HOW TO RUN "AUTOMATIC PILL COUNTER" ON YOUR DEVICE?

Prerequisites

1. Install python(preferably 3.8 or 3.9), required packages, and cocoa pods.

Code Explanation ***(*** *the final code that was implemented for pill counter i.e with watershed algorithm)*

Initially, we load the image from google drive(So in order to test the code we are providing images in the zip folder) and check if the size of the image is less than 200X200. If the image size is less than that it calls the superresolution() function and returns the enlarged image. We have used pre-trained EDSR opencv model for super resolution . In order to load the model in super resolution function we are providing the EDSR model(EDSR_x4.pb) in the Zip folder, so add path which leads to EDSR model.
The next step is applying pyramid mean shift filtering. Mean shift filtering is done to smoothen the image so that it makes the thresholding accurate.

It is important to know the background color of the image before applying the threshold. The threshold we apply is different according to the background color. To check the background color, we have considered 2 corner edges of the image with minimum dimensions. We have used the mostCommon() function to check the most occured RGB values in that area. The function returns the most occured RGB values the borders.The most common pixel returned by the mostCommon() function is used to decide between threshold methods. If any of the values in the pixel array is less than 128 then Thresh_Binary_INV is used else Thresh_Binary is used along with the Thresh_OTSU.

First the image is converted to gray from bgr using cv2.cvtcolor(). If the pills we need to count are in lighter colors and the background is dark then we apply cv2.THRESH_BINARY_INV or cv2.THRESH_OTSU threshold method so that it highlights the area where the pills are in white and rest will be covered in black color. Similarly, if the pills are in a darker color and the background is lighter then we apply cv2.THRESH_BINARY or cv2.THRESH_OTSU threshold method. As we have a threshold image, We can apply the watershed algorithm now.
The first step in applying the watershed algorithm for segmentation is to compute the Euclidean Distance Transform (EDT) via the distance_transform_edt function. As the name suggests, this function computes the Euclidean distance to the closest zero (i.e., background pixel) for each of the foreground pixels. We take D, our distance map, and find peaks (i.e., local maxima) in the map. We'll ensure that is at least a 20-pixel distance between each peak. Taking the output from the peak_local_max function we applied connected-component analysis using 8-connectivity. Connected Components in an image have a set of pixels with the same value connected through

Eight Pixel Connectivity. The labels of connected components are used to detect the connected regions in the image.
The output of this function gives us our markers which we then feed into the watershed function. Since the watershed algorithm assumes our markers represent local minima (i.e., valleys) in our distance map, we take the negative value of D. The watershed function returns a matrix of labels, a NumPy array with the same width and height as our input image. Each pixel value has a unique label value. Pixels that have the same label value belong to the same object.
The last step is to simply loop over the unique label values and extract each of the unique objects. If the label is zero, then we are examining the "background component", so we simply ignore it. Otherwise, we allocated memory for our mask and set the pixels belonging to the current label to 255 (white).We detected contours in the mask and extracted the largest one — this contour will represent the outline/boundary of a given object in the image. Finally, given the contour of the object, all we need to do is draw the enclosing circle boundary surrounding the object.

Code Implementation

We have initially implemented the code in google colab before implementing on the IOS app. So the code can be tested even on google colab or jupyter notebooks .
The below link directs to the direct implementation of code in google colab
https://colab.research.google.com/drive/1cud6j9iABm1XJTc3C9VGYFhsRseWASl2?usp=sharing

Install Xcode

1. Install Xcode 12.1 from the app store or the website.
2. Create a new project in Xcode -
   1. Choose "App for IOS" from the various templates provided.
   2. Select "Storyboard" as the interface for UI designing and "Swift" as the programming language.
3. Select the target for your application.

Using Storyboard for UI

1. Create any UI components depending on the requirements.
2. Make sure to add the labels and their respective actions into the swift code.

This application needs access to the camera and photo library on your device. So some steps need to be followed in Xcode to grant permissions to it.

1. Under the project folder, click on “Info.plist” on the left pane.
2. Click on the add button and then select “Privacy - Camera Usage Description” and “Privacy - Photo Library Usage Description”. This will allow your app to access the Camera and Photo Library.

1. Snapshot of Info.plist

Now open “APC.xcworkspace” from the attached zip folder. Before building this swift code, we need to create Azure BLOB containers to store the photos.

How to create Azure BLOB containers?

1. In portal.azure.com, create an Azure account.
2. On the home page, create a storage account either under the existing resource groups or a new resource group.
3. Click on review+create.

Next, we need Azure Storage Client Library to build iOS applications that use Microsoft Azure Storage.

How to import Azure Storage Client Library?

This can be done by installing Cocoapod or by building a framework.
I have used Cocoapod for installation. Steps for installation are -

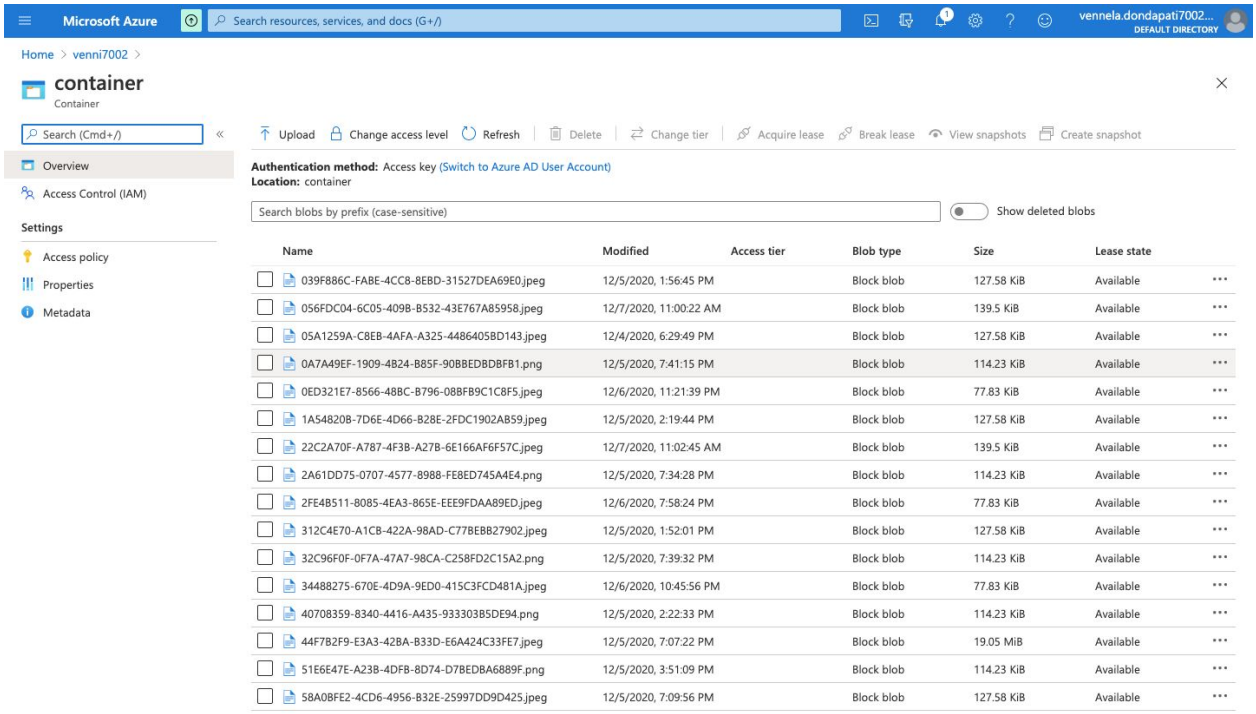
1. In the terminal, run the command `$ sudo gem install cocoapods`
2. Create a new pod file and overwrite the contents with the following code:

platform: ios, ‘8.0’

target ‘TargetName’ do
  pod ‘AZSClient’
end

3. Install the dependencies using `$ pod install.`
4. Make sure to always open the Xcode workspace instead of the project file when building the project.
   Use `$ open App.xcworkspace` in the terminal to open a workspace.

This will import the library into the code and the Azure Storage Client library will be ready to use.
Now you can upload images into the container using the app. You can see the uploaded images in the Azure portal.



2. Snapshot of uploaded images in Azure BLOB storage.

<u>Why Azure Function App?</u>

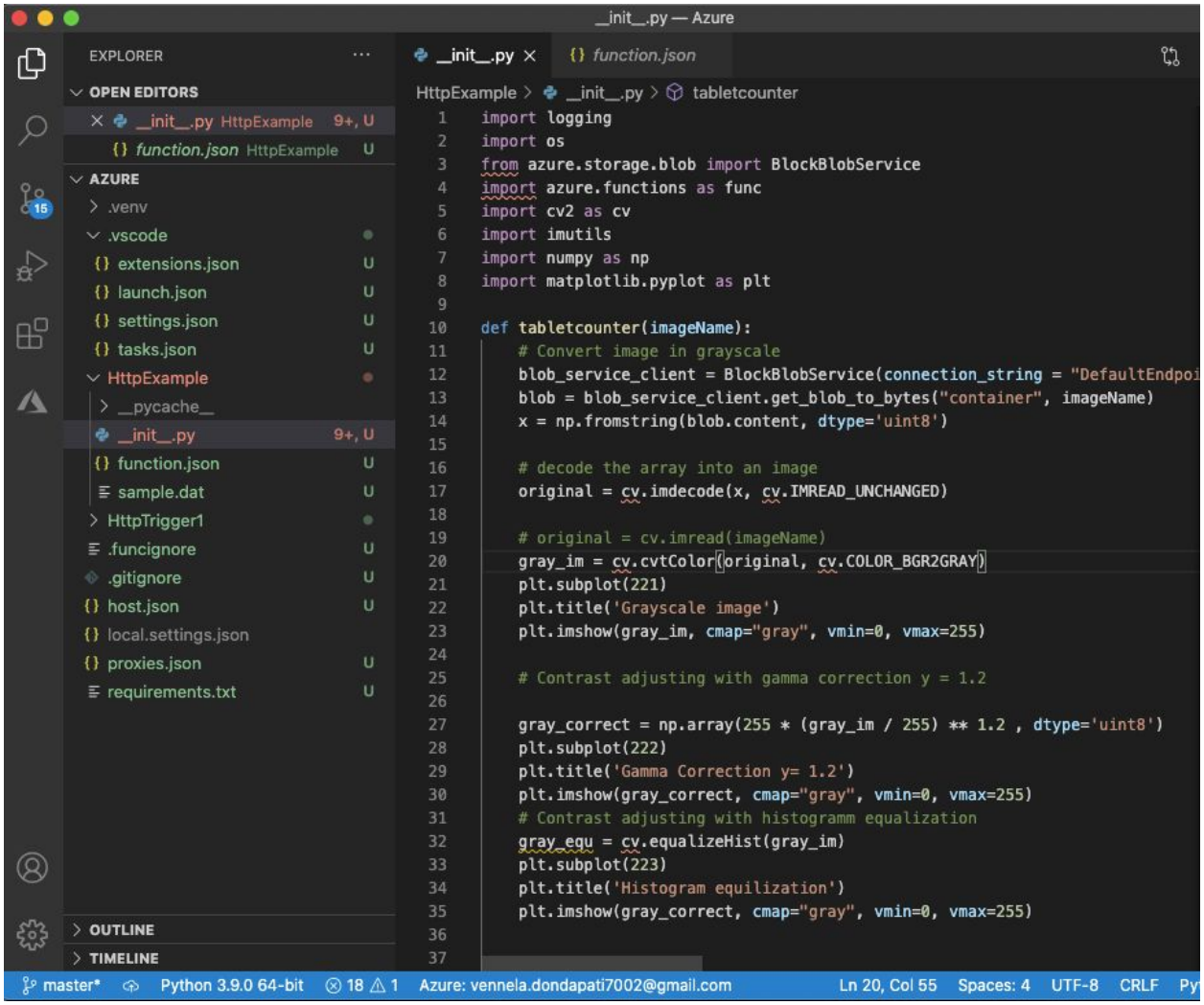
As we won’t be able to run the .py script locally in swift, we can call it a URL and have the server do the work for you.
So I used the Azure function App to run a python file.

Configuration of Azure function app environment -

1. Install Visual Studio Code and create a new project.
2. I have chosen Python as the language for the function and selected a template to create the first function.
3. Once the .py file is created, paste the python code and deploy it to the function app.
4. After the deployment, you can view the output in the terminal.
5. Now to run the function in azure, expand the functions and then choose the copy function URL.
6. Paste the URL in the browser to see the output.

Now open the "APC.py" file and run the code to check the output in the terminal. Once the code is successfully executed, it should be deployed. These are the following steps for deployment :

1. Click on Azure Function and select Local project.
2. Click on the dropdown and select functions, under the functions select the function you created.
3. Now deploy your function to function app. Once the code gets deployed you can check the status of deployment in the output console.



3. Snapshot of python code in visual studio.

After the deployment, build the Xcode project and you can see the output(number of pills) displayed on the view controller.

These are the steps to be followed for using an Automatic pill counter application on your device.

For further queries, feel free to contact us.