# CPEN 291 Project
# Final Report
# Team Spinosaurus
# PS4 Game Recommender App

**Group Members:**
**Aayush Behl, Harshil Rajesh Patel,**
**Nick Zhang, Kolton Luu**

## Table of Contents:

---

## Introduction:

Video games offer a unique way for individuals to simulate what cannot exist in reality. Over time, gamers began to explore a variety of different genres , such as action, adventure, role playing, shooter, sport and countless others. However, we all get bored of playing the same stuff over and over again and we want to push ourselves to explore new things we might like, but we also do not want to deviate too far from what we already enjoy.

We, as Team Spinosaurus, set out to develop an application that would allow users to input images of game covers that they enjoy and in return, receive other game titles that suit their tastes. We want to offer our users the convenience of searching for games that are specifically picked for their taste.

Our app is called the PS4 Game Recommender and is available for use on Android smartphones. It targets an audience with the interest in console gaming. The purpose of the application is to provide a platform to recommend PS4 games depending on a user's taste while taking into account the multiple factors that can influence their affiliation to any game - such as sequels to the title, other titles from the same developers, and the genre of the game. We aim to provide the users with a seamless experience as they search for games that are similar to the ones that they input. In doing so, we will make it more convenient for such an audience to spend less time searching, and more time playing the games.

In addition to its primary functionality, our team also prioritized the ease of operation and the processing time of the app in order to maximize convenience for our users. The following report will offer deeper insight into how the application works, the technical details regarding the Machine Learning and Non-Machine Learning components, the challenges we faced as a group, and our work as a team.

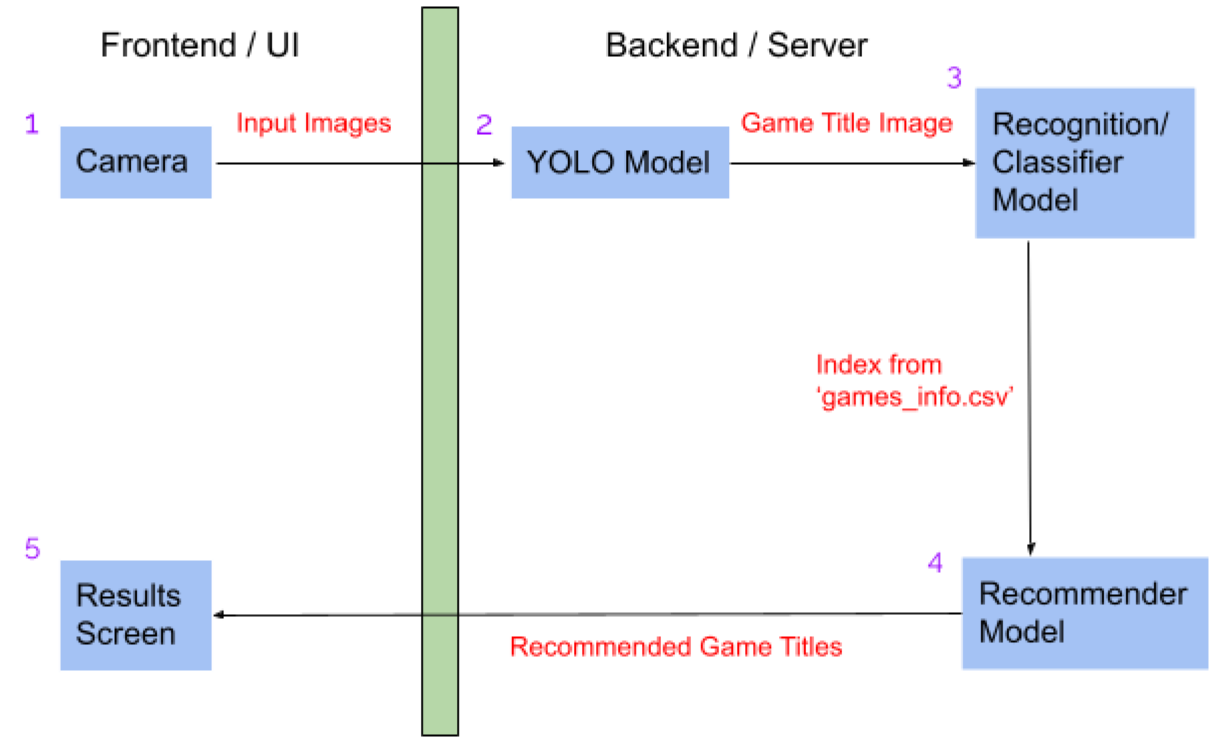
## Flow of Data:

This section offers an insight into the whole structure of our project and the form in which the data is passed into each component of our application. Firstly, there are 8 primary components in our project out of which 3 are datasets (game_info.csv, user_ratings.csv, titles.zip), 3 are ML models (YOLO, recognition/classifier, recommender), a frontend (App UI) and the backend (Node.js server).

The flow of data is as follows:

- The user input images are passed as inputs into the YOLO model through a network from the frontend. The YOLO model extracts only the title portion from the image and passes it into the classifier model. Then the classifier model outputs the prediction as the corresponding indices in the games_info.csv dataset.
- These indices are passed as inputs into the recommender model. The recommender model then computes the recommendations and outputs the titles of the recommended games. These titles are sent back to the frontend to display the results to the user.

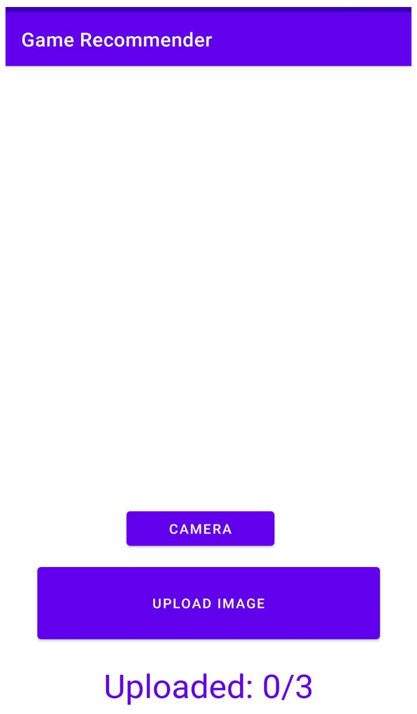
The following diagram summarizes the flow explained above:



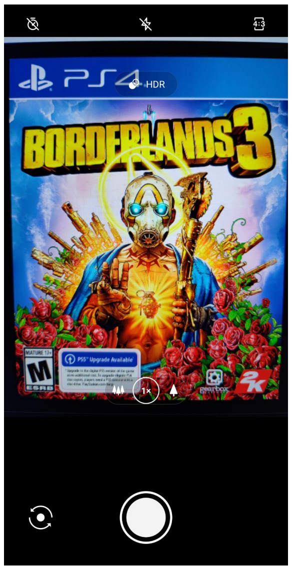
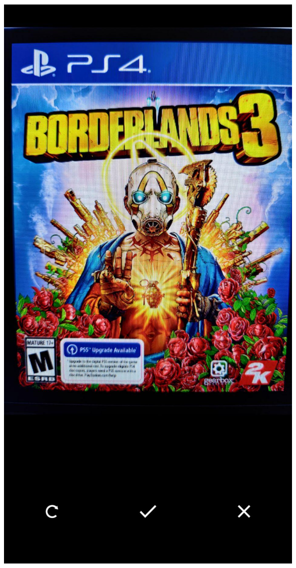
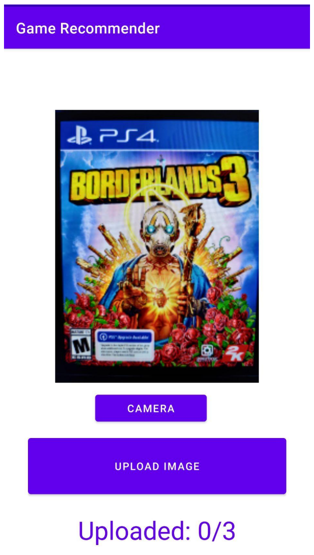
**Walkthrough Example of the Application:**

---

- When the app gets opened, it first shows the initial screen that prompts the user for taking an image with the phone's camera:

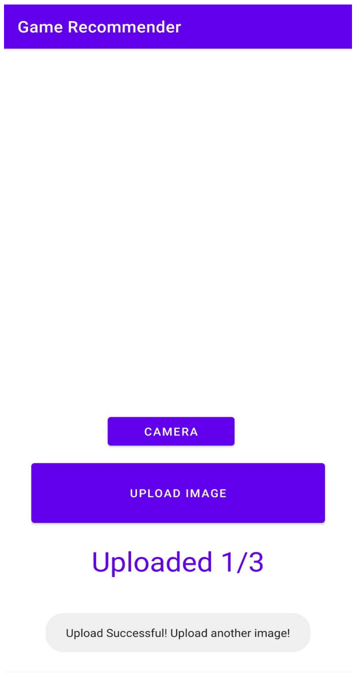


- By clicking on the “camera” button, the app will automatically start the default camera app on the phone and the user can take an image of the cover of a game. After confirming the image in the camera app, the game recommender app will also show the image on the screen for the user to preview:

Uploaded: 0/3

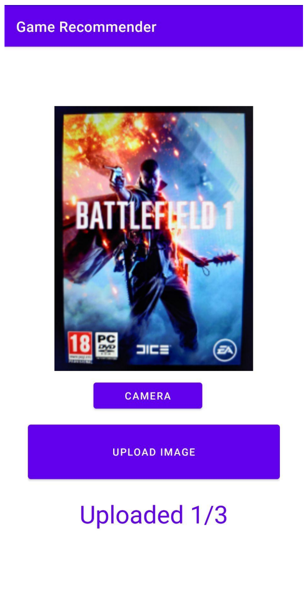
- If the user is satisfied with the current image, the next step is to click the “upload image” button so that the app uploads the image to the backend server. Otherwise the user can click the “camera” button again to retake the image.
- After the upload is finished, the app returns a message onto the screen telling the user that the image upload is successful and updates the number of uploaded images:

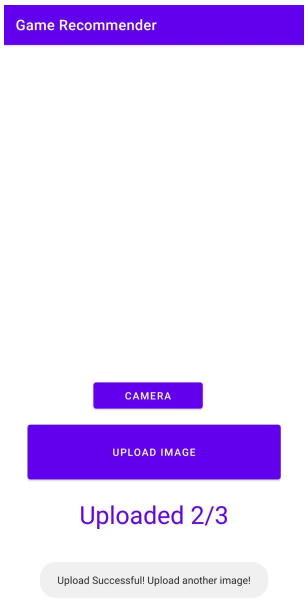


Uploaded 1/3

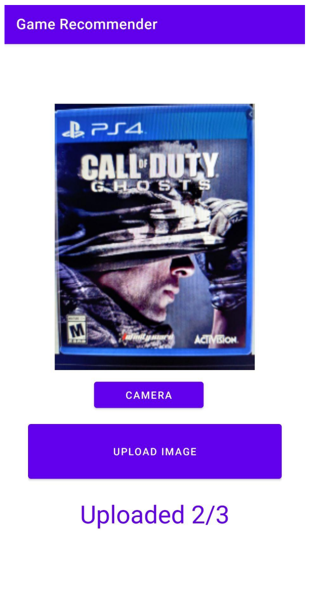
Upload Successful! Upload another image!

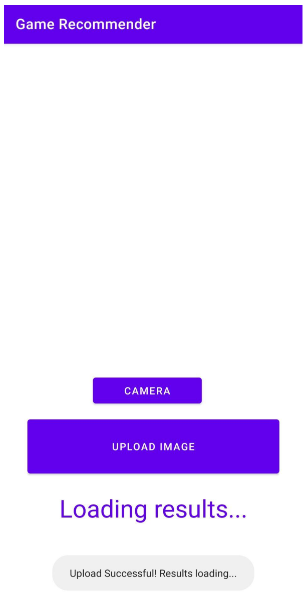
- Then, the user will just will follow the same steps above to upload two more images:





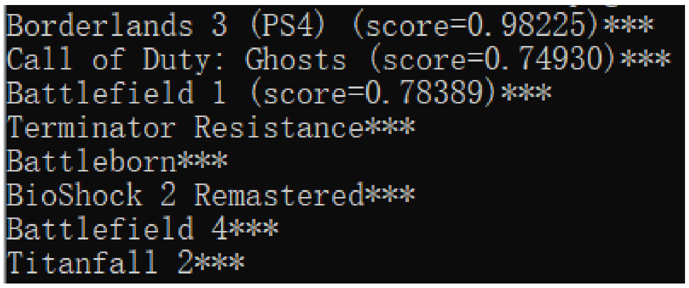
Upload Successful! Upload another image!





Upload Successful! Results loading...

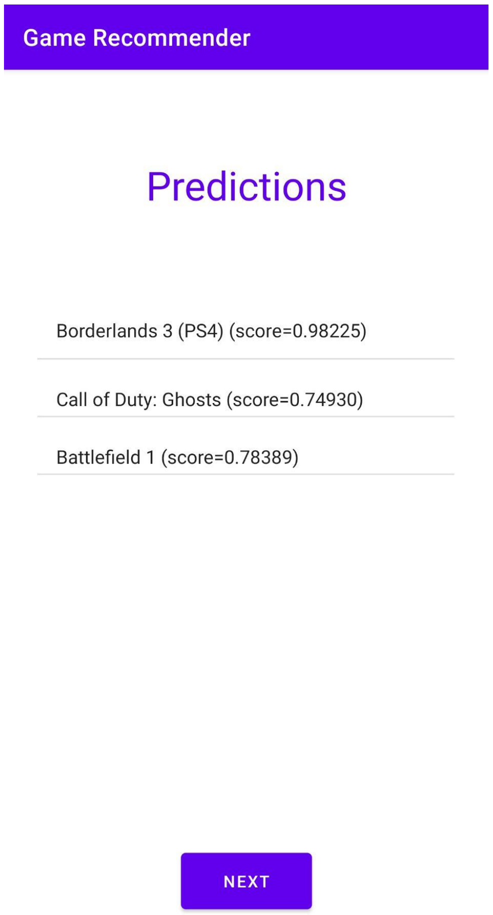
- After all three images are uploaded, the app will show "Loading results" and wait until it gets a response from the backend server. Following is the output of the script run in the backend. The first 3 lines are predictions for the uploaded images and the 5 lines left are the recommendation according to the input:



```
Borderlands 3 (PS4) (score=0.98225)***
Call of Duty: Ghosts (score=0.74930)***
Battlefield 1 (score=0.78389)***
Terminator Resistance***
Battleborn***
BioShock 2 Remastered***
Battlefield 4***
Titanfall 2***
```

- The prediction scores correlate to the confidence with which the model recognizes the games. Our scripts pick the prediction based on which title has the highest score per title:

- When the app receives the response from the backend server, it parses the response and creates a new screen to clearly show the results in a list. It first shows the predictions for the uploaded images, and then shows the recommendations after the user clicks on the "next" button:

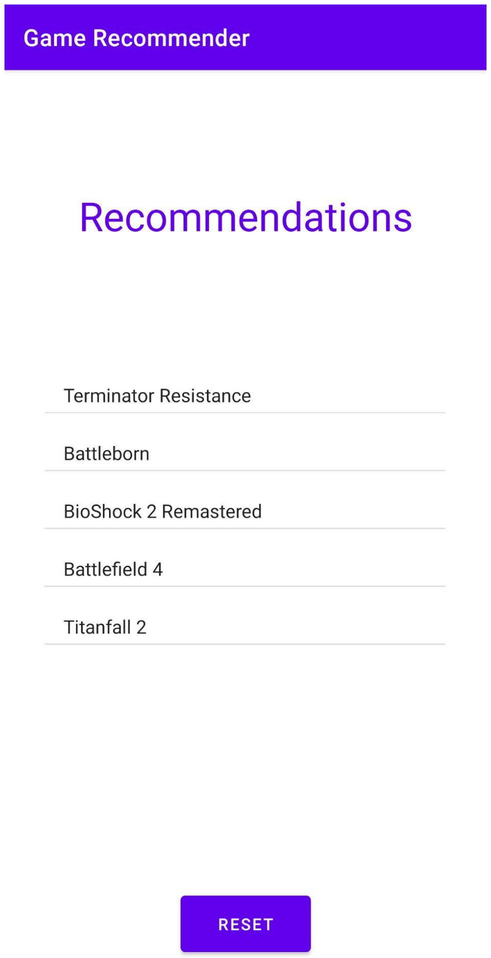

Game Recommender

Predictions

Borderlands 3 (PS4) (score=0.98225)

Call of Duty: Ghosts (score=0.74930)

Battlefield 1 (score=0.78389)


Game Recommender

Recommendations

Terminator Resistance

Battleborn

BioShock 2 Remastered

Battlefield 4

Titanfall 2

- We can see that our app performs very well in this example. It correctly predicts all the input images and produces very reasonable results for the recommendations, since all the recommended games are first-person-shooter games, which correctly matches the input games. If the user wants to go to the initial screen again and input other games, one can click on the “reset” button to do this.

## Major Components:

1. *Datasets:*

Our project uses three datasets - the games information dataset, the image dataset, and the user ratings dataset. The games information dataset includes the titles and the associated metadata of about 1600 games, and these are all the games that we have considered for building the app. This dataset was built upon the "PS4 Games" dataset from Kaggle, with extra columns that we added which contain the information for all the games that we deem useful, such as genres and developers. When the app is in operation, this dataset is used in the backend to ensure that the data communication between the different models are consistent, and it is also used as a lookup table in many places.

The image dataset contains the images of the covers of all the games in the games information dataset and the corresponding title portion extracted from those. The cover images were first web-scraped (using Selenium), and the title portions were then extracted manually. The image dataset was manually labelled and then used to train the YOLO title detector, while the title images were augmented in order to produce about 600 images per title. This collection was then used for training the classifier.

Lastly, the user ratings model contains about 100000 entries of a user's rating for a certain game, which are collected from Metacritic. This dataset of about 1000 games, which forms a subset of the games information dataset, and this is the list of games that our app can recommend. The other games are excluded either because they are duplicates or because we can't find any ratings for it on Metacritic. After data augmentation (by adding entries of user reviews for the users with very few reviews according to the metadata we collected in the games info dataset), this dataset was used to train the recommender model.

2. *YOLO Model:*

The YOLO title-detection model is the first of the two components of the image recognition in our application. It is responsible for segregating the title from the rest of the cover image in order to facilitate easier recognition.

Trained (using darknet) on a dataset of about 1600 images labelled manually, this model computes the coordinates of a bounding box surrounding the game title in the input cover image. This portion is extracted from the image and saved as a temporary file to be further used by the Recognition model.

3. *Recognition/Classifier Model:*

The Recognition model, trained to classify over 350 PS4 game titles, was trained using Google's MobileNet model which uses Tensorflow. We decided to use only a subset of the entire dataset of 1600 games for the image classifier because the dataset contained many unpopular, small-scale games which are not popular enough to be a user's favourite. So, we decided to include games that are over 10 GB in size so that we can get a more accurate classification for such games. The interesting part about the training of this model is that we only had a single image title photo per game (one image per class). But, since title classification is quite simple, we performed heavy data augmentation on each title image applying color overlays, skews, rotations etc. to the images in order to produce about 600 images per game. This was followed by 18 hours of training on a CPU. The final model was then tested with 15-20 images taken off Google Images (images that had some disturbances in addition to the cover image, for example an image clicked from a phone in non-ideal lighting conditions), and it was found to have an accuracy of about 85%.

4. *Recommender Model:*

The recommender model was built based on the idea of collaborative filtering. This model was trained on the augmented version of the user ratings dataset. It contains two embeddings, one for the users and another for the games. The embeddings are lookup tables for vectors with a certain dimension. For the user embeddings, one entry in it represents the preference of that user. And for the game embeddings, one entry represents the attributes of that game. The dot product of two vectors from the user and the game embeddings respectively is the predicted rating that the user would give for that game. However, for the purpose of our app, the user embeddings is useless for us since every time when someone uses the app and uploads images, we should treat the person as a new user.

Therefore, when our app is in operation, it only uses the game embeddings to generate recommendations on an item-to-item basis by comparing the similarities between the games' attribute vectors. For example, when three games are input into the recommender model, we calculate the cosine similarity values between each of the input games and all the other games. Then we take the mean of the three similarity values for all the other games and sort them in descending order to find the games that are most similar to all the input games, and use those as our recommendation.

5. *Frontend / App UI:*

The frontend provided the basic functionalities for the user to communicate with the machine learning models on the backend server. It was coded using Kotlin, with a very simple display, making it easy for the user to understand. When the user takes a picture using the camera, there will be a thumbnail display of the image to allow the user to ensure that the image was taken correctly. After 3 images are uploaded, the images get formatted and compressed to be used by the backend, then the 3 image predictions will be displayed on the next screen over in text. Finally, on the next screen after, 5 game recommendations will be displayed that are similar to the 3 games that the user inputted.

6. *Backend:*

For our application, a backend was required because the Machine Learning models operated using Python scripts, and the best way to run Python scripts in Android Studio applications is via a backend. We decided to use Node.js to design our backend. The backend stores the Machine Learning models as well as the scripts that run them. The main component of the backend is the file called "multipart.js". This is the main module which starts an Express server, handles requests and uses 'spawn' to run the Python scripts. The frontend first uses the Retrofit library to upload three game images in favourable quality to the backend. Once the three images have been uploaded, an HTTP request on the frontend signals the server to run the Python script "integrated.py" which is responsible for using the three Machine Learning models to compute the final output, which is then communicated to the frontend. In case of exceptions, measures have been taken to avoid issues such as crashing of the app, corruption of the uploads folder, etc.

Once the system was observed to be working satisfactorily, we decided to set the backend up on the Google Cloud Platform. For that, we are using an Ubuntu 18.04 Virtual Machine on the Compute Engine. We installed all dependencies such as Tensorflow, pandas, torch, etc. using the terminal, cloned the backend code from GitHub and transferred over the large Yolo model using Google's storage bucket. It has been observed that the computation is much faster when the backend is hosted on the VM as compared to when it is hosted locally on our computers.

## Workarounds and Bypassing Bottlenecks:

Throughout the course of our project, we faced many challenges and dead ends that we had to overcome by making alternative design decisions as a team. The most significant bottlenecks we encountered and the workarounds we devised for them across the different milestones are described below:

*Milestone 1:*

- The first challenge was training a game recognition model that could classify the game cover images according to its index in the dataset with good accuracy. We decided that if we were not able to train an accurate model, we would work on an image-text to text conversion model which would satisfy the same purpose. However, we weren't required to because we were successfully able to implement our initial idea by choosing to work with only a subset of carefully picked games and introducing the YOLO model to improve accuracy.

*Milestone 2:*

- Next we realized that to build a game recommender model, we could not only use the game information from the games information dataset as it cannot be directly used to train our model. We hence decided to build another dataset on the user ratings from Metacritic. In doing so, we were able to train the recommender model and finally achieved a good accuracy for the recommended games.

*Milestone 3:*

- By this point in the project, we were in the final stages of completion of our ML components and had to begin working on the backend which was new to each of us. We explored numerous options for building a backend such as Flask, Streamlit and Node.js. We hit a dead end with our Flask and StreamLit based servers, however we were able to get the Node.js server running.

*Milestone 4:*

- In this final milestone, most of the difficulties we faced came from establishing the communication between our android application and the Node.js backend. Unlike the other challenges, we realized that we could not devise a workaround for this but rather keep trying until we got it right. After exploring numerous online tutorials and websites and carefully debugging both the frontend and the backend, we were finally able to establish the connection that satisfies our requirements.

## Team Member Responsibilities:

*Aayush:*
Responsible for the image recognition and classifier models, collating all the models into a single script/module, Node.js backend, establishing the connection between the backend and frontend, adding functionality to the app UI. Worked on testing for edge cases / exceptions for the app. Deployed backend to the Google Cloud Platform.

*Harshil:*
Worked on manual adjustments of user_ratings.csv dataset (duplicates, 0 reviews), the game recommender model, a scrapped Flask backend, get/post requests, the Milestone reports, and logistics . Was responsible for researching and sharing resources on Google Cloud deployment, Retrofit, backend connections.

*Nick:*
Responsible for web scraping the information required for the games information and the user ratings dataset, worked on and tested/improved the accuracy of the game recommender model. Researched on Retrofit and worked on image uploading from the frontend to the backend. Tested and debugged the final working app.

*Kolton:*
Worked on the Android App UI / Frontend and helped the team with Kotlin. Worked on manual adjustment of the user_ratings.csv dataset as well and worked on the scrapped Flask backend. Added and changed functionality of the app upon design decisions made by the team during the meetings (buttons, screens, ImageView). Tested the app frontend.

*Additional Comments:*
- In addition to aforementioned responsibilities, each team member was present in all group meetings and offered substantial help towards the project.
- Most of the paired programming took place using screen sharing.
- Teammates juggled responsibilities and worked together on multiple tasks throughout the course of the project as well.

## Conclusion:

Through the development of this application, we were able to create an accurate recommendation system that we hope will benefit all gamers. By allowing users to take and upload 3 game images, and generating recommendations through existing reviews, it provides the user an unique list of recommendations compared to if you were to simply search up for a recommendation to a single game. The variability in the recommendations also comes from our machine learning models, taking the genre and developers into consideration.

As well, with the structure of the application, it makes the use of the app simple and straightforward. With all of the machine learning models hosted on an external server, it saves a lot of memory and takes away a large amount of burden on the phone. Overall, we hope that with a new way of recommending games, it will provide PS4 gamers a fresh new way to engage with a wide variety of games.