Get Your Entertainment Recommendations Based on Your Selfie

Xiangzhuo Ding, Qi Wang, and Shahen Mirzoyan
Columbia University
116th and Broadway, New York, NY 10027
xd2212@columbia.edu, qw2261@columbia.edu, sm4775@columbia.edu

Abstract

In this project, we built a web application that personalizes media recommendations using only facial sentiment data. Recommending systems typically rely on implicit feedback from user-item interactions for the purposes of developing a preference profile for users. Our system approaches this problem in a novel and interesting way. Our data servers run both locally and on GCP with high stability and speed. With more user data collected, the system will become smarter and generate better recommendations.

1. Introduction

We live in the age of information explosion. Tech giants like Google and Amazon rely heavily on the vast amount of user data they possess to optimize the user experience and recommend the most relevant content[8]. Luckily, with the development of big data tools and artificial intelligence, we have increasingly robust and accurate methods to help personalize a user's app experience.

Traditional approaches implementing content recommendation systems, such as collaborative filtering, often rely on the user's history ratings, which typically comes explicitly from the user's item ratings, or implicitly from their browsing frequency[2]. Potential issues exist in current recommendation systems. For instance, the system needs a large amount of accumulated history data to make a reliable prediction, which means it might not work well for new users. Another potential issue is that the system can't distinguish real users apart from the account name, i.e., one can't get the correct recommendation when using a borrowed phone from friends. Furthermore, those systems didn't use any other information from users other than browsing history, which is less robust.

To address these issues, our objective is to design an end-to-end system that can identify different users, obtain their facial information, learn their interests, and make recommendations. To achieve that, at least three major subsystems need to be established. We first built a facial recognition and analysis system. This system takes images as input and combines the results of several Convolution Neural Networks. The second is a recommendation system, which can analyze the result of the first system and produce a corresponding recommendation. Last we have a web server system, which works as a framework to provide the user interface and send commands to other subsystems.

2. Related Work

Although many other works try to solve the problem of recommending relevant media content to users, we are unaware of any applications or attempt to use user's facial information to generate recommendations. Fortunately, the idea of using Convolution Neural Networks to recognize and analyze faces has been raised years ago, and many useful tools are now available[6, 11, 5]. Some related works are MTCNN[10], InceptionResnetV1[7], and densenet121[4]. Those models are well known and widely used in different areas. Here, we use them to analyze the facial feature of users and extract important information for the recommendation. Another paper we brought ideas from is *A Compact Embedding for Facial Expression Similarity*[9]. Instead of focusing on discrete emotion recognition, like pre-defined semantic categories, Agarwala *et al.* describe facial expressions in a continuous fashion using a compact embedding space that mimics human visual preferences.

The approach used to generate recommendations for this application was first developed in 2008 by Hu *et al.*[2]. The authors' approach to the problem of generating recommendations for items that a user has not explicitly rated was to introduce a degree-of-certainty term into the loss function, which is proportional to the "implicit" signal that they have a preference for a given item.

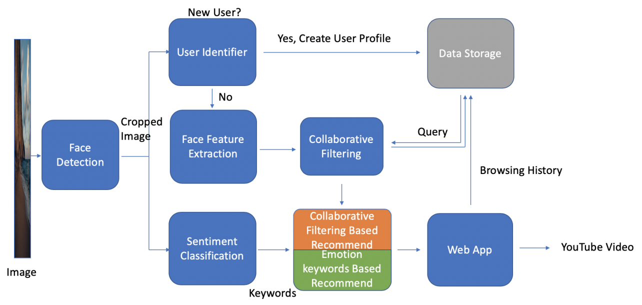


Figure 1. System Design

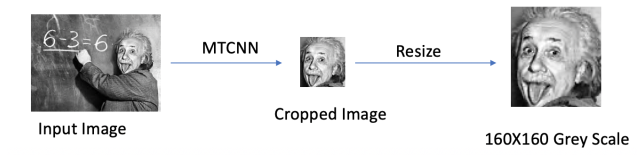


Figure 2. Face Detection

3. System Overview

The whole system is shown in Fig. 1. The first four modules: Face Detection, User Identifier, Face Feature Extraction, and Sentiment Classification, are all CNN based Deep Neural Networks but each with different architecture and different outputs. The collaborative filtering model for generating recommendations is based on the ALS algorithm for matrix factorization.

The neural networks were loaded and trained using OpenCV and PyTorch with GPU-acceleration on Google Cloud Platform. Our scripts for scraping, loading, and training our data were written in Python. The front and back-end of our web app was implemented in Django, and the collaborative filtering is done using Spark.

Face Detection The first step for a user to get a recommendation is to take a selfie using our web application. After preprocessing, the raw image is pass to Face Detection Module, which is a pre-trained MTCNN[10]. Next, the Face Detection Module produces bounding boxes correlated to the faces detected in the images. After that, the system picks up the face with the largest bounding box and crop the face out. The face image then be resized to 160×160 and converted to a grayscale image, before being fed to the next module.

Facial Analysis Once a cropped image is received, the User Identifier and the Sentiment Classification module start working simultaneously. Here, the Sentiment Classification module can estimate the emotion of the current user and generate a keyword based on the emotion. On the other hand, the User Identifier can extract a 512-dimension feature vector based on facial image and compare it to the previously stored faces. If the user is identified as a new user, then a new user profile will be created to store his/her facial features and browsing history. In the other case, if the user is a returning user, the fourth module will be activated to generate a 16-dimension feature vector. Unlike the previous 512-dimension vector, which is used to identify the user, the 16-dimension vector is used for evaluating the user's reaction to the current playing video. More details about the rating are discussed in section 5.

User ID	User Face Feature
0	[0.435, 0.332, . . .]
1	[0.221, 0.566, . . .]
...	...

Table 1. User ID Storage

User ID	Item ID	Rating
0	1	2
1	1	1
...	...	...

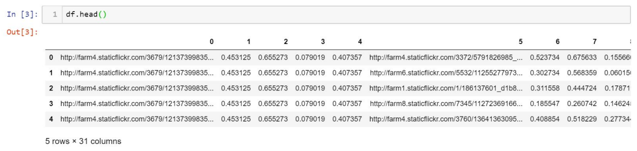
Table 2. User Rating Storage

Item ID	URL
0	"Rydi3AqMgcw"
1	"3LvWTmqn7A4"
...	...

Table 3. Item ID and URL

Recommendation Generation The final recommendation is composed of two parts. The first part is Emotion Keywords Based Recommend. The keyword is acquired from the Sentiment Classification module, and it is used to generate a YouTube URL link. In this way, we can generate recommendations for any user, no matter he/she is a new or returning user. The second part is Collaborative Filtering, which generates recommendations using the user's previous data. One thing to note is that this step will be skipped if the user is a new user, and all recommendation content will be keyword based.

Data Storage With Spark, the web server maintains a database to store all the user data. Inside the database, unique user ids and its correspondence feature vectors are saved here, which will be updated once a new user registers. Moreover, unique item ids, which represent YouTube URL, and user's ratings are also preserved here. Table 1-3 are examples of the database.



```
In [3]:  1 df.head()
Out[3]:
                                                0         1         2         3         4                                                5         6         7
0  http://farm4.staticflickr.com/3679/12137399835...  0.453125  0.655273  0.079019  0.407357  http://farm4.staticflickr.com/3372/5791826985_...  0.523734  0.675633  0.15566
1  http://farm4.staticflickr.com/3679/12137399835...  0.453125  0.655273  0.079019  0.407357  http://farm6.staticflickr.com/5532/11255277973...  0.302734  0.568359  0.06015
2  http://farm4.staticflickr.com/3679/12137399835...  0.453125  0.655273  0.079019  0.407357  http://farm1.staticflickr.com/1/186137601_d1b8...  0.311558  0.444724  0.17871
3  http://farm4.staticflickr.com/3679/12137399835...  0.453125  0.655273  0.079019  0.407357  http://farm8.staticflickr.com/7345/11272369166...  0.185547  0.260742  0.14624
4  http://farm4.staticflickr.com/3679/12137399835...  0.453125  0.655273  0.079019  0.407357  http://farm4.staticflickr.com/3760/13641363095...  0.408854  0.518229  0.27734

5 rows × 31 columns
```

Figure 3. FEC Dataset

4. Data

Two separate datasets were used for this project. For emotion classification, we used a dataset from the Kaggle competition: Emotion and Identity Detection from Face Images. It contains 35903 48×48 greyscale images with human classified labels.

The dataset that was used to train the embedding network is the same dataset that was used by the original authors of the FEC network[9]. The dataset can be found here https://research.google/tools/datasets/google-facial-expression/.

A preview of the dataset is shown in Fig. 3. There are 31 different fields including 3 web urls containing the face images for each triple, along with 4 coordinates for each triple that define the bounding box of the face to be analyzed. Moreover, each triple was classified into 3 mutually exclusive categories by at least 6 human annotators according to similarity of facial expression (a rating of 1 means that expressions 2 and 3 are most similar, etc.). We scraped these images slowly over the course of 48 hours and our dataset before preprocessing was over 6 GB in size. We should note that since the time the original paper was published, about 10% of the urls were no longer active in the training set. Moreover, our test set was approximately 40% smaller than that of the original authors.

5. Methods

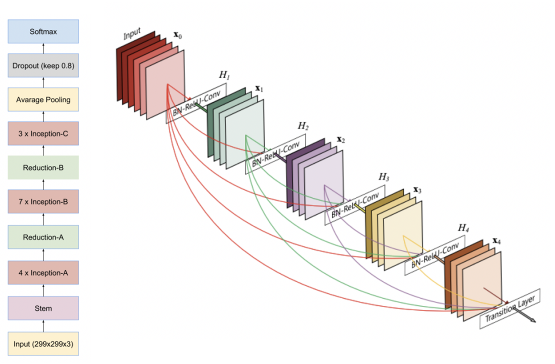
The pipeline of our project takes a raw image from the webcam or phone camera of a user, detects the faces in it, analyzes facial information and generates recommendations based on a collaborative filtering model that runs on the back-end.

5.1. Neural Networks

5.1.1 Architecture

Torchvision provides many neural network architectures that perform well in many tasks. Among them, we used InceptionResnetV1[7] and densenet121[4] to train the Emotion Classifier and Face Feature Extraction Model.

To acquire the desired output shape, we change the last few layers of InceptionResnetV1 model. The output dimension of the Average pooling layer is 1792. After Dropout,



A. InceptionResnet V1 B. Densenet 121

Figure 4. Network Architectures

we added two linear layers to reduce the dimension to 512 and then to 4.

5.1.2 Training

To train models with such vast numbers of parameters and large datasets, Fine-tuning adapted to reduce the time for training. We also created a VM instance on GCP. Following is the environment settings for the training:

4 core CPU

Tesla k4 GPU

7.5 GB Memory

PyTorch - running on parallel computing

Face Detection Model and Face Identification Model are pre-trained[1].

5.2. Recommendation System

5.2.1 Collaborative Filtering

Traditional collaborative filtering methods rely on an implicit rating for every item a user has consumed. Some signals that are typically used to generate ratings include the number of times a user has clicked an item, how long a user watches a video or listens to a song, etc. The aim of our application is to generate recommendations for users that rely exclusively on implicit feedback received from their facial expressions. By running images of a user's facial expression through our neural network, we obtain a 16-dimensional vector representation of the user's expression, which we then transform into an implicit rating for the item. This transformation relies on the fact that the embedding preserves similarity, in the sense that similar expressions get mapped to nearby points in the embedding space. Examples are shown in Fig. 5

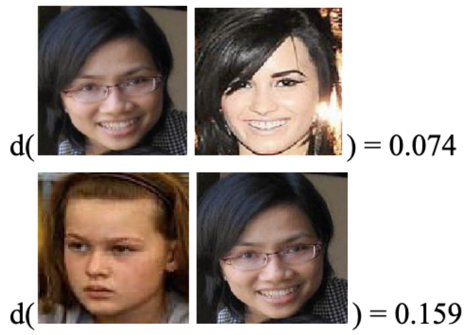


Figure 5. Expression Similarity

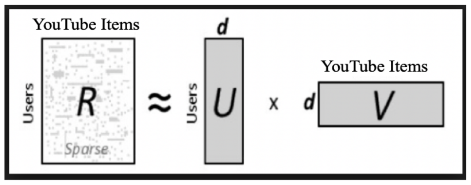


Figure 6. ALS Algorithm

With this assumption, we took a small sample of images which we consider to be representative of expressions that are happy, interested, or engaged from our training set and defined a user's rating of an item to be:

$$f(user) = \begin{cases} 0 & \min_i(distance(user, sample_i)) > \tau \\ 1 & \min_i(distance(user, sample_i)) \leq \tau \end{cases}$$

where τ is a hyperparameter of our model, experimentally derived to be 0.9. To summarize, a user's preference for an item is updated with a +1 if the user's facial expression embedding falls within a threshold of any other expression we've chosen in our sample. Otherwise, a user's interest in an item is assumed to be 0.

After obtaining implicit ratings through user-item interactions, we generate new rating predictions for items a user has not seen using the Alternating Least Squares (ALS) algorithm implemented in Spark. The algorithm works by transforming the ratings into a sparse matrix representation and factoring this matrix into its singular value decomposition (Fig. 6).

The (i,j)-th entry in the sparse ratings matrix is given by user i's rating of item j. In order to account for the degree of uncertainty that comes with generating implicit recommendations, our loss function will need to take into account how many times a user has rated an item favorably in the past. With this aim, the model defines two new terms, p_{ij} and c_{ij} as follows:

$$p_{ij} = \begin{cases} 0 & r_{ij} = 0 \\ 1 & r_{ij} > 0 \end{cases}$$

$$cij = 1 + \alpha * r_{ij}$$

where r_{ij} is the implicit rating by user i of item j. The term c_{ij} represents the degree of confidence in the implicit rating, and is proportional to the number of times a user has favorably rated an item, and the term α is a hyperparameter. The loss function we seek to minimize is the sum of squared errors of known ratings in the training set, weighted by our degree of confidence in the rating:

$$L(x,y) = \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i^2) + \lambda(\sum_u |x_u|^2 + \sum_i |y_i|^2)$$

Our collaborative filtering function is written in Spark, and runs on GCP. We feed user and item data periodically to our database in BigQuery, and run collaborative filtering on user-item ratings. Once recommendations have been generated for all of our users, we update our BigQuery database with these recommendations, and later query this database to generate recommendations for our returning users.

5.2.2 The Cold Start Problem

How do we generate recommendations for users who are new and who our model does not yet recognize or have ratings for? This is a general problem in the field of recommender systems known as the Cold Start Problem. We approach this problem using our second neural network, which is trained to identify discrete emotions, such as happy, neutral, or sad. The output of this network is used to generate keywords for YouTube's API, and the recommendations shown to a cold-start user consist entirely of YouTube queries based on this search result. For instance, a new user who appears to be in a happy mood will typically be recommended upbeat music content. This approach allows the system to recommend novel items to users before it learns what their individual preferences are.

5.3. Web Server

5.3.1 Layout Design

The web application of the project contains 5 parts: YouTube Video Frame, Webcam Frame, Selfie Frame, Recommendation Frame, and Functional Elements Frame (Fig. 7(A)). When a user enters the main page, "getRecommend" page, all elements of the web application are loaded, and the server requests the permission of accessing webcam[3] (Fig. 8). The authorization of webcam use is a prerequisite

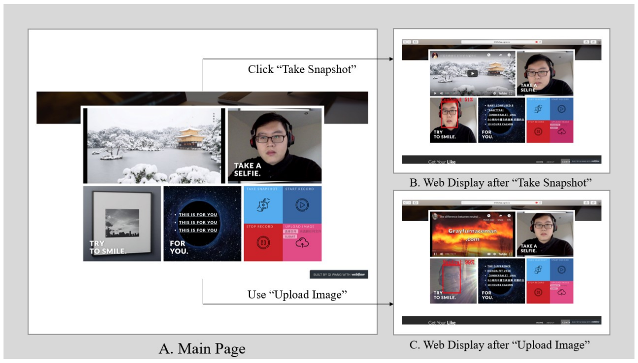
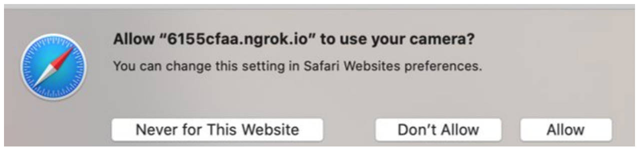

Click "Take Snapshot"
Use "Upload Image"
A. Main Page
B. Web Display after "Take Snapshot"
C. Web Display after "Upload Image"

Figure 7. Web Display



Allow "6155cfaa.ngrok.io" to use your camera?

You can change this setting in Safari Websites preferences.

Figure 8. Requesting the Permission of Accessing Webcam

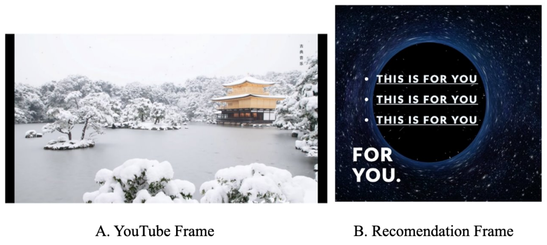


Figure 9. Default Frames

of the regular running of the web application.

When the web application maintains a regular status, the Webcam Frame shows the video recorded by the front-facing camera, which generally is the face of the user. In the meantime, the YouTube Video Frame, serving as an entertainment service to load YouTube videos of potential attraction to the user, will start a Chinese Guzheng music (Fig. 9 (A)) on snow scene playing by default. The Selfie Frame is empty with a photo frame as the background image and Recommendation Frame loads 3 links (Fig. 9 (B)), "This Is For You", as default.

The 4 Functional Elements, including "Take Snapshot", "Start Record", "Stop Record", and "Upload Image", enable the user to interact with the web application. When the user click "Take Snapshot", the JavaScript function uploads the current snapshot of the webcam to the back-end and demands the back-end to process the image and provide a recommendation list. After image processing with multiple trained models, including Face Detection model, Sentiment

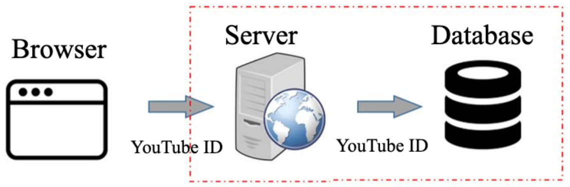


Figure 10. Method of Front-End Sending YouTube ID to Back-End

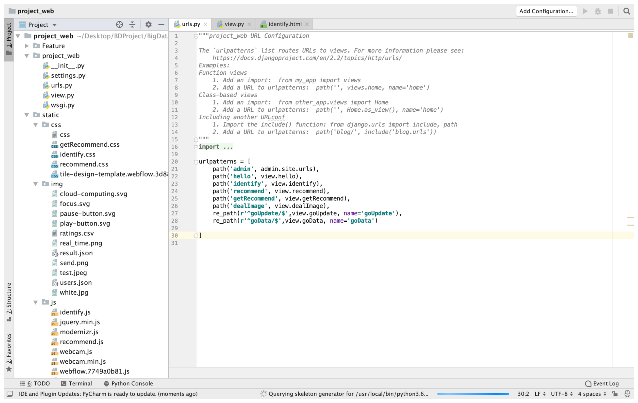
Classification model and User Identifier model, the web loads the selfie with emotion analysis in Selfie Frame and update Recommendation Frame by loading the item list, *e.g.*, (YouTube URL links) generated by Recommendation System (Fig. 7(B)). As for "Start Record" and "Stop Record", the "Start Record" aims to automate continuous clicks of "Take Snapshot", which frequently reloads selfies with emotion analysis to Selfie Frame and update recommendation list until the user clicks "Stop Record" to terminate the process. The paired functional elements well embody the user-friendly logic of web application design. Besides using the webcam, the web application allows the user to upload his/her images regardless of having access to a webcam. Selecting an image from local and click "submit" also uploads the image to the back-end through the "POST" request, and the back-end returns the image with emotion analysis and recommendation list to refresh the web frames (Fig. 7(C)).

Another design of great significance in the web application is the Recommendation Frame. Every time a new image with emotion analysis is presented in Selfie Frame, a "GET" request is sent to the back-end to obtain the recommendation list (5 YouTube items in the format of {Name, URL}) based on Recommendation System. Also, based on the thought that the clicks of a user on a link affect the renewal of the Recommendation System, the web will not only load the video into YouTube Frame but also send a "GET" request with "?YouTube_ID" as the tail to the back-end to inform the YouTube ID of the user's click (Fig. 10) to update and strengthen the Recommendation System.

5.3.2 Back-End Design

The back-end of the web application is built by Django and made public on the internet using Ngrok. There are four main parts of the back-end, including "getRecommend", "recommend", "goData" and "goUpdate" (Fig. 11).

The path "getRecommend" is the main page, as shown before, allowing the user to access the service from the



```
project_web ~/Desktop/BDProject/BigDat
  Feature
  project_web
    __init__.py
    settings.py
    urls.py
    view.py
    wsgi.py
  static
    css
      css
      getRecommend.css
      identify.css
      recommend.css
      tile-design-template.webflow.3d8
    img
      cloud-computing.svg
      focus.svg
      pause-button.svg
      play-button.svg
      ratings.csv
      real_time.png
      result.json
      send.png
      test.jpeg
      users.json
      white.jpg
    js
      identify.js
      jquery.min.js
      modernizr.js
      recommend.js
      webcam.js
      webcam.min.js
      webflow.7749a0b81.js
```

```
"""project_web URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/2.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
import ...

urlpatterns = [
    path('admin', admin.site.urls),
    path('hello', view.hello),
    path('identify', view.identify),
    path('recommend', view.recommend),
    path('getRecommend', view.getRecommend),
    path('dealImage', view.dealImage),
    re_path(r'^goUpdate/$',view.goUpdate, name='goUpdate'),
    re_path(r'^goData/$',view.goData, name='goData')

]
```

```
IDE and Plugin Updates: PyCharm is ready to update. (moments ago)    Querying skeleton generator for /usr/local/bin/python3.6...    30:2  LF  UTF-8  4 spaces
```

Figure 11. Back-End Design

browser. The “recommend” path is designed for image processing from webcam or image of submission, which identifies whether the request is “POST” and decodes the image file inside the request. After receiving the image, the path incurs the Face Detection model, Sentiment Classification model, and User Identification model to handle the image processing. Once completion, the “recommend” path saves the image with emotion analysis for the front-end to display on the Selfie Frame. Meanwhile, the front-end sends a “GET” request to path “goData” to call the Recommendation System to generate a recommend list in the format {Name, URL}, and reload the Recommendation Frame. As for the path “goUpdate”, once the YouTube link in Recommendation Frame is clicked, the video will be loaded to YouTube Frame and the id of clicked video will be sent to “goUpdate” as the format of “goUpdate?YouTube_ID”. After a series of simple decoding operations, the back-end acquires the YouTube ID of clicked video and the Recommendation System gains fresh improvement with the help of the ID information and reliable updating algorithm.

6. Experiment

6.1. Emotion Classification

The original dataset has seven categories, and the validation accuracy is 63%. One potential reason for low accuracy is that it is hard for the model to distinguish similar emotions such as anger and disgust. After merging similar emotions and reducing the number of categories to four, the final validation accuracy reached 81%.

6.2. Facial Expression Embedding

We tried various architectures for the Facial Expression Embedding Module. At first we tried to use the same architecture as the Emotion Classifier, since they share a similar purpose. However, the testing result only shows 59% accuracy. To resolve this issue, we turned to densenet12, which was proposed by Landola *et al.* We found the resulting accuracy with this architechture to be sufficient for our purposes.

6.3. Generating implicit ratings

The problem of inferring the level of a user’s content satisfaction based solely on their facial expression was a novel one, and we contemplated many different approaches throughout our project. Some considerations we had were using a translation model or RNN to find a mapping from the output of the embedding network to generate keywords which we would then feed into the YouTube API directly. This approach would require a lot of additional training and additional data which we did not have.

The other approach was to find a mapping from the 16 dimensional embedding to a scalar which represents a user’s implicit rating of the item. Although there are many different ways to construct this mapping, the most interpretable approach was to use similarity. We know what ”interested” and ”amused” expressions look like, and because our embedding preserves expression similarity, we can assume a user is enjoying content if their expression embedding is approximately close to the embedding of other facial images which are ”interested” or ”amused.” We manually tuned the threshold τ in our model until we felt that it was sufficiently good at distinguishing cues that a user is interested in content, and we also tried using different sample images to define our clusters.

7. Result

A demo of our project can be found here: https://youtu.be/Gku3ZdkLNOY. As shown in the video, once a snapshot is taken, the system can identify the user in a short time. And it starts to search the database and retrieve the user’s profile. Six recommended YouTube videos will be provided. Among them, the first three videos are generated by collaborative filtering, and the rest is generated by random keywords searching. The next run is a new user using the web app. In this case, the system labels the user as unknown and create a new user profile. Since there is no history rating data of this user, all six recommended videos are generated by random keywords searching. Meanwhile, the system keeps a log of the video that the user watched and update his/her profile for later recommending.

8. Conclusion

We designed an entertainment recommendation system, which has been proven to be able to help users find relevant videos. We used several big data analytic technologies to make the system fast and stable. Four separate Deep Neural Networks and a collaborative filtering model were trained and deployed to analyze user identities and preference profiles. We used BigQuery to generate and maintain a user database. The whole system is running based on a web server written by Django. Compared to other recommendation systems, our system has the advantage that it relies exclusively on implicit data and can generate novel recommendations for new users. The principle of the system dictates that it will perform even better when more users use our system and generate more user data.

9. Individual Contribution

9.1. Qi Wang

- Put forward the idea of the whole project.
- Built the back-end by Django to handle various requests.
- Merged trained models into the back-end to serve as functional support of web application.
- Wrote the front-end using HTML, CSS, and Javascript.
- Added YouTube API as additional support to the Recommendation System.
- Introduced Ngrok technique to make the web application public on the internet.
- Managed web branch of GitHub repository.

9.2. Xiangzhuo Ding

- Designed the framework of the project workspace.
- Managed the GitHub repository of the project, including version control, creating and merging branches.
- Literature view of related works and read source code, including Multi Task Convolution Neural Network, InceptionResNet, and DenseNet.
- Created a PyTorch environment with CUDA support.
- Designed an image processing pipeline for the project.
- Wrote modules for data loader and training script that can handle multiple GPU in parallel mode.
- Trained and evaluated several Neural Networks.
- Wrote different high-speed inference engines for different face analysis tasks.
- Merged inference engines into the web server backend.

9.3. Shahen Mirzoyan

- Discovered and reviewed the paper "A Compact Embedding for Facial Expression Similarity" (Vemulapalli & Agarwala, 2018) which was the main inspiration and reference for our project
- Scraped over 80,000 URLs to obtain the original training and test data used by the authors of the above paper
- Implemented the facial identification function in the backend, which recognizes a user's identity using the VGGFace2 model in PyTorch and the cosine distance metric, and stores the user's identity in a database
- Merged a BigQuery database into the backend for storing all user and item data in the cloud
- Literature review for implicit recommender systems and collaborative filtering. Particularly the main method used in this application, which is based on "Collaborative Filtering for Implicit Feedback Datasets" (Hu, Koren, Volinsky, 2008)
- Explored various models for interpreting facial expressions as item ratings
- Wrote backend functions to map facial expression inferences to an item preference profile for a user
- Wrote the collaborative filtering function in PySpark to recommend the top N items for every user, and update our BigQuery database with new recommendations periodically

References

[1] T. Esler. Face recognition using pytorch. `https://github.com/timesler/facenet-pytorch`, 2019.

[2] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. *IEEE International Conference on Data Mining*, 2008.

[3] J. Huckaby. Webcamjs. `https://github.com/jhuckaby/webcamjs`, 2012.

[4] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv*, 2014.

[5] O. M. Parkhi, A. Vedaldi, A. Zisserman, et al. Deep face recognition. In *bmvc*, volume 1, page 6, 2015.

[6] Y. Sun, X. Wang, and X. Tang. Hybrid deep learning for face verification. *IEEE International Conference on Computer Vision*, 2013.

[7] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *AAAI Conference on Artificial Intelligence*, 2017.

[8] W. Tan, M. B. Blake, I. Saleh, and S. Dustdar. Social-network-sourced big data analytics. *IEEE Internet Computing*, (5):62–69, 2013.

[9] R. Vemulapalli and A. Agarwala. A compact embedding for facial expression similarity. *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[10] J. Xiang and G. Zhu. Joint face detection and facial expression recognition with mtcnn. *International Conference on Information Science and Control Engineering*, 2017.

[11] X. Zhao, X. Shi, and S. Zhang. Facial expression recognition via deep learning. *IETE Technical Review*, 2015.