# O where the party at?

**Jules Belveze**
s182291

## 1 Introduction

This report aims to detail the creation process of a convolutional neural network capable of identifying digits in an image. To do so we have been through multiple steps. First of all, we trained a convolutional neural network that can classify single digits. Then we used this network with a convolutional implementation of the sliding window algorithm in order to detect bounding boxes around digits. We finally kept only the best bounding boxes by using NMS (Rasmus Rothe and Gool 2015) and IoU.

## 2 Background

We have already been through most of the fundamental notions in the *Hotdog / not hotdog* report, but we will describe the remaining used materials.

### 2.1 Convolutional implementation of sliding window

The idea here is to train a convolutional network to detect digits within an image and use windows of different sizes that we slide on top of it. Then for each window we perform a classification. To do so we need to switch the fully connected layers into convolutional layers, as shown in Fig.1.

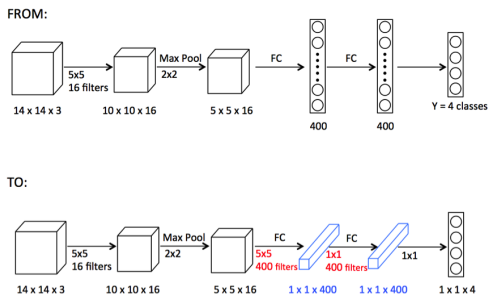


Figure 1: Example of transformation of fully connected layers into convolutional ones

If our test image is of dimension $16*16*3$ performing the sliding window would leave to create 4 different windows of size $14 * 14 * 3$ out of the original test image and run each one through the network.

### 2.2 Intersection-over-union

Intersection over union is a way to ensure that object detection is performing well. It is given by the following formula:

$$IoU(A, B) = \frac{Area(A \cap B)}{Area(A \cup B}$$

where $A$ and $B$ are two bounding boxes. The higher the $IoU$ is the more accurate we are. To prevent from wrongly detected objects we only consider the ones such that:

$$IoU(A, B) < 0.5$$

### 2.3 Non-maximum suppression

In order to avoid detecting the same object multiple times we will perform non-maximum suppression. In fact, when multiple bounding boxes overlap and tend to detect the same object NMS will only keep the one with the highest probability and fire the other ones.

## 3 Methods

### 3.1 Datasets and preprocessing

To train our convolutional neural network to classify single digits it was necessary to fed it with both images containing single digit and images without digits. Then we labeled each image by the digit it contains and by $10$ if there were none. For the numbers we used the SVHN dataset provided by *PyTorch* where images are cropped and only contain one single number. For the non-digit data, we used the CIFAR10 dataset. Then we merged and shuffled those two datasets into one training set.
We have tried multiple data transformations in order to investigate which ones lead to best detection. Once our network is trained on single digit classification, we test it on bigger images, coming from the SVHN dataset, in order to check if it can detect digits or not.

### 3.2 Neural network

As described in section 2.1 we implemented a convolutional sliding window. Initially our neural network was composed of 4 convolutional layers and 3 fully connected ones. We then modify the last 3 into convolutional layers to be able to use sliding windows.
We inserted a batch normalization layer after the first layer

in order to speed up learning by ensuring there is no activation that has gone in extreme values. In addition, we used two dropout layers to prevent from overfitting.
Due to the fact that our classes are present in different proportions we use a weighted loss function to penalise the majority classes.

### 3.3 NMS and IoU

Once we have passed an image through the model we need to extract positions where the network has identified digits and remove the ones labeled as "background". We then only keep the best bounding boxes by implementing our own IoU and NMS. We use a score threshold of $0.7$ and an IoU threshold of $0.5$

### 3.4 Digits detection

In order to detect digits that might not be contained in $32*32$ square we perform multi-scale testing. That is, for a given image we resize it in different manners (expansion or shrinkage) and perform sliding window on this new batch of images. Then we scale back the found bounding boxes in the initial image. Let say we have an image of size $300 * 300$ that we shrink to get an image of size $200 * 200$. We use our sliding window of size $32 * 32$ on it. Once a box is found it is resized in the original dimension of the image. That is:

$$32 * \frac{300}{200} = 48$$

meaning that we will have a bounding box of size $48 * 48$ in our first image.

## 4 Results

As described in section 3.1 we tried to preprocess our data in different manners to find which one leads to best identifications. We have trained two models one on gray scale and the other one on RGB images. It turns out that they perform digit classification with almost the same accuracy ($\sim 88\%$). We then use them to identify digits in our second set of images.
We can see in Fig.2 the bounding boxes for the two different transformations, before performing non-maximum suppression. As we can observe, the identification seems to perform

(a) RGB (b) Gray scale

Figure 2: Bounding boxes for different transformations

better using gray scale. In fact, it is messier using RGB, the network tends to identify more nonexistent digits. We will keep the first one for further analysis.

We then color the boxes borders with different colors in order to evaluate how accurate is the detector. In Fig.3 we can see the bounding boxes detected by the network using the single-scale method. The above figure reveals weaknesses of

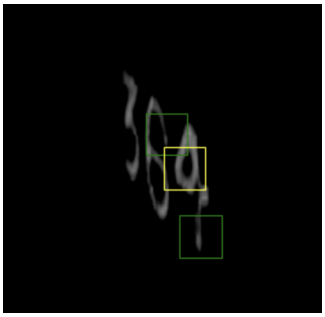

Figure 3: Digit detection using the single-scale method. Green boxes are for the digit 1 and yellow for 0.

the model. It is identifying ones (green boxes) where there are straight lines and 0 (yellow box) where there are curves. This happens even with a score threshold of $0.9$.
In order to try to solve that problem we implement a multi-scale version of our digit detector. We generate 10 resized images of different ratios from our initial one. Larger images are cropped from the center, smaller ones are padded.

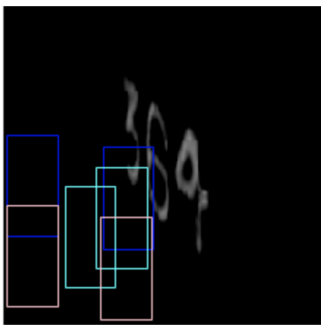

Figure 4: Digit detection using the multi-scale method. Blue boxes are for the digit 2, pink for 3 and cyan for 6.

As Fig.4 is depicting, the digit location using multi-scaling is performed poorly. In fact, we are detecting 2 correct digits present in the picture out of 3, but the bounding boxes locations are meaningless. The problem might come from the way we compute the bounding boxes coordinates because it looks like the network detects the coherent digits and the boxes fit the numbers closely.

## 5 Conclusion

It turns out that even if our classification model is accurate, it fails at locating digits on an image. However we have seen that the multi-scale method can identify digits better than the single-scale one which is mostly detecting curves and lines. For further work we could try to implement the YOLO (Joseph Redmon ) algorithm which performs object detection on chunks of images after diving an input one into grids.

## References

Joseph Redmon, A. F. Yolov3: An incremental improvement.

Rasmus Rothe, M. G., and Gool, L. V. 2015. Non-maximum suppression for object detection by passing messages between windows. *Lecture Notes in Computer Science 9003*.