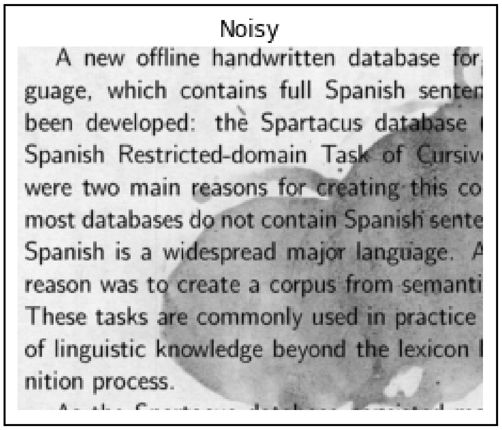
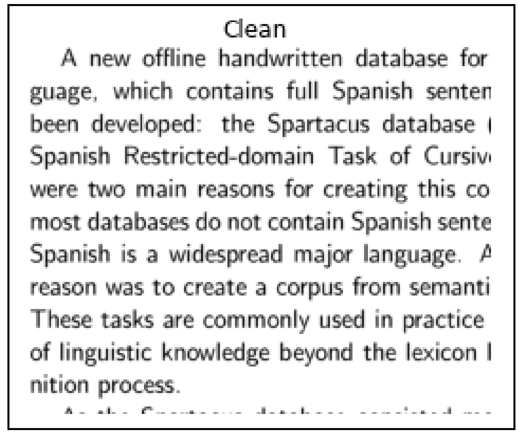
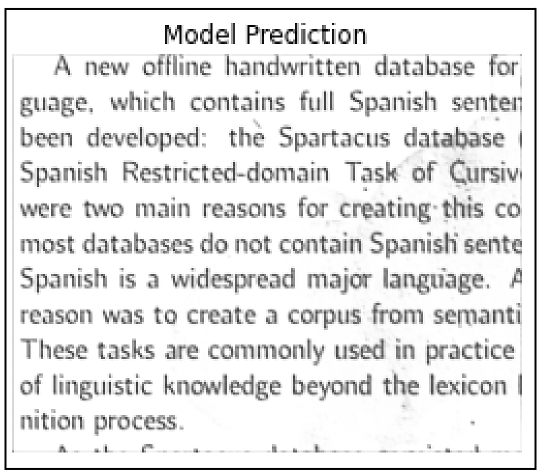
Denoising with Autoencoders





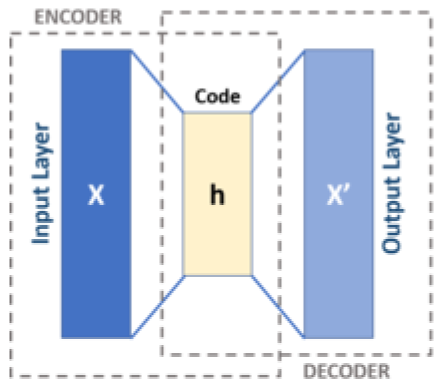


Shubham Kaushal
Contact: +91 7045 603 792
shubhamkaushal765@gmail.com
Gmail | LinkedIn | GitHub

Table of Contents

AutoEncoders

An autoencoder is a type of artificial neural network used to learn efficient codings of data. It learns a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore insignificant data (“noise”).



source: Wikipedia

It has two main parts: an encoder that maps the input into the code and a decoder that maps the code to reconstruct the input.

Dataset

Dataset downloaded from: https://archive-beta.ics.uci.edu/ml/datasets/NoisyOffice

NoisyOffice is a multivariate dataset hosted by the UC Irvine ML repository. It consists of 54 images of clean documents and 216 images of tampered documents (noise is introduced in four different ways).

The task is to design an autoencoder using convolutional neural networks (CNNs), to be able to eliminate the different noises in the images (denoising). Here I have built two different models: the first one is a basic CNN with one encoding layer, one decoding layer followed by an output layer. The second one is deep CNN with three encoding and three decoding layers followed by an output layer.

Preprocessing

Looking at the dataset, there are four types of noises for each clean image. Hence mapping and separation of images are required. Each noisy image is named as :

FontABC_NoiseD_EE.png

where FontABC: name of the font; NoiseD: type of noise;
EE: TR->Training set, TE->Testing set, VA-> Validation set

Example:

Names of noisy Images

```
['FontLre_Noisec_TE.png',
 'FontLre_Noisec_TR.png',
 'FontLre_Noisec_VA.png',
 'FontLre_Noisef_TE.png',
 'FontLre_Noisef_TR.png']
```

Names of clean images

```
['FontLre_Clean_TE.png',
 'FontLre_Clean_TR.png',
 'FontLre_Clean_VA.png',
 'FontLrm_Clean_TE.png',
 'FontLrm_Clean_TR.png']
```

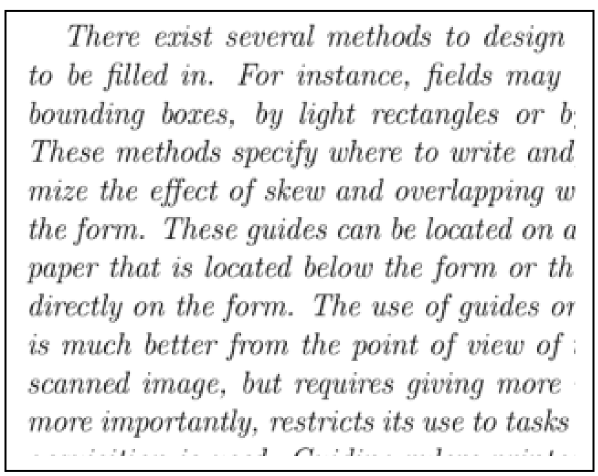
Here are the preprocessing steps:

1) Separating the names into separate lists of training, validation, and testing set.
2) Splitting the dataset based on the index of the names in the list.
3) As the number of Noisy images is four times that of the Clean images, mapping is required from noisy images to clean images.

Data Insights

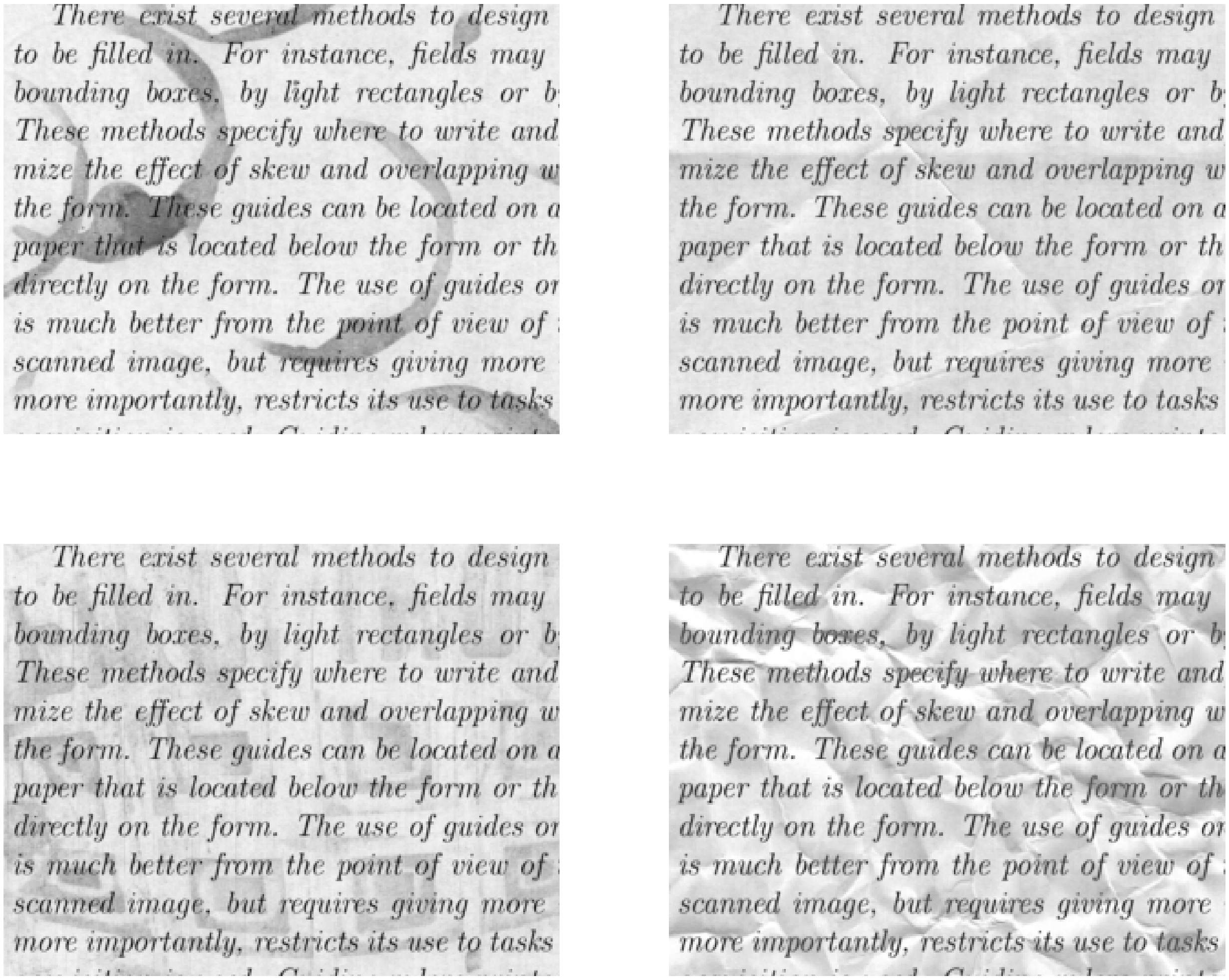
After the preprocessing steps, let us look at a clean image and the different noises that have been introduced. We see that three of the four noises are introduced by crushing the paper, and the remaining noise is due to coffee stains. The cropping and the font face matched (as given), making it easier to train the models.

Clean Image:



There exist several methods to design to be filled in. For instance, fields may bounding boxes, by light rectangles or b These methods specify where to write and mize the effect of skew and overlapping w the form. These guides can be located on a paper that is located below the form or th directly on the form. The use of guides or is much better from the point of view of scanned image, but requires giving more more importantly, restricts its use to tasks

Noisy Images for the above clean image:



There exist several methods to design
to be filled in. For instance, fields may
bounding boxes, by light rectangles or b
These methods specify where to write and
mize the effect of skew and overlapping w
the form. These guides can be located on a
paper that is located below the form or th
directly on the form. The use of guides or
is much better from the point of view of
scanned image, but requires giving more
more importantly, restricts its use to tasks

Basic AutoEncoder [CNN]

To get a basic understanding of the performance, a basic model is defined. This allows to quickly gauge the pros and cons of using a model based on CNN. Many different choices of optimizers are also available to choose from. For the basic model, Adam optimizer is used as it usually performs well. The following figure shows the model summary:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 420, 540, 8)       80
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 420, 540, 8)       584
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 420, 540, 1)       73
=================================================================
Total params: 737
Trainable params: 737
Non-trainable params: 0
_________________________________________________________________
```

The model consists of three convolutional layers. The first layer corresponds to the encoder. The next layer corresponds to the decoder. The last layer is the output layer of the model.

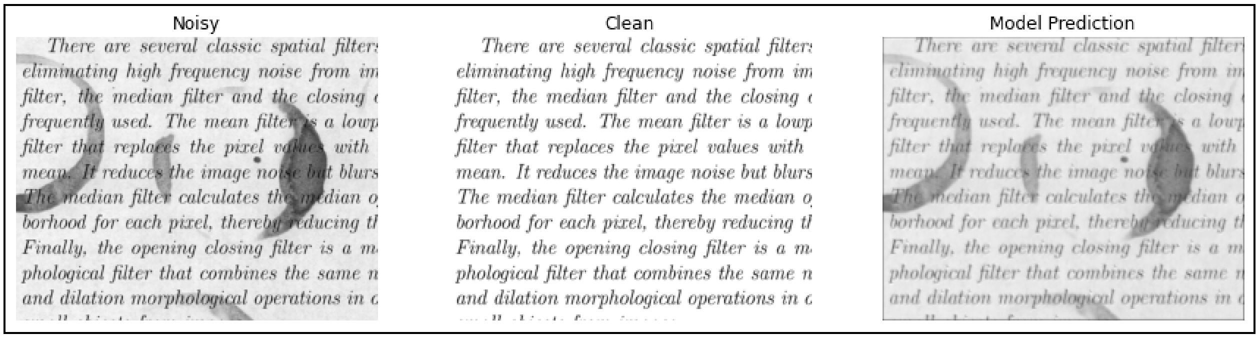
Performance of Basic Model

From the following figure, we can see the model's loss parameter on the training and the validation instances:

```
Epoch 5/10
3/3 [==============================] - 19s 6s/step - loss: 0.6319 - val_loss: 0.6155
Epoch 6/10
3/3 [==============================] - 19s 6s/step - loss: 0.6114 - val_loss: 0.5932
Epoch 7/10
3/3 [==============================] - 19s 6s/step - loss: 0.5889 - val_loss: 0.5688
Epoch 8/10
3/3 [==============================] - 19s 6s/step - loss: 0.5643 - val_loss: 0.5423
Epoch 9/10
3/3 [==============================] - 19s 6s/step - loss: 0.5375 - val_loss: 0.5135
Epoch 10/10
3/3 [==============================] - 19s 6s/step - loss: 0.5084 - val_loss: 0.4829
```

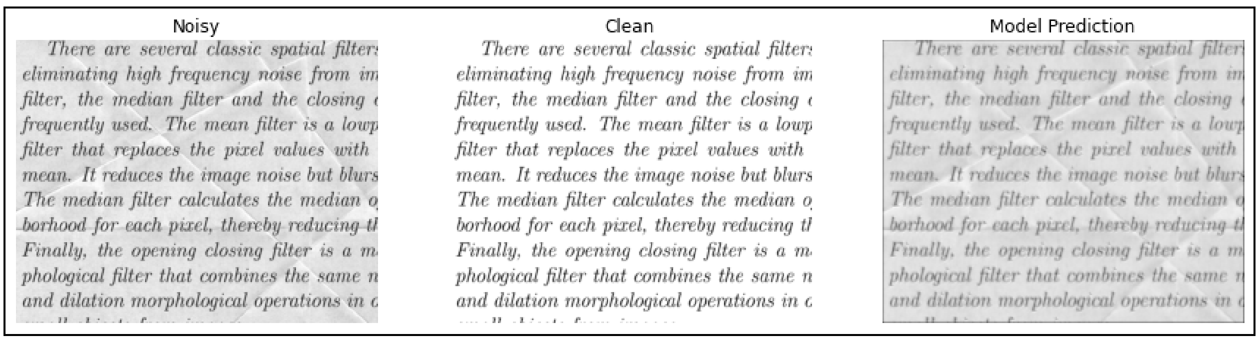
The model has been trained for 10 epochs. The figure above shows the loss in the last six epochs. The loss has reduced quite significantly from 0.61 to 0.48.

Now let us look at the denoised images predicted by the model. From the following figure, we can see the model's output on the validation set:



We notice that the texts have been preserved quite well. The coffee stains have been lightened, but the basic model is unable to eliminate the noise.

Looking at the model’s performance on the noise by paper crushing:



The text is lightened and blurry. The noise is less, but the background has also become gray, which is undesirable. To ameliorate the situation, we do one or more of the following:

- Get more training datasets. This cannot be done in this case as we have used all the available datasets for training. We can concatenate the training and the validation set and use them for training, but it will require more training time. This method is used to train the last model.
- Artificially introduce noise to increase the dataset (Data Augmentation). Also, the addition of different noises from the provided dataset can be used. This technique is not used for this project.
- Make more complex and deep models. As we see that CNN models are quite capable of reducing the noises, we have utilized this technique in the following section.

Autoencoder using Deep CNN

Now that the performance of a basic autoencoder using CNN is understood. Let us build a complex model. The following figure shows the model summary:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 420, 540, 32)      320
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 420, 540, 16)      4624
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 420, 540, 8)       1160
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 420, 540, 8)       584
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 420, 540, 16)      1168
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 420, 540, 32)      4640
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 420, 540, 1)       289
=================================================================
Total params: 12,785
Trainable params: 12,785
Non-trainable params: 0
_________________________________________________________________
```

The model consists of seven convolutional layers. The first three layers correspond to the encoder. The next three corresponds to the decoder. The last layer is the output layer of the model.

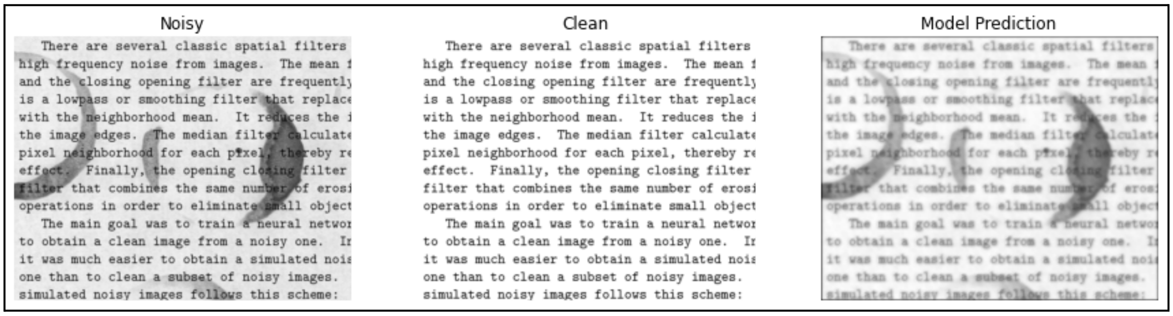
Optimizers

Three optimizers are used: adam, RMSprop, and SGD. After training and predicting from the models, the losses and the prediction on the validation set are as follows:

- Adam: The losses on the training and the validation set are:

```
Epoch 1/5
3/3 [==============================] - 104s 32s/step - loss: 0.6945 - val_loss: 0.6881
Epoch 2/5
3/3 [==============================] - 107s 33s/step - loss: 0.6867 - val_loss: 0.6804
Epoch 3/5
3/3 [==============================] - 103s 33s/step - loss: 0.6774 - val_loss: 0.6630
Epoch 4/5
3/3 [==============================] - 103s 33s/step - loss: 0.6561 - val_loss: 0.6164
Epoch 5/5
3/3 [==============================] - 103s 33s/step - loss: 0.5975 - val_loss: 0.5131
```

The optimizer works quite well as the losses are reduced considerably in just five epochs.
The resulting prediction from this optimizer is:



.

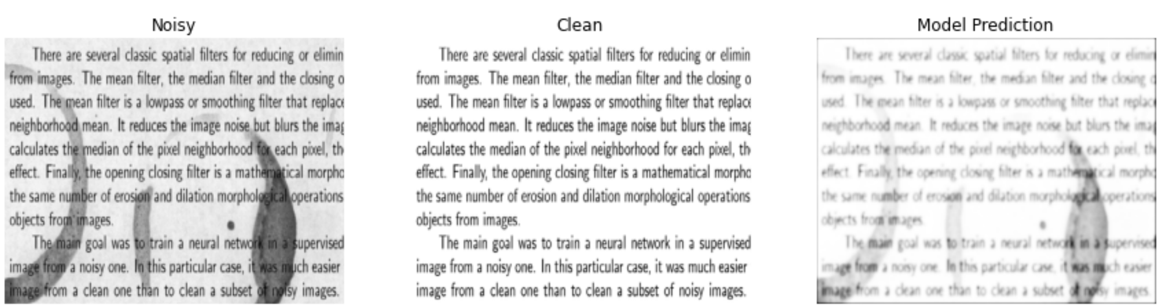
This is a significant improvement from the basic model. A lot of noise has been eliminated. However, the text is a bit blurry.

- RMSprop: The losses on the training and the validation set are:

```
Epoch 1/5
3/3 [==============================] - 104s 33s/step - loss: 0.6749 - val_loss: 0.4550
Epoch 2/5
3/3 [==============================] - 106s 34s/step - loss: 0.3772 - val_loss: 0.3062
Epoch 3/5
3/3 [==============================] - 105s 33s/step - loss: 0.3034 - val_loss: 0.2918
Epoch 4/5
3/3 [==============================] - 105s 33s/step - loss: 0.3008 - val_loss: 0.2886
Epoch 5/5
3/3 [==============================] - 103s 33s/step - loss: 0.2781 - val_loss: 0.2924
```

The losses have improved as compared to the adam optimizer. The first validation loss is even better than adam's loss after five epochs.

The resulting prediction from this optimizer is:



.

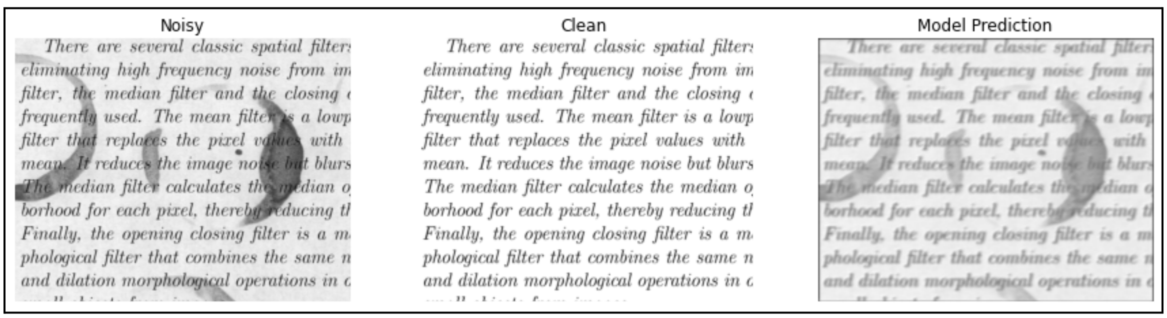
A lot of noise is eliminated as compared to the adam model. The background is also clearer. The text in the model's prediction can be improved with more training.

- SGD: The losses on the training and the validation set are:

```
Epoch 1/5
3/3 [==============================] - 104s 33s/step - loss: 0.6818 - val_loss: 0.6702
Epoch 2/5
3/3 [==============================] - 104s 33s/step - loss: 0.6674 - val_loss: 0.6568
Epoch 3/5
3/3 [==============================] - 107s 33s/step - loss: 0.6540 - val_loss: 0.6428
Epoch 4/5
3/3 [==============================] - 103s 33s/step - loss: 0.6399 - val_loss: 0.6281
Epoch 5/5
3/3 [==============================] - 104s 33s/step - loss: 0.6249 - val_loss: 0.6117
```

This model performs worse than the basic model (loss is 0.61 after five epochs). Hence this is not a suitable choice for this task.

The resulting prediction from this optimizer is:



.

Nothing is visible clearly. A lot of information is lost with this optimizer.

Final Model - RMSprop

From the last section, we see that the deep model with RMSprop as its optimizer performs the best. Hence we will use it as the final model. It will be trained on the concatenated dataset obtained from the training and the validation set.

The model summary looks as follows:

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 420, 540, 32)      320
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 420, 540, 16)      4624
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 420, 540, 8)       1160
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 420, 540, 8)       584
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 420, 540, 16)      1168
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 420, 540, 32)      4640
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 420, 540, 1)       289
=================================================================
Total params: 12,785
Trainable params: 12,785
Non-trainable params: 0
_________________________________________________________________
```

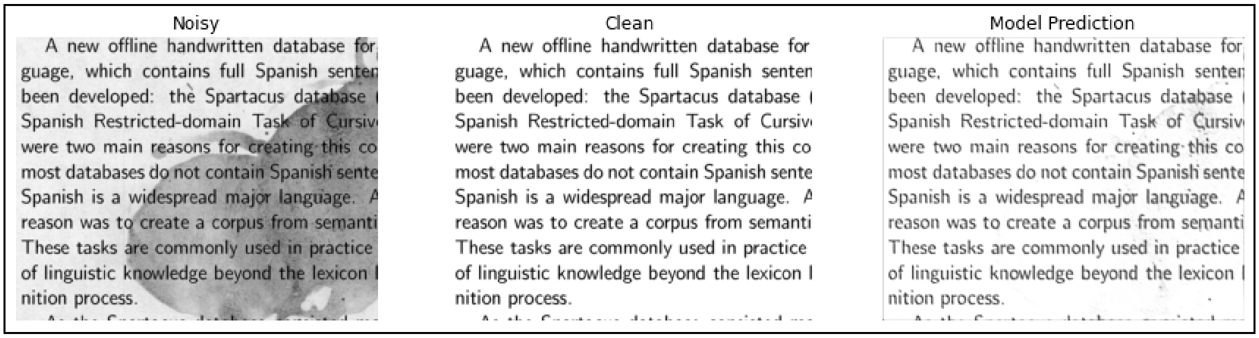
.

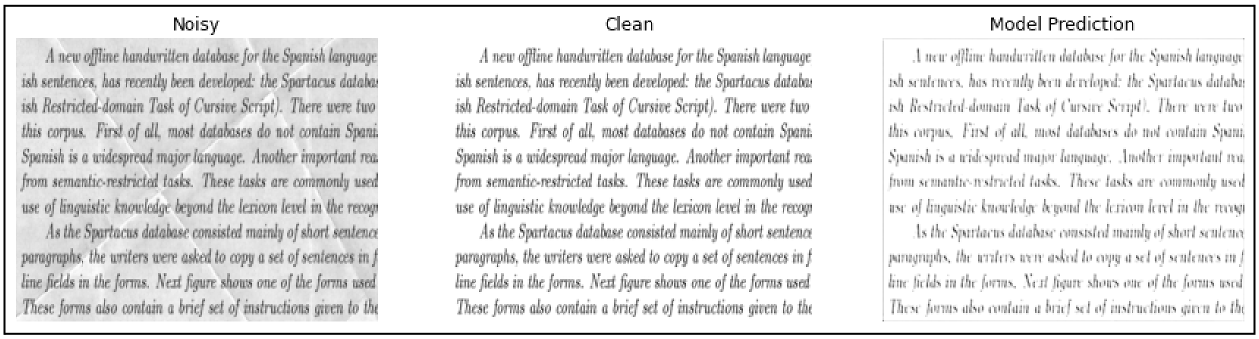
Training is done for ten epochs. After training, the losses look as below:

```
Epoch 6/10
5/5 [==============================] - 188s 38s/step - loss: 0.1950 - val_loss: 0.1843
Epoch 7/10
5/5 [==============================] - 188s 38s/step - loss: 0.1869 - val_loss: 0.1574
Epoch 8/10
5/5 [==============================] - 192s 38s/step - loss: 0.1574 - val_loss: 0.2080
Epoch 9/10
5/5 [==============================] - 189s 38s/step - loss: 0.1638 - val_loss: 0.1354
Epoch 10/10
5/5 [==============================] - 188s 38s/step - loss: 0.1343 - val_loss: 0.1445
```

With the concatenated data, it takes longer to train. The losses have been significantly reduced (more than five-fold) as compared to the basic model.

The resulting prediction looks as below:





The words are visible clearly here. The coffee patches have been largely eliminated while preserving textual information. The background is also white. The performance of this model seems great. However, we can see that the texts in the second case are very light. That is, some information has been lost. We can further improve the performance by using various other techniques or using deeper models.

What more?

The final model with the RMSprop optimizer performs much better than the other models mentioned in this report. To further improve the performance:

- Collecting more training data: physically collecting more data set.
- Data Augmentation: The noises can be added in various combinations. Artificial noise can also be introduced in the training set to increase the training set, allowing the model to learn on a diverse dataset.
- Training a more complex model: A more deep and complex model can be built to improve the performance further.
- Other Optimizers: There are other optimizers that can be explored, like Nadam, Adamax, Ftrl, etc. Custom models can also be built according to the requirements.

References

https://en.wikipedia.org/wiki/Autoencoder
https://archive-beta.ics.uci.edu/ml/datasets/NoisyOffice