AUTOMATIC IMAGE CAPTIONING AND SPEECH GENERATION

Ge Zhang

Hanwen Liang

Yaoxian Su

REPORT SUBMITTED IN PARTIAL FULFILLMENT FOR ECE1779 INTRODUCTION TO CLOUD COMPUTING IN MASTER OF ENGINEERING AT UNIVERSITY OF TORONTO

December 2018

1. PROBLEM STATEMENT AND SOLUTION OVERVIEW

Problem Visual impairment, also known as vision impairment is a decreased ability to see to a degree that causes problems not fixable by usual means, such as glasses. According to the World Health Organization, 285 million people are visually impaired worldwide, including over 39 million blind people. Living with visual impairment can be challenging, since many daily-life situations are difficult to understand without good visual acuity.

Technology has the potential to significantly improve the lives of visually impaired people, such as image reader. Given an image to image caption generator, it has the capability to generate natural language sentences which will be read out aloud to describe the content of this image, thus enabling the blind to access previously inaccessible information. However, technique complexity of image captioning has limited its performance and impeded its spread and application. Also, in the model experimental stage, it's hard to give a valid evaluation of its describing performance due to lack of public feedback. Implementation of automatic image captioning into applications such as web interfaces has the potential to assist visually impaired surfers as well as collecting feedbacks from the users for model and algorithm improvement.

Technical Complexity Being able to automatically describe the content of an image using properly formed English sentences is a very challenging task. This task is significantly harder, for example, than the well-studied image classification or object recognition tasks, which have been a main focus in the computer vision community. Indeed, a description must capture not only the objects contained in an image, but it also must express how these objects relate to each other as well as their attributes and the activities they are involved in. Moreover, the above semantic knowledge has to be expressed in a natural language like English, which means that a language model is needed in addition to visual understanding.

Solution Overview In this project, we have addressed the visual impairment and feedback collection issue by developing a web-based image captioning application called "Visual Speaker" that implemented a model by fusing both image captioning prototype and a text-to-speech synthesizer. The image captioning part takes advantage of neural network models, a variant of Recurrent neural network (RNN) coupled with a Convolutional neural network (CNN). We used MSCOCO dataset and deployed, trained and tested model ourselves. The captions are then transformed into natural-sounding speech using the IBM Text to Speech service. We embedded the whole model on the web application based on Amazon Web Service platform. We use multiple modules including Lambda, DynamoDB, S3 and EC2 to realize our functionality.

On one hand, by capturing photographs of the surroundings etc, the visually impaired can upload the images to our website to generate captions that can be read out loud to give visually impaired people a better understanding of the world. On the other hand, those who are interested in the functionality of the website, like families of the visually impaired, are able to create their own account in "Visual Speaker" community, where they can go through all the updated images from all users as well as image's captions. They can give their grades to the captions where will be collected for future research usage.

2. Application Architecture

2.1 Web Architecture

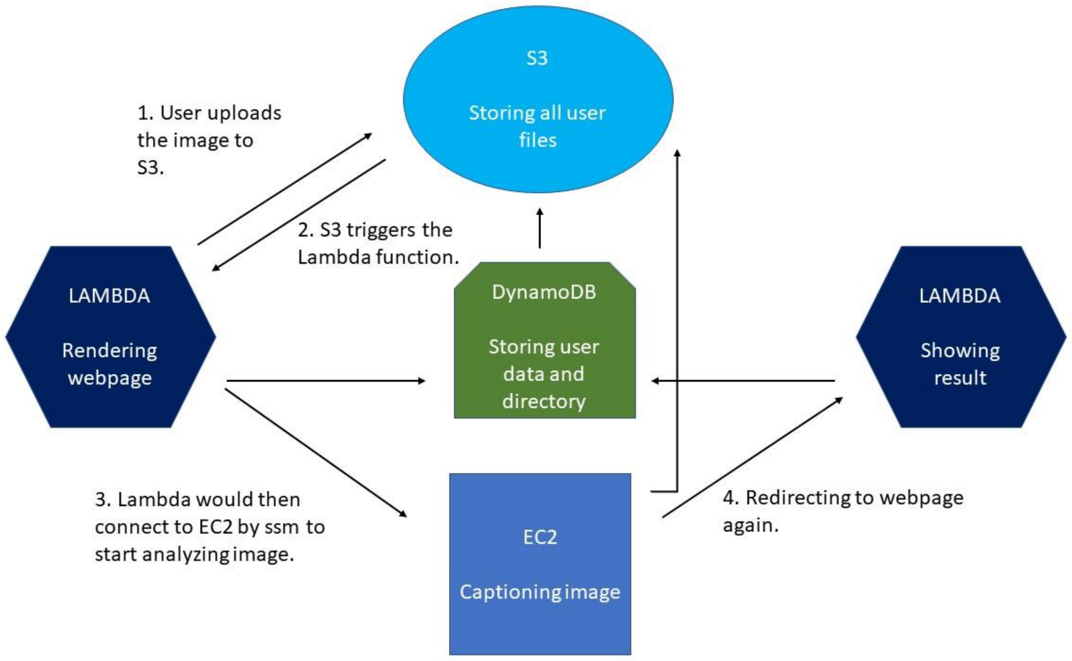

S3
Storing all user files
1. User uploads the image to S3.
2. S3 triggers the Lambda function.
LAMBDA
Rendering webpage
DynamoDB
Storing user data and directory
LAMBDA
Showing result
3. Lambda would then connect to EC2 by ssm to start analyzing image.
EC2
Captioning image
4. Redirecting to webpage again.

Figure 1: Architecture of Web Application

Above is the developing architecture of our web application. The web application is mainly running at Lambda serverless module. Since the dependencies of deep learning model is too large to be deployed on lambda, we use EC2 instance to handle image captioning task. The data is recorded in DynamoDB, and all the files are saved in S3 bucket, and the most http requests are handled by Lambda. When a user uploads an image, it will be automatically saved in user's personal folder in S3 bucket. Then, the saving operation will trigger Lambda handler function to inform the EC2 instance to process the newly uploaded image. Deep learning model on EC2 instance will generate text and audio file and push them into S3 bucket. Once detecting the newly generated text and audio file in S3 bucket, Lambda handler will update the webpage. In this process, both EC2 instance and Lambda will interact with DynamoDB to record file information.

2.2 Model Architecture

The generation of captions replies on a deep neural network consisting of a CNN type model as encoder at image end, an RNN type model as decoder at language end, followed by a speech synthesizer. It can generate complete sentences in natural language as shown by the example in Figure 2. In general, a raw image is pre-processed and fed to a CNN model to produce a representation of the it. Such a feature representation is map into a pretrained RNN model which utilizes this piece of information to generate words one by one to composite a sentence. Afterwards, the sentence will be read out by a Text to speech API.

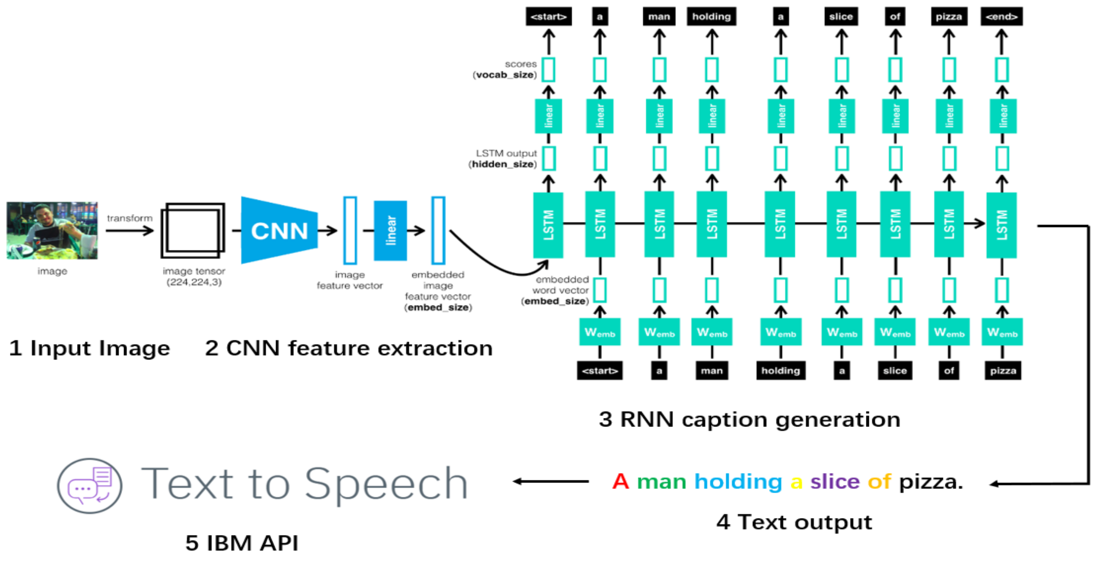


Figure 2. Architecture of Model

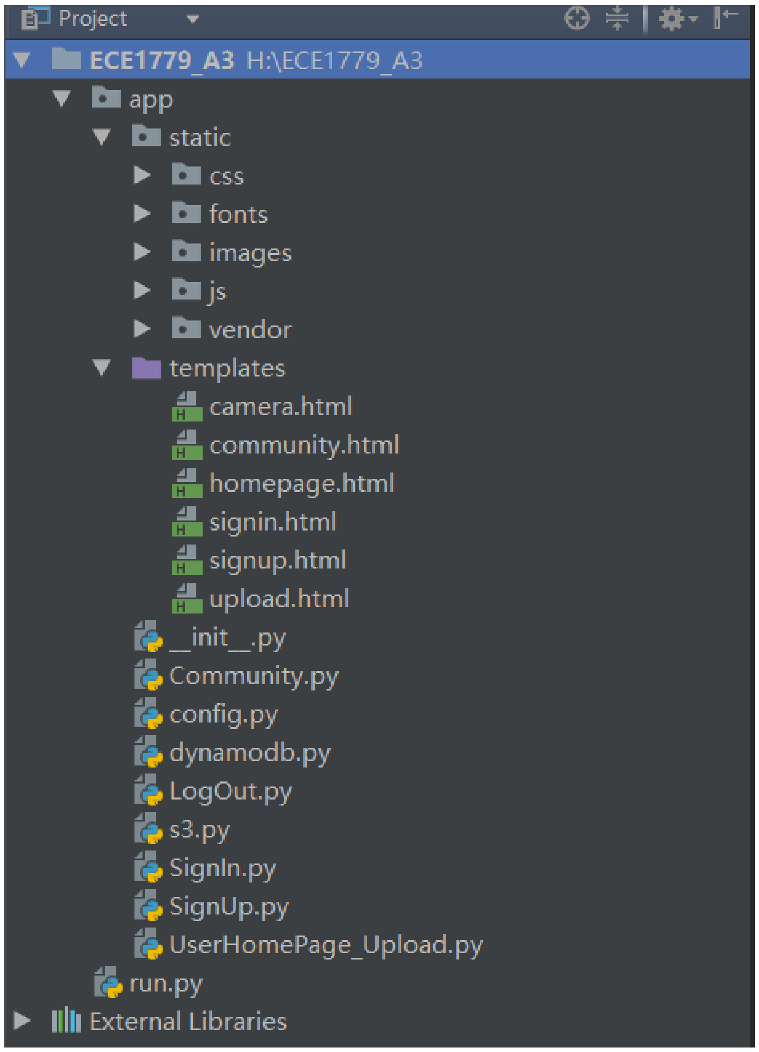
3. DEPLOYMENT DETAIL

3.1 Browser

The web browser serves as the graphical user interface to general users and as the client to the Flask web development server. The main functionality of the web browser is to collect information from users in the format of forms and to display the uploaded image and its audio as well as any additional information such as errors and warnings from the backend Flask web server to the users in a user-friendly environment. The HTTP post and put request methods will be issued by the web browsers to push collected information to the Flask web server and to retrieve information from it.

3.2 Flask Web Server

The web server used for this web application is named as Flask. Flask is a micro web framework written in Python which based on Werkzeug toolkit and Jinha2 template engine. The Flask web server in this architecture plays the backend web server role to process the HTTP request from the web browser and the client role to send requests to S3 bucket for storing user specific file. Meanwhile, the Flask web server also interact with DynamoDB for recording user data information.



```
Project
▼ ECE1779_A3  H:\ECE1779_A3
    ▼ app
        ▼ static
            ▶ css
            ▶ fonts
            ▶ images
            ▶ js
            ▶ vendor
        ▼ templates
            camera.html
            community.html
            homepage.html
            signin.html
            signup.html
            upload.html
        __init__.py
        Community.py
        config.py
        dynamodb.py
        LogOut.py
        s3.py
        SignIn.py
        SignUp.py
        UserHomePage_Upload.py
    run.py
▶ External Libraries
```

Figure 3. Project structure in Flask Web Server

3.2.1 HTML and CSS

We store all html files under the '/app/static' directory and the CSS stylesheet for each HTML page under the '/app/templates' directory. Bootstrap framework is used to program HTML and CSS sheets due to its responsive grid system and extensive prebuilt components. It enables application to readjust its components and provides more friendly and usable experience to users.

3.2.2 JavaScripts

We store JavaScript files under './app/static' which are used to perform several complementary functions and add decoration tags for html files. Besides, it also completes the web camera and snapshot front-end page.

3.2.3 Python files

We store all python files under './app'. Python files define all web behaviors including implementation of uploading image to S3 bucket, communicating with DynamoDB and rendering html files, etc..

3.2.4 Cryptography

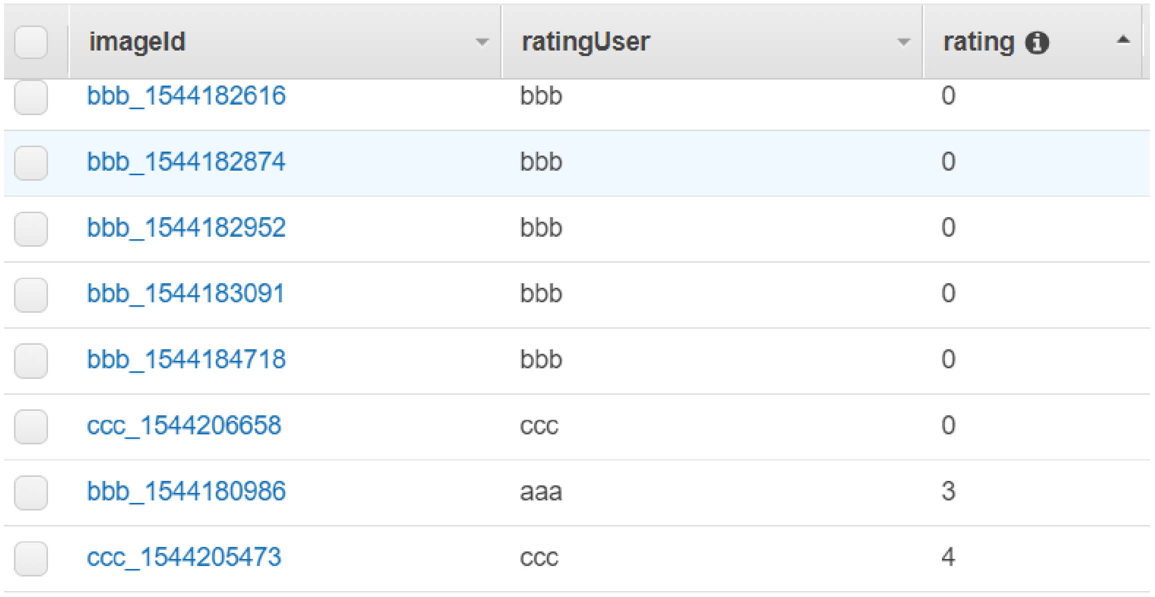
To store the hash of password concatenated with a per-user salt value, hashlib module is imported to Flask in both SignIn.py and SignUp.py files. For new user who sign up to our web application, first we will check if user name and email are valid and hasn't been used before. Then, for security purpose, we will save sha-256 hash value of the password added with some randomly generated per-user salt, so that the password won't be stored in plain text and the hash value of password for different user will be different.

When registered user want to sign in again, application will read user information from DynamoDB and then use salt and entered password to regenerate sha-256 hash value to compare with saved sha-256. If regenerate hash value equals to saved hash value, then password entered correctly, otherwise, deny this sign in.

3.3 DynamoDB

DynamoDB database in this architecture is used to store information submitted by user during the signup phase, the information of images as well as image captions submitted by the user during the upload images phase and the rating of images whenever a user rates images in community. Here we use a total of three tables to store the information:

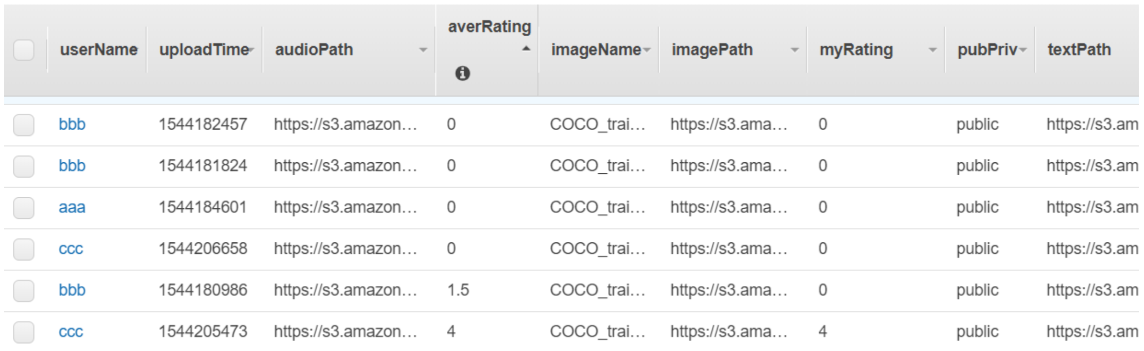
1). 'ratingTable': used to save user's rating information for each image, including 'imageId' (Primary partition key), 'ratingUser' (Primary sort key) and 'rating'.



imageId	ratingUser	rating
bbb_1544182616	bbb	0
bbb_1544182874	bbb	0
bbb_1544182952	bbb	0
bbb_1544183091	bbb	0
bbb_1544184718	bbb	0
ccc_1544206658	ccc	0
bbb_1544180986	aaa	3
ccc_1544205473	ccc	4

Figure 4: Schema of 'ratingTable'

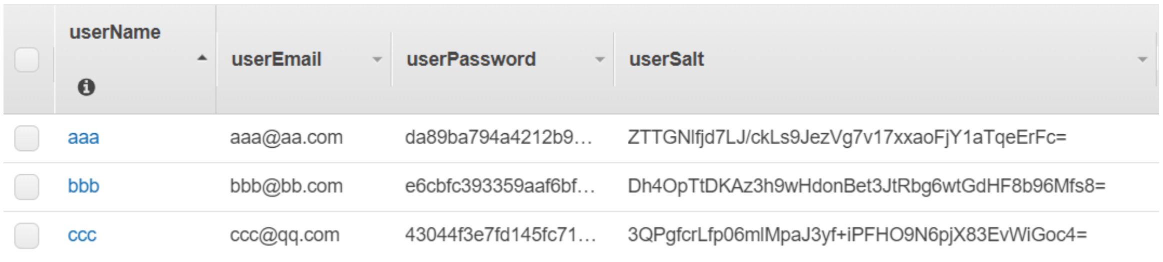
2). 'user2Image': used to save images' information uploaded by users, including 'userName' (Primary partition key), 'uploadTime'(Primary sort key), 'audioPath', 'averRating', 'imagePath', 'imageName', etc.. We also add "Global Secondary Index" to this table which uses "imageName" as partition key and "uploadTime" as sort key.



userName	uploadTime	audioPath	averRating	imageName	imagePath	myRating	pubPriv	textPath
bbb	1544182457	https://s3.amazon...	0	COCO_trai...	https://s3.ama...	0	public	https://s3.am
bbb	1544181824	https://s3.amazon...	0	COCO_trai...	https://s3.ama...	0	public	https://s3.am
aaa	1544184601	https://s3.amazon...	0	COCO_trai...	https://s3.ama...	0	public	https://s3.am
ccc	1544206658	https://s3.amazon...	0	COCO_trai...	https://s3.ama...	0	public	https://s3.am
bbb	1544180986	https://s3.amazon...	1.5	COCO_trai...	https://s3.ama...	0	public	https://s3.am
ccc	1544205473	https://s3.amazon...	4	COCO_trai...	https://s3.ama...	4	public	https://s3.am

Figure 5: Schema of 'user2Image'

3). 'userInfo': used to save user's personal information, including 'userName'(Primary partition key), 'userEmail', 'userPassword'(in sha-256 hash), 'userSalt'.



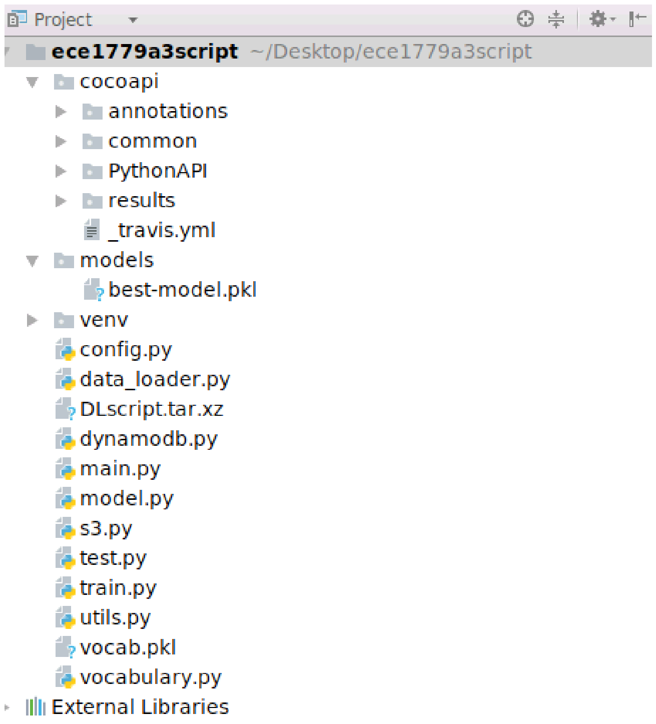
userName	userEmail	userPassword	userSalt
aaa	aaa@aa.com	da89ba794a4212b9...	ZTTGNlfjd7LJ/ckLs9JezVg7v17xxaoFjY1aTqeErFc=
bbb	bbb@bb.com	e6cbfc393359aaf6bf...	Dh4OpTtDKAz3h9wHdonBet3JtRbg6wtGdHF8b96Mfs8=
ccc	ccc@qq.com	43044f3e7fd145fc71...	3QPgfcrLfp06mlMpaJ3yf+iPFHO9N6pjX83EvWiGoc4=

Figure 6: Schema of 'userInfo'

When user sign in or sign up, we use "put_user" and "get_user" functions to help Lambda to retrieve user information from 'userInfo' table. When a user views images or uploading image, we take advantage of functions like "get_image_item" and "get_all_pub_image" to help Lambda obtain information like "upload time" and "average rating" from 'user2Image' table to display images. Whenever a user rates an image, the data in 'ratingTable' will be updated by "update_rating_item" function.

3.4 EC2 instance (computation module)

Our machine learning model and dependencies are too large to be directly deployed on Lambda. Instead, we put the model on a computational optimized EC2 instance. When user upload a new image to community, it will first be uploaded to S3. Then, lambda handler function will be triggered and pass the URL address of this image to the EC2 instance to analysis. On EC2 instance, the machine learning model will upload generated audio and text file to S3.

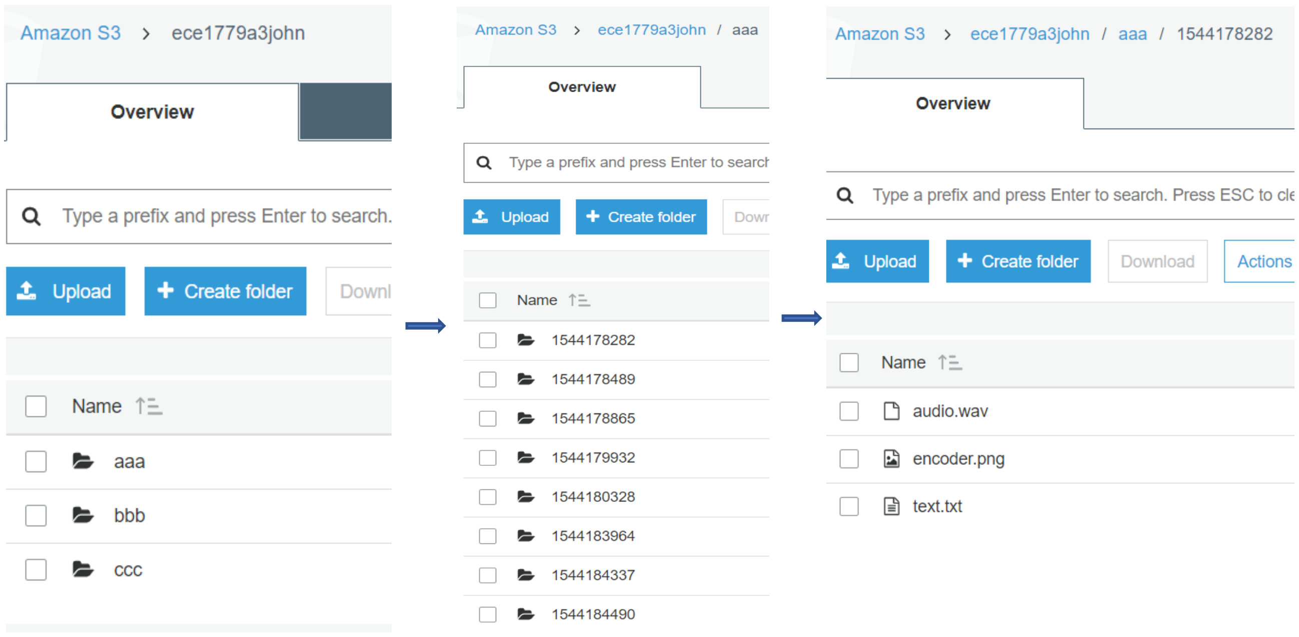


```
Project
ece1779a3script ~/Desktop/ece1779a3script
  cocoapi
    annotations
    common
    PythonAPI
    results
    _travis.yml
  models
    best-model.pkl
  venv
  config.py
  data_loader.py
  DLscript.tar.xz
  dynamodb.py
  main.py
  model.py
  s3.py
  test.py
  train.py
  utils.py
  vocab.pkl
  vocabulary.py
External Libraries
```

Figure 7. Model Implementation structure in EC2 instance

3.5 S3 bucket

In our web application, we will save all uploaded image files together with generated audio and text files on S3. Each user will have their own folder. In their personal folder, different images will be stored in different folders, which are distinguished by upload time (UNIX Time). The file structure of S3 bucket is shown in Figure 8 below.



Amazon S3 › ece1779a3john

- aaa
- bbb
- ccc

Amazon S3 › ece1779a3john / aaa

- 1544178282
- 1544178489
- 1544178865
- 1544179932
- 1544180328
- 1544183964
- 1544184337
- 1544184490

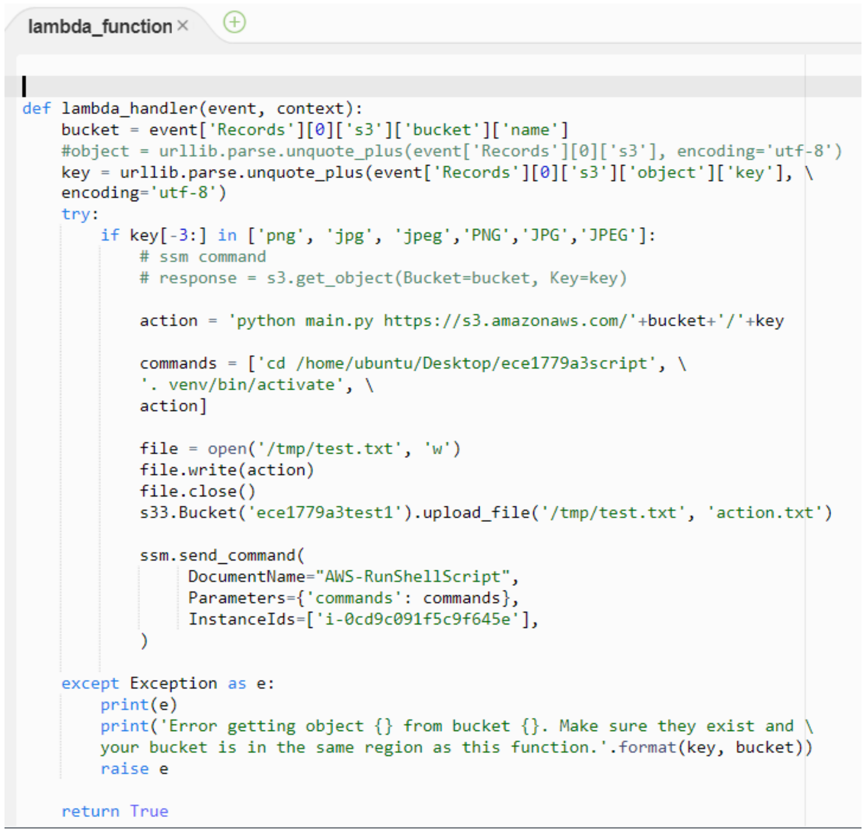
Amazon S3 › ece1779a3john / aaa / 1544178282

- audio.wav
- encoder.png
- text.txt

Figure 8. File Architecture in S3 bucket

3.6 Lambda/Zappa - Serverless Python

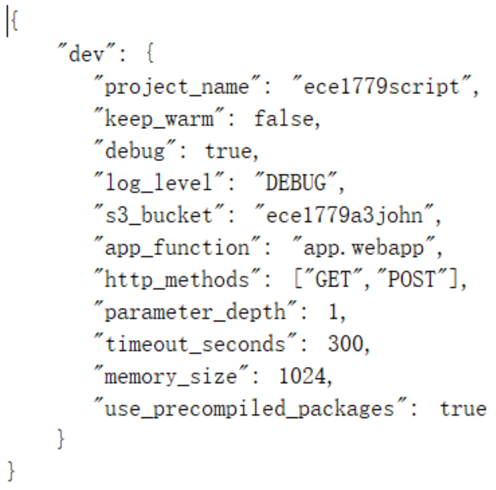
We take advantage of Zappa to deploy Amazon Web Service Lambda function. The flask application is implemented as a lambda function on AWS. When user at website requests to upload an image, the image will be uploaded to S3 first, which will trigger Lambda to active EC2 to do image captioning task and save generated file to S3. Then S3 will trigger Lambda again to update user websites.



lambda_function

```python
def lambda_handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    #object = urllib.parse.unquote_plus(event['Records'][0]['s3'], encoding='utf-8')
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], \
    encoding='utf-8')
    try:
        if key[-3:] in ['png', 'jpg', 'jpeg','PNG','JPG','JPEG']:
            # ssm command
            # response = s3.get_object(Bucket=bucket, Key=key)

            action = 'python main.py https://s3.amazonaws.com/'+bucket+'/'+key

            commands = ['cd /home/ubuntu/Desktop/ece1779a3script', \
            '. venv/bin/activate', \
            action]

            file = open('/tmp/test.txt', 'w')
            file.write(action)
            file.close()
            s33.Bucket('ece1779a3test1').upload_file('/tmp/test.txt', 'action.txt')

            ssm.send_command(
                DocumentName="AWS-RunShellScript",
                Parameters={'commands': commands},
                InstanceIds=['i-0cd9c091f5c9f645e'],
            )

    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. Make sure they exist and \
        your bucket is in the same region as this function.'.format(key, bucket))
        raise e

    return True
```

Figure 9. Lambda handler function

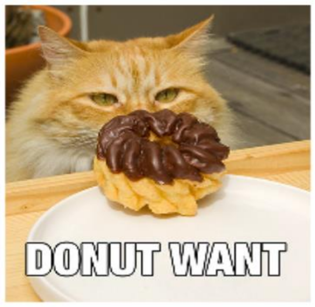


```json
{
    "dev": {
        "project_name": "ece1779script",
        "keep_warm": false,
        "debug": true,
        "log_level": "DEBUG",
        "s3_bucket": "ece1779a3john",
        "app_function": "app.webapp",
        "http_methods": ["GET","POST"],
        "parameter_depth": 1,
        "timeout_seconds": 300,
        "memory_size": 1024,
        "use_precompiled_packages": true
    }
}
```

Figure 10. Zappa deployment file

3.7 Deep Neural Network Implementation

We trained our model based on MSCOCO dataset which consists of around 80K training images, 40K validation images and 40K test images. Each image comes with five descriptive captions, as shown in figure 3. We implemented the pretrained model ResNet50 from PyTorch framework as CNN decoder and constructed RNN decoder based on LSTM cell-model. The model is trained using the mini-patch stochastic gradient descent (SGD) method with learning rate 0.001. Then the best model is selected based on total loss value on the validation set. Training model on GTX TITAN X 12GB takes about 3 days.

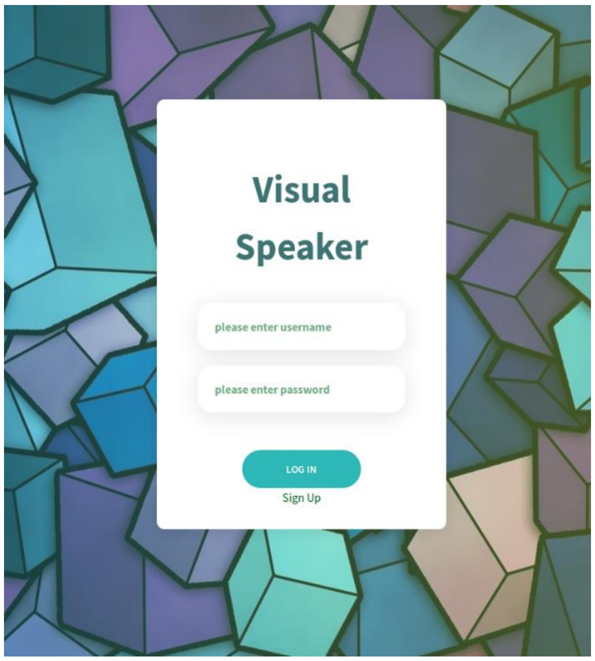


```
A cat sitting in front of a delicious chocolate donut.
A cat looking at a donut on a plate.
A cat is looking over a chocolate covered donut on a plate.
A cat stares at a chocolate topped donut, with the caption reading, "DONUT WANT."
Cat looks at donut with words "donut want" along bottom
```

Figure 11: Sample Image Captioning training data

4. USER INSTRUCTION

Open your favorite browser on your computer and enter https://lzw8wn3n40.execute-api.us-east-1.amazonaws.com/dev in the address bar to access the login page of “Visual Speaker”, which is shown in Figure 12. If you already have an account, please sign in by typing in your username and password. Clicking “LogIn” button and after verification, you will log into visual speaker community page where you can see all the images in our community.



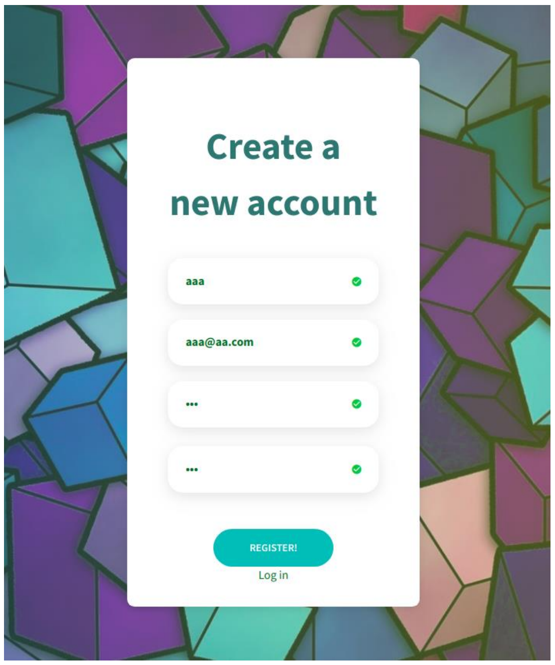
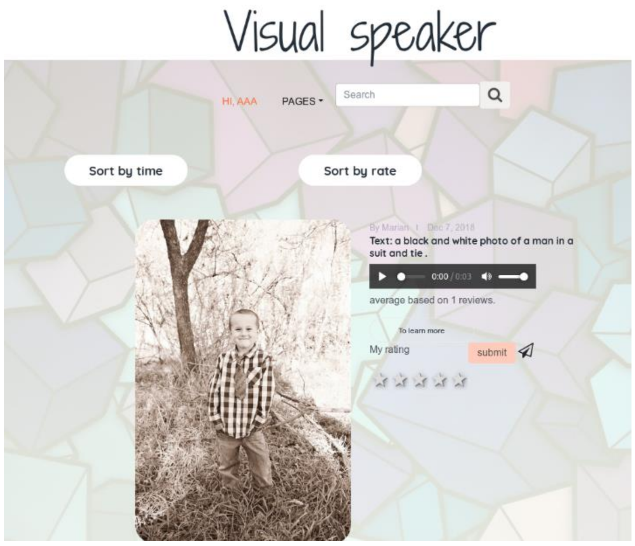


Figure 12: LogIn and SignUp page

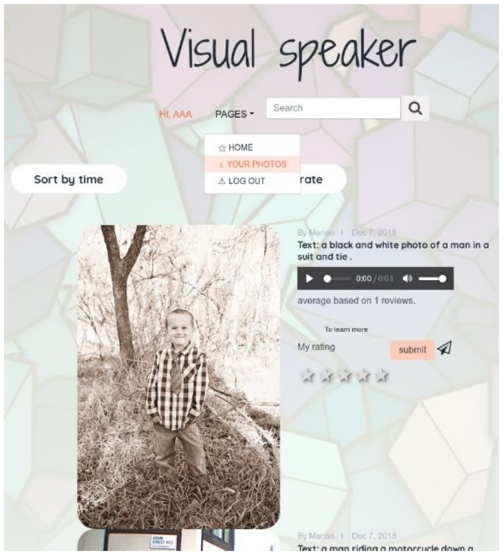
However, if you are first time to Visual Speaker community, please create a new account by clicking on the SignUP button to access the signup page for creating user account and password. Once finished, please click the "Register" button to redirect to the community page. Note that Username, Email, Password fields are mandatory. The retyped password must be the same as first password. The username and email address cannot be identical to other users. Once logged into the community page, you are shown all the images uploaded by all users, as is shown in Figure13.



Text: a black and white photo of a man in a suit and tie .

average based on 1 reviews.

Figure 13. Community Page



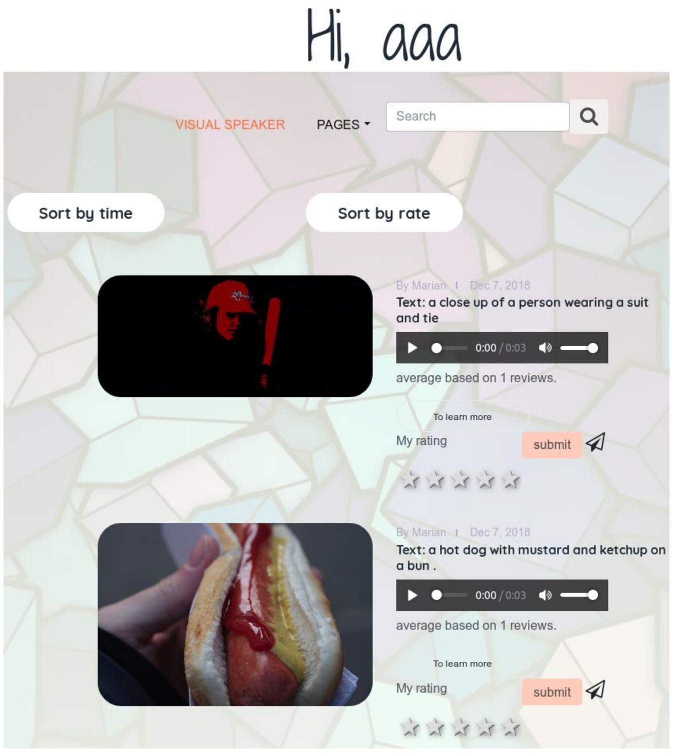
Text: a black and white photo of a man in a suit and tie .

average based on 1 reviews.

Text: a man riding a motorcycle down a

Figure14. To direct to personal page

For each image, there is “Text” which describes the image and an audio which speak out the “Text”. You are able to give rating to all the images about the captions and submit by clicking “submit” button. In this page, we also have buttons like “Sort by time” and “Sort by rate” which can arrange the images according to updated time or rating score. We also develop search bar where you can search image by image name. As shown in Figure 14, clicking “pages”, you can see “Home” choice which redirect you to current page to update, “YOUR PHOTOS” which redirect you to your personal page.



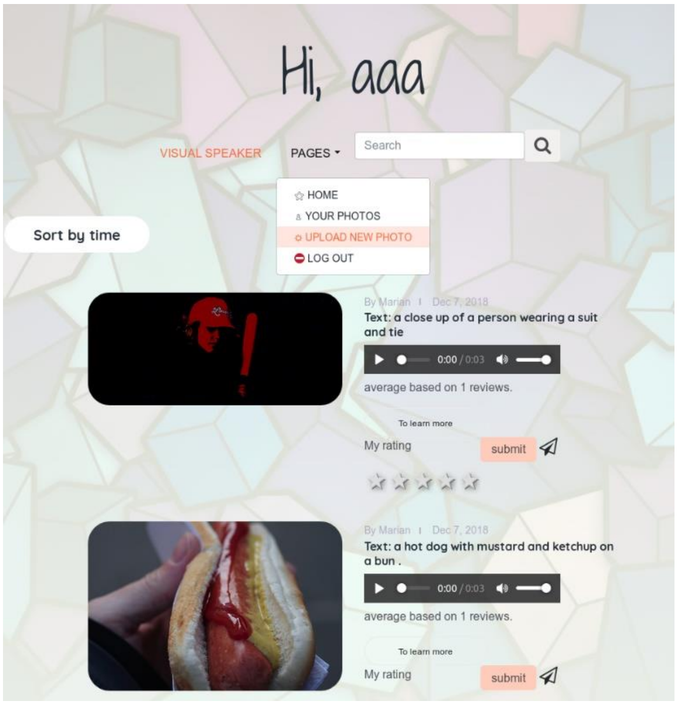


Figure15. Personal Page of user “aaa”

Figure 16. To redirect to upload image page

The personal page as shown in figure15 shows all the images upload by yourself. Clicking “PAGES” button, you have many choices like “HOME” which redirects you to community page, “UPLOAD NEW PHOTO” which directs you to upload new images. In the upload image page, by clicking “Drop a photo here or click to select one” window, you can select image from your local laptop and upload it by clicking “Upload” button.

For those visually impaired people, to facilitate them to easily use the application in practical life, we also develop a camera page which is accessible from “LogIn” page. After redirected to camera page, users are given a camera window where they can use modern devices to take pictures of their surroundings by clicking “Take Snapshot” button. Then the image will be uploaded directly to

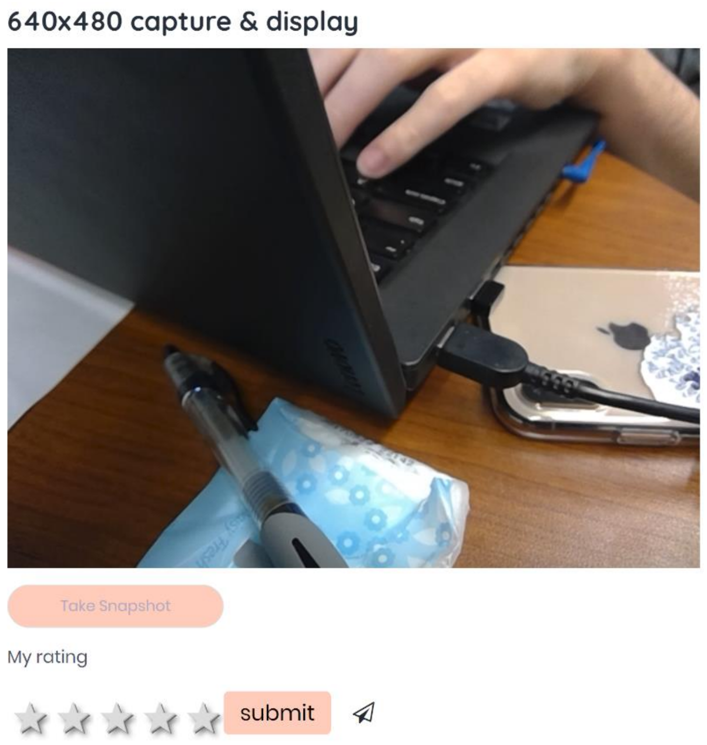


Figure 17: Camera Page

generate captions and audios which could be read out aloud to tell the users the surrounding information.

5. CONCLUSION AND PROMOTION

Being able to automatically describe the content of an image using properly formed English sentences is a challenging task, but it could have great impact by helping visually impaired people better understand their surroundings. In this project, we successfully built a web application called "Visual Speaker" that could facilitate the visually impaired to better understand the world. This application takes advantage of multiple AWS services as well as state-of-art deep learning models. Equipped with this application, visually impaired people are able to catch information which is previously inaccessible by a simple click. Also, many leading technology companies like Google, Amazon and IBM have invested large amount of money on AI research including image captioning and lack of feedback data has always been an issue. Our application creates a community which could not only entertain the public but also collect valuable feedbacks from users to improve their models. In a nutshell, our application has extraordinary investment value.