UCLouvain

École polytechnique de Louvain

Multiple Flying Bird and Bird Keypoint Detection Toolbox for processing bird videos.

Author: **Adrien Delhez**
Supervisor: **Renaud Ronsse**
Readers: **Gianmarco Ducci, Christophe De Vleeschouwer**
Academic year 2021–2022
Master [120] in Mathematical Engineering

Abstract

Multiple Bird detection is challenging due to their diversity, their sizes and their overlaps in flock, but is essential to derive information on birds in many application fields. A multiple bird detection toolbox is proposed with several deep learning models with different accuracies and speeds, trained on a bird dataset. A new dataset, constructed by adding non-bird images, is investigated to increase the robustness of the models on non-bird objects. In addition to that, an algorithm to improve the bird detection on large images, is tested. Finally, a very simplified keypoint detection method is explained. The detection results show that some deep learning models have very good performance and are able to detect slightly overlapping birds. Some models obtain good precision while being fast which is a real advantage on large images. Adding the non-bird dataset reduces false detections and gives promising detection results.

Chapter 1

Introduction

Flying Bird detection is very useful in many applications. In the papers [26] (2014) and [25] (2017), it is used to increase the flying aircraft security in low-altitude airspace. The bird-aircraft collisions can be avoided by alerting in time the air traffic controllers, who will take decisions accordingly. In the paper [2] (2020), a flying bird detection method is used to get information on bird population movement trends and assist traditional bird counting techniques, often made by keen observers. Tracking the bird population is important since it is a direct indicator of environmental health and biodiversity, due to the sensitivity of the birds to environmental changes [11]. Depending on the application field, the detection method has to meet certain requirements in accuracy and speed. For example in aviation safety, the controllers must have the warning as fast as possible with high relevancy (few false detections). Moreover, the detection of flying birds can be challenging due to their variety in colors, shapes, flight poses and sizes. It is the case for small birds, especially on large images. Photos or videos of flying birds taken from afar have a very large resolution. For example, large images used in this thesis have a 3456×6144 resolution. In addition to that, birds flying in a flock tend to overlap with each other, leading to occlusions, and are difficult to distinguish.

In this thesis, a multiple flying bird detection toolbox for processing bird videos is developed using deep learning. The link between video and image is very big since a video is a sequence of images. The bird detection is made on every image independently and so, the thesis focuses on images. The aim of the toolbox is to provide and investigate several methods with different purposes while dealing with slightly overlapping birds and small birds on large images. Moreover, a very simplified method [26] is proposed in Chapter 5 to extract the key points (head, tail and wings) of the birds once the detection has been made. One problem in deep learning techniques is the lack of data. In fact they require a lot of data to train them. To resolve this problem, transfer learning allows to reuse a model

already trained on very large dataset and train it again for a specific purpose but with a dataset of limited size. One can obtained acceptable performance with around 5000 data when using transfer learning [13, p. 20]. In my case, the dataset contains less than 1000 images but one can increase considerably this number using data augmentation. It is assumed that the birds are flying in an uniform sky and not above the ground since it is the case for most of the birds in the dataset. Nevertheless to increase the robustness of the models on non-bird objects and the ground, a second dataset is constructed by adding an additional non-bird dataset to the initial bird dataset. Comparisons of the models trained with and without it, are made. The Chapter 3 analyses the datasets and explains the data augmentation performed. After that, the Chapter 4 explains the deep learning methods. Faster R-CNN network [12, 22] is used with three different backbones which are the part of the network extracting the relevant features of the image. These backbone architectures bring different accuracies and speeds. This is the basis of any Convolutional Neural Network (CNN) [5] model which is explained at the beginning of the chapter. After the CNN review which gives the key components of the Faster R-CNN models, these are explained in details. The outputs (bounding boxes) given by the models are then filtered by a NMS algorithm [7, 17] to improve the bird detection. Moreover, an algorithm, called Slicing Aided Hyper Inference (SAHI) [3, 4], is used to improve the detection for small birds and for large images. The Section 4.3 explains how the models are trained and validated. Only the performances with the bird datatset is presented. Finally the detection results are shown in Section 4.4. The models are tested on images with the same resolution as the training dataset and on very large images with/without SAHI. The visual comparison between the models trained with/without the non-bird dataset is also shown there.

Chapter 2

Literature review

The detection of multiple flying birds on videos can be challenging due to the variety of bird types and the many possible flight positions. In particular when there are a lot of birds flying in different directions with possible slight overlaps between them. Qunyu Xu and Xiaofeng Shi [26] (2014) propose to use the distinguishable shape of the birds by extracting a simplified bird skeleton. To do so, they perform image segmentation to obtain the moving objects of interest using two consecutive frames. After that, they find the four extreme points of the contour of the object which should correspond to the four key points (head, tail and two wings) of a bird. The two frames are used to have the flying direction and associate properly the head and the tail to their key point. From these points, a feature vector is created to describe the bird and is used by a linear SVM to classify the object as a bird or not. The SVM classifier is trained with an annotated flying bird dataset where the birds fly in side-view, but also a non-flying bird dataset. This method gives efficient results for its simplicity but is limited to very specific flying poses in side-view. Tianhang Wu, Xiaoyan Luo and Qunyu Xu [25] (2017) solve this problem by introducing multiple pose-specific classifiers. A feature vector is extracted the same way as above with hand-crafted rules to manage the key points occlusions. For each frame of two consecutive frames, objects of interest are generated and classified based on their feature vectors (flying poses) using these classifiers. After that, the two classification results are combined based on a pose-change pair probability distribution to have the final detection results. They use the fact that the bird poses between two consecutive frames are related and are linked to certain types of movement. This method succeeds to handle more different poses while being robust and efficient. But the pose problem still remains since a flying bird has many possible flying directions which further increase the number of possible poses. Moreover for both papers, the methods were only tested on videos containing only one flying bird. If there are several birds that are close to each other while flying in different directions and slightly overlapping, there could

be association (tracking) problems over two consecutive frames. In addition to this, the detection is made with a skeleton-based feature vector. Even if the skeleton is quite discriminative for birds, there are many other features, maybe not obvious, that could be useful to identify the birds. In recent years, deep learning methods have been widely used and have achieved better results for object detection. The features extracted by these methods are semantically relevant and often abstract for a human.

Hüseyin Gökhan Akçay et al. [2] (2020) use deep learning to automatically count birds on images. The deep learning method is a Faster R-CNN network with Inception-v2 as feature extraction network. This fully convolutional network extracts feature maps of the image using convolutions. After that, a Region Proposal Network (RPN) generates regions of interest (ROIs) on the last feature map. For each ROI, a fast R-CNN detector (fully connected layers) decides if the ROI contains a bird or not. The outputs of the model are the confidence scores and the bounding boxes for all objects classified as birds. The Faster R-CNN has a good detection accuracy but which comes with a bad detection speed. However, it has difficulties to recognize completely the birds when they are close and when they are occlusions with other birds or other objects. It's a little less of a problem for bird counting since detecting a small part of the bird is sufficient to count the bird. Moreover, it struggles to detect very small birds and make false detections on non-bird objects. The model only uses the last feature map to perform. Even if this one is rich in semantic, its resolution is low due to pooling operations. Therefore the details of very small birds may have disappeared in the last feature map.

In this thesis, a Faster R-CNN is used with different backbones, ResNet50 [15], MobileNetv3-Large and MobileNetv3-Large 320 [16] which have different accuracies and speeds. The Pytorch ecosystem is used for the models, which is easy to use and well documented. The first one is 5x faster on CPU than the second and 25x faster than the last one, but with a decrease in accuracy [20]. This allows to analyse the trade-off between accuracy and speed. The backbones are used with a Feature Pyramid Network (FPN) [18, 23] which allows to use a pyramid of semantically rich feature maps with different resolution for detection and not just the last feature map. Therefore the model should be able to better detect small birds from the high-resolution feature map. The predicted bounding boxes can still overlap and be redundant. Then, a box filter algorithm, called non-maximum suppression, is used to keep only the relevant boxes. In addition to this, a Slicing Aided Hyper Inference (SAHI) algorithm is used to divide the images into overlapping slices and perform bird detection on each of them. This increases the pixel areas of the small birds compared to the image. Finally, to deal with false detections, a non-bird dataset is

added to try to increase the robustness of the Faster R-CNN models. Regarding the detection of key points, the same skeleton extraction method discussed above is used on every detected bird if its size allows it. There exists deep learning methods to perform key points detection but require labeled data. The current datasets available are mainly about humans, cars and some specific animals.

Chapter 3

Datasets

The dataset is based on a collection of bird photographs from [2] where more details can be found on the observation locations or the camera properties for examples. The photos were made at different places and environmental conditions with different qualities. It contains birds of several species with different flying poses and directions making the dataset very diversified and complex. Most of the images (around 82%) contain flying birds while the rest contains non-flying birds. Every bird in the images is labeled with a bounding box that surrounds it. A sample of the dataset with ground-truth labels is shown in Figure 3.1.
The label files for some images were missing and some birds were not properly labeled or sometimes didn't have a bounding box at all. It was especially the case for some overlapping birds where the bounding box covered only a part of the bird. Therefore I reorganized the dataset to make sure that every image is associated to the correct label files and added or changed the bounding boxes by hand. The dataset, called bird dataset, consists of 656 images with 4982 birds. It can be found on the following Github repository [10].

The dataset is separated into two, a training set used to train the models and a validation set used to validate these trained models. A common 80/20 split is used meaning that 80% of the dataset constitutes the training set and 20% the validation set. It corresponds to 630 and 157 images respectively. The images are picked randomly to ensure that the validation set is representative of the training set. For example, that the images with birds of different sizes or different species are contained in both sets. In general, we assume that the unseen images on which the models will be used, are similar to the ones used for training and this is why it is important to have a diversified dataset. The resolution of the images is shown on the left image in Figure 3.2. Most of the images (633) have a resolution of 1024×600. Images of a resolution of 576×576 is extracted for training and testing. A interesting feature to consider is the size of the birds given by the bounding box

Figure 3.1: Sample of the bird dataset with the ground-truth bounding boxes in red. The entire dataset can be found on the following Github repository [10]

compared to the size of the image used for training (576×576). The figure 3.3 shows the ratios (%) of the bounding box area of each bird in each image to the image area. The minimal ratio is 0.07%. The histogram contains 4452 ratios which means that almost $89,4\%$ of the birds have a ratio between 0.07% and 10%. The mean on the histogram is 4.33% but it can be noticed that many birds have a ratio around 0.25%. To give you a visual idea of the ratios, let's look at the first two images of the first row of Figure 3.1. The first one has a mean ratio of 0.25% with a minimum and a maximum of 0.14% and 0.46% respectively. The second one has a mean ratio of 3.21% with a minimum and a maximum of 1.63% and 4.12% respectively.

Finally, many videos of flying birds of different bird species were collected by the professional photographer Xavi Bou [24] and can be found on the Github repository ©. These high resolution videos are used to test the trained final models. They also contains birds of varying size which will allow to see the limitations of the models according to the size of the bird.

3.1 Addition of a non-bird dataset

Even if the landscape background change through the dataset, it is mostly uniform for each image especially for flying bird scenes. For example, the images with a blue sky background. The detection model could learn to do background subtraction

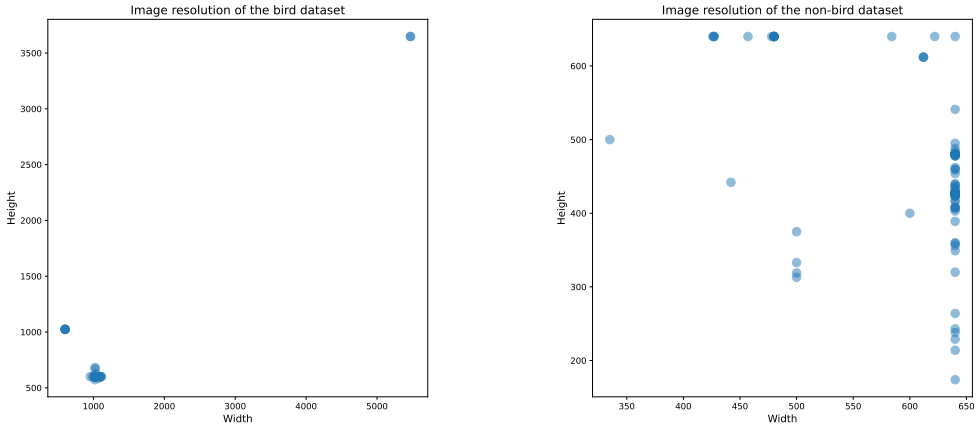


Figure 3.2: Plot of the resolutions of the bird dataset on the left and the non-bird dataset on the right. For the bird dataset, most of the images have a resolution of 1024×600. For the non-bird dataset, almost all images have at least one side of 640 pixels.

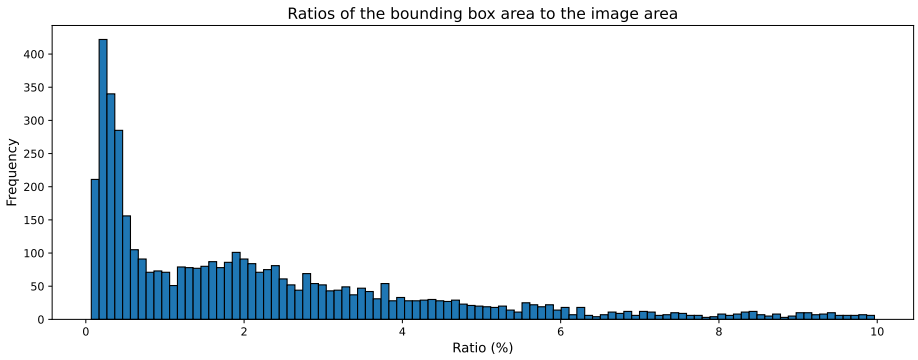


Figure 3.3: Histogram of the ratios (%) of the bounding box area of every bird to the image area (576×576). The histogram represents $89,4\%$ of the birds with ratios between 0.07% and 10%. $35,6\%$ of the birds are contained between 0.07% and 1%.

without learning to recognize the birds. This could lead to false detections on non-bird objects. Therefore to try to increase the robustness of the models, I add a non-bird dataset of 131 non-bird images without bounding boxes corresponding to 20% of the bird dataset. One third of them are flying planes, one third are non-flying planes and the rest of them are different type of images. This creates a second dataset on which the models are trained and compared with the ones trained on the bird dataset. A few samples are shown in Figure 3.4. The plane images

come from the airplane class of the COCO dataset and the others are arbitrarily picked from other classes [19]. The resolution of the images of this non-bird dataset is shown on the right image in Figure 3.2. Most of the images have 640 pixels maximum in height or in width. If the image is too small, padding is applied to obtain the required resolution of 576×576.

Figure 3.4: Sample of the non-bird dataset obtained from the COCO dataset [19] and which can be found on the same Github repository [10].

3.2 Data Augmentation

I use the Albumentations Python library [9] which is fast, work with the Pytorch ecosystem and, most importantly, can simultaneously augment an image and its corresponding bounding boxes. The transformations applied on the images are in order:

- Flip the image either horizontally, vertically or both horizontally and vertically with probability 0.5.
- Randomly changes the brightness, contrast, and saturation by a factor chosen uniformly from $[0.8, 1.2]$.
- Crop a random part of resolution 576×576 of the image.

The two first transformations are not applied when testing the models. The figure 3.5 shows some results of these transformations. For the non-bird dataset, I add padding (black) to obtain the required 576×576 resolution.
As said previously, this library is able to augment the bounding boxes. When cropping the image, the size of the bounding boxes could change and could make some birds unrecognizable. The visibility of a bounding box is the ratio of its area after augmentation to its original area. If the visibility is less than 0.6, I consider that there are not enough information to recognize the bird and the bounding box is removed. It can be observed in the first image of Figure 3.5.

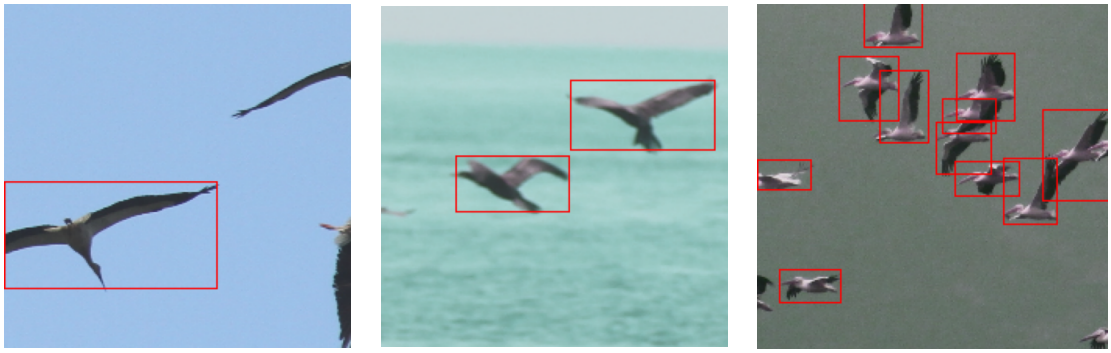


Figure 3.5: Images from Figure 3.1 on which data augmentation is applied. One can remark that some birds lose their bounding boxes since their visibility is too low i.e. below the threshold 0.6.

Chapter 4

Multiple Bird Detection

4.1 Faster R-CNN Approach

Faster R-CNN is a two-stage detection method that uses an additional module, namely Region Proposal Network (RPN) to bring attention to some regions of interest (ROI) and generate bounding boxes based on the features extracted by a fully convolutional backbone with a Feature Pyramid Network (FPN). After that the proposed bounding boxes will be classified by a Fast R-CNN detector which decides if the ROIs contain a bird or not. An illustration of the architecture of the Faster R-CNN with the details discussed in the next paragraphs is shown in Figure 4.1.

Convolutional Neural Network Architecture

A convolutional neural network (CNN) is composed of different layers:

- A convolutional layer which is the most important part of the network since it extracts automatically the relevant features of the input image. It performs multiple convolutions with different convolutional filters, called kernels. At the beginning the kernel values, called kernel weights, are initialized randomly. Then the weights are tuned by training the network to perform a specific task. Most of the recent CNNs have a deep architecture which requires a large image dataset to adjust these weights. When using transfer learning, the weights are initialized by the ones trained on a very large dataset. Of course the networks have to be the same for weight correspondence. After that one can fine-tune these pretrained weights for our task.

  The image input format of a convolutional layer is (Height, Width, Number of Channel). For the first input of the network which is the image itself, the

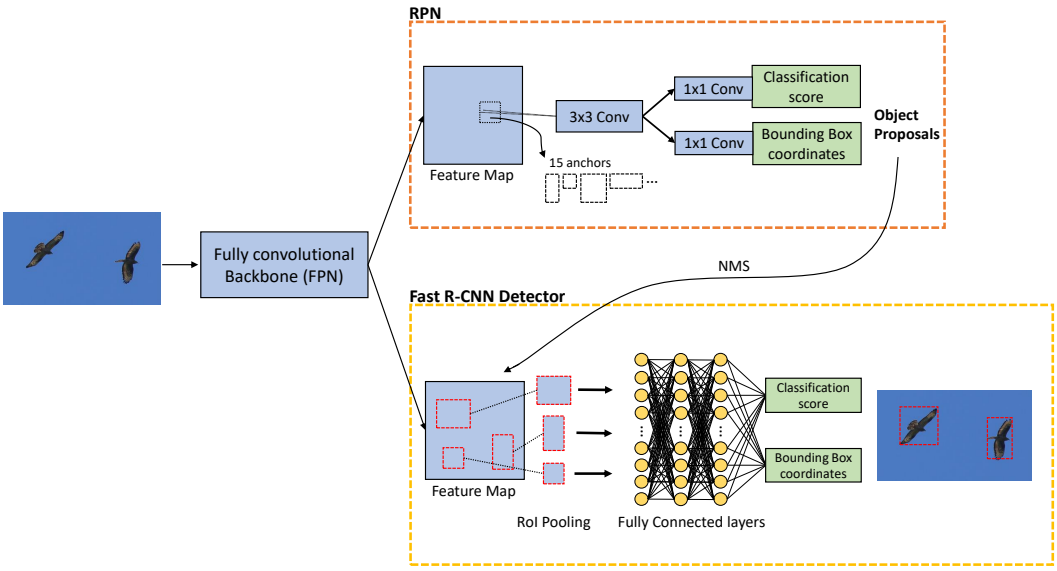


Figure 4.1: Faster R-CNN architecture. The input is a image of birds. Firstly, feature maps of different scales are extracted from the image by a fully convolutional backbone used with Feature Pyramid Network to increase their semantic meaning especially for the high-resolution feature maps. Secondly, a Region Proposal Network (RPN) generates regions of interest (ROIs) on them which are filtered by the NMS algorithm to reduce redundancy. The ROIs are associated to the proper feature map. After that a ROI pooling layer is applied to extract the ROIs and fully connected layers decide for each of them if they contain a bird or not.

format is (m, n, 3) since it is a $m \times n$ colored image with RGB channels. The set of kernels have a size of (k, k, Number of Feature Channels) where k must be lower that the resolution of the input and is often small around 2 or 3. The number of feature channels is the number of filters and also the number of channels of the layer output. The output, often called feature map, has more and more semantic meaning going through the convolution layers. Moreover, the resolution of the feature maps decreases with the depth of the network while the number of channels increases. The CNN takes full advantage of the rectangular grid structure of the images since the kernels go through every pixel as a sliding window. The convolution operation is illustrated with a small example in Figure 4.3. The kernel slides over the feature map and merges its values with the ones of the corresponding feature map section. It is done by a element-wise multiplication followed by a sum to obtain one value.

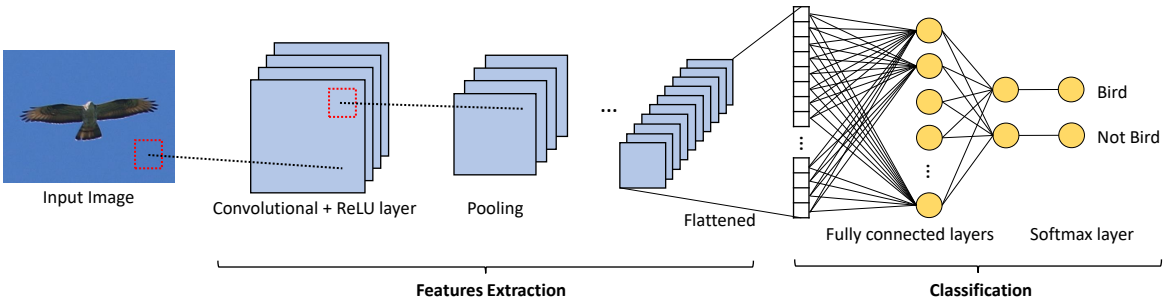


Figure 4.2: Simple example of a Convolutional Neural Network Architecture. The input is a bird image on which feature maps are extracted. This is done by applying a convolution with ReLU layer followed by a pooling layer. These three steps are applied several times until a semantically strong low-resolution feature map is obtained. This one is then flatten to feed fully connected layers followed by a softmax for the final classification. Note that it shows the elementary layers used in the fully convolutional backbone in Figure 4.1.

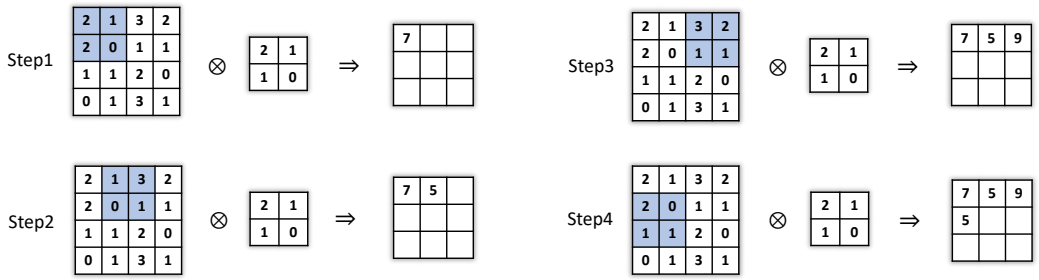


Figure 4.3: Convolution example with a 2×2 kernel on one channel of a 4×4 feature map. The numbers are only meaningful for the example. Only the four first steps are shown. The kernel slides through the feature map and perform an element-wise multiplication followed by a sum, with the corresponding section of the feature map. The number of pixel shifts by the kernel, called the stride, is 1.

- A pooling layer which resumes the most important local features and reduces the large resolution of the feature maps. It then reduces the number of parameters and improves the training. Every local 2×2 non-overlapping square of the feature map is down sampled to a certain value. There are different types of pooling operations. The two most popular and used ones are average pooling and maximum pooling. They are illustrated on a simple example in Figure 4.4. The average pooling keeps the average value of every 2×2 square while the max pooling keeps the maximum value.

- A fully connected layer (FC layer) which is the final classification layer located

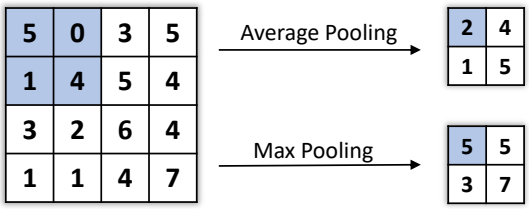


Figure 4.4: Example of the average and max pooling operation with a 2×2 window on 4×4 feature map. The numbers are only meaningful for the example. The non-overlapping 2×2 window slides through the feature map. For every window, the average number and the maximum is taken for the average and max pooling respectively.

at the end of the architecture. The input is a feature map from the last convolutional or pooling layer which is flattened to form a vector. There are multiple fully connected layers where all outputs are connected to each input of the previous layer. The goal of this layer is to make linear combinations of the input. Due to the activation function bringing non-linearities, the FC layer makes non-linear combinations of the features. It also has to be trained to learn how to make these. The final outputs is generated using a softmax activation function which gives the final results.

- An activation function layer which comes after the layers with learnable weights i.e. the convolutional layer and the fully connected layer. There are different types of activation function. The two most used are the Rectified Linear Unit (ReLU) and the Softmax activation function. The first one allows to bring non-linearities in the network by deciding to keep or not a specific value of the input. It is defined as follows:

$$g(x) = \max(0, x) \tag{4.1}$$

where g refers to the ReLU activation function and x is one of the value of the input. There are other similar functions for the same purpose, for example sigmoid or tanh, but ReLU seems to work best.
The last one is the final layer of the network coming after the fully connected layer. It transforms the outputs of this one to usable probabilities. It is then possible to know how likely an image is a bird or not. They are defined as follows:

$$p_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}, \tag{4.2}$$

where x_i is the ith-component of the final output vector of the fully connected

layer, N is the size of this vector and M is the number of outputs (classes) which is 2 for the example in Figure 4.2.

An example gathering these layers is shown in Figure 4.2. The Faster R-CNN method is more complex than that but its architecture is based on these layers. There are other layers used to obtain good training and performance as for example dropout layer or normalization layer [5].

Faster R-CNN Backbone

Three different backbones are used in Faster R-CNN, namely ResNet50-FPN, MobileNetV3-Large FPN and MobileNetV3-Large 320 FPN tuned for mobile use-cases. For simplicity, they are called ResNet50, MobileNetv3 and MobileNetv3 320. FPN means Feature Pyramid Network and will be explained at the end of the section once we have a clear view of the entire Faster R-CNN model. The backbone corresponds to the features extraction part of the CNN. It is then a fully convolutional network (FCN) i.e. a CNN without fully connected layers. The Region Proposal Network and the Fast-RCNN detector share the same set of convolutional layers which allows to train the network only once. The architecture of the backbones can be found in their respective papers [15] and [16].

These three backbones give a different accuracy and speed to the Faster R-CNN model. Accuracy often comes with a sacrifice of speed. The ResNet50 network should give the best accuracy but is the slower. MobileNetV3-Large should give competitive accuracy to ResNet50 while being 5x faster on CPU. Finally, MobileNetV3-Large 320 using reduced resolution is 25x faster on CPU but by sacrificing the speed [20].

Region Proposal Network

The Region Proposal Network (RPN) is a fully convolutional network which takes as input an image of any size and gives as output a list of region proposals with their corresponding objectness score. The input image is a feature map from the FCN backbone. The network is composed of three layers. There is a 3×3 convolutional layer followed by two fully-connected layers implemented as two 1×1 convolutional layers named reg and cls layer. This network is slid over this feature map and takes as input the corresponding $n \times n$ window of the input feature map. It is the same for all the windows. For each sliding window, k region proposals are predicted based on k reference boxes called anchors. The anchor is centered at the window and has two parameters: a scale (resolution) and a ratio. The RPN used in this thesis uses 5 areas $(32^2, 64^2, 128^2, 256^2, 512^2)$ and 3 aspect ratios $(0.5, 1.0, 2.0)$

[21]. This gives $k = 5 * 3 = 15$ different anchors for each window.
The convolutional layer extracts a feature map with 256 channels from the $n \times n$ window , which is the input of the box-regression layer (reg) and the box-classification layer (cls). The reg layer has $4k$ outputs corresponding to the coordinates of k anchor boxes and the cls layer has 2k outputs corresponding to the probabilities to have a object of interest or not for each box. The Figure 4.5 shows the network applied on one sliding window.

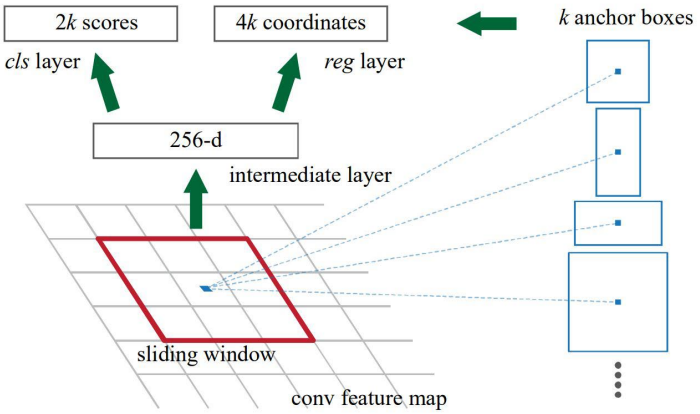


Figure 4.5: Region Proposal Network (RPN). A 256-d feature map is extracted from the sliding window. At most k region proposals with different shapes are predicted for each window (k=15). After that the feature map is passed to two convolutional layer which give the $2k$ scores and the $4k$ predicted boxes coordinates corresponding to the k anchor boxes. The illustration comes from the Faster R-CNN paper [22].

The list of RPN proposals contains boxes that high overlap with each other. The non-maximum suppression (NMS) algorithm is used on all the proposals to reduce this redundancy. The IoU threshold for NMS is set to 0.7. The Section 4.2 defines IoU and explains how the NMS works. The number of proposals is greatly reduced while keeping at least the same detection accuracy. After that the best proposal boxes according to their scores are selected to perform detection by the Fast R-CNN detector.

Fast R-CNN Detector

The Fast R-CNN detector takes as inputs a feature map and a list of regions of interest described by their coordinates. The regions of interest (ROIs) are the proposal boxes obtained by the RPN and the feature map must be the same one used as the input to the RPN. It consists of a ROI pooling layer [14] followed

by several fully connected layers ended by two separated FC layers, namely the box-regression layer and the box-classification layer. A problem is that the regions of interest have different sizes while the FC layers requires a fixed-size input. The ROI pooling layer solves that by producing a fixed-size feature map from the non-uniform ROIs. The resolution of the feature map output is $l \times l$ defined a priori and it doesn't depend on the resolution of the feature map or the size of the boxes. For every ROI, it extracts the part of the feature map corresponding to it and divides it into l equal-sized sections. After that a max pooling is applied meaning that the maximum value of every section is kept forming a fixed-size feature map. This new feature map is flatten to create a feature vector to feed the FC layers. An example of the ROI pooling layer with a 4×4 output is shown in Figure 4.6. Moreover, this process increases significantly the detection speed in general. If one

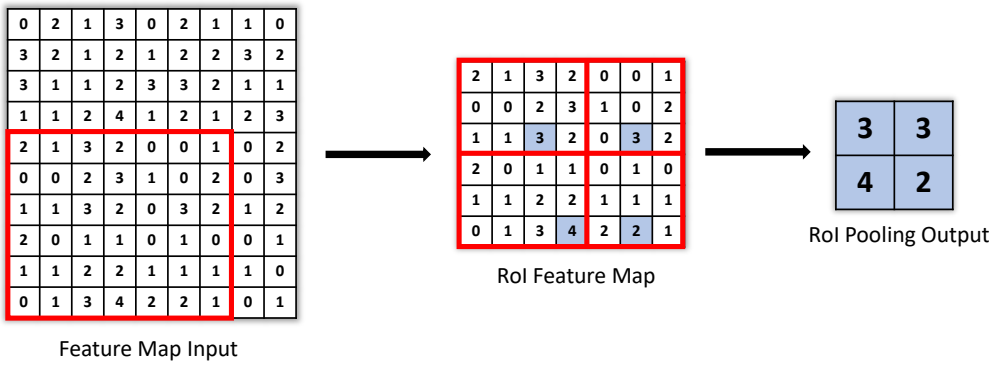


Figure 4.6: ROI pooling example on one channel of 10×9 feature map and a 6×7 ROI in red. The outputs for all ROI have 4×4 resolution. The numbers are only meaningful for the example. The section of the feature map corresponding to the ROI is extracted and divided in 4 pieces. A max pooling is applied on every section.

should use a CNN as explained in the first paragraph 4.1, one should pass every ROI of the image into it and get different feature maps for each of them (as done by a R-CNN). In this case, the same feature map is used for each proposal which allows to avoid the calculation of many features and thus save time.

Once each of the regions of interest has been processed, each extracted feature map is passed to several fully connected layers. Afterwards the last output of these is used by two FC layers in a way similar to the RPN. The box-classification layer gives the scores of having a bird or not in the boxes (a softmax layer can be used to get the probabilities) and the box-regression layer gives the coordinates of the final bounding boxes of the birds.

Feature Pyramid Network

The Feature Pyramid Network (FPN) is added to the Faster R-CNN backbone to generate multiple feature map layers with different scales while improving the relevancy of the features compared to the classic backbone. These new feature maps are going to be used for detection. The idea is to combine the strong semantic features from the low-resolution feature maps with the high-resolution ones which have weak semantic features. Therefore a feature pyramid is obtained with rich semantics at all levels [18]. The FPN backbone is composed of a bottom-up and a top-down pathway as shown in Figure 4.7. The bottom-up pathway corresponds to the fully convolutional backbone discussed previously, ResNet50, MobileNetv3 Large or MobileNetv3 Large-320. The backbone can be decomposed into blocks of convolutions. I will continue to explain with the ResNet50 backbone and its last blocks where the last outputs of them are denoted by $\{C_2, C_3, C_4, C_5\}$. The resolution of the feature maps decreases while their semantic meaning increases with the depth of the network. The top-down pathway constructs high-resolution feature maps by upsampling the last strong semantic features of the the bottom-up pathway using convolutions. Moreover the new feature maps are linked via lateral connections to their corresponding feature maps of the bottom-up pathway. The lateral connection merge them together. This avoids to lose spatial location information of the objects due to downsampling and upsampling.

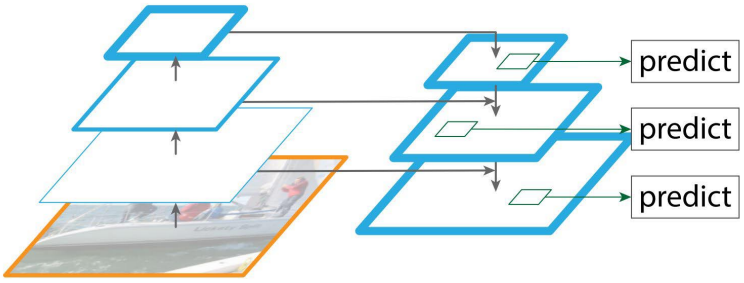


Figure 4.7: Feature Pyramid Network Diagram. The top-down (right) is connected via lateral connections to the bottom-up pathway (left). The blue outlines indicate the feature map and thicker outlines denote semantically stronger features. The last strong semantic feature map is upsampled and combined with the feature maps of the bottom-up pathway to create a feature pyramid (right) with rich semantics at all levels. The illustration comes from the Feature Pyramid Network paper [18].

To do so, a first 1×1 convolution is applied on the last feature map C_5 to reduce the channel depth to 256-d. It becomes the first feature map layer called P_5 for detection as shown in Figure 4.8. After that it is upsampled by a factor 2

using nearest neighbor upsampling and added to the corresponding feature map C_4 on which a 1×1 convolution is also applied. The second layer P_4 is then obtain by applying a 3×3 convolution to reduce the aliasing effect of upsampling. This process is repeated until the block C_2 included.

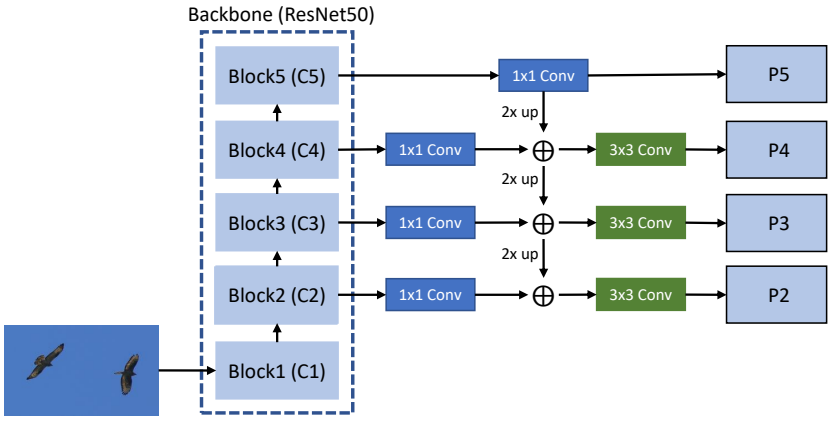


Figure 4.8: Feature Pyramid Network Architecture with a ResNet50 backbone. The backbone can be divided into five blocks of convolutions. $\{C_1, C_2, C_3, C_4, C_5\}$ correspond to the last feature map of each block and $\{P_2, P_3, P_4, P_5\}$ are the strong semantic feature map obtained by FPN. The upsampling by a factor of 2 is done using nearest neighbor upsampling (2x up).

The feature maps $\{P_2, P_3, P_4, P_5\}$ calculated by the FPN are passed to the RPN explained above. The RPN generates ROIs for each of them. Note that the RPN is the same for each scale level and so its parameters are shared. After that the ROIs must be assigned based on their size to the most appropriate feature map. The feature map P_k is chosen for a ROI of width w and height h using the following formula [18, p. 4]:

$$k = \lfloor k_0 + log_2(\sqrt{wh}/224) \rfloor, \tag{4.3}$$

where $k_0 = 4$. Once the feature map is chosen for each ROI, the Fast R-CNN detector is applied to get the final predictions.

4.2 Bounding Box Filtering

4.2.1 Non-maximum Suppression

Intersection-over-union

The intersection-over-union (IoU) measures the overlap between two bounding boxes. It is defined as the ratio of the area of overlap to the area of their union as illustrated in Figure 4.9. If the two boxes intersect perfectly then IoU equals 1. If there is no intersection, IoU equals 0. Its value is between 0 and 1. The IoU can be used as a performance metric where the two bounding boxes are a ground-truth box and a predicted box by the models. But in this section or in the Faster R-CNN approach, the two bounding boxes correspond to predicted boxes.

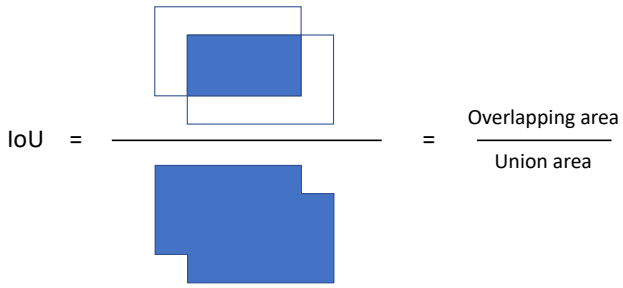


Figure 4.9: Definition of the Intersection-Over-Union. It is the ratio of the area of overlap to the area of their union. They are both highlighted in blue.

Non-maximum Suppression (NMS)

Once we have the detections given by the Faster R-CNN models, the non-maximum suppression algorithm is used to filter most of the false positives and obtain the final bird detections i.e. the final bounding boxes. NMS is based on the IoU and the confidence score associated to each detection. The inputs are the set of predicted bounding boxes $\mathcal{B}$, the list of their confidence scores and an IoU threshold N. The NMS algorithm works as follows:

1. Find the bounding box in $\mathcal{B}$ with the highest confidence score, put it in the final detection list and remove it from $\mathcal{B}$.

2. Calculate the IoU between this box and the rest of the boxes called $\mathcal{D}$. For each of these boxes, remove it from $\mathcal{B}$ if the IoU is greater than N.

3. Repeat the two first steps by finding again the bounding box with the highest score and use it to filter the other boxes which overlap too much with it, until $\mathcal{B}$ is empty.

One problem with NMS is the threshold itself, which is often the case for algorithms using threshold. Firstly, it can be difficult to find the threshold that works the best in most cases and gives the best performance. Secondly, the threshold gives strict decisions, a detection is correct or not without other possibilities. It is especially a problem with overlapping birds detection. For example, if two overlapping birds are properly detected with high confidence scores but with an IoU greater than the threshold N, the bird with the highest score is kept while the other is discarded. Therefore a bird is missed and the precision of the models decreases. A revisited version of the NMS, called Soft-NMS, has been developed to overcome this problem. It's an improvement track and is not done in this thesis.

Soft-NMS

Instead of directly remove the bounding boxes i.e. the birds with IoU values above the hard threshold, the idea is to decrease their confidence scores according to their IoU values i.e. how they overlap with the bounding box with the highest score. The confidence score decreases if the overlap increases. In this case, no detection is suppressed. The Soft-NMS algorithm works in the same way as the NMS algorithm described above but the step 2 updates the score instead of removing the box. Let $\mathcal{D}$ be the remaining boxes which changes at each iteration. The confidence score s_i of the boxe $d_i \in \mathcal{D}$ is updated using a Gaussian function as follows:

$$s_i = s_i e^{-\frac{IoU(h,d_i)^2}{\sigma}}, \tag{4.4}$$

where h is the box with the highest confidence score and σ is set to 0.5 as in the paper [7].

4.2.2 Simple Threshold Filtering

After NMS, I filter the detections which have a confidence score greater than a common hard threshold of 0.1. The birds are mostly flying in the sky but sometimes there is a background or a ground. Sometimes some objects, for example a part of a tree, are detected with very low confidence scores close to zero. The correctly detected birds have a confidence score close to 1. The threshold depends on the application of the toolbox. If we want a toolbox more robust, a threshold of 0.5 is good. If we want to count the birds for example, a threshold of 0.1 is more suitable to just eliminate the very bad results.

4.3 Training & Validation

The models are pretrained with two different big and well-known datasets. The backbone is pretrained on ImageNet and the Faster R-CNN models are pretrained on a subset of COCO train2017. From this starting point, I fine-tune the models with the bird dataset. Note that the object classifier in the Fast R-CNN detector is made for the classes of the COCO dataset. Therefore the classifier is replaced by a new one with two classes (bird and not a bird) which is trained from scratch. Due to the limited size of the dataset, I performed data augmentation to increase the amount of data available for training and validation described in Chapter 3. The results of this section are made without the non-bird dataset. A performance comparison of the dataset with and without an additional non-bird dataset is made in Subsection 4.4.1.

The Faster R-CNN network can be seen as a function that takes as inputs, images and gives as outputs, bounding boxes coordinates and bird scores. This function depends on a lot of unknown parameters, the weights of the convolutions or fully connected layers for example. A loss function is used to tell us how bad the outputs are and change these parameters accordingly. It makes the difference between the true outputs (given by the dataset) and the predicted outputs. Train the models is then learn the parameters that reduce as much as possible the loss function value. It's a optimisation problem where the difference between the prediction and the goal is minimized. The optimization algorithm used is the Stochastic Gradient Descent (SGD) [8]. It updates the parameters based on the gradient of the loss function with respect to these parameters, and a learning rate (LR) which determines the fraction of the gradient used to adjust the parameters. The LR is very important since it is responsible for the good convergence of the model. Other hyper-parameters are used as for instance, momentum and weight decay for faster convergence and regularization respectively. I tested the widely used Adam optimizer but it didn't work as well as SDG. The dataset is split into small groups of images, called batches. The network is applied on all the images of a batch before updating the weights. To be more precise, the name of the algorithm is then Batch Gradient Descent where SGD is used on batches [5]. Once the network is trained over all the batches i.e. the entire dataset, the process is repeated several times defined by the number of epochs. Moreover, a LR scheduler is used with the optimizer. It is a pre-defined schedule that reduces the learning rate after each epoch during the training and allows to improve the convergence to the minimum. Finally, the RPN and the Fast R-CNN detector can not be trained separately since they share the same convolutional layers. Therefore, a 4-step Alternating Training algorithm is adopted to train them as a unified network [22].

The models are trained on 576×576 images and a batch size of 8 to fit the GPU memory. The training is analyzed using the learning curves discussed in Subsection 4.3.1 which shows the evolution of the training and validation losses (from RPN and Fast R-CNN) during the training. For Faster R-CNN with a ResNet50 backbone, the learning rate for SGD is 0.005 found from another project [1]. I tested LRs of different orders, 0.05 and 0.0005 but they didn't improve the validation losses. For the two MobileNetv3 backbone, I found that a learning rate of 0.0005 works best. For the three backbones, a ReduceLROnPlateau scheduler reduces the learning rate when the average of the four validation losses remains unchanged over the epochs.

4.3.1 Learning curves of the models

The learning curves (LCs) allow to observe how the model performs on the unseen data from the validation set during the training. The performance of the trained models on the training and validation set is evaluated after each epoch. The performance in this subsection is given by the different loss values (4) of the Faster R-CNN models. The smaller the loss, the better. There are two loss values from the Region Proposal Network for the object classifier and the box regressor, namely RPN Objectness Loss and RPN Regression Loss respectively. There are also two ones from Fast R-CNN for the final bird classifier and the final bounding box regressor, Classifier Loss and Box Regression Loss. The epoch loss values are the averages of the values obtained for each batch. Moreover, cross-validation is used over 30 runs to estimate the expected performance of the models. After each run, the entire dataset is shuffle which changes the training and validation set. Note that there is no link between the runs, models are retrained from the beginning for each run. The LCs in Figure 4.10 show the mean and the 95% confidence interval of the loss values obtained for these runs. For each graphic, one can observe that a gap is made between the training and validation learning curves. It means that the training set isn't enough to learn properly the problem and obtain good performance on the validation set. The models don't learn any new information since the validation loss stops improving while the training loss continues to decrease. Therefore it seems that data augmentation isn't enough to solve the lack of data and that more data need to be added to the dataset. Moreover for some graphs, mainly the Box regression loss, the validation loss starts to increase after some epochs. The models overfit the training data and lose its ability to generalize to new data. But in this case, the overfitting problem is small and only concerns one loss. The Figure 4.11 compares the overall loss i.e. the mean of the four loss values, for the three backbones. If one looks at the best validation loss obtained for all epochs, the ResNet50 backbone obtains a minimum loss (0.26) almost three times larger than the MobileNetv3 minimum loss (0.74). And it is

more than four times the MobileNetv3 320 minimum loss (1.13). The final models used to perform the detection in Section 4.4 are those that give the best overall validation losses over the epochs.

4.3.2 Performance evaluation

The performance of the final trained models is evaluated on the validation set with the mean average precision (mAP) which is a widely used metric in object detection. The images of the validation set weren't used for training and are fed to the models to perform bird detection. For each image, one have the ground-truth bounding boxes, the predicted bounding boxes and the corresponding confidence scores. A detection is considered as correct (true positive) if the IoU of its predicted bounding box with a grounth-truth box is greater than a specific threshold. If there are several correct predictions for one ground-truth box, only one is considered as true positive (TP) while the rest as false positive (FP). The detections below the threshold are incorrect (FP). Once every detection has been classified as TP or FP, they are sorted by their confidence score in descending order. After that, the precision-recall curves can be calculated. The curves are constructed by calculating the precision and the recall and accumulating the detections one by one. The precision is the ratio between the number of true positives and the number of detections. The recall is the ratio between the number of true positives and the number of birds in the validation set (constant). Finally, the average precision is obtained by calculating the area under the precision-recall curve. The mAP is the average of AP of the classes of the models. It is then equivalent to the AP since there is only one relevant class, the bird. Two mAPs are calculated with an IOU threshold of 0.5 and 0.75. The results are given for each model in Table 4.1 with the CPU time in second to perform detection on one image of resolution 576×576 (mean over five runs).

The precision-recall curves for the three models in Figure 4.12 show different performances. The "true" performance of the models, the performance when NMS is applied with a IOU threshold of 0.2, and the performance when NMS is applied followed by a filter (4.2.2) with a threshold of 0.1. By looking at the mAPs, it seems that the NMS doesn't decrease the performance of the models but if one looks at the prec-rec curves, the last precision is much higher than without NMS. The recall is a bit smaller which decreases the area and so the mAP. NMS rejects many overlapping bounding boxes, which is often relevant, but if some birds overlap significantly, it will result in missed detections and reduce the recall. The filter increases the precision and keep the same recall by filtering the bounding boxes with very low confidence scores. The $mAP^{.75}$ is more strict in the way it chooses if a box is a true positive or not. One can observe that it drops greatly for the MobileNetv3 320 backbone compared to the others.

	$mAP^{.50}$	$mAP^{.50}_{NMS}$	$mAP^{.75}$	$mAP^{.75}_{NMS}$	CPU Time(s)/img
ResNet50	97.1	94.6	75.5	74.3	5.09
MobileNetv3	94.3	90.6	66.7	65.8	0.69
MobileNetv3 320	77.9	73.9	44.8	43.7	0.19

Table 4.1: Mean average precision (mAP) with two different IOU thresholds, 0.5 and 0.75 for the three models. NMS means that the NMS algorithm (with a 0.2 threshold) is applied after the detections obtained by the models. The last column shows the detection time in second for an 576×576 image.

4.3.3 Slicing Aided Hyper Inference

The Slicing Aided Hyper Inference (SAHI) has been developed for the detection of small objects, especially on very large images. The image is divided into small overlapping slices which are fed to the detection model. It allows to increase the pixel areas of small objects compared to the image. After that, the detection is performed independently on every slice. The predicted results are then merged using NMS with a threshold of 0.5 and resized for the original image.

4.4 Results

The detection results on 576×576 images in Figure 4.13 show that the Faster R-CNN with a ResNet50 backbone is able to correctly detect multiple birds even when they overlap. The Faster R-CNN with a MobileNetv3 backbone gives really good results and is 7x times faster than the ResNet50 backbone according to Table 4.1. Sometimes they both miss overlapping birds. The Faster R-CNN with MobileNetv3 320 are not able to deal with birds of small size. It can't detect overlapping birds or birds close to each others but isn't so bad to detect isolated non-overlapping birds. It seems to work well if the birds have a sufficient size.

The results on large images on which the SAHI algorithm is used, is shown in Figure 4.14. SAHI cuts the images into overlapping 224×224 slices. The images have a resolution of 3456×6144 corresponding to 527 slices per image. The first image gives the detections of the ResNet50 model without SAHI. It cannot detect precisely the birds. The other models don't detect anything. With SAHI, it seems that the two MobileNetv3 models work better than the ResNet50 model. They are even able to detect the very small birds above the horizon. Moreover the three models make many false detections on non-birds objects which is not a surprise. The detection times with the Resnet50, MobileNetv3 and MobileNetv3 on one

image, are 222.61, 203.1 and 43 seconds respectively.

4.4.1 Comparison with/without additional non-bird dataset

The results in Figure 4.13 and 4.14 are obtained by the models trained on the bird dataset. In this subsection, they are compared with the models trained with an additional non-bird dataset (nbd) explained in Section 3. The three Faster R-CNN models are trained in the same way as in Section 4.3. The image on the left in Figure 4.15 shows that the ResNet50 Faster R-CNN without nbd can't make the difference between a plane and a bird, and make false detections on non-bird objects. The idea with this additional non-bird dataset, was to make the models more robust for this kind of situation. In fact, the same model trained on nbd doesn't detect anything on this image. But the image on the right shows that it has difficulties that detect overlapping birds or birds very close to each others. It may be due to the fact that the non-bird dataset reduces the percentage of bird images in the entire dataset. This problem may be solved by adding more data and increase the data augmentation. This observation can also be found in Figure 4.16 where the models trained with nbd, are used with SAHI on a large image. One can see that the ResNet50 model with nbd, is able to detect every bird and doesn't make any false detections. For the other models, one can remark that the number of false detections of non-bird objects is reduced.

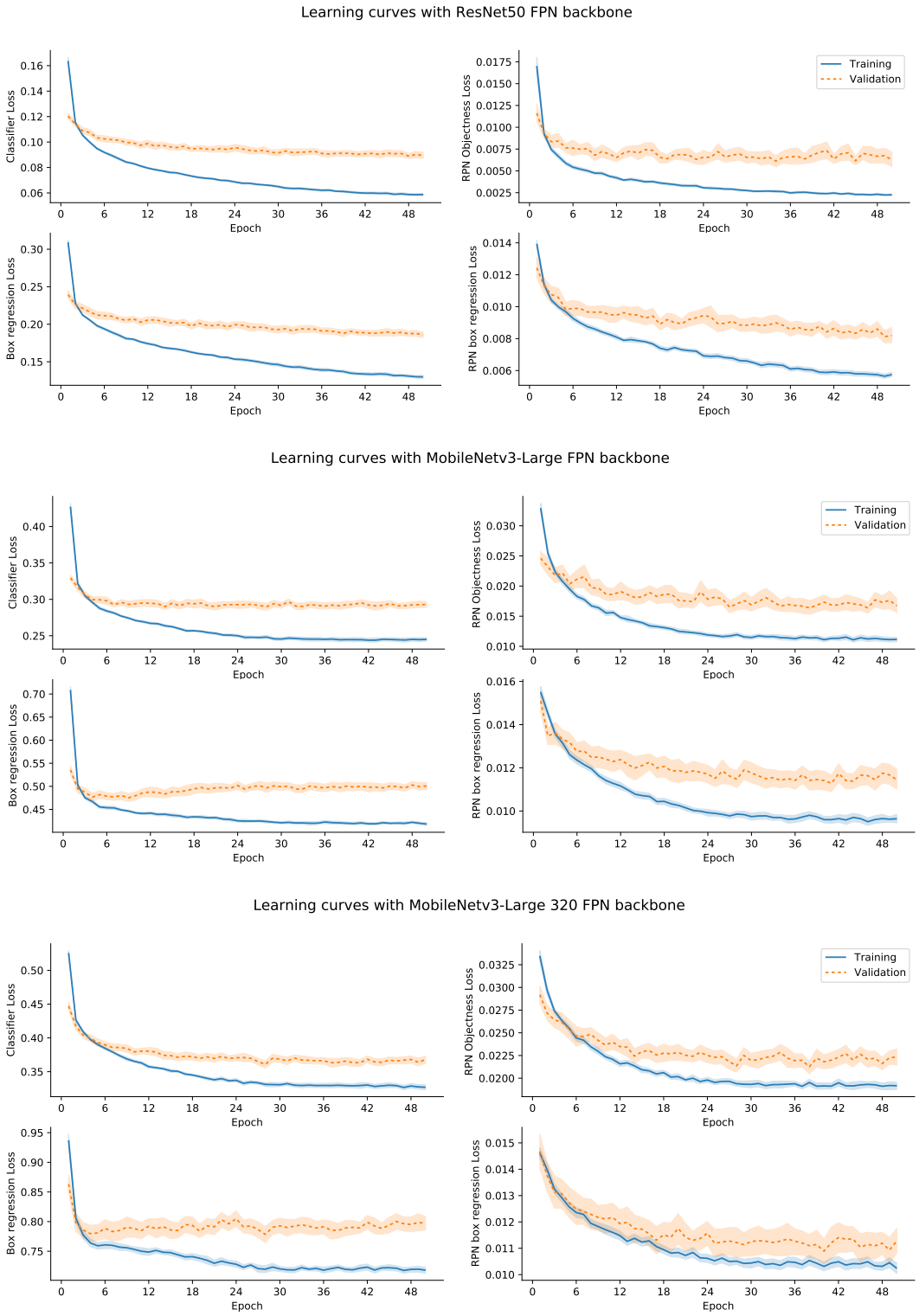


Figure 4.10: Learning curves for the three models. They show the mean and the 95% confidence interval of the loss values at each epoch over 30 independent runs.

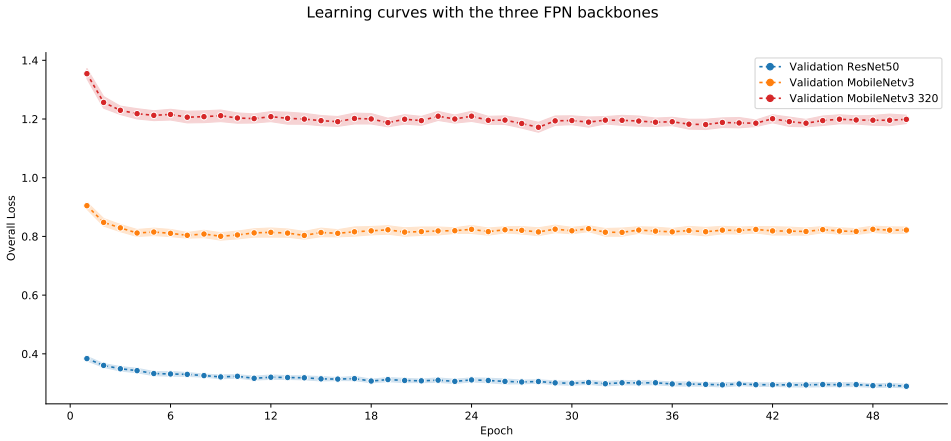


Figure 4.11: Learning curves of the overall validation loss for the three models. The overall loss is the average of the four validation losses. The minimum overall loss for ResNet50, MobileNetv3 and MobileNetv3 320, are 0.26, 0.74 and 1.13 respectively.

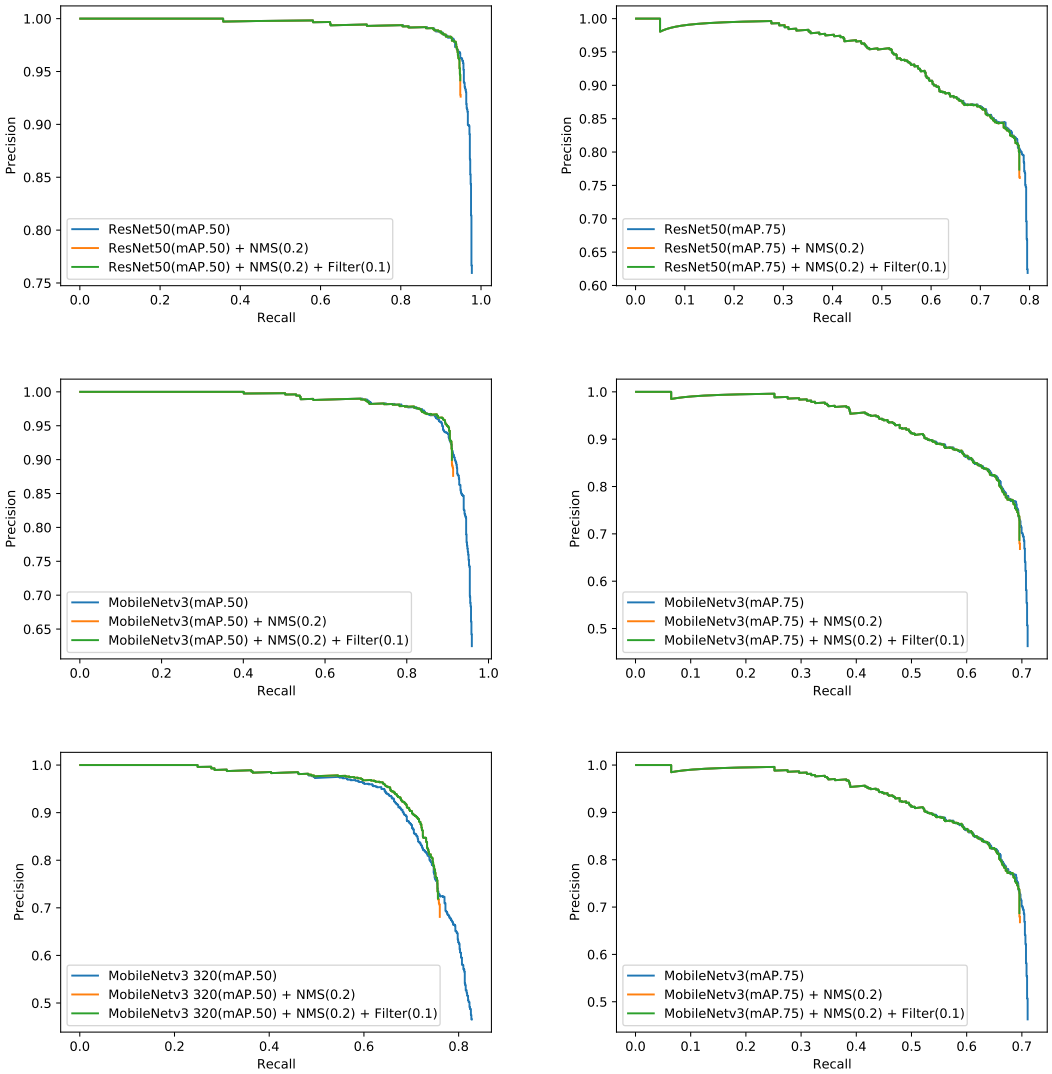
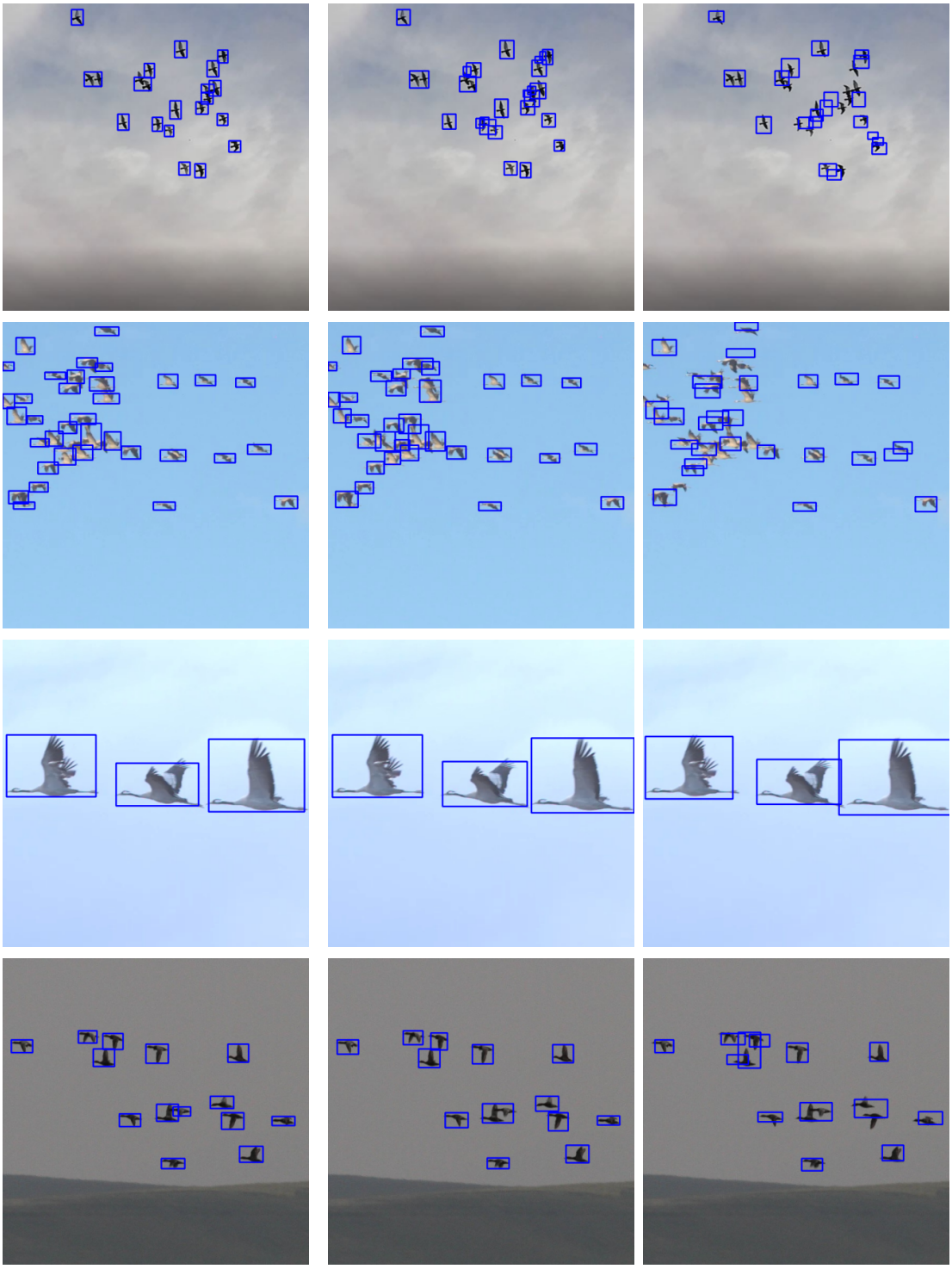
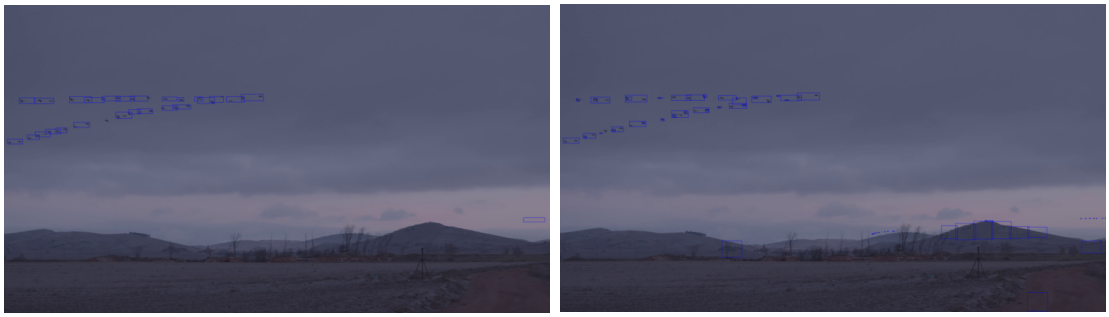


Figure 4.12: Precision-recall curves for the three models and the two different mAP with a threshold of 0.5 and 0.75. The blue curve shows the detection results of the models. The orange curve shows these results after NMS with an IOU threshold of 0.2. The red one shows the detection results after NMS and a filter with a threshold of 0.1.



ResNet50 MobileNetv3 MobileNetv3 320

Figure 4.13: Detection examples with the three models. The first column corresponds to the ResNet50 model, the second to MobileNetv3 model and the last one to MobileNetv3 320 model. The images have a resolution of 576×576 as the images from the dataset.



ResNet50 ResNet50 + SAHI

MobileNetv3 + SAHI MobileNetv3 320 + SAHI

Figure 4.14: Detection with the three models using the SAHI algorihtm with 224×224 slices. The first image is without SAHI. The images have a resolution of 3456×6144.

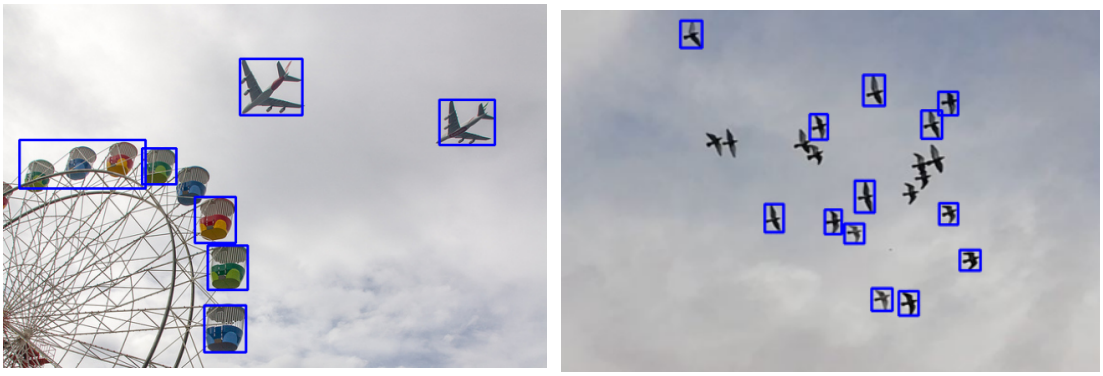


Figure 4.15: **Left:** Detection with the ResNet50 model trained without the non-bird dataset. The model trained with it, doesn't detect anything. **Right:** Detection with the ResNet50 model trained with the non-bird dataset. The detection without it, is shown in Figure 4.13.

ResNet50+SAHI without nbd

ResNet50+SAHI with nbd

MobileNet+SAHI with nbd

MobileNet 320+SAHI with nbd

Figure 4.16: Detection comparison of the three models on large image trained with an additional non-bird dataset (nbd). The SAHI algorithm is used with 224×224 slices. The images have a resolution of 3456×6144.

Chapter 5

Keypoint Detection

In this section, a very simple method based on [25] and [26] is used to extract the key points of the birds. The key points are the head, the tail and the two wings. For the moment it is limited to birds flying in side view but it can be extended as explained in the papers. Moreover, this method can not handle overlapping birds or small birds. Once the bird detection is made on the image, every region surrounded by a bounding box is extracted. An example is shown in Figure 5.1. The key point extraction is made on every region. It is then important to have a good bird detector. The idea is to substract the background from the image and isolate the bird (2). Then the contour of the bird is found (3). If there are more than one contour, the biggest is kept. After that, the extreme point on the left (red) and on the right (blue) are extracted. The black point is the middle of the line joining these points. By looking at two consecutive frames, the head and the tail can be associated to these points knowing the flying direction of the bird. Finally the wings (green) are the farthest points from the longitudinal axis. The graphic 5.2 shows the distance between each point of the contour and this axis. The two local maximum of the curve are the wings of the birds (in the flying position). It is difficult to distinguish the left from the right wing. The image (4) shows the final result.

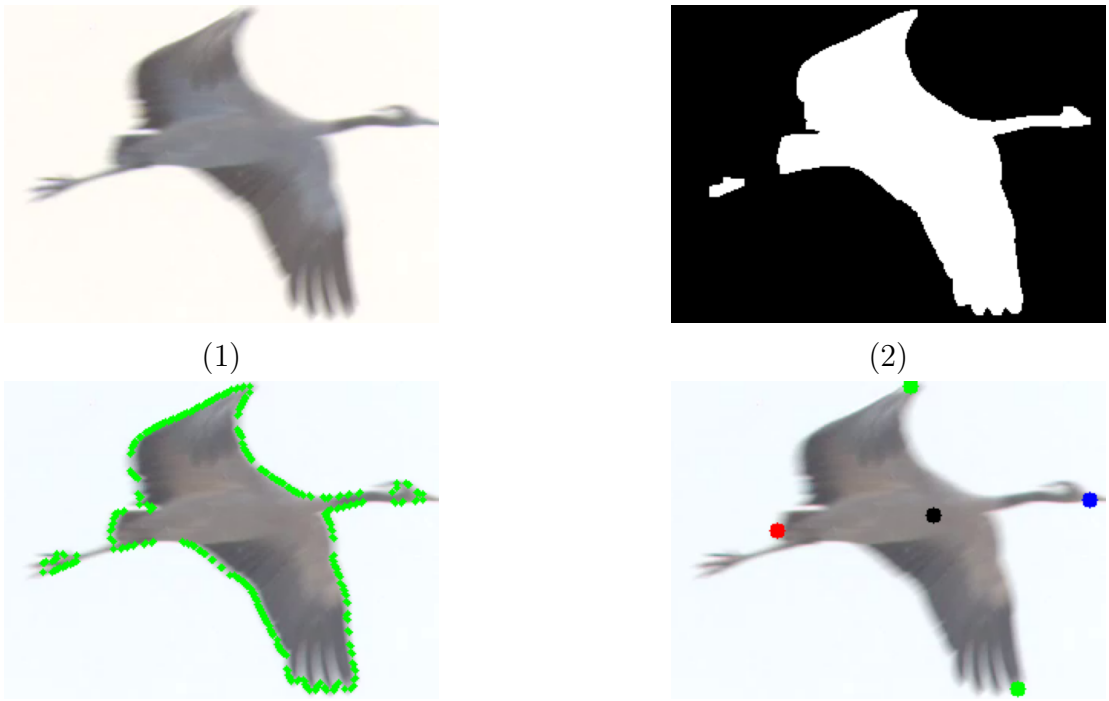


Figure 5.1: Illustration of the steps to extract the key points. (1) Extracted region after the detection made by Faster R-CNN. (2) Background substraction. (3) Contour of the bird. Only the biggest one is kept. (4) Final results. Green points are the wings, blue point the head and red point the tail.

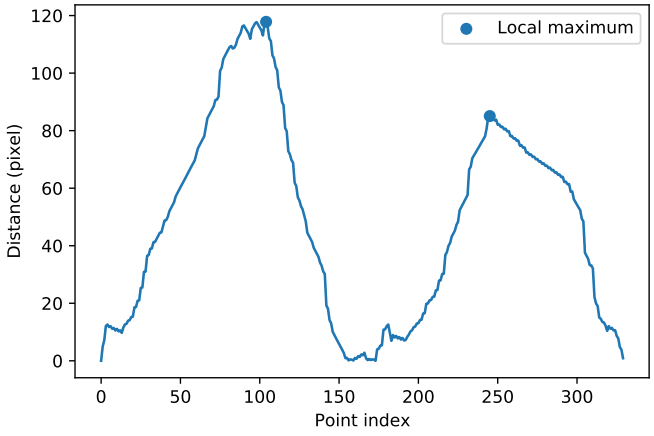


Figure 5.2: Distances between the points of the contour of the bird and the longitudinal axis. The two local maximum correspond to the wings.

Chapter 6

Conclusion

I investigated three different Faster R-CNN models, with a ResNet50, MobileNetv3 and MobileNetv3 320 feature extractor. I studied the trade-off between accuracy and speed. MobileNetv3 get very good detection results compared to ResNet50 while being much faster. They are both able to detect multiple bird and deal with overlapping birds. The speed of the MobileNetv3 is very impressive but comes with a very bad accuracy. It can only detect non-overlapping birds which are not too close and have a sufficient size. Moreover, the SAHI algorithm allows to detect the birds, potentially small, on very large images which is often the case with bird photographs. Despite the false detections made on the ground, MobileNetv3 and MobileNetv3 320 get good detection results (with non-overlapping birds) while being 11 and 52 times faster respectively than ResNet50 on a 3456×6144 image. With this algorithm for large image, one can see the advantage to have a fast model even if one loses a bit of accuracy. In addition to that, by training these models on the same dataset with an additional non-bird dataset, promising results were obtained. The idea with the non-bird dataset is to increase the robustness on non-bird objects, especially planes which have a similar shape as the birds. In fact, the false detections on non-bird objects are reduced but the models, mainly ResNet50, are no longer able to detect properly the overlapping birds or birds that are close to each other. These observations are probably due to the number of non-bird images which only corresponds to 20% of the bird dataset. By increase the size of the dataset, one could solve these problems and obtain much more robust and efficient models. In this thesis, videos are processed frame by frame independently and so, the movement of the bird isn't taken into account. I tested the SORT tracking algorithm [6] to track the birds into the videos and try to improve the detection, for example when birds start to overlap in future frames. I didn't succeed to get relevant results but it needs more investigations.

In regards to the keypoint detection, the provided method is very simplified

and can be improved. However, deep learning techniques are very popular today. The problem is always the lack of data. For the moment, most of the available datasets for keypoint detection, are for humans, cars and some specific animals. To use the deep learning techniques for birds, one should first construct a labeled dataset. I labeled 60 images from the bird dataset with the four key points. Antoine Vanderschueren, research assistant at Université catholique de Louvain, trained the OpenPifPaf algorithm with them. The results have many errors but are interesting especially with the very small size of the dataset. Finally, the KeyPoint R-CNN, a extension of the Faster R-CNN, could be interesting. A branch is added next to the Fast R-CNN to perform the keypoint detection. Therefore, the bounding boxes and the key points are obtained in the same time using the same feature extractor. This could also lead to better detections by forcing the network to learn more about the representation of birds.

Bibliography

[1] Peter (pestipeti). *Pytorch Starter - FasterRCNN Train*. Ed. by kaggle.com. 2020. URL: https://www.kaggle.com/code/pestipeti/pytorch-starter-fasterrcnn-train/notebook.

[2] Hüseyin Gökhan Akçay et al. "Automated Bird Counting with Deep Learning for Regional Bird Distribution Mapping". In: *Animals* 10.7 (2020). ISSN: 2076-2615. DOI: 10.3390/ani10071207. URL: https://www.mdpi.com/2076-2615/10/7/1207.

[3] Fatih Cagatay Akyon, Sinan Onur Altinuc, and Alptekin Temizel. "Slicing Aided Hyper Inference and Fine-tuning for Small Object Detection". In: *arXiv preprint arXiv:2202.06934* (2022).

[4] Fatih Cagatay Akyon et al. *SAHI: A lightweight vision library for performing large scale object detection and instance segmentation*. Nov. 2021. DOI: 10.5281/zenodo.5718950. URL: https://doi.org/10.5281/zenodo.5718950.

[5] Laith Alzubaidi et al. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". In: *Journal of Big Data* 8.1 (Mar. 2021), p. 53. ISSN: 2196-1115. DOI: 10.1186/s40537-021-00444-8. URL: https://doi.org/10.1186/s40537-021-00444-8.

[6] Alex Bewley et al. "Simple online and realtime tracking". In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

[7] Navaneeth Bodla et al. *Soft-NMS – Improving Object Detection With One Line of Code*. 2017. DOI: 10.48550/ARXIV.1704.04503. URL: https://arxiv.org/abs/1704.04503.

[8] Léon Bottou. "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *Proceedings of COMPSTAT'2010*. Ed. by Yves Lechevallier and Gilbert Saporta. Heidelberg: Physica-Verlag HD, 2010, pp. 177–186. ISBN: 978-3-7908-2604-3.

[9] Alexander Buslaev et al. “Albumentations: Fast and Flexible Image Augmentations”. In: *Information* 11.2 (2020). ISSN: 2078-2489. DOI: 10.3390/info11020125. URL: https://www.mdpi.com/2078-2489/11/2/125.

[10] Adrien Delhez. *Multiple Bird & Skeleton Detection Toolbox*. https://github.com/Leo166/BirdDetection-FeaturesExtraction.git. 2022.

[11] Sara Fraixedas. “Bird populations in a changing world: implications for North European conservation”. PhD thesis. May 2017.

[12] Ahmed Fawzy Gad. *Faster R-CNN Explained for Object Detection Tasks*. Ed. by blog.paperspace.com. 2020. URL: https://blog.paperspace.com/faster-r-cnn-explained-object-detection/#:~:text=Faster%5C%20R%5C%2DCNN%5C%20is%5C%20a%5C%20single%5C%2Dstage%5C%20model%5C%20that%5C%20is,vector%5C%20from%5C%20each%5C%20region%5C%20proposal..

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[14] Tomasz Grel. *Region of interest pooling explained*. Ed. by deepsense.ai. 2017. URL: https://deepsense.ai/region-of-interest-pooling-explained/.

[15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[16] Andrew Howard et al. *Searching for MobileNetV3*. 2019. DOI: 10.48550/ARXIV.1905.02244. URL: https://arxiv.org/abs/1905.02244.

[17] Sambasivarao K. *Non-maximum Suppression (NMS) - A technique to filter the predictions of object detectors*. Ed. by towardsdatascience.com. Oct. 2019. URL: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c.

[18] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2016. DOI: 10.48550/ARXIV.1612.03144. URL: https://arxiv.org/abs/1612.03144.

[19] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2014. DOI: 10.48550/ARXIV.1405.0312. URL: https://arxiv.org/abs/1405.0312.

[20] Team PyTorch. *An overview of the ML models introduced in TorchVision v0.9*. Ed. by pytorch.org. 2021. URL: https://pytorch.org/blog/ml-models-torchvision-v0.9/.

[21] Team PyTorch. *Source code for TORCHVISION.MODELS.DETECTION.FASTER$_R$CNN*. Ed. by pytorch.org. URL: https://pytorch.org/vision/stable/_modules/torchvision/models/detection/faster_rcnn.html#fasterrcnn_resnet50_fpn.

[22] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. DOI: 10.48550/ARXIV.1506.01497. URL: https://arxiv.org/abs/1506.01497.

[23] Sik-Ho Tsang. *Review: FPN — Feature Pyramid Network (Object Detection)*. Ed. by towardsdatascience.com. Jan. 2019. URL: https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610.

[24] *Website of Xavi Bou - Ornithographies*. URL: https://www.xavibou.com/.

[25] Tianhang Wu, Xiaoyan Luo, and Qunyu Xu. "A new skeleton based flying bird detection method for low-altitude air traffic management". In: *Chinese Journal of Aeronautics* 31 (Jan. 2018). DOI: 10.1016/j.cja.2018.01.018.

[26] Qunyu Xu and Xiaofeng Shi. "A simplified bird skeleton based flying bird detection". In: *Proceeding of the 11th World Congress on Intelligent Control and Automation*. 2014, pp. 1075–1078. DOI: 10.1109/WCICA.2014.7052867.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**