Bleed Through Removal

Linda Simoncini and Johan Bosso
(Dated: April 15, 2019)

I. ABSTRACT

In this project we propose to remove from old documents the seepage of ink from one side of a printed page to the other, called bleed-through. Bleed-through severely impairs document readability and makes it difficult to decipher the contents.

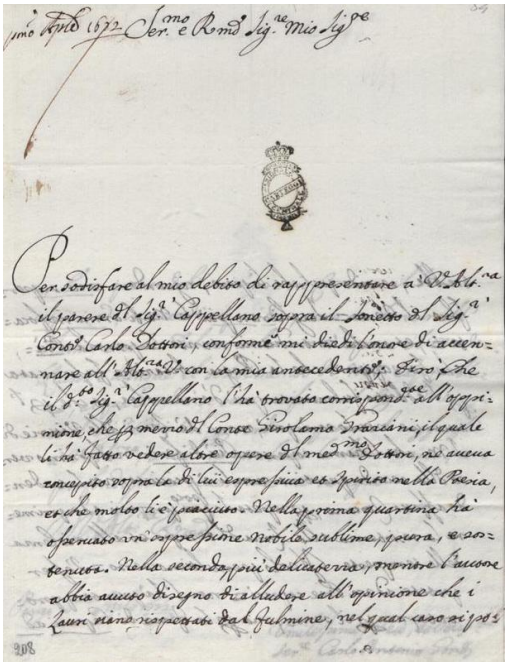


FIG. 1: An example of bleed-through

In the study of this phenomenon we have come across many articles that deal with the subject and propose different removal techniques. The various articles show that there are two types of removal methods

- **Blind method**, one-sided information is used.
- **Non-blind method**, information is acquired from the front and back of the page.

Both methods have defects:

- Blind methods are based on ink intensity, which can be problematic because there is a risk of eliminating "good" parts of the document, such as stamps and annotations.
- Regarding non-blind methods, we observed that the biggest problem concerns the perfect alignment between the recto and verso pages. Obviously, since these are online documents, the scanning may have distorted the actual alignment of the pages.

Most proposed methods are based on non-blind methods.

In order to clean the image, we experimented both types of methods. In a first study we considered a non-blind method. We tried to create a filter that would remove dirt, comparing the two sides of the sheet, precisely subtracting them. We then set ourselves the problem of alignment which we could not be able to solve. Later on, we created a blind method which, through a Gaussian filter applied to the grayscale image, cleans the image from the bleed. We applied that method on 40 images and we created a database to learn a neural network that can automatically clean the image.

II. CLEANING IMAGE

A. Non blind method

In order to remove bleed through we considered Recto and Verso of the image.

- At first we create a mask consisting of a two-dimensional tensor composed initially by two recto and verso images.
- **The mask**: Recto and verso images are sequentially subtracted in order to obtain, in the first part of the two-dimensional tensor, a filter.
- Then we applied the mask to the image we wanted to correct.

```
def maskElab(self,y):
        groups=[]
        groups2=[]
        groups3=[]
        for j in range(2):
            group = layers.Lambda(lambda z:
                z[:,j,:,:])(y)
            groups.append(group)
        group2 = layers.Average()([groups
            [0], groups[1]])
        groups2.append(group2)
        group2 = layers.Subtract()([groups
            [0], groups[1]])
        group2= layers.AveragePooling2D(
            pool_size=(1,1),strides=(1,1))(
            group2)
        groups2.append(group2)
        for gr in groups2:
            gr = layers.Lambda(lambda i : K.
                expand_dims(i, axis=1))(gr)
            groups3.append(gr)
        y= layers.concatenate(groups3,axis
            =1)
```

```
    y= layers.BatchNormalization()(y)
    return y
```

B. Alignment

1. Problems

As we said before, the main problem of this approach is the alignment. We considered recto and verso and we have subtracted them, this operation mean that, for example considering recto as the image we want to clean, the recto's bleed should coincide with verso's written. This doesn't happen when we consider scanned document because, when a document is scanned or photographed, some amount of skew inevitably occurs in these documents. Even automatic image scanners are unable to perfectly align a document so that it is not tilted one way or the other.

2. Proposed Solution

We therefore thought about a solution to align the images extracting only the bleed from one of the two images (e.g. verso) and comparing it with the image cleaned by the bleed of the other direction (e.g. recto). We realized two problems during the process:

- We cannot bring out only the bleed because the contours of the "good" letters of the recto cannot be isolated from the bleed.
- Even if we could bring out only the bleed, we couldn't compare it with the verso image because it has the entire written and we would have an incomplete image. (see figure 3)

C. Blind Method

During the first part we were able to do the opposite of what we had proposed: considering an only image (recto or verso) we got the clean image through processing pixel to pixel. By observing greyscale, we noticed that our document was composed of three levels:

- Black: the written text we wanted to save.
- Grey: bleed and "good" written's boundaries.
- White: background.

Image	Color	Threshold
Written front	black	0-50
Written verso	grey	50-180
Background	white	180-255

Threshold	Normalization
0-50	0
50-180	255
180-255	255

So we have eliminated the bleed, bringing the gray levels corresponding to the background color. We have eliminated pixel-to-pixel processing by assigning the task of bringing the gray pixels to 255 to a Gaussian function, with variance and mean variability depending on the image.

1. Results

As shown in the image 5, we obtained an image cleaned up from the bleed in the background. During cleaning, the edges of the writing have eroded because, as already mentioned, they have the same gray tone as the bleed, but the result is to be considered satisfactory. We have also created a dataset consisting of 40 clean images obtained by appling the Gaussian filter methodology and we used them to instruct our neural network.

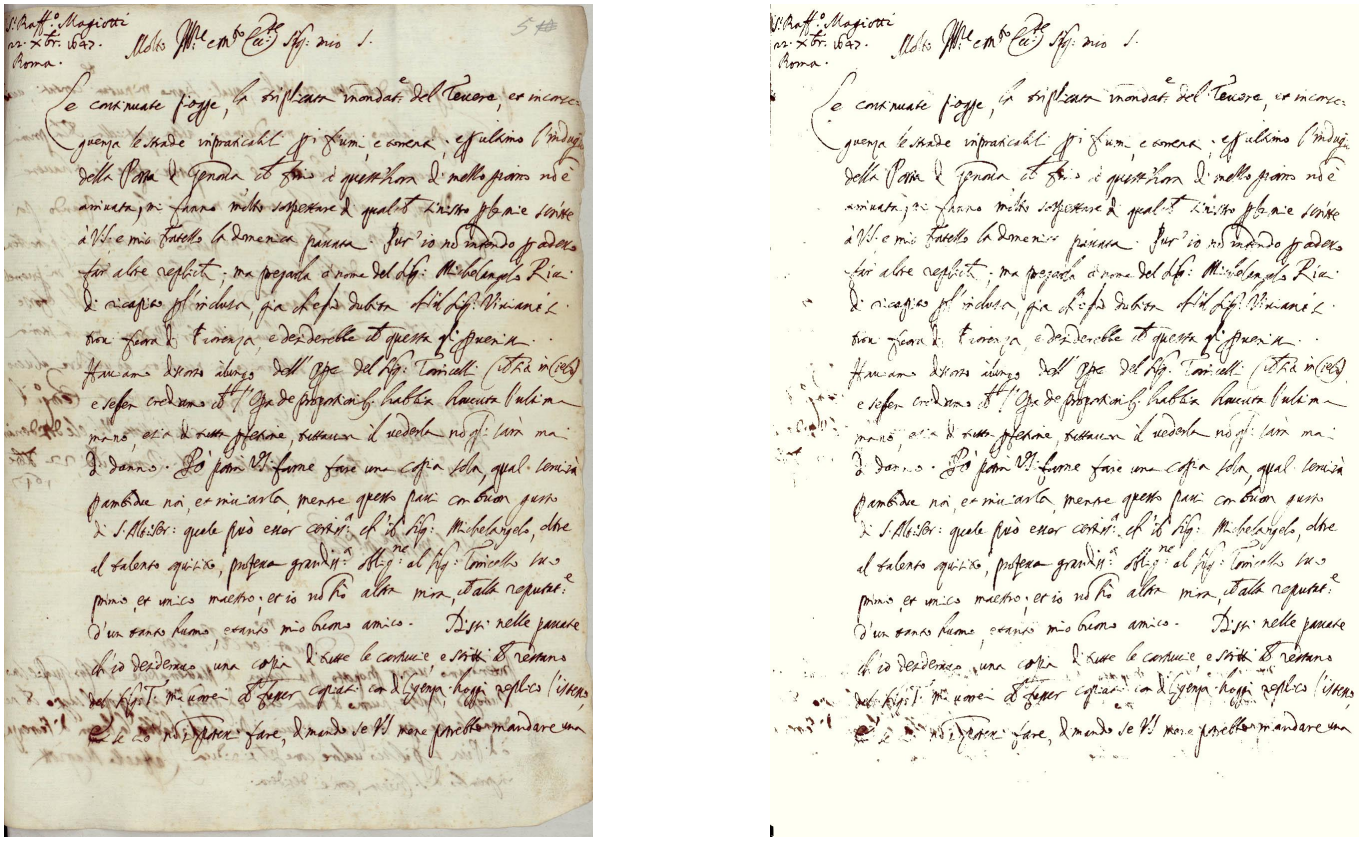
III. NEURAL NETWORK

To learn the network we used autoencoder. An autoencoder is an unsupervised machine learning algorithm that takes an image as input and reconstructs it using fewer number of bits. The main difference between an autoencoder and a general purpose image compression algorithms is that in case of autoencoders, the compression is achieved by learning on a training set of data.

There are two parts to an autoencoder:

- **Encoder**: This is the part of the network that compresses the input into a fewer number of bits. The space represented by these fewer number of bits is called the "latent-space" and the point of maximum compression is called the bottleneck. These compressed bits that represent the original input are together called an "encoding" of the input.
- **Decoder**: This is the part of the network that reconstructs the input image using the encoding of the image.

In the figure 6, we show a vanilla autoencoder (a 2-layer autoencoder with one hidden layer). The input and output layers have the same number of neurons. We feed five real values into the autoencoder which is compressed by the encoder into three real values at the bottleneck (middle layer). Using these three real values, the decoder tries to reconstruct the five real values which we had fed as an input to the network.



(a) Dirty image

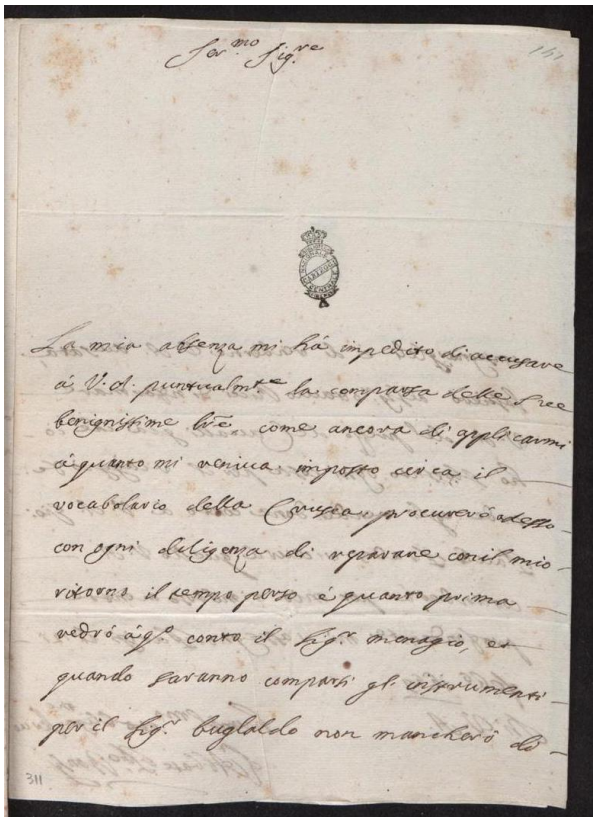
(b) Clean image

FIG. 2: First Elaboration Project

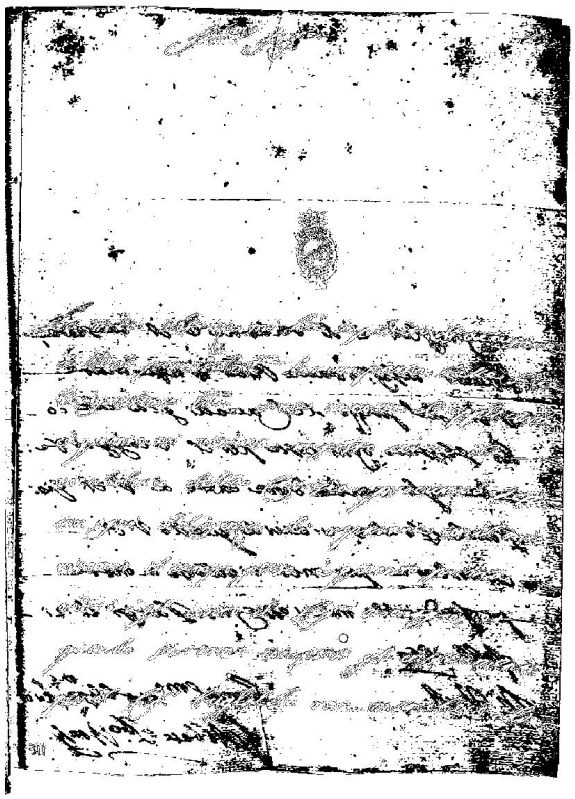
A. First autoencoder

Our first idea was to generate a classic autoencoder as shown in the figure. We applied our encoder to a dataset found in Internet, already divided into patches (100×100). Two sets of images, train and test are provided. These images contain various styles of text, to which synthetic noise has been added to simulate messy and real-world artifacts. The training set includes the noise-free test (train-cleaned). An algorithm must be created to clean the images in the test set. We used a convolutional autoencoder that use a "trick" consisting in replaceing fully connected layers by convolutional layers. These, along with pooling layers, convert the input from wide and thin (eg. 100×100 pixel with 3 channels RGB) to narrow and thick. This helps the network extract visual features from the images, and therefore obtain a much more accurate latent space representation. The reconstruction process uses upsampling and convolutions. Convolutional autoencoders can be useful for reconstruction, for example, learn to remove noise from picture, or reconstruct missing parts.

```
def build_autoencoder(input_size):
    input_img = Input(shape=(input_size,
        input_size,1), name= 'image_input')
    #encoder
    x = Conv2D(32,(3,3), activation='relu',
        padding='same', name='Conv1')(
        input_img)

    x = MaxPooling2D((2,2), padding='same',
        name='pool1')(x)
    x = Conv2D(64, (3,3), activation='relu',
        padding='same', name='Conv2')(x)

    x = MaxPooling2D((2,2), padding='same',
        name='pool2')(x)
    x = Conv2D(64, (3,3), activation='relu',
        padding='same', name='Conv3')(x)

    x = MaxPooling2D((2,2), padding='same',
        name='pool3')(x)
    x = Conv2D(64, (3,3), activation='relu',
        padding='same', name='Conv4')(x)
    #decoder
    x = MaxPooling2D((2,2), padding='same',
        name='pool4')(x)
    x = Conv2D(64, (3,3), activation='relu',
        padding='same', name='Conv5')(x)

    x = UpSampling2D((2,2), name='upsample1'
        )(x)
    x = Conv2D(64, (3,3), activation='relu',
```



(a) Dirty image



(b) Bleed retro on recto

FIG. 3: Alignment problem

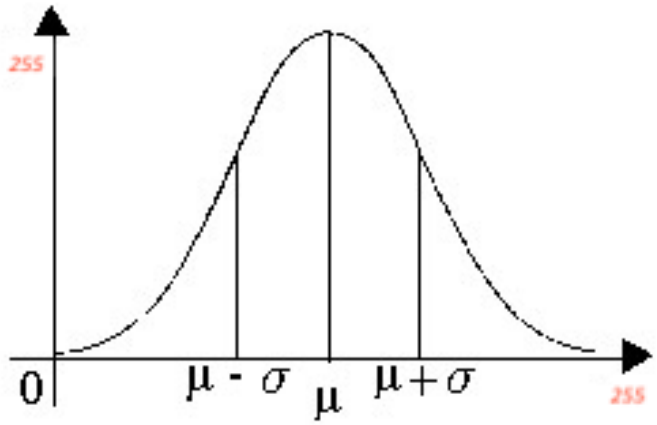
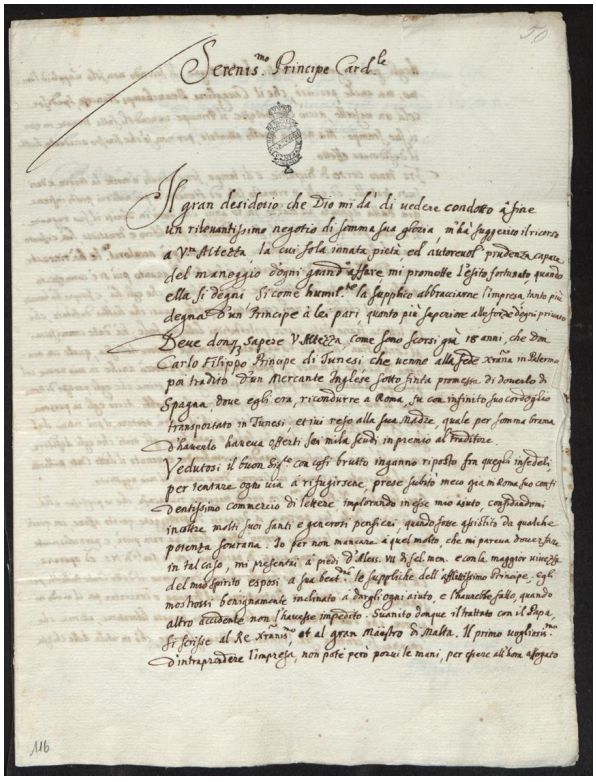


FIG. 4: Gaussian Filter

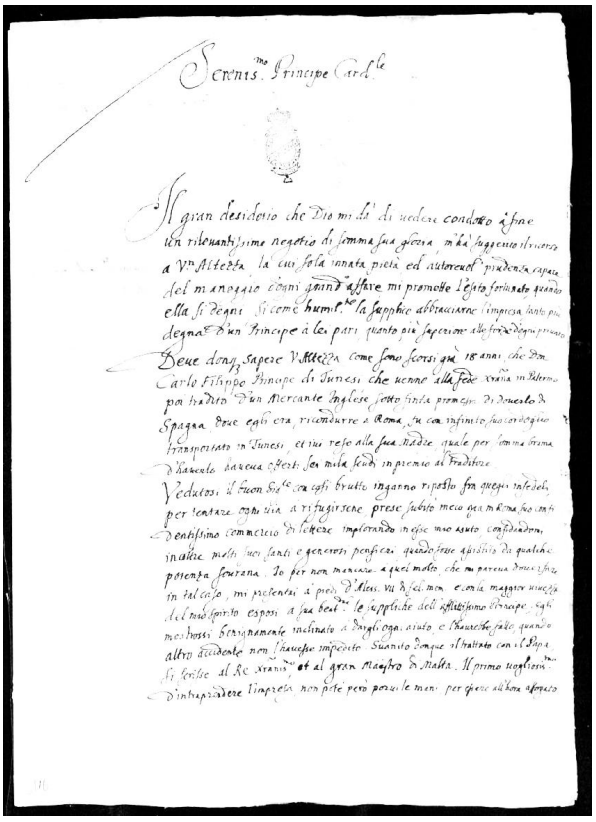
```
        padding='same', name='Conv6')(x)

    x = UpSampling2D((2,2), name='upsample2'
        )(x)
    x = Conv2D(64, (3,3), activation='relu',
        padding='same', name='Conv7')(x)

    x = UpSampling2D((2,2), name='upsample3'
        )(x)
    x = Conv2D(32, (3,3), activation='relu',
        padding='same', name='Conv8')(x)

    x = UpSampling2D((2,2), name='upsample4'
        )(x)
    x = Conv2D(1, (3,3), activation='sigmoid
        ', padding='same', name='Conv9')(x)
    #model
    autoencoder = Model(inputs=input_img,
        outputs=x)
    autoencoder.compile(optimizer='adagrad',
        loss='binary_crossentropy')
    return autoencoder
```

B. Second autoencoder

The results were not satisfactory because the network is composed of multiple layers of convolution operators, end-to-end learning mappings from corrupted images to the original ones. The convolutional layers act as the feature extractor which encode the primary components of image contents while eliminating the corruption. The deconvolutional layers then decode the image abstraction to recover the image content details. We found a study that propose to add skip connections between corresponding



(a) Dirty image



(b) Clean image

FIG. 5: Results

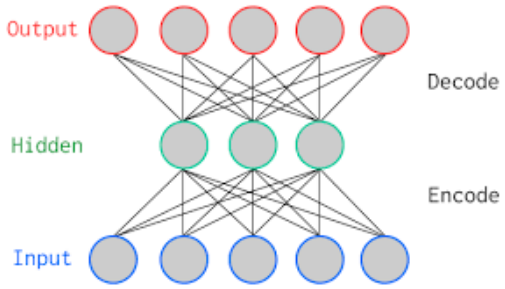


FIG. 6: Autoencoder model

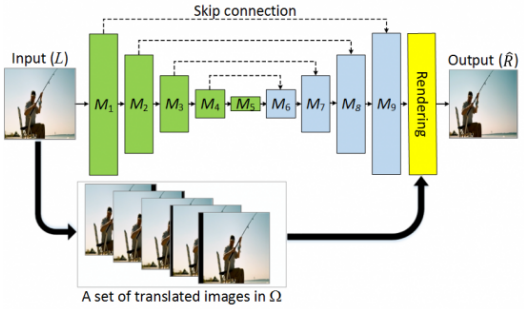
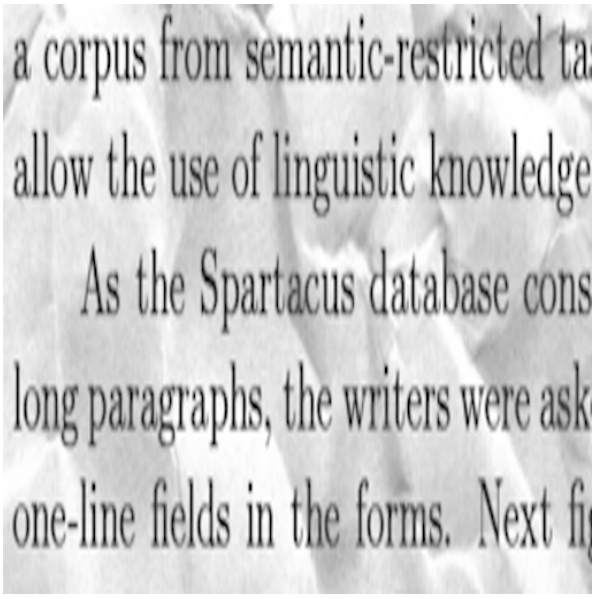


FIG. 7: Skip layers

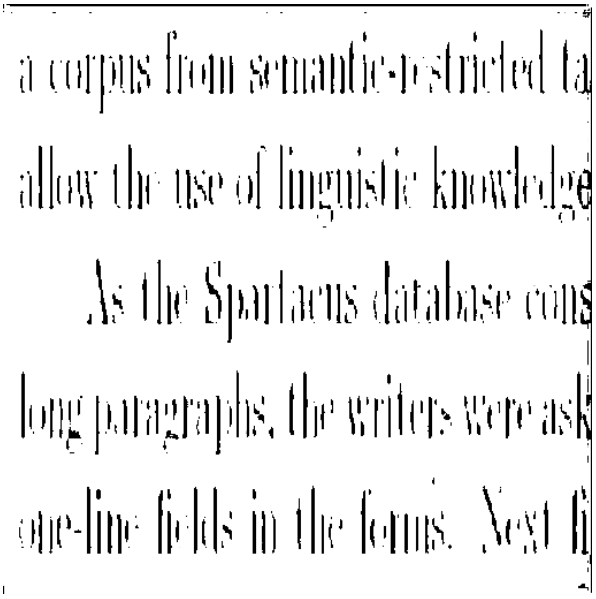
convolutional and de-convolutional layers. Our autoencoder recursively creates convolution levels (in the encoder) and deconvolution levels (in the decoder) and in the meantime it creates a link between the respective levels by adding them together.

```
def recurrent_block(i,y):
    i -=1
    x = MaxPooling2D((2,2), padding='same')(
        y)
    x = Conv2D(64, (3,3), activation='relu',
        padding='same')(x)
    r = x
    r = BatchNormalization()(r)
    if i < 0:
        x = MaxPooling2D((2,2), padding='
            same')(x)
        x = Conv2D(64, (3,3), activation='
            relu', padding='same')(x)
    else:
        x= recurrent_block(i,x)
    x = UpSampling2D((2,2))(x)
    x = Conv2D(64, (3,3), activation='relu',
        padding='same')(x)
    x= BatchNormalization()(x)
    x= Add()([x,r])
    return x
```

Finally we set ourselves the problem of choosing the optimizer and we setted on gradient descent, that is one of the most popular algorithms to perform optimization and the most common way to optimize neural networks. Gradient descent is a way to minimize an objective function J(θ) parameterized by a model's parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ with respect to the parameters. The learning rate determines the size of the steps we take to reach a (local) minimum. At first we used Adam, an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks. We found many studies that demonstrates that to manage with RRN (recurrent networks) rmsprop algorithm was better than the other because these networks have a tendency to either vanish or explode as the energy is propagated through the function. And the effect has a cumulative nature, the more complex the function is, the worse the problem becomes. Rmsprop can resolves the problem. It uses a moving average of squared gradients to normalize the gradient itself. That has an effect of balancing the step size, decrease the step for large gradient to avoid exploding, and increase the step for small gradient to avoid vanishing.

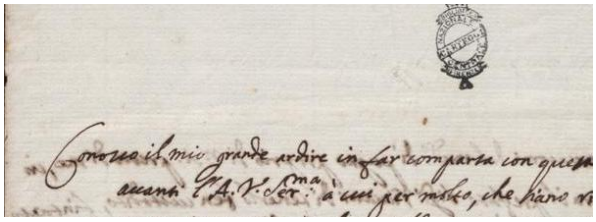


(a) Original

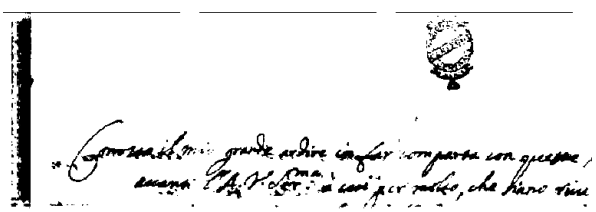


(b) Result first Autoencoder

FIG. 8: First Result



(a) Original



(b) Result second Autoencoder

FIG. 9: Second Result