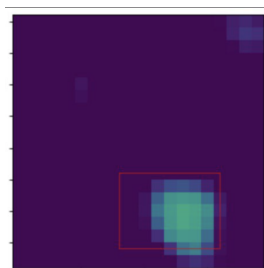


## SKA Data Challenge<sup>1</sup>



## Deep Learning course final project

Alice Zandegiacomo ([alice.zandegiacomo@studio.unibo.it](mailto:alice.zandegiacomo@studio.unibo.it))  
Lorenzo Cellini ([lorenzo.cellini3@studio.unibo.it](mailto:lorenzo.cellini3@studio.unibo.it))

September 12, 2021

<sup>1</sup>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) [**ska-site**]: given a large image, of about 32000 pixels on each side and 4GB in size, the goal is to detect astronomical sources and classify them among three possible classes.

This project focuses on the SKADC1 dataset, in particular on the 560MHz-1000h high S/N sky image, that contains more than 19000 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 field of 196 [**vgg**].

Each model consists of the same input and output (Region Proposal Network + Detector) structure, 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 [**claran**],[**transfer-learning-1**], [**transfer-learning-2**]. More specifically, we applied transfer learning only on the very first layers in order to inherit and retain the more basic feature, while letting the model learn deeper representation of them.

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

- the objects to be located are very small, as we will show later, and
- the dataset is extremely unbalanced with respect to 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 ...

## 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 ) Model Family: it was firstly introduced by Girshick et al. in 2014 [**rcnn**], then it was improved with the Fast [**fast-rcnn**] and Faster R-CNN in 2015 [**faster-rcnn**], and finally with the Mask R-CNN in 2017 [**mask-rcnn**]. In the first version of R-CNN the network involved 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 [**faster-rcnn**] substitutes the Selective Search of the Fast R-CNN with a Region Proposal Network without losing 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) Model Family: was proposed in 2015 by Joseph Redmon, et al [**yolo**]. 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  $S \times S$  grid, where  $S \times S$  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. [**ssd**] 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 [bonaldi2021square]. Images are a simulated SKA continuum image in total intensity at 3 frequencies: 560 MHz, 1.4 GHz and 9.2 GHz, each of which 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) [deg] Right ascension of the source core
- DEC (core) [deg] Declination of the source core
- RA (centroid) [deg] Right ascension of the source centroid
- DEC (centroid) [deg] 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 [deg] 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. We assigned to each source a class label, given by the concatenation of SIZE and CLASS and all sources with selection equal to 0 were discarded. 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.

	Epochs	Batch size	Optimizer	Learning rate
<b>Baseline</b>	30	128	Adam	$1e - 3$
<b>BiDAF</b>	12	60	Adadelta	0.5
<b>BERT</b>	3	8	Adam	$5e - 5$
<b>DistilBERT</b>	3	16	Adam	$5e - 5$
<b>ELECTRA</b>	3	8	Adam	$5e - 5$

Table 3.1: Hyperparameters

## 3.2 Image analysis

The image size is (32768 32768) pixels while the portion of the image that contains the ground truth boxes is 4000 x 4000 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. Negative values are due to the noise simulation. We convert the fits image in png image, in order to have patches with 255 gray levels. The brightest source in the fits image has value of 0.006586 while the minimum gray level is ... due to noise. To transform the float values into integer ones we tried different approaches, firstly we applied a linear transformation considering the maximum GL value as 255 and set the lower value equal zero. In this way we delete the negative noise signal. We discarded this attempt because the distribution of sources intensities was not linearly distributed between min and max range, but the majority of them has a very low intensity. The image transformed into integers was almost completely black with the exception of very few sources and most of the ground truth boxes were undetectable (fig ...). A second attempt was performing a gamma transformation in order to enhance the signal of low intensitys sources. We tried several gamma values between 0.2 and 1 but also with this method many sources were too dark. The choose 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. The image noise is caused by electronic noise and is well fitted by a gaussian distribution centered in zero. We obtain the standard deviation looking at the negative values of the gray level distribution, which are necessary caused by noise.

## 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 ??.

### 4.1 Feature Extraction Backbone

Basically, in this work we tested:

1. whether a simple model, meaning "not very deep", is enough for the small 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 backbone 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  $3 \times 3 \times 64$  filters, followed by a ReLU activation layer and a  $2 \times 2$  MaxPooling layer with stride 2;
2. Block 2: consists of 2 convolutional layers with  $3 \times 3 \times 128$  filters, followed by a ReLU activation layer and a  $2 \times 2$  MaxPooling layer with stride 2;

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

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 [**cnn-features**]. 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 ??.

With this architecture, the final feature map size 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  $3 \times 3 \times 64$  filters, followed by a ReLU activation layer and a  $2 \times 2$  MaxPooling layer with stride 2;
2. Block 2: consists of 2 convolutional layers with  $3 \times 3 \times 128$  filters, followed by a ReLU activation layer and a  $2 \times 2$  MaxPooling layer with stride 2;
3. Block 3: consists of 3 convolutional layers with  $3 \times 3 \times 256$  filters, followed by a ReLU activation layer and a  $2 \times 2$  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  $3 \times 3 \times 64$  filters, followed by a ReLU activation layer and a  $2 \times 2$  MaxPooling layer with stride 2;
2. Block 2: consists of 2 convolutional layers with  $3 \times 3 \times 128$  filters, followed by a ReLU activation layer and a  $2 \times 2$  MaxPooling layer with stride 2;
3. Block 3: consists of 3 convolutional layers with  $3 \times 3 \times 256$  filters, followed by a ReLU activation layer and a  $2 \times 2$  MaxPooling layer with stride 2;
4. Block 4: consists of 3 convolutional layers with  $3 \times 3 \times 512$  filters, followed by a ReLU activation layer and a  $2 \times 2$  MaxPooling layer with stride 2;
5. Block 5: consists of 3 convolutional layers with  $3 \times 3 \times 512$  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 on in two step:

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 feed 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 [**nms**]) based on the intersection-over-union score (IoU [**iou**]). 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." [**faster-rcnn**] 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 [**faster-rcnn**], 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, 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 [huber-loss]. The  $p_i^*$  multiplier acts as a kind of mask, activating the regression loss only for positive proposals.

#### 4.2.2 Roi Pooling Layer

ROI max pooling works by dividing the hwx RoI window into an HxW 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 [faster-rcnn].

#### 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 with  $num\_classes$  units, initialized to 0 and followed by a softmax activation;
- 1 fully connected 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  $smoothL_1$  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 [focal-loss]. Authors in [focal-loss] proposed focal loss as a way to down

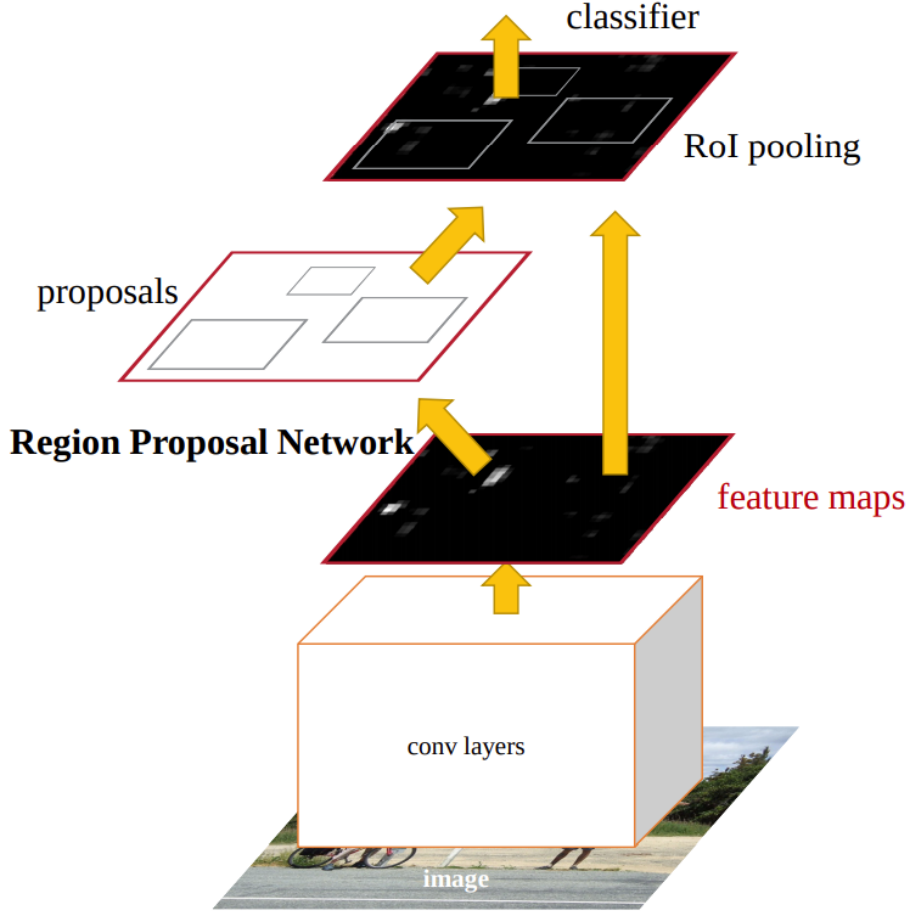


Figure 4.1: ROI Pooling Layer (image courtesy of [roi-pooling])

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 positive samples. Indeed we maintained the fixed RPN proposals and applied 1:1 biased sampling to the ROI proposed. In this way we think the model could converge faster, and, once it starts detecting objects, the focal loss should help him in better discriminate between classes. Moreover, in [focal-loss] 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 [**focal-loss**].

## 5 Experimental setup and results

DA METTERE DA QUALCHE PARTE: [As mentioned before, the goal was to detect very small objects, with respect to the dimensions usually detected in other deep learning works [CITAZIONE QUI].

B16 As we've seen before, the bounding box shapes distribution shows that the 99% of the objects to detect is smaller than 18x18 pixels and the 90% is smaller than 8x8 pixels, before rescaling. After rescaling these values become: 99% smaller than 90x90 pixels and 90% smaller than 40x40 pixels.

backbone with a receptive field smaller than the objects to detect performs on this task. Indeed the receptive field of B16 backbone (the feature extraction part) is 16, with a stride of 4. So, given an input image of size 100x100px, the last layer feature maps have a size of 25x25.

In this model we used the following values for anchor sizes and ratios:

- anchor size: [4, 8, 16, 24, 32, 64]
- anchor ratio: [2: 1, 1: 2, 1:1]

This imply that, at training time, the model generates 6x3x25x25 anchors ground truth.

B44 In this model we used the same values for anchor sizes and ratios as in the B16:

- anchor size: [4, 8, 16, 24, 32, 64]
- anchor ratio: [2: 1, 1: 2, 1:1]

This imply that, at training time, the model generates 6x3x12x12 anchors ground truth.

VGG In this model, anchor sizes and ratios used are:

- anchor size: [128, 256, 512]
- anchor ratio: [2: 1, 1: 2, 1:1]

This imply that, at training time, the model generates 3x3x37x37 anchors ground truth.]

The hyperparameters listed in table ?? were mainly taken from the corresponding models' papers, in order to have a well-established benchmark, even though some of them were adjusted based on available resources (e.g. the batch size).

B16\_50\_100 B16\_20\_100 + FL B16\_20\_100 + BD

### 5.1 Data pre-processing

Given the image and the dataset described in ?? 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.

	Epochs	Batch size	Optimizer	Learning rate
<b>Baseline</b>	30	128	Adam	$1e - 3$
<b>BiDAF</b>	12	60	Adadelata	0.5
<b>BERT</b>	3	8	Adam	$5e - 5$
<b>DistilBERT</b>	3	16	Adam	$5e - 5$
<b>ELECTRA</b>	3	8	Adam	$5e - 5$

Table 5.1: Hyperparameters

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 than  $n$ . 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 calculated 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 [claran] but we discovered that patch normalization worked best.

Our code supports also augmentation on the fly, and in particular we implemented horizontal and vertical flipping, and the four  $90^\circ$  rotations. However we think this is not strictly necessary in this specific projects because shapes to predicts are very isomorphic with no intrinsic orientation, and so we did not use it in our experiments.

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 ???. 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  $n$ , the patch size.

The following are the class distribution for each set of patches:

Figure 5.1: Class distribution for 20\_100 set

Figure 5.2: Class distribution for 50\_100 set

Figure 5.3: Class distribution for 100\_600 set

## 5.2 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.[citazione per giustificare map] On the training set we track the four losses and classification accuracy of Detector.

## 5.3 Environment

The main third-party libraries on which the project is based on are Tensorflow [**tensorflow**] and Keras [**keras**], 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 [Inserisci pc Alice].

We tried also to use Google Colaboratory [**colab**], 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.

## 5.4 Results

Table ?? shows the results obtained on the validation set for each model, with the hyperparameters listed in table ??.

colonne: num epoche corrispondenti ai valori riportati + mAP + mPrec + mRecall

	Training		Validation		Test	
	F1 (%)	EM (%)	F1 (%)	EM (%)	F1 (%)	EM (%)
<b>Baseline</b>	36.21	21.27	37.19	26.47	38.30	27.15
<b>BiDAF</b>	63.42	43.27	68.40	55.31	71.15	60.09
<b>BERT</b>	73.68	56.14	80.28	67.80	83.17	74.17
<b>DistilBERT</b>	73.72	55.95	79.26	66.97	82.44	73.64
<b>ELECTRA</b>	<b>76.42</b>	<b>58.89</b>	<b>84.45</b>	<b>71.83</b>	<b>88.27</b>	<b>80.60</b>

Table 5.2: Best results

## 6 Analysis of results

For the results analysis we report here some examples of prediction for each best model for different patches. We chose patches with different characteristics, like number, dimension and brightness of objects.

Figure 6.1: Example of predictions from B16 model on the 20\_100 set.

Figure 6.2: Example of predictions from B16 model on the 50\_100 set.

Figure 6.3: Example of predictions from B44 model.

...

As we can see...



Figure 6.4: Example of predictions from VGG16 model.

## 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, based on the following ideas:

- objects are very small, so the receptive field of the network has to be chosen accurately [effective-rf];
- 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.

As described in ?? the best performing model is ... although its mAP score is not comparable to SoTA models.

– FARE RIASSUNTO DELLE DIFFERENZE TRA GLI ERRORI

The interesting this is that a simple model as B16 outperforms a complex one like VGG16. [CONTROLLARE QUI]

We didn't took 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 model 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 [fpn] in order to produce more accurate feature maps for smaller objects, or to implement the tweaks described in [frcnn-small-obj] 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.