Mine-RCNN

Loi Dario (1940849), Marincione Davide (1927757), Barda Benjamin (1805213)

Abstract—Real time object detection has recently been made possible due to steady state-of-the-art advancements in the field [1], [2], these methods propose the use of a Region Proposal Network to identify Regions of Interest (RoIs) in the image and correctly classify them, we aim to reproduce the architecture proposed by [2] applied to a novel environment, that of the popular sandbox Minecraft, both for the ease-of-collection of the required data and for a number of graphical properties possesed by the game that make such a complex problem more approachable in terms of computational resources, moreover, due to the novelty of the environment, we also train the entirety of the network from the ground up, having no pre-trained backbone at our disposal.

Index Terms—Object Detection, Convolutional Neural Network, Sandbox, Region Proposal, Real Time Detection

1 Introduction

Traditional visual recognition algorithms employ the use of algorithms such as SIFT or SURF [3], [4], to extract a feature descriptor from key points in the image, a set of sample images is therefore processed in order to obtain descriptors for each class, inference is then performed by extracting the descriptor from the input image and then comparing it with our samples, using a distance criterion to select the closest class.

While these traditional methods are mostly invariant to scale and translations, issues such as variability in the brightness or rotation of the target object are problematic, moreover, the computational cost of extracting features is prohibitively high, effectively preventing the possibility of deploying the model in a real-time context.

Our Approach: One way to reduce the cost in performance while gaining a higher expressive capacity for our model is to employ modern Deep Learning techniques to build our detector, our aim is therefore to design a scaled-back version of [2], stopping ourselves at the classification stage, the reason for this will be further clarified in the rest of the paper, our architecture[1] is composed of a FCNN [5], [6] Backbone, this is to keep the number of parameters as low as possible as well as allow the initial layer of our architecture to be applicable to images of any resolution, our Convolutional Backbone must be trained from the ground up in the task of simply classifying the classes that we intend to detect, the classifying layer is then removed from the model and we're then left with its feature map output which we then pre-process by *splashing* anchors through a sliding window approach.

After this, both feature maps and anchors are then passed to our region proposal layers, which in turn is composed of a twin ensemble of Neural Networks that perform Regression and Classification on the bounding boxes, the outcome is a set of *positive* and *negative* bounding boxes, the boxes that are classified as positive go through NMS [7] using Intersection over Union (IoU) [8] as a metric for suppression in order to reduce the amount of individual boxes that classify the same object.

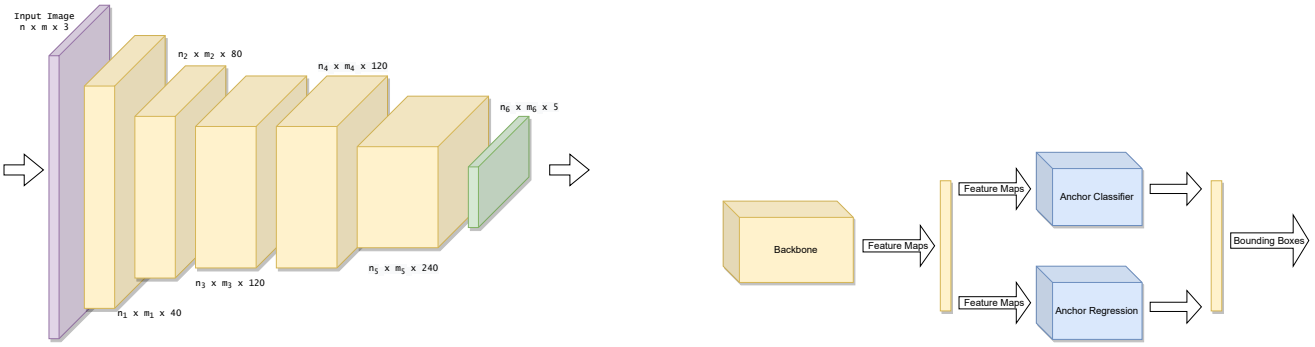
Our Environment: In order to keep our project in the realms of feasibility, especially considering the constraints imposed on our team in the realms of time and processing capabilities of our hardware, we chose to detect objects in a virtual environment, this is because the possibility of having complete control over the environment allows us to artificially create scenarios from which we can gather samples of a large quantity, for this reason, we chose the popular sandbox game Minecraft [9], other than the advantage of control over the environment, the game is well known for it's simplistic approach to Voxel Graphics, rendering the world as a set of blocks of cubic shape, the game also possesses a simplistic lightning model that simply shifts the brightness of blocks' textures when in the vicinity of a light source, thus being void of effects such as bloom, specular components on objects, reflections and other complex techniques that introduce sources of unexpected noise or variability in our samples, this, in turn, allows our model to work in a context that, while still posing a significant challenge, is of suitable difficulty to the scope of this project.

2 Method

2.1 The dataset

We started first by building the dataset. We recorded one-minutes long videos of minecraft using commercial screen captures softwares. We then loaded those shorts into python, using the OpenCV library, in order to sample one frame per second as we beleived would have given enough time for the next sampled frame to have significative diffrences with respect to the previous ones. We then downsampled the images in order to compress the size of the dataset in order to

1. see Figure 1 for a diagram of our architecture



(a) Our Backbone

(b) Our Network

Figure 1: Our FCNN backbone (a) employes a series of convolutions to obtain a feature map to pass to the rest of the network as displayed in (b), the feature maps from our Backbone are first pre-processed by a sliding-window approach that spreads our anchors over the image, this is then passed to a twin neural network, one part of the network performs classification, determining if each bounding box is containing an object or not, the other part performs regression on our bounding box in order to make it better fit an eventual target object.

Figure 2: A Screenshot of the world of Minecraft, note the simplistic graphics and the absence of Bloom (except the illusion thereof provided by the skybox) even though we are staring directly at the sun, note also the presence of a *pig*, one of the entities that we are going to detect in our model.

be able to share it with ease. To further reduce the problem we adopted for the final image a one-to-scale ratio, thus making the image squared.
At this time we opted to limit ourselves to only five classes we were aiming to classify: Zombies, Creepers, Pigs, Sheeps, Nothing.
The first version of the dataset was composed of images where in each frame were present multiple mobs of diffrent types at the same time. This turned out to be the wrong approach, since this made the training of the backbone network (see network implementation details) naturally overfitt considering the relative small dimension of our dataset. Having considered those implications we decided to have single mob or no mob per frame. The next step was labeling each sampled frames. We developed a simple but effective tool that allowed us to draw boundng boxes(BBox), and assign to each one of them a label corresponding to a class mentioned above. During this process we pruned images that we considered unfit to be part of the dataset (e.g. frames inside the game menu or outside of the game). After standardizing the coordinates of BBoxes we saved them into JSONs files. Having our JSONS files ready we group them into a single .dtst file for better integration with the PyTorch library, which is the one we decided to use for this project. From the sampled images we collected 3920 ,manually labeled, valid frames to which we applied three diffrent tranformations : one ColorJitter, and two Sharpness adjuster bringing the total amount of images in our dataset to roughly 8000 images for the final iteration, which we considered large and diverse enough for our purposes. Great consideration went also in deciding whether to include the raw images into the dataset itself or have them loaded at run-time. While making the file to be shared considerably heavier we decided to store the images in .dtst itself to make the training phase easier.

2.2 The models

Our architecture is divided into two main modules: The back-bone which we decided to call the Ziggurat (ZG), and the Region Proposal Network (RPN).

2.2.0.1 The backbone: Ziggurat: The ZG is a Fully convulutional neural network consisting of 6 layers as described in table 1. The sixth layer will be dropped after the pre-training phase. The purpose of this network is to extract feature maps from the input image for the RPN that follows. It is important then to pre-train this network so that it learns the general embedding in order to reduce the complexity of training the RPN. We decided to add dropout as it provides a usefull guard against overfitting [**?**], and again considering the dimensions of our dataset we decided to add it to each layer except the last one.
As for the activation function we decided to use MISH opposed to ReLU used in the original Faster-RCNN

Table 1: ZG

Layer	Kernel size	IN/OUT channels	padding	stride	Layer Structure
1	7x7	3 / 40	replicate(1)	2	BATCHNORM(3) + CONV + MISH + DROPOUT(.2) + BATCHNORM(40)
2	7x7	40 / 80	replicate(1)	2	CONV + MISH + DROPOUT(.2) + BATCHNORM(80)
3	3x3	80 / 120	replicate(1)	2	CONV + MISH + DROPOUT(.2) + BATCHNORM(80)
4	3x3	120/120	None	1	CONV + MISH + DROPOUT(.2) + BATCHNORM(120)
5	3x3	120/240	replicate(1)	2	CONV + MISH + DROPOUT(.2) + BATCHNORM(240)
6	1x1	240/5	None	1	CONV + BATCHNORM(5)

as it is already widely accepted as superior in terms of stability and accuracy. We initialized the wieghts using the kaiming method. [**?**]

2.2.0.2 Anchors: Before diving into the region proposal architechture, we need to pay extra attention to the anchors, which are the core of whole process. An anchor is nothing more than a rectangle to which is associated a center, a base size, and tranformation on both scale and ratio between it's sides. We then can define a set of anchors which consists of a base size, a center, and sets of different aspect ratios and scales. If we then denote with K the cardinality of the ratio set, and with H the cardinality of the scale set, the total number of anchors in the set of anchors is $A = K * C$.
For our implementation we used $0.25, 0.5, 1, 2$ for the scale transformation and $0.5, 1, 2$ for the ratio one using a base size of 40, thus having $A = 9$.
After processing the image through the ZG we obatin what is our base feature map (BFM) of size H, W that will be processed further by the RPN. On each feature of the BFM we apply a set of anchor as described above, centered on that feature. This allow us to easily map from the feature map to the orignial image by computer the total stride of the ZG up to the fifth layer. At tend of this process, which we like to call the "splash", we will have a total of $H * W * A$ anchors in total.

2.2.0.3 RPN: The rpn is the last part of our model. We went through many iterations, trying to find a solution that would do good in terms of both accuracy, performance and size. While the specific architechtures have varied a lot during the build of the project the core idea behind all of them stayed the same. After retreiving the BFM and splashed the anchors pass the BFM into an additional convolutional layer; We then have two siblings layers: One for classification and one for regression. The classification layer needs to predict a score of being an object or not for each splashed anchor based on the Intersection over union (IoU) with any ground truth box. The regression layer on the other hand tries to predict a set of offsets for each anchor that when applied to the corresponding one will give us our Region Of intrest (RoI).
As for the specific architechture we tried first to use only convolutional layers for both regression and classification, using 1x1 convolution over the BFM; while getting good results on the classification task, the regression proved to be complex for the small capacity of our network, so we decided for a linear model instead.
The architecture chosen is reported in table 2. For the classification we used a Sigmoid activation function as the scoring is given in the $0/1$ range.
A crucial aspect in this model is how to deal with overlapping anchors. It is easy to see that the amount of anchors for each image would be very large even for a small BFM (e.g. for a 19x19 BFM and A = 9 we would have around 80'000 anchors), moreover most of them would be classified as non object which as we will see do not contribute to the regression task at all. For those reasons we perform non maximum suppression (NMS) over the tranformed anchors, allowing us to prune overlapping anchors, keeping only the ones with the highest predicted score. Attention went also in dealing with out of image anchors. During training we decided to remove to ignore them completetly as they would introduce more complexity to an already difficult task, while during testing we just clip them to the image borders.

Table 2: RPN

Layer	Size IN / OUT	Layer Structure
Classification Layer	$C * H * W$ / 4332	Linear + Sigmoid
Regression Layer	$C * H * W$ / 4332 * 4	Linear

2.2.0.4 Training: We pre-trained ZG on a classification task over the 5 labels mentioned above, using just a portion of our dataset splitted in 60 - 20 - 20 for training validation and testing respectively using frames where only one mob, or less, of one kind was present.
For training we used Cross-Entropy loss function in conjuntion with AdamW optimizer using the AMSGrad variant of the algorithm with a fixed learning rate set to 0.0002.
Having our backbone pre-trained we cropped the last layer and attach the untrained RPN; worth to note that already at this point, even with no additional training the RoIs proposed were already somewhat close to the

real target.
For the loss function we implemented the one described in [2]:

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

where i are the indexes of the anchors in the image, p_i is the perdicted score for that anchor and p_i^* is a binary label : we assign 1 to those anchors that have an IoU with respect to any GT BBox greater than 0.45, and 0 to those with IoU smaller than 0.05 we ignore the anchors that falls in between. For L_{cls} we use a BCE loss over the two classes of being an object or not.
On the other hand t_i and t_i^* are parametrized vectors containing the coordinates of the predicted positive anchors, and that of the ground thruth associated to that anchor respectively.λ is a normalization constant that we set to 10 to prevent that the classification would overwhelm the regression in the objective function. [1] suggest furter normalization but we decided not to implent it as it would not have changed the curvature of the loss. The coordinates are parametrized as described in [2] :

$$\begin{aligned} t_x = (x - x_a)/w_a \quad t_y = (y - y_a)/h_a \\ t_w = log(w/w_a) \quad t_h = log(h/h_a) \\ t_x^* = (x^* - x_a)/w_a \quad t_x^* = (y * - y_a)/h_a \\ t_w^* = log(w * /w_a) \quad t_h^* = log(h^*/h_a) \end{aligned} \tag{1}$$

Where x, y, w, h rappresent the center's coordinates, height, and width of the predicted box, anchor box, and ground thruth (e.g. x, x_a, x^*).
We decided to split the dataset into 80% training and 20% validation; We trained the network using SGD as suggested in [2] with a fixed learning rate of 10^-5 and a momentum of 0.9

3 Results

Given the inexperience, the difficulty of the task and the (inadequate) hardware at hand, we think to have reached positive results, the system shows signs of being able to recognize traits of the mobs we've trained it on, even though sometimes they are just cases of pareidolia. Before giving some manner of statistics over its capabilities we would like to point out an important decision: that of the threshold for deciding whether, given a score, the anchor for which it is related to is actually a positive one or not. To decide this fundamental hyperparameter we resolved in sampling five hundred (500) images from our training dataset and, after letting the system apply non-maximum suppression, recovering the scores for all the remaining anchors and pairing them to their true labels. Once this preprocessing was done we plotted the ROC:

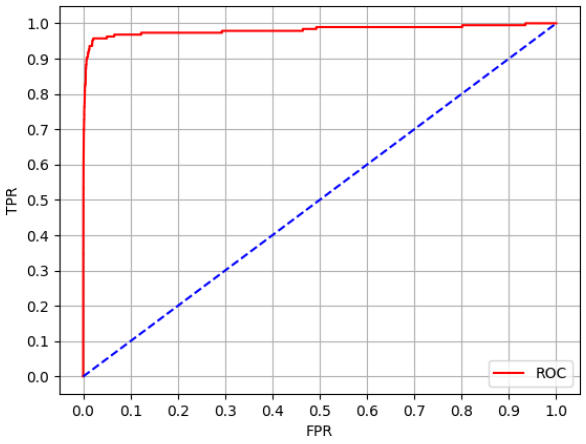


Figure 3: Linear RPN's ROC

And a graph, showing the decrease of the *true positive ratio* (TPR) and of the *false positive ratio* (FPR) as the threshold increased, as following:

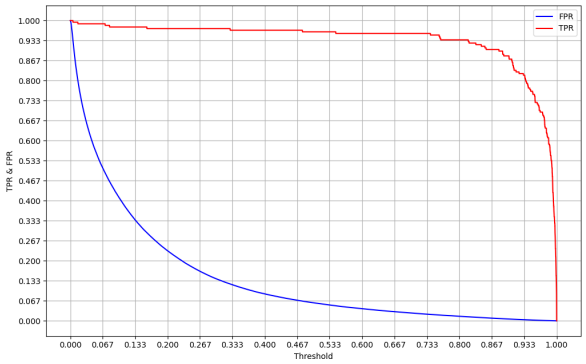


Figure 4: Linear RPN's TPR & FPR as threshold changes

From both these graphs we notice our network is indeed able to separate positive and negative anchors with very clear-cut definition. Furthermore, we decided that a good compromise between (TPR) and (FPR) could be achieved by choosing a threshold between 0.8 and 0.9: since we didn't want to remove positive anchors too much we put ours at 0.81. We posit that, to avoid false positives as much as possible, 0.9 should work fine too.

After this, in our opinion, fundamental decision was made: we resolved to open Pandora's box and create a test set of around 250 images with multiple mobs in the same shot. Unfortunately, as said previously, we didn't develop our network up to classification of the boxes, and thus a full confusion matrix of those is out of the question. At any rate we show the misclassification table for the anchors (after non-maximum-suppression) scored by our network over the whole test set:

Telling us that, indeed, our threshold works, since the TPR is 0.78 and the FPR is 0.03. We would like to point out that, since the labelling of the anchors

	Label pos.	Label neg.
Pred. pos.	62	3545
Pred. neg.	17	127789

Table 3: Misclassification matrix

is itself hyperparameter driven (given the choice of anchors and that of the IoU thresholds used to label them) and that we think to have chosen a combination of these parameters that biases the labels towards the non-object side and, furthermore, given this label by itself already dominates the distribution: it is only normal for the number of positive labels to be so low in the set. If we look at the actual images with the positive region proposals added, we'll appreciate much more positive-looking (that may not actually be labelled as positive) proposals than the anchor labelling would actually suggest.

As said above, we also tried fitting a fully convolutional version of the RPN, unfortunately though the results were pretty mediocre, its ROC pretty clearly showing it:

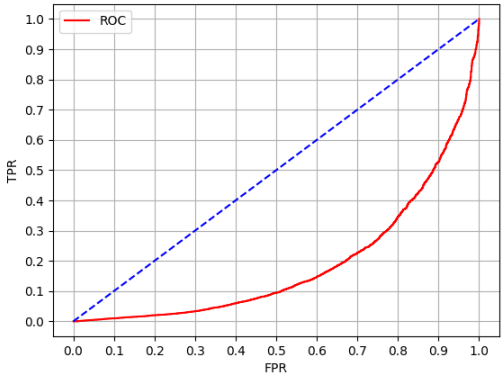


Figure 5: Fully convolutional RPN's ROC

Indeed this version is unable to sufficiently separate positive and negative samples, leading to unacceptable predictions…a shame, since it would've been several magnitudes lighter (weighing just 3.5 mbs) and faster (both to execute and to train) than its linear counterpart.

Let's now give a look at some proposals formulated by our network:

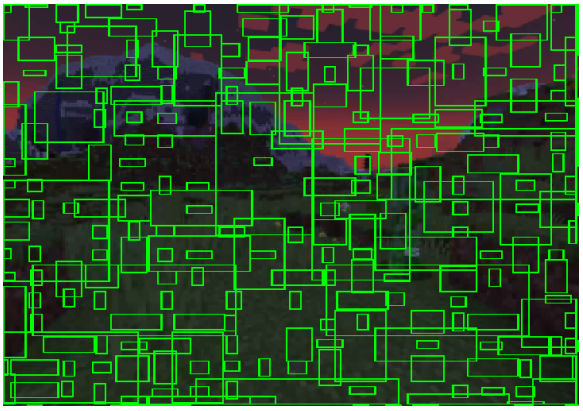


Figure 6: An *interesting* sunset

Before looking at the good stuff, we would like to show the bad/interesting stuff. TODO, Dario.

Figure 7: A creeper in his natural habitat

In this case we can see an excellent result. Alas, one may see there's part of a pig in the bottom-left corner; we don't think that to be an error, to recognize a speck so small and without features would be pretty difficult even for state-of-the-art systems.

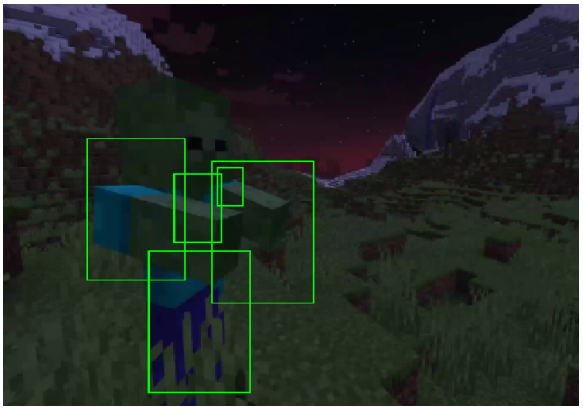


Figure 8: A zombie, piece-wise

Here's another good, albeit different, result: in our opinion (and as how the problem is currently formulated, mathematically speaking) a box doesn't have to recognize an object in its entirety, that's why we accept this sundered recognition as correct. As you may have realized, it is because of this type of proposals that 3 holds so many false positives, during the automatic labeling of anchors it is more than possible that the inner anchors of this zombie are classified as negative.

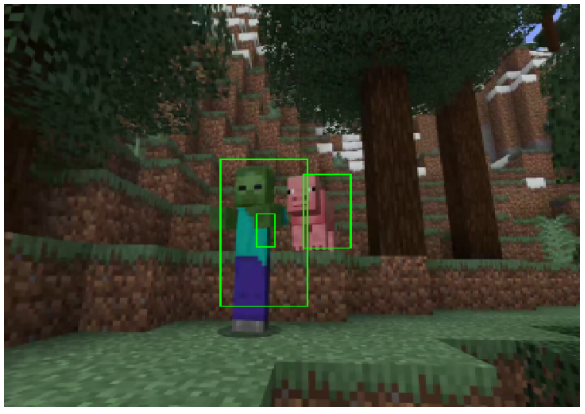


Figure 9: Zombie and pig

In this case we see another sufficiently nice result…let's end with two other interesting proposals.

Figure 10: Pareidolia in the background

Leaving the interesting proposals for the pig, the creeper and the sheep in the image, we can see a pine in the upper left corner which has been spot-on proposed by our network: we can't say for certain, but we assume this to be a case of misclassification given by pareidolia, after all to a fast glance that may *seem* like a zombie.

Figure 11: Spot the sheep

The network gave no proposals for this image, but unfortunately there *is* a sheep in there; but it is probably too green and not in the foreground to be detected.

In light of these agreeable and interesting results, we think to have reached our goals, not without hiccups nor errors that is. But the network seems to be, generally, well behaved and relatively correct.

References

[1] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

[2] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[3] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

[4] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346 – 359, 2008, similarity Matching in Computer Vision and Multimedia. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314207001555

[5] Q. Chen, J. Xu, and V. Koltun, "Fast image processing with fully-convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[7] A. Neubeck and L. Van Gool, "Efficient non-maximum suppression," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 3. IEEE, 2006, pp. 850–855.

[8] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," 2019. [Online]. Available: https://arxiv.org/abs/1902.09630

[9] Mojang, "Minecraft," https://www.minecraft.net/en-us, Nov. 2011.