

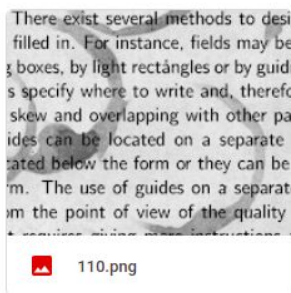
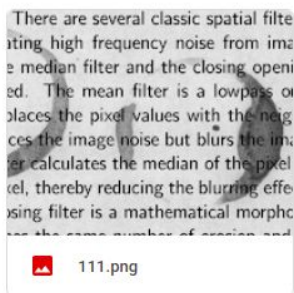
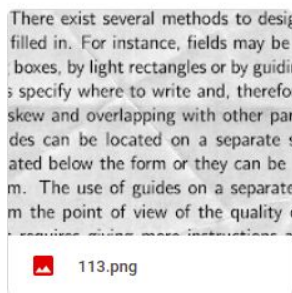
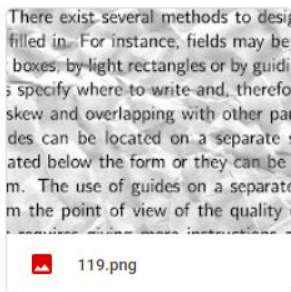
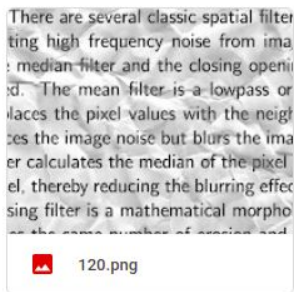
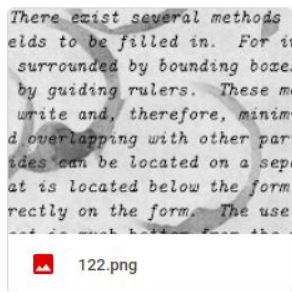
DENOISING AUTOENCODER

Denoised text images (and black and white drawings) using three different types of architecture MLP and CNN based:

1. MLP
2. CNN + post process
3. CNN + 2 post process

Data

The train data has the same text with different damage and fonts.



There are several classic spatial filters for removing high frequency noise from images. The mean filter and the closing opening filter are frequently used. The mean filter is a lowpass or smoothing filter that replaces the pixel values with the neighborhood mean. It reduces the image noise but blurs the image edges. The median filter calculates the median of the pixel neighborhood for each pixel, thereby reducing the blurring effect. Finally, the opening closing filter is a mathematical morphological operation that combines the same number of erosion and dilation operations in order to eliminate small objects from images. The main goal was to train a neural network to obtain a clean image from a noisy one. In this case, it was much easier to obtain a simulated noisy image than to clean a subset of noisy images. The simulated noisy images follows this scheme:

There are several classic spatial filters for removing high frequency noise from images. The mean filter and the closing opening filter are frequently used. The mean filter is a lowpass or smoothing filter that replaces the pixel values with the neighborhood mean. It reduces the image noise but blurs the image edges. The median filter calculates the median of the pixel neighborhood for each pixel, thereby reducing the blurring effect. Finally, the opening closing filter is a mathematical morphological operation that combines the same number of erosion and dilation operations in order to eliminate small objects from images. The main goal was to train a neural network to obtain a clean image from a noisy one. In this case, it was much easier to obtain a simulated noisy image than to clean a subset of noisy images. The simulated noisy images follows this scheme:

Architecture 1: MLP (Multilayer perceptron)

- Not very effective
- Overfitting

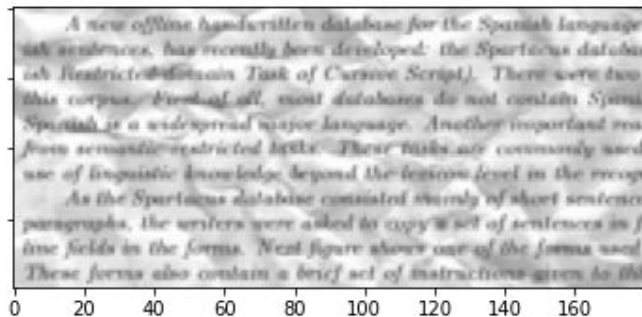
```
37 class Autocoder(nn.Module):
38     def __init__(self):
39         super(Autocoder, self).__init__()
40         self.encoder = Encoder()
41         self.Decoder = Decoder()
42
43     def forward(self, image):
44         z = self.encoder(image)
45         out = self.Decoder(z)
46         return out
47
48     def reiniciar(self):
49         super(Autocoder, self).__init__()
50         self.encoder = Encoder()
51         self.Decoder = Decoder()
```

```
1 class Encoder(nn.Module):
2     def __init__(self):
3         super(Encoder, self).__init__()
4         self.f1 = nn.Sequential(
5             nn.Linear(180*80, 180*10),
6             nn.ReLU()
7         )
8
9         self.f2 = nn.Sequential(
10             nn.Linear(180*10, 180*10),
11             nn.ReLU()
12         )
13
14     def forward(self, image):
15         out = self.f1(image)
16         z = self.f2(out)
17         return z
18
19 class Decoder(nn.Module):
20     def __init__(self):
21         super(Decoder, self).__init__()
22
23         self.f1 = nn.Sequential(
24             nn.Linear(180*10, 180*10),
25             nn.ReLU()
26         )
27
28         self.f5 = nn.Sequential(
29             nn.Linear(180*10, 180*80),
30         )
31
32     def forward(self, z):
33         out = self.f1(z)
34         out = torch.tanh(self.f5(z))
35         return out
```

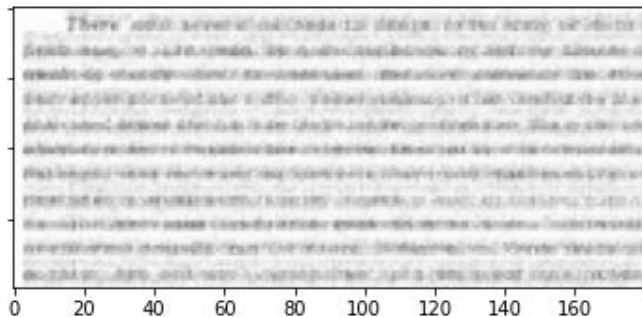
Results

You may notice that there is a overfitting: The original image starts with “A new” but the output has a “There” at the beginning, that is the same text as the train data.

Original image:



The image after the autoencoder:



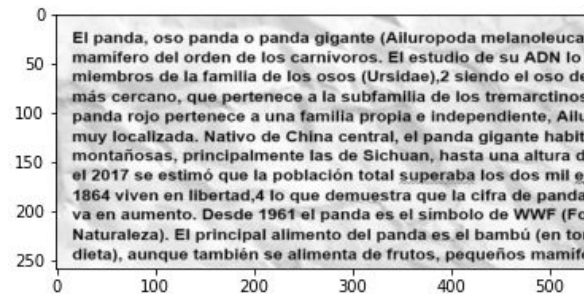
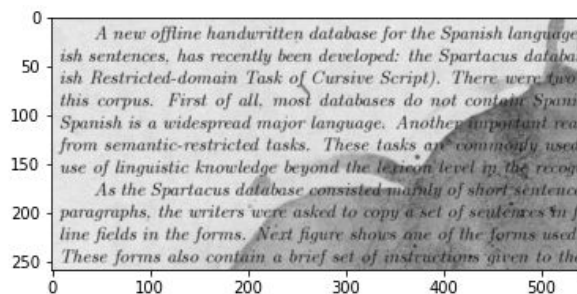
Architecture 2: CNN (Convolutional neural network)

The architecture 2 is a CNN autoencoder that can denoising images, but the output does not look that great. That's why it is added another CNN, called Process, that makes the image aesthetic.

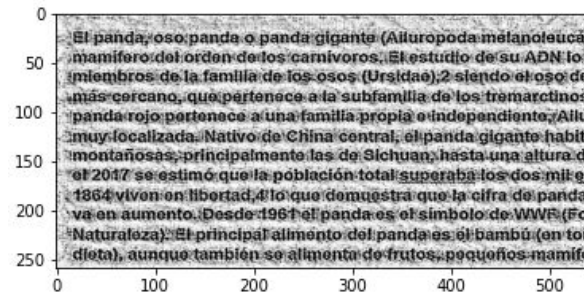
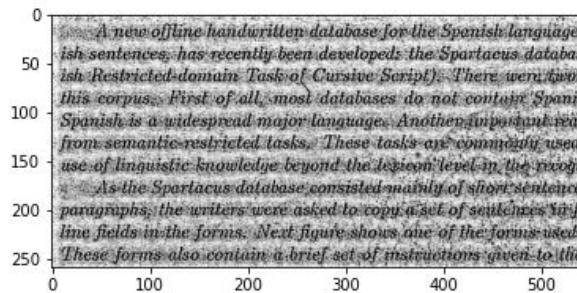
```
class Autoncoder(nn.Module):  
    def __init__(self):  
        super(Autoncoder,self).__init__()  
        self.encoder = Encoder()  
        self.Decoder = Decoder()  
        self.Process = Process()  
    def forward(self, image):  
        z = self.encoder(image)  
        out = self.Decoder(z)  
        out = self.Process(out)  
        return out  
    def reiniciar(self):  
        super(Autoncoder,self).__init__()  
        self.encoder = Encoder()  
        self.Decoder = Decoder()  
        self.Process = Process()
```

Results

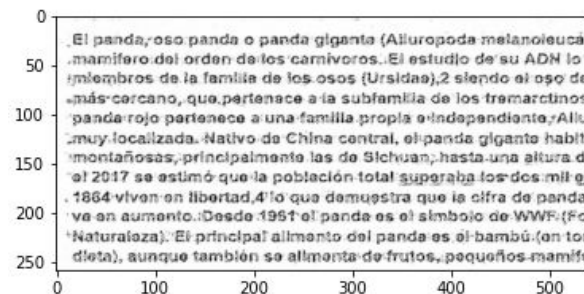
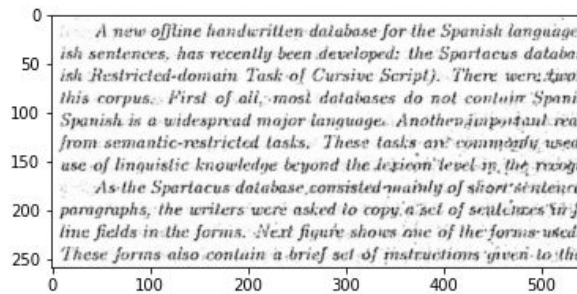
Original image:



The image after
the autoencoder:

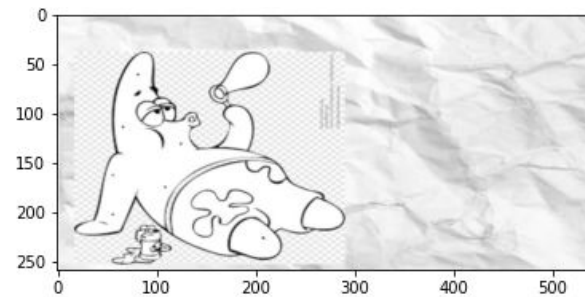


The image after
Process:

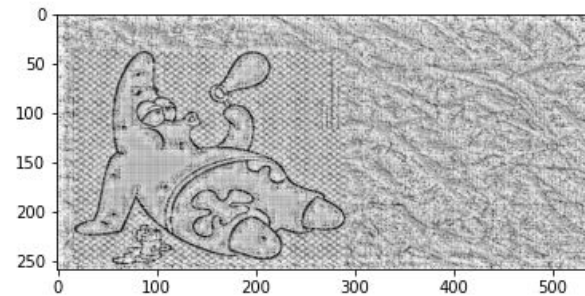


You may observe that the autoencoder makes the image background homogeneous, and highlights the text (and the lines of the drawings). After the autoencoder, Process clear up the background.

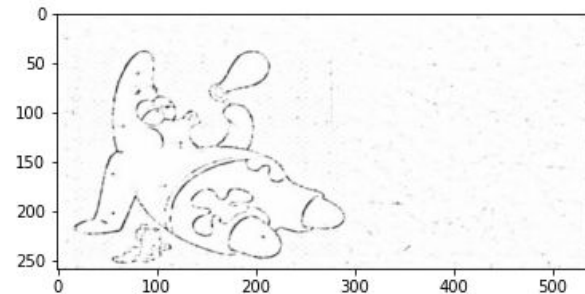
Original image:



The image after the autoencoder:



The image after Process:



Architecture 3: CNN (Convolutional neural network)

Architecture 3 follow the same idea of architecture 2, but has two Process to improve the results. And it works.

Keep in mind that both Process are CNN and they get trained as a whole to archive clearing the background, so the result images after each Process will be different that the previous one.

```
class Autoncoder(nn.Module):  
    def __init__(self):  
        super(Autoncoder, self).__init__()  
        self.encoder = Encoder()  
        self.Decoder = Decoder()  
        self.Process = Process()  
        self.Process2 = Process()  
    def forward(self, image):  
        z = self.encoder(image)  
        out = self.Decoder(z)  
        out = self.Process(out)  
        out = self.Process2(out)  
        return out  
    def reiniciar(self):  
        super(Autoncoder, self).__init__()  
        self.encoder = Encoder()  
        self.Decoder = Decoder()  
        self.Process = Process()  
        self.Process2 = Process()
```


Results

Original image:

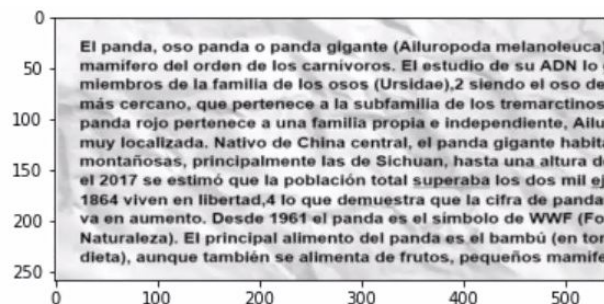


Image after the autoencoder:

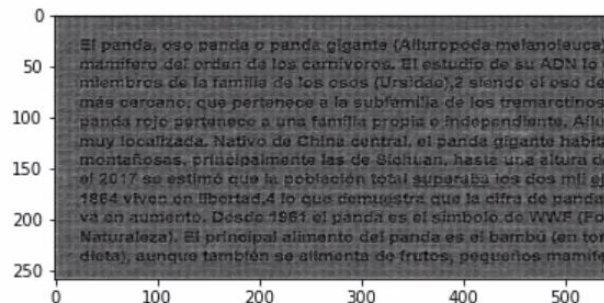


Image after the first Process:

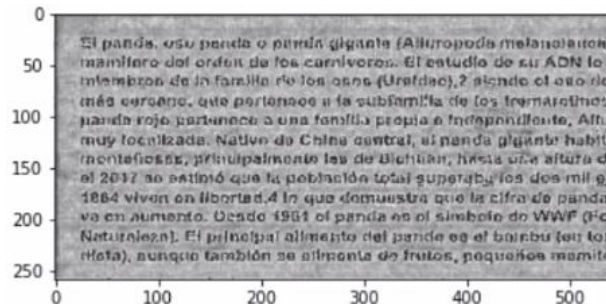
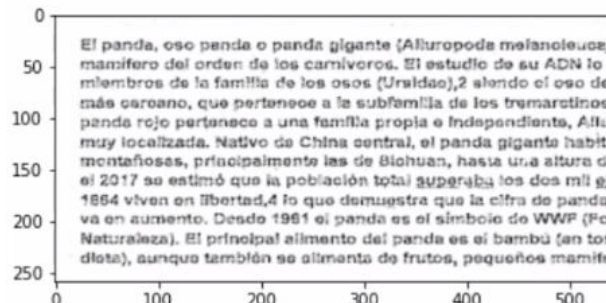


Image after the second process:



Original image:

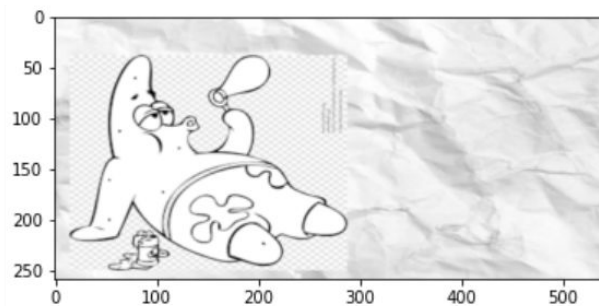


Image after the
autoencoder:

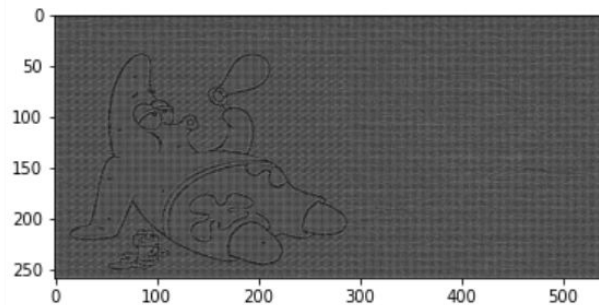


Image after the
first Process:

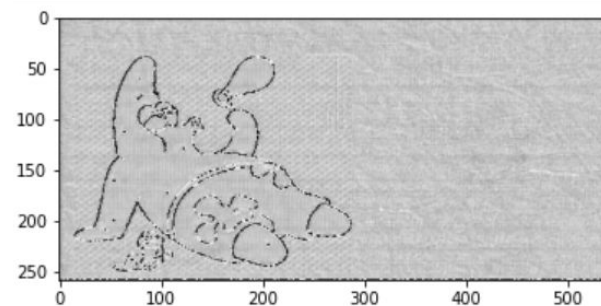
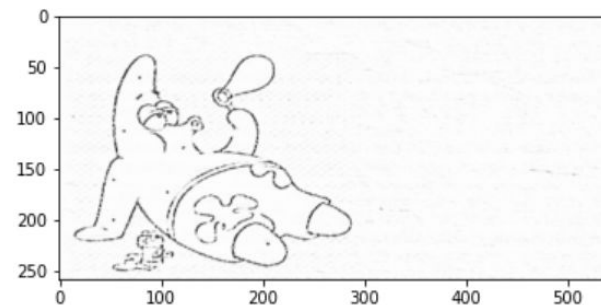
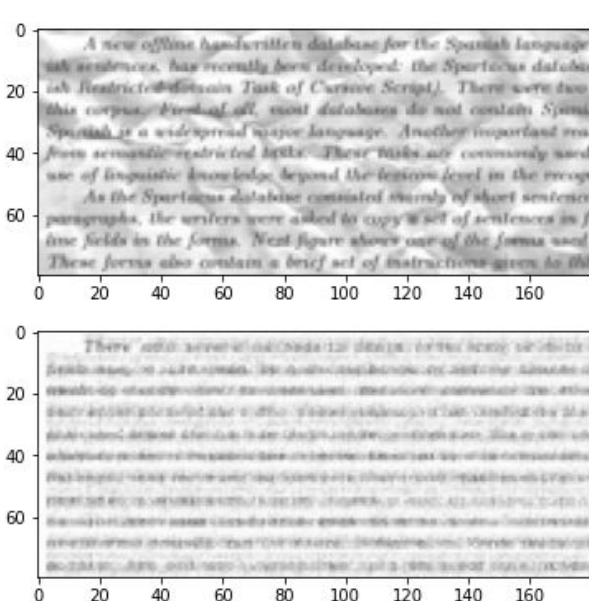


Image after the
second process:

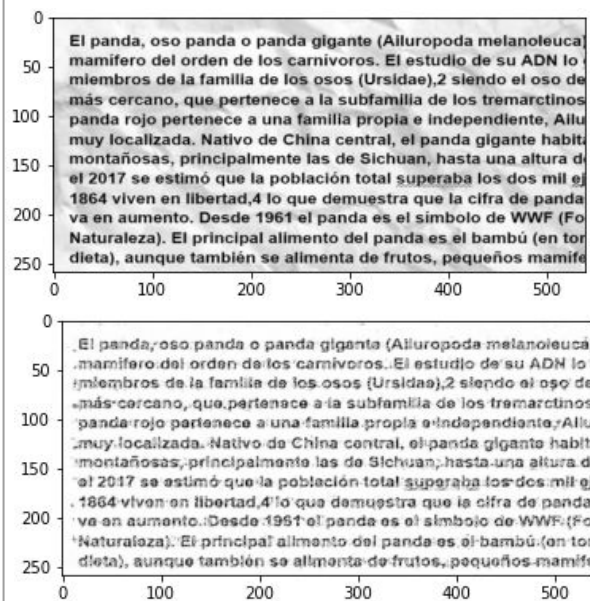


Comparison

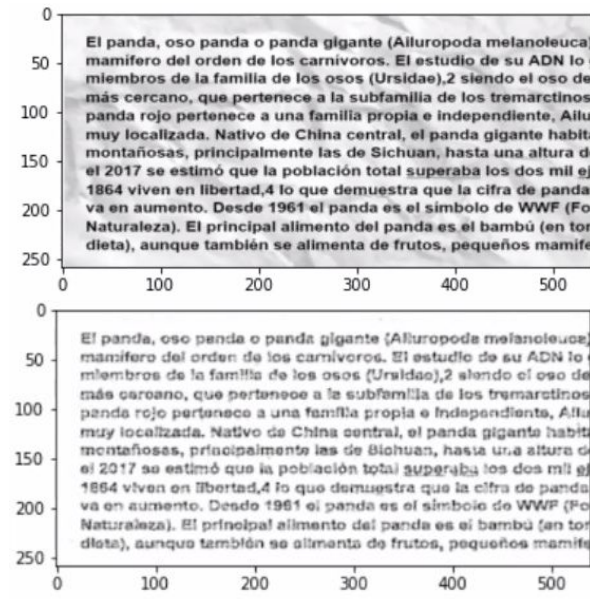
MLP




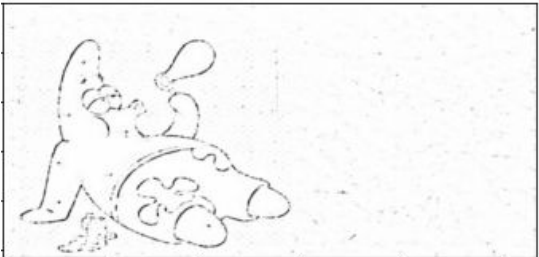

CNN+ Process



CNN + 2 Process



Comparison

Original image	Autoencoder 2: CNN + P	Autoencoder 3: CNN + 2P
		

Notes

- For the encoders, dimensional reduction strongly affect the training time and results. It was observed that the better options were dimension reduction (only 10 dimensions for the encoder output)
- For the decoders, error is important for better results as it is common that a CNN get stuck between values and so not get a good performance. This situation can be recognized when the output is a complete white image. As a error function compare each pixel of the image, a complete white image will has low error because we look to clear the background.

Future work

- For upgrade the autoencoders, it is need to explore another types of error functions that give more importance to little differences between predictions and expected outputs. Because, as mentioned earlier, the neural network could get stuck and produce empty images.