CS 6643 - Computer Vision, Spring 2020
Denoising Noisy Documents

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

Problem

Numerous scientific papers, historical documentaries/artefacts, 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/Expected Results

1. Use **median filter** to get a "background" of the image, with the text being "foreground" (due to the fact that the noise takes more space than the text in large localities). Next subtract this "background" from the original image.
2. Apply **canny edge detection** to extract edges. Perform **dilation** (i.e. make text/lines thicker, and noise/lines thinner) then **erosion** while preserving thicker lines and removing thinner ones (i.e. noisy edges)
3. Use **adaptive thresholding**. (works really well since often text is darker than noise). Thus preserve pixels that are darkest **"locally"** and threshold rest to 0 (i.e. foreground)
4. **CNN Autoencoder:** The network is composed of **5 convolutional layers** to extract meaningful features from images.

- During convolutions, **same padding** mode will be used**.** We pad with zeros around the input matrix, to preserve the same image dimensions after convolution.
- 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.
- Perform **batch-normalization** as required. For the output, we use **sigmoid activation** to predict pixel intensities between 0 and 1.

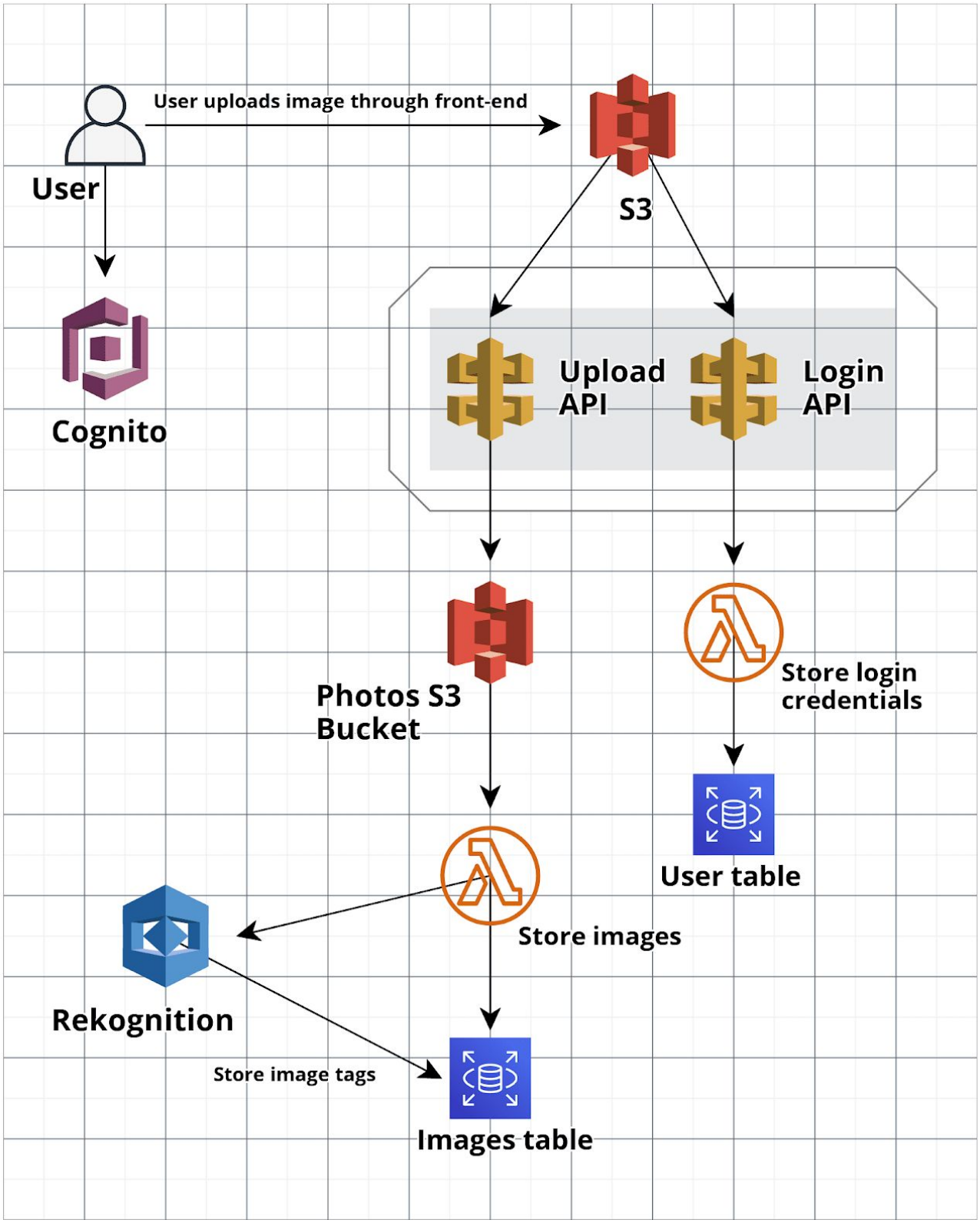
5. Compare results from {1, 2, 3, 4} using the following metrics **RMSE, PSNR, SSIM, UQI**

Architecture

The entire project will be hosted on **AWS** as a web application. AWS Rekognition service is used for labelling and tagging the images uploaded by the users. Once the task of labelling and tagging is complete, the images will go through various algorithms and image processing techniques mentioned in the **'Analysis Approach'** section.

The output from the web application will be surface graphs depicting the changes the image has gone through when it's passed through the algorithmic engine. Along with the graphs, the web application will output the cleaned version of images using the underlying algorithms so the user recovers the original version of the noisy images to improve readability and accessibility of the images.

- **S3:** To store the images
- **EC2:** To host the web pages
- **RDS:** User and image database
- **Rekognition:** Tag the image with labels
- **Lambda:** To integrate the services
- **Analysis:** Using openCV and skimage


User uploads image through front-end
User
S3
Cognito
Upload API
Login API
Photos S3 Bucket
Store login credentials
User table
Store images
Rekognition
Store image tags
Images table