Image Analysis

Sam Beaulieu - srb2208
Columbia University

Abstract

Image Analysis predicts how an image will be received, in number of likes and comments, when it is uploaded to Instagram. Image Analysis bases its predictions on a user specific machine learning algorithm. The algorithm determines the contents of a user's posted images and uses that data, along with those images' likes and comment data, to find a relationship between image content and reception metrics. When making a prediction, the contents of a new image are determined and run through the model to estimate the number of likes and comments the image will receive. In this project, we've successfully implemented this service using Amazon Web Services.

1. Introduction

Image Analysis is a service that predicts the number of likes and comments an image may receive when uploaded to Instagram. The service is built on many Amazon Web Services (AWS) and uses a simple machine learning algorithm in the backend. The algorithm determines the contents of a user's posted images and uses that data, along with those images' likes and comment data, to find a relationship between image content and reception metrics. When making a prediction, the contents of a new image are determined and run through the model to estimate the number of likes and comments the image will receive.

1.1 Motivation

Instagram is a popular medium to share images and videos. As of September of 2017, Instagram had 800 million active monthly users [2]. Since Instagram's sole purpose as a social media platform is to share images and videos, it is fair to assume that this massive user base wants to share their media and have it be appreciated by their followers. This appreciation is shown through likes and comments.

Though many users may just *want* their followers to like and comment on their posts, others have a financial *need* for their posts to be received well. Many users on Instagram make a living from advertising products and many accounts are run by companies for public relations. For these users, ill received posts could hurt their personal or corporate brand and result in real consequences.

Image Analysis adds an objective opinion to how an image will be received. Since it is based on users' past posts, predictions are individualized. A company's public relations account will receive a very different recommendation than a college student's. Image Analysis is not meant to be a total replacement for human judgement, however it is meant to augment a decision with algorithmic results.

1.2 Problem

As motivated by the previous section, our objective is to provide Instagram users with a machine learning based platform to help them determine how well an Image might be received by their followers. For this project, we focus solely on predicting the response of images, however it could be expanded in the future to include videos. To achieve this goal, we examine four different sub-problems. These four sub-problems will drive the architecture and implementation of our project as described in Sections 2 and 3.

The first sub-problem is how to facilitate a user experience on our website. We want users to be able to have a machine learning model that learns over time and to be able to re-use a model once it's trained. For this, we need individualized accounts for our users. To make logging in and account creation easier, we will also allow users to login with Facebook and Amazon.

Our next sub-problem revolves around how we will target the machine learning model specifically to a single user. Because different users will have wildly different content, we must ensure that our website builds a model off of a single user's images. To make this a reality, we will need to integrate with each user's Instagram account. Once users link their account, we will build our model based off their specific user data.

Once we have user-specific Instagram data, the next problem is what machine learning algorithm to use and how to train it on a per-user basis. For simplicity, we decided to use a linear regression based model. We make the assumption that the response to a user's post is directly related to the content of the image. We realize that other factors may also affect a post's response - time of day posted, day of week posted, filter used, caption, tags, location, etc. - however we choose to focus on the image's content for the sake of this project. Future work could see more of these parameters taken into account.

Finally, just having a user specific model is not enough. We need to use the model to infer the response new images may receive. To do this, we will allow users to upload images to our site which we will run through the model in the backend. An estimated number of likes and comments will be returned to the user and displayed.

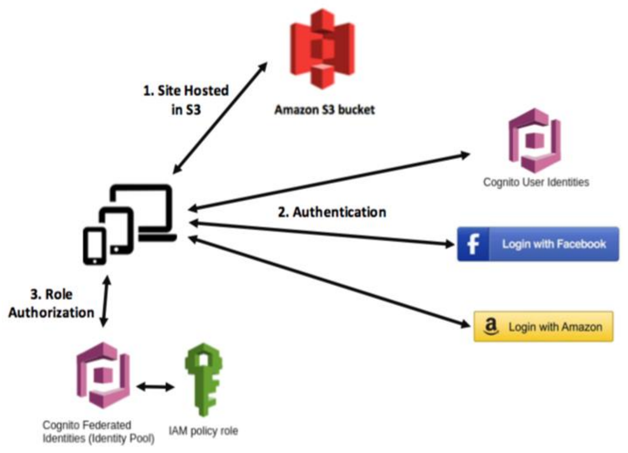

Figure 1: Authentication Flow

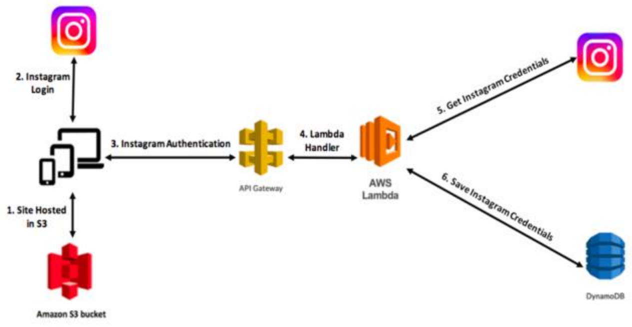

Figure 2: Connect Account Flow

2. Solution Architecture

To realize the service outlined in Section 1, we decided to utilize AWS. Specifically, we use Cognito for out site-specific login and temporary credential distribution and Identity and Access Management (IAM) roles for managing access to services. Our site and images will be stored in Amazon's Simple Storage Service (S3) and our user data and model will be stored in DynamoDB. Backend computation will be primarily handled by Lambda functions while Rekognition will be used to determine the content of images. Finally, the API Gateway will be used to ferry authenticated requests from the frontend website to the backend computation.

The following four sections discuss the architecture we use to solve the four problems explained in Section 1.2. Section 2.1 explains how we authenticate users while Section 2.2 examines how we connect Instagram accounts to our user accounts. Section 2.3 details the architecture behind training a model while Section 2.4 explains how images are inferred and results are provided.

For convenience, Figures 1, 2, and 4 are included in full size in Appendix B.

2.1. Authentication

Our users have three methods of authenticating with our service. They can create an account directly with our website or login using Facebook or Amazon. The authentication flow can be seen in Figure 1.

If users decide to create an account with our website, the whole account creation and login process is handled by Cognito User Identities. The service handles the secure login process at a different url and redirects users back to our main site when they have successfully logged in. When they return, there is a code in the url which is stripped using JavaScript. The code is then exchanged with Cognito User Identities (using Javascript and a JQuery POST request) for a token which is used later in the authentication flow.

Logging in with Facebook and Amazon is similarly straightforward. When users click to do so, they are brought to Facebook and Amazon to complete the login process. Once successfully logged in, they are returned to our site with a token.

Once authenticated with one of the three services, users will have a temporary token. The token, however, does not provide any access to our backend resources. Once a token is received, JavaScript takes the token and exchanges it with Cognito Federated Identities. Federated Identities returns temporary user credentials which last about an hour and contain the correct IAM permissions to make authenticated calls to our API. Using the API and these credentials, users are able to interact with our backend.

2.2. Connecting With Instagram

Once a user is authenticated and has proper credentials to integrate with the backend, they are able to connect to their Instagram account. Since training a model is dependent on a user's Instagram account, users must connect their account before training a model or making a prediction.

The flow to connect an account is shown in Figure 2. When a user clicks the connect with Instagram button, they are directed to the Instagram login page. Once they log in, they are redirected back to our site with an authorization code in the url. Frontend JavaScript strips this code and makes an API call to finish authentication. The request is handled by a Lambda function which exchanges the code with Instagram for full credentials and then saves the credentials to the user's entry in DynamoDB where they can be used by future backend flows.

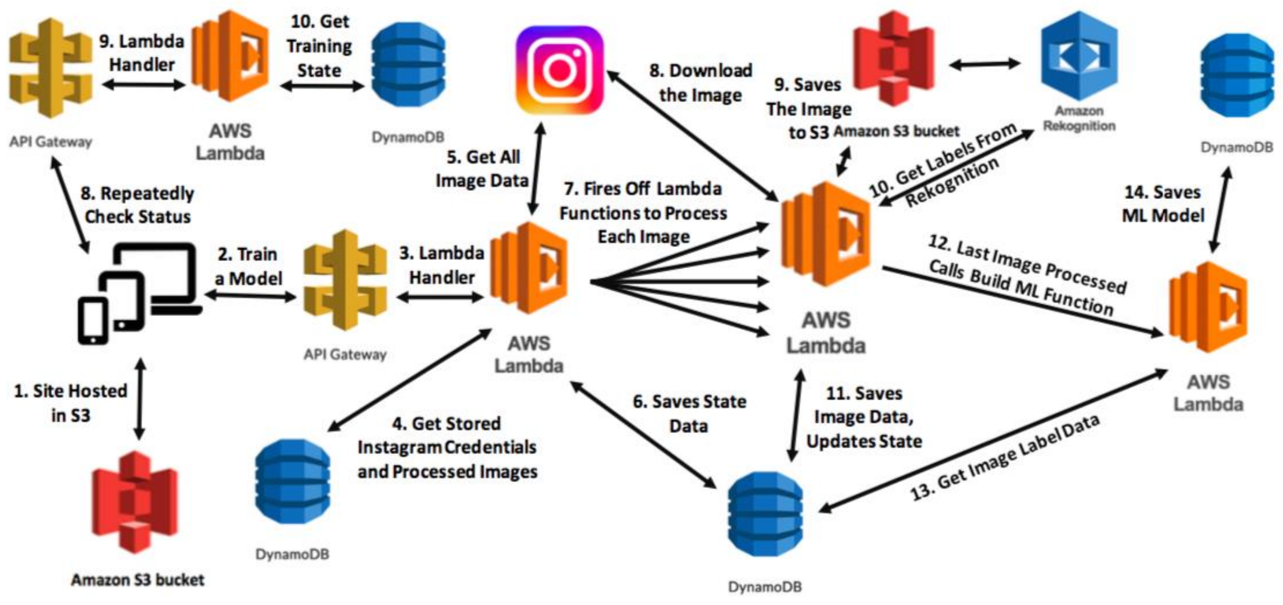

Figure 3: Flow To Train A Model

2.3 Training A Model

After users have connected their Instagram account, they have the option to train a model. The model is trained specifically to each user based on their previous Instagram posts. Once trained, users can use it to predict the response of new images. The flow for this is depicted in Figure 3. There are two sub-flows to this process as explained in the following two subsections.

2.3.1 Primary Flow

When a user first selects to train a model, a request is made to an API endpoint and is handled by a Lambda function. The Lambda function accesses the stored Instagram credentials from the flow in Figure 2 along with a list of ids of images that have already been processed. Then, the Lambda function makes a request to Instagram using the credentials to pull the metadata for all of a user’s images. The ids of which are cross referenced with the list of already processed image ids to generate a list of images that have not been processed yet.

Once the list is made, a request is sent to update the user in DynamoDB, Step 6 in Figure 3, to indicate that there is a model in training, no images have been processed yet, and there are N images to be processed where N is the length of the new image list. Once the database is prepared, the Lambda function asynchronously invokes a new Lambda function for each image in the new image list as seen in Step 7 in Figure 3. These new Lambda functions are passed all the metadata for one image from the list to be processed. We chose to asynchronously invoke these Lambda functions so all the images can be processed in parallel and the user can get feedback right away from the first Lambda function’s callback that a model is being trained.

If no new images are found, the whole train model flow stops after the user is updated in DynamoDB in Step 6. Instead of being updated to indicate a model is in progress, it updates to indicate a model is not in progress and training has been completed.

After the process image Lambda functions have been invoked in Step 7, they process their image metadata and download the actual image from Instagram. The image is then saved to a user specific folder in an S3 bucket.

After downloading an image, we need a way to determine the contents of the image. To do this, we use Amazon Rekognition, a service that analyzes images and returns a list of labels that correspond with that image. The image processing Lambda function makes a request to Rekognition to analyze the image for a list of labels and then updates the user in DynamoDB.

The DynamoDB update adds the image metadata and label data to a list of images as well as increments the number of processed images. The update request returns the updated user data. If the updated number of processed images is equal to the number of images to be processed (entered into the database in Step 6 in Figure 3), the Lambda function knows there are no more images that need to be processed.

If there are no more images to be processed, the Lambda function asynchronously invokes another Lambda function to finish building the model. This function pulls all the image data from DynamoDB and makes an object mapping unique image labels to the average number of likes and comments of the images those labels were seen in. It then saves this map to the users entry in DynamoDB where it can be used as the model for the user's predictions.

2.3.2 Check Status Flow

When the Lambda function invoked in Step 3 in Figure 3 returns to the user, it indicates whether or not a model is being trained. Since the rest of the training flow is called asynchronously, there is no way for the last Lambda function, invoked in Step 12 in Figure 3, to return to the website that the model has been completed. This feature is handled by a second, complementary flow that runs in parallel. In Figure 3, this flow can be seen in Steps 8, 9, and 10 in the top left.

After the Lambda function that handles the initial API request returns to the user indicating a model is being trained, frontend JavaScript repeatedly makes API requests to check the status of the model. These requests are handled by another Lambda function which gets the current model's status from DynamoDB. It is then able to return to the user whether a model is being trained and how many images have been processed out of how many that need to be processed. The frontend displays this data to the user and repeatedly updates until the model is fully trained.

2.4 Inferring An Image

The final step in providing a prediction to a user is to infer the number of likes and comments a new image will receive. The flow for this process is much simpler than the one to train a model and can be seen in Figure 4.

The flow is initiated when the user selects to upload and analyze a new image. Frontend JavaScript enables the image to be uploaded and displayed locally. After the upload is complete, it triggers another API request with the image which is once again handled by a Lambda function. The Lambda function saves the image to a user specific upload folder in S3 and then calls Rekognition to provide a list of labels that describe the image.

The Lambda function then makes a request to DynamoDB to get the model mapping labels to likes and comments as seen in Step 7 in Figure 4. Then the function calculates the average number of likes and comments across the labels provided by the new image that are found in the model. For instance, if the new image has the labels "forest," "stream," and "person" and "forest" and "person" are in the model and associated with 100 and 200 likes respectively, then the new image will be projected to receive 150 likes (assuming "stream" is not found in the model).

After determining the estimated number of likes and comments for the uploaded image, the Lambda function returns them to the website where they are displayed to the user.

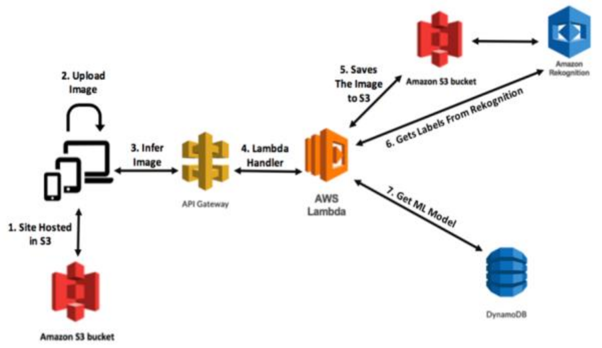

Figure 4: Flow To Infer An Image

3. Implementation Details

While Section 2 describes the architecture we use to solve the main objectives from Section 1, this section focuses on the actual implementation of this architecture and how it integrates with the frontend.

The frontend website for this project is hosted in a publicly accessible S3 bucket. This bucket is seperate from the bucket where user's images are stored. The frontend and backend are linked through an API hosted using the API Gateway. The API Gateway validates that requests come from an authenticated source and are formatted correctly. If so, the requests are passed to Lambda functions where they are processed accordingly.

3.1 Frontend Implementation

The frontend of our application is written in HTML, CSS, and JavaScript. There are three HTML pages for the index, login, and privacy policy pages and CSS files for the index and login. Though we will not go into the details of the HTML or CSS files, we use them to create a nice user experience. JavaScript is used to implement the different functions required by the application. The following subsections describe the JavaScript execution on the login page and homepage.

3.1.1 Login Page

The login page allows users to log into our site or create an account. Users can login with Facebook, Amazon or our site-specific login. To integrate with Facebook and Amazon, we had to register as a developer on each of their respective sites and register our application as a client application. We provided the redirect urls, privacy policy urls, etc. in order to set up the client to enable a successful integration.

When a user clicks a login button, a JavaScript function handles the redirection to the proper page and then another function processes the response when users are redirected back to our site. At this point users are given a token which is exchanged for temporary credentials as explained in Section 2.1. These credentials are then stored using session storage so the page can be refreshed or left and returned to without having to log back in. Once the credentials are stored, users are redirected to the homepage.

After about an hour, these credentials expire. Whenever a user opens the site, regardless of the page, it is first checked to see whether the user has valid credentials. If so, they are redirected to the homepage. If not, they are redirected to the login page.

3.1.2 Homepage

When the homepage first loads, session storage is checked for stored credentials. If they are found, the site makes a request to the API to verify the credentials. If it returns successful, another request is made to get the status of the user. The status includes whether the user has valid Instagram credentials connected to their account, whether the user has a model trained, and whether there is a model currently being trained. Using the status, the JavaScript displays the right buttons to the user.

The homepage has 3 main action buttons and two display sections. The action buttons are for selecting and analyzing an image, training a model, and connecting or disconnecting an Instagram account. If there is no trained model and the user is not connected to Instagram, all that appears is the connect to Instagram button. If the user is connected to Instagram, the train model button appears. Regardless of whether a user is connected to Instagram or not, if they have a model trained, the select and analyze image button is shown.

Each of these buttons trigger the three flows described in Sections 2.2, 2.3, and 2.4. When the buttons are clicked, a request is made using the API SDK provided for the API by the API Gateway.

When a model is currently being changed, there are two changes to the view. First, the train model button becomes a cancel model button. If this button is pressed, a request is made to the API which updates the user in the database to implement the cancellation. The second change to the view is that while the model is being trained, another subroutine is running in the frontend JavaScript to request updates on the model's progress. The subroutine makes requests every half second to the check status API endpoint. It receives whether the model is still in training and what fraction of the images have been processed. If the model is done training it returns the train model button back from cancel model to train model. Otherwise, the train/cancel model button acts as a status bar and displays the progress of the training.

There are two sections on the frontend that are only for display. One of them is always shown and provides details on how to use the site. The other, is only shown if the user is analyzing an image. This section shows the user the image being analyzed and the predicted response.

One last features of the homepage is the logout button. The user is logged out if they click this button or if their cognito credentials expire. The JavaScript that implements the logout first interacts with any third party service as required and then clears local credential variables and session storage and returns the user to the login page.

3.2 Backend Implementation

The backend is implemented using a variety of AWS services. This section will touch on each one used in this project and explain a bit about how it is used.

3.2.1 Simple Storage Service (S3)

S3 is used to host our website and store user's images. We have two buckets, one for each of these purposes. The one our website is hosted from is publicly readable while the other is only accessible using our internal service role (explained in Section 3.2.3).

The bucket with user's images has a folder for each user. Within each user's folder is a subfolder for new images they upload directly to the site and for those downloaded from Instagram and integrated into the model.

3.2.2 Cognitio

We use Cognito for logging in directly to our site and for distributing temporary credentials. We have a Cognito User Pool set up for our site which has our website as an app client. The user pool and app client are what allow users to login using our site-specific login. We also have a Cognito Federated Identities service set up for our website. This service is what actually distributes the temporary credentials that allow users to access the API. Our federated identities service is linked with our user pool and Facebook and Amazon app clients to allow users logging in with any login source to receive the proper credentials.

3.2.3 Identity Access Management (IAM)

IAM roles facilitate access to different AWS services. Our service uses two distinct roles. One role is attached to the credentials provided by Cognito Federated Identities. This role only allows access to the API and nothing more.

The other IAM role is an internal service role. All the internal services use this role and it allows them to interact with each other securely. Since this role is different from the one users are provided, user's ability to interact with the backend is limited to what the API offers which is a key security feature.

3.2.4 API Gateway

The API Gateway is used as a proxy between requests made from the frontend and the backend services. It validates that requests have the proper IAM role and then calls the backend Lambda function handlers using the internal service role.

Our API has the following endpoints:

- */insta/delete* for disconnecting user accounts from their Instagram accounts.
- */insta/post* for authenticating with Instagram and checking Instagram credential validity.
- */insta/put* for starting the process to train a model.
- /*model/build/post* for rebuilding a model with already stored data (not used from the frontend).
- */model/cancel/post* for canceling a model currently being trained.
- */model/get-status/post* for getting the current status of a model in training.
- */model/infer/post* for inferring an image using a stored model.

All of these endpoints are called from the frontend except for */model/build/post*. This endpoint is not explicitly called by the frontend because it is called implicitly after the model has started training from the */insta/put* endpoint.

3.2.5 Lambda

This project uses eight different Lambda functions to implement the different execution flows. They perform the following functions:

Insta: This function handles the */insta/post* endpoint. It connects user accounts to Instagram and updates the database as required. It also validates that Instagram credentials are still valid.

DisconnectInstagram: This function handles the */insta/delete* endpoint. It disconnects user accounts from Instagram and deletes the Instagram credentials from the database.

Refresh: This function handles the */insta/put* endpoint and starts the training of a model. It refreshes the user's model based on any new images it can pull from Instagram. If new images are found, the **ProcessImage** function is called to process them.

ProcessImage: This function is invoked asynchronously from the **Refresh** function. It processes individual images and updates the database as required. If it is processing the last image that needs to be processed it calls the **BuildML** function.

BuildML: This function is invoked asynchronously by the **ProcessImage** function and handles the */model/build/post* endpoint. It builds the model based on the image data that has been prepared by the **ProcessImage** functions.

CancelModel: This function handles the */model/cancel/post* endpoint and updates the database to cancel a model in training.

GetStatus: This function handles the */model/get-status/post* endpoint and gets the current status of a model in training from the database and returns it to the caller.

Inference: This function saves and analyzes an image and uses the model stored in the database to predict likes and comments. This function handles the */model/infer/post* endpoint.

Together, these Lambda functions are the execution backbone of the project. They integrate all the components and handle requests made to the API endpoints.

3.2.6 DynamoDB

DynamoDB is the database service used to store user data. We have a table that each user is given a row in. Users have attributes for their unique ids, Instagram credentials, Instagram image data, training status data, and model data. The table is updated by the different Lambda functions.

3.2.7 Rekognition

AWS Rekognition is the service we use to get an image's contents. Rekognition returns a list of labels associated with an image which are used when building our model and inferring the number of likes and comments an image will get. Rekognition analyzes images from S3 and is called from the **ProcessImage** and **Inference** Lambda functions.

3.3 Machine Learning Model Details

The machine learning model we use is based on a simple linear regression. When the model is trained, it generates label coefficients for likes and comments. In the basic model we use, the coefficients are the average number of likes or comments of images with that label. Inference can be described by:

$$likes = (1/N) \cdot (\gamma_1 + \gamma_2 + \ldots + \gamma_{n-1} + \gamma_n)$$
$$comments = (1/N) \cdot (\alpha_1 + \alpha_2 + \ldots + \alpha_{n-1} + \alpha_n)$$

Where N is the number of labels in the image being inferred, γ_i is the "likes" coefficient for a label in the new image, and α_i is the "comments" coefficient for a label in the new image. These coefficient values are calculated like so:

$$\gamma_i = (1/M) \cdot (L_1 + L_2 + \ldots + L_{n-1} + L_n)$$
$$\alpha_i = (1/M) \cdot (C_1 + C_2 + \ldots + C_{n-1} + C_n)$$

Where M is the number of images a user has posted to Instagram with a specific label, $L_i$is the number of likes each of the images with the label have, and C_i is the number of comments each of the images with that label have.

In the database, we save a mapping of the string value of the label to the likes and comments coefficients (γ_i and α_i) for that label. When new images are sent through Rekognition, the returned labels are used to retrieve these coefficients from the model and generate the estimated number of likes and comments.

4. Results

Overall, our project works quite smoothly. The user interface is intuitive and it is easy for users to login, connect their Instagram accounts, train a model, and infer results. The parallelism provided by AWS Lambda makes training time extremely fast while the use of DynamoDB and S3 work great to store user data and images.

In our testing, we have successfully been able to make reasonable predictions for images that have not yet been posted to Instagram. When uploading test images that are similar to ones on our Instagram account, we found that they were successfully predicted to have similar like and comment numbers.

One downside to our service is that our Instagram client application is in sandbox mode. Until it gets approved by Instagram (not likely for the sake of this project), we are unable to download and train on more than 20 images. For this reason, we tried testing our service on the 21st image. Unfortunately the results were not very accurate but for a good reason. The most recent 16 images on our test account have the same magnitude of likes and are from when the account was used actively. The older images were infrequently posted and all have significantly fewer likes than the most recent 16. As a result, we cannot expect the model to perform well on those older images.

To round out our testing, we also tried analyzing a few of the images that our model was trained on. As expected, two of the ones we tested were within 5 “likes” and 2 “comments” while another was not quite as good (30 “likes” off but perfect on comments).

From these results, it is clear that the machine learning model may not be the best. As a project, however, we believe the service works as it is intended to. It facilitates communication between the frontend and backend and effectively utilizes many AWS services. Further, because user’s data is kept in the database and S3, future updates would be able to improve the machine learning model. Improvements to the model could take into account parameters like time of day posted, day of the week posted, recency of posts, etc. The type of model used could also be updated. Convolutional Neural Networks, for example, could provide a better way than Rekognition and linear regression to analyze images.

5. Conclusion

As a whole, this project met its goals. By using a variety of AWS services, we have been able to develop a service that integrates with user’s Instagram accounts and provides a prediction of what the response of a new image may be. Though our machine learning model is not the most advanced, the backend infrastructure is in place to make future improvements. In the future, we would not be surprised to see services similar to this being used by individuals and corporate account managers.

6. Resources

Site Link: https://s3.us-east-1.amazonaws.com/image-analysis-project/login.html

GitHub: https://github.com/saribe0/COMS-E6998-Instagram-Predictor

YouTube Video:
https://www.youtube.com/watch?v=hYAPURgjEL4&feature=youtu.be

7. References

[1]https://github.com/saribe0/COMS-E6998-Instagram-Predictor

[2] “Instagram Monthly Active Users 2017 | Statistic.” Statista, www.statista.com/statistics/253577/number-of-monthly-active-instagram-users/.

[3] “Top Level Namespace.” Amazon, Amazon, docs.aws.amazon.com/AWSJavaScriptSDK/latest/top-level-namespace.html.

8. Appendices

Appendix A: Code Documentation

This project's git repository [1] contains the website source code, lambda function source code, api description, and documentation. The following four sections explain each of these in more depth.

Website Source Code

The website source code for this project can be found at `./root/`. In this directory, there are three html files for the different web pages:

```
./root/index.html
./root/login.html
./root/terms_privacy.html
```

The homepage is described by index.html, the login page is described by login.html and the terms of use and privacy policy page is described by terms_privacy.html. Each of these files uses the contents of `./root/css/`, `./root/js/`, and `./root/media/`. The media folder contains the refresh icon which is used when transitioning pages or something is loading. The js folder contains all the JavaScript and the css folder contains all the style files. Specifically, the css folder contains:

```
./root/css/login.css
./root/css/style.css
```

The login.css file is the styling used by the login page while style.css is the styling used by the home page. The privacy policy page uses the styling in style.css as well.

The js folder contains all the JavaScript for the project. It contains the following:

```
./root/js/amazon.js
./root/js/facebook.js
./root/js/google.js
./root/js/imageanalytics.js
./root/js/index.js
./root/js/login.js
./root/js/sdk/
```

The first 4 files (amazon.js, facebook.js, google.js, and imageanalytics.js) contain all the JavaScript required to implement logging in with each of the associated services. We originally allowed users to login with Google. However, due to issues with our site being hosted in S3, we were unable to implement that login at the same url as the other login services. As a result, we have removed that capability. The index.js file contains the JavaScript for the home page (index.html) and contains most of the front end logic. The login.js file contains the JavaScript for the login page. It integrates with the first four files to implement logging users in. The sdk folder contains the AWS JavaScript SDK [3] and the API SDK. Both of these were downloaded from AWS and are used to integrate the frontend with the backend.

Lambda Function Source Code

The backend execution for this project is facilitated by 8 AWS Lambda functions. These functions handle requests made to the API and interact with each other in order to implement the website's features. These 8 functions can be found at:

```
./lambda/Insta.js
./lambda/Refresh.js
./lambda/ProcessImage.js
./lambda/BuildML.js
./lambda/GetStatus.js
./lambda/Inference.js
./lambda/CancelModel.js
./lambda/DisconnectInstagram.js
```

For more information on these functions and how they interact with the backend, please see Section 3.2.5.

API

The api description for this project can be found at `./api/InstaAnalysis.yaml`. The description is in swagger format and exported from AWS API Gateway. More on the API can be found in Section 3.2.4.

Documentation

This report is also included in the repository and can be found at `./docs/report.pdf.`

Appendix B: Full Size Figures

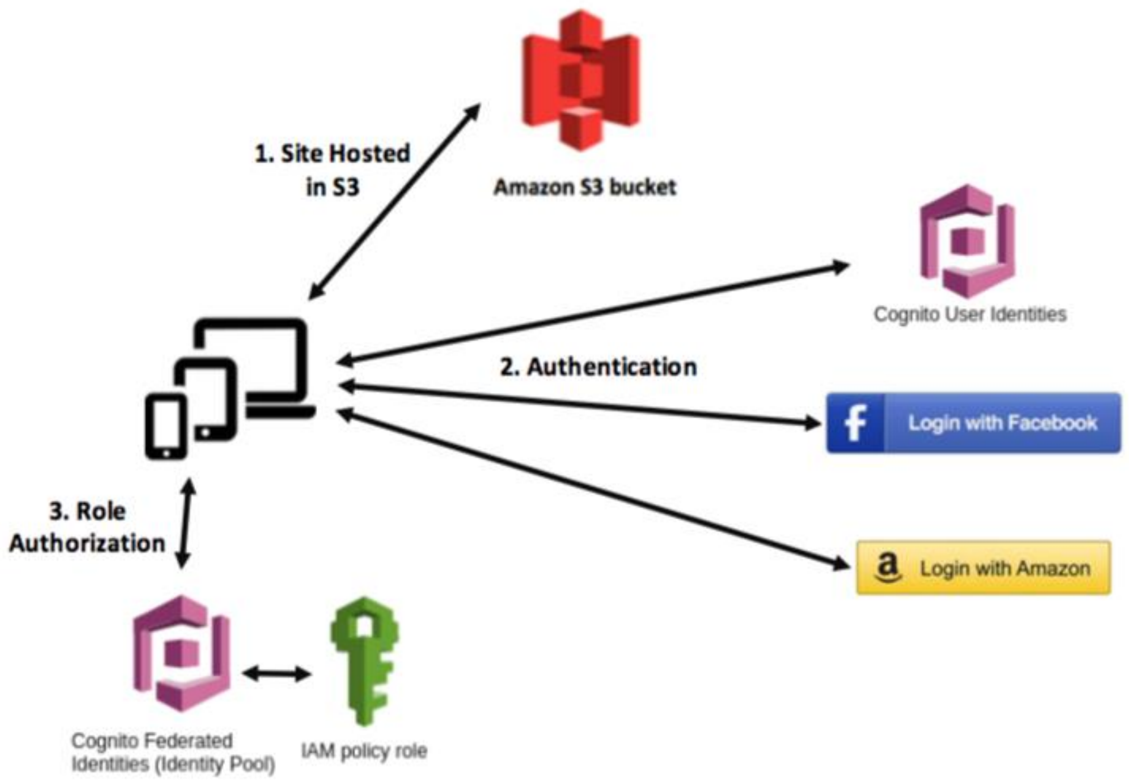

Figure 5: Full Size Authentication Flow

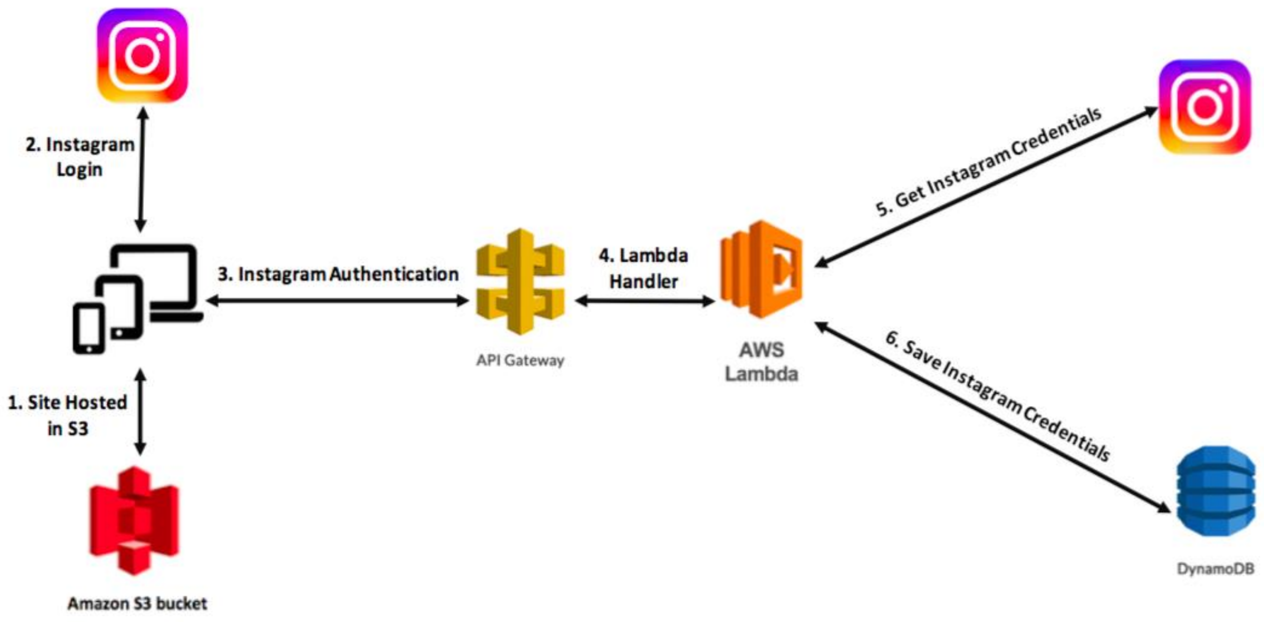

Figure 6: Full Size Connect Account Flow

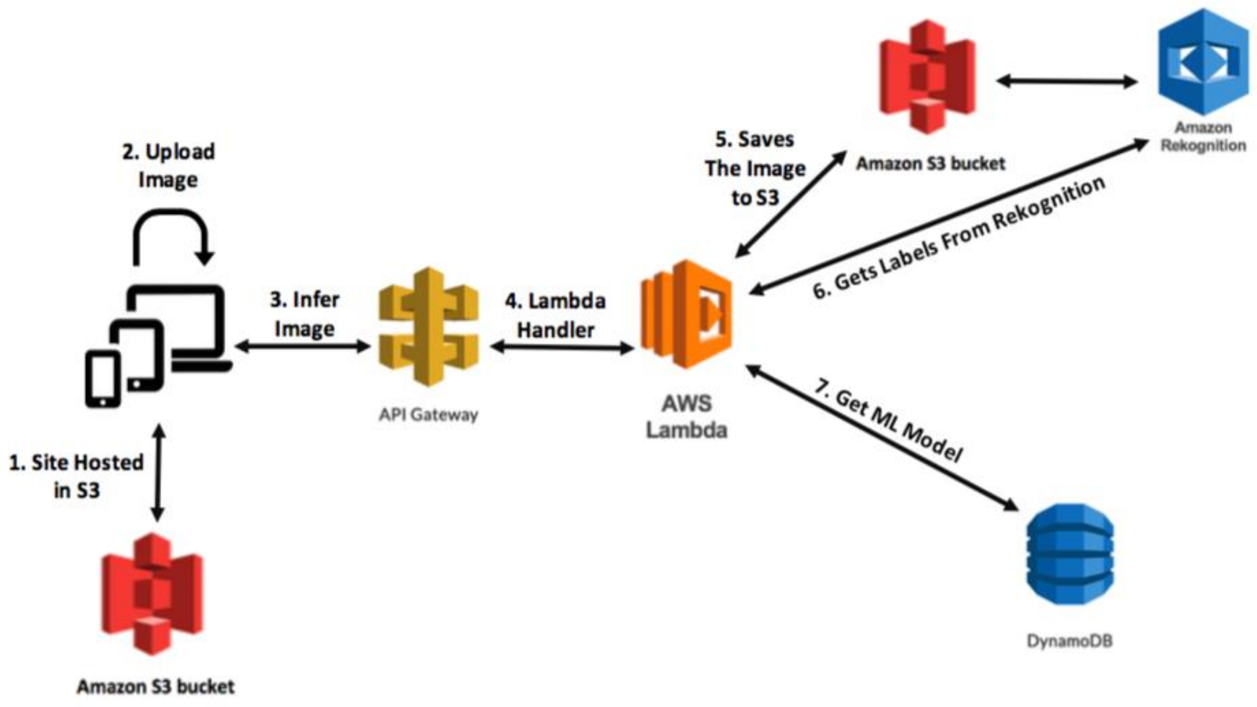

Figure 7: Full Size Flow To Infer An Image