Fast Bib Number Detection And Recognition – Deep Learning Implementation

Jean Luc ANOMA

Summary

Recognizing text in natural photographs is a hard problem due to the multiple sub-tasks that it entails : detecting and localizing the text, segmenting the text region and recognizing the characters. A few solutions have been implemented for this problem. Traditional solutions use morphological image processing to detect and segment the text region and pass this region proposal to an Optical Character Recognition tool. The more recent development of deep learning algorithms for computer vision led a few researchers to tackle this problem from a different angle. A unified solution fully based on machine learning has been developed to recognize bounded sequences of numbers in a natural scene. However, no performance details regarding the speed of the algorithm were provided. It appears that deep learning algorithms tend to be very challenging to deploy on consumer hardware. For that reason, building efficient and fast deep learning algorithms has recently been a very active area of research.

Keywords: Text detection, character recognition, Deep Learning, Convolutional Neural Networks.

1. Introduction

As expressed in [2], Recognizing arbitrary multi-character text in unconstrained natural photographs is a hard problem. Compared to scanned documents, the challenge is much harder due to added noise, interfering objects, occlusion of part of the text, illumination, shadows, rotation ect…. With the view to approaching the bib number detection and recognition problem with the best tools available, a thorough state of the art needs first to be done. The objective of this report is to draw up an overall picture of the state of the art approaches in use for character chain detection and recognition in pictures or videos, pointing out the advantages and drawbacks of each of them. A strong focus will be put on deep learning methods as is it the selected set of methods for our problem.

2. Problem formulation

The final aim is to time an athlete by videoing the finish line and reading the bib numbers in ‘quasi’ real-time. So given an image, the task is to identify the bib present in the image, as a sequence of digits of bounded length. The number will have to be identified entirely and in the right order. A sequential implementation for this problem already exists and allows to recognize a few bib numbers by second. Our work consists of implementing this problem using deep learning models to try and achieve better speed and/or accuracy, with the help of GPU chips.

3. Methods based on pre-processed images

The character recognition methods which are based on pre-processed images are often composed of 3 main phases:

- Pre-processing of the image to increase uniformity
- Text spotting
- Optical Character Recognition

3.1. Image pre-processing methods

Those methods are usually based on **mathematical morphology**. As explained in [3], mathematical morphology is used as a tool for extracting image components that are useful in the representation and description of region shape. The basic mathematical operations, which are the building blocks for most complex operations, are **dilation** and **erosion**. Based on the latter, we can build **opening** and **closing** operations.

Dilation and erosion are set theory operations. Those operations transform a set *A* into a new set *A’.* The transformation of the set *A* uses another set, the **structuring element**, as a parameter. As pointed out in [3], the structuring elements are small sets of sub images used to probe an image for properties of interest. The structuring element has both a shape and an origin as shown in *figure 1*.

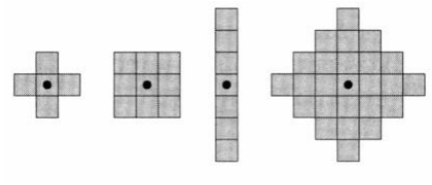


Figure 1: Examples of structuring elements

Dilation is used to enlarge the original set *A* while erosion is used to shrink it. Based on those basic operations, in the context of computer vision, the opening operation generally smooths the contours of an object and eliminates thin protrusions. On the other hand, closing fuses narrow breaks and long and thin gulfs and eliminates small holes and gaps in the contour ([3]).

As a result, Morphological Image Processing enhances degraded noisy images and can be used to fix incomplete or broken characters.

In addition, in [1], a filtering process is applied with a view to sharpening foreground/background transitions and reducing image complexity. Traditional filtering (i.e. order-statistic filters such as Gaussian, Median Blur etc) is not applied since it manages noise by using weighted averages of the neighborhood around each pixel. This approach increases image uniformity but also smear the image and lose definition around high-contrast boundaries. Instead, a Majority Vote Processor (MVP) is used, which consists in modifying the median blur filter to select the most common value in a pixel's neighborhood. This is done in two steps :

- First the image resolution is reduced to limit the range of available grays (in the case of a gray-scale image)
- Then an aperture is run across the image and the most common value is selected in the neighborhood of each pixel

After applying this MVP transformation to an athlete's bib image, the result is higher image uniformity together with sharp boundaries around the bib. According to [1], this MVP process enhances the speed of traditional Extremal Regions (ER) algorithms for text detection by forming blobs of gray which the ER algorithm will exploit.

To conclude on this part, experience has proved that the methods described above tend to ease the text detection process. However the main drawback is that some of them (filtering for instance) incur a high processing overhead [1], even though a subsequently more efficient text detection may compensate for part of or all of this overhead.

3.2. Text detection

This part of the process is about detecting the presence of text in an image and locate this text by drawing boundaries around it. The end objective is to have a region which can be successfully submitted to an Optical Character Recognition tool [1]. Two categories of methods can be quoted:

- Character Region Method
- Sliding Window method

a. Character Region method

An example of Character Region method is the use of Extremal Regions. Extremal Regions are dark regions fully surrounded by a lighter region (maximum intensity region) or light region surrounded by a darker region (minimum intensity region) [6].

Extremal Regions can be built ‘pixel by pixel’. In [1], a quicker algorithm based on blobs is implemented and proves to be 50% quicker than ‘pixel by pixel’ methods. The algorithm is called “Building Extremal Regions using Blobs” (bERb). It uses the blobs of gray formed after applying the MVP algorithm. First the bERb algorithm groups adjacent pixels with the same shade of gray into the same blob using a linked list. Then, after initializing the Extremal Regions with the black blobs, existing Extremal Regions iteratively absorb the blobs in their neigbourhhod starting from the darker ones. After a few iterations the algorithm is stopped, as soon as the bib numbers are perfectly formed and can be successfully submitted for Optical Character Recognition.

In [1], experiments show that the use of MVP plus bERb algorithm saves significant processing time with negligible cost in terms of accuracy. However, bERb algorithm’s main inconvenient is that it requires pre-processed images that are highly uniform with the risk of terminating with insufficient memory resources. The other inconvenient is that the algorithm is less robust to dysfunctional images than other ER-based methods.

b. Sliding window method

This family of methods consists in sliding a box around the image and applying a classifier to each image crop to decide whether it is containing text or not. The result is a text saliency map which can be used to draw boundaries around the region predicted to contain the text.

In [7], every image crop is analyzed using feature extraction and is injected as an input into a linear Support Vector Machine to detect the presence of humans in an image. This process has been reused by [8] in their end-to-end scene text recognition.

4 Deep Learning

4.1 Basic Deep feedforward neural network models

Computer vision has traditionally been one of the most active research areas for deep learning applications [2]. In this section, we will introduce Deep Learning and its applications to computer vision.

Traditional machine learning algorithms like support vector machines, k-nearest neighbors, decision trees..., have worked well on a wide variety of problems but have not succeeded in solving the central problems in AI, such as recognizing speech or recognizing objects. The main barrier has been what has been called “the Curse of Dimensionality”: those types of problems impose to learn complicated functions belonging to a very high dimensional space ([2]). This has made generalization to new examples very challenging. These challenges have laid the ground for the development of deep learning algorithms to tackle these classes of problems.

Deep feedforward networks, also often called feedforward neural networks, or multilayer perceptrons (MLPs) are the the typical models used in deep learning, the main other one being **Recurrent Neural Networks** (RNNs). In deep feedforward networks, the flow of operations goes from the input, through the intermediate computations and finally to the output. Unlike RNNs, there is no feedback connection in which outputs of the model are fed back to itself ([2]). A **convolutional neural network** is a specific type of deep feedforward network which is based on convolution operations on the input image.

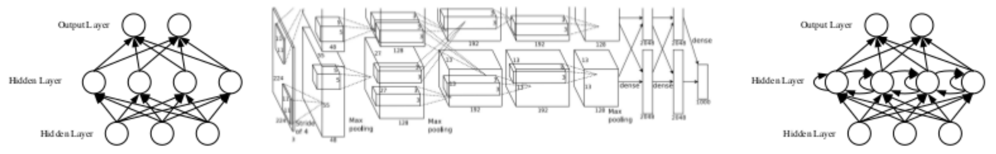


Figure 2: On the left, a fully connected deep feedforward neural network; in the middle a convolution neural network; on the right, a recurrent neural network

Typically, deep feedforward networks are composed of several layers, each of which represent a function to be learned. Those functions are then connected into a chain to produce the desired output([2]). The length of the chain gives the depth of the model. “Deep” models typically have at least two layers, the final one being the **output layer** and the previous ones being the **hidden layers**. The first layer, which contains the inputs, is called the **input layer**.

Each layer is composed of a certain number of units called **neurons** by analogy to the neurons of the brain in the sense that they receive inputs from many other units and compute their own activation value. To move forward from one layer *A* to the next layer *B,* the output of the neurons of layer *A* are transformed linearly to compute **scores :** $s = Wx$ (with x being the input to the layer and W the **weight matrix**, each weight being a parameter of the model to be optimized). Those scores are then used as input for a **non-linear activation function** which computes the outputs of the neurons of layer *B*. According to [2], in modern neural networks, the default recommendation for the non-linear activation function is to use the rectified linear unit or ReLU defined by *g(s) = max{0,s}*. The weight matrix *W* is the one which has to be learned during the training phase.

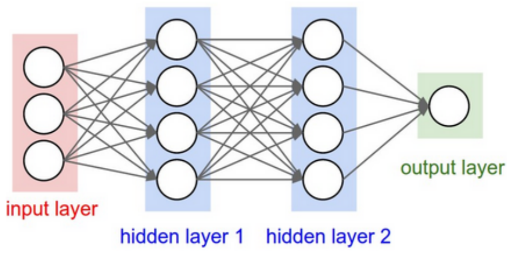


Figure 3: A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer.

4.2 Training the model

Deep feedforward neural networks are **supervised learning** algorithms, meaning that the models are trained with a dataset containing features, but each example is also associated with a **label** or **target** ([2]). First, the dataset should be split between a training set, a validation set and a test set. The central challenge in machine learning is that the model must perform well on new, previously unseen inputs—not just those on which the model was trained([2]). We measure how well the model performs by setting a **loss function** that the model has to minimize. Hence we will have a training error, a validation error and a test error measured by this loss function. The objective is to have the lowest possible test error.

We will try to do so by minimizing the training error which should act as an estimator of the test error. To minimize the training error, nearly all of deep learning is powered by one very important algorithm : **stochastic gradient descent** or SGD ([2]) with **backpropagation**. SGD is an extension of the Gradient Descent algorithm. First we initialize the parameters of the model (the weight matrices W from the previous paragraph) randomly. Then, in order to minimize the loss function L according to the parameters W, we apply iteratively the following steps until the loss function is below a threshold or until a predefined number of iterations is reached:

- Sample a mini-batch of examples from the training dataset
- Input those examples into the model to do a forward pass and compute the average loss for those examples
- The partial derivatives $\frac{\partial L}{\partial W_i}$ of L with respect to the weight matrix W_i of layer i are calculated for each layer, using the back-propagation algorithm which allows to propagate backwards the differential of L using the chain rule:

$$\frac{d}{dx}\left[f\left(u\right)\right] = \frac{d}{du}\left[f\left(u\right)\right]\frac{du}{dx}$$

- For each layer i, update all the weights in W_i using the formula $\mathrm{W}_i = W_i - \lambda \frac{\partial L}{\partial W_i}$, where λ is the learning rate which is an hyperparameter of the model to be optimized manually.

In order to reduce the risk of overfitting due to the large capacity of the model, **regularization** should be applied. According to [2], regularization is any modification we make to a learning algorithm that is intended to reduce its test error but not its training error. Regularizing is as important as minimizing the loss function. It is usually done by adding a regularizing term to the loss function, for example in the case of weight decay, we will add the term $\alpha \mathbf{W}^{\intercal}\mathbf{W}$ to express a preference for the parameters with smaller L^2 norm, α being a value chosen ahead of time (another hyperparameter) that controls the strength of our preference for smaller values of the weights.

An application of these deep feedforward models for computer vision has been implemented for the MNIST dataset on the Tensorflow documentation website [4]. The MNIST dataset consists of images of handwritten digits. It is split into three parts: 55,000 data points of training data, 10,000 points of test data, and 5,000 points of validation data. Each image in MNIST has a corresponding label, a number between 0 and 9 representing the digit drawn in the image. The task consists in producing, for each image, a probability distribution on the set {0, 1, … 9} corresponding to the likelihood of each number being drawn on that image.

The proposed model in [4] is a fully connected feedforward neural network consisting of one input layer and one output layer followed by a softmax classifier. The model is called fully connected since all the neurons of the input layer are connected to all the neurons of the output layer through a weight. The scores of the output layer are obtained by a matrix multiplication plus a bias term. In vectorial notation, if the input data is X and the weight matrix W, the operation can be written as follows:

$$\mathrm{score} = \mathrm{WX} + \mathrm{B}$$

The score of each neuron is injected as input into the softmax classifier to obtain a probability distribution over the {0,…,9} space. The softmax classifier is well suited for this task of building a probability distribution since its definition is :

$$\forall i \in \{0,..,9\}, softmax(score_i) = \frac{exp(score_i)}{\sum_j exp(score_j)}$$

The output of this function is then a list of scalars between 0 and 1 for each class, which can be interpreted as probabilities.

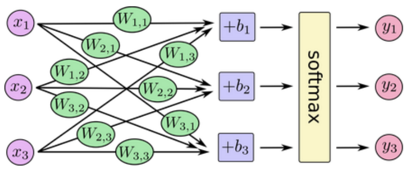


Figure 4 : Fully connected feedforward neural network implemented for the MNIST dataset

This very basic model achieves a test accuracy of 92%. However state of the art models achieve more than 99% accuracy on this dataset. Those models are based on convolutional networks.

4.3 Convolutional networks

One of the first models using only feedforward neural networks with backpropagation was the model developed by Lecun Y. and al who introduced their convolutional network applied to a zipcode number recognition problem [9].

According to [2] convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. [9] explains further the motivation of such an operation. Indeed, a traditional fully connected feedfoward neural network with enough discriminative power for the task of recognizing the digits would have far too many parameters to be able to generalize correctly. Hence, the connections between the neurons have to be restricted. Inspired by the usual practice in computer vision to detect and combine local features when doing shape recognition, [9] made their model do the same by constraining the connections in the first few layers to be local. Having in mind that a feature detector which is useful on one part of the image, is likely to be useful on other parts of the image as well, they came up with the solution to scan the input image with a single neuron that has a local receptive field, and store the states of this neuron in corresponding locations in a layer called a **feature map.** This is equivalent to performing a convolution with a small kernel. A feature map is then a plane of neurons whose weights are constrained to be equal since they perform the same operation on different parts of the image. The number of feature maps generated is called number of **channels** of this convolution.

This trick reduced significantly the number of free parameters and allowed the model to learn, through backpropagation, to extract the relevant kinds features for each problem instead of doing it by hand. [9] stacked multiple feature maps in order to give flexibility to the model to extract different features from the same image. They then re-injected the resulting feature maps as inputs for another convolution operation in order to extract features of increasing complexity and abstraction.

Another remark was that higher level features required less precise coding of their location. Hence, each convolution operation leading to a feature map was followed by a another layer performing a local averaging and subsampling on this feature map in order to reduce its resolution, which increased the robustness of the model to distortions and translations. These types of layers are called pooling layers in recent models. Together with average-pooling, max-pooling is currently often implemented after a convolution layer [2].

At the end of the model was a series of traditional fully connected neuron layers connected lastly to a classifier.

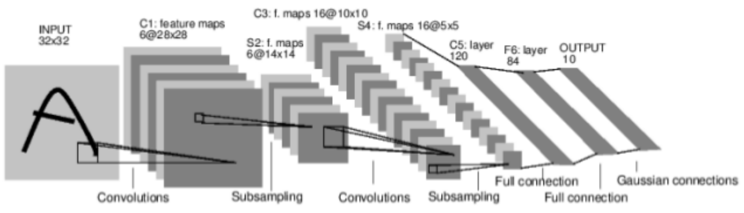


Figure 5 : LeNet-5 convolutional neural network, taken from LeCun et al., 1998

On the whole test set (2007 handwritten plus 700 printed characters), [9] achieved an error rate of 3.4% and all the classification errors occurred on handwritten characters.

The task seen so far – MNIST dataset and Zipcode dataset- are all based on recognizing scanned handwritten digits. It would be interesting to document whether specific work has been done in deep learning for recognizing text in natural scenes.

4.4 A unified convolution network for the natural text recognition task

[5] focuses on resolving the task of recognizing multi-digit numbers from Street Views panoramas. While reminding the complexities of reading characters in a natural scene, [5] points out that for those reasons, traditional approaches have solved that problem by separating the overall task into 3 sub-tasks : localization, segmentation and recognition.

A year before [5], a step towards unifying those tasks had been done by [10] who proposed to approach the problem using a convolutional network as feature extractor for a system that performs object detection and localization. However, the system as a whole was larger than the neural network portion trained with backpropagation, and had special code for handling the task of coming up with region proposals.

[5] proposes a fully unified learned model to localize, segment and recognize multi-digit numbers from street level photographs, with no need for a separate component to propose region proposals or provide a higher level model of the image.

The base assumption of the model is that the number of digits is not greater than a constant N (equals 5 in the paper). For each image, the number to be identified is a sequence of digits, $\mathrm{s} = \mathrm{s}_1, s_2, \ldots, s_n$ plus an additional number l representing the length of the number. Hence, the output of the model is a conditional probability distribution of a vector S of N+1 random variables :

- L (the number of digits in the image with possible values ranging from 0 to N plus one additional value to account for the case where L is greater than N)
- $\mathrm{S}_1, S_2, .., S_N$, which is a sequence of N random variables, with S_i representing the $\mathrm{i}$eth digit in the given image if it exists. Each of those random variables can take values ranging from *0* to *9.*

The random variables are assumed to be independent. They are discrete. The goal is then to learn a model of $P(S|X)$ by maximizing $logP(S|X)$ on the training set, X being the input image. The random variables being discrete, it is possible to use a separate softmax classifier for each of them and apply backpropagation separately for each of them.

For the inference part, at test time, the task is to predict

$$\mathrm{s} = (\mathrm{l}, \mathrm{s}_1, ..., s_l) = \mathrm{argmax}_{L,S_1,...,S_L} logP(S|X)$$

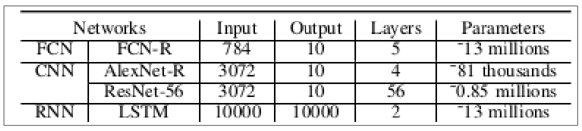
Given that the random variables are independent, this quantity can be maximized by maximizing separately the log-likelihood for each random variable. On public Street View House Numbers dataset, the best model in [5] has obtained a sequence transcription accuracy of 96.03%.

The results of the model also show a positive correlation between the depth of the model (number of layers) and its accuracy. This leads to the main issue concerning deep feedforward neural network models : resource consumption. Although the progress achieved in the hardware industry has led to the revival of deep learning with massive usage of GPUs, deep learning applications are still challenging to deploy into everyday devices like cellphones, personal computers, robots etc…

In terms of computational complexity, common convolutional network architectures like AlexNet requires about 2.5M operations per image, and close to 10 million operations for Resnet. Other famous architectures like VGG requires between 30 to 40 million operations per image, with 16 to 19 layers, more than 100 million parameters and 96MB/image of total memory needed to predict the class of only one image. Hence the need for fast and efficient convolutional networks.

4.5 Fast and efficient convolutional networks

The first way to speed up inference is the **use of GPU** and **parallelization** of the operations. In [11], the impact of running the deep learning algorithms on GPU and the impact of parallelization are measured during training time (number of batches of images treated per second). [11] carries out a benchmark on a few common deep learning algorithms for computer vision. The chosen algorithms were the following (input/output is the dimension of the input/output. An input of dimension 3072 correspond to a 32 by 32 pixels color image : 32x32x3) :



Networks		Input	Output	Layers	Parameters
FCN	FCN-R	784	10	5	~13 millions
CNN	AlexNet-R	3072	10	4	~81 thousands
	ResNet-56	3072	10	56	~0.85 millions
RNN	LSTM	10000	10000	2	~13 millions

Table 1 : Architecture of the deep learning models tested in [11]

As a comparison, the best algorithm obtained in [5] contained 11 hidden layers. In [11], the size of the images used as input were 54x54x3. For the AlexNet architecture running on Torch, the gain (with no parallelization) from desktop CPU (Intel i7-3820 CPU @ 3.60GHz) to GPU (NVIDIA Telsa K80 @ 562MHz with Kepler architecture) is a 97% speedup of training time. In terms of parallelization, when AlexNet is run with Torch on 4 K80 GPUs, a speedup of 67% is observed compared to the usage of just one GPU. However those gains involve only training time but not inference time.

Another method to achieve gains during inference time is proposed in [12]: it is about **pruning the network**. It is based on the idea that in deep learning algorithms, many of the parameters end up to be redundant. The pruning method removes redundant connections and learns only important connections. After an initial training phase, [12] prunes the model by removing all connections whose weights are lower than a threshold. They then retrain the sparse network so that the remaining connections can compensate for the removed connections. The phases of pruning and retraining may be repeated iteratively to further reduce network complexity. Their results show that they are able to prune out up to 95% of the parameters before experiencing significant decrease in accuracy:

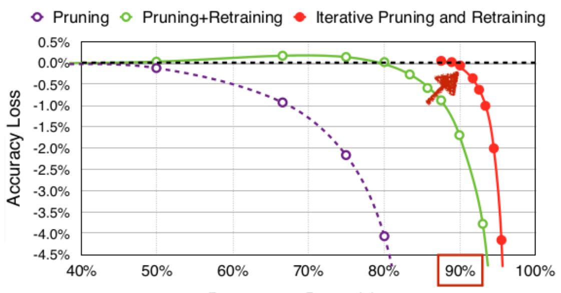


Figure 6 : Impact of pruning on a neural network

On top of pruning, [12] proposes methods to reduce the number of bits required to represent each weight. For instance, **weight sharing** proposes to group the weights into groups of similar value. Two weights will belong to the same group if and only if they differ only by a small precision number. Each group of weights will share only one single value. Thus for each weight, we need to store only a small index into a table of shared weights. Below is an illustration from [12] explaining the process. The groups of weights are materialized by their color. During the Stochastic Gradient Descent update, all the gradients are grouped by the color and summed together, multiplied by the learning rate and subtracted from the shared centroids from the previous iteration.

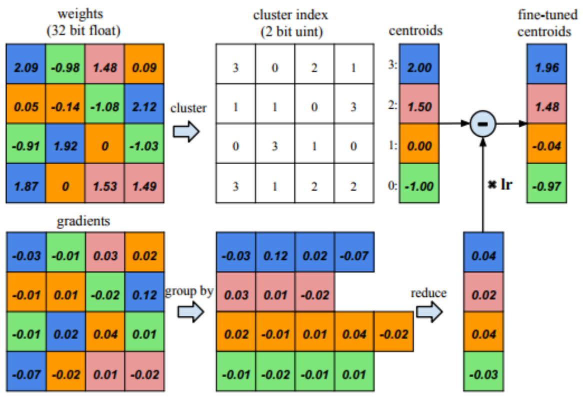


Figure 7 : Trained quantization by weight sharing (top) and centroids fine-tuning (bottom).

After implementing weight sharing, the weights become discrete, which mean that it is possible to use 4 bits to represent each weight without losing accuracy. When combining pruning and quantization, [12] shows that the network can be compressed down to 3% of its original size with no loss of accuracy.

In [13], another method called **low-rank approximation** is used for the convolution layers. It is based on the observation that the response at a position of a convolutional feature map approximately lies on a low-rank subspace. So if the response at one position of the feature map is $\mathrm{y} = \mathrm{Wx} + \mathrm{b}$, with $\mathrm{y} \in \mathbb{R}^d$, this convolution can be decomposed into two convolutions $\mathrm{y} = \mathrm{PW}'x + b'$ *(with* $\mathrm{W}'x \in \mathbb{R}^{d'}$ *and d > d')* which will have a lower combined complexity than the initial convolution. The matrix decomposition can be obtained by using Singular Value Decomposition. So this method entails to modify the architecture of the model by breaking down any low-rank convolution of filter size k and number of channels d into one convolution of same filter size and d' number of channels, plus another 1x1 convolution to scale up to d channels :

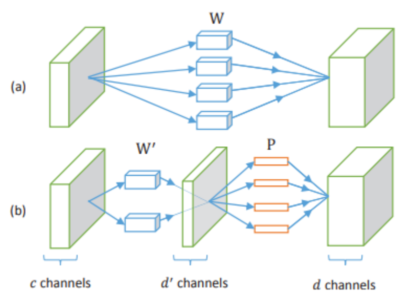


Figure 8 : Low rank approximation principle :
(a) An original layer with complexity *O(d.k².c)*
(b) An approximated layer with complexity reduced to *O(d'.k².c) + O(d.d')*

Finally, [14] has developed an algorithm based on **Recurrent Neural Networks** (RNNs) that is capable of extracting information from an image or video by adaptively selecting a sequence of regions or locations and only processing the selected regions at high resolution. It is called **Recurrent Attention Model**.

The attention problem is formulated as the sequential decision process of a goal-directed agent interacting with a visual environment. The system is trained using reinforcement learning methods which consist of adjusting the agent behavior based on a system of state/actions and rewards. Based on a given state, the agent takes an action and receives a reward for that action. Based on the magnitude of the reward, the agent modifies its behavior.

In this case, the state is the set glimpses taken so far in the image (the sub-regions already visited). The action consists in choosing the *(x,y)* coordinates (center of the glimpse) of where to look next in the image. The reward is positive if the image is correctly classified in the end otherwise it is zero.

On the MNIST dataset, the algorithm achieves accuracy results which are comparable to state of the art models. More interestingly, the overall capacity and the amount of computation of the recurrent attention model does not change from 60 × 60 images to 100 × 100 images, whereas the hidden layer of the standard convolutional networks which are connected to the linear layer grows linearly with the number of pixels in the input.

5 Deep Learning frameworks

In terms of tools, deep learning algorithms are usually developed on Python or R. However, a few frameworks have been developed to facilitate building models. The main ones are **TensorFlow** (from Google) and **PyTorch** (from Facebook). Those tools provide a more abstracted way of building complex deep learning models by easily stacking layers of different kinds (convolution layers, affine layers, easy choice of non-linear functions and of loss functions etc). On top of that, in the background, those tools build a computational graph corresponding to the model which is assembled, allowing them to automatically compute the gradient and perform the backpropagation. Those tools also take care of running everything efficiently on GPU.

The difference between those two tools is mainly that TensorFlow builds a static computational graph while PyTorch maintains a dynamic computational graph. Hence development on TensorFlow requires two phases: first, since the computational graph is static, we need to describe its nodes upfront and then in a second phase we run the graph multiple times. In PyTorch, the computational graph is built dynamically, so that there is no need to predefine the it upfront. Each forward pass rebuilds a new graph. The main advantage of static computational graphs is that they can be serialized, exported and run without access to the original code. Dynamic graphs cannot be separated from the code which generates them. However, they greatly facilitates coding many instructions like loops, conditional operations etc… Hence they make it easier to be creative in building models using RNNs, or combining together different existing models.

Besides, TensorFlow benefits from a larger community and has many high level libraries of top it (some of those libraries are Keras, TFLearn, Sonnet). In addition, TensorFlow has some pre-trained models (with the Keras or TF-Slim libraries) in order not to start from scratch, but from a model whose weights have already been learned from solving a similar problem. This process, called transfer learning, can speed up the training phase. However, it may be tricky to chose among all the available libraries. On its side, PyTorch is shipped with one high level library called *modules* which makes it unnecessary to use a third party library. This library also contains pre-trained models.

To wrap-up, we could say that PyTorch is more oriented towards research and allows to be creative and build more compact code. On the other hand, TensorFlow is very useful in deployment scenarios, in which we would need to make the graph run on different devices.

6 Conclusion and perspectives

This study has shown that the problem of finding a fast algorithm for recognizing bib numbers in natural scenes is feasible. It has allowed to find practical and theoretical tools for the resolution of a few difficult points like achieving a unified algorithm for the task of detection, segmentation and recognition and the problem of achieving a relatively fast algorithm. The challenge and contribution of our work will be to manage to build a unified model which would have sufficient accuracy and practicable size and speed to work as well on GPU chips as on standard CPU.

References

1 ‘*Fast Detection of Text in a Natural Scene using Blobs*’.

2 Goodfellow I., Benjio Y., Courville A., *Deep Learning*, , MIT Press, 2016.

3 Rane M.A., ‘*Fast Morphological Image Processing on GPU using CUDA* ’, *IEEE Transactions on Communications*, 40 (8) : College of Engineering, Pune , 2013.

4 www.tensorflow.org, MNIST For ML Beginners, 2017.

5 Goodfellow I., Bulatov Y., Ibarz J., Arnoud S., Shet V. ‘*Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks’*, ICLR, 2014.

6 Matas J., Chum O., Urban M, Pajdla T., *‘Robust Wide Baseline Stereo from Maximally Stable Extremal Regions’,* BMVC, 2002.

7 Dalal N., Triggs B., *‘Histograms of Oriented Gradients for Human Detection’, CVPR,* 2005.

8 Wang K., Babenko B. and Belongie S., *‘End-to-End Scene Text Recognition’, ICCV,* 2011.

9 Lecun Y., Boser B., Denker J. S., Henderson D., Howard R. E., Hubbard W., Jackel L. D., *‘Backpropagation applied to Handwritten Zip Code Recognition’,* 1989.

10 Girshick, R., Donahue, J., Darrell, T., and Malik, J. *‘Rich feature hierarchies for accurate object detection and semantic segmentation’*. Technical report, arXiv:1311.2524, 2013.

11 Shaohuai S., Qiang W., Pengfei X., Xiaowen C.,Benchmarking *‘State-of-the-Art Deep Learning Software Tools’,* arXiv:1608.07249v7, 2017.

12 Han S., *‘Efficient Methods And Hardware For Deep Learning’,* Stanford University, 2017.

13 Zhang X., Zou J., Ming X., He K., Sun J., ‘*Efficient and Accurate Approximations of Nonlinear Convolutional Networks’*, CVPR 2015.

14 Mnih V., Graves N. H. A., Kavukcuoglu K., ‘Recurrent Models of Visual Attention*’*, Google Deep Mind, ArXiv: 1406.6247, 2014.