COS 429 Project: Object Recognition in Pixel Art

Tam, Howard htam@princeton.edu
Lazzeretti, Ludovico ll9@princeton.edu

January 16, 2017

Contents

1. Introduction

The goal of the project is to learn a Convolutional Neural Net (CNN) that accurately recognizes objects in art pieces. Specifically, we aim to recognize human characters in pixel art pieces, each called a Pixorama, created by the German pixel art group Eboy. The Pixoramas vary in density and complexity, presenting a non-trivial object recognition task. Figure 1.1 shows an example of a densely drawn Pixorama.

The recognition task is performed using a sliding window approach combined with the CNN classifier. The CNN was trained through exploring different hyper-parameters to optimize accuracy and to avoid overfitting. Although the classifier is trained on a small training sample the recognition method provides high accuracy and low false positive rates.

Convolutional Neural Networks are our chosen tool for this particular object recognition task due to its immense capabilities to learn the relevant complexities and patterns of images for the purpose of completing a particular task, usually classification or localization oriented [1]. These neural networks contain a large number of parameters that are trained by a specific learning algorithm and in the next section we will explain why the choice and generation of the training and test data is especially important in developing an accurate CNN classifier.

In Section 2 we begin by explaining our approach towards generated the data sets; Section 3 illustrates the implementation and results from our chosen CNN architecture; Section 4 explains our proposed methodology for localization; Section 5 provides a brief summary of the paper and discusses the opportunity for future work; and Section 6 provides an appendix for the rest of the results from the project.

Figure 1.1: Sample Pixorama

2. Data Generation

A CNNs large set of parameters require a lot of training data in order to be tuned to converge towards a meaningful state for classification. Data generation was performed manually as there does not exist a public repository with all of the human characters used in the Eboy Pixoramas.

For training the CNN, we needed a data set consisting of images containing characters with positive labels and images not containing characters with negative labels. After examining all of the Pixoramas we noticed that the scale of the characters (and hence other objects like buildings, trees, animals etc.) remained fairly consistent throughout the entire data set. For this reason, we decided that the data set should be generated from cropping out patches of size 30 pixels wide by 50 pixels tall from the Pixoramas. The consistency of image dimensions in the training data set is necessary for training the CNN, the choice of the dimensions is lenient towards allowing the human characters taking up various positions and orientations.

We first gathered a collection of 122 Pixoramas from Eboy on their website. Then, we split the data into 70 images (57%) for training and testing the classifier, and 52 images (43%) for testing the localization algorithm. Since the Pixoramas vary between dimensions, density and complexity, we aimed to maintain a similar level of variance between the two data sets.

In gathering training data consisting of cropped images without characters, or 'background patches', we developed a simple MATLAB script to implement a sliding window algorithm for cropping out fixed-sized patches (30x50) from each Pixorama and saving each patch into its corresponding directory. The stride for the sliding window was chosen so that the windows will not overlap each other at all (stride is 30 pixels horizontally, 50 pixels vertically). The reason for this was to avoid generating similar background patches which would cause our neural network model to overfit.

It is inevitable that this implementation will result in cropping out images that contain characters, hence it was necessary to manually filter through the cropped images to remove such patches after running the script. Eventually we managed to gather 13,501 background patches across all 70 of the Pixoramas. Figure 2.1 shows examples of background patches used. We decided to include significantly more background patches than people patches so as to reduce the number of false positives.

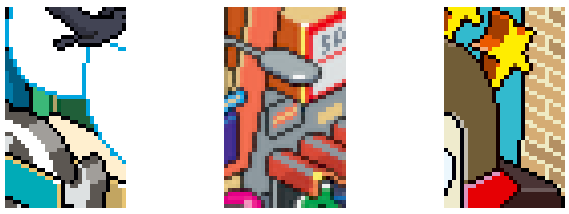


Figure 2.1: Background patches

In gathering training data consisting of image patches with characters, or 'people patches', we manually cropped characters in each Pixorama, with each character contained within an image of size 30x50. This was obtained efficiently through using the exporting tool provided by Affinity Designer. Eventually we managed to gather 2,003 characters across all 70 of the Pixoramas. Figure 2.2 shows examples of cropped charac-

ters.

Figure 2.2: Cropped characters

Additionally, Eboy personally provided us with a collection of image patches containing just characters against a transparent background. In order to generate patches that are consistent with the rest of those that were manually cropped, we could not directly use these images because of their transparent backgrounds, and instead proceeded to superimpose them against the background patches generated from before. The background patches used for this purpose were excluded from the set of background patches all with negative labels. Figure 2.3 shows a collection of patches with transparent background.

We needed to superimpose the character onto a background patch without affecting any of the pixels beyond the character inside the image. That meant we needed to figure out exactly how the character is described, pixel-by-pixel. To do so, we developed a tool to recursively trace the colored pixels belonging to the character in each of the character images, which then stores the exact intensity of each pixel in each channel (R,G,B) and its position relative to a starting point, which usually was taken as simply the top left corner of the image as (0,0). Once these descriptions for each character were generated, the remaining step was to apply the same pixel intensities at the corresponding positions in the background patches, essentially overwriting the pixel intensities at those particular positions in the background patches. The choice of each character and its corresponding background patch is entirely randomized.

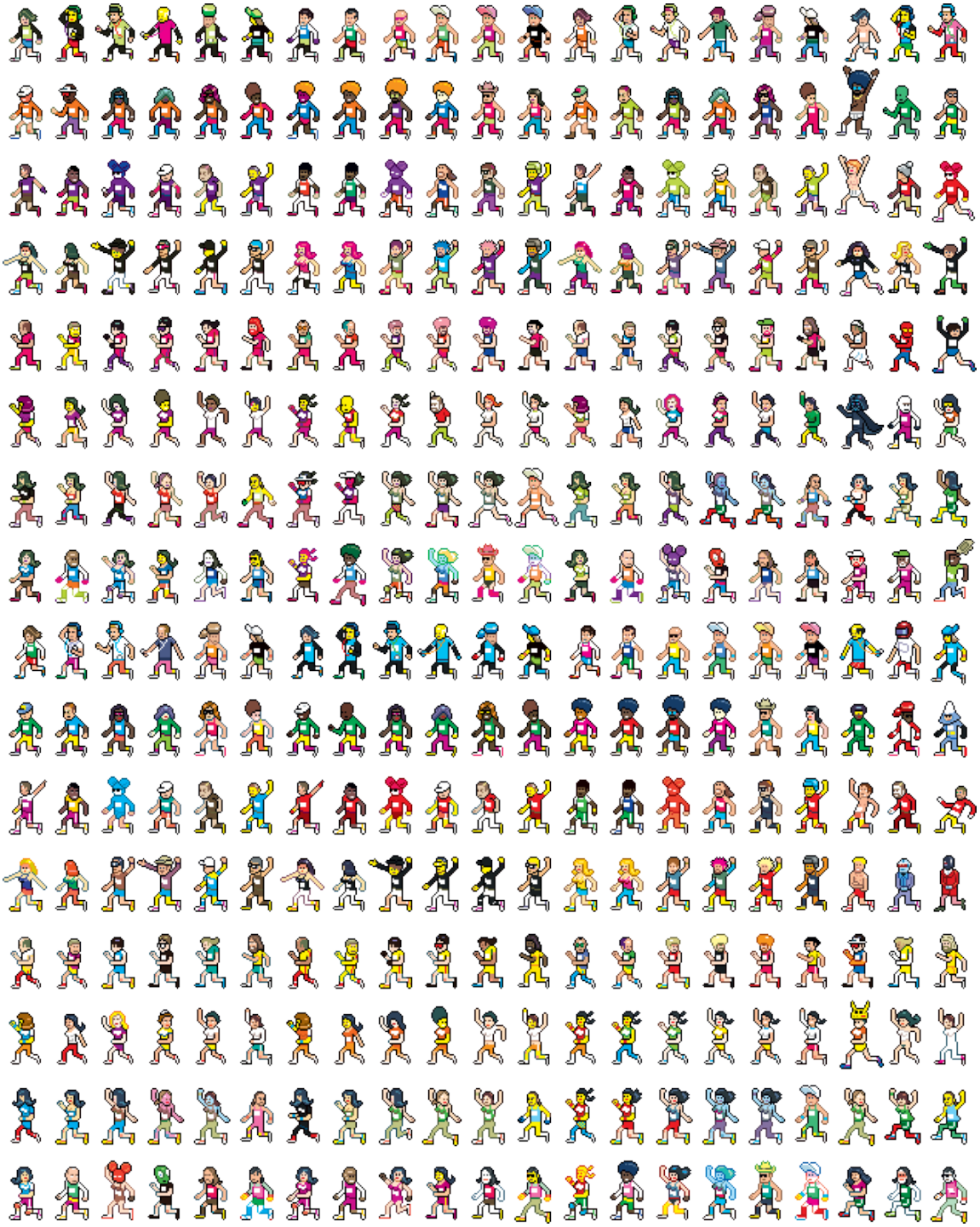


Figure 2.3: Sample collection with transparent background

3. Classification

3.1 Neural Net Architecture

We decided to use MatConvNet [2] - a MATLAB toolbox implementing CNNs for computer vision applications - for setting up and training the CNN. It is simple, efficient, and can run many readily available, state-of-the-art CNNs for tasks including image classification, segmentation and many more.

We started off using the default architecture provided by MatConvNet, a 13-layer CNN (simplenn) consisting of 3 blocks of convolutional, ReLU and pooling layers, a block of convolutional and ReLU layers, and a single convolutional layer before the final softmaxloss calculation. The final layer is a softmaxloss calculation because our objective is to train a CNN to classify whether each patch of 30x50 cropped image contains a human character or not. Hence a single softmaxloss scalar is needed to be compared against the +1 / -1 labels for the positively and negatively labelled data sets. Our optimal configuration involved a slight adaptation to the default setup. Listed below is the sequence of layers in our CNN:

1. Convolutional layer consisting of 16 filters each having 5x5x3 dimensions, with stride 1 and 0 padding
2. Max pooling layer with a 3x3 pooling filter, with stride 2 and padding [0 1 0 1]
3. ReLU unit
4. Convolutional layer consisting of 32 filters each having 5x3x16 dimensions, with stride 1 and 0 padding
5. ReLU unit
6. Convolutional layer consisting of 32 filters each having 5x3x32 dimensions, with stride 1 and 0 padding
7. ReLU
8. Average pooling layer with a 3x3 pooling filter, with stride 2 and padding [0 1 0 1]
9. Convolutional layer consisting of 64 filters each having 5x3x32 dimensions, with stride 1 and 0 padding
10. ReLU unit
11. Convolutional layer consisting of 64 filters each having 3x2x64 dimensions, with stride 1 and 0 padding
12. ReLU unit
13. Convolutional layer consisting of 2 filters each having 1x1x64 dimensions, with stride 1 and 0 padding
14. Softmaxloss layer

```
25  [net,info] = cnn_train(net, imdb, @getBatch, trainOpts) ;
      layer|    0|     1|      2|     3|     4|     5|     6|     7|      8|     9|     10|     11|     12|     13|     14|
       type|input|  conv|  mpool|  relu|  conv|  relu|  conv|  relu|  apool|  conv|   relu|   conv|   relu|   conv|softmxl|
       name|  n/a|layer1|  layer2|layer3|layer4|layer5|layer6|layer7|  layer8|layer9|layer10|layer11|layer12|layer13|layer14|
----------|-----|------|-------|------|------|------|------|------|-------|------|-------|-------|-------|-------|-------|
    support|  n/a|     5|      3|     1|   5x3|     1|   5x3|     1|      3|   5x3|      1|    3x2|      1|      1|      1|
   filt dim|  n/a|     3|    n/a|   n/a|    16|   n/a|    32|   n/a|    n/a|    32|    n/a|     64|    n/a|     64|    n/a|
filt dilat|  n/a|     1|    n/a|   n/a|     1|   n/a|     1|   n/a|    n/a|     1|    n/a|      1|    n/a|      1|    n/a|
 num filts|  n/a|    16|    n/a|   n/a|    32|   n/a|    32|   n/a|    n/a|    64|    n/a|     64|    n/a|      2|    n/a|
    stride|  n/a|     1|      2|     1|     1|     1|     1|     1|      2|     1|      1|      1|      1|      1|      1|
       pad|  n/a|     0|0x1x0x1|     0|     0|     0|     0|     0|0x1x0x1|     0|      0|      0|      0|      0|      0|
----------|-----|------|-------|------|------|------|------|------|-------|------|-------|-------|-------|-------|-------|
    rf size|  n/a|     5|      7|     7| 15x11| 15x11| 23x15| 23x15|  27x19| 43x27|  43x27|  51x31|  51x31|  51x31|  51x31|
 rf offset|  n/a|     3|      4|     4|   8x6|   8x6|  12x8|  12x8|  14x10| 22x14|  22x14|  26x16|  26x16|  26x16|  26x16|
 rf stride|  n/a|     1|      2|     2|     2|     2|     2|     2|      4|     4|      4|      4|      4|      4|      4|
----------|-----|------|-------|------|------|------|------|------|-------|------|-------|-------|-------|-------|-------|
 data size|50x30| 46x26|  23x13| 23x13| 19x11| 19x11|  15x9|  15x9|    7x4|   3x2|    3x2|      1|      1|      1|      1|
data depth|    3|    16|     16|    16|    32|    32|    32|    32|     32|    64|     64|     64|     64|      2|      1|
  data num|  100|   100|    100|   100|   100|   100|   100|   100|    100|   100|    100|    100|    100|    100|      1|
----------|-----|------|-------|------|------|------|------|------|-------|------|-------|-------|-------|-------|-------|
  data mem|  2MB|   7MB|    2MB|   2MB|   3MB|   3MB|   2MB|   2MB|  350KB| 150KB|  150KB|   25KB|   25KB|   800B|     4B|
 param mem|  n/a|   5KB|     0B|    0B|  30KB|    0B|  60KB|    0B|     0B| 120KB|     0B|   96KB|     0B|   520B|     0B|

parameter memory| 312KB (8e+04 parameters)|
     data memory|22MB (for batch size 100)|
```

Figure 3.1: CNN configuration

Refer also to Figure 3.1 for a more visual representation of the neural network.

The results of feeding forward the test set into such a CNN can be seen in the section below.

3.2 Training Neural Net

In order to train the neural net and evaluate its performance, both the people and backgrounds patches were divided into 3 sets: 70% training, 15% validation, 15% test. The validation set was used to tune the training parameters while the test set was left as final evaluation on the performance of the classifier. The patches were randomly divided into different sets but the ratio of people patches to background patches was kept the same in every set, this was done to ensure that each set is as representative of the dataset as possible.

After selecting an architecture for the neural net, the training hyper-parameters had to be tuned to minimize errors on training and validation set without incurring in overfitting. To simplify this process default values for hyper-parameters provided by matconvnet were used as a starting point. Figure 3.2 shows the results from training the classifier through default hyper-parameters.

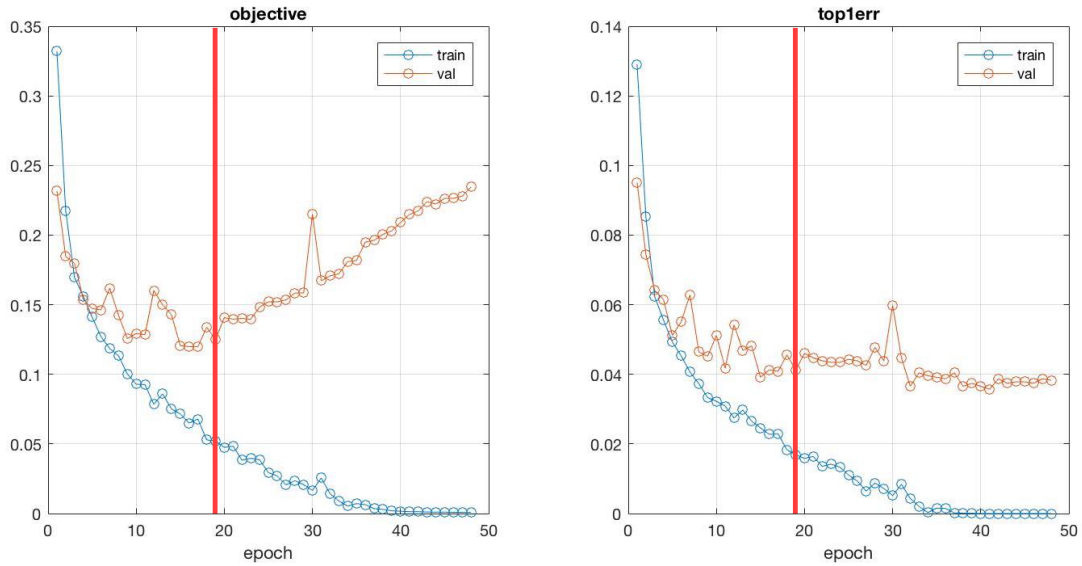


Figure 3.2: Error curves for default parameters

The resulting weights at epoch 19, marked by vertical red line, were used for neural net with default parameters. This was done since the neural network clearly overfits the training data as the learning process progresses. This effect is shown by the increase in the validation objective function, while the training objective function, and classification error, decrease to zero.

In tuning the neural network towards optimality we identified the set of hyper-parameters that could be manipulated, as shown in the lists below, with the value in the brackets showing the default values provided by MatConvNet.

- Hyper-parameters:
  - Learning Rate (0.001)
  - Weight Decay (0.0005)
  - Momentum (0.9)
  - Batch Size (100)
  - Number of Epochs (300)
  - Layers
  - Padding
  - Stride
  - Weight initialization
  - Convolutional filter dimensions

Among the list, tuning the learning rate, weight decay and momentum presented the most significant changes in the performance of the neural network.

The updated weights are calculated using Equation 3.1 where w_i is the vector of weight at iteration i, η is the learning rate, α is the momentum and λ is the weight decay. Momentum smooths the effect of the gradient term by including the weights

of the previous iteration. Weight decay penalizes large weights, it regularizes the cost function to avoid overfitting. To improve the results of the classifier it was decided to increase both terms to regularize cost function and avoid overfitting. The momentum was changed from 0.9 to 0.95. The resulting accuracies of the new classifier are shown in Figure 3.3.

$$\Delta\omega_i(t+1) = -\eta\frac{\partial E}{\partial w_i} + \alpha\Delta\omega_i(t) - \lambda\eta\omega_i \tag{3.1}$$

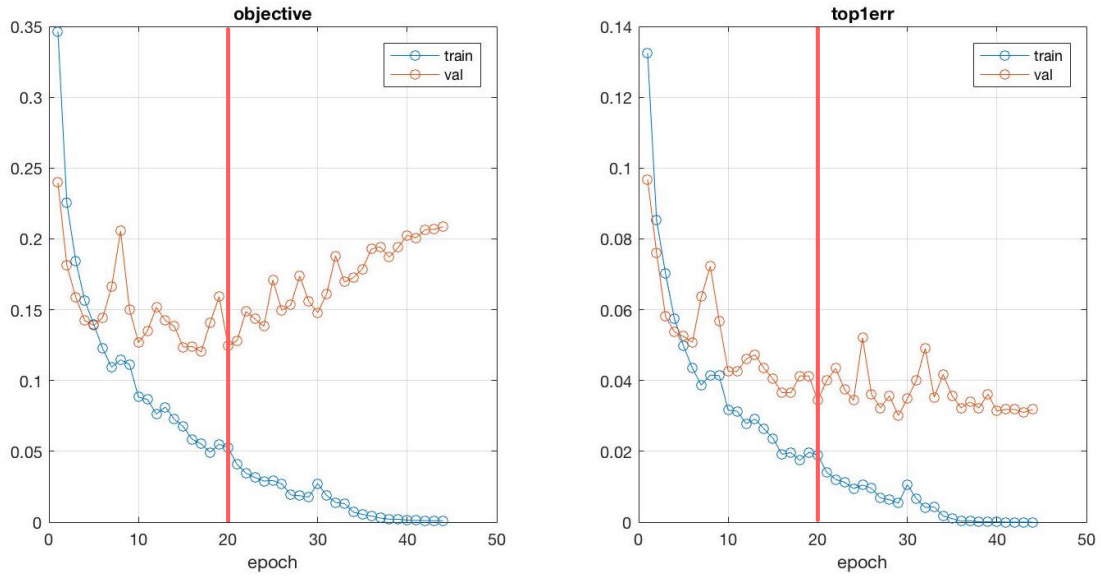


Figure 3.3: Error curves for configuration 1

The figures shows how momentum reduces overfitting and provides better performance on the validation set. To avoid overfitting the resulting weights at epoch 20 were used as final weights for this network. The next test was to increase the weight decay to further regularize the cost function and reduce overfitting. Keeping the momentum to 0.95 the weight decay was increased from 0.001 to 0.005. The results are shown in Figure 3.4.

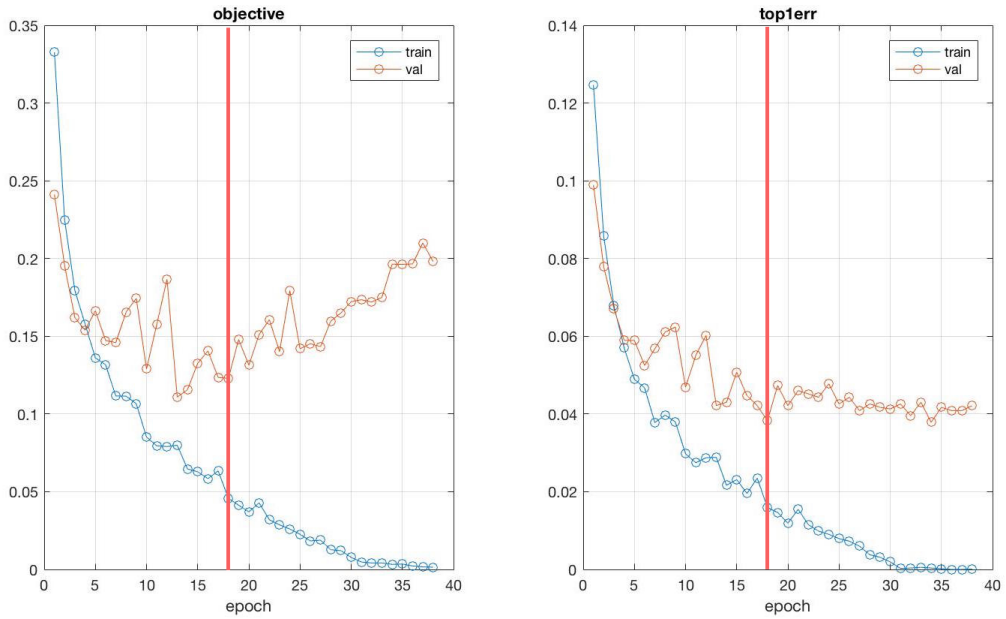


Figure 3.4: Error curves for configuration 2

The results show that the increase in momentum results in a worse accuracy in the validation set, when using result from epoch 18. For this reason the hyper-parameters in configuration 1 were used for the final classifier, i.e. momentum = 0.95 and weight decay = 0.001. The chosen CNN was tested on the test set and provided an accuracy of 96.7 %.

Regarding the rest of the hyper-parameters,

- MatConvNet uses Stochastic Gradient Descent (SGD) to compute the optimal neural network weighting parameters to converge towards. Since the gradient is averaged over each input minibatch, the batch size will essentially affect how 'noisy' the gradient update will be. The default value of batch size (100) is a reasonable compromise between the rate of convergence and the ability to escape local minima, against the accuracy and noise in the gradient updates [3].

- Number of epochs was set to a large number since we did not intend on running through all the epochs anyways, what was necessary was to continuously iterate through sufficient epochs to draw the red line, beyond which the classifier would start to overfit, as discussed earlier.

- The number of layers given by default performed sufficiently well as observed in the results. We did not proceed to add additional layers for modelling greater complexities in the images so as avoid exacerbating the overfitting.

- The choice of the convolution kernel was kept small to extract key features from the previous layers.

4. Localization

The classifier described in section 3 was implemented to perform detection on complete images. Since people in images have similar size it was decided to use a single scale window sliding detection algorithm. As described in section 2, people drawn on the images used for training and testing have similar size and can fit in a window of 30x50 pixels. For this reason it was decided to use a single scale window sliding detection algorithm. A window of 30x50 is slided across the image with a stride of 3 px, at each iteration the window is fed to the classifier and the softmax function is used to obtain a probability of the patch being a person. If this probability is above the threshold then it is marked in the image, if the probability is lower than the threshold but above 90% the threshold then the stride is decreased to 1 to maximise detection accuracy close to possible people. This approach involves a large number of classifications that can lead to a large number of false positives. To avoid this a high threshold of 0.9 was used, although this might reduce the true positive rate it was decided to give more importance to avoiding false positives.

To further reduce the false positive rate non maximum suppression (nms) was applied after the proposal boxes were produced. Nms was implemented using an overlapping ratio defined as in Equation 4.1.

$$Ratio = \frac{area(A \cap B)}{area(A \cup B} \tag{4.1}$$

If two boxes proposals have an overlap greater than the threshold, the box with the lowest score is removed. The choice of the nms ratio has two opposite effects: setting it too small will cause a large number of overlapping detection, setting it too large will compromise detection in dense scenes. The compromise between these two effects was found in setting the nms ratio to 0.2. The effect of applying different nms ratio in dense scenes is shown below for two images. A subset of the results of the localization with sliding window are shown in section 6.

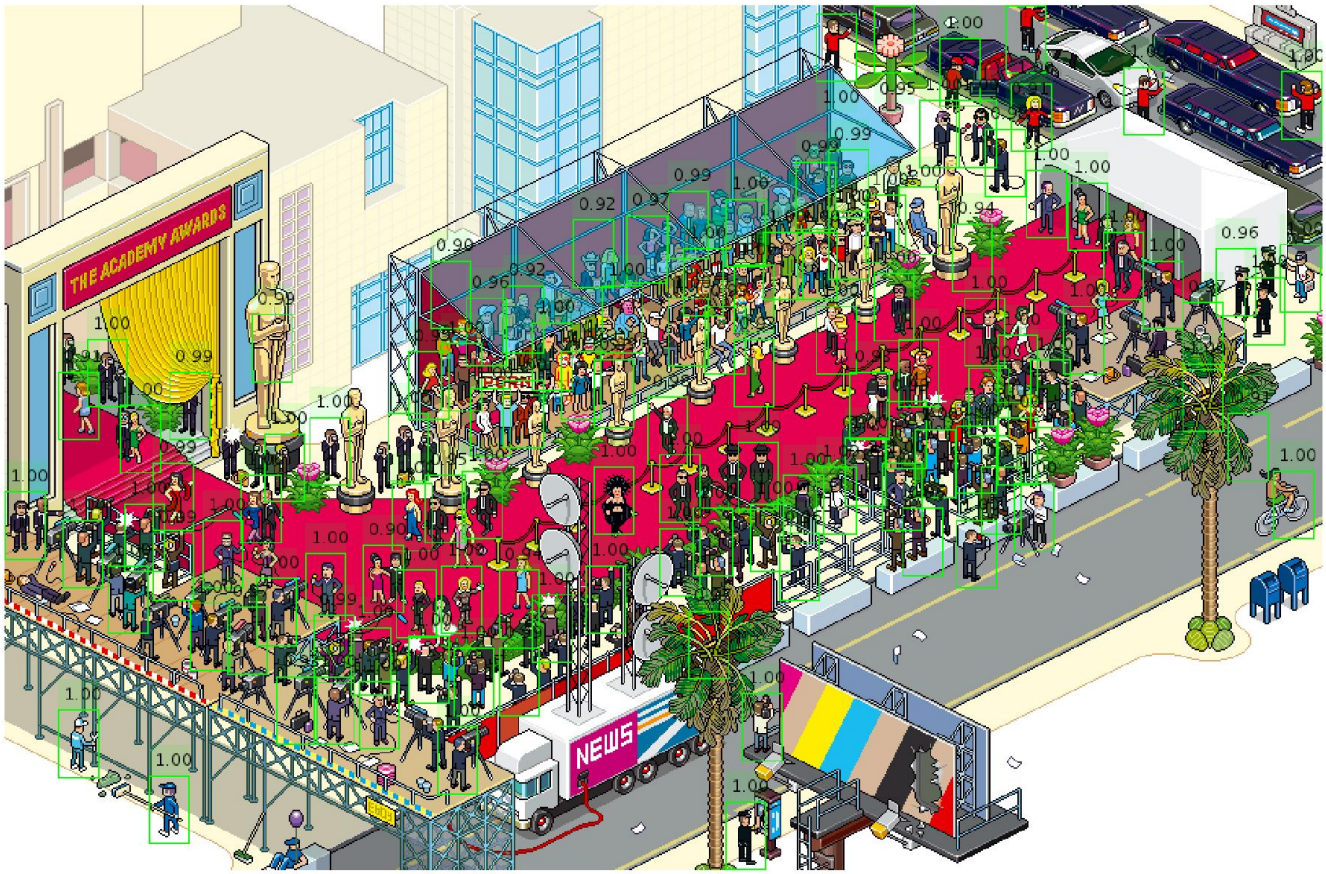


Figure 4.1: red carpet image (overlap = 0.1)

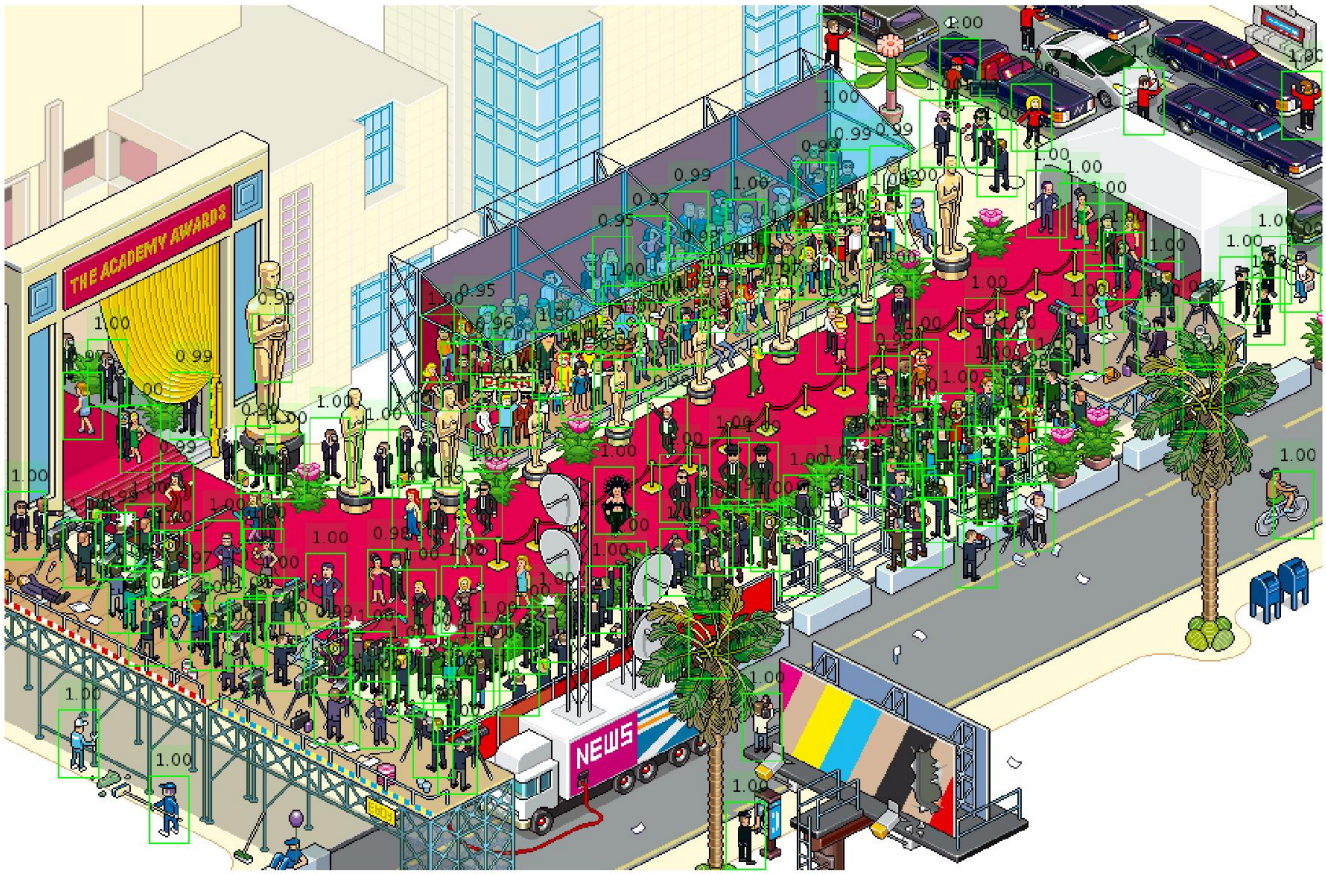


Figure 4.2: red carpet image (overlap = 0.2)

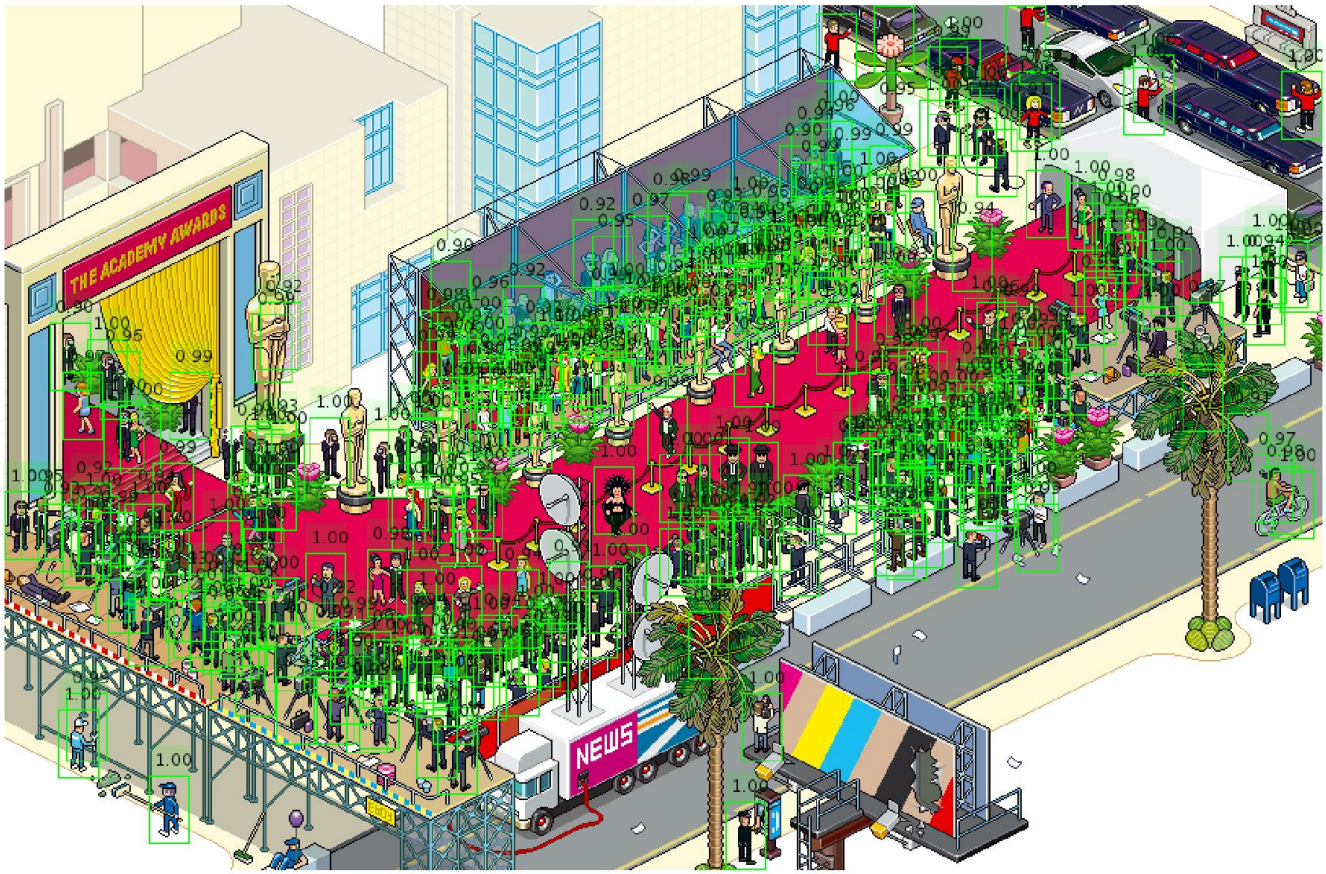


Figure 4.3: red carpet image (overlap = 0.5)

Figure 4.4: concert image (overlap = 0.1)

Figure 4.5: red carpet image (overlap = 0.2)

Figure 4.6: red carpet image (overlap = 0.5)

5. Summary

In this project we developed a localization method to detect people in pixel art pieces, called Pixoramas. The first step was to develop and optimize a CNN to perform classification of 30x50 people patches. To train the classifier a dataset of people and background patches was created. After trying different configurations and tuning the hyperparameters of the network, the CNN was able to classify people correctly with 96.7% testing accuracy. This classifier was then used in a sliding window localization algorithm to perform localization on complete Pixoramas. The resulting method provides good detection and low false positive rates.

Although the classifier requires images of 30x50 pixels size as an input, it is still possible to run a localization algorithm through larger patches of the images. This requires subsampling of the image patch to reduce it to the classifier input size.

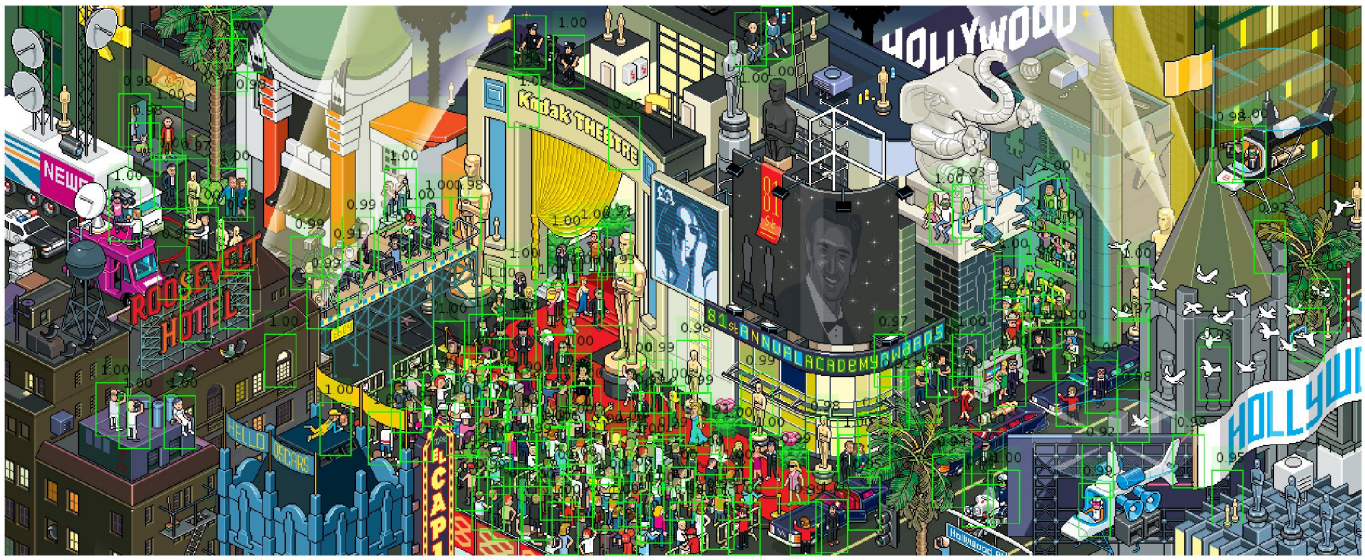
To perform localization without previous knowledge of the correct size of people in the image, hence localization on multiple scales, the window sliding algorithm is not suitable for the task. Images from Eboy are usually size 800x1200 pixels. Performing a window sliding algorithm on multiple scales on images of this size will result in a huge number of classification instances, compromising the computational efficiency of our program as well as its classification accuracy due to its susceptibility to high false positive rates. To solve this problem a proposal method can be used to pre-select patches to be fed to the classifier. A possible proposal method to be considered is edge boxes [4]. Edge boxes is a proposal method that gives a score based on the edges belonging to a contour fully contained within a box subtracting those that belong to contours overlapping the boxs boundary. As people in these images are characterised by a thick black edge, this method is expected to give good proposals. This can potentially enable the extension of the localization algorithm to other pixel art artists who use different scales for their drawings.

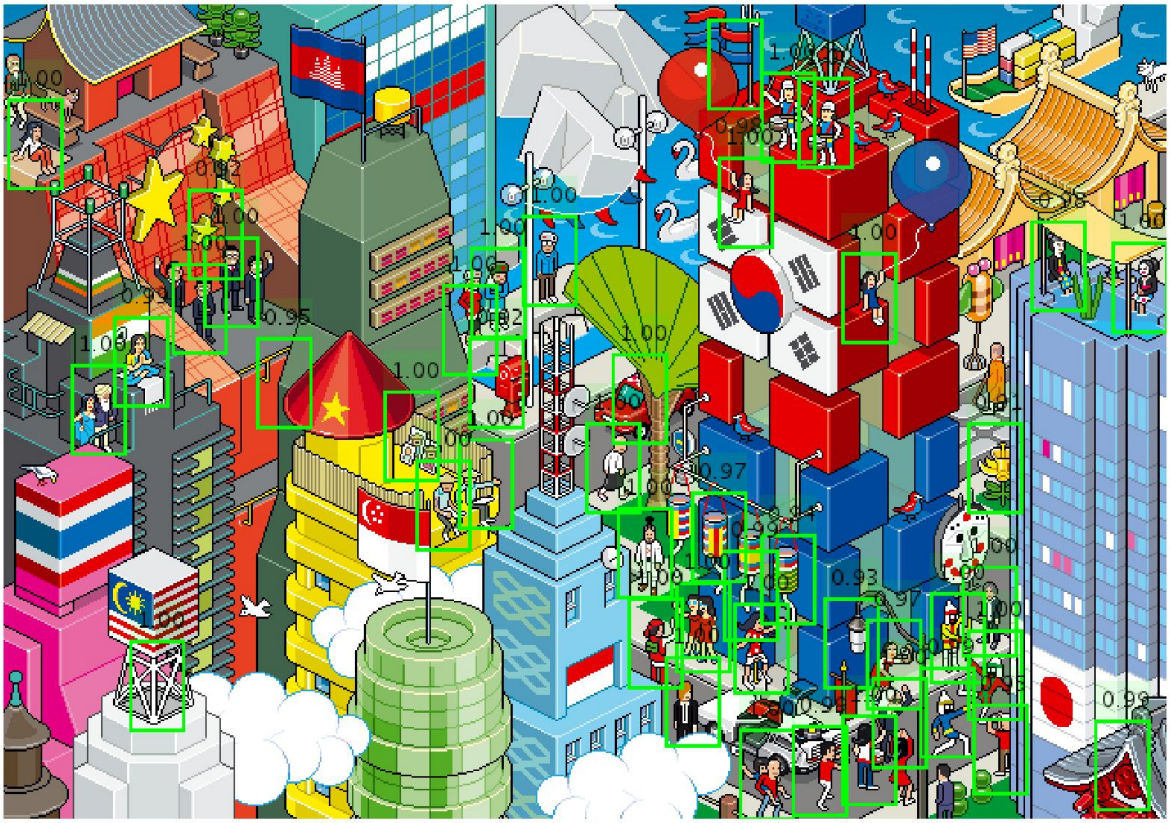
6. Supplementary Information

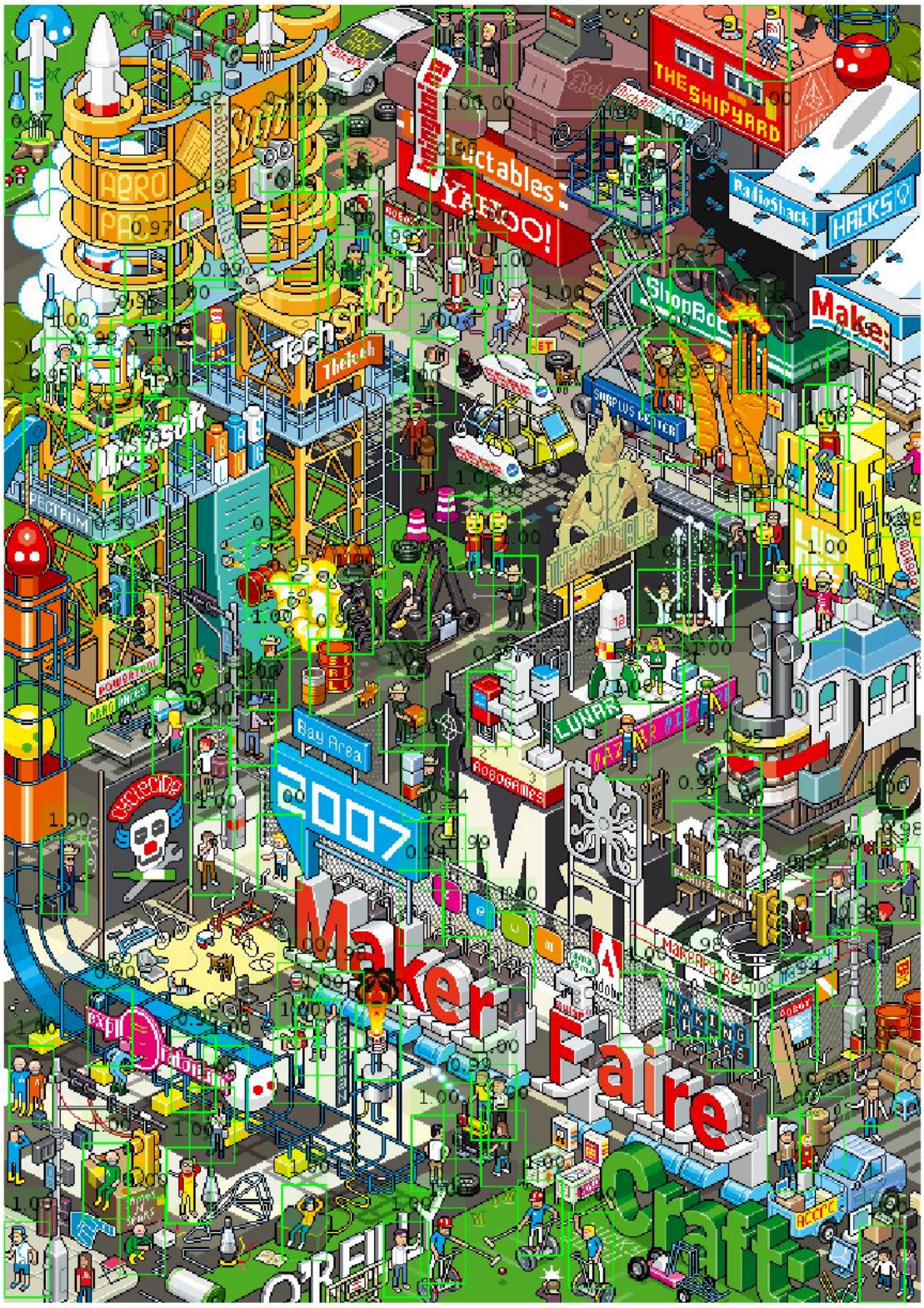
This section shows some of the results obtained from the recognition method described in section 4.


T-CITY
MARKET


BADEMEISTER
NICHTSCHWimmer


HOLLYWOOD
Kodak THEATRE
ROOSEVELT
HOTEL
84th ANNUAL ACADEMY AWARDS




THE SHIPYARD
Tructables
YAHOO!
RadioShack
HACKS
Make:
ShopBot
Tech Shop
Thetech
SURPLUS CENTER
SPECTRUM
Bay Area
2007
Maker
Faire
Craft
Robogames

Bibliography

[1] Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven A Siegelbaum, and AJ Hudspeth. *Principles of neural science*, volume 5. McGraw-hill New York, 2012.

[2] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for matlab. In *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.

[3] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[4] C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *European Conference on Computer Vision*, pages 391–405. Springer, 2014.