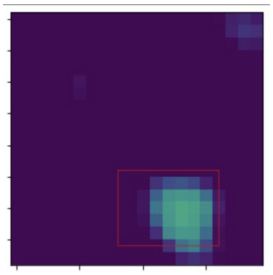
ALMA MATER STUDIORUM · UNIVERSITY OF BOLOGNA

SKA Data Challenge[1]



Deep Learning course final project

Lorenzo Cellini (lorenzo.cellini3@studio.unibo.it)
Alice Zandegiacomo (alice.zandegiacomo@studio.unibo.it)

September 24, 2021

[1] The code for the project is publicly available on GitHub

Contents

1 Summary

The tasks of object detection and classification have gained significant popularity over the past years within the deep learning and computer vision communities. Systems trained end-to-end now achieve great results on a variety of tasks in the video and image domains.

In this work, we address an object detection and classification problem on the Square Kilometer Array Dataset (SKADC1) [22]: given a large image, of about 32000x32000 pixels and 4GB in size, the goal is to detect astronomycal sources and classify them among five possible classes.

This project focuses on the SKADC1 dataset, in particular on the 560MHz-1000h high S/N sky image, that contains more than 19 000 radio sources.

Among the state of the art types of network, we opted for a two-stage system, specifically a Faster R-CNN.

In this work we implemented and compared three different models:

1. B16: a Faster R-CNN with a naive feature extraction backbone with only 4 convolutional layers and with a receptive field of 16 on the last convolutional layer, that will act as our baseline model;
2. B44: a Faster R-CNN with a feature extraction backbone with 7 convolutional layers and a receptive field of 44 on the last convolutional layer;
3. A Faster R-CNN with a larger backbone, specifically we implemented a VGG16 backbone without the last max pooling layer, with a receptive filed of 196 [21].

Each model consists of the same input and output structures (Region Proposal Network + Detector), and what changes is the deepness of the feature extraction network (usually called backbone).

In training the listed models, we adopted a transfer learning technique, because it has been proven to be effective in speeding up the training phase [25],[23], [24]. More specifically, we applied transfer learning and freezed the very first layers in order to inherit and retain the more basic features, while letting the model learn deeper representation of them.

While this could seem an easy object detecion and classification problem, it turns out that it is an hard task on both the goals, because:

- the objects to be located are very small, and
- the dataset is extremely unbalanced with respect to the class distribution.

For this two reasons we developed more than one model and we made some choices that will be discussed later.

Our experimental evaluations show that the best model is B16, the shallowest one.

2 Background

Our project is an object recognition tasks, which is a general term to describe the identification of objects in digital photos. The object recognition task consists in two aspects: object localization, which implies the drawing of an axis-aligned bounding box around one or more objects, and classification which consists in predicting the class of objects.

Among the state of the art top-performing deep learning models for object detection there are:

- R-CNN (Region-Based Convolutional Neural Network): it was firstly introduced by Girshick et al. in 2014 [7], then it was improved with the Fast [6] and Faster R-CNN in 2015 [18], and finally with the Mask R-CNN in 2017 [9]. In the first version of R-CNN the network had the duty to classify and find coordinates of objects on the regions proposed by a Selective Search algorithm that previously ran on the image. The computation of the feature map for each proposed region was carried out separately and this makes the R-CNN very slow. Fast R-CNN improved the execution speed of the first part of the network by computing the feature maps for the whole image and then using the SS proposals to cut feature map regions. The Faster R-CNN [18] substitutes the Selective Search of the Fast R-CNN with a Region Proposal Network without loosing accuracy. The Mask R-CNN is similar to the Faster R-CNN, but it uses the feature map to predict not only the class and bounding box for each region of interest, but also the pixel-level position of the object through an additional convolutional network.
- YOLO (You Only Look Ones): was proposed in 2015 by Joseph Redmon, et al [17]. The approach consists in a single neural network trained end to end that takes an image as input and directly predicts bounding boxes and their class labels. The network divides the input image into a SxS grid, where SxS is equal to the width and height of the tensor which presents the final prediction. In case the center of an object is in a grid cell, the grid cell takes responsibility for detecting that object. Moreover, each grid cell is simultaneously responsible for predicting bounding boxes and confidence scores which represent how confident is the model about bounding box containing an object. YOLOv2 uses anchor boxes as Faster R-CNN.
- SSD (Single Shot Detector) was proposed by Wei Liu et al. [13] in 2016, with the aim of having the high computational speed of YOLO, while maintaining the accuracy of Faster R-CNN. SSD enhances the speed of running time with respect to Faster R-CNN by eliminating the need of the Region Proposal Network. Therefore, it causes a few drop in mAP, and SSD compensates this by applying some improvements including multiscale features and default boxes. These improvements allow SSD to gain the same of Faster R-CNN using lower resolution images, which then further speeds up the processing of SSD.

3 Dataset description

The data provided by the SKA (Square Kilometer Array) challenge, consists in a series of astronomical high resolution images created through data simulations [2]. Images are a simulated SKA continuum image in total intensity at 3 frequencies: 560 MHz, 1.4 GHz and 9.2 GHz, each of witch has 3 different exposures: 8 h, 100 h, 1000 h. For each frequency band a catalogue revealing only a fraction of the simulated galaxies was released. We choose to analyse the image at 560 MHz with 1000 hours of exposure.

3.1 Ground truth data

Each row of the ground-truth catalogue contains the subsequent information of the correspondent source:

- RA (core) [degs] Right ascension of the source core
- DEC (core) [degs] Declination of the source core
- RA (centroid) [degs] Right ascension of the source centroid
- DEC (centroid) [degs] Declination of the source centroid
- FLUX [Jy] integrated flux density
- Core frac [none] integrated flux density of core/total
- BMAJ [arcsec] major axis dimension
- BMIN [arcsec] minor axis dimension
- PA [degs] PA (measured clockwise from the longitude-wise direction)
- SIZE [none] 1,2,3 for LAS, Gaussian, Exponential
- CLASS [none] 1,2,3 for SS-AGNs, FS-AGNs,SFGs
- SELECTION [none] 0,1 to record that the source has not/has been injected in the simulated map due to noise level
- x [none] pixel x coordinate of the centroid, starting from 0
- y [none] pixel y coordinate of the centroid,starting from 0

We used declination, right ascension, minor and major axis to obtain the pixel coordinates of the source center and its bounding box. At first we tried to filter the dataset out using the flux information and the primary beam given by the SKA challenge in order to discard sources with too low signal to noise ratio, but, at the end, we decided to adopt the cleaned dataset courtesy of the ICRAR group (The International Centre for Radio Astronomy Research) to have a comparison with their results.

We discarded all sources with SELECTION equal to 0 and assigned to each source a class label, given by the concatenation of SIZE and CLASS. In total there are 5 classes. Figure 3.1 shows an example of sources from each different class.

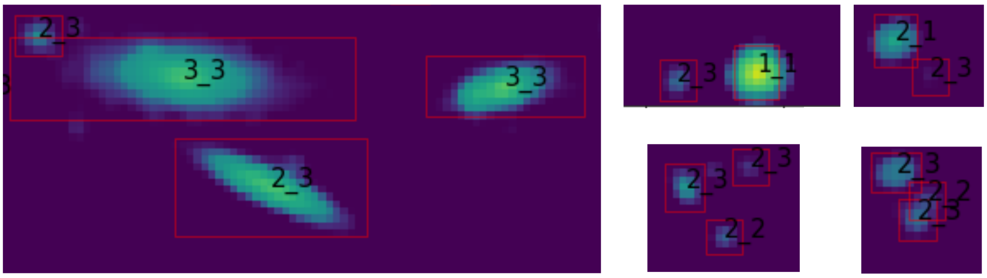


Figure 3.1: Examples of different classes of the ground truth dataset.

It is worth to notice that the differences in class are apparently not reflected in differences in shape or dimension. This is a crucial aspect that we will discuss later on.

An analysis of the distribution of the classes shows that the dataset is highly unbalanced, as can be seen from the scatter plots 3.2 in table 3.1. In particular class 2_3 is highly over represented, while the number of sources for the other classes is 2-3 orders of magnitude lower. We also analyzed the distribution of height and width by classes: the dimensions seem to be isotropic, but the standard deviation is very high for some classes, while extremely low for others. This can be seen in the scatter plot in 3.2.

Balancing this dataset is a tricky task because patches contain several sources of different classes, depending on patch dimension. Indeed we tried two different approaches to overcome this problem: find and re-sample patches that contain only the rare classes and use focal loss for the classification task.

CLASS	**number of sources**	**width (px)**	**height (px)**
1_1	112	15.9 ± 21.9	15.1 ± 13.2
2_1	34	6.2 ± 1.3	6.2 ± 1.5
2_2	262	5.0 ± 0.2	5.0 ± 0.2
3_3	234	16.1 ± 8.1	17.6 ± 10.8
2_3	18580	6.0 ± 2.1	6.1 ± 2.0

Table 3.1: Number of sources, width and height distribution for each class

3.2 Image analysis and preprocessing

The image size is 32768x32768 pixels while the portion of the image that contains the ground truth boxes is around 4000x4000 pixels. Images were given in FITS format. Since they are a simulation of a radiotelescope acquisition, the range of each pixel is between 0 and 1. The brightest source in

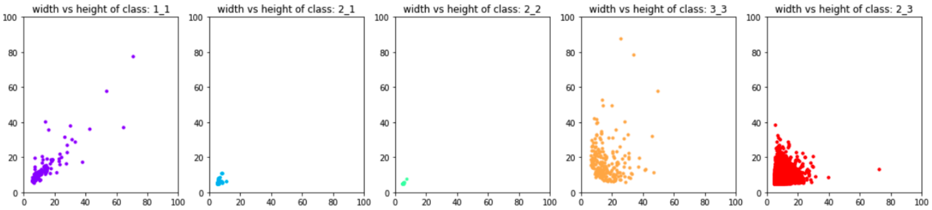


Figure 3.2: Dataset class distribution

the fits image has value of 0.006586 while the minimum gray level is -1.9×10^{-6} due to noise simulation.

We converted the pixel value range from [0,1] to [0,255] experimenting with different approaches: firstly we applied a linear transformation considering the maximum gray level value as 255 and set the lower value equal to 0. In this way we removed the negative noise signal. Unfortunately, because the sources intensity was not linearly distributed in the range [0,1], but the majority of them had a very low intensity, the resulting image was almost completely black with the exception of very few sources and almost all of the ground truth boxes were undetectable, 3.3 (on the left).

The second attempt was using a γ function in order to enhance the signal of low intensity sources. We tried several γ values between 0.2 and 1, but again many sources were too dark.

Finally, the chosen method was to scale the image intensity range using a base 10 logarithm scale: the magnitude range was defined by the magnitude of the noise standard deviation and of the maximum gray value, 3.3 (on the rigth). Indeed, the image noise is caused by electronic noise and is well fitted by a Gaussian distribution centered in zero. We obtained the standard deviation looking at the negative values of the gray level distribution, which are necessary caused by noise, and we use it to scale pixel values up.

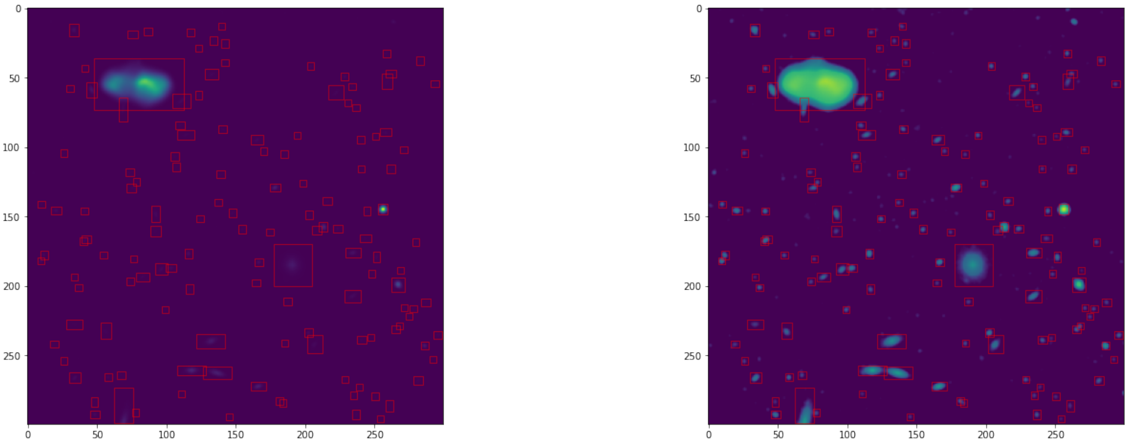


Figure 3.3: Linear transformation on the left and logarithmic transformation on the right, of the given image.

4 System description

In this work, we implemented three different models. All of them are a Faster R-CNN network and they differ on the backbone and hyperparameters.

In order to perform an effective comparison, the models share the same output architecture.

The following is a detailed description of the models. Beware that in each model section we report everything but what happens in the output layers, which is described once in 4.2.

4.1 Feature Extraction Backbone

Basically, in this work we tested:

1. wether a shallow model is enough for the small and simple-shape object detection task, and;
2. how detection performance varies when changing the receptive field size of the model with respect to objects size.

For this reasons we implemented three different feature extraction backbones with different receptive field size.

4.1.1 B16 - Baseline 16

The first backbone we implemented has the following structure:

1. Block 1: consists of 2 convolutional layers with 3x3x64 filters, followed by a ReLU activation layer and a 2x2 MaxPooling layer with stride 2;
2. Block 2: consists of 2 convolutional layers with 3x3x128 filters, followed by a ReLU activation layer and a 2x2 MaxPooling layer with stride 2;

Both the blocks have been initialized with the public available VGG16 weights and **freezed**.

It has been shown that, in computer vision tasks, the very first layers of a convolutional pipeline, learn the most basic features, as strokes or circles [26]. So, given that the objects we have to detect have simple shapes, like circles or ellipses, and that we think these kind of basic feature are common across different domains, we tried to transfer learning from the VGG16 and freezing the first layers, the ones that learn most basic features.

With the B16 model, we wanted to test how a shallow feature extraction network performs in predicting small and simple shapes objects.

Furthermore, we tested how the relation between the receptive field size and the object size, influences the learning: indeed, B16 has a receptive field size of 16 and this is smaller than any object size for the 20_100 set, but bigger than the 80% of the objects in the 50_100 set. More details in chapter 5.

With this architecture, the final feature map size r is $\frac{1}{4}$ of the input image.

4.1.2 B44 - Baseline 44

The second backbone we implemented has the following structure:

1. Block 1: consists of 2 convolutional layers with 3x3x64 filters, followed by a ReLU activation layer and a 2x2 MaxPooling layer with stride 2;
2. Block 2: consists of 2 convolutional layers with 3x3x128 filters, followed by a ReLU activation layer and a 2x2 MaxPooling layer with stride 2;
3. Block 3: consists of 3 convolutional layers with 3x3x256 filters, followed by a ReLU activation layer and a 2x2 MaxPooling layer with stride 2;

The first two blocks are the same as in the B16 model.

By adding the third block we wanted to test how performance changes with a deeper model.

The purpose of this backbone is also to make a direct performance comparison between B16 and B44 on the same input image size. Indeed, B44 has a receptive field size of 44, that is greater than the 90% of objects to detect (in the 20_100 set).

Each block is initialized with the VGG16 weights. Block 1 and 2 have been freezed, while block 3 is left free to learn.

With this architecture, the final feature map size is $\frac{1}{8}$ of the input image.

4.1.3 VGG16

The third backbone we implemented is the well known VGG16 backbone, without the fully connected part and without the last MaxPooling layer.

It has the following structure:

1. Block 1: consists of 2 convolutional layers with 3x3x64 filters, followed by a ReLU activation layer and a 2x2 MaxPooling layer with stride 2;
2. Block 2: consists of 2 convolutional layers with 3x3x128 filters, followed by a ReLU activation layer and a 2x2 MaxPooling layer with stride 2;
3. Block 3: consists of 3 convolutional layers with 3x3x256 filters, followed by a ReLU activation layer and a 2x2 MaxPooling layer with stride 2;
4. Block 4: consists of 3 convolutional layers with 3x3x512 filters, followed by a ReLU activation layer and a 2x2 MaxPooling layer with stride 2;
5. Block 5: consists of 3 convolutional layers with 3x3x512 filters, followed by a ReLU activation layer;

The first two blocks are the same as in the B16 model, while the third is the same as in the B44.

Each block is initialized with the VGG16 weights. Block 1 and 2 have been freezed, while block 3, 4 and 5 are left free to learn. The receptive field of this backbone is 196.

With this architecture, the final feature map size is $\frac{1}{16}$ of the input image.

4.2 Output Architecture

The output architecture is the same as of a standard Faster R-CNN module and consists of a Region Proposal Network (RPN) and a Roi Pooling Layer coupled with a fully connected network that we called "Detector".

This kind of network is called "two-stage" network because the training phase is carried out in two steps:

1. The feature maps generated by the backbone are feed into the RPN network. It generates a batch of proposals and pass them down to the Detector;
2. the Detector takes the feature maps and the proposals, uses the latter to cut the corresponding regions on the feature maps and classifies the cuts.

RPN and Detector share the same backbone weights and have distinct loss functions.

Depending on how the network works, the training phase is carried out in a stochastic gradient descent fashion: the network is fed with one single image at a time, but besides the feature extraction backbone there are some specific functions that generate the actual ground truth batch that RPN uses in order to compute its loss. The output of the RPN is then filtered through a non-maximum-suppression algorithm (NMS [16]) based on the intersection-over-union score (IoU [20]). From the result, an equal number of positive (foreground) and negative (background) samples are drawn and passed to the Detector. This number is one of the hyper-parameters of the model.

4.2.1 Region Proposal Network

"A Region Proposal Network (RPN) takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score." [18] The structure of this network is the following:

- 1 convolutional layer with 3x3x512 filters, initialized with a Gaussian distribution centered in 0 with a stdev of 0.01 [18], followed by a ReLU activation layer;
- 1 convolutional layer with 1x1xnum_classes filters, initialized with a uniform distribution;
- 1 convolutional layer with 1x1x4xnum_classes filters, initialized to the constant 0.

The second layer is in charge to determine the "objectness" score of a candidate region, (which can be interpreted as the probability that the region contains an object) while its sibling, the third layer, is in charge of outputting the difference in coordinates of such proposals with respect to model anchors.

The output of the RPN net goes trough the NMS algorithm based on IoU score. The results is the set of ROI proposals that the Detector will have to classify and locate.

Hyper-parameters of RPN module are:

- the number of regions to evaluate: 256;
- number of output ROI from NMS: 2000;
- NMS positive overlap threshold: 0.7, negative overlap threshold: 0.3.

The RPN loss function is:

$$L(p_i, t_i) = \frac{1}{N_{cls}^{rpn}} \sum_i L_{cls}^{rpn}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}^{rpn}} \sum_i p_i^* L_{reg}^{rpn}(t_i, t_i^*)$$

where the first term is the binary cross entropy and represents the loss related to the "objectness" score; the second term is the smooth L_1 score as in [10]. The p_i^* multiplier acts as a kind of mask, activating the regression loss only for positive proposals. λ is the weight of the regression term and in our experiments was set equal to 1 because in [18] has been proven that it doesn't make a great difference in performance.

4.2.2 Roi Pooling Layer

ROI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of approximately size $\frac{h}{H} \times \frac{w}{W}$ and then max-pooling the values in each sub-window. Pooling is applied independently to each feature map channel.

ROI pooling layer is needed because the last Faster R-CNN module is a fully connected network that needs fixed-length feature vector as input, but proposals from RPN can have different height and width. So, ROI pooling layer scales down and resizes proposals to a $n \times m$ map, where n and m are model hyper-parameters. We chose $n = m = 7$ as in [18].

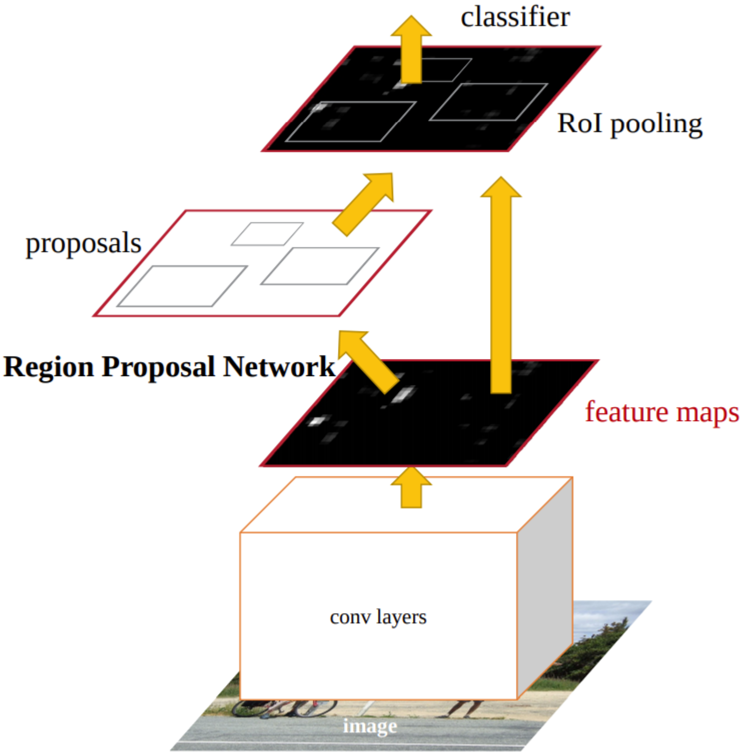


Figure 4.1: ROI Pooling Layer (image courtesy of [15])

4.2.3 Detector

The last Faster R-CNN module is a fully connected network that aims to classify RPN proposals among possible object categories and output bounding boxes coordinates refinements.

It consists of:

- ROI pooling layer (described above);
- 2 fully connected layers with 4096 units and 0.5 of dropout probability, each one followed by ReLU activation;
- 1 fully connected layer with *num_classes* units, initialized to 0 and followed by a softmax activation;
- 1 fully connected layer with $4 * num\_classes$ units, initialized to 0 and followed by a linear activation.

In our dataset we had $num\_classes = 6$, where 5 were actual classes and the last was the background class.

The model loss is:

$$L(p_i, t_i) = \frac{1}{N_{cls}^{det}} \sum_i L_{cls}^{det}(q_i, q_i^*) + \lambda \frac{1}{N_{reg}^{det}} \sum_i L_{reg}^{det}(s_i, s_i^*)$$

where the first term is the categorical cross entropy and represents the classification loss with respect to object classes and the second term is again the $smooth - L_1$ that represents the shift regression from the proposed regions to the actual coordinates shifts.

4.2.4 Focal Loss

In some of our experiments we also tried to overcome the unbalancedness of the dataset by exploiting focal loss [12]. Authors in [12] proposed focal loss as a way to down weight background examples with respect to foreground ones. Thus they claim that thanks to focal loss it is not necessary to restrict RPN proposal to a fixed number and ratio and to perform biased sampling on the Detector stage.

Anyway we implemented focal loss with a slightly different goal: to help the model to distinguish between rare and common classes positive samples. Indeed we maintained the fixed RPN proposals and applied 1:1 biased sampling to the proposed ROI. In this way we think the model could converge faster, and, once it starts detecting objects, the focal loss should help him better discriminate between classes. Moreover, in [12] they proposed focal loss for one-stage detectors, while we applied it to a two-stage detector.

In the experiments where focal loss is applied, the Detector loss becomes:

$$L(p_i, t_i) = \frac{1}{N_{cls}^{det}} \sum_i FL_{cls}^{det}(q_i, q_i^*) + \lambda \frac{1}{N_{reg}^{det}} \sum_i L_{reg}^{det}(s_i, s_i^*)$$

where

$$FL_{cls}^{det}(q_i, q_i^*) = \alpha(1 - \gamma)^2 CE(q_i, q_i^*)$$

where $CE(q_i, q_i^*)$ is the categorical cross entropy.

We adopted $\alpha = 0.25$ and $\gamma = 2$ as the best performing ones found in [12].

5 Experimental setup and results

We defined 8 experiments, which are a combination of the three different backbone models discussed before and the following parameters:

- dimensions of the input patches: we tried 20×20, 50×50 and 100×100 pixel patches (see section 5.2). The patches size causes both a variation in the number of ground truth boxes per patch and their size. The latter because, before feeding the network, we scaled patches up to the network input size: 100×100 px for B16 and B44, 600×600 px for VGG16;
- size and aspect ratio of anchors: for each model we defined a set of anchors given by the combination of their possible sizes and ratios. Anchors are drawn on the input image and the distance between the centers of two anchors sets is given by the stride of the network, being the latter the re-sizing factor between the input image and the last feature map. In this work strides were: 4 for B16, 8 for B44 and 16 for VGG16. Thus, for each model, the number of anchors is $anchor\_sizes \times anchor\_ratios \times feature\_map\_size$, where $feature\_map\_size$ is in the $r \times r$ format;
- focal loss and dataset balancing: as shown in section 3.1 the dataset is strongly unbalanced, so we tried to overcome this by using focal loss (chap 4.2.4) or by balancing the training dataset (section 5.2). These two approaches have been applied separately in different training configurations in order to make a meaningful comparison;
- normalization of the values of the input patches: in some of the experiments, patches pixel values have been normalized by dividing for the max patch pixel value. A more detailed explanation is given in the next section.

In table 5.1 and following chapters we adopt the nomenclature *baseline_n_m + FL(focal loss) or BD (balanced dataset)* to unequivocally identify the combination of feature extraction backbone and parameters used in the various tests. Here n and m represent the original patches dimension and the dimension after patch resizing.

The hyperparameters listed in table 5.2 were mainly taken from [18], in order to have a well-established benchmark, even though some of them were adjusted based on available resources (e.g. epochs num) or on our personal thoughts (e.g. we think that increasing Detector ROI would speed the training up).

In each test, 1 epoch consisted in 250 iterations.

5.1 Receptive Field

In this section we want to remark one of the central aspects of this project: we experimented on how the network receptive field interacts with the size of objects to be detected. Indeed we developed 3 different feature extraction backbones with different receptive field size.

As depicted in 5.1, the ground truth shapes distribution shows that the 99% of the objects is smaller than 18x18 pixels and the 90% is smaller then 8x8 pixels, before patch resizing.

Test name	Rec. field	Patch(px)	Input(px)	Anchors	Norm.	Focal loss	Balancing
B16_50_100	16	50x50	100x100	[8,16,32,64] [1:1,1:2,2:1]	Yes	No	No
B16_50_100+FL	16	50x50	100x100	[4,8,16,24,32,64] [1:1,1:2,2:1]	Yes	Yes	No
B16_20_100	16	20x20	100x100	[4,8,16,24,32,64] [1:1,1:2,2:1]	Yes	No	No
B16_20_100+FL	16	20x20	100x100	[4,8,16,24,32,64] [1:1,1:2,2:1]	Yes	Yes	No
B16_20_100+BD	16	20x20	100x100	[4,8,16,24,32,64] [1:1,1:2,2:1]	Yes	No	Yes
B44_20_100	44	20x20	100x100	[4,8,16,24,32,64] [1:1,1:2,2:1]	Yes	No	No
B44_20_100+FL	44	20x20	100x100	[4,8,16,24,32,64] [1:1,1:2,2:1]	Yes	Yes	No
VGG16_100_600	196	100x100	600x600	[32,64,128] [1:1,1:2,2:1]	No	No	No

Table 5.1: Experiments setup, defined by: test name, receptive field of the feature extraction backbone, patch size, input image size after scaling up, anchors size and ratio, normalization of the gray level values, whether focal loss or balancing were used.

	Epochs	RPN regions	Detector ROI	NMS neg-pos	Optimizer	Learning rate
B16	200	256	16	$0.7 - 0.3$	Adam	$1e - 2$
B44	400	256	16	$0.7 - 0.3$	Adam	$1e - 2$
VGG16	26	256	16	$0.7 - 0.3$	Adam	$1e - 2$

Table 5.2: Hyperparameters

In the *20_100* set, after resizing the distribution becomes: 99% smaller than 90x90 pixels and 90% smaller then 40x40 pixels. We used this set to feed B16 and B44 models that have respectively a receptive field size smaller then the 100% of objects to detect (B16) and bigger then the 90% of objects (B44).

In the *50_100* set, after resizing the distribution becomes: 99% smaller than 36x36 pixels and 90% smaller then 16x16 pixels. We used this set with B16 that has a receptive field equal to the size of 90% of objects to detect.

In the *100_600* set, after resizing the distribution becomes: 99% smaller than 108x108 pixels and 90% smaller then 48x48 pixels. We used this set with VGG16, which has a receptive field size of 196, bigger then any object to be detected.

We will discuss in 7 how the receptive field influenced the results.

5.2 Data pre-processing

Given the image and the dataset described in 3 we firstly cut out an around 4000x4000px image from the original image: this is the region that contains the ground truths from the dataset. Then we generated 3 set of patches, [20_100], [50_100], [100_600], in order to run different experiments. In the naming convention [*n*_*m*], *n* is the size of the cut out patch from the 4000x4000px image, and *m* is the scaled patch size before feeding the network.

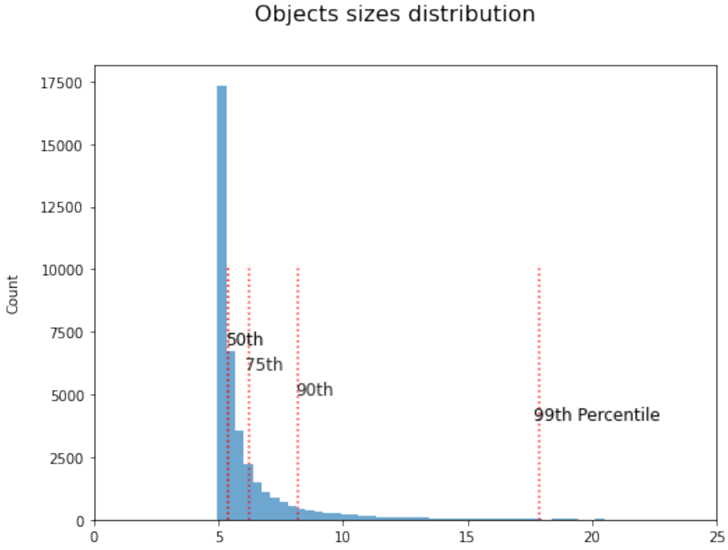


Figure 5.1: Objects size distribution.

During patches generation we cut a square of side $\sqrt{10000}$ in order to have a maximum amount of 10000 patches. We decided to use a step of size $\frac{n}{2}$ in order to avoid to lose the complete shape of the objects. From this set we removed patches with no ground truth, patches that contained part of blacklisted objects and patches smaller then n. The latter was in order to avoid issues on borders. We also discarded objects if they were captured for less than 80% of their surface.

Each patch was casted to the range 0-255 using the base 10 logarithmic scale, where the minimum value is given by the standard deviation of the noise computed over the entire image, while the maximum value is given by the magnitude of the highest value in the patch.

At training time we normalized patches values by dividing for their maximum value. We also tried zero-centering as in [25] but we discovered that patch normalization worked best.

Before feeding the network we replicated the same input image 3 times, one for each channel. Our code supports also an "Expander" layer, a 1x1x3 convolutional layer that converts a 1-channel image into a 3-channel one. At the end we decided to not use this layer based on experimental evidences.

Our code supports also augmentation on the fly, and in particular we implemented horizontal and vertical flipping, and the four 90° rotations. However we think this is not strictly necessary in this specific problem because shapes to be predicted are isomorphic with no intrinsic orientation. So we used it only when we balanced the dataset in order to add some training noise.

Regarding the splitting strategy for the dataset, we went for a simple holdout, thus setting aside 20% of the whole dataset for validation purposes. During splitting we can also decide whether to balance the dataset: we ran different experiments in order to compare this strategy with the focal loss mentioned in 4.2.4. In order to balance the dataset we simply find patches that contains the rare classes alone, we compute the frequencies of rare classes and we sample with repetition from these sets until the final class distribution becomes uniform. As it is this approach is not very robust, but it can be made better coupling the balancing with the augmentation described before.

We decided to run all of our experiments on a total amount of 350 patches, where 280 for the training set and 70 for the evaluation set. Obviously objects contained in a patch varies depending on the patch size.

Figures 5.2, 5.3 and 5.4 show the class distribution for the three set of input patches.

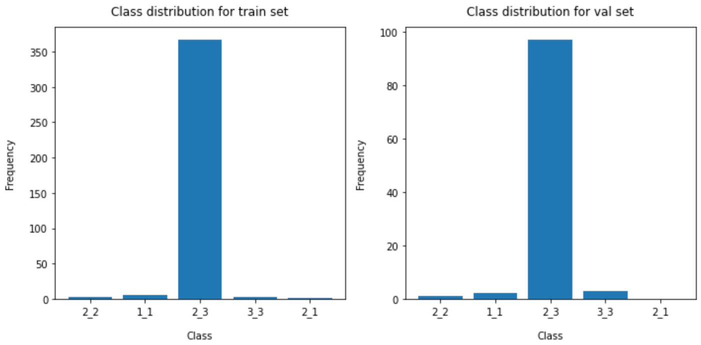


Figure 5.2: Class distribution for 20_100 set

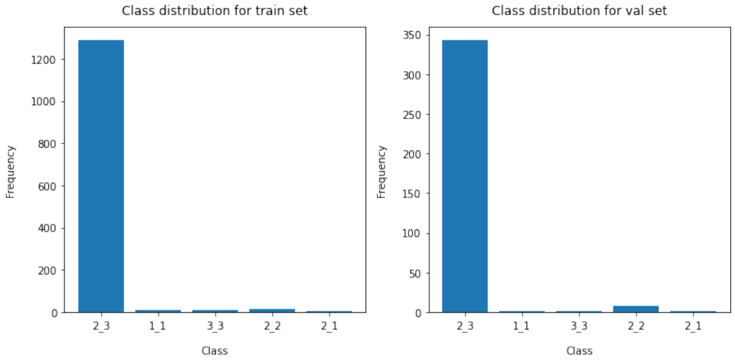


Figure 5.3: Class distribution for 50_100 set

5.3 Metrics

Metrics adopted to evaluate the models are mAP for the IoU threshold 0.5, macro-precision and macro-recall. The evaluation step is carried out at the end of each epoch: in this way we track metrics on validation set and save weights when mAP increases [5]. On the training set we track the four losses and classification accuracy of Detector.

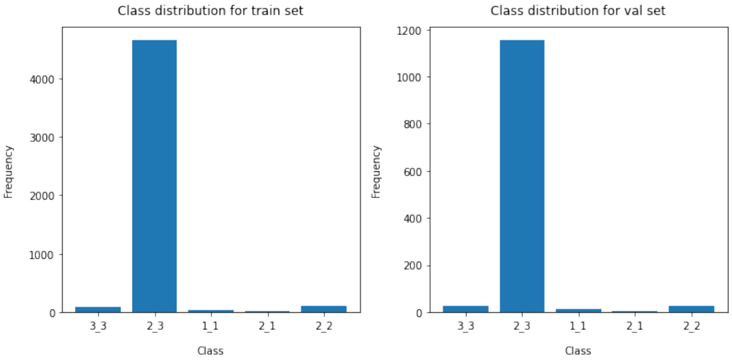


Figure 5.4: Class distribution for 100_600 set

5.4 Environment

The main third-party libraries on which the project is based on are Tensorflow [1] and Keras [4], being the first a machine learning system that operates at large scale and in heterogeneous environments and the latter an API that speeds deep learning model implementation up.

All of the training and validation processes were executed on a MacBook Pro with 2GHz Intel Core i5 quad-core, 16 GB RAM and an Intel Iris Plus Graphics 1536 MB, and on a Dell Precision 5540, 2.60 GHz Intel core i7, 16 Gb RAM.

We tried also to use Google Colaboratory [8], a platform that gives the possibility to exploit some computational resources for free, but because of its commercial limits, after few hours of batch computation they cut down our resources and training of models became 30x slower than on the MacBook.

5.5 Results

Table 5.3 shows the results obtained on the test set for each model, with the hyperparameters listed in table 5.2. Unfortunately, VGG16 required a lot of computational resources and we couldn't manage to train it for more than 26 epochs. Anyway, the first 2000 iterations of a Faster R-CNN are usually considered as warm-up [27] and in our case VGG16 ran for a total of 6500 iterations.

6 Analysis of results

For the results analysis we report here some examples of predictions for the best baseline model and for VGG16. Patches reported are among the ones with the highest $mAP_{.5}$ score.

As we can see in 6.1 a very shallow model as B16 is quite good in locating the objects: indeed, mAP score is mainly affected by the misclassification of objects of class different from "2_3". Instead, in fig 6.1 we can see that VGG16 has not reached good performance and the mAP result is due only to predicted boxes occasionally overlapped to ground truth.

	Training			Test		
	mAP$_{.5}$ (%)	**mPrec** (%)	**mRec** (%)	**mAP$_{.5}$** (%)	**mPrec** (%)	**mRec** (%)
B16_50_100	25.07	2.63	6.54	25.98	2.23	6.06
B16_50_100 + FL	41.72	3.13	3.73	**42.17**	1.77	1.63
B16_20_100	**41.74**	**23.29**	**38.27**	**39.50**	**23.64**	**43.45**
B16_20_100 + FL	28.61	1.25	3.21	30.48	2.67	5.48
B16_20_100 + BD + Aug	–	–	–	–	–	–
B44_20_100	15.18	2.65	13.36	15.46	3.27	14.52
B44_20_100 + FL	17.32	2.47	10.08	17.83	4.05	16.67
VGG16_100_600	27.22	1.01	1.46	25.97	0.56	0.78

Table 5.3: Best results

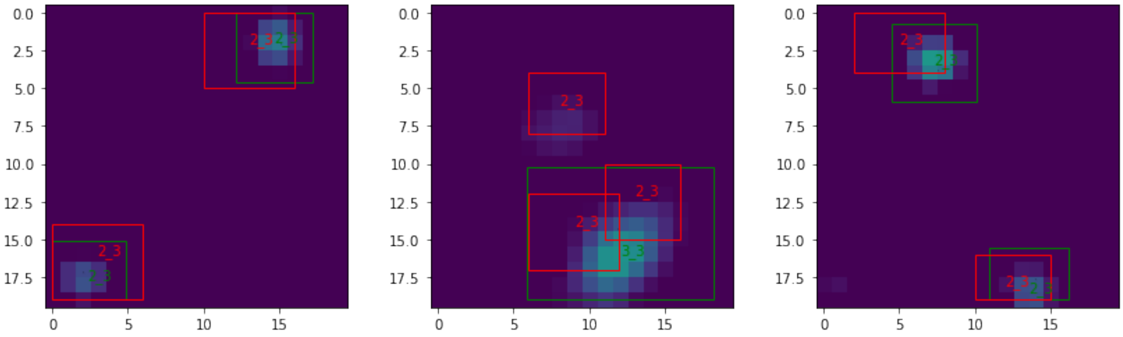


Figure 6.1: Examples of predictions for B16 model.

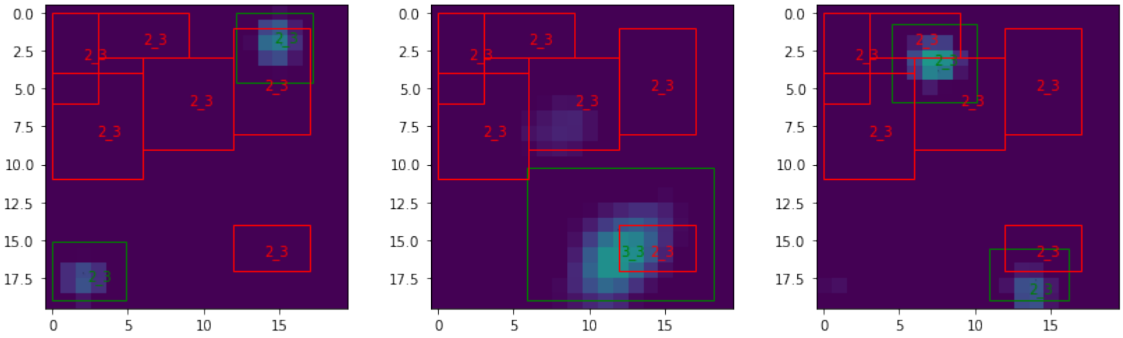


Figure 6.2: Examples of predictions for B44 model on the same images.

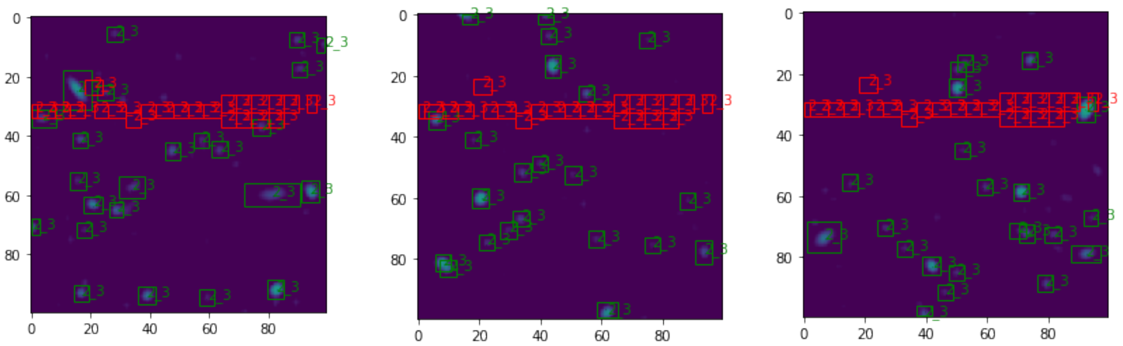


Figure 6.3: Examples of predictions for VGG16 model on images with the highest mAP.

7 Discussion

In this work, we addressed the problem of small object detection and classification on the SKADC1 dataset by leveraging a Faster R-CNN model coded from scratch. In particular we adopted Faster R-CNN architecture for the detection and recognition, but implemented our own backbones for feature extraction, besides the well known VGG16, based on the following ideas:

- objects are very small, so the receptive field of the network has to be chosen accurately [14];
- objects have very simple shapes, so there is no need for a very deep convolutional pipeline;
- objects are very similar, so there is no need for a large dataset;
- the vast majority of objects are isomorphic, so there is no need for image flipping and rotation during training, although we adopted augmentation when balancing the dataset;
- the overlapping of objects is very rare, so no complex shapes are created;
- the background is approximately uniform.

As described in 6 the best performing model is B16 without Focal Loss or Dataset Balancing, although its mAP score is not comparable to SoTA models, trained on PASCAL-VOC dataset or COCO annotation [18].

We think that Dataset Balancing, as it has been implemented in this work, is not working due to lack of characteristic differences between different objects classes as can qualitatively assessed by looking at figure 3.1. Feeding the model with objects very similar but from different classes, the result is to confuse it and making it not able to classify. Indeed, when balancing the dataset, as in B16_20_100+BD experiment, the model is not anymore able to learn in the time frame of 200 epochs. We think that this is due to the strong similarity in shape and brightness between classes (at least in the 560MHz and 1000h exposure). It could be interesting to try addressing this point by using other images of the same frequencies and different exposure time as the other channels of the input image, instead of repeating 3 times the same input image. We believe this could be a valid point because we are talking about astronomical objects which could have different emission spectrum.

For what concerns Focal Loss and the novel way we exploited it, we think that it is not sufficient to correct the deep unbalancedness of the dataset.

The principal source of error, the one that makes mAP scores to be low, is the object misclassification: all the baseline networks overfitted on the most common class and this makes mAP to be low also when the object is detected.

The interesting thing is that a simple model as B16 outperforms, in metrics and efficiency, a complex one like VGG16. We must say that the complexity and deepness of VGG16 needs it to be trained for hundreds of epochs, as in [25], while we managed to train it only for a total of 26 epochs, meaning 6500 iterations on images, because it took 45" per image on our hardware, and so, training it for 80000 iterations as in [25] would require approximately 45 days of training. Furthermore, we observed better results when normalizing input images for B16 and B44, but we skipped normalization for VGG16 as in [21]. It could be interesting to try training VGG16 for more epochs and comparing trainings with and without normalization.

We don't take for granted the fact that the best setting was freezing the first layer in each model. Indeed it could be interesting to try letting also the first B16 layers to learn (after having transferred learning from VGG16) and see what happens. Obviously this would require a greater computational effort.

Future improvements could be using Feature Pyramid Network as in [11] or feature fusion as in [19], in order to produce more accurate feature maps for smaller objects, or to implement the tweaks described in [3] to enhance Faster R-CNN performance specifically for small objects.

We think also that combining the different sky image provided in the challenge, as they were different channel in an RGB image, could bring more information to the network and could lead to a performance improvements.

A crucial aspect that could make the difference is the presence/absence of context as in [19]. They proved that context around small objects helps models to recognize them (that is why feature fusion works), while in our problem, the background around the object is pretty much the same everywhere, so the task is even harder.

Another hypothesis, that we would like to test with more time, is that the Faster R-CNN network is not well suited for this specific problem: we think that it would be worth to try a system where the dection is carried out through a **segmentation** approach and, as a second step, a classifier that classifies detected objects.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] A Bonaldi, T An, M Brüggen, S Burkutean, B Coelho, H Goodarzi, P Hartley, PK Sandhu, C Wu, L Yu, et al. "Square Kilometre Array Science Data Challenge 1: analysis and results". In: *Monthly Notices of the Royal Astronomical Society* 500.3 (2021), pp. 3821–3837.

[3] Changqing Cao, Bo Wang, Wenrui Zhang, Xiaodong Zeng, Xu Yan, Zhejun Feng, Yutao Liu, and Zengyan Wu. "An Improved Faster R-CNN for Small Object Detection". In: *IEEE Access* 7 (Aug. 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2932731.

[4] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[5] Ahmed Fawzy Gad. *Evaluating Object Detection Models Using Mean Average Precision (mAP)*. URL: https://blog.paperspace.com/mean-average-precision/.

[6] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV].

[8] Google. *Colaboratory*. URL: https://colab.research.google.com/.

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV].

[10] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73 –101. DOI: 10.1214/aoms/1177703732. URL: https://doi.org/10.1214/aoms/1177703732.

[11] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV].

[12] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV]. URL: https://arxiv.org/abs/1708.02002.

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. "SSD: Single Shot MultiBox Detector". In: *Lecture Notes in Computer Science* (2016), 21–37. ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[14] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. *Understanding the Effective Receptive Field in Deep Convolutional Neural Networks*. 2017. arXiv: 1701.04128 [cs.CV].

[15] Kaushik Patnaik. *Annotated RPN, ROI Pooling and ROI Align*. URL: https://kaushikpatnaik.github.io/annotated/papers/2020/07/04/ROI-Pool-and-Align-Pytorch-Implementation.html.

[16] Kaushik Patnaik. *Non Maximum Suppression*. URL: https://paperswithcode.com/method/non-maximum-suppression.

[17] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[18] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[19] Yun Ren, Changren Zhu, and Shunping Xiao. "Small Object Detection in Optical Remote Sensing Images via Modified Faster R-CNN". In: *Applied Sciences* 8.5 (2018). ISSN: 2076-3417. DOI: 10.3390/app8050813. URL: https://www.mdpi.com/2076-3417/8/5/813.

[20] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*. 2019. arXiv: 1902.09630 [cs.CV].

[21] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[22] SKA. *SKA web site*. URL: https://www.skatelescope.org/news/ska-launches-science-data-challenge/.

[23] Jonti Talukdar, S. Gupta, P. Rajpura, and Ravi Hegde. "Transfer Learning for Object Detection using State-of-the-Art Deep Neural Networks". In: Feb. 2018, pp. 78–83. DOI: 10.1109/SPIN.2018.8474198.

[24] Kafeng Wang, Xitong Gao, Yiren Zhao, Xingjian Li, Dejing Dou, and Cheng-Zhong Xu. "Pay Attention to Features, Transfer Learn Faster CNNs". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=ryxyCeHtPB.

[25] Chen Wu et al. "Radio Galaxy Zoo: Claran – a deep learning classifier for radio morphologies". In: *Monthly Notices of the Royal Astronomical Society* 482.1 (Oct. 2018), pp. 1211–1230. ISSN: 0035-8711. DOI: 10.1093/mnras/sty2646. eprint: https://academic.oup.com/mnras/article-pdf/482/1/1211/26205089/sty2646.pdf. URL: https://doi.org/10.1093/mnras/sty2646.

[26] Matthew D Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. 2013. arXiv: 1311.2901 [cs.CV].

[27] Zhi Zhang, Tong He, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. *Bag of Freebies for Training Object Detection Neural Networks*. 2019. arXiv: 1902.04103 [cs.CV].