Mine-RCNN

Loi Dario, Marincione Davide, Barda Benjamin

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 Method

1.1 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 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 runtime. While making the file to be shared considerably heavier we decided to store the images in .dtst itself to make the training phase easier.

1.2 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).

1.2.0.1 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 [3], 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 as it is already widely accepted as superior in terms of stability and accuracy. We initialized the wieghts using the kaiming method. [4]
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.

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)

1.2.0.2 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 $A = 40$

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] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015. [Online]. Available: https://arxiv.org/abs/1502.01852