## Denoising Dirty Documents

Kartikeya Shukla (ks5173)
Chinmay Wyawahare (cnw282)
Michael Lally (mfl340)

### Problem:

Numerous scientific papers, historical documentaries/artifacts, recipes, books are stored as papers be it handwritten/typewritten. With time, the paper/notes tend to accumulate noise/dirt through fingerprints, weakening of paper fibers, dirt, coffee/tea stains, abrasions, wrinkling, etc. There are several surface cleaning methods used for both preserving and cleaning, but they have certain limits, the major one being: that the original document might get altered during the process. The purpose of this project is to do a comparative study of traditional computer vision techniques vs deep learning networks when denoising dirty documents.

### Data:

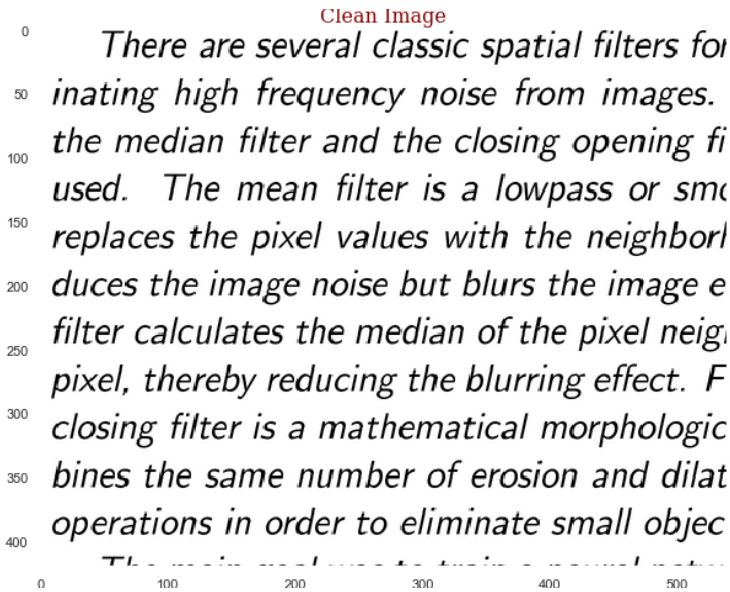
It's a dataset of images containing text that has seen "better days" (i.e. Coffee stains, faded sun spots, dog-eared pages, and extreme wrinkles etc.). It includes images with both synthetic/real noise and also a set of clean images to train our neural net

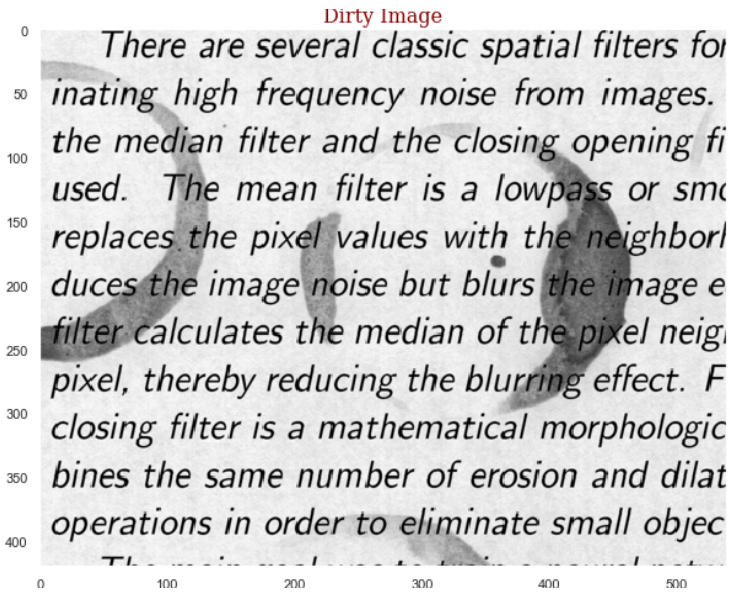
Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science
https://archive.ics.uci.edu/ml/datasets/NoisyOffice

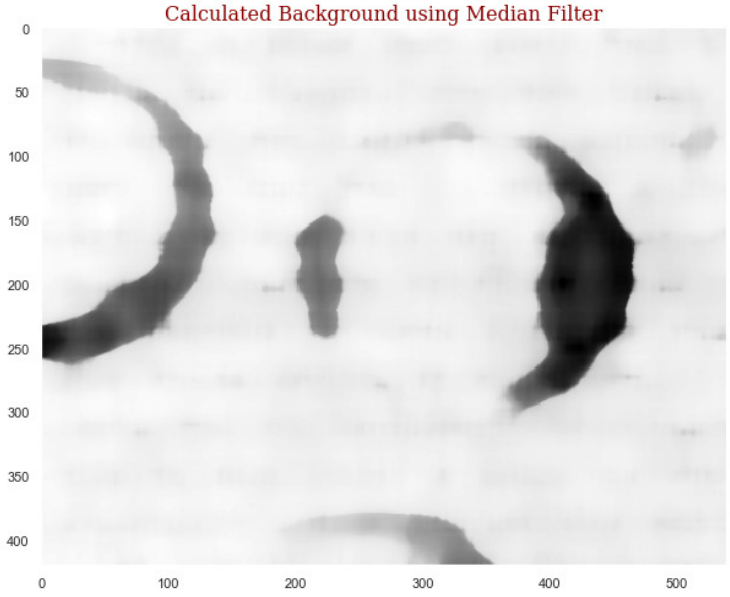
# Analysis Approach

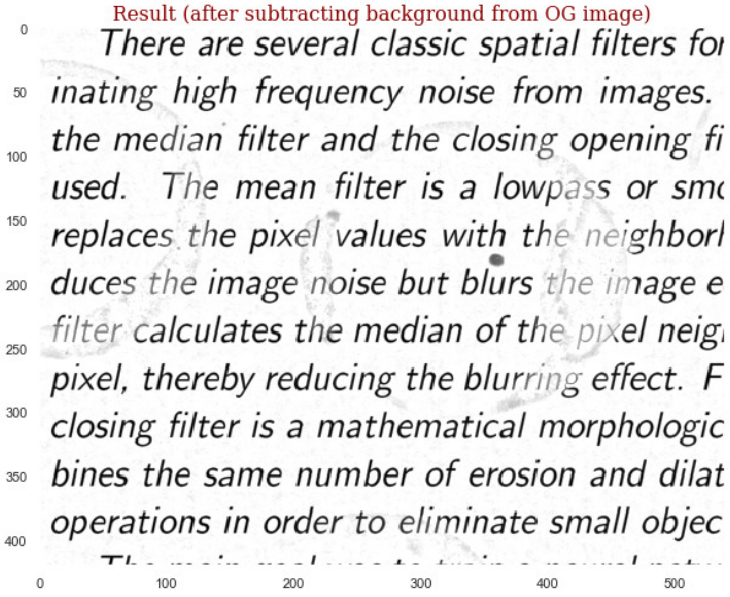
**Median Filtering:**

- Used Median Filter to obtain the "background" of our image, while the text we want to preserve was considered "foreground". The kernel size was 23 x 23. The smoothing effect is to our advantage since the noise takes up more space than the text.
- After applying median filtering, we simply subtract the obtained "background" from the original image. The RMSE, UQI, PSNR were approximately 21, 99 and 21 respectively. The results are shown below.



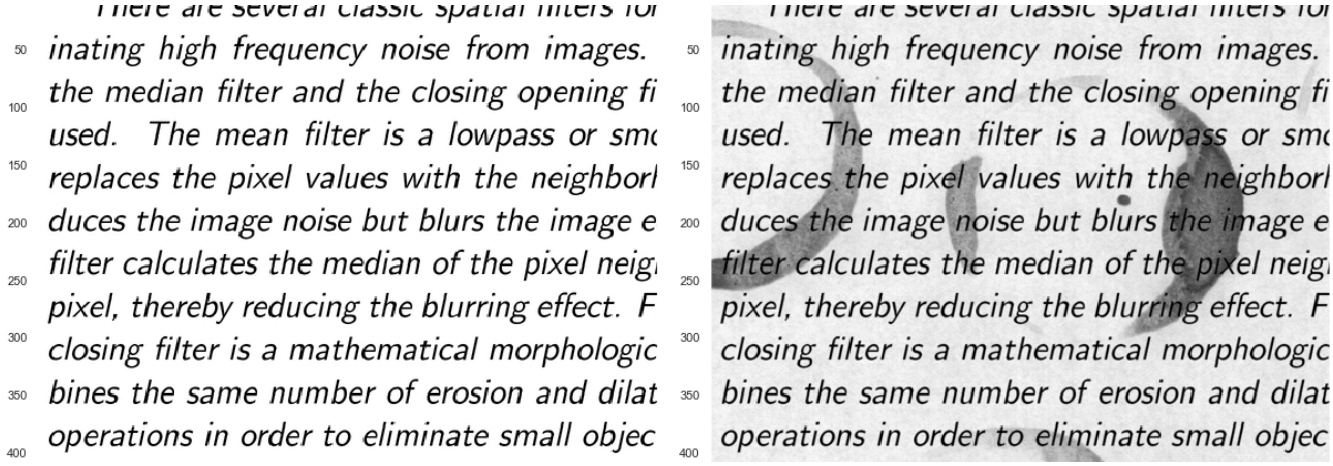



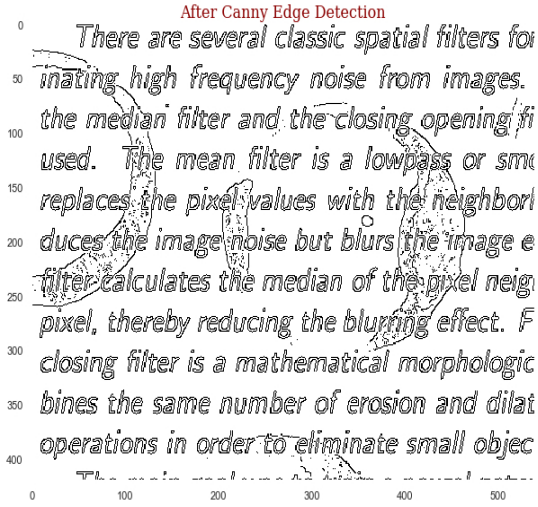



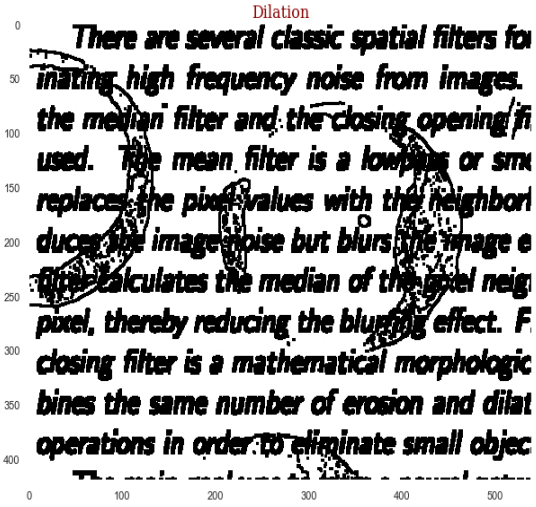

## Edge Detection, Dilation & Erosion

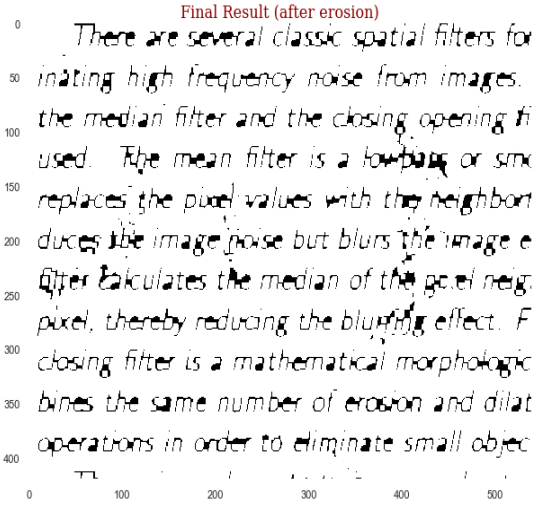
- Edge detection methods identify the points where the image brightness changes sharply, to organize them into **edges**. Used Canny edge detection to extract edges. In the example below, we have already gotten rid of some of the coffee stains.
- Cleaned away the edges of noise. First apply dilation, which makes lines **thicker** by adding pixels to boundaries. Notice this results in “filling in” the text, while edges surrounding stains remain hollow.

- Then, by applying the reverse operation, erosion, one can completely remove thin lines while preserving the thicker lines. The RMSE, UQI & PSNR were 63, 94 and 12 respectively. The results are shown below:



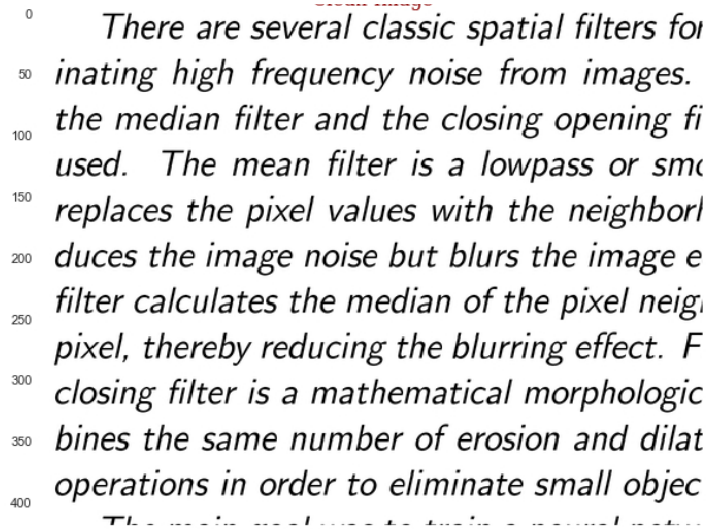


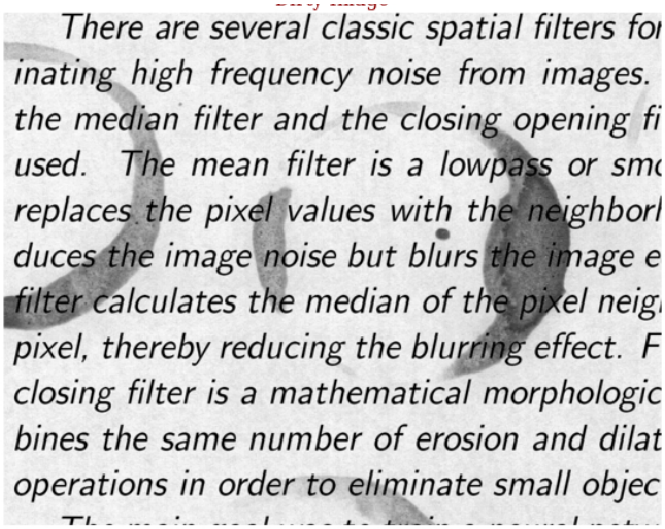



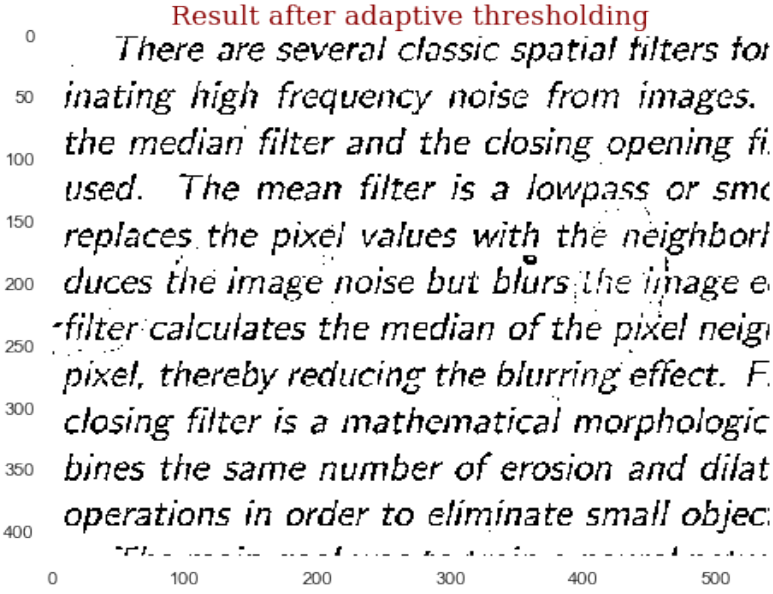
## Adaptive Thresholding

- Another characteristic of the dirty images is that the text tends to be darker than the noise. Within dark noises, the text inside is even darker. Thus the objective is to preserve pixels that are the darkest **locally**.





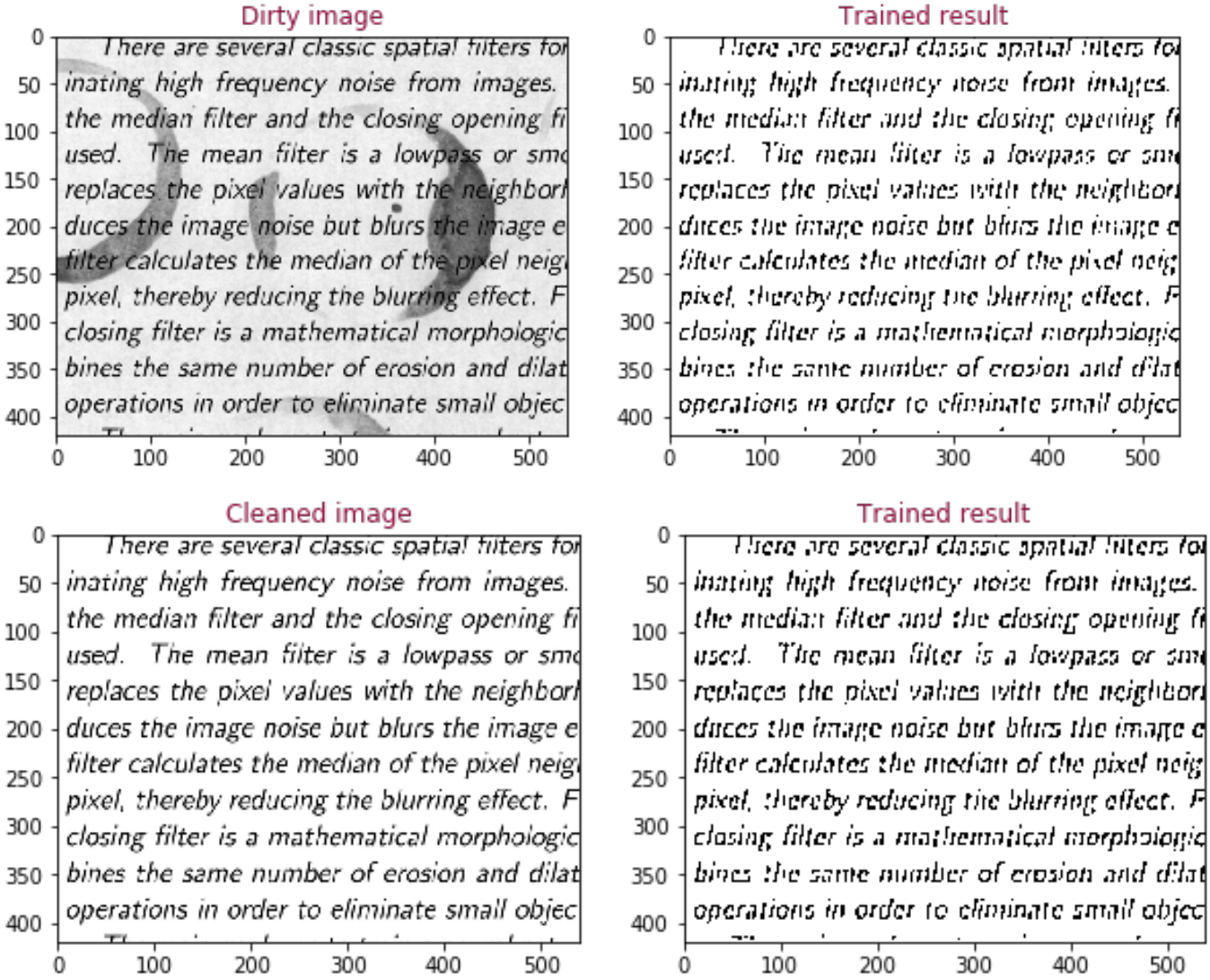
- Thresholding sets all pixels whose intensity is above a threshold to 1 (background), and the remaining pixels to 0 (foreground). During adaptive thresholding, there is no single global threshold: the threshold value is computed for each pixel. To determine the threshold, we use Gaussian Thresholding. The threshold value is the weighted sum of neighboring pixel intensities, where the weights are a gaussian window.
- The RMSE, UQI & PSNR were 30, 98 & 18 respectively. Here are the results:



## Autoencoders:

Autoencoders are neural networks composed of an encoder and a decoder. The encoder compresses the input data into a lower-dimensional representation. The decoder reconstructs the representation to obtain an output that mimics the input as closely as possible. In doing so, the autoencoder learns the most salient features of the input data. Autoencoders are closely related to principal component analysis (PCA). If the activation function used within the autoencoder is linear within each layer, the latent variables present at the bottleneck (the smallest layer in the network, aka. code) directly correspond to the principal components from PCA.

The **RMSE** and **UQI** were approximately 234.26, 6.066 respectively.



**Hyperparameter tuning in autoencoders:**

An autoencoder consists of 3 components: **encoder**, **code** and **decoder**. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code.

There are 4 hyperparameters that we need to set before training an autoencoder:

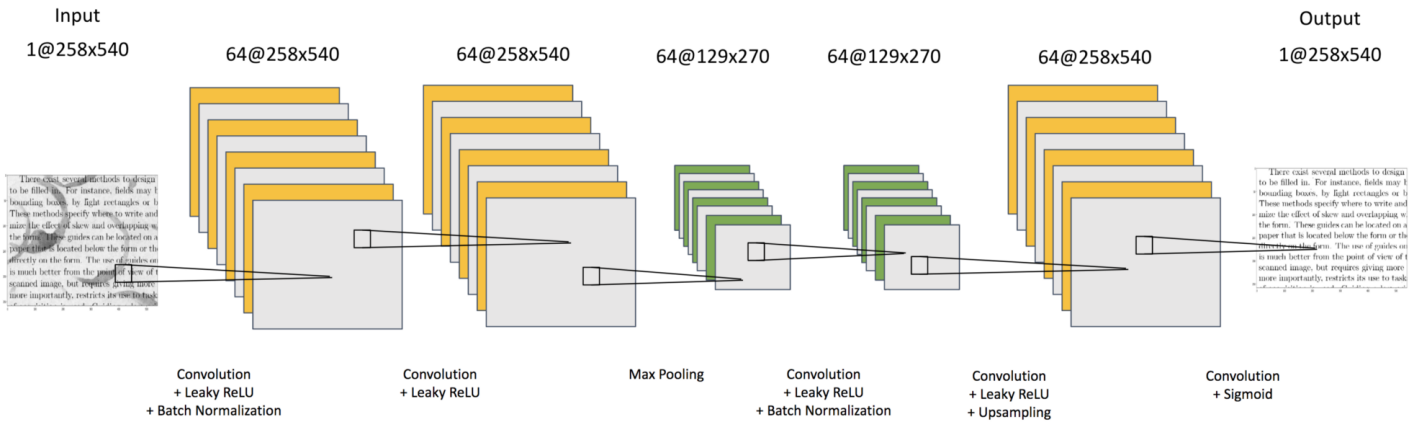
**Code size:** Number of nodes in the middle layer. Smaller size results in more compression.

**Number of layers:** The autoencoder can be as deep as we like. We have 2 layers in both the encoder and decoder, without considering the input and output.

**Number of nodes per layer:** The autoencoder architecture we're working on is called a stacked autoencoder since the layers are stacked one after another. Usually stacked autoencoders look like a "sandwich". The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. Also the decoder is symmetric to the encoder in terms of layer structure. As noted above this is not necessary and we have total control over these parameters.

**Loss function:** We either use mean squared error (mse) or binary cross entropy. If the input values are in the range [0, 1] then we typically use cross entropy, otherwise we use the mean squared error. For more details check out this video.

## Architecture:




The network is composed of **5 convolutional layers** to extract meaningful features from images. To execute a convolution, a convolutional kernel slides over the input. At each

location, matrix multiplication is performed between the kernel and the overlapping region of the input, to produce the feature map passed to the next layer. The values of the kernel matrix are learned during training, using backpropagation with gradient descent. These layers are well-suited to image inputs as they successfully capture spatial dependencies.

In the first four convolutions, we use **64 kernels**. Each kernel has different weights, perform different convolutions on the input layer, and produce a different feature map. Each output of the convolution, therefore, is composed of 64 channels. Thus, in the first convolution, each kernel has dimension **3x3x1**, while in the next ones, the kernels are of dimension **3x3x64** in order to convolve every channel. The last convolution uses a single **3x3x64** kernel to give the single-channel output. During convolutions, we use same padding. We pad with zeros around the input matrix, to preserve the **same image dimensions** after convolution.

To contain non-linearity in our model, the result of the convolution is passed through **Leaky ReLU activation** function. The encoder uses max-pooling for compression. A sliding filter runs over the input image, to construct a smaller image where each pixel is the max of a region represented by the filter in the original image. The decoder uses up-sampling to restore the image to its original dimensions, by simply repeating the rows and columns of the layer input before feeding it to a convolutional layer.

**Batch normalization layers** are included to improve the speed, performance, and stability of the model. We normalize values from the previous layer by subtracting the batch mean and dividing by the batch standard deviation. Batch normalization reduces covariance shift, that is the difference in the distribution of the activations between layers, and allows each layer of the model to learn more independently of other layers.

**Conclusion:**

1. Without any machine learning, using only image processing techniques, three different insights were obtained on how the documents can be cleaned. However, the three methods are “imperfect” on their own.
2. The autoencoder seems to be best at removing most “noise” (i.e. stains). As one can observe, there is no hint of any *background* stain left on the resultant image. It’s as if it never got stained.

**Limitations:**

1. Cannot use the same threshold method/kernel size for each image. Depending on the noise we'll have to change our kernel size or the threshold value/method. For these images — since the noise & features are "quite" similar we used the same kernel size/ threshold method for all images.
2. For the final product, a user might upload a dirty image, but we might not have a clean version of it. Thus, rendering measures such as RMSE, UQI & PSNR useless. At most we can measure the structural similarity using SSIM of a noisy and denoised image, if we don't have a clean reference.
3. The autoencoder seems to "overclean" the document. Although it's the best at removing stains, it's also "whitening" text in the process. One reason for the whitening could be — Dataset consisted of a few noisy and cleaned images. We had to add synthetic noise such as white noise and apply linear transformations to generate more images. Since these newly generated images aren't accurate enough for their corresponding cleaned versions of images, we could find discrepancies in the output of autoencoder
4. Another reason for "whitening" of images for autoencoders could be the number of epochs the autoencoder was trained to compute the resultant images. If we provide quality dataset as input for the autoencoder, we can observe a significant improvement in the corresponding cleaned versions of noisy images

**Future Work**:

1. Developing a webapp dashboard using dash/plotly, flask and AWS for the above analysis, which cleans an image when the user uploads a document.
2. Use **linear regression** — Instead of modelling the entire image at once, one will predict the cleaned-up intensity for each pixel within the image and construct a cleaned image by combining together a set of predicted pixel intensities. (is it "really" necessary for our model to account for neighboring intensities?)
3. Each of the algorithms given in the analysis provide a different insight into how the document can be cleaned, with its strengths and weaknesses. Can each output be **combined** to produce an output that outperforms all previous ones?
   **Maybe** - Our "stacked" neural network will again be composed of 6 convolutional layers with **5 input channels**:
   the original image the output from median filtering, from edge detection, from adaptive thresholding, and from the CNN autoencoder. The stacked model thus uses all five information to produce the final output.

4. The autoencoder was trained for 10 epochs for the output presented in the report. We can try to hyperparameter tune the autoencoder and adjust the number of epochs to increase the learning for the model.
5. Along with Autoencoder, we would be adding techniques like **CycleGAN** and compare the results between the autoencoder output and CycleGAN using RMSE, UQI and PSNR metrics.

*ps— for the “foreseeable” future: MVP will be the primary objective.