UNIVERSIDADE ESTADUAL PAULISTA
"JULIO DE MESQUITA FILHO"

Optical Character Recognition Using Deep Learning

Claudio Filipi Gonçalves dos Santos

São José do Rio Preto
2018

Claudio Filipi Gonçalves dos Santos

Optical Character Recognition using Deep Learning

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação – Área de Concentração em Computação Aplicada, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Financiadora: CAPES

Orientador: Prof. Dr. Fabricio Aparecido Breve

São José do Rio Preto
2018

Santos, Claudio Filipi Gonçalves dos.
Optical Character Recognition using Deep Learning /Claudio Filipi Gonçalves dos Santos. -- São José do Rio Preto, 2018
83 p. : il., tabs.

Orientador: Fabricio Aparecido Breve
Dissertação (Mestrado) – Universidade Estadual Paulista “Júlio de Mesquita Filho”, Instituto de Biociências, Letras e Ciências Exatas

1. Computação. 2. Redes neurais. 3. Inteligência artificial. 4. Computadores neurais. 5. OCR. I. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. II. Título.

CDU – 518.72

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Claudio Filipi Gonçalves dos Santos

Optical Character Recognition using Deep Learning

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação – Área de Concentração em Computação Aplicada, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Financiadora: CAPES

Comissão Examinadora

Prof.(a). Dr.(a). Fabricio Aparecido Breve
UNESP- Câmpus de Rio Claro
Orientador

Prof.(a). Dr.(a). João Paulo Papa
UNESP- Câmpus de Bauru

Prof.(a). Dr.(a). Ricardo Cerri
Universidade Federal de São Carlos

São José do Rio Preto
26 de abril de 2018

Resumo

Detectores óticos de caracteres, ou *Optical Character Recognition (OCR)* é o nome dado à técnologia de traduzir dados de imagens em arquivo de texto. O objetivo desse projeto é usar aprendizagem profunda, também conhecido por aprendizado hierárquico ou *Deep Learning* para o desenvolvimento de uma aplicação com a habilidade de detectar áreas candidatas, segmentar esses espaços dan imagem e gerar o texto contido na figura. Desde 2006, *Deep Learning* emergiu como uma nova área em aprendizagem de máquina. Em tempos recentes, as técnicas desenvolvidas em pesquisas com *Deep Learning* têm influenciado e expandido escopo, incluindo aspectos chaves nas área de inteligência artificial e aprendizagem de máquina. Um profundo estudo foi conduzido com a intenção de desenvolver um sistema OCR usando apenas arquiteturas de *Deep Learning*. A evolução dessas técnicas, alguns trabalhos passados e como esses trabalhos influenciaram o desenvolvimento dessa estrutura são explicados nesse texto. Essa tese demonstra com resultados como um classificador de caracteres foi desenvolvido. Em seguida é explicado como uma rede neural pode ser desenvolvida para ser usada como um detector de objetos e como ele pode ser transformado em um detector de texto. Logo após é demonstrado como duas técnicas diferentes de *Deep Learning* podem ser combinadas e usadas na tarefa de transformar segmentos de imagens em uma sequência de caracteres. Finalmente é demonstrado como o detector de texto e o sistema transformador de imagem em texto podem ser combinados para se desenvolver um sistema OCR completo que detecta regiões de texto nas imagens e o que está escrito nessa região. Esse estudo demonstra que a idéia de usar apenas estruturas de *Deep Learning* podem ter performance melhores do técnicas baseadas em outras áreas da computação como por exemplo o processamento de imagens. Para detecção de texto foi alcançado mais de 70% de precisão quando uma

arquitetura mais complexa foi usada, por volta de 69% de traduções de imagens para texto corretas e por volta de 50% na tarefa ponta-à-ponta de detectar as áreas de texto e traduzi-las em sequência de caracteres.

Palavras-Chave: Aprendizado Profundo, Redes Neurais Convolucionais, Redes Neurais Recorrentes, OCR

Abstract

Optical Character Recognition (OCR) is the name given to the technology used to translate image data into a text file. The objective of this project is to use *Deep Learning* techniques to develop a software with the ability to segment images, detecting candidate characters and generating text that is in the picture. Since 2006, *Deep Learning* or hierarchical learning, emerged as a new machine learning area. Over recent years, the techniques developed from deep learning research have influenced and expanded scope, including key aspects of artificial intelligence and machine learning. A thorough study was carried out in order to develop an OCR system using only *Deep Learning* architectures. It is explained the evolution of these techniques, some past works and how they influenced this framework's development. In this thesis it is demonstrated with results how a single character classifier was developed. Then it is explained how a neural network can be developed to be an object detector and how to transform this object detector into a text detector. After that it shows how a set of two *Deep Learning* techniques can be combined and used in the task of transforming a cropped region of an image in a string of characters. Finally, it demonstrates how the text detector and the Image-to-Text systems were combined in order to develop a full end-to-end OCR system that detects the regions of a given image containing text and what is written in this region. It shows the idea of using only *Deep Learning* structures can outperform other techniques based on other areas like image processing. In text detection it reached over 70% of precision when a more complex architecture was used, around 69% of correct translation of image-to-text areas and around 50% on end-to-end task of detecting areas and translating them into text.

Keywords: Deep Learning, Convolutional Neural Network, Recurrent, Neural Network, OCR

List of Figures

Contents

1 Introduction

The purpose of this work is to create a survey about general text detection and word recognition techniques, so when these tasks are combined, they create a robust framework for text extraction from images, allowing the computer to read information.

The OCR (*Optical Character Recognition*) technology is used around the world to solve several problems that are very easy for humans but take a long time or need a lot of people working to finalize a job, such as passport data reading and car plate detectors in order to automatic generate fines. In the listed cases, the reading window is very specific and there is not a lot of variation on size, font and format of the characters. Technically speaking, it tends to be easier to develop the software necessary to read the text in these contexts.

Traditional OCR frameworks usually work very well for pure text images, like printed version or photo of black-and-white documents.[1] This is because there is not a lot of "pollution" in this kind of image, such as huge variation of colors, other objects in the scene or inclination and translation of the words. This extra information can compromise the final result of detecting text only, adding some characters that do not belong to the scene or even not detecting some words because it is color or border mixed.

A robust OCR system is very useful on several tasks. For example, autonomous cars need a high performance framework to detect anomalies on the road (ex: to read the speed change on an area or to detect plaques on the highway). Other interesting application is real-time translation of printed books.

To accomplish a relevant result on the described tasks, several techniques used on computer vision and artificial intelligence are necessary. Currently, there is a specific field on Machine Learning which has been

[1] https://www.abbyy.com

achieving very good results on both jobs: the *Deep Learning*. It consists on several techniques that allows the computer to learn different tasks like the brain does.

Deep learning is a machine learning branch based on a set of algorithms which attempt to model high-level abstractions for data processing using multiple layers, with complex structures composed of several linear and/or non-linear transformations[Bishop, 2006].

Many OCR systems have serious problems on detecting text on real scenes like photographs from places or even more simple figures like digital text images from *spam*. The purpose of this work is to create a survey about general text detection and word recognition techniques, so when these tasks are combined, they create a robust framework for text extraction from images, allowing the computer to read information.

On Chapter 2 it is presented a more specific discussion about *Deep Leaning*. It shows some available architectures and describes how each one of them works.

On Chapter 3 it is explained how some more complex structures can be assembled from Chapter 2 to solve different types of problems, including image denoising and object localization.

On Chapter 4 a more specific description of the proposed framework is available. It defines how each of the techniques reported on Chapter 3 are intended to work and when each one of them is applied. This unit is divided in two main parts: what was the initial idea and how was the framework developed using only *Deep Learning* techniques.

On Chapter 5 it is again divided in two main components: the initial results on a competition available online using a technique of *Deep Learning* and the results on three main tasks proposed by this work.

In the following section, a deep study about *Deep Learning* is shown detailing about how the structures are trained and more about their behavior.

2 Deep Learning

One branch of Machine Learning which has been developed along last few years is *Deep Learning*. The name comes back from middle-2000's from Hinton's paper[Hinton, 2007] but become a really *buzz* word in 2012 after Google researches advance on image recognition[Le et al., 2012].

The objective of this field is to copy the way the brain learns and execute tasks, based on several layers of artificial neural network that allows the computer to be trained to simulate how animals solve problems.

In this Chapter it is firstly explained how this type of network works in general. After that it explains how *Deep Learning* is trained and some details about Gradient Descent and its upgrades. Then it details deeper how Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) performs.

2.1 Artificial Neural Network (ANN)

As said before, there are problems which could be solved faster if there was a way to mimic how human or other animals' brain solve issues like image interpretation or transform sounds in useful information. Based on threshold calculus, McCulloch and Pitts[Mcculloch and Pitts, 1943] created what is considered the first computational model inspired in the brain to solve problems. Figure 1 shows its structure and how it basically works.

From this publication, several other studies[Burks and Wang, 1957, Farley and Clark, 1954, Rosenblatt, 1958] were published, giving the idea that in little time it would be possible to solve several types of problems using neural network based techniques. But in 1969 Minsk and Papert[Minsky and Papert, 1969] proved two important points for neural networks:

- A basic perceptron[Rosenblatt, 1958] could not process exclusive-or

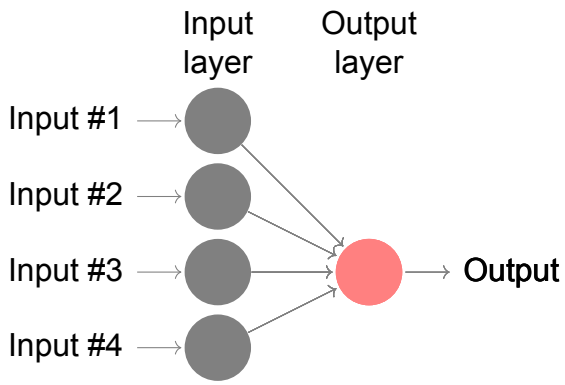

Figure 1: An example of a simple Perceptron

(XOR) circuit because it involves non-linear functions. This is important because if this computation could not be done by neural networks, it would imply that other problems (non-linear function-based problems) could not be solved using this technique.

- The process power required to handle the computation for a neural network to solve bigger problems was huge and there was not a computer with enough power available yet.

These problems lasted until 1975 when Werbos[Rumelhart et al., 1988] created the *Backpropagation* algorithm. Werbos' approach was good not only because it could solve the XOR circuit problem but it could train a bigger neural network much faster. This technique is still used in neural network training task.

In the 1980's, John Hopfield presented the first popular model of recurrent neural network[Hopfield, 1982] that provided a model for understanding human memory.

In the 1990's, Yan LeCun showed a new type of neural network architecture[LeCun et al., 2001] that could successfully recognize handwritten digits with very good precision. Based on the mammals' visual cortex, it used several layers which task was to get as many features as possible

from a given image and then this image could be classified in a 0-9 digit.

From 2000, other architectures[Vincent et al., 2010] appear, using different approaches. Csáji [2001] proved the Universal Approximation Theorem, which says that a neural network can represent any function when training data represents all the domain and backpropagation algorithm with Gradient Descent is used. Mathematically speaking, an ANN is a composed function where each neuron of a given layer represents a part of this composition. So any function represented by the input domain can be represented when the neural network has enough neurons in its hidden layer (in the study, it says one layer is enough to represent any transformation) and the training algorithm is executed.

2.2 Training and Optimization

Any artificial neural network usually has two main steps: the training, when the structure tries to learn the best values for each parameter based on the training data; and the execution, where after gaining the values, the network is able to execute the desired task.

Execution part is straightforward: the network receives the input values, each layer calculates a transformation and in the end it provides the output based on the middle layer calculation. The trick section is the training: how exactly should the neural network learn the correct values for each layer? The most used technique nowadays[Abadi et al., 2015, Chollet, 2015] is the backpropagation algorithm. It basically works following these steps:

- The network calculates its output based on the training data(Forward pass);
- In this step the data is labeled, so it is possible to calculate the error;
- Based on the learning rate, this error is multiplied by a factor;

- The variation rate is calculated on the final layer (gradient) based on the derivative of its activation function and the value of error;
- This process is made layer by layer until the first hidden layer (back-propagation of error).

The difference is how the error and the learning rate is calculated. Gradient Descent is still largely used however new techniques have appeared in last years. In this section it will be detailed the differences between them and why ADAM Optimizer[Kingma and Ba, 2014] was chosen in this work.

2.2.1 Gradient Descent

Gradient Descent was the first function used to enhance the parameters of a neural network inside the backpropagation algorithm. For each epoch, it evaluates the error based on the available data and then it updates the parameters based on the error and on the learning rate. Algorithm 1 describes how the code looks like.

Algorithm 1: Gradient Descent Algorithm

Data: epochs, lossFunction, learningRate, data, parameters
Result: Trained parameters value

```
1 while training < epochs do
2     paramsEpoch = evaluate(lossFunction, data, parameters)
3     parameters = parameters - learningRate * paramsEpoch
4     training++
```

This code has two main problems: if $data$ is bigger than memory available, it will generate an out-of-memory error; the other is because the parameters are not update very frequently, so it tends not to converge so quickly as expected. To solve them both, a stochastic version of the code is used. The parameters are updated for each instance of $data$ instead of calculated

for only each epoch. It is described on Algorithm 2

Algorithm 2: Stochastic Gradient Descent Algorithm

Data: epochs, lossFunction, learningRate, data, parameters
Result: Trained parameters value

```
1 while training < epochs do
2     shuffle(data);
3     for example in data do
4         paramsEpoch = evaluate(lossFunction, example,
            parameters) parameters = parameters - learningRate *
            paramsEpoch
5     training++
```

This is still not the best solution. It may cause the parameters to converge in a local minimum instead of the global minimum. To solve this last problem mini batches of data are used: it updates the parameters for a mini set of $data$ rather than for each instance. It makes the function fluctuates more, avoiding the situation. Algorithm 3 explains its work.

Algorithm 3: Mini-Batch Stochastic Gradient Descent Algorithm

Data: epochs, lossFunction, learningRate, data, parameters, batchSize
Result: Trained parameters value

```
1 while training < epochs do
2     shuffle(data);
3     for batch in createBatch(data, batchSize) do
4         paramsEpoch = evaluate(lossFunction, batch, parameters)
            parameters = parameters - learningRate * paramsEpoch
5     training++
```

Most available Deep Learning frameworks [Abadi et al., 2015, Chollet, 2015] makes use of Mini-Batch Stochastic Gradient Descent even when it calls only Stochastic Gradient Descent (SGD).

2.2.2 Momentum

In order to improve the velocity of the function convergence, Momentum (from physics) was adapted to work with Gradient Descent. Imagine there is a ball falling from a hill, it accumulates momentum every time it goes down, increasing the speedy. This is the same principle used on this algorithm.

For each interaction, it calculates the error like SGD and then add a fraction(γ) of the previous interaction of the looping. This value is usually 0.9. Algorithm 4 shows more details

Algorithm 4: Mini-Batch Stochastic Gradient Descent Algorithm with Momentum

Data: epochs, lossFunction, learningRate, data, parameters, batchSize
Result: Trained parameters value

```
1 v_t = 0
2 while training < epochs do
3     shuffle(data);
4     for batch in createBatch(data, batchSize) do
5         paramsEpoch = evaluate(lossFunction, batch, parameters) +
           γ * v_t;
6         parameters = parameters - learningRate * paramsEpoch ;
7         v_t = paramsEpoch;
8     training++
```

2.2.3 Nesterov Momentum

The inclusion of Momentum showed a decreased of time needed for training. However, in 1983, Yurii Nesterov figured out a problem with this method: in some cases, the Momentum value can be very high near the

global minimum, making the parameters not converge on the ideal place. Imagine again the ball running in a hill, if it reaches the bottom too fast, it may go up again and get stuck on another local minimum.

To prevent this problem, he proposed another way to calculate this variation, making the jump before calculates the new Momentum value. Algorithm 5 shows the differences against Algorithm 4.

Algorithm 5: Mini-Batch Stochastic Gradient Descent Algorithm with Momentum

Data: epochs, lossFunction, learningRate, data, parameters, batchSize

Result: Trained parameters value

```
1 v_t = 0
2 while training < epochs do
3     shuffle(data);
4     for batch in createBatch(data, batchSize) do
5         paramsEpoch = evaluate(lossFunction, batch, (parameters -
            γ * v_t) ) + γ * v_t;
6         parameters = parameters - learningRate * paramsEpoch ;
7         v_t = paramsEpoch;
8     training++
```

It shows in line 4 the difference. By doing this change on values of parameters before evaluate, it helps to decrease the Momentum when it is about to get to a global minimum, avoiding the function to converge to a local minimum instead.

2.2.4 AdaGrad

Until now it is shown how to adapt the Momentum to prevent some types of problems. It is possible to adapt the learning rate too, speeding up the convergence.

AdaGrad (Adaptive Gradient) adapts the learning rate to each parameter, making the optimization in a hasty manner. It makes bigger changes for parameters that changes less and smaller jumps for those which changes more frequently. Algorithm 6 shows more details. It is possible to see the learning rate is not a value anymore but a list of values, one for each parameter.

Algorithm 6: AdaGrad Algorithm

Data: epochs, lossFunction, ϵ, data, parameters, batchSize, learningRates

Result: Trained parameters value

G = 0;
while *training* < *epochs* **do**
    $shuffle(data)$;
    for *batch in createBatch(data, batchSize)* **do**
        paramsEpoch = evaluate(lossFunction, batch, parameters);
        for *i in size(paramsEpoch)* **do**
            $G_i = paramsEpoch_i - parameters_i$;
            $G_i = G_i * decay$;
            $paramsEpoch_i = -(learningRates_i / \sqrt{(\sum_{n=0}^{i} G_{training,n}) + \epsilon})$;
    training++

2.2.5 AdaDelta

AdaGrad performs really well because it adapts the learning rate according to the frequency of parameters' update. However, it is easy to detect that in a long-term training, the gradient tends to go to 0, not optimizing the values anymore because of the division of sums of square roots sequentially. This problem is known as the "vanishing gradient". To solve this problem AdaDelta performs a slightly different approach, it limits the sum to the last x interactions (usually $x = 2$, meaning it uses only the last and the current calculated gradients). The algorithm can be seen in Algorithm 7.

Algorithm 7: AdaDelta Algorithm

Data: epochs, lossFunction, ϵ, data, parameters, batchSize, learningRates, decay

Result: Trained parameters value

G = 0;
while *training* < *epochs* **do**
    $shuffle(data)$;
    for *batch in createBatch(data, batchSize)* **do**
        paramsEpoch = evaluate(lossFunction, batch, parameters);
        for *i in size(paramsEpoch)* **do**
            $G_i = paramsEpoch_i - parameters_i$;
            $G_i = G_i * decay$;
            $paramsEpoch_i = -(learningRates_i / \sqrt{(\sum_{n=i-x}^{i} G_{training,i}) + \epsilon})$
    training++

2.2.6 ADAM Optimizer

AdaDelta is already a very good optimizer for *Deep Learning* architectures because it calculates individual learning rates for each parameter, calculates momentum and prevents decaying to 0 the value of the learning rate. However, in 2014 Diederik Kingma and Jimmy Ba realized there is still one thing that could speed up the learning: if the momentum is already calculated, it could be adapted to each parameter[Kingma and Ba, 2014].

ADAM (Adaptive Moment Estimation) follows these steps: for each parameter it first calculates its learning rate and then adjusts the momentum. It makes use of the benefits of AdaDelta and can control the variation rate by accelerating or delaying the velocity of the changes depending on values of β_1 and β_2. Algorithm 8 represents its functionalities.

Algorithm 8: ADAM Algorithm

Data: epochs, lossFunction, ϵ, data, parameters, batchSize, learningRates, decay, β_1, β_2

Result: Trained parameters value

$m_0 = 0$;
$v_0 = 0$;
while *training* < *epochs* **do**
    $shuffle(data)$;
    for *batch in createBatch(data, batchSize)* **do**
        t = training;
        paramsEpoch = evaluate(lossFunction, batch, parameters);
        $m_t = m_{t-1}/(1-\beta_1)$;
        $v_t = v_{t-1}/(1-\beta_2)$;
        paramsEpoch = paramsEpoch - $m_t * learningRates/(\sqrt{v_t} + \epsilon)$;
    training++

2.3 Deep Learning Structure

All the Deep Learning structures are basically neural networks with several layers. What differentiate these architectures from the traditional neural networks is the way they threat input data.

One characteristic found in several deep learning architectures[Hinton, 2007, LeCun et al., 2001, Vincent et al., 2010] is the internal transformation of the input data before generating the desired output information. Some layers work on the creation of several middle-input data based of the input data. This creation can be based on several types of transformation, such as image filters or denoising-trying techniques.

2.3.1 Feature Detection

To accomplish a relevant success rate, the brain's cortex needs to read and translate the signals (synapses) that comes from the feature-detecting neurons. Learning procedures like Backpropagation[Rumelhart et al., 1988] and ADAM optimization[Kingma and Ba, 2014] fits the weights to enhance the performance on the classification task. However, it needs labeling data to perform the neural network training.

The problems starts when there is not enough labeled data to perform this training, so some techniques other than only classification are desired. Suppose there are some layers of a neural network that are not trained for classifying data but, instead, it is trained to recognize only features of the input. It is expected these characteristic detectors will guide the final part of the network to take the appropriated actions according to the features detected.

These layers on the neural network are responsible to acquire the maximum possible data before the work of the generative layer. It is done by artificially creating copies of the input data and trying to highlight the most

relevant information. By doing this, the last layers can perform a more precise output because important information is easily detected.

2.3.2 Generative Model

The final section of a Deep Learning model is to generate the desired output according to the input.

It is expected, before this step is computed, the relevant features are detected, so the output will be relevant to the user. This final step can be trained in two different ways: using fine-tuning after the feature detector is trained or it can be trained together with the feature detector.

In some architectures like Deep Belief Networks(DBN)[Hinton, 2007] or Stacked Denoising Auto-Encoders(SdA)[Vincent et al., 2010] the training is performed in two steps. Initially the initial layers are induced to learn the features from the input data in an unsupervised way. Then in the second step, called fine-tuning, the entire network is trained to create output according to the label of each input.

In other architectures like Convolutional Neural Networks (CNN)[LeCun et al., 2001] the feature detector and the generative output training are performed together. In this specific neural network architecture, the filters of the convolutional layers and the generative layer are trained according to the input data and to the label of the input at once.

2.4 Multi-Layer Perceptron (MLP)

The very first implemented architecture of Deep Learning is the Multi-Layer Perceptron. It is compounded by several perceptrons that are organized in multiple layers, allowing the computer to learn abstraction from an input data. It uses supervised learning and can learn non-linear data

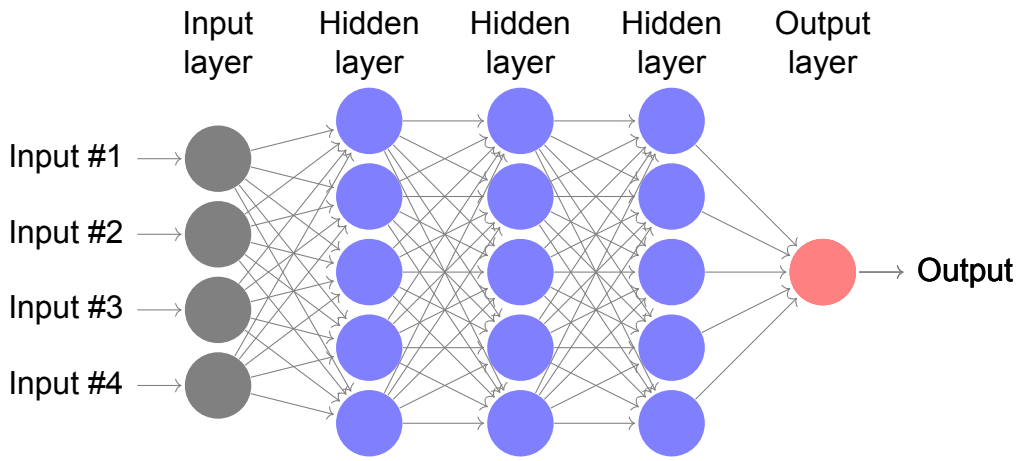

Figure 2: Example of a Multi-Layer Perceptron Architecture

models. It is considered the first *Deep Learning* architecture because it was the first structure to be used with more than two intermediate layers, which describes a Deep Neural Network architecture.

It is known as universal function approximator[Cybenko, 1989]. Because of this, it was largely used in initials speech and image recognition and other artificial intelligence softwares[Wasserman and Schwartz, 1988].

From the beginning of 2000's, new approaches have been developed and they have had more success than traditional MLP's. However, traditional MLP's are still often present as a step in Deep Learning approaches.

It is possible to see how an MLP basically works in the Figure 2. The input is presented to the first layer (left side), all values pass through every neuron on intermediate layers and then the final values are calculated. Each neuron is a structure called Perceptron [Minsky and Papert, 1969] which is usually used as a binary classifier (functions that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not) or as a regressor, giving a value as output.

Perceptrons were first described by Rosenblatt (1957) in 1957 and it

was so exciting by that time it was largely covered by the media.[2] It was wildly used as regressor and classifier until 1969 when Marvin Minsky and Seymour Papert attacks its problems[Minsky and Papert, 1969]. They mostly focused on the inability of a single neuron to approximate to a XOR function (known as a non-linear problem). The fact is with a single layer it is impossible to get satisfactory close to this type of function.

In order to solve this point Rumelhart et al. [1988] published the back-propagation algorithm[Rumelhart et al., 1988]. This technique allows multilayer perceptrons to interactively learn their parameters' values, granting the network to get near to a non-linear function.

2.5 Convolutional Neural Network (CNN)

Based on how the animal visual cortex works, the Convolutional Neural Network (CNN) works on small and overlapped regions that detect light and specific edge regions of the input data. It is largely used in the image processing task of detecting a given object or to classify the image. Figure 3 shows all the basic structures of a CNN.

CNN differs from a MLP because of the local connectivity of the input data from each layer. On a MLP, the perceptron from a given layer receives data from all the perceptrons from the previous layer. On a CNN, there are some different ways these connections are made:

- Convolution layers: the core of the CNN. A set of layers (also called kernels or filters) is responsible to generate a final output, usually a cube (but it is not uncommon to see 4-D or even higher dimension outputs). The most important point here is each layer has a small receptive field but it goes through all the input, generating a unique transformation on this data for each filter. After the neural network

[2] http://archives.newyorker.com/?i=1958-12-06#folio=044

training, each filter activates when it sees some specific type of feature at some spatial position of the input.

- Pooling layers: right after the work of the convolution layer, the pooling layer is responsible for down-sampling a given region. This is very interesting on the computation side because it critically reduces the dimension of the data, reducing the work of the processor. The most common function used here is the Max-Poolling function, it keeps only the most significant value of a given region, which has given the best results in several researches [Nielsen, 2016].

- Classification layers: all CNN work done until now has the function of facilitating the data classification. After all transformation of a given input, only the most relevant data is kept for classification. This layer is usually made of MLP or Support Vector Machines (SVM) that classifies the remaining data.

Some techniques are added to the CNN to increase the final results. Listed below are the most common ones:

- Dropout: it literally removes a number neurons from the classification layer during the training session. It is possible to see this technique helps to reduce overfitting because for each training session, if neurons of a certain region are dropped, the remaining ones are forced to learn more information. In addition, it reduces the amount of computation necessary for training, cutting down time for this job.

- Artificial Data Augmentation: depending on the job desired for a CNN, sometimes there is not enough labeled data available for a neural network to become useful. Artificial data augmentation uses the data available and perform variations which, even after the transformation, it continues to be the same type of data on the label. This technique also prevents the CNN to overfit during the training.

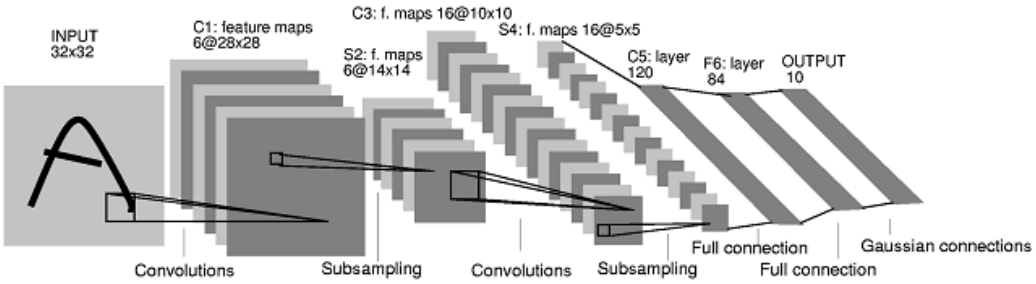

Figure 3: Example of a Convolutional Neural Network (CNN) [LeCun et al., 2001]

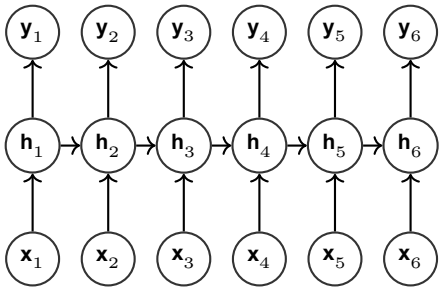

Figure 4: A simple Recurrent Neural Network

2.6 Recurrent Neural Network(RNN)

There are some cases the approaches described until here are not enough to give a good result. For example, it is hard to detect if a given object (Ex: a bike) appears in a video because theirs architecture works on static input. The problem here is these techniques do not work well on temporal series, like a video or an audio stream.

The Recurrent Neural Network (RNN) works by transforming the input in a time series representation where each timestep represents a segment of the data, solving this kind of problems. The connections between the neurons form cycles, creating a type of "memory" of what has or has not been on this neural network. In theory, this network can process long sequences of input data[Hochreiter and Schmidhuber, 1997]. Figure 4 shows how a RNN works.

2.6.1 Long-Short Term Memory - LSTM

The traditional RNN is very powerful by itself but it may be affected when it works with numerous steps. Suppose it is needed to work with the classification of a very long text (a large book for example). The traditional Recurrent Neural Network may not remember what is written on the beginning and just use the information at the end of this book. For this case, a slightly different but still powerful approach is necessary.

By 1997 Sepp Hochreiter introduced the Long-Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997], a new type of RNN cell with gateways for different situations. Figure 5 describes exactly how the entire neuron and each gate works.

The first step in the LSTM cell is to decide whether the information will be thrown away from the cell state or not. This decision is made by a sigmoid layer called the "forget gate layer". It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. The value 1 represents "completely keep this information" while a 0 represents "completely get rid of this information".

The next step is to decide what new information will be stored in the cell state. This is made in two steps. First, a sigmoid layer called the "input gate layer" decides which values will be updated. Next, a hyperbolic tangent (*tanh*) layer creates a vector of new candidate values, C_t which could be added to the state. In the next step, these two will be combined to create an update to the state.

It is needed to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, providing the necessary information to do it now. It multiplies the old state by f_t, forgetting the things decided to be forgotten earlier. Then it is added $i_t * C_t$. These are the new candidate values, scaled by how much it is decided to update each state value.

Finally, it is necessary to measure the output. This is based on the cell

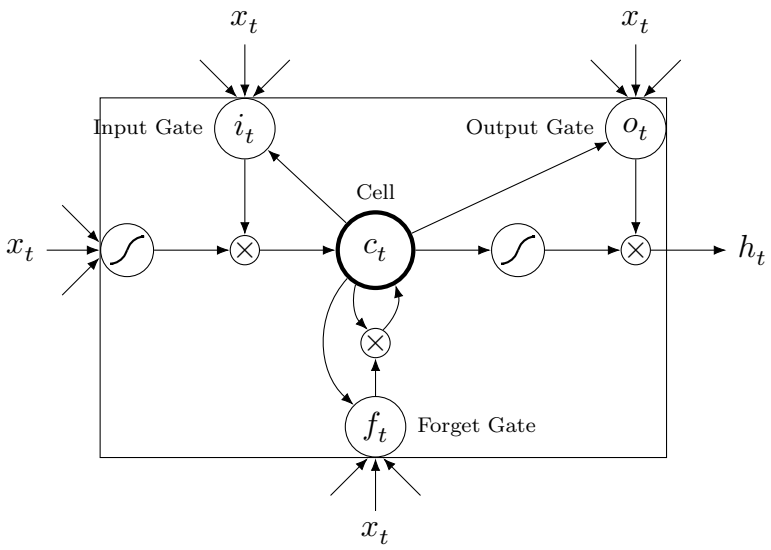

Figure 5: Image description of a LSTM cell

state but will be a filtered version. First, a sigmoid layer decides what parts of the cell state will be the output value. Then, the cell state is set through *tanh* (to push the values between −1 and 1) and multiply it by the output of the sigmoid gate, so that the neuron only output the parts calculated.

2.6.2 Gated Recurrent Unit - GRU

Another great variant of RNN which makes use of gates to improve performance is the Gated Recurrent Unit (GRU) introduced by Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau and Yoshua Bengio in 2014[Cho et al., 2014]. The idea is similar to LSTM however it has fewer gates and works slightly different. While in LSTM there is a gate to control the input, another gate to control what to forget and one last gate controlling the output, GRU lacks the last one: it just does not filter the output result calculated by the cell and the other two gates. Figure 6 illustrates better this difference: it is possible to realize there is no gate controlling in the end.

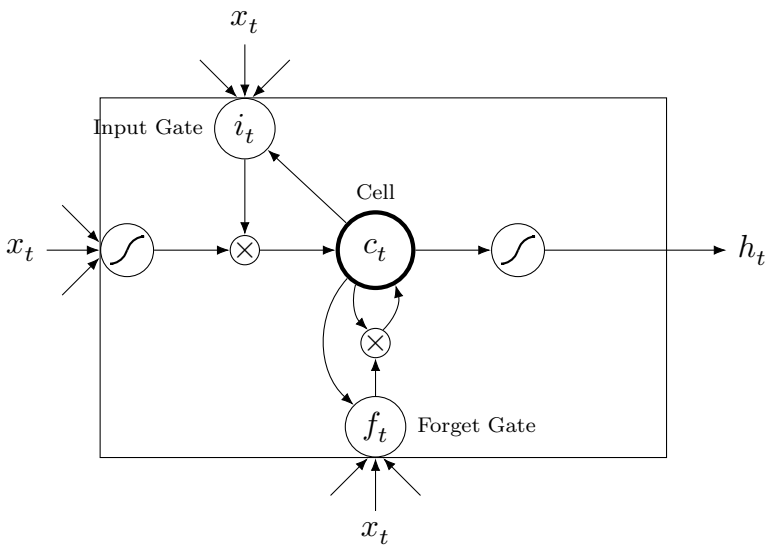

Figure 6: Image description of a GRU cell. The most significant difference is the ”output gate”: it lacks in GRU’s

This missing gate is very important in two different aspects:

- Computationally it is more interesting because it is one gate less to calculate the output and to train, so it is faster than LSTM;
- Some studies [Chung et al., 2014, Józefowicz et al., 2015] suggest results of LSTM and GRU are identical. In some works with polyphonic music modeling and speech signal modeling, GRU outperformed LSTM.

In the next section it is discussed how all *Deep Learning* structures can be combined in order to achieve relevant results on different Artificial Intelligence and Computer Graphics Processing tasks.

3 Applicable Structures and Other Techniques

From now, it will be shown some *Deep Learning* architectures, one Computer Graphic method to detect blobs which could be very useful and how they were able to solve several different problems. It is good to say these architectures, in general, are a combination of different types of *Deep Learning* basic structures, specialized in solving a single type of problem, in order to solve a bigger problem.

3.1 Maximally Stable Extremal Regions (MSER)

There was a possibility that using only *Deep Learning* algorithms may not be enough to accomplish a good result because when this work started, it was not found any object detector which could solve the problem of finding text areas. In this case, there is one other technique from Image Processing area that may be used for helping on improving the results.

Maximally Stable Extremal Regions (MSER) is a blob detector that extracts from an image a number of co-variant regions, called MSER's: an MSER is a stable connected component of some gray-level sets of the image [Matas et al., 2002]. It is possible to see in the Figure 7 how MSER basically works. From the original paper[Matas et al., 2002], a MSER can be defined as follows:

Image: I is a mapping $I : D \subset \mathbb{Z}^2 \rightarrow S$. Extremal regions are well-defined on images if:

- 1 - S is totally ordered (total, antisymmetric and transitive binary relations $\leq$ exist);
- 2 - An adjacency relation $A \subset D \times D$ is defined.

Region: Q is a contiguous subset of D. (For each $p, q \in Q$ there is a sequence $p, a_1, a_2, .., a_n, q$ and $pAa_1, a_iAa_{i+1}, a_nAq$.)

(Outer) Region Boundary: $\partial Q = \{q \in D \quad Q : \exists p \in Q : qAp\}$, which means the boundary ∂Q of Q is the set of pixels adjacent to at least one pixel of Q but not belonging to Q.

Extremal Region: $Q \subset D$ is a region such that either for all $p \in Q, q \in \partial Q : I(p) > I(q)$ (maximum intensity region) or for all $p \in Q, q \in \partial Q : I(p) < I(q)$ (minimum intensity region).

Maximally Stable Extremal Region: Let $Q_1, .., Q_{i-1}, Q_i, ...$ be a sequence of nested extremal regions ($Q_i \subset Q_{i+1}$). Extremal region Q_{i*} is maximally stable if and only if $q(i) = |Q_{i+\Delta} \quad Q_{i-\Delta}|/|Q_i|$ has a local minimum at $i*$. $\Delta \in S$ is a parameter of the method.

Figure 7: Example of MSER blob detection [Matas et al., 2002]

MSER is based on gray-scale images: it detects the variants between luminosity to detect stable areas and return them as features. This technique showed very good results on area detection but it does not have a very good result when it comes to images with blur. It can be a problem

because photos from real world usually have at least a little blur because of influences of nature.

It has already been shown the use of MSER combined with Canny edges detector reaches relevant results on text detection on real world scenes[Chen et al., 2011]. This technique could potentially be used to improve text detection problem of this OCR.

3.2 Autoencoder + Convolutional Neural Network

The Autoencoder structure is known by its power of denoising media and capability of working in unsupervised way. As said before, it is basically a MLP with one hidden layer able to learn how to encode-decode a given data type. This ability is very important in at least two tasks:

- Denoising data: as this neural network knows the structure of the data being used, it knows when the structure is modified, included by noise. For this reason, Denoising Autoencoders[Mao et al., 2016] are very powerful on the task of removing noise from images;
- Generate artificial data: suppose there is a task of classifying a given type of data but the data set available for training does not have enough information to create an accurate classifier. One way to acquire more data is very simple: someone could find more data which looks like the one given. However, this task can take very long time until the amount of information needed is available. Another way to do is to use a Generative Adversarial Networks (GAN)[Goodfellow et al., 2014] trained with the available data to create artificial but reliable new information that can be used in this classifier.

In both cases, the structure of the Autoencoder is changed to work better with images: instead of using MLP in a hidden layer, they use stacks of

CNN to improve the performance of the neural network.

3.3 Convolutional Neural Network + Long-Short Term Memory

The combination of these two *Deep Learning* architectures has generated very interesting results in combination of image processing with natural language processing. It looks like a lot with the way people interpret images into words.

There are several works using this technique [Karpathy and Li, 2014, Vinyals et al., 2014] to generate image description and the results are really impressive. In many cases, the description of a given image is as good as a human being would predict.

How about, given a description, the computer creates an image based on those words? This is something that would be unthoughtful until some years ago, now it is real.

It is already possible by using a combination of two GAN's[Zhang et al., 2016]: the first one takes a sentence as input and outputs an image with primitive shapes and basic colors. The second uses this primitive image and the original sentence as input and generates a high resolution version of the image, with the missing details.

3.4 Fully Convolutional Network - FCN

Deep Learning has proven to be a very good classifier for images [LeCun et al., 2001, Vincent et al., 2010]. However, there are several other tasks which would be very interesting if could be accomplished by these techniques, including object detection and image segmentation.

It is possible to reach a very good result using a slight different architecture of traditional CNN, known as Fully Convolutional Network (FCN). This network does not use MLP on the top, only convolution layers. Each blade is responsible for a class or a value for regression, indicating what is the context of a given region of the picture.

To illustrate it better, Long et al.[Long et al., 2014] shows a very good example: it receives an image as input and pass it through some layers of convolutions and pooling. On the top of the network there is a final set of convolutional layers, each generated blade provides a probability chance for each pixel to belong to one of known classes. With this information, it is possible to segment an image in a pixel level, defining a satisfactory contour for each object that belongs to one of acknowledged category.

Another example is to use this architecture to define the bounding box of known items in the image. R-FCN[Dai et al., 2016] has convolution layers on the top which provide information about region of a given object and to what class it belongs. This type of work could be used for tracking objects in video.

3.5 YOLO - You Only Look Once

When the proposed framework was first thought, the idea was to use only *Deep Learning* techniques but there was a gap in the first task: finding some approach that reach a relevant result on character/word detection on real world images.

YOLO (*You Only Look Once*)[Redmon et al., 2015] was designed to work with object detection at first. It does not only claim it could detect with a good precision and a good recall, but it is able to work in real-time.

The speed on making the bounding-box and predictions are possible because YOLO uses a very traditional approach on the initial and mid-level

layers: an image as input, followed by sequences of convolutions 3x3 and max-pooling. The final layer is a smaller convolution (1x1) aimed to make 5 bounding-box prediction for each 13x13 square of the image. It just keeps the bounding-boxes that achieves a higher score than a given threshold.

After a deep search for studies, it was not found any work using this network being used for text detection on photographs. In this chapter it is explained how YOLO works and how it is trained. After that, in Chapter 5 it is shown how this network performed when used as an OCR.

The implementation details can be seen in the following sections.

3.5.1 Loss Function

There is a special loss formula for YOLO network. Equation 1 details all parts.

$$
\begin{aligned}
loss = & \\
& \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \left[\mathbb{1}_{ij}^{\text{obj}} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \left[\mathbb{1}_{ij}^{\text{obj}} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2) \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
& + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
& + \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{1}
$$

Where:

- line 1 calculates the error for the center of the prediction (x and y);
- line 2 calculates the error for space from the center (width and height);
- line 3 corrects the confidence score where the detector should show a relevant object (confidence score is set to 1);
- line 4 corrects the confidence score where the detector should not show a relevant object (confidence score is set to 0);

- line 5 is the error function for a wrong object predicted (categorical cross-entropy);
- $\mathbb{1}_{ij}^{\text{obj}}$ denotes where there should be an object;
- $\mathbb{1}_{ij}^{\text{noobj}}$ denotes where there should not be an object;
- λ_{coord} and λ_{noobj} are constant values for correction where there is and there is not an object, respectively.

3.5.2 Non-Maximum Suppression Algorithm

As described before, YOLO can detect a relevant quantity of interesting areas in an image. Even considering the confidence score to eliminate some areas that does not contain any pertinent object, there is still one problem: suppose there is only one item to be detected. It might happen more than one detection of it by the last layer. How should the program say what is the correct detection?

Non-Maximum Suppression Algorithm works on this kind of problem. It looks on every detected boxes around a region and outputs the best guess. To execute this task, it needs to calculate the Intersection over Union (IoU) of detected areas and then decides if keep or if eliminates a given area.

The algorithm 9 describes it details:

Algorithm 9: Non-Maximum Suppression Algorithm

Data: boxes, overlappingThreshold

Result: Relevant Bounding-Boxes According Intersection

```
1 mainBox == getNextBox();
2 while has boxes do
3     while aBox == getNextBox() do
4         if IoU(mainBox, aBox) > overlappingThreshold then
5             Remove aBox from list;
6         mainBox == getNextBox();
```

3.5.3 Training

At first it may not be cleared but the most important item on the loss function is the confidence score. This value represents whether to show the bounding box or not. According to [Redmon et al., 2015], when this value is higher than 0.3 it means this area shows one of the objects that the network can recognize, otherwise, it is just background. As explained before, sometimes an item can be recognized by more than one bounding box, so it is necessary to use the Non-Maximum Suppression algorithm to set the correct confidence score for each bounding box representation.

Suppose one of the images has only one object to be detected and during the training the network says there are two bounding boxes around the same area. So for this region it must calculate which of them has the best intersection with the ground truth. The one with the best result receives a score of 1 in the confidence score and all other losses (area and class) are calculated. The other just receives a score of 0 and there is no need to evaluate all other values.

With this information, the neural network learns when it should show or not more than one bounding box in some regions.

3.5.4 Output Results of Prediction

Let's consider a trained version of YOLO is available to detect some objects. After predicting a given image, it will output a lot of information and all of it should be interpreted correctly to have the desired result. For a better explanation, it will be divided in three parts:

- the confidence result: first information to look for each region. If the confidence score is higher than 0.3, the information is plotted according to the following two data;
- the bounding-box result: it defines the region where an object is placed;
- the class result: defines what object is in the defined region

With all this explanation, the proposed framework is detailed in the next section.

4 Proposed Framework

Even with the advances of Artificial Intelligence and Computer Graphics Processing, it is still very hard to accomplish near 100% of detection and transcription of image-to-text. For solving this issue, a combination of Deep Learning techniques was chosen in order to build a new framework with the capacity of detecting characters in this kind of image.

Deep Learning techniques have shown it is possible to accomplish better results on this problem[LeCun et al., 2001, Zheng et al., 2015]. As it is a relative recent area of research, it is possible the best result is not accomplished yet.

In this section there is a discussion about the motivation for developing this framework and then it describes in more detail about how each part of the framework is expected to work.

4.1 Motivation

Current OCR techniques can not extract 100% of the present text in images correctly, even when the same text is easily recognized by a person. With the new techniques of artificial intelligence and computer vision being developed it is very likely to be possible to increase the accuracy of recognition with respect to those obtained by current software. *Deep Learning* is a branch of the machine learning area based on a set of algorithms which aims to model high-level abstractions of data using multiple processing layers with complex structures composed of non-linear transformations. Deep learning is characterized by:

- use of a cascade of many layers of nonlinear processing units for extraction and processing characteristics. Each layer uses the output

data of the previous layer processing. The algorithms can be supervised (for classification) or unsupervised (for pattern analysis).

- It is based on multiple learning characteristics or levels of data representations. Higher level features are made from lower-level characteristics to form a hierarchy of representations.
- learn multiple levels of representations that correspond to different levels of abstractions: levels form a hierarchy of concepts.

Neural networks are usually quite rapid in their tasks once they are trained, but the training stage can be quite time-consuming, depending on the complexity of the network, training data set and the problem being treated. Any change in the structure of a network requires a new training, which can take many hours (or even days), using the same machines with high processing power. Therefore, the network optimization process must be well planned, since the number of combinations and candidate architectures which can be tested within the proposed schedule is limited.

4.2 Initial Idea

This Section details how the proposed framework was first thought to work.

4.2.1 Candidate Areas

The very first step is to find what areas have more chance to contain a character. In this step, all candidate word areas are isolated to be used in the next step.

In this step, two main approaches seen to be the best to accomplish a good result. The first one is the use of the Maximally Stable Extremal Regions (MSER) [Matas et al., 2002]. Originally created as a blob detector, it has been used with some success for detecting regions in images where apparently there is some text [Chen et al., 2011]. The negative point of this technique is that it can not differentiate a character from other, but it could still be very helpful on this task. Figure 8 represents an output from MSER on some images where there is some text.

Figure 8: Example of text detection in images using MSER

Another apparently robust technique to this job, however still not tested, is the use of Conditional Random Fields (CRF) developed as Recurrent Neural Network [Zheng et al., 2015]. In the original format described by the author, it has been used as an image semantic parser, obtaining very good results in detecting stuff in images, like bicycles, tables or humans. It might be possible to do changes in the original code that would allow the computer to detect characters instead of objects in scene. Figure 9 shows an example of CRF working on object detection, segmenting them by pixel.

Figure 9: Semantic parser of image. In the left, the original image. In the right, bikes and people detect by using this technique

4.2.2 Character Segmentation

Continuing the process of getting text from images, there is a necessity to find out the bounds that defines each character. It is very important to find out the regions that defines each letter, so the accuracy tends to be better.

There are some Deep Learning techniques that appears to have very good results on that. Again, the Conditional Random Fields as Recurrent Neural Network[Zheng et al., 2015] appears as a promise not only on defining text candidates areas but in the task of setting the bounds for each character. The precision on this job is relevant.

Convolutional Neural Networks (CNN) have had impressive results too. Some studies [Dai et al., 2015, Simonyan and Zisserman, 2014] have shown that it is possible not only to classify images into classes but semantically segment images for objects and animals. There is a possibility that an adaptation or improvement on these techniques could be used to literally read images directly into computer text, it means it is possible to get a very large amount of information.

4.2.3 Character Classification

For each segmented area, there is now the necessity of translating the image into a literal character. For each segmented area, a Convolutional Neural Network (CNN) will be trained with the task of classifying a given image.

It looks like a simple task but the problem is that we may be working with images from the real world, like a photo from a museum. The input image can contain a character with noise, rotation, translation and other common influences over it. The classifier needs to be powerful enough so this set of variation will not influence in the final translation of this image in some character

4.2.4 Final Text

The final step consists in organizing all information obtained by the Character Classification step in a text document. It is literally the translation of the image in text data, so it can be easily read by a computer. After having the final text, it will be possible to know how good the framework worked. The metrics used and recommended by ICDAR 2015 Competitions[Karatzas et al., 2015] are:

- Precision: it calculates how well the techniques detects the text in the image but punishes the grade for detected text that is not really a character in some controlled images. It is calculated using this formula:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

- Recall: very similar to the Precision, but it punishes the score for each

area that is a text but it is not detected by the framework. The final Recall score is giving by the calculus above:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

- Harmonic Mean: this calculates how good a given framework detects the exact area of the text[Wolf and Jolion, 2006].

The final evaluation of this framework will be against the ICDAR 2015 Competition data set[Karatzas et al., 2015].

4.2.5 Working Example

It is possible to see in the Figure 10 how the framework was intended to work. It shows on the image the job of this initial proposed framework.

4.3 A Fully *Deep Learning* Approach

The idea described above is a real possibility of a good working framework but it runs away of the main concept: using only *Deep Learning* techniques for a full OCR. This is now how it is intended to work

4.3.1 Candidate Areas using YOLO

As described before, YOLO[Redmon et al., 2015] has achieved a very impressive result on object detection. It looks like a good idea to get the same model to detect areas with text. Considering not only the precision (high score on true positives) but recall (a low score on false negatives) and

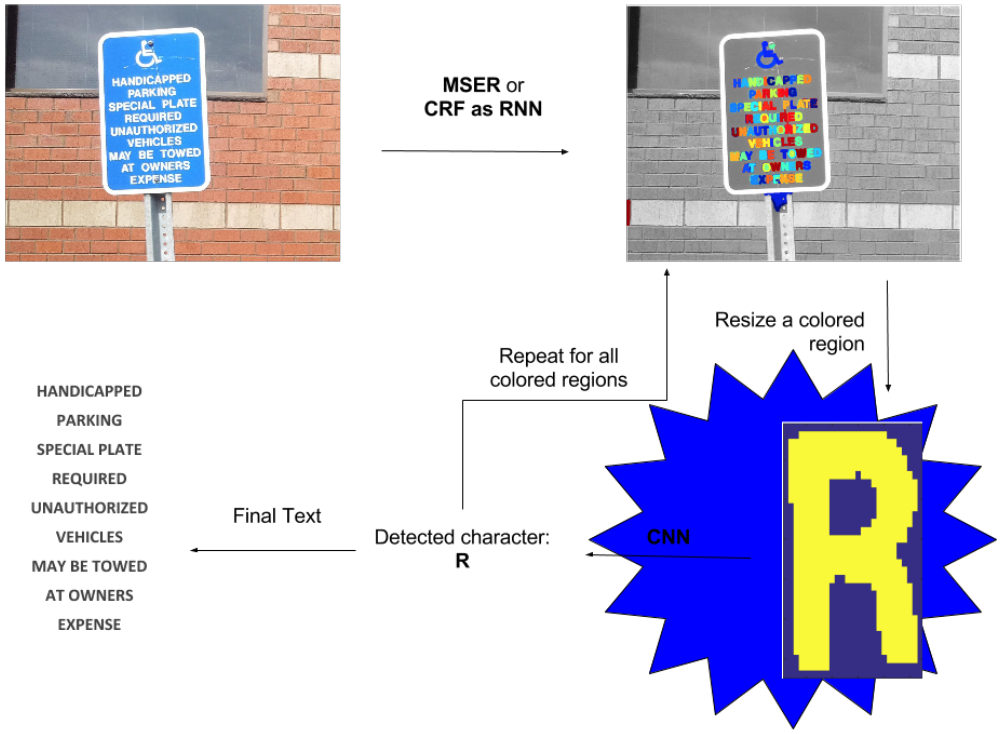

Figure 10: How the framework is intended to work

the speed on the detection, having YOLO working on this task looks like a very good approach for using Deep Learning on an entire OCR framework. Figure 11 shows how YOLO works.

4.3.2 Word Prediction

First of all it was considered to predict the words to be split in two: the first one for segmenting the area into individual character and then another to recognize what character each piece of image represents. It looks very hard because there is the necessity of two neural networks that will need:

- First: to be trained on different data sets;

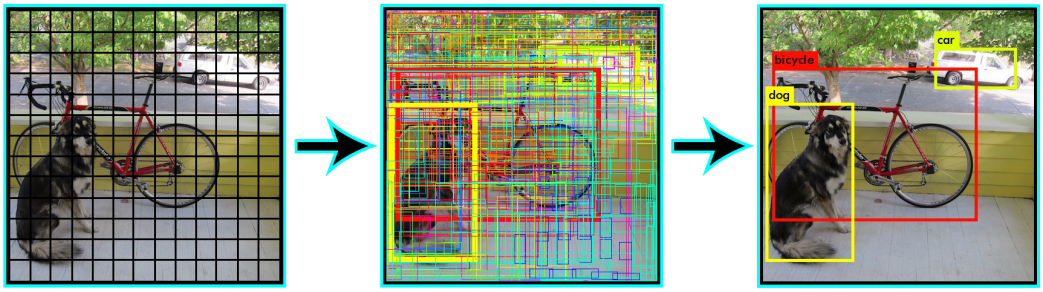

Figure 11: How YOLO works

- Second: to be executing two different tasks during executing time.

What if it could be done in only one step? The combination of using CNN with LSTM has been seen very useful on word prediction[Graves and Schmidhuber, 2008], so it might be time to compare how the combination of CNN with GRU or some other gated recurrent network will perform.

4.3.3 Connectionist Temporal Classifier - *CTC Loss Function*

Connectionist Temporal Classifier is a system created by [Graves et al., 2006] to calculate the error on prediction of sequences of labels from unsegmented data. In the original work it was applied to improve speech recognition but it has presented very good results on several other tasks. CTC is a function to get around when the alignment between the input and the output are not known.

To illustrate it better, let's use Figure 12 as example. Both visible images shows exactly the same text ("OCR") however there are visible differences on size, space between characters and thickness. This might be a problem because a sequence predictor assumes the correct label for each step is in the right position (Ex: for a 16-step RNN, the correct label for the first image could be –O-C–R——- and for the second image would be —OC-R———, where "-" represents a blank space). In real-world data sets, both images

would have the exactly same label (OCR————-), it would be very time-consuming to align every single label to every correct space of the image.

Figure 12: Same words with different characteristics

Formally speaking, the desired behavior is to map $X = [x_1, x_2, ..., x_a]$ to $Y = [y_1, y_2, ..., y_b]$, where X is the input sequence, Y is the output array and $b \leq a$. Using traditional supervised learning algorithms to come across this problem can be tricking because beside other points:

- X and Y can diverge in length;
- The proportion of X and Y length may be different;
- There is no accurate calibration (elements' correspondence) between X and Y

CTC algorithm is align-free, it means it does not need adjustment between input and output, being a very good solution for this problem. It can use this distribution either to infer a likely output or to assess the probability of a given output. The following example explains better how it works: consider an input which length is 10 and $Y = [b, a, b, y]$. A naive approach is to assign one character for each output step and then collapse every repetition. Table 1 shows it working.

This method has two problems:

Table 1: Example of naive approach

x0	x1	x2	x3	x5	x6	x7	x8	x9	Input
b	b	a	a	a	b	b	y	y	Alignment
b			a			b		b	Output

Table 2: Example of CTC working on sequence output

t	t	ϵ	o	ϵ	o	t	h	h	Merge all repetitions
t		ϵ	o	ϵ	o	t	h		Remove all ϵ
t			o		o	t	h		The output is what remains

- There are some cases which there is no necessity to output anything in a given step. In text recognition, usually there is some space between the characters;
- It is not possible to produce outputs with repeated characters. For example, the output $[t, t, o, o, o, o, t, h, h, h]$ produces “toth” and not “tooth”.

CTC solves this problem introducing a *empty* token to list of possible outputs[Hannun, 2017]. Understand this is not a *blank* space like “ ”, it would be like a *“null”* in programming. For explaining how it works, it will be called ϵ. First step is to collapse all repetitions. After that, all □are removed. The correct output is what persist after this process. Table 2 shows an example.

Consider (X,Y) a pair of input and output, T the number of time steps of a RNN. The loss function for a given pair (X, Y) is:

$$p(X|Y) = \sum_{A \in A_{X,Y}} \prod_{t=1}^{T} p_t(a_t|X)$$

where $\sum_{A \in A_{X,Y}}$ represents the marginalization over the set of valid alignments.

4.3.4 Working Example

It is possible to see in Figure 13 how the framework is now intended to work. It shows on an image what algorithm is responsible to do a given job with the changes related on this sub-section.

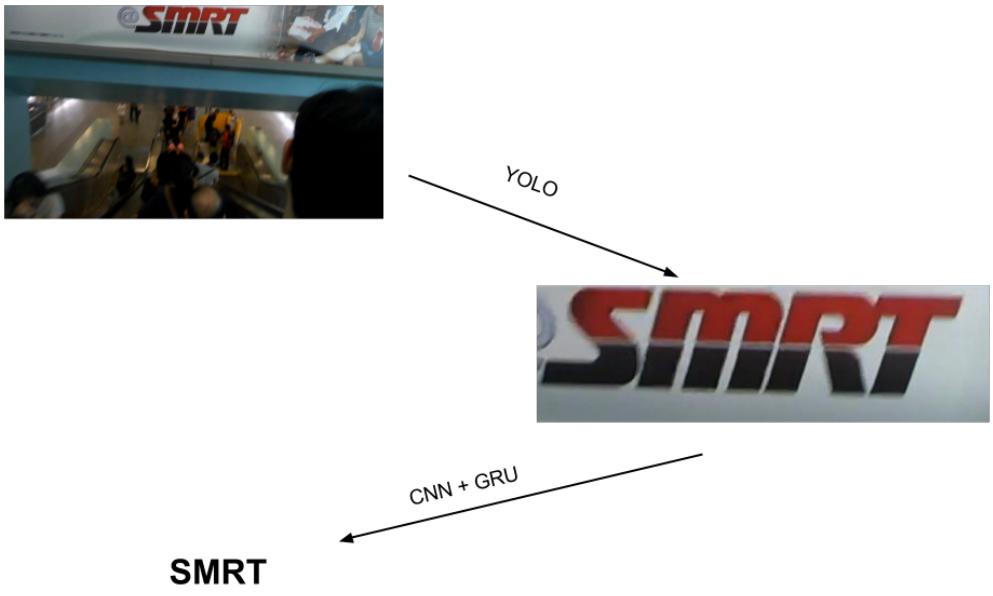

Figure 13: The new architecture using YOLO, CNN and GRU. The most significant differences are: 1 - it is fully based on *Deep Learning* techniques; 2 - It works in two steps instead of three as thought initially.

This section detailed everything needed to develop the proposed work. In the following section, the achieved results are shown.

5 Results

This chapter is divided into two main parts: Section 5.1 explains how the introductory studies helped to reach relevant results on some tasks and Section 5.3 demonstrates all the work done to arrive to a relevant result on text detection in images.

For the purpose of training and test execution, the following hardware was used:

- Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
- 32GB RAM DIMM 1333 MHz
- GPU Nvidia Quadro M5000 8GB

5.1 First Results

The amount of work to develop the desired framework is huge. Not only programming and deciding what techniques apply to each job but the time spent with the computation and training of the neural network is relevant. In the assignment of classifying digits after the segmentation, it is currently taking around 8 hours of training so it is possible to have a relevant result. The problem here is to define what is a relevant result in this job. Luckily, to have a better idea how images in real world are and how good was the results, it was found a competition in the Kaggle website [kag, 2016] challenging researchers of everywhere in the world to classify characters in the real world.

This section shows how good was the results of one of the steps that will be needed in order to create the entire framework. Figure 10 shows the entire framework working. Figure 15 highlights what step of the framework is discussed on this section.

Figure 14: Example of characters used in the competition

5.1.1 Language, Technologies and Frameworks

One relevant part of this work was to search and test some languages and frameworks that could speed up the development and aggregate good results. It is important to remember the time it takes to train a Deep Learning architecture is relevant, so it is vital to use a language that is easy to learn and it is not so slow to compile and execute the code.

In the first part of this work Python was used with TensorFlow [Abadi et al., 2015]. This combination was chosen because the available documentation and the number of available examples are huge, so it would not take too long to learn how to use and develop the first models. However, after some tests it was realized it was taking too much time on training. With the available hardware it was not possible to run the training sessions

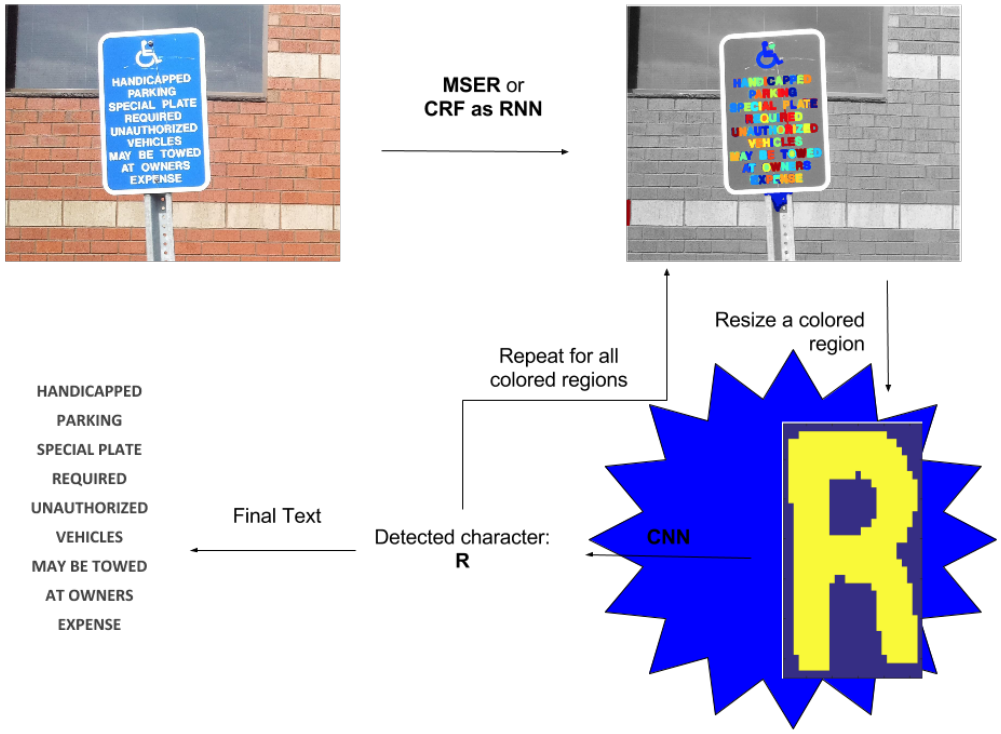

Figure 15: Highlight the part of the framework discussed in this section

over GPU, only on CPU.

In order to get rid of this problem, it was discovered that Theano[Theano Development Team, 2016] was able to work over the available GPU and it really sped up the training sessions. However, using this framework is significantly more complex than TensorFlow[Abadi et al., 2015] so this was another bottleneck.

The best training session time and low complexity was accomplished using Keras[Chollet, 2015]. It encapsulates all code from Theano[Theano Development Team, 2016] and TensorFlow[Abadi et al., 2015] and provides a friendly interface to create all kinds of Deep Learning architectures using little and easily reading code.

From November 2016 Nvidia provided a Quaddro M5000 GPU for this

research. It was a huge speed up on training sessions, decreasing the same neural networking training time from 17 seconds to 2 seconds. It is now possible to use TensorFlow over the GPU.

5.1.2 Convolutional Neural Network For The Classification Task

A CNN can be set up depending on what problem is desired to solve. For example, LeCun et al. [2001] used a very shallow network to classify handwritten digits while Simonyan and Zisserman [2014] used a much deeper architecture in the task of object classification.

In fact, most CNN architectures can be considered way deeper than the first one proposed. Szegedy et al. [2015] for instance has dozens of layers while He et al. [2015] has more than one hundred.

It will be described how each network performed.

5.1.3 Classical Architecture

The first try was using the classical CNN described in the work to classify handwritten digits (0-9)[LeCun et al., 2001]. The idea was to adapt the structure to accept numbers and characters(a-z, A-Z), using the CNN below:

- Input size: 20x20, black-and-white;
- Convolutional Layer 5x5 with 128 layers, followed Pooling 2x2;
- Convolutional Layer 5x5 with 128 layers, followed Pooling 2x2;
- Multilayer Perceptron Layer with 2048 neurons, Dropout set to 75% and Rectified Linear Unit (ReLu) as output;

- Multilayer Perceptron Layer with 2048 neurons, Dropout set to 75% and Rectified Linear Unit (ReLu) as output;
- Logistic Regression Classifier for 62 classes(A-Z, a-z, 0-9)

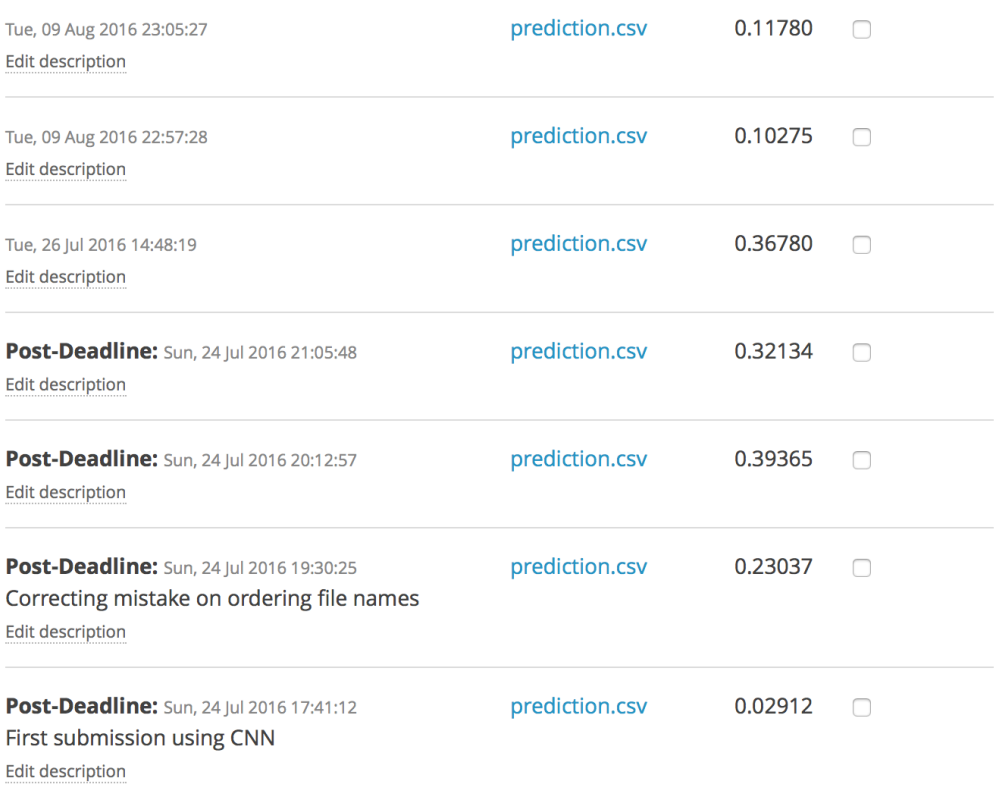

Figure 16: Initial Results

Using it as described network, it was possible to have around 39% of correct predictions (based on website evaluation), placing in 39th from 50 competitors. Figure 16 shows the initial results and Figure 17 shows the initial classification. This number means the precision of each participant's techniques on the validation data set. For starters, it was a very good result, thought with time and some parameters' correction it would be possible to get better results. Unfortunately it did not occur as initially thought, it is

		Julia Random Forest Benchmark	0.42932		
39	↓7	Abhijith CHandraprabhu	0.42408	1	Mon, 06 Jun 2016 17:16:18
40	↓7	crwang	0.42408	2	Tue, 14 Jun 2016 06:05:03 (-0.4h)
41	↓7	SiwenYan	0.42408	2	Thu, 21 Jul 2016 19:55:48 (-0.5h)
42	↓7	Takuya	0.42343	1	Sat, 11 Jun 2016 18:44:54
		Julia kNN Benchmark	0.40576		
43	↓7	Anton L	0.40151	13	Tue, 02 Aug 2016 06:23:12 (-8.9d)
44	↓7	**claudiosantos**	**0.39365**	**5**	**Tue, 26 Jul 2016 14:48:19 (-42.6h)**
45	↓7	scotto3394	0.38220	1	Tue, 28 Jun 2016 19:49:02
46	new	IceBear	0.08901	4	Wed, 03 Aug 2016 08:53:56
		All A's Submission	0.07297		
47	↓8	amyaramine	0.07297	1	Sun, 05 Jun 2016 13:02:33
48	↓8	Changani Jagdish G	0.07297	1	Mon, 20 Jun 2016 07:59:50
49	↓8	mldm16SGL	0.07297	1	Tue, 28 Jun 2016 16:00:56
50	↓8	Paul Pajo	0.07297	3	Mon, 25 Jul 2016 04:44:58 (-0.9h)

Figure 17: Initial Classification

possible to see the results did not improve and it took a time studying other possible implementations.

5.1.4 VGG Architecture + Data Augmentation

One that work very well in the first try was the VGG style [Simonyan and Zisserman, 2014] network. The architecture is described bellow:

- Input size: 32x32, black-and-white;
- Convolutional Layer 3x3 with 128 layers;
- Convolutional Layer 3x3 with 128 layers, followed Pooling 2x2;
- Convolutional Layer 3x3 with 256 layers;

- Convolutional Layer 3x3 with 256 layers, followed Pooling 2x2;
- Convolutional Layer 3x3 with 512 layers;
- Convolutional Layer 3x3 with 512 layers;
- Convolutional Layer 3x3 with 512 layers, followed Pooling 2x2;
- Multilayer Perceptron Layer with 4096 neurons, Dropout set to 75% and Rectified Linear Unit (ReLu) as output;
- Multilayer Perceptron Layer with 4096 neurons, Dropout set to 75% and Rectified Linear Unit (ReLu) as output;
- Logistic Regression Classifier for 62 classes(A-Z, a-z, 0-9)

In some papers[He et al., 2015, Szegedy et al., 2015] it is related that in order to improve the result, artificial augmentation of data available for training helps to avoid overfitting and significantly upgrade the outcome. In this work, for each training iteration, the following working on training data set was done:

- random rotation in 30 degrees for any direction;
- random shift for height and width of 15%
- random zoom in and out of 15%
- random shearing of 20%

With the described changes in the network and training data, it was accomplished over 82% of correct predictions as it is possible to see in Figure 18, placing this work in 4th of 41 competitors, significantly improving the results. Figure 19 shows the new position.

Submission	Files	Public Score	Selected?
Mon, 26 Sep 2016 11:17:31 Edit description	CNN_pred_0.csv	0.83508	☐
Sun, 25 Sep 2016 23:13:20 Edit description	CNN_pred_1.csv	0.82657	☐
Sun, 25 Sep 2016 10:43:25 Edit description	CNN_pred_0.csv	0.83082	☐
Sun, 25 Sep 2016 02:28:24 Trying with vgg style + data augmentation Edit description	CNN_pred_1.csv	0.82657	☐

Figure 18: Current Results

5.2 Metrics

In order to understand how good were the results, it is now explained how each metric is calculated and in what cases it may be applied.

For each formula, the following convention names are used:

- P defines Precision;
- R defines Recall;
- D is the area detected by some method;
- G is the ground-truth of a given area;

5.2.1 Intersection over Union - IoU

The Intersection over Union (IoU) metric is used in several object detection competitions. As the name says, it calculates how well the relevant

#	Δ1w	Team Name * in the money	Score	Entries	Last Submission UTC (Best – Last Submission)
1	–	Fabien Tencé *	0.86780	1	Fri, 16 Sep 2016 11:00:59
2	–	Loan Tricot	0.85438	1	Thu, 28 Jul 2016 17:21:07
3	new	xiaobai	0.84719	1	Mon, 26 Sep 2016 13:08:28
4	↑29	**claudiosantos**	**0.83508**	**7**	**Mon, 26 Sep 2016 11:17:31**
5	↓2	Zhuravlev Nikita	0.81512	4	Fri, 23 Sep 2016 19:14:05 (-3.2d)
6	↓2	Dawid Loranc	0.76767	5	Sat, 06 Aug 2016 00:17:28

Figure 19: Current Classification

region was detected by a given technique according to the intersection and the union of the area. The mathematic definition for one area is:

$$IoU = \frac{D \cap G}{D \cup G}$$

5.2.2 DetEval

IoU is a very good metric to define how good was the detection of interesting areas but in ICDAR competitions uses other metrics. Calculation of DetEval is slightly more complex because it considers not only one-to-one detection, but one-to-many detection (when a single rectangle detects more than one area) and many-to-one detection (when several rectangles detect a single area)

First, from the two sets D and G of detected rectangles (regions) and ground truth rectangles, we can construct two recall and precision matrices σ and τ of the area overlap where the rows of the matrices correspond to the ground truth rectangles and the columns correspond to the detected rectangles. The values of the i^{th} row and j^{th} column are defined as:

$$\tau_{ij} = P_{AR} = \frac{Area(G_i, G_j)}{Area(D_i)}$$

$$\sigma_{ij} = R_{AR} = \frac{Area(G_i, D_j)}{Area(G_i)}$$

$Area$ is rectangle region. The detected region is relevant if:

$$\tau_{ij} > t_p = 0.4, \sigma_{ij} > t_r = 0.8$$

This evaluation strategy deals with over-split or over-merge of detection by supporting one-to-one, one-to-many, and many-to-one matches among ground-truth objects and detection. Based on this information, the P and R in a given image are calculated using the following formulas:

$$P(G, D, t_r, t_p) = \frac{\sum_i Match_D(D_j, G, t_r, t_p)}{|D|}$$

$$R(G, D, t_r, t_p) = \frac{\sum_i Match_G(G_i, D, t_r, t_p)}{|G|}$$

where $Match_D$ and $Match_G$ are defined as:

$$Match_D = \begin{cases} 1, & \text{if } D_j \text{ matches a single detected rectangle,} \\ 0, & \text{if } D_j \text{ does not match any detected rectangle,} \\ f_{sc}(k) & \text{if } D_j \text{ matches several } (k) \text{ detected rectangles} \end{cases}$$

$$Match_G = \begin{cases} 1, & \text{if } G_i \text{ matches a single detected rectangle,} \\ 0, & \text{if } G_i \text{ does not match any detected rectangle,} \\ f_{sc}(k) & \text{if } G_i \text{ matches several } (k) \text{ detected rectangles} \end{cases}$$

For the ICDAR competitions, the following convention was used:

- for $Match_D$, $f_{sc}(k)$ = 1, so it does not penalize over or under segmentation;
- for $Match_G$, $f_{sc}(k)$ = 0.8.

Finally, the F-Score is calculated as:

$$f = \frac{1}{\frac{1}{0.5 * P} + \frac{1}{0.5 * R}}$$

5.2.3 ICDAR2013

This metric defines the final score on "Born-Digital Images - Task 1: Text Detection". It considers the values for IoU and DetEval. According to Nourbakhsh et al. [2011], this score is calculated in the following equation:

$$ICDAR2013 = \frac{IoU + DetEval}{2}$$

5.2.4 *editdistance*

To define the final result on Word Recognition, ICDAR uses *editdistance* metric. According to [Karatzas et al., 2015, 2013], it quantifies how similar

two strings are. To get this value, it compares how many insertions, deletions and substitutions of character are needed to both strings become equals. ICDAR compares in two ways: case-sensitive where mistakes of capital letters are considered and case-insensitive where they are not considered.

For each change necessary in the word, the score is increased by 1. The final score is the sum of the modifications. It means the best score a method can achieve is 0.

5.2.5 End-to-End

One metric used to evaluate the word detected in an image is the End-to-End. According to Karatzas et al. [2015] the following pipeline computes this score:

- It first calculates the IoU of a detected region. If $IoU > 0.5$ it considers a successfully detection;
- For a correct detection it now calculates if the word was written correctly. If $editdistance = 0$, the score is considered. If not, the region is discarded.

5.2.6 Word Spotting

The second metric used to detect and translate image's regions into text is the Word Spotting. It follows the same protocol as End-to-End with one step more: it considers only words provided by a vocabulary.

To understand the difference between both metrics: while End-to-End is interested in detect and translate all words in an given image, Word Spotting considers only some relevant words highlighted by the competition.

5.3 Final Implementation Details

All the work done until here was a very good case of study to learn how *Deep Learning* structure works. Unfortunately it was not relevant on the final result of the framework. From now it will be detailed what was done, how it works and all difficulties found during the implementation

5.3.1 YOLO

To better understand how YOLO works, it was re-implemented using Keras[Chollet, 2015] with TensorFlow[Abadi et al., 2015] in the backend. Even considering it is only a object detection task (considering "text" as object), the implementation is ready to detect and classify more than one class. It only needs to be trained accordingly.

For the task of text detection, YOLO was trained with data from 3 different data sets showed in Figures 20, 21 and 22.

Figure 20: Images from Born Digital data set

Figure 21: Images from COCO-Text data set

Figure 22: Images from Incidental Scene Text data set

5.3.2 Image-to-Word

This network was surprisingly straight-forward to implement. Stacking two layers of convolutional layers followed by a stack of bi-directional GRU's

or LSTM's got a relevant result on the translation of images to sequence of characters. As this implementation worked fine at first and it was very fast to train and execute, this was the architecture chosen. Figure 25 illustrates an example of this network.

In this case, images from [Jaderberg et al., 2014b] and [Karatzas et al., 2015] were used. Examples are shown in Figures 23 and 24.

Figure 23: Images from Born Digital data set

Figure 24: Images from MJSynth data set

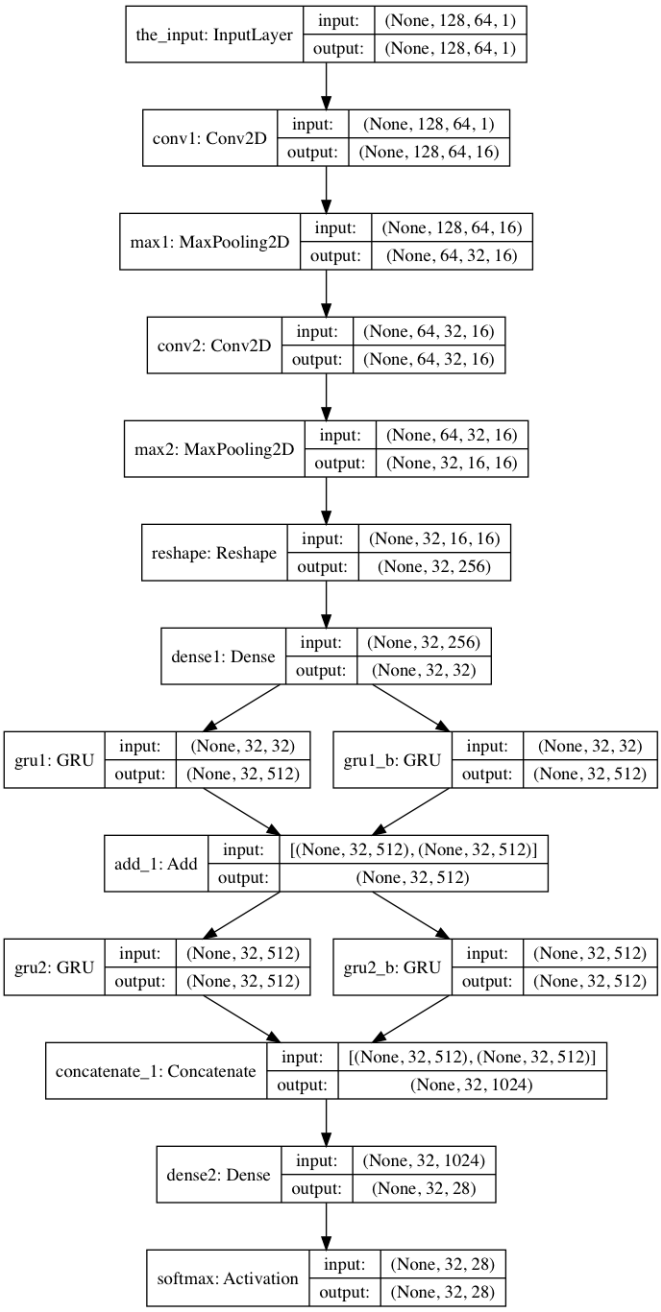

Figure 25: Example of Network tested for Image-to-Word

Each rectangle of Figure 25 represents:

- Left rectangle: type of neuron used;
- Right rectangles: size of input and output of layer.

Most implementations seen before uses unidirectional LSTM to do this kind of work. The aim of this work was to prove GRU could work as good as the previous RNN cited.

For a better comparison, 2 different architectures were tested to check what would be better for the final framework:

- Bi-directional LSTM;
- Bi-directional GRU;

For all of them all other structure was kept: gray scale image size of 128x64 followed by two sets of convolutional-maxpooling of 32 layers each. Adam Optimizer[Kingma and Ba, 2014] was used and loss function based on Connectionist Temporal Classifier[Graves et al., 2006].

5.4 Framework Results

With all explanation until here, it is time to test on real-world examples. For the first and second task there is more detailed explanation of each data set used for training and validation. Best results on both duties were used to assembly the final framework and execute the last assignment.

5.4.1 Text Localization

All starts on finding what areas of a given picture is text. As said before, the task of detecting words in real-world images is still an open problem with no perfect technique or *Deep Learning* architecture to solve it. Turns

out it is way more difficult than first thought. Even using a very powerful neural network like YOLO[Redmon et al., 2015] it shows there is still room for new systems.

In this work, it was used two versions of this neural network: the Tiny YOLO, which the author claims it could be used for real time object detection even in not so powerful devices; and the Full YOLO, a more robust architecture that still could track items in real time but needs equipment with a strong GPU.

These are the data sets used for training and validation in this task:

- Born Digital Images (ICDAR 2011 - 2015)[Jaderberg et al., 2014a,c]: 551 images divided in 410 for training and 141 for test. All images are generated by digital printer;
- Incidental Scene Text (ICDAR 2015 -)[Jaderberg et al., 2014a,c]: Contains 1500 images in 1024x768 RGB, split in 1000 for training and 500 for validation. All images are photos from different places.
- COCO-Text[Su, 2016, Veit et al., 2016]: this huge data set provided by Microsoft has 63686 images available for text challenges: 43686 for training, 10000 for test and 1000 for validation, with no public annotation available according to web page.[3]

Both networks were trained for 120 epochs using COCO-text Training data set and Incidental Scene Text Training data set. ADAM was chosen as the optimizer. Figure 26 shows where this method work well and Figure 27 shows examples where there were some problems.

[3]https://vision.cornell.edu/se3/coco-text-2/

Figure 26: Images with good detection areas. Left images are the ground truth and right images are the detection.

Figure 27: Images with bad detection areas. Left images are the ground truth and right images are the detection. Green squares show the correct detections, red squares are wrong detections and brown are middle-term detections

Table 3: Results on Born Digital Images - ICDAR2013 Metric

	Precision	Recall	H-Mean
Tiny YOLO	56.63	53.89	55.23
Full YOLO	**70.56**	**60.65**	**65.23**

Table 4: Results on Born Digital Images - DetEval Metric

	Precision	Recall	H-Mean
Tiny YOLO	56.63	53.89	55.23
Full YOLO	**71.48**	**61.45**	**65.23**

Tables 3, 4, 5 and 6 show results of each architecture on Born Digital Images validation data set. *FPS*(Frames Per Second) was calculated as the average of 100 executions over the validation data set, with the following steps:

- open file from disk
- predict relevant areas with the neural network trained

For the Incidental Scene Text, results are in Table 7. It is visible the difference between results on both data sets. The main reasons are:

- images on ”Incidental Scene Text” are way more complex than in ”Born Digital Images”. There are many more influences (noise, blur, inclination and others) in the first data set. The second one is basically clean text from computer source.

Table 5: Results on Born Digital Images - IoU Metric

	Precision	Recall	H-Mean
Tiny YOLO	41.08	38.03	39.49
Full YOLO	**60.96**	**51.65**	**55.92**

Table 6: Results on Born Digital Images - Speed in Frames per Second (FPS)

	FPS
Tiny YOLO	**42.21**
Full YOLO	21.19

Table 7: Results on Incidental Scene Text

	Precision	**Recall**	**H-Mean**	**FPS**
Tiny YOLO	14.39	14.54	14.46	**30.73**
Full YOLO	**31.54**	**22.00**	**25.92**	18.18

- YOLO yields (x_{center}, y_{center}, width, height), where x_{center} and y_{center} are the center point of the bounding-box. "Born Digital Images" requires the output for each instance to be (x_{min}, y_{min}, x_{max}, y_{max}), which define the top-left and right-bottom coordinates of each region which is a very easy value translation. "Incidental Scene Text" needs the result to be (x_1, y_1,x_2, y_2, x_3, y_3,x_4, y_4) where each tuple (x, y) defines one corner of a given location. Eventually, these values may not generate a perfect rectangle, which makes YOLO's output to loss at least some points on Intersection over Union (IoU).
- speed: images on "Incidental Scene Text" are bigger than in "Born Digital Images". They take more time to load.

5.4.2 Word Recognition

Word Recognition is the second and final step of entire framework. As it is possible to see in this section, this task can be evaluated alone too. As said before, this task consists basically in a sequence classification, where the sequence can theoretically have size between 0 and infinity.

For practical purpose, in this work it was limited to 21 characters in a single image. Training and evaluation was made in these data sets:

- Born Digital Images (ICDAR 2011 - 2015) - Task 3: Word Recognition[Jaderberg et al., 2014a,c]: 3564 images of cropped words from image are used for training and 1439 for validation, on a total of 5003 images;
- Incidental Scene Text (ICDAR 2015 -) - Task 3: Word Recognition[Jaderberg et al., 2014a,c]: Contains 6545 images, split in 4468 for training and 2077 for validation.
- MJSynth data set[Jaderberg et al., 2014b]: a huge data set with 9 million images from 90 thousand English words

The main question in this task is: does GRU perform better than LSTM? Both are RNN types and LSTM is more used than GRU (maybe because it is better known). For evaluation, two neural networks, each with one type of neuron was trained with this configuration set:

- images resized by 100x30 in gray scale;
- two layers of 32 CNN followed by MaxPooling;
- bi-directional RNN with 512 neurons;
- alphabet defined in a-z, A-Z, 0-9 and white space (64 classes);
- CTC loss;
- mini-batch of 32;
- ADAM optimizer;
- train performed in 3 data sets in this sequence:

Table 8: Results on Incidental Scene Text validation data sets. TED = Total Edit Distance, CRW = Correct Recognized Words, CI = Case Insensitive, CS = Case Sensitive

	Incidental Scene Text			
	TED-CI	CRW-CI	TED-CS	CRW-CS
LSTM	**623.18**	**40.20%**	**767.35**	**34.28%**
GRU	953.45	26.81%	1073.22	23.78%

- trained in MJSynth data set, each epoch using clusters of 12000 images. Each cluster was trained twice for 130 epochs. Experiments showed it was enough to converge the result;
- trained in Born Digital Images data set twice;
- trained in Incidental Scene Text twice;

Tables 8 and 9 shows results on Born Digital Images and Incidental Scene Text validation data sets respectively. It was initially thought GRU would have similar performance on character detection, however LSTM was significantly better. It shows in this particular case, the additional gate makes difference. Besides the results on, GRU is faster than LSTM: during training, each epoch took 19s on GRU while in LSTM it took 22s, around 10% of faster.

A problem was found in all data sets: the performance for translating normal words like “cat”, “COMPUTER” or names was really good. However, there are few examples for training with numbers and other symbols like ”$”, ”€” and others not listed here. Even with this problem, results indicate that if a data set with enough words with these listed symbols is used in training, both neural networks can improve their results.

Table 9: Results on Born Digital Images validation data sets. TED = Total Edit Distance, CRW = Correct Recognized Words, CI = Case Insensitive, CS = Case Sensitive

	Born Digital Images			
	TED-CI	CRW-CI	TED-CS	CRW-CS
LSTM	**164.17**	**69.07%**	**190.48**	**67.26%**
GRU	250.03	54.13%	269.99	52.95%

5.4.3 End-to-End

Both steps before were largely evaluated for a reason: what combination would create a single framework for detecting text areas and translating them into text for a given image?

ICDAR provides two good challenges for evaluation: “End-To-End Text Detection and Recognition for Born Digital Image” and “End-To-End Text Detection and Recognition for Incidental Scene Text”. Data sets to be evaluated are the same for Text Detection task however here it is necessary to translate every single piece of detected text into a sequence of characters.

The full working pipeline is described below:

- According to the results, Full-YOLO architecture performed better in text detection for both data sets. It performs the first stage of this framework.
- For each detected area, a hybrid architecture using CNN-LSTM translates each detected in a sequence of characters. If a given region is translated as “”(empty), “ ”(white space) or a sequence of white spaces, this region is discarded. Otherwise, this region is considered and the final text is the output of the second neural network.

According to each web page[4],[5] a protocol proposed by [Wang et al., 2011] is used. It considers a detection as a match if it overlaps a ground

[4] http://rrc.cvc.uab.es/?ch=4&com=tasks
[5] http://rrc.cvc.uab.es/?ch=1&com=tasks

Table 10: “Born-Digital Images” Blind Detection Results

	Precision	Recall	H-Mean
End-To-End	0.2435	0.2280	0.2355
Word Spotting	0.2617	0.2519	0.2567

Table 11: “Incidental Scene Text” Blind Detection Results

	Precision	Recall	H-Mean
End-To-End	0.0651	0.0428	0.0516
Word Spotting	0.0672	0.0454	0.0542

truth bounding box by more than 50% (as in [Everingham et al., 2015]) and the words match, ignoring the case.

Results in this section are separated in two main sections: first it is shown the performance on blind detection, it means how the framework performed by itself. The second part is following the rules for task “Strongly Contextualized” from ICDAR End-to-End text detection for “Born-Digital Images” and “Incidental Scene Text”.

Starting from blind detection, results are show on Table 10 for “Born-Digital Images” and Table 11 for “Incidental Scene Text” Validation data sets. In the first case results are somewhat good, detecting correctly around one fourth of total desire areas and then translate these regions in words. However, it may be considered a very bad result on real word photos getting only 5% of correct detection process. These results make clear there is a lot of room for improvement in both parts of the framework if it is intended to work with no help from any type.

The described problems go significantly down when results are combined with auxiliary files from “Strongly Contextualized Results”. To perform this task, the following pipeline was used:

- The framework performs its detection using YOLO for relevant areas and the CNN+LSTM combination to translate this area in word;

Table 12: “Born-Digital Images” Strongly Contextualized Results

	Precision	Recall	H-Mean
End-To-End	0.4959	0.4645	0.4797
Word Spotting	0.5478	0.5273	0.5373

Table 13: “Incidental Scene Text” Strongly Contextualized Results

	Precision	Recall	H-Mean
End-To-End	0.2136	0.1405	0.1695
Word Spotting	0.2237	0.1511	0.1804

- For each word it is performed a search in the auxiliary files. If the word is found, then this word is set to this region. If not, the word of the list with the smaller *editdistance* described before is set to the region.

Results are show on Table 12 for “Born-Digital Images” and Table 13 for “Incidental Scene Text” Validation data sets. It performs two times better in the first data set and near three times better in the last image set.

These last results complete the first intention of this work: to understand how a OCR pipeline works.

For summarizing, Table 14 shows the number of instances used on each task of this work.

Table 14: Data sets used in this work divided in Training Set and Validation Set

Data Set	Training Set Size	Validation Test Size
Born-Digital Images - Tasks 1 / 4	410	141
Born-Digital Images - Task 3	3,564	1,439
Incidental Scene Text - Tasks 1 / 4	1,000	500
Incidental Scene Text - Task 3	4,468	2,077
MJsynth	9,000,000	0
COCO-Text	43,686	0

In the next section, the final considerations are detailed.

6 Final Considerations

This study aimed to create a OCR system for text in several types of images. It shows it is possible to create a full pipeline to perform this task but it is still necessary some improvement in order to reach state-of-art performance. This conclusion highlights the scientific contribution of this work and some points that would boost the general performance.

6.1 Scientific Relevance

6.1.1 Object Detector as Text Detector

The first relevant point of this study is the proof of using a *Deep Learning* architecture for object detection can be trained and used as a text detector in photographs and other types of images, performing well when YOLO Redmon et al. [2015] was used. It suggests other architectures Dai et al. [2016], Zheng et al. [2015] could be re-trained in order to accomplish relevant result on this task.

6.1.2 Debate on Other Research Study

Another innovation was the idea that GRU Cho et al. [2014] could perform better on the word recognition task than LSTM Donahue et al. [2014]. Chung et al. Chung et al. [2014] suggested the first type of RNN could

even outperform the second with the advantage of being faster. However, this study proof it is not always the case, with a relevant performance difference. It suggests these architectures could be challenged on other sequence tasks like video classification or other list dependent tasks.

6.2 Neural Network Architectures

Both steps on this work showed some good results even using relatively simple architectures. It has been shown when deeper architectures are used, results tend to be better.

In this work, YOLO's architecture was trained only using the Tiny and Full versions, which are relatively simple. In its original paper Redmon et al. [2015] got better results using deeper architectures as backend, such as VGG16[Simonyan and Zisserman, 2014], MobileNet[Howard et al., 2017], Inception-V3[Szegedy et al., 2015] and ResNet[He et al., 2015].

Another point that should improve results is a scalar pre-training. On YOLO9000 [Redmon and Farhadi, 2016], the entire neural network is trained in two different image sizes before training on desired scale. In its report, it improves significantly the results, showing this technique could be used in text detection too.

Last point regarding improving YOLO's detection is how bounding-box regions are formed. The original implementation detects $(x_{min}, y_{min}, x_{max}, y_{max})$ (2 points) while most of the text detections are based on $(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$ (4 points). This can cause more errors when IoU is calculated. Perhaps if the neural network were developed using this system, these results could be improved too.

The CNN + RNN architecture for text recognition is shallow too. It has only two layers of convolutions and uses a relatively small input (100x30). In order to detect more features, this window could be bigger to enable this

network to have more layers of transformation and detection. It is not clear if the second part would perform better if there was more recurrent layers and more neurons but as said before, deeper networks tend to perform better than flat ones.

6.3 More Training Data

Deep Learning has shown state-of-art results on several different tasks but it is not perfect. One of the reasons for this problem is because it is data hungry, it means it needs a lot of data to understand what it needs to classify or detect. This point may be affecting the results on both models.

YOLO was trained using COCO-Text and both ICDAR data sets for text detection which when summed up gives around 45000 images for training. This quantity is significantly smaller than the original COCO data set, it claims to have more than 220000 labeled images for training.[6] Even using artificial data augmentation, it may be not enough to reach the best result of this network.

For the word recognition part, the number of labeled images looks enough to detect characters (A-Z, a-z) but it does not contain any numbers or other symbols (0-9, !, ?, * and others) that appears on validation data sets. It may exist a data set that could solve this problem however it was not found until this work was finished.

[6] http://cocodata set.org/

7 References

(2016). Kaggle. http://www.kaggle.com. Accessed: 2016-07-26.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Burks, A. W. and Wang, H. (1957). The logic of automata - part i. *J. ACM*, 4(2):193–218.

Chen, H., Tsai, S. S., Schroth, G., Chen, D. M., Grzeszczuk, R., and Girod, B. (2011). Robust text detection in natural images with edge-enhanced maximally stable extremal regions. In *2011 IEEE International Conference on Image Processing*, Brussels.

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259.

Chollet, F. (2015). Keras. https://github.com/fchollet/keras.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evalu-

ation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Csáji, B. C. (2001). Approximation with artificial neural networks. *MSc Thesis, Eötvös Loránd University (ELTE), Budapest, Hungary*.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

Dai, J., He, K., and Sun, J. (2015). Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. *CoRR*, abs/1503.01640.

Dai, J., Li, Y., He, K., and Sun, J. (2016). R-FCN: object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409.

Donahue, J., Hendricks, L. A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2014). Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389.

Everingham, M., Eslami, S. M., Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *Int. J. Comput. Vision*, 111(1):98–136.

Farley, B. G. and Clark, W. A. (1954). Simulation of self-organizing systems by digital computer. *Trans. of the IRE Professional Group on Information Theory (TIT)*, 4:76–84.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA. ACM.

Graves, A. and Schmidhuber, J. (2008). Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 545–552.

Hannun, A. (2017). Sequence modeling with ctc. *Distill*. https://distill.pub/2017/ctc.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Hinton, G. E. (2007). Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11:428–434.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861.

Jaderberg, M., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014a). Reading text in the wild with convolutional neural networks. *CoRR*, abs/1412.1842.

Jaderberg, M., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014b). Synthetic data and artificial neural networks for natural scene text recognition. *arXiv preprint arXiv:1406.2227*.

Jaderberg, M., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014c). Synthetic data and artificial neural networks for natural scene text recognition. *CoRR*, abs/1406.2227.

Józefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2342–2350.

Karatzas, D., Gomez-Bigorda, L., Nicolaou, A., Ghosh, S., Bagdanov, A., Iwamura, M., Matas, J., Neumann, L., Chandrasekhar, V. R., Lu, S., Shafait, F., Uchida, S., and Valveny, E. (2015). Icdar 2015 competition on robust reading. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 1156–1160.

Karatzas, D., Shafait, F., Uchida, S., Iwamura, M., Bigorda, L. G. i., Mestre, S. R., Mas, J., Mota, D. F., Almazàn, J. A., and de las Heras, L. P. (2013). Icdar 2013 robust reading competition. In *Proceedings of the 2013 12th International Conference on Document Analysis and Recognition*, ICDAR '13, pages 1484–1493, Washington, DC, USA. IEEE Computer Society.

Karpathy, A. and Li, F. (2014). Deep visual-semantic alignments for generating image descriptions. *CoRR*, abs/1412.2306.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J., and Ng, A. (2012). Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (2001). Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press.

Long, J., Shelhamer, E., and Darrell, T. (2014). Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038.

Mao, X.-J., Shen, C., and Yang, Y.-B. (2016). Image Restoration Using Very Deep Convolutional Encoder-Decoder Networks with Symmetric Skip Connections. *ArXiv e-prints*.

Matas, J., Chum, O., Urban, M., and Pajdla, T. (2002). Robust wide baseline stereo from maximally stable extremal regions. In *Proceedings of the British Machine Vision Conference*, pages 36.1–36.10. BMVA Press. doi:10.5244/C.16.36.

Mcculloch, W. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147.

Minsky, M. and Papert, S. (1969). Perceptrons : an introduction to computational geometry.

Nielsen, M. A. (2016). Neural networks and deep learning.

Nourbakhsh, F., Mas, J., Mestre, S. R., Roy, P. P., and Karatzas, D. (2011). Icdar 2011 robust reading competition - challenge 1: Reading text in born-digital images (web and email). In *2011 International Conference on*

Document Analysis and Recognition(ICDAR), volume 00, pages 1485–1490.

Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. (2015). You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640.

Redmon, J. and Farhadi, A. (2016). YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

Su, P. (2016). Coco-text explorer. In *Cornell University CS Department MEng Report*.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567.

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.

Veit, A., Matera, T., Neumann, L., Matas, J., and Belongie, S. (2016). Coco-text: Dataset and benchmark for text detection and recognition in natural images. In *arXiv preprint arXiv:1601.07140*.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2014). Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555.

Wang, K., Babenko, B., and Belongie, S. (2011). End-to-end scene text recognition. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 1457–1464, Washington, DC, USA. IEEE Computer Society.

Wasserman, P. D. and Schwartz, T. (1988). Neural networks. ii. what are they and why is everybody so interested in them now? *IEEE Expert*, 3(1):10–15.

Wolf, C. and Jolion, J.-M. (2006). Object count/Area Graphs for the Evaluation of Object Detection and Segmentation Algorithms. *International Journal of Document Analysis and Recognition*, 8(4):280–296.

Zhang, H., Xu, T., Li, H., Zhang, S., Huang, X., Wang, X., and Metaxas, D. N. (2016). Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *CoRR*, abs/1612.03242.

Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., and Torr, P. H. S. (2015). Conditional random fields as recurrent neural networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1529–1537.