Deep Learning and balls detection

Martin Franceschi, Romain Perrone
Group 1

November 29, 2020

I. Introduction

To gain a complete understanding of an image, we should not only concentrate on classifying different images, but also try to precisely estimate the location of the objects contained in each image. This task is referred to as object detection. This field has been popular among researchers in recent years. Object detection can provide valuable information for the semantic understanding of images and can be used in a wide variety of settings: motion tracking, obstacle avoidance, ... However slight modifications in the environment such as changes in lighting, perspective, contrast and possible occlusions make it difficult to perfectly accomplish this task.

In this paper, we will explore three different object detection techniques and compare their performance:

1. Computer Vision : One of the main benefits of using computer vision is that it does not require any training. The performance on a very small dataset (100 images) is almost identical to the performance on a very large dataset (100k+ images). In addition, some computer vision techniques can have a very good performance on unseen samples. Deep Learning on the other hand usually performs poorly on images that are very different from the ones found in the training set.

2. Transfer Learning using ResNet-18 : Transfer learning makes use of a pre-trained model to solve a new problem. Here we use ResNet-18, a convolutional neural network with 18 layers, trained on the ImageNet data set. Transfer learning is a very popular option in deep learning because it can train deep neural networks with very little data. The main advantages of transfer learning are saving training time, better performance, and not needing a lot of data.

3. Convolutional Neural Networks : In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. Here we will use a simple Convolutional net similar to the one introduced by LeCun et al. in 1998 to identify the color of the balls present in the image.

In addition we will also explore the topic of **Bounding box detection**. We will devise a single approach for evaluating the position of the balls in the image, or in other words *bounding boxes regression*: transfer learning. We will extend the work we have done using ResNet-18 so that the network is able to give us both information: the color and position of each ball in the image.

II. Methods

Our dataset is composed of **21 000 images**, that were initially generated by a computer program.

Among these images: 16 800 **(80%)** were assigned to the **training set** and 4 200 **(20%)** were assigned to the **validation set**.

There are **4 different kinds of background** with different texture models. On every image, 3 balls are randomly positioned. Their color is uniquely defined among the following: blue, cyan, green, lime, magenta, orange, purple, red, yellow.

From a machine perspective, the blue squares from the third background image can easily be confused with a blue ball. So the problem of identifying the balls' color is not as easy as it may seem.

In the following sections, we will explain the

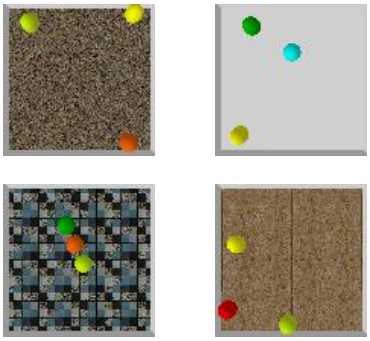


Figure 1: *Images used in the training set and in the validation set*

different methods we used to identify the balls.

i. Computer Vision

The process is divided into two parts.

In the first part, we apply **Histogram Backprojection** for each color: blue, cyan, green, lime, magenta, orange, purple, red, yellow. That gives us a **probability image**.

Histogram Backprojection was first proposed by **Michael J. Swain , Dana H. Ballard** in their paper *Indexing via color histograms*. It is used for image segmentation or finding objects of interest in an image. To perform this Backprojection, we generate the histogram of a target image containing our object of interest (a blue ball, a green ball, ...). We then "back-project" this histogram over our sample image where we need to find the object, that is: we calculate the probability of every pixel belonging to the target image. The resulting output is then thresholded to obtain **a black and white image.**

From this probability image, we apply a morphological transformations called opening. This corresponds to an erosion followed by a dilation. This morphological transformation helps us remove the background noise and stabilize the image.

We then use the **circle Hough Transform**, a basic feature extraction technique, to detect circles in this image. We narrow down our search by using a specific radius range. Finally,

Figure 2: *Opening transformation on the letter "j"*

we have an $(x, y, r, color)$ vector where (x, y) corresponds to the center of the circle, r to the radius and $color$ to the color of the ball that was detected.

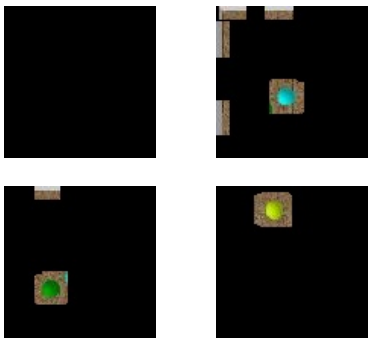


Figure 3: *Histogram Backprojection on the following colors (a) blue (b) cyan (c) green (d) lime*

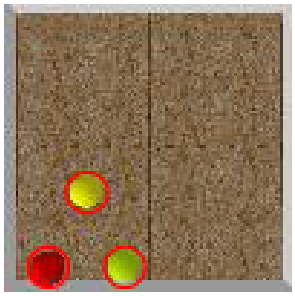


Figure 4: *Hough Transform detected 3 circles*

One of the key benefits of using computer vision is that the programme doesn't require any training. One of the main downside however is that it takes an average 8.7s to run. In practice, we can still have a very good performance. We can bring down that time to 0,96s / image using parallel processing, as each color can be processed independently.

ii. Transfer Learning

We have chosen to use ResNet-18 after it has been trained on ImageNet. What we will explain next is how we removed the final layer and replaced it with something that fulfills our requirements. The final layer appeared to be a fully-connected layer with 512 neurons as input, which is something interesting for us because the layers we will plug in there will have 512 different information to work on.

The actual data representation for color detection is a binary vector of 9 rows, one row per color, and the value of a row is 1 if the associated color is present and 0 if not. It is then expected that exactly three colors are detected. To achieve such result, we simply used a linear layer with 9 values as output, with each value being a decimal number between 0 and 1.

In order to exploit such output in which we need only 3 detected colors. We have two possibilities:

- We can use a threshold. It needs to be a user-defined hyper-parameter and it does not guarantee that three values will be outputted, but in some way it can be quite relevant because we are sure that the outputted colors are all of the colors the network is sure are present.
- We can simply take the top 3 values or, in a more correct language, take the three higher values from the 9 ones. It is simpler to use in practice (among others, for computing the accuracy) and we are sure that we will only work with three values. The downside is that the network may have as top 3 values $(0.95, 0.93, 0.27)$ for example, and the third one is obviously much less relevant and interesting to deal with.

We selected the "top 3" method for accuracy and the binary cross-entropy (after computing a sigmoïd) for the loss function.

Here is the workflow scheme we used in this project, along with the PyTorch library:

1. Load the DataSet.
    - We are using the cache option: it loads all files in memory at start, so that we do not have to open and close files at each iteration or epoch.
    - Another optimization we thought of but implemented quite late: when initializing the DataSet instance, we run the data through the 17 first layers of ResNet-18 and only remember the 512 features as input of the 17th layer. These values will not change over iterations or epochs (the reason is below) so that we don't have to compute them again.
2. Fetch a pre-trained instance of ResNet-18. Edit its last module (or output layer) to some item which fulfills our needs. Freeze all other layers by setting their flag *requires grad* to false, so that they do not evolve during training. For the single purpose of classification, the last module becomes a 512-to-9 linear layer.
3. Set a cost function: BCE with sigmoïd.
4. Set an optimizer: we tried SGD + a 7-epochs scheduler and an Adam optimizer, and finally adopted Adam.
5. Set an accuracy function, as explained above.
6. Finally, do the training and analyze the results.

We ran our transfer learning script on 3000 training images and 1000 validation images because it takes some very long time. We chose to do 20 epochs, which is an intermediate value between 11 (minimal epoch number according to some articles) and 25 (epoch number mostly relevant to explore as much data as potentially interesting).

iii. Convolutional Neural Networks

In this section, we describe the CNN we used to predict the color of the balls in the image.

We found that a simple Convolutional net, derived from the one introduced by LeCun et al. in 1998 could achieve very good results.

Table 1: *Our CNN model*

Layer	Parameters
Conv2d(3,50)	Kernel size (3,1)
ReLU	-
Conv2d(50,100)	Kernel size (3,1)
ReLU	-
Maxpool2d	-
Dropout	0.25
Fully connected Layer	(52900, 64)
ReLU	-
Dropout	0.5
Fully connected Layer	(64, 9)

We used *BCEWithLogitsLoss* as our loss function. This loss combines a Sigmoid layer and the BCELoss in one single class.

The performance was evaluated by thresholding the output vector. All values superior or equal to 0.5 were set to True, while the rest was set to False.

To train the model, we used Adam with a learning rate of 0.002. We decreased the learning rate by a factor of 0.9216 every step.

Figure 5: *Model loss at each epoch*

We had to reset the Adam learning rate after the 60th and the 90th training epoch as the *training* loss wasn't decreasing any further.

With hindsight, we should have decreased the learning rate every two steps by a factor of 0.95 or more. It is evident from *Figure 5* that decreasing the learning rate too fast wasn't a good strategy.

Results : We were able to achieve **99,4%** accuracy on the validation set. The CNN surpassed by far the performance of the Transfer Learning and Computer Vision techniques we presented previously.

iv. Transfer learning for bounding boxes regression

Here we are about to extend the approach used with transfer learning from a pre-trained ResNet-18. We are changing requirements: now the output we need is actually composed of two different and (partially) independent information. We then decided to use a fork logic: we duplicate the flow of 512 information from the 17 first layers and redirect both flows to two layers or chains of layers. This is quite convenient because it allows us to apply different methodologies and operations on one chain of layers without it to affect what happens at the other chain.

The actual data representation for bounding boxes is (to keep it simple) a matrix which contains, for each detect ball, the coordinates of the top-left point of the bounding box and the same for bottom-right[1].

We chose to use some linear layers for that task. We ran attempts with a single linear layer or several ones, but putting several layers do not seem to affect the accuracy or loss values a lot so we decided not to investigate more. The loss function selected is Mean Squared Error, because we are dealing with distances.

For the accuracy, here are our three possibilities. Note that we skip all cases where the expected values are zeroes.

- Count a bounding box as correct if both output points are below some max euclidean distance of their respective expected points.
- Count a bounding box as correct if the center of the output box is below some max euclidean distance with the expected box center. It is relevant because boxes are expected to be some fixed-size squares,

[1]The actual data format is a $9 * 4$ matrix with one line per color. If the color is missing then the row is zeroed and if the color is present the values are (top-left X, top-left Y, bottom-right X, bottom-right Y).

but we might neglect cases where the output box has the correct center but wrong actual points.
- Count a bounding box as correct if the ratio (intersection over union) between both box surfaces is under a certain threshold value.

We decided to use the first method in which we need to compute two euclidean distance.

III. Results

i. Computer Vision

Table 2: *Ball detection performance*

Test samples	Precision	Recall
100	0.87	0.93
1 000	0.85	0.91
10 000	0.83	0.90

The confusion matrices for N=100 test samples are presented below:

Table 3: *Confusion Matrix (Red)*

Prediction \Real	Red	¬*Red*
Red	60	0
¬ Red	10	30

Precision: 100%; Recall: 80%

Table 4: *Confusion Matrix (Green)*

Prediction\Real	Green	¬ Green
Green	50	0
¬ Green	0	50

Precision: 100% *; Recall:* 100%

We observe a drop in performance for the following colors: Purple, Magenta, Cyan and Lime.

Our analysis suggests that the *Histogram Back-projection* technique fails to discriminate

Table 5: *Confusion Matrix (Blue)*

Prediction\Real	Blue	¬ Blue
Blue	90	0
¬ Blue	0	10

Precision: 100% *; Recall:* 100%

Table 6: *Confusion Matrix (Yellow)*

Prediction\Real	Yellow	¬ Yellow
Yellow	60	0
¬ Yellow	20	20

Precision: 100% *; Recall:* 75%

Table 7: *Confusion Matrix (Lime)*

Prediction\Real	Lime	¬ Lime
Lime	70	10
¬ Lime	20	0

Precision: 87,5% *; Recall:* 77%

Table 8: *Confusion Matrix (Purple)*

Prediction\Real	Purple	¬ Purple
Purple	80	20
¬ Purple	0	0

Precision: 80% *; Recall:* 100%

Table 9: *Confusion Matrix (Orange)*

Prediction\Real	Orange	¬ Orange
Orange	50	0
¬ Orange	0	50

Precision: 100% *; Recall:* 100%

Table 10: *Confusion Matrix (Cyan)*

Prediction\Real	Cyan	¬ Cyan
Cyan	40	40
¬ Cyan	0	20

Precision: 50% *; Recall:* 100%

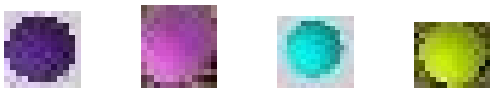
Table 11: *Confusion Matrix (Magenta)*

Prediction\Real	Magenta	¬ Magenta
Magenta	50	40
¬ Magenta	0	10

Precision: 55,5% *; Recall:* 100%

between certain colors. Purple and Magenta for instance are very similar. And so are cyan and lime.

This might explain why certain balls (Red, Green, Blue, ...) have such a good detection rate, while others tend to be recognized incorrectly.



ii. Transfer learning

During training, the loss value per epoch is plotted below. Using the GPU, training the model on approx. 10% of the dataset for 20 epochs took 27 minutes. The results seems to oscillate, which indicates that we should have chosen a smaller learning rate for example.

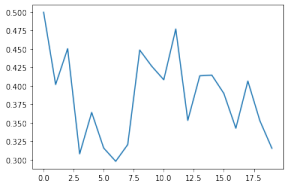


Figure 6: *Loss value per epoch during the training of transfer learning classifier.*

The ball detection performance is displayed below. We did not have enough computational power to run on ten thousand samples.

The confusion matrices for N=1000 test samples are presented below:

The performances are in general less good than the first method, though most colors have

Table 12: *Ball detection performance*

Test samples	Precision	Recall
100	80%	80%
1 000	78%	78%
10 000	N/A	N/A

Table 13: *Confusion Matrix (Red)*

Prediction \Real	Red	¬*Red*
Red	276	15
¬ Red	35	674

Precision: 89%; Recall: 95%

Table 14: *Confusion Matrix (Green)*

Prediction\Real	Green	¬ Green
Green	226	62
¬ Green	97	615

Precision: 70% *; Recall:* 78%

Table 15: *Confusion Matrix (Blue)*

Prediction\Real	Blue	¬ Blue
Blue	153	8
¬ Blue	172	667

Precision: 47% *; Recall:* 84%

Table 16: *Confusion Matrix (Yellow)*

Prediction\Real	Yellow	¬ Yellow
Yellow	262	48
¬ Yellow	64	626

Precision: 80% *; Recall:* 85%

Table 17: *Confusion Matrix (Lime)*

Prediction\Real	Lime	¬ Lime
Lime	350	178
¬ Lime	14	458

Precision: 96% *; Recall:* 66%

Table 18: *Confusion Matrix (Purple)*

Prediction\Real	Purple	¬ Purple
Purple	164	34
¬ Purple	176	626

Precision: 48% *; Recall:* 83%

Table 19: *Confusion Matrix (Orange)*

Prediction\Real	Orange	¬ Orange
Orange	306	7
¬ Orange	34	653

Precision: 90% *; Recall:* 98%

Table 20: *Confusion Matrix (Cyan)*

Prediction\Real	Cyan	¬ Cyan
Cyan	266	53
¬ Cyan	69	612

Precision: 79% *; Recall:* 83%

Table 21: *Confusion Matrix (Magenta)*

Prediction\Real	Magenta	¬ Magenta
Magenta	328	264
¬ Magenta	8	400

Precision: 98% *; Recall:* 55%

a precision of 80%+ which seems quite acceptable but not enough for a real-world exploitation.

We observe a huge precision drop for cyan and purple. We believe that the dark and paved background causes issues because it is harder to detect some item. That hypothesis might be wrong if the ResNet-18 layers include layers dedicated to ignore such backgrounds.

We can see that, according to results, the model does not mismatches the same colors as in Computer Vision, which lead us to say that it is less prone to mismatch similar colors.

iii. Convolutional Neural Network

The confusion matrices for N=4200 validation samples are presented below.

Table 22: *Performance on the validation set after X epoch*

Training epoch	Precision	Recall
0 - 30	0.80	0.79
30 - 60	0.83	0.84
60 - 90	0.97	0.97
90 - 120	0.99	0.99

Table 23: *Confusion Matrix (Red)*

Prediction \Real	Red	¬*Red*
Red	11337	0
¬ Red	0	5463

Precision: 100%; Recall: 100%

Table 24: *Confusion Matrix (Green)*

Prediction\Real	Green	¬ Green
Green	11175	0
¬ Green	0	5625

Precision: 100% *; Recall:* 100%

Table 25: *Confusion Matrix (Blue)*

Prediction\Real	Blue	¬ Blue
Blue	11335	0
¬ Blue	0	5465

Precision: 100% *; Recall:* 100%

Table 26: *Confusion Matrix (Yellow)*

Prediction\Real	Yellow	¬ Yellow
Yellow	11205	381
¬ Yellow	30	5184

Precision: 96,7% *; Recall:* 99,7%

Table 27: *Confusion Matrix (Lime)*

Prediction\Real	Lime	¬ Lime
Lime	11041	0
¬ Lime	12	5728

Precision: 100% *; Recall:* 99,8%

Table 28: *Confusion Matrix (Purple)*

Prediction\Real	Purple	¬ Purple
Purple	11055	175
¬ Purple	229	5341

Precision: 98,4% *; Recall:* 97,9%

Table 29: *Confusion Matrix (Orange)*

Prediction\Real	Orange	¬ Orange
Orange	11190	0
¬ Orange	0	5610

Precision: 100% *; Recall:* 100%

Table 30: *Confusion Matrix (Cyan)*

Prediction\Real	Cyan	¬ Cyan
Cyan	11081	40
¬ Cyan	0	5719

Precision: 100% *; Recall:* 100%

Table 31: *Confusion Matrix (Magenta)*

Prediction\Real	Magenta	¬ Magenta
Magenta	11110	0
¬ Magenta	0	5690

Precision: 100% *; Recall:* 100%

Using a GPU, it took us an average of 30 minutes to train the CNN through 30 epochs.

Our approach was to "babysit" one single model, because we lacked the resources and computing power necessary to run multiple models (initialized with different hyperparameters) in parallel.

We had to reset the Adam learning rate after the 60th and the 90th training epoch as the *training* loss wasn't decreasing any further.

The CNN performed very well on the validation set, with a 99,4% accuracy after the 120th training epoch.

The model also generalizes well from the training set to the validation set. We believe this is a result of the dropout layers we used to prevent overfitting.

Conclusion : The model has an excellent performance and requires very little computation time after training. We believe it would perform well in a controlled environment.

However, modifications in the environment such as changes in lighting, perspective, contrast and possible occlusions may have a negative impact on the performance of the model.

iv. Transfer learning for bounding boxes regression

During training, the loss value per epoch is plotted below.

The accuracy values were unfortunately extremely low, at such a point that they are simply not usable. We then decided not to showcase the results. The validation accuracy values are all within the interval $[0.1\%, 2.1\%]$.

As an example, two visualizations are shown in the following images. We can see that nei-

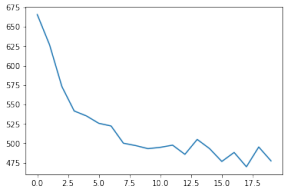


Figure 7: *Loss value per epoch during the training of transfer learning classifier.*

ther the position or the size of the boxes are accurate.

Figure 8: *Bounding box illustration.*

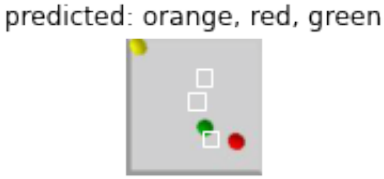


Figure 9: *Bounding box illustration.*

IV. Discussion

In this paper, we've explored three different object detection techniques and compared their performance.

1. Computer vision. This technique performed relatively well with an **83%** accuracy on the validation set.

Our analysis suggests that the *Histogram Back-projection* technique fails to discriminate between certain colors. So we won't be able to improve the accuracy with this technique.

Once again one of the main benefits of using computer vision is that it does not require any training. The performance on a very small dataset (100 images) is almost identical to the performance on a very large dataset (100k+ images).

The performance of our Computer Vision program is as good as the performance of our CNN after the 60th training epoch. (83%)

2. Transfer Learning using ResNet-18. Transfer learning makes use of a pre-trained model to solve a new problem. Here we used ResNet-18, a convolutional neural network with 18 layers, trained on the ImageNet data set. Transfer learning is a very popular option in deep learning because it can train deep neural networks with very little data. The main advantages of transfer learning are saving training time, better performance, and not needing a lot of data.

However, training the last layer of ResNet-18 doesn't perform that well. The loss function fluctuates a lot, and doesn't go down as expected. Resnet has been trained on the ImageNet dataset and as such, is able to recognize birds, dogs, cars, ... and many other objects ; our images are very different from the ones present in the Image Net dataset which might explain the gap in performance. That being said, an accuracy of 80% seems quite good knowing the downsides of our approach.

3. Convolutional Neural Networks. In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. Here we used a simple Convolutional net similar to the one introduced by LeCun et al. in 1998 to identify the color of the balls present in the image.

The model had an excellent performance (99,4%) and **was able to discriminate between similar colors** such as purple and magenta, and cyan and lime.

We were able to achieve near human-level performance on the validation set for the ball detection task.

Transfer learning for bounding boxes regression. As shown in the Results part of this

document, we could not manage to make that part to work correctly. Especially the loss value remains always extremely high, which we tend to say is not normal. It might be a bias issue. The accuracy is also extremely low.

The following actions could be taken to enhance our model:

- Tweaking hyper-parameters: try different (bigger) initial learning rates, change the optimizer...
- Finding a way to optimize
- Using and adding more complex layers: dropout, bigger linear layers...
- Change the accuracy function, or maybe even dynamically change the accuracy's settings (mainly the threshold value) through epochs.

We could not experiment as much as we intended to because the training time takes a very long time, even on a small part of the original dataset.