Handwriting is a skill of man that is done to express thoughts, ideas, and language. Doctors are well-known for having illegible cursive handwriting and it is a generally accepted matter. It is proven that even pharmacists who must administer the distribution of medicines prescribed for patients are also having difficulties reading doctors' handwriting due to clinical notes that are illegibly written, and the content is unclear which resulted in many cases of medical errors. In this project, we train a Handwriting Recognition system with labeled data. Specifically, we train a deep convolution recurrent neural network (CRNN) system on manually labeled text-line data from a specific doctor's prescription datasets and propose an incremental training procedure that covers the rest of the data as and when available to perform at the highest level of accuracy for the model to be able to adapt to individual doctor's handwriting.
This project consists of multiple segments as mentioned below:
1. Labeling data using image annotation tools
2. Raw image into line segmented image
3. Splitting the images into different frames
4. Time Distributed Convolution Recurrent Neural Network (CRNN)
5. Implementation of Connectionist Temporal Categorical (CTC) loss function
6. Nearest word prediction from the pre-created dictionary

Section 1:

Labeling of data using image annotation tools

This is the first and most time-consuming step of the project. VGG image annotator by Oxford is best for our purpose. This tool allows us to select areas in the prescription by drawing bounding boxes for the words, and by entering an annotation corresponding to the area selected. Once the annotation is done, we will be able to download the annotation in CSV format for further processing in Python. The downloaded file presents information like the region of annotation in x and y coordinates, the file name, and the region attributes, which is the actual annotation of the word itself.

Section 2:

Raw image converted into line segmented image

This is done with the help of the OpenCV library in python. The raw image is read in its original resolution by the process. It will then be converted into a greyscale value. Once this is done, then the contours in the image are identified. The contours help identify the boundaries for the pixels having the same color or intensity. After the bounding boxes are drawn, we then crop the image to take out the individual line segmented image.
In the process of getting the proper line segmented data, we ignore all the images which have height bounding boxes below a certain threshold limit.
Also, the order of the image should be retained. For this, we ensure that we read the images from contours based on the starting pixel value of the image. Thereby the images will be read from the top left corner to the bottom right corner.

Section 3:

Splitting the images into different frames

Keras wrapper TimeDistributed to be used to feed the CNN layers. This time distributed wrapper takes different frames of the input and processes it frames by frame. Then frames of 64 x 64 pixels images are obtained from the input image with a stride of 4 for each subsequent frame. The sliding window is applied in the direction of the writing which is from left to right. Keras TimeDistributed wrapper is used for passing the frames of input to the CNN layers. The time-distributed wrapper helps preserve the temporal sequence of the frames of images that we get from the input.

Section 4:

Time Distributed Convolution Recurrent Neural Network (CRNN)

13 convolution layers followed by three bidirectional LSTMs consisting of 256 units in each of the directions are to be used. Max pooling is applied after some convolution layers, a total of 5 max-pooling layers are applied. To introduce non-linearity in the convolution layers, the activation function ReLU is used. There is a dense layer between the output of the CNN and the input of the LSTMs, which is very effective in reducing the number of parameters that are used for our training.
We set the total number of output classes that are applicable for our problem to 91 (A-Z, a-z, 0–9, all standard special characters on an English keyboard, plus one additional class for unknown).

Section 5:

Implementation of Connectionist Temporal Categorical (CTC) loss function

The objective function which is used to minimize the loss is the Connectionist Temporal Categorical (CTC) loss function. While other loss functions optimized a single objective function, the CTC loss is specially designed to optimize both the length of the predicted sequence and the classes of the predicted sequence, as the input image varies in nature. The convolution filters and the LSTM weights are jointly learned within the back-propagation procedure.

Section 6:

Nearest word prediction from the pre-created dictionary

The metrics that are used to track the performance of the network are Levenshtein distance, which is a commonly used metric to measure the string metrics by measuring the difference between the observed sequence and the predicted sequence.
For this, we built a vocabulary of all the words in the input. The CTC loss function does not give the entire word correctly as there is an overlap between the frames fed using the TimeDistributed wrapper. Since a single alphabet could be in multiple frames, the output will contain a repetition of words. So we

have to use edit distance as the metric to find out what is the nearest word in our entire corpus which matches closest to the word given as output.

Section 7:

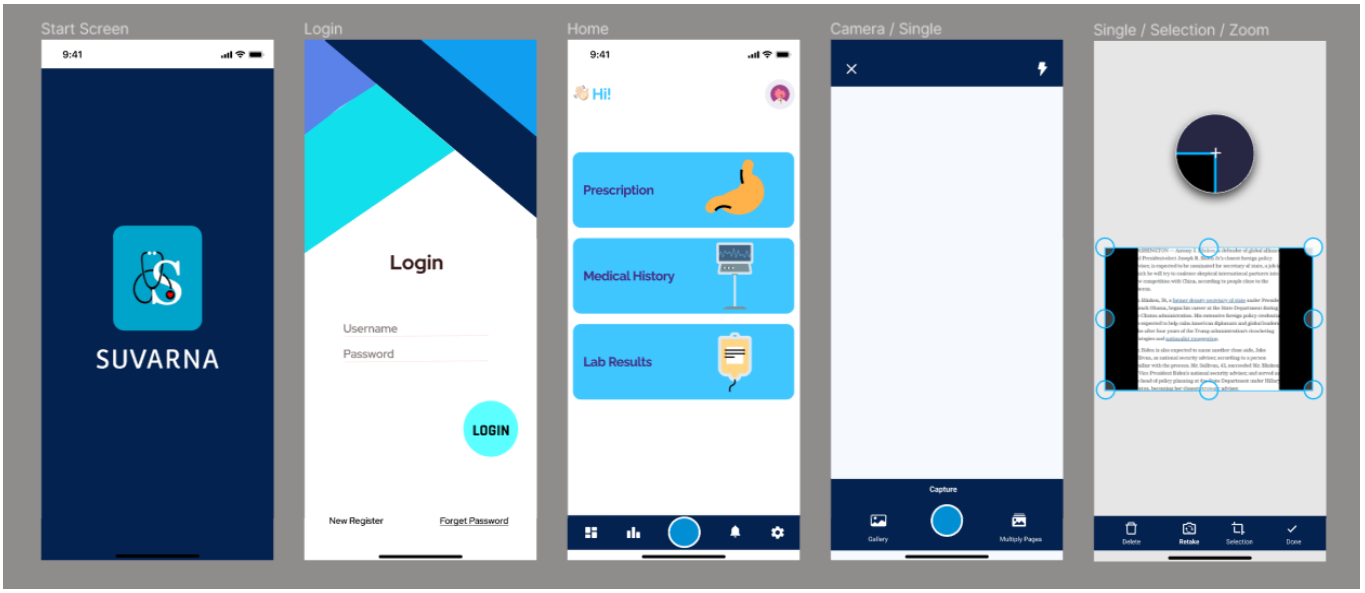
Model Deployment Using Flask

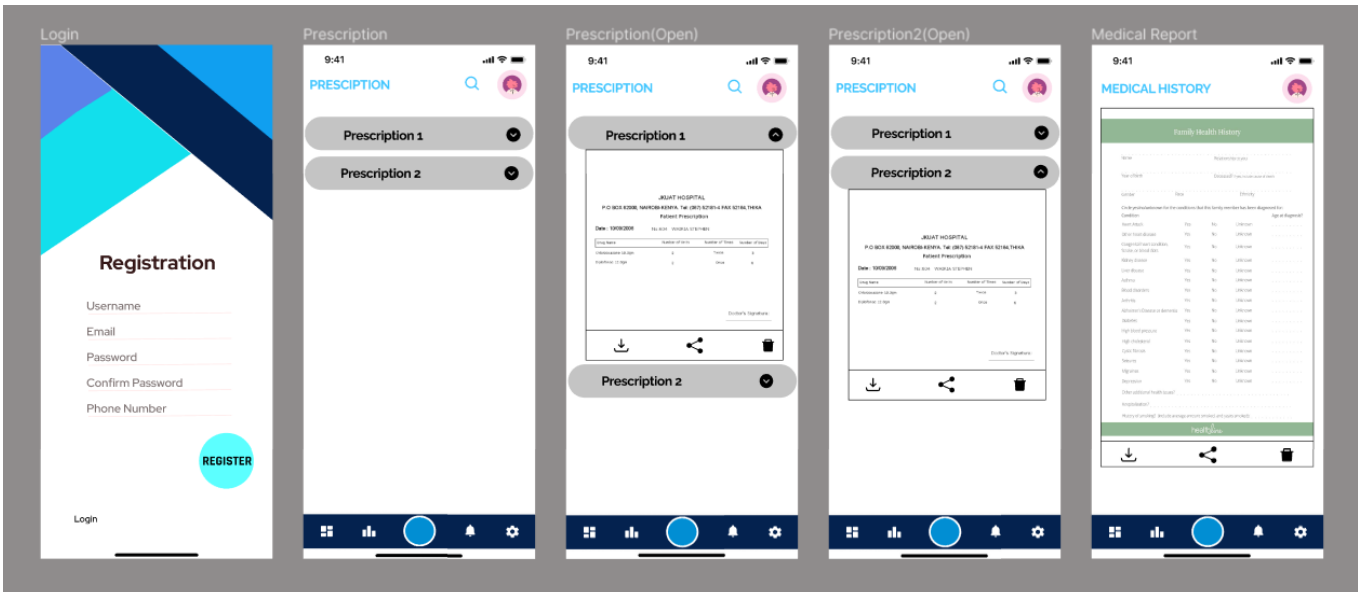
We will use Flask (Python framework) as the back end for our REST API, Flutter for mobile app, and Keras for image classification. We will also use MongoDB as our database to store data about the images and classify images using the Keras ResNet50 model, using a pre-trained model seems to be useful for this purpose. We can use a custom model if needed by saving it save_model() and load_model() methods available in Keras.

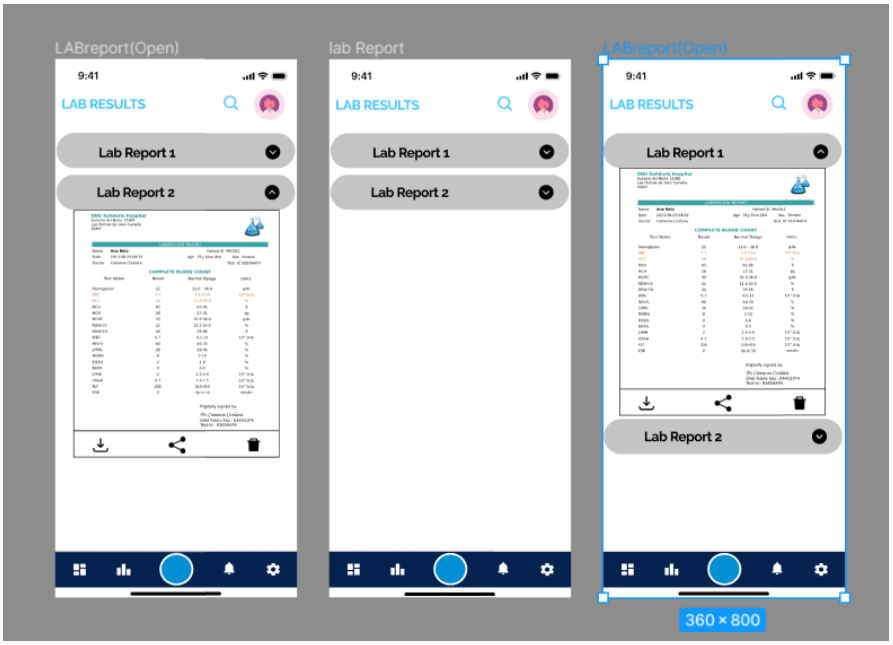
Section 8:

Flutter Application:

The application will use the REST API to fetch an image and display it, we can also search an image by the content. Our app will look like this:




Login
Registration
Username
Email
Password
Confirm Password
Phone Number
REGISTER
Login
Prescription
9:41
PRESCIPTION
Prescription 1
Prescription 2
Prescription(Open)
9:41
PRESCIPTION
Prescription 1
Prescription 2
Prescription2(Open)
9:41
PRESCIPTION
Prescription 1
Prescription 2
Medical Report
9:41
MEDICAL HISTORY


LABreport(Open)
9:41
LAB RESULTS
Lab Report 1
Lab Report 2
lab Report
9:41
LAB RESULTS
Lab Report 1
Lab Report 2
LABreport(Open)
9:41
LAB RESULTS
Lab Report 1
Lab Report 2
360 × 800