At the end of this tutorial, we will be able to describe the image, display the description on the image, detect any text written on the image, recognize number of individual present in the image, recognize the celebrity, recognize the people who are not celebrity (basically providing our own database to refer for matching the faces) and also draw a rectangle around the identified face and lastly use AWS Polly to speak out every above detail.

Yup, the aim looks big, but AWS Polly will make it look great.

In this module we will see how to work with AWS Polly

Open PyCharm -> New Project -> Pure Python -> Name it: SayIt -> Create.

Now, let’s install the packages:

Press Ctrl+Alt+S -> Project: SayIt -> Project Interpreter -> Click on the Setting Icon -> Create VirtualEnv -> Location: Locate your SayIt Project and save the location inside the Project; Name: Chalice -> Check the Inherit global site-packages -> OK.

Then PyCharm will create the appropriate environment for us. After that, install the packages by Clikcing on the “+” icon. Install basic packages like: boto3, awscli, requests, and chalice. What does Chalice do? It sets up the environment and permission required for the program to work.

Now, go to Terminal (Located at the bottom of PyCharm). Write following commands:

* cd..
* chalice new-project SayItApp

Open your SayIt project in File Explorer. Press back to go to the previous folder. There you will see your SayItApp folder. Open it, it will contain a chalice folder and some files. Copy all the files to SayIt project location. And do the following changes:

* chalice -> config.json - “app-name”:”SayIt”
* app.py -> app\_name:’SayIt’

Go to Terminal -> Type “chalice local” -> Click the check boxes that shows up -> Allow access.

Now we a localhost running on localhost: 8000. Copy that and paste it in your browser. And it will display the {‘hello’:’world’}.

Copy the following files in your folder:

* <a href=”app.py.html” target=”\_blank”> app.py </a>
* <a href=”get\_voices.py.html” target=”\_blank”> get\_voices.py </a>
* <a href=”index\_html.html” target=”\_blank”> index.html </a>
* <a href=”style.css.html” target=”\_blank”> style.css </a>
* <a href=”apptext.py.html” > target=”\_blank”> apptext.py</a>

Open Python Console (Located at the bottom of PyCharm), and write the following code:

* from app import \*
* from apptext import \*
* text\_to\_speech(‘Hello’, ’Amy’)

“text\_to\_speech” is a function in our apptext.py file. This function uses AWS Polly and spells out the given text using the given voice. There are more than 14 voices present in AWS Polly. We are opting ‘Amy’ for our project. If you want to know what other voices AWS Polly provides, we can Google “AWS polly voices” and open the “Available Voices – Amazon Polly – AWS Documentation” link. We can also save these audio file on our S3 Bucket. To explore more about S3 option, watch the recommended YouTube videos.

Write a simple python calling client and before the return in the method call the function text\_to\_speech() and pass it the required values. Don’t forget to import the apptext.

Now explore more voices provided by AWS Polly and try different text.

# In this module we will see how to work with AWS Rekognition.

# We will be using Boto3 for working with AWS Rekognition. Boto3 provides variety of functions that makes interacting with AWS simple. We will be covering the following functions:

* detect\_labels
* recognize\_celebrities
* detect\_faces
* compare\_faces
* detect\_text
* create\_collection
* index\_faces
* search\_faces\_by\_image

Try exploring other functions provided by Boto3. One can find all the functions provided by Boto3 on their documentation page, the link of the page is given on the home page.

We will discuss the first 4 function briefly, as they have already been explained to us by the professor, in this module. Let’s get started with detect\_labels. This function outputs all the labels that an image contains. In other words, it will describe the image in words. For example, there is an image of nature, detect\_labels will detect all the components in the image like clouds, river, nature, stone, sand, trees, forest. This methods accepts two parameters, image bytes and confidence level (optional). Confidence level is the AWS Rekognition percentage of surety that the component is what it has described as. We can also provide an image from S3 Bucket.

To convert image into bytes we will use the image\_helper.py program given by Professor. The program just identify whether the image is located on local machine or on internet (url). Then accordingly use content function or read function to convert the image into byte. “detect\_labels” function returns a Label List but we are only interested in ‘Name’ filed of the list. We can perform a for loop on our response[‘Label’] and inside the loop we can do, item[‘Name’] (item is the name of the variable declared in for loop and used to traverse the Label). Now we have all the names. We can use the “join” function to join the list elements. In our case it will look like:

Str=', '.join(name\_label)

Pass this “STR” string in the text\_to\_speech(Str,’Amy’).

Next is “recognize\_celebrities”, this function detects the faces from the image and match them with the AWS Rekognition Celebs database. It has one compulsory parameter, i.e., image bytes. This function returns List of response[‘CelebrityFaces’]. Same as above, we only require the ‘Name’:

for face in rekresp['CelebrityFaces']:  
 result.append(face['Name'])

Again perform the join and pass it to the text\_to\_speech() function:

Str=', '.join(name\_celeb)

text\_to\_speech(Str,’Amy’)

Next we have detect\_faces() function. It describes how a face looks and also outputs it position. We will use AWS Polly to describe image via speech and outputs for drawing the box around the face. Bound Box attribute of the result contains the positions. The emotion attribute describes the state of the person like calm, confuse. detect\_faces() describe every details of the face: age, gender, sunglasses, Beard. But we modify the output to our requirements and describe number of faces, gender, age, emotion. The code will look like:

client = boto3.client('rekognition')

rekresp = client.detect\_faces(Image={'Bytes': imgbytes},  
 Attributes=['ALL'])

numfaces = len(rekresp['FaceDetails'])

if numfaces == 1:  
 found = 'Found ' + (str)(numfaces) + ' face : '  
else:  
 found = 'Found ' + (str)(numfaces) + ' faces : '  
  
for facedeets in rekresp['FaceDetails']:

fmtstr = '{gender} age {lowage}-{highage},'

if facedeets['Mustache']['Value'] and facedeets['Beard']['Value']:  
 fmtstr += ' with beard and mustache,'  
elif facedeets['Mustache']['Value']:  
 fmtstr += ' with mustache,'  
elif facedeets['Beard']['Value']:  
 fmtstr += ' with beard,'

if facedeets['Sunglasses']['Value']:  
 fmtstr += ' wearing sunglasses,'  
elif facedeets['Eyeglasses']['Value']:  
 fmtstr += ' wearing glasses,'  
  
fmtstr += ' looks {emotion}'  
  
result1=(  
 fmtstr.format(  
 gender=facedeets['Gender']['Value'],  
 lowage=facedeets['AgeRange']['Low'],  
 highage=facedeets['AgeRange']['High'],  
 emotion=facedeets['Emotions'][0]['Type'].lower()  
 )  
)

result=found+result1+'. '

Next is “compare\_faces”.It takes three parameters: Similarity Threshold, source image bytes, and target image bytes. The function will look like:

client.compare\_faces(SimilarityThreshold=70,  
 SourceImage={**'Bytes'**: imgsourcebytes},  
 TargetImage={**'Bytes'**: imgtargetbytes})

From this we might be getting an idea that for comparing faces that are stored locally, we would be using the compare\_faces function and output the result. The answer is maybe or maybe not.

Logically, it is possible. Like take every stored face and compare it to every face in the provided image. It seems lengthy. Now let’s see other details, how will you provide the stored image? How will you store the image? We cannot just save the image file locally and provide the path of the folder. There is no function that takes input as folder or can traverse a folder.

Well the answer to the above question is No. Instead we have concept of Collection. And we will discuss this in the last module.

In this module we will learn how to detect text from an image?

The detect\_text() function, as the name suggest detects text from an image. It can also detect hidden text that are not visible to human eye. This function accepts a parameter, the image byte.

response = client.detect\_text(Image={**'Bytes'**: imgbytes})

pprint(response)

As you might be able to observe in the Boto3 documentation that there is a function detect\_text() to perform this task, but when you run this function, you might be getting an error saying Rekonigtion has no method as detect\_text().

The reason why we are getting this error, is because this function was introduced in the Boto3’s newer version, 1.4.8, released on 22nd November, 2017 and we are working on 1.4.7.

To update your version of boto3, in PyCharm hit Ctrl-Alt-S to bring up the Project settings.  Go to the Project and click on Project Interpreter.  There you will see a list of all the modules/packages.  Find boto3 in the list, and click on it.  Then over to the right you will see a plus, a minus, and an upward arrow.  Click the upward arrow (see screenshot).

Screenshot boto3 upgrade.

Now again try the detect\_text() function. We only require text therefore we require “**DetectedText**” field of “**TextDetections**”. Let’s store the result in a string:

imgString=**""  
for** img **in** rekresp[**'TextDetections'**]:  
 imgString=imgString+img[**'DetectedText'**]+**" "**

As we can observe that the text is repeated two times but we require the text as once. Well, there is a work around this, divide the string in two parts:

half, rem = divmod(len(imgString), 2)  
frontA = imgString[:half + rem]  
backA = imgString[half + rem:]

Now print only the “frontA”. Done

In this module we will learn how to make AWS Rekognition identify our self.

As we discussed in the previous module, that we will be using Collection. Let’s start with creating a collection. To create a collection for storing images, that function provided by Boto3 is create\_collection():

rekresp = client.create\_collection(  
 CollectionId=**'myCollection'**,  
)

pprint(rekresp)

We are using “**myCollection**” as our collection name, but we can also give it some other name. It creates a collection in an AWS Region. The above program output will look like:

*{'CollectionArn': 'aws:rekognition:us-east-1:050122124972:collection/myCollection',  
 'FaceModelVersion': '2.0',  
 'ResponseMetadata': {'HTTPHeaders': {'connection': 'keep-alive',  
 'content-length': '124',  
 'content-type': 'application/x-amz-json-1.1',  
 'date': 'Sat, 25 Nov 2017 00:43:12 GMT',  
 'x-amzn-requestid': '9d729749-d179-11e7-a131-8fd57d6e1f6f'},  
 'HTTPStatusCode': 200,  
 'RequestId': '9d729749-d179-11e7-a131-8fd57d6e1f6f',  
 'RetryAttempts': 0},  
 'StatusCode': 200}*

Now that we have a collection, let’s add faces to the collection. “index\_faces” function does that for us. We would be providing four parameters for this function: Collection name, Image byte, Name to the image, Attributes to be stored.

In coding the parameters would like:

rekresp = client.index\_faces(CollectionId=**'myCollection'**,Image={**'Bytes'**: imgbytes},ExternalImageId=**'Name\_OF\_Person'**,  
 DetectionAttributes=[**'ALL'**])

And we are done. We have stored the image, now what’s left is to compare each face with the input image and check if they match or not. Again compare\_faces is not the answer. Boto3 provides another method to perform this task. But remember, if it gets face match for one face, it will return with that value. The function will look like:

response = client.search\_faces\_by\_image(CollectionId=**'myCollection'**, Image={**'Bytes'**: imgbytes},FaceMatchThreshold=70)  
*# return response*str=**""  
if** len(response[**'FaceMatches'**])==0 :  
 str=str  
**else**:  
 **for** r **in** response[**'FaceMatches'**]:  
 str=str+r[**'Face'**][**'ExternalImageId'**]+**" "  
  
return** str

This function also return the Bound Box of the image. Therefore we can draw the rectangle using this value.

Now let’s see how this will look like when implemented together:

#Getting the labels for the given image

name=get\_labels(img,confidence)  
**for** label **in** name:  
 nametext.append(label[**'Name'**])  
text=**', '**.join(nametext) #text is the value that we would be passing for printing the labels on the image.

#Let’s write code for wishing/ greeting

hour=datetime.datetime.now().time().hour  
**if** hour<=10:  
 texttpspeech = **"Good Morning. "  
elif** hour>10 **and** hour<=16:  
 texttpspeech = **"Good Afternoon. "  
else**:  
 texttpspeech = **"Good Evening. "**

#texttsspeech is the variable that will be storing all the content to be giving to the AWS POLLLY for speaking

# append the labels to the AWS POLLY variable

texttpspeech=texttpspeech+**"The image contains : "**+text+**". "**

#Calling the function to check whether the image contains text or not

text\_on\_image = text\_image.textImage(imgbytes)

**if** text\_on\_image == **"" or** text\_on\_image == **" "**:  
 texttpspeech = texttpspeech  
**else**:  
 texttpspeech = texttpspeech + **" The text written on image is "** + text\_on\_image + **". "**#creating a list to check whether the image contains any humans so that we can scan for face match with celebrity or with myCollection collection

myList = [**'Human'**,**'People'**,**'Person'**,**'Female'**,**'Male'**,**'Girl'**,**'Boy'**,**'Woman'**,**'Man'**,**'Face'**,**'Laughing'**,**'Smile'**,**'Blonde'**,**'Crowd'**]  
flag = 0  
celebrity = []  
**for** i **in** myList:

# If not a celebrity then search in myCollection collection  
 **if** flag == 0 **and** texttpspeech.\_\_contains\_\_(i):  
 flag=1  
 *#pprint("Text found "+i)* texttpspeech = texttpspeech + (str)(face\_detect.describe\_faces(imgbytes))  
 celebrity = celebs.celeb(imgbytes)  
 **if not** celebrity:  
 foundFaces=searchFacesInCollection.search\_faces(imgbytes)  
 **if** foundFaces==**""**:  
 texttpspeech=texttpspeech  
 **else**:  
 foundFaces=foundFaces.replace(**"\_"**,**" "**)  
 texttpspeech = texttpspeech + **"I can recognize the personal as "**+ foundFaces+**". "** face\_detect\_draw.acceptImagebytes(imgbytes)  
 **else**:  
 texttpspeech = texttpspeech + **"I can recognize the personal as : "** + **', '**.join(celebrity) + **'. '** text = **', '**.join(celebrity)  
 **break**

#Final give the whole text to AWS Polly to speck.

apptext.text\_to\_speech(texttpspeech,**'Amy'**)

Outputs Screenshot

I am creating tutorial on “Cool uses of Polly, Rekognition”.

I basically created a program that will describe the labels, print labels on the image, detect text, find celebrity, print the names of the celebrity on the image, find non-celebrity face (therefore, face stored in collection), Call AWS Polly to wish/greet and speak out these details.

“SayItApp” is the name of the project. When you open it in PyCharm, first run “createCollection.py” to create the collection, in which we will store the faces. Then run “addImageToCollection.py” to add faces to the collection. Run the “addImageToCollection.py” again with following changes:

imgfile = **'image/** **293af07.jpg'**;

rekresp = client.index\_faces(CollectionId=**'myCollection'**,Image={**'Bytes'**: imgbytes},ExternalImageId=**'** Phil\_Ventura **'**,  
 DetectionAttributes=[**'ALL'**])

Now we are good to run the “labels\_graphical.py”. Don’t run the “Doctests in labels\_graphical.py”, make the changes suggest by Professor in Mid Term.

The “index.html” is the default page. There is a link of “Tutorial”, hover over it. The tutorial pages will be displayed.