SSY098 Image Analysis - Digit Recognition in Scale Space

Stefano Ribes
Chalmers University of Technology
ribes@chalmers.se

Abstract

This report describes the work done within the final project of the course SSY098 Image Analysis at Chalmers University of Technology. The aim of the project is to implement a digit recognition system in a scale space. In particular, it outlines the design of a Fully Convolutional Neural Network (FCN) and an algorithm for removing duplicate detections of the same digits via Non-Maximum Suppression (NMS). The combined use of an FCN and NMS defines a semantic segmentation system which is able to identify the location and label of digits that are present in a given image. The system has been designed, tested and evaluated on the handwritten digits MNIST dataset. The final semantic segmentation algorithm is able to achieve a mAP of 0.64 and 0.12 on two images containing computer generated and handwritten digits respectively.

1. Introduction

In the computer vision and image analysis field, the semantic segmentation problem can be summarized as detecting the position and the type of objects within an images. Semantic segmentation is a natural steps forward in complexity from classifying a single object in a given image.

In recent years, the performance of semantic segmentation algorithms has been revolutionized by the introduction of Fully Convolutional Neural Networks (FCNs) [9]. FCNs consist of a deep learning neural network where there are no Fully Connected (FC) layers and its typically made of two parts: a downsampling network and an upsampling network. The two parts are then connected not only back-to-back, but also via skip connections, effectively linking layers of the downsampling part to the upsampling part. The output of the network is a feature map of the same width and height of the input image but with as many channels as the object classes to classify. Thanks to this architecture, FCNs are able to assign classification labels directly to the pixel of a given image. Deep learning systems exploiting FCNs such as U-Net [11], R-CNN [7] and then YOLO [10] achieved impressive results in terms of accuracy and mAP [1].

As mentioned above, the overall idea behind these systems is to color the pixels of the given image corresponding to the detected classes. To do so, the training set consists of the actual images, and the classes and the Region Of Interests (ROI) [3, 6] of the objects present in the images. At a simpler level, the deep learning system is instead trained to identify and assign a classification label (and in turn a confidence probability) to the *bounding boxes* of objects in the provided images. Bounding boxes essentially represent the ROIs of an image as rectangular shapes. In order to remove duplicate detections, a Non-Maximum Suppression (NMS) algorithm is typically applied [12]. The NMS allows the system to get rid of overlapping boxes with a low confidence score. Another challenge faced by semantic segmentation algorithms is to identify objects at different scales. More advanced models, like U-Net, are able to detect objects at different scales, but when the input feature map of a FCN is fixed, then the image needs to be resized in order to provide scaled image patches to the FCN to detect.

This report builds on top of the aforementioned concepts and describes the implementation of a simple semantic segmentation system within the final project of the course *SSY098 Image Analysis* at Chalmers University of Technology.

In the remaining of the report, Section 2 explains the methodology of my work. Section 3 shows instead the experimental results of the system evaluation. On the other hand, Section 4 includes the answers to the theoretical questions of the project. Finally, Section 5 summarizes the work done and provides some conclusions.

2. Methodology

In this section are reported a detailed description of the FCN, how the bounding boxes are generated and how they are filtered via NMS. In the end, it gives an overview of the designed system.

2.1. FCN Design

Starting from the FCN architecture, the FCN has been designed as a semantic segmentation network rather than

a classification one. Because of that, there are no FC layers and the last one is of type *pixelClassificationLayer*, allowing the FCN to accept images of any size as its inputs. However, at training time, the input layer accepts an image of size $28 \times 28 \times 1$ and performs no automatic normalization. Normalization is instead done on the dataset images themselves by normalizing the pixel values between 0 and 1. The output of the FCN consists instead of a feature map of dimension $1 \times 1 \times 10$, one for each possible class in the dataset. Between the input and output layers, there are a total of three Convolutional layers.

Since the project required the network to have an output feature map of dimensions $28 \times 28 \times 1$ and since I do not have the proper dataset at hand, I'm not using a U-Net style architecture for the project.

2.2. Bounding Boxes

In order to determine the image patches to feed to the FCN, a simple sliding window approach is utilized. In particular, the procedure aims at collecting the bounding boxes information encoded as $(x, y, width, height)$, where x and y represent the coordinates of the top left corner of the rectangle. The first step is to define a fixed bounding box dimension, in our case, a square of size equal to the FCN input image size. Next, a *stride* parameter allows to slide the window by skipping some pixels, effectively generating a series of overlapping boxes. In our case, the stride size should be selected between 1 and the image size, in order to avoid missing image pixels. However, there is a trade-off when selecting the stride size: a too small stride size will generate many bounding boxes and might affect the performance of the NMS step. Moreover, it might slow down the semantic segmentation processing, since the FCN needs to be executed for a larger number of times and the number of IoU comparisons increases.

The network is applied to the bounding boxes through the `semanticseg()` function, which returns the classification probabilities of each of the available classes. Afterward, the maximum probability is selected as a *confidence score* for the given bounding box. In practice, each bounding box has a score generated by the FCN that determines the level of confidence the FCN has on whether the image contains a particular digit.

2.3. Scale Space

Since the bounding box size is fixed, the FCN would not be able to detect digits represented at different scales than the one used for training. Because of that, an image pyramid is constructed [8]. In practice, a discrete scale space is generated to extract bounding boxes of the same size at different level of scaling. The scale space is generated as follows: first, a Gaussian filtering is applied to the image. Next, pixel down- or up-sampling are applied to generate a new, resized, image. This procedure reduces the aliasing introduced by pixel sampling and prevents the resized digit images to become too different from the training samples seen by the FCN. As a final note, the new image pixels are then normalized between 0 and 1, since the FCN is typically not scale invariant.

Algorithm 1: Semantic Segmentation algorithm.

Inputs: img input image, N number of discrete scale levels
Outputs: b list of bounding boxes, their confidence scores s and their labels l.

begin
  $final\_b \leftarrow [\,]$
  $final\_s \leftarrow [\,]$
  $final\_l \leftarrow [\,]$
  $T \leftarrow [1, ..., N, 1/2, ..., 1/N]$
  for *t* ***in*** T **do**
    $scaled\_img \leftarrow rescale(img, t)$
    $b \leftarrow get\_bboxes(scaled\_img)$
    $b \leftarrow b \cdot t$
    for *i* ***in*** $length(b)$ **do**
      $[s(i), l(i)] \leftarrow FCN(b(i))$
    end
    $[b, s, l] \leftarrow apply\_confidence\_threshold(b, s, l)$
    $[b, s, l] \leftarrow NMS(b, s, l)$
    $final\_b \leftarrow [final\_b, b]$
    $final\_s \leftarrow [final\_s, s]$
    $final\_l \leftarrow [final\_l, l]$
  end
  $[b, s, l] \leftarrow NMS(final\_b, final\_scores, final\_l)$
  $return [b, s, l]$
end

2.4. Non Maximum Suppression

The NMS follows a greedy algorithm in discarding overlapping bounding boxes with a low confidence score. In our system NMS is applied at two stages: first after getting the boxes at a specific scale level, then once all scale levels have been explored. In fact, the bounding boxes dimensions and positions are also scaled accordingly, in order to have a consistent coordinate system. In this way, the NMS algorithm can remove bounding boxes duplicates of the same size and of different scales. In particular, at each scale level, since there might be many duplicates of the same digit, the NMS will discard overlapping boxes which are classified with the same label. On the other hand, when we are considering boxes of different scales, the NMS is applied only considering the best score, independently on the box label.

2.5. Semantic Segmentation Algorithm

Given an image to apply semantic segmentation, the overall algorithm is illustrated in Algorithm 1.

3. Experimental Evaluation

This Sections details the performed experiments and the performance of the implemented system.

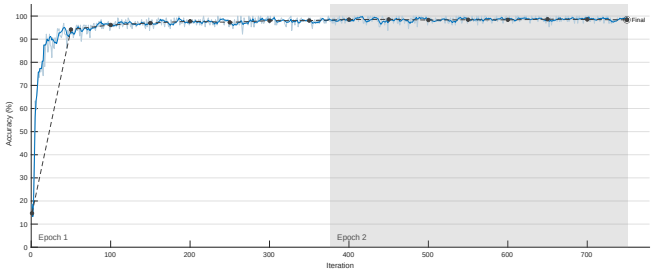

Figure 1. Training accuracy over epochs. The dotted line represents the validation accuracy.

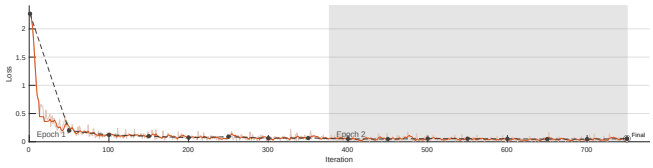

Figure 2. Training loss over epochs. The dotted line represents the validation loss.

3.1. FCN Training

The original training set of the MNIST dataset has been split in 80% of the data being the training set and 20% as the test set. The cross entropy function has been used as the training loss function. The FCN has been trained with an Adam solver [13] with learning rate of 0.002. The number of training epochs was limited to 2, since the training and validation loss reached a value of 0.0356 and 0.541, respectively, with a training and validation accuracy of 98.44% and 98.37%, respectively. The training and validation accuracy and loss over epochs can be viewed in Figures 1 and 2. On the test set, the FCN scored an accuracy of 98.367%.

3.2. AP Score

Figures 5 and 6 show the identified bounding boxes and the respective classified labels in case of the images containing computer generated digits and handwritten digits respectively. In comparison, Figures 3 and 4 show the original images. Table 1 reports instead the mAP results of applying semantic segmentation to two aforementioned images. The presented results are the best, in terms of average mAP, which were obtained by a design exploration phase reported in the attached MATLAB Live Script.

The mAP score has been calculated with an IoU threshold of 0.3. The confidence level threshold is 0.9 and it's proportionally reduced when applying a scale level. This is done in order to keep the detections with lower confidence even in cases where the input patch might be "more" distorted due to rescaling.

We can notice that the average mAP of the system on the computer generated digits is significantly higher than

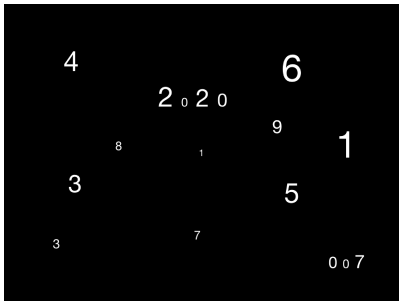

Figure 3. Original image containing computer generated digits.

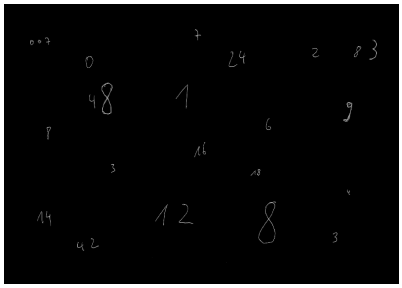

Figure 4. Original image containing handwritten digits.

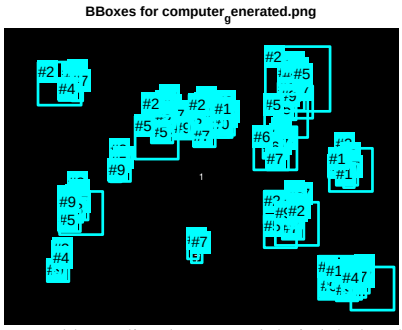

Figure 5. Detected bounding boxes and their labels when applying the designed system to an image containing computer generated digits.

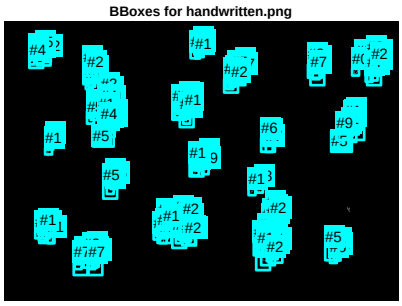

Figure 6. Detected bounding boxes and their labels when applying the designed system to an image containing handwritten digits.

the mAP over the handwritten digits, being 0.64528 and 0.12371 respectively.

Table 1. The average mAP score per digit, evaluated for two images: one containing computer generated digits and another handwritten digits.

Digit	Computer Generated mAP	Handwritten mAP
0	0.50	0
1	0.50	0.0232
2	0.25	0.1806
3	1.00	0
4	1.00	0
5	1.00	0
6	1.00	0.50
7	0.50	0
8	0	0.20
9	0	0.33
Avg.	0.64528	0.12371

3.3. Discussion

The system has been tested over only two images and its performance are not considerably impressive. I believe that more extensive hyperparameter tuning might improve the mAP of the system in handwritten digits as well. For example by carefully tune the amount of scale levels, their scale, and the stride size accordingly. Visually, the system is able to distinguish the digits' pixels from the background pixels. In general, however, the system is generating far too many boxes, as it is not able to efficiently remove all duplicate detections.

4. Theoretical Part

Fully convolutional neural networks vs. neural networks with fully connected layers. Intuitively, CONV layers have the ability to extract more high level features from the input feature map/image, since they "see" different patches of the input. This becomes even more pronounced when the CONV layers are cascaded together in a deep neural network. Compared to FC layers, CONV layers work best with image data because they are efficient in extracting the structured features of an image (like a face image that has a set of eyes, which are made up of edges, et cetera...).

On the other side, FC layers are typically placed *after* the CONV layers in order to learn a non linear function which maps the features extracted by the CONV layers to the expected data at the output. More generally, FC layers are able to densely consider all the input space and proven to be *universal function approximators* [5]. In practice, they work well in modeling non linear functions.

FC networks are simpler to design and provide quite remarkable performance when the data does not present marked spatial features, for instance when dealing with regression tasks. On the other hand, CONV networks are highly suitable when dealing with structured data such as, for example, images or audio signals.

Lastly, the number of complete connections, and in turn parameters, of a FC layer is considerably higher than the one utilized by a CONV layer of same dimensions. Therefore, one should take care in dimensioning FC layers, as their computational requirements might become higher than the ones required by CONV layers.

Converting a fully connected to a fully convolutional layer. The biggest advantage is that CONV layers do not need to have a fixed input size (that needs to be true only during training), but larger images can also be fed into the layer. In this way, when applied to larger inputs, *e.g.* images, the CONV output will be a *spatial map* of output vectors, instead of being a single vector as in FC layers.

Imagine a FC with a weight matrix W of dimensions of $m \times n$, then its output would be:

$$f(x) = ReLU(W \cdot x + b)$$

where x is the input feature map of dimension m and b the bias term of size n. On the other side, a CONV weight kernel has dimensions $k \times k \times d$, meaning that there are d filters of dimensions $k \times k$ that are convolved over the input image/volume. This means that, in order to obtain the same function as of an FC, we would need to *stack* n filters (the same number as the output of the FC layer) of dimensions $1 \times 1 \times m$ [2].

Negative training examples. A possible solution would be to add an extra label to the label set indicating a background. The training dataset would need to be modified to include background patches and their respective labels. In practice the output feature map would have an extra dimension, *e.g.* moving from $1 \times 1 \times 10$ to $1 \times 1 \times 11$. The loss function could then still be a cross entropy function. During the NMS phase, the algorithm would add an additional step in checking first the confidence scores related to the background class, and remove them accordingly. Then it would proceed to remove the once with a lower confidence in the other classes.

A possible way to generate negative examples from the available dataset might be to use a *Bootstrapping* training technique [12, 4]. The idea would be to sample, at each training pass, a certain amount of images generating the lowest possible confidence score and add them to the training data as background. This process is then repeated a fixed amount of time, effectively enlarging the original dataset and training the network.

Otherwise, a simpler solution for MNIST dataset in particular might be to artificially construct background images. For example by adding as background objects parts of the

original images and then fill the remaining parts with black pixels. For instance, one could divide the image into four smaller squares, then select only one of them and fill the rest with black pixels.

A final solution might also be to exploit Variational Autoencoders to generate handwritten digit and then add some noise to the images.

5. Conclusions

In this report I described my work on the design and evaluation of a semantic segmentation system. Overall, the system was able to achieve an average mAP score of 0.64 and 0.12 on the provided test images, with a maximum in the image containing computer generated digits.

References

[1] Breaking Down Mean Average Precision (mAP). https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52. Accessed: 2022-03-20. 1

[2] Convolutional Neural Networks (CNNs / ConvNets). https://cs231n.github.io/convolutional-networks/. Accessed: 2022-03-20. 4

[3] H. Azizpour, A. Sharif Razavian, J. Sullivan, A. Maki, and S. Carlsson. From generic to specific deep representations for visual recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 36–45, 2015. 1

[4] C. Banga and F. Ghorbel. Optimal bootstrap sampling for fast image segmentation: application to retina image. In *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 638–641. IEEE, 1993. 4

[5] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. 4

[6] M. Everingham, S. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015. 1

[7] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 1

[8] T. Lindeberg. *Scale-space theory in computer vision*, volume 256. Springer Science & Business Media, 2013. 2

[9] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 1

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1

[11] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 1

[12] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3626–3633, 2013. 1, 4

[13] Z. Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–2. IEEE, 2018. 3