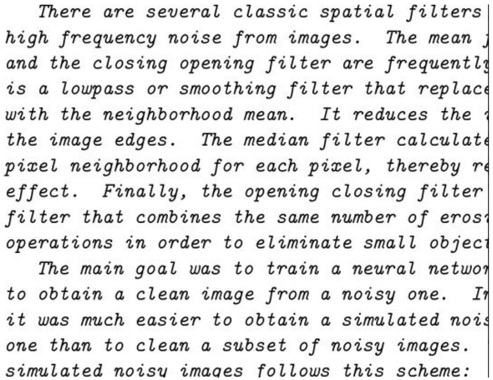
DENOISING AUTOENCODER

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

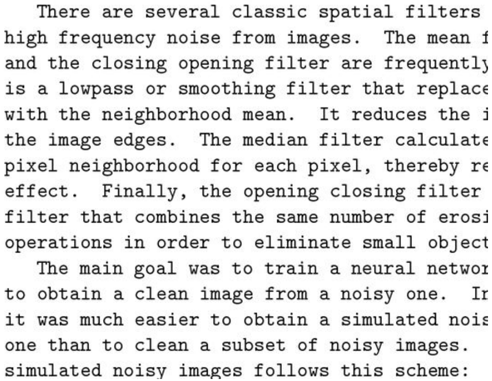
1. MLP
2. CNN + post process
3. CNN + 2 post process

Data

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



There are several classic spatial filters
high frequency noise from images. The mean f
and the closing opening filter are frequently
is a lowpass or smoothing filter that replace
with the neighborhood mean. It reduces the i
the image edges. The median filter calculate
pixel neighborhood for each pixel, thereby re
effect. Finally, the opening closing filter
filter that combines the same number of erosi
operations in order to eliminate small object
The main goal was to train a neural networ
to obtain a clean image from a noisy one. In
it was much easier to obtain a simulated nois
one than to clean a subset of noisy images.
simulated noisy images follows this scheme:



There are several classic spatial filters
high frequency noise from images. The mean f
and the closing opening filter are frequently
is a lowpass or smoothing filter that replace
with the neighborhood mean. It reduces the i
the image edges. The median filter calculate
pixel neighborhood for each pixel, thereby re
effect. Finally, the opening closing filter
filter that combines the same number of erosi
operations in order to eliminate small object
The main goal was to train a neural networ
to obtain a clean image from a noisy one. In
it was much easier to obtain a simulated nois
one than to clean a subset of noisy images.
simulated noisy images follows this scheme:

Architecture 1: MLP (Multilayer perceptron)

- Not very effective
- Overfitting

```python
class Encoder(nn.Module):
  def __init__(self):
    super(Encoder,self).__init__()
    self.f1 = nn.Sequential(
        nn.Linear(180*80,180*10),
        nn.ReLU()
    )

    self.f2 = nn.Sequential(
        nn.Linear(180*10,180*10),
        nn.ReLU()
    )

  def forward(self, image):
    out = self.f1(image)
    z = self.f2(out)
    return z

class Decoder(nn.Module):
  def __init__(self):
    super(Decoder,self).__init__()

    self.f1 = nn.Sequential(
        nn.Linear(180*10,180*10),
        nn.ReLU()
    )

    self.f5 = nn.Sequential(
        nn.Linear(180*10,180*80),
    )

  def forward(self, z):
    out = self.f1(z)
    out = torch.tanh(self.f5(z))
    return out
```

```python
class Autoncoder(nn.Module):
  def __init__(self):
    super(Autoncoder,self).__init__()
    self.encoder = Encoder()
    self.Decoder = Decoder()

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

```

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:

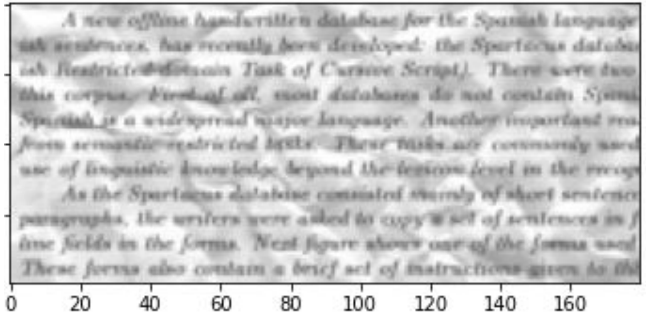
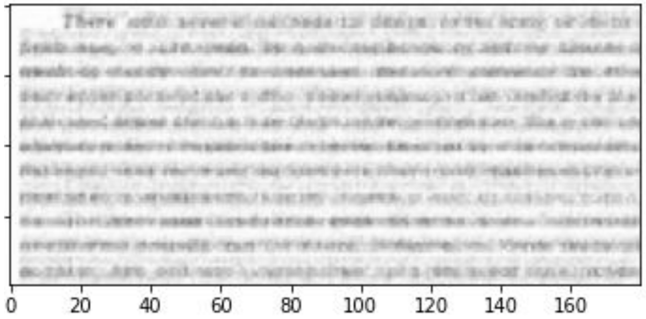


Image after the autoencoder:



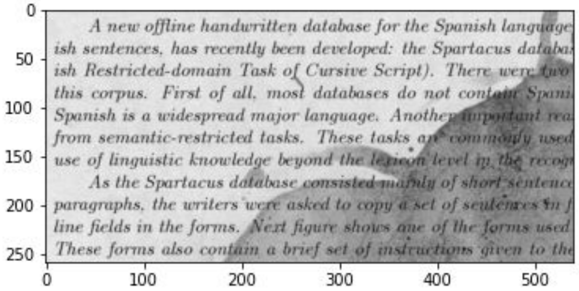
Architecture 2: CNN (Convolutional neural network)

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:



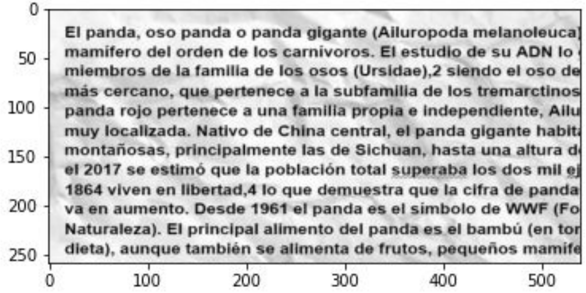
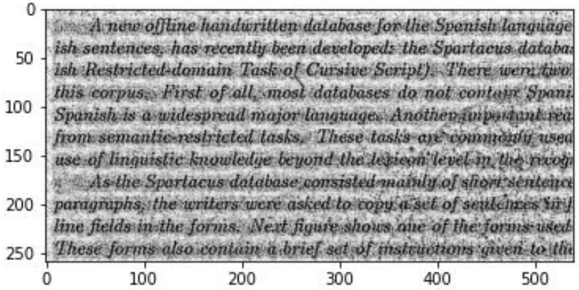


Image after the autoencoder:



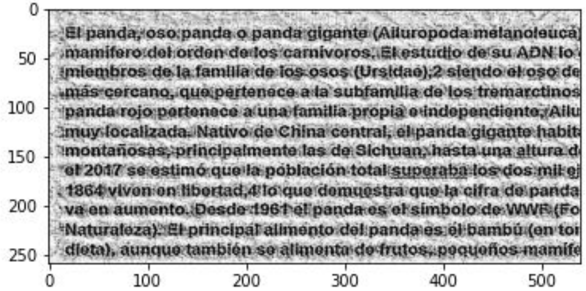
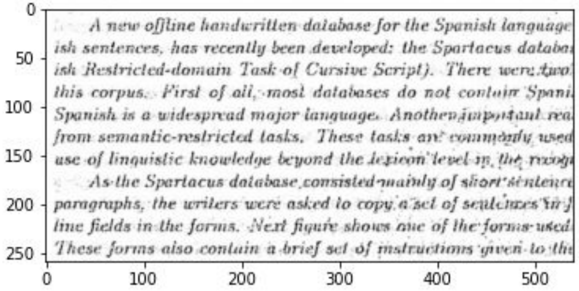
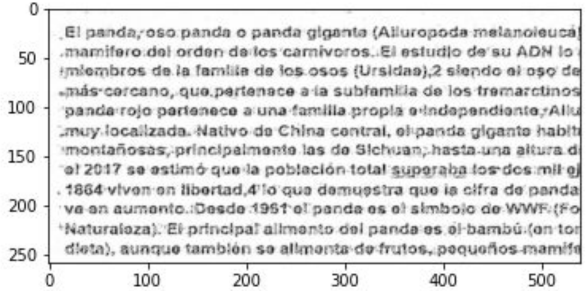


Image after Process:





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:

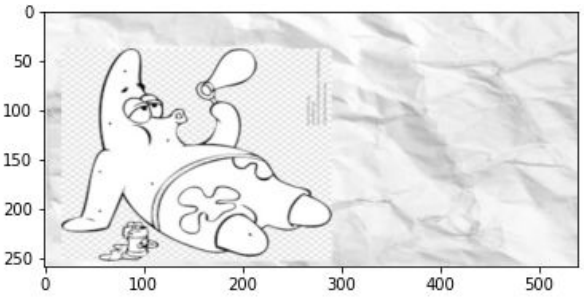


Image after the autoencoder:

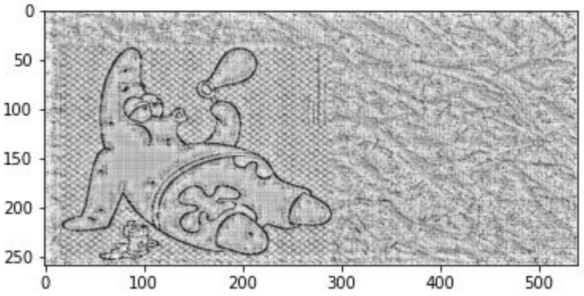
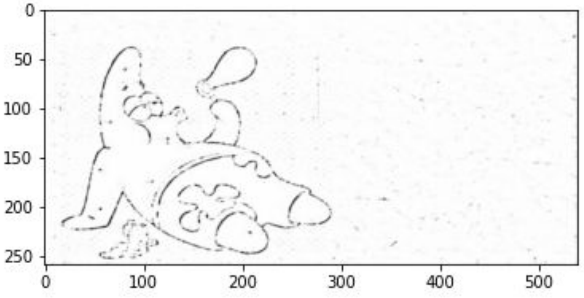


Image after Process:



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:

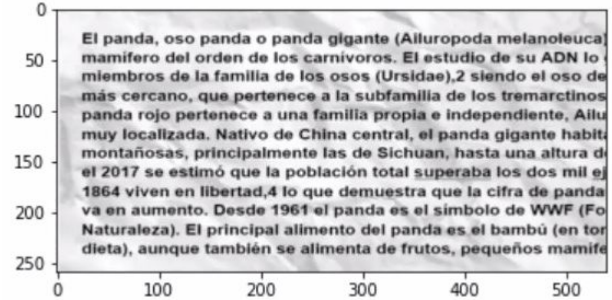


Image after the first Process:

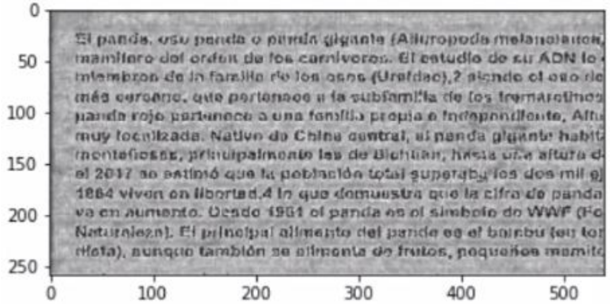


Image after the autoencoder:

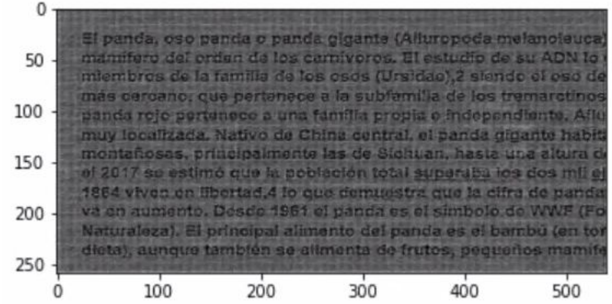
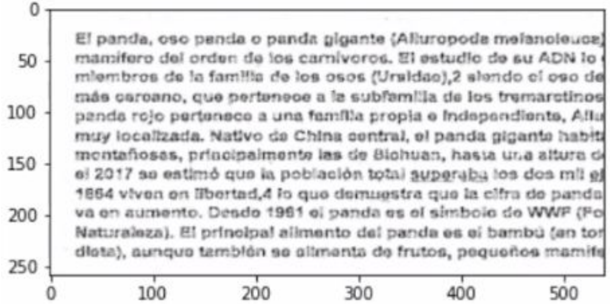


Image after the second process:



Original image:

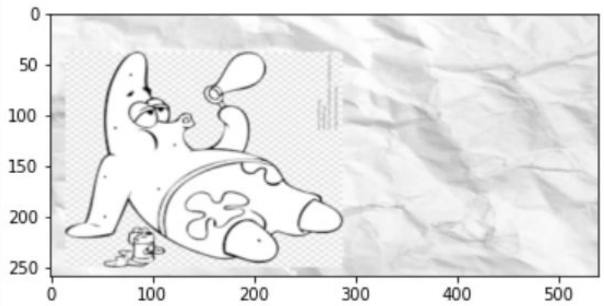


Image after the first Process:

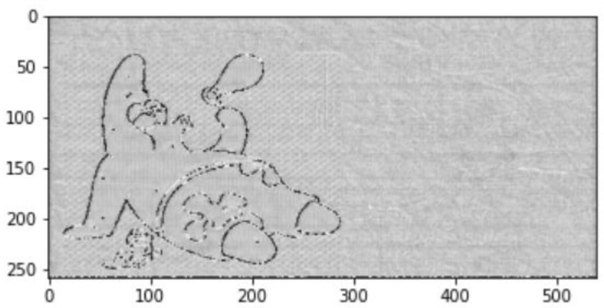


Image after the autoencoder:

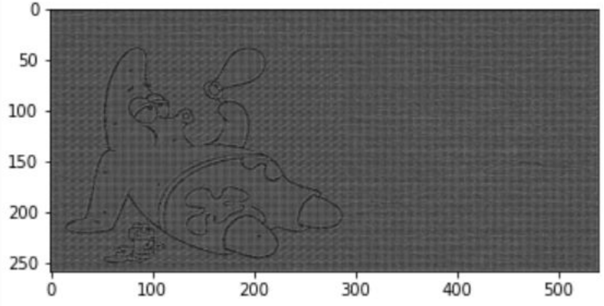
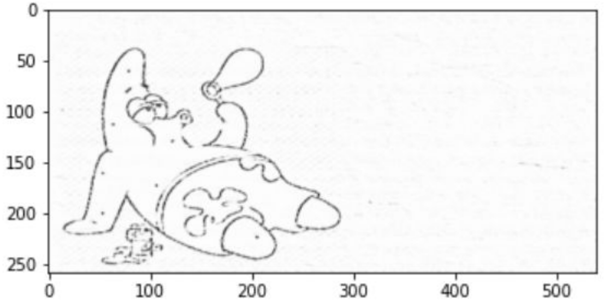
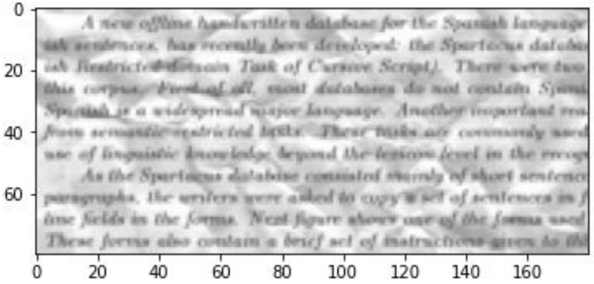


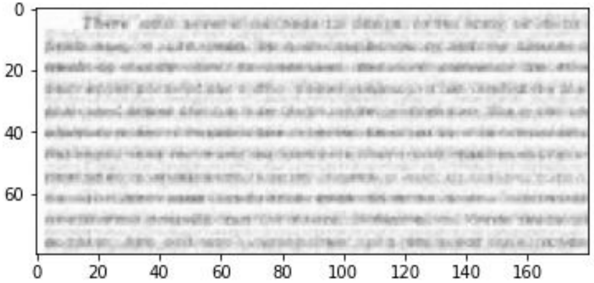
Image after the second process:



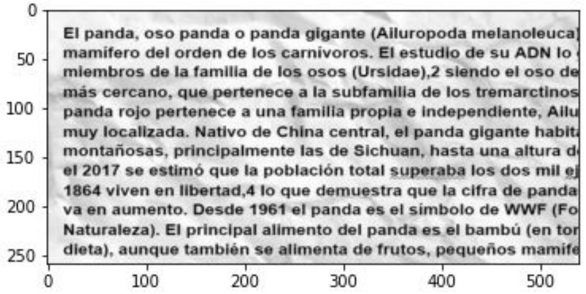
Results

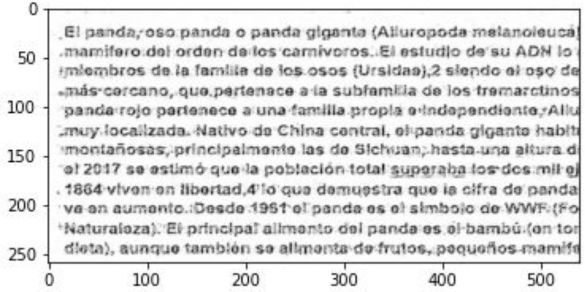
<table>
<tr><th>MLP</th><th>CNN+ Process</th><th>CNN + 2 Process</th></tr>
<tr>
<td>



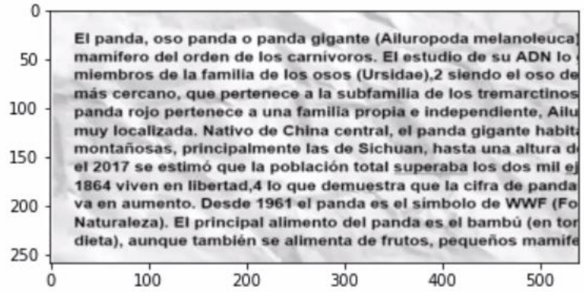


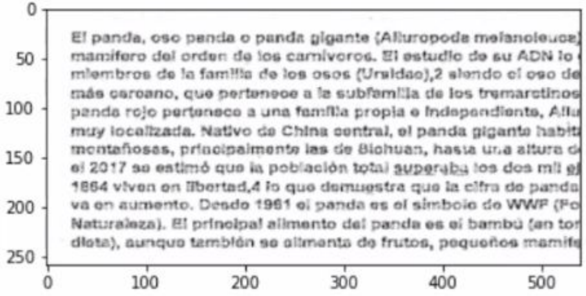
</td>
<td>





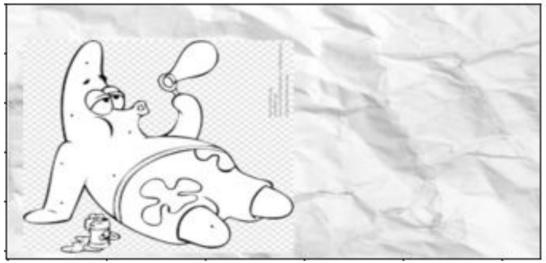
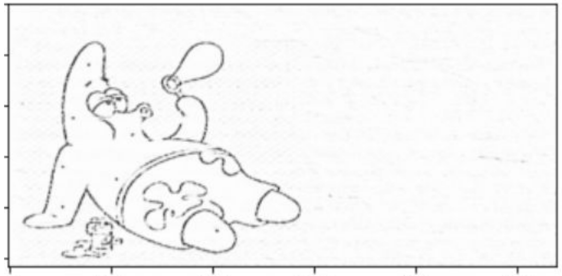
</td>
<td>





</td>
</tr>
</table>

Results

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.