fineran, Daniel Khromov, Ding Yiwei, Doug, Durgesh, and Francisco Ingham. "ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations". Version v5.0. Apr. 2021. DOI: 10.5281/zenodo.4679653. URL: https://doi.org/10.5281/zenodo.4679653 (page 19).

[34] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018) (page 19).

[35] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359 (page 20).

[36] Matthia Sabatelli, Mike Kestemont, Walter Daelemans, and Pierre Geurts. "Deep transfer learning for art classification problems". In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2018, pp. 0–0 (pages 21, 22).

[37] Wei Ren Tan, Chee Seng Chan, Hernán E Aguirre, and Kiyoshi Tanaka. "Ceci n'est pas une pipe: A deep convolutional network for fine-art paintings classification". In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 2016, pp. 3703–3707 (page 21).

[38] Eva Cetinic, Tomislav Lipic, and Sonja Grgic. "Fine-tuning convolutional neural networks for fine art classification". In: *Expert Systems with Applications* 114 (2018), pp. 107–118 (page 21).

[39] Ahmed Elgammal, Bingchen Liu, Diana Kim, Mohamed Elhoseiny, and Marian Mazzone. "The shape of art history in the eyes of the machine". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018 (page 21).

[40] Elliot J Crowley and Andrew Zisserman. "The art of detection". In: *European conference on computer vision*. Springer. 2016, pp. 721–737 (page 21).

[41] Adrian Lecoutre, Benjamin Negrevergne, and Florian Yger. "Recognizing art style automatically in painting with deep learning". In: *Asian conference on machine learning*. PMLR. 2017, pp. 327–342 (page 21).

[42] Sergey Karayev, Matthew Trentacoste, Helen Han, Aseem Agarwala, Trevor Darrell, Aaron Hertzmann, and Holger Winnemoeller. "Recognizing Image Style". In: *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014. DOI: http://dx.doi.org/10.5244/C.28.122 (page 21).

[43] Stanislav Smirnov and Alma Eguizabal. "Deep learning for object detection in fine-art paintings". In: *2018 Metrology for Archaeology and Cultural Heritage (MetroArchaeo)*. IEEE. 2018, pp. 45–49 (page 21).

[44] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. "A neural algorithm of artistic style". In: *arXiv preprint arXiv:1508.06576* (2015) (page 21).

[45] Thomas Mensink and Jan Van Gemert. "The rijksmuseum challenge: Museum-centered visual recognition". In: *Proceedings of International Conference on Multimedia Retrieval*. 2014, pp. 451–454 (page 22).

[46] Yihui He. "Object Detection with YOLO on Artwork Dataset". In: *Advanced Computer Vision at Jiaotong University* (2016) (page 22).

[47] Shiry Ginosar, Daniel Haas, Timothy Brown, and Jitendra Malik. "Detecting people in cubist art". In: *European Conference on Computer Vision*. Springer. 2014, pp. 101–116 (page 22).

[48] Nicholas Westlake, Hongping Cai, and Peter Hall. "Detecting people in artwork with cnns". In: *European Conference on Computer Vision*. Springer. 2016, pp. 825–841 (page 22).

[49] Nicolas Gonthier, Yann Gousseau, Said Ladjal, and Olivier Bonfait. "Weakly supervised object detection in artworks". In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2018, pp. 0–0 (page 22).

[50] Raphael Maree, Loic Rollus, Benjamin Stevens, Renaud Hoyoux, Gilles Louppe, Remy Vandaele, Jean-Michel Begon, Philipp Kainz, Pierre Geurts, and Louis Wehenkel. "Collaborative analysis of multi-gigapixel imaging data using Cytomine". In: *Bioinformatics* 32.9 (2016), pp. 1395–1401 (page 23).

[51] Alex Conway. "Deep Learning for Computer Vision". Oct. 2017. URL: https://speakerdeck.com/pyconza/deep-learning-for-computer-vision-by-alex-conway (page 31).

[52] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255 (page 31).

[53] Christiane Fellbaum. "WordNet: An Electronic Lexical Database". Bradford Books, 1998 (page 31).

[54] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I Jordan. "How does learning rate decay help modern neural networks?" In: *arXiv preprint arXiv:1908.01878* (2019) (page 36).

[55] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2014 (page 64).

[56] Stefan Hinterstoisser, Vincent Lepetit, Paul Wohlhart, and Kurt Konolige. "On pre-trained image features and synthetic images for deep learning". In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2018, pp. 0–0 (pages 64, 68–70).

[57] Xin Wang, Thomas E Huang, Trevor Darrell, Joseph E Gonzalez, and Fisher Yu. "Frustratingly simple few-shot object detection". In: *arXiv preprint arXiv:2003.06957* (2020) (page 88).

[58] Gongjie Zhang, Zhipeng Luo, Kaiwen Cui, and Shijian Lu. "Meta-DETR: Few-Shot Object Detection via Unified Image-Level Meta-Learning". In: *arXiv preprint arXiv:2103.11731* (2021) (page 88).

[59] Licheng Zhang, Xianzhi Wang, Lina Yao, and Feng Zheng. "Zero-shot object detection with textual descriptions using convolutional neural networks". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–6 (page 88).

[60] Afroz Chakure. "Convolutional Neural Networks (CNN) in a Brief". URL: https://dev.to/afrozchakure/cnn-in-a-brief-27gg (visited on 06/05/2021) (page 90).

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Spatial pyramid pooling in deep convolutional networks for visual recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 37.9 (2015), pp. 1904–1916 (page 91).