Final Project

Report

BIU CS – Final Project, Year 3.

Students:

## Omri Fridental

## Guy Wolf

## Oren Schwartz

Instructors:

Dr. Ariel Roth

Asi Barak

Course:

סדנה לפרויקטים – 89385

# 01 Our vision

Our vision was based on a real-life problem – gallery management. Gallery management is important for many different reasons (memory management, quick search, easy access and more). We wanted to explore a subproblem of gallery management which we call “AI filtering”.

The basic idea behind “AI filtering” is that some users sometimes want to filter their gallery based on some non-trivial mathematical function. This function is often hard to define mathematically. Therefore, we use a machine learning algorithm to train on datasets that have been pre-classified, in hope that our model can estimate the unknown function with little to no mistakes[[1]](#footnote-1).

With the main idea in mind, Omri has suggested that we explore a specific unknown function so that we can create an application that can be used by users who commonly search those pictures in their gallery. We have decided to choose a complicated yet useful filter function – Memes.

What is a meme?

It is a bit complicated to explain for those who have not heard the term before. Wikipedia defines a meme as “an idea, behavior, or style that spreads by means of imitation from person to person within a culture and often carries symbolic meaning representing a particular phenomenon or theme”. A good rule of thumb is that a meme is a picture with some funny text on it though there are many memes that do not fall into that category.

A very common example of a meme is the “brain meme”: (2 examples)



In the example we can see 2 memes that are created using the same pattern. The 2 memes have different jokes though, one call easily tell that they are indeed created using the same pattern.

Why did we choose to filter by memes?

We chose to filter by memes because of 3 main reasons:

1. Our personal galleries are filled with hundred or even thousands of pictures that can be classified as memes.

2. We thought this is a unique idea for a classifier, as far as we know there is not an easy tool that can accomplish this specific task.

3. The choice of this topic has led us to another very interesting problem, creating another model that takes our main model’s results and classify the memes into groups aka finding the “Template” of the meme.

What is a meme template?

A meme template is a plain image that has room for text, or other modifications. These images are specifically designed to create memes. A very common meme template is the “brain template” is used to create the 2 memes above. Our goal is for our second model (aka the “perceptual hash” model) to group memes based on their template.



# 02 Models

As we previously mentioned our application features 2 different models –

1. A model that classifies whether a specific image is a meme (aka the “Meme Classifier”).

2. A model that finds a given meme its template (aka the “Perceptual Hash”).

How do the Meme Classifier work?

The meme classifier was created using machine learning. There were two goals when making said model. First, the model had to be accurate. A small number of false positives was especially important, as the user would get irritated more by an obvious false positive than a false negative which would be less noticeable.

However, even a perfectly accurate model would have no use if it would not be fast and require little hard drive memory. No one would wait a long time for the app to classify all the images on their phone, let alone install such an app with a large memory requirement. Therefore, the accuracy of the app had to be balanced with its runtime and memory requirements.

To begin the making of the model, a dataset was needed. We created a script using the API of Reddit, a popular site that hosts many forums, to extract images – both from communities which are dedicated to memes and from those which are not. Special care was given to choose communities that may create false positives or false negatives – for instance, memes and screenshots from social media share superficial similarities, and therefore the dataset included screenshots from social media to ensure the model could differentiate between both. 10000 images from each class were extracted from Reddit, which were split to a training and validation sets in a 9-1 ratio.

After the extraction from Reddit the next step was to make the model. We decided to use the Python Tensorflow package, for several reasons:

1. It was easy to create models in, which helped to create, train, and test many models.

2. Tensorflow models can be used in other programming languages by an API, which makes the final model easier to integrate in any framework that we chose.

3. Tensorflow models can be specifically converted to models suited for model using a framework named Tensorflow Lite. This allowed for fast, efficient, and small models, which was exactly what was needed for this project.

The model we decided to use was a Convolutional Neural Network. Such a network was created and trained to great results – over 95% accuracy on the validation set. However, the network was ridiculously big – over 1.5GB before being converted to Tensorflow Lite and over 500MB after the conversion – and very slow. Therefore, the network had to be cut down.

Many different network designs were tested, with the final network being:

[Code snippet that I will add to the .md file later]

This model achieved an accuracy of 88% on the validation set, while only having a memory footprint of 5MB! However, false positives were still a problem – even 14% is noticeable. For that we needed to adjust the threshold of the results of the model.

When given an image, our model returns 2 values – the predicted chance of the image to be a meme, and the predicted chance of the image to not be a meme. Usually we pick the bigger number, as that is the more probable result. To minimize false positives, however, we could adjust said threshold to reflect our need for a more rigorous process for an image to be defined as a meme. The final threshold that was set on for an image to be declared a meme was 0.7, which minimized false positives to less than 2% while keeping the false negatives low. Thus, the meme classifier was ready.

How do the Perceptual Hash work?

Our second model is known as a “Perceptual Hash” algorithm[[2]](#footnote-2). The idea is quite simple:

Perceptual Hash is an algorithm that produces a “fingerprint” of an image – a small memory piece. We say that 2 images are similar if they have a similar/identical fingerprint.

A simple implementation (that is very similar to the one we used)

TODO: fix mistake as pointed out by Omri

flatten the image into a very small space (for example 16x16 pixels) and save the new compressed image as the original’s fingerprint.

Comparing fingerprints is quite easy: use the Hamming Distance algorithm to compare 2 fingerprints.

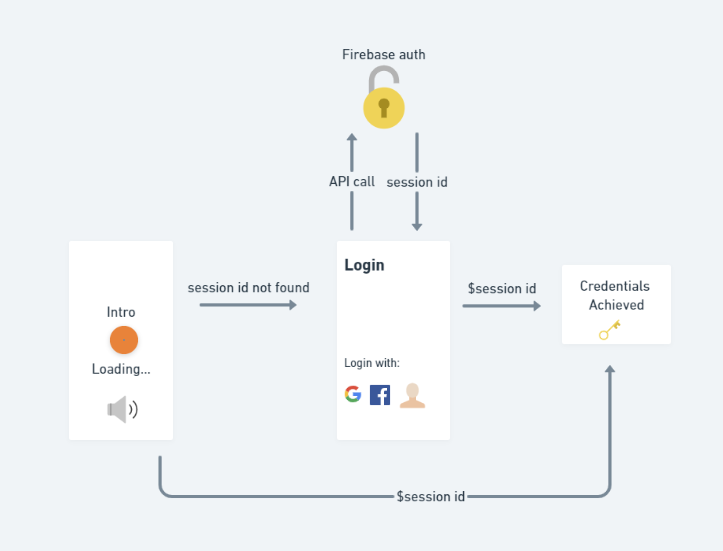
This gives us results that are good enough in our tests.

# 03 Application

Our goal with the application, was to create a user-friendly environment that can help the user perform the filtering that was discussed in our vision. We have decided to go with a 3-Part design that can be seen in our architecture graph[[3]](#footnote-3).

## Part 1 – Authentication

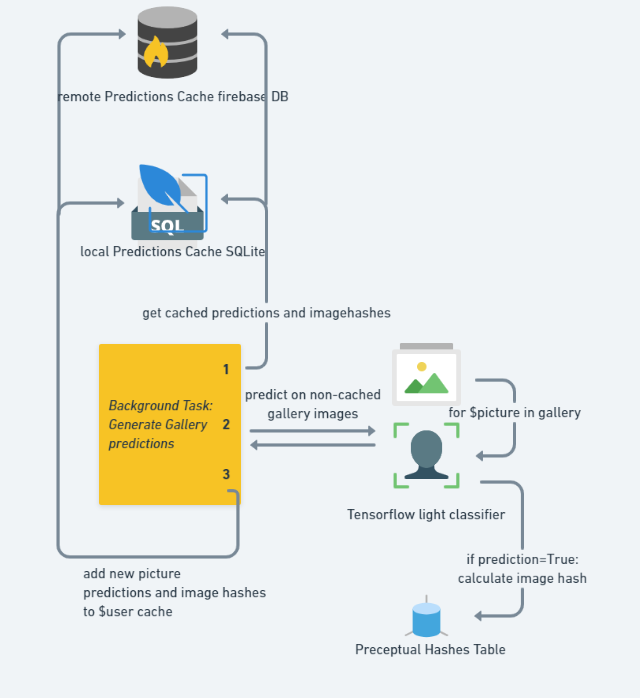
TODO: add reason for authentication



## Part 2 – Preprocessing

Preprocessing is the main part of our program this part functions on the 2 models discussed earlier in the “Models” section. The preprocessing calculates the predictions on the “Meme Model” for each picture in the gallery and then, if the prediction is true, it calculates the “image hash” with the second model “perceptual hash”. The calculation is done with a background task as can be seen the figure 2 below.

TODO: maybe add database explanation?



## Part 3 - GUI and Functionality

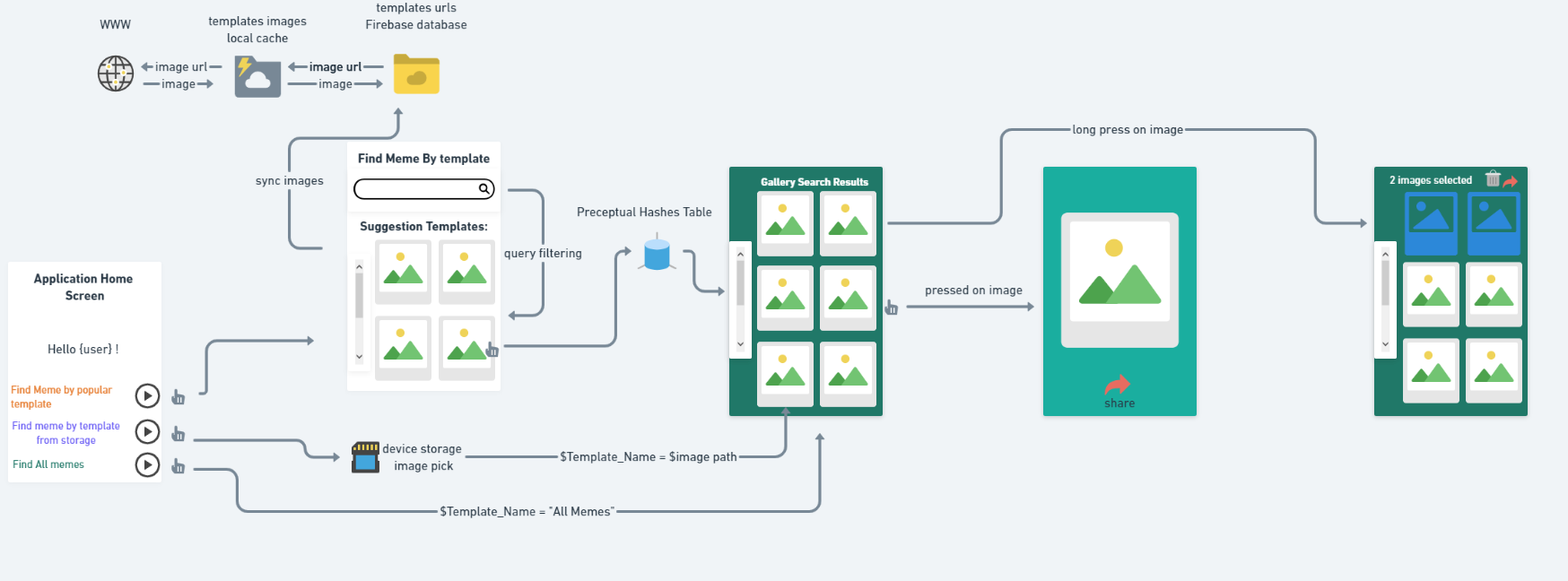
This part opens a menu that has 3 main functions:

1. “Find meme by popular template” – functions on the memes found via the meme model and groups them into popular templates (templates are downloaded from our list of popular templates).

2. “Find meme by template from storage” – allows the user to choose their own template to group by.

3. “Find all memes” – allows the user to see all the true predictions of our meme model.

Each will open a menu where the user can see the list of results and operate the images found as he wishes (delete, share, and other functionality is included).



# 04 Data base

## Cache Database

## 1. remote cache: firebase DB – No SQL

Schema:

1. <user id> → {
3. <path>: gallery image path on phone
5. <prediction>: isMeme prediction [yes/no]
7. <imageHash> imageHash
9. }

 Example:

1. {
2. "0ilcOZgolYWebTVeTI4GZr8kYMi2" : {
3. "SLASHstorageSLASHemulatedSLASH0SLASHDCIMSLASHCameraSLASH019ru2eclu251POINTpng" : {
4. "prediction" : true
5. "imageHash" : "0000000000001111000000000100111100000000110001110000000011101111000000000000111100000000000010010000001111111101000000101100101100000000000011010000000000001101000101001000111100000100111111110001010011111111000001001111110100000100111011010000111111111111",
7. },
8. "SLASHstorageSLASHemulatedSLASH0SLASHDCIMSLASHCameraSLASH01gx2tropv251POINTjpg" : {
9. "imageHash": null,
10. "prediction" : false
11. },
12. "SLASHstorageSLASHemulatedSLASH0SLASHDCIMSLASHCameraSLASH0225mth1sv251POINTjpg" : {
13. "imageHash" : "10011000000001111111000000100111100000000111111110000110011101111100100000110111100000001000010010000001111111101000000101100101100000000000011010000000000001101000101001000111100000100111111110001010011111111000001001111110100110100111011010000100001111111",
14. "prediction" : true
15. },
16. }

## 2. local cache: SQLite:

database manager and helper code at “src/main/java/com/example/memesfilter/sqlite\_local\_db” package.

1. // Table Name
2. public static final String TABLE\_NAME = "PREDICTIONS";
4. // Table columns
5. public static final String \_PATH = "\_path";
6. public static final String PREDICTION = "prediction";
7. public static final String IMAGE\_HASH = "image\_hash";

## Templates Database

Firebase DB – No SQL

Schema:

1. <random-template-id> → {
3. <path>: url to the image resource
5. <title>: template title
7. }

 Example:

1. {
2. "-M9dUsBalqBlTZZx6M\_M" : {
3. "path" : "https://imgflip.com/s/meme/Distracted-Boyfriend.jpg",
4. "title" : "Distracted Boyfriend"
5. },
6. "-M9dUsN9NQEyS2Xtm684" : {
7. "path" : "https://imgflip.com/s/meme/Drake-Hotline-Bling.jpg",
8. "title" : "Drake Hotline Bling"
9. },
10. "-M9dUsXR6BpNvHyZfz8G" : {
11. "path" : "https://imgflip.com/s/meme/Two-Buttons.jpg",
12. "title" : "Two Buttons"
13. }
14. }

# 05 Technologies

## **Languages used:**

* Python - model data scraping, model training
* Java - android mobile application development.

## **Development Tools used:**

* Jira - agile task management
* Confluence - knowledge sharing (characterization and project planning)
* git + github - version control system. ()
* Whimsical - system design and architecture planning.

## **Main Technologies used:**

* Android & Android Studio - mobile development.
* Firebase - cloud service for mobile application development:
  + Authentication
  + Storage
  + Database
  + Machine learning: ML-kit: on-device model predictions.
* Tensorflow, Keras - model training
* Reddit-API – free API, used to scrape images.

# 06 Final result

The result is a working application

TODO: finish conclusion WIP – Oren

1. Note that our model is a filtering model therefore we want as to optimize False-Positives [↑](#footnote-ref-1)
2. A good example of a working perceptual hash model can be found here <https://www.phash.org/demo/> [↑](#footnote-ref-2)
3. If you have permission the link is <https://whimsical.com/MsxyiraKAPwyeeEeWKk5u8> (you can ask for permission if you want to see the full graph) [↑](#footnote-ref-3)