Transfer Learning for Food Classification

Alejandro Esquivias
UPM
alejandro.esquivias.canadas@alumnos.upm.es

Przemyslaw Lewandoski
UPM
p.lewandowski@alumnos.upm.es

Ismael Sánchez
UPM
ismael.sanchez.rivero@alumnos.upm.es

Jiachun Chen
UPM
jiachun.chen@alumnos.upm.es

Ákos Kuti
UPM
akos.kuti@alumnos.upm.es

David Engelstein
UPM
david.bengelstein@alumnos.upm.es

Abstract

In this paper we describe the development of an Android application capable of classifying 16 different types of desserts, with an accuracy of 71.5%. To tackle this problem, we first tried a dimensionality reduction to train our model with our images on an SVM. Due to the low accuracy we were obtaining we decided to use transfer learning with three different types of algorithms, by adjusting their learning weights and the type of optimizer used. We developed a REST API in Python with different useful functions to retrieve a prediction, once the model was called with the image to be classified in our localhost. Finally, we describe and show images of the Android application.

Keywords- Android, SVM, Convolutional Neural Networks Transfer Learning VGG16, ResNet50, InceptionV3.

1. Introduction

Food image recognition is one of the most promising applications of object recognition and classification, due to its vast number of applications. Many works on this topic have been published so far. We are focusing specifically on 16 different types of desserts. This task is not trivial, because as we will see there are some desserts that are similar and are very hard to differentiate even for an average human.

The problem which we were trying to solve by the software was related to the consciousness of the possible negative impact of food on the human body. Nowadays many people struggle with overweight or obesity. In 2016, 39% of adults aged 18 years and over (39% of men and 40% of women) were overweight [1]. Of course, there is already an application which are supporting users with a conscious nutrition plan, but the main barrier in those concepts (and on the other hand, the core advantage of our project) is the efficiency of data input. It requires a lot of data to be provided manually so the interaction is

intrusive. Taking a photo is much easier and it would be good enough in situations like shopping or eating lunch in a restaurant. After taking a photo, the application would display the nutritional values of the food. There are also other possible use cases of the app like:

- Informing about possible allergies for ingredients in a product
- A guide for people with vision problems
- App for calculating stocks in a shop

Taking into consideration the primary function of the app, it could be developed into a more complex system. Still, the base function relies on food image recognition but what could be done more in the future is:

- Storing nutritional value – then the app has a full potential to be useful for people who follow a diet
- Give diet advice – another feature, which would send pieces of advice based on gathered food information
- Scan to buy – Nowadays some shops use take away code readers, so then customers don't have to put out all the purchases at cash points, they just pay for all scanned products. Mobile applications with product recognition also could be such a scanner.

First, we will review some related work, then we will describe the materials and methods used, by describing the dataset, the back-end, the front-end and the connection between them. In section 4, we will explain our results with a heavy focus on error analysis describing some identified patterns.

2. Related Work

As far as we know there are no applications that focus solely on dessert recognition. However, there are a few interesting articles that aim to classify food in general. In [2], the authors use CNNs (recognized as state-of-the-art approach in deep learning) and fine tune the algorithms to recognize more than 1000 different classes. They later use feature extraction (Color FV, RootHOG …etc) to achieve nearly 95 % accuracy.

[3] follows an interesting approach, consisting of directly measuring food calorie intake in a picture, by using deep learning neural networks. The user takes a picture, the model recognizes it and is able to calculate the calories in the uploaded picture. The most remarkable result in this paper is that the performance is generally better, when using graph color segmentation for single food (orange, banana, apple…etc).

NutriNet [4] is a new deep neural network architecture that promises a surprisingly high accuracy (86.72%) considering the number of classes that are dealt with (520). Additionally, this model is being used as part of a mobile app for the dietary assessment of Parkinson's disease patients. In terms of the network, NutriNet is a modification of the AlexNet architecture, with an additional convolution layer at the start (6 convolution layers in total) and accepts different image sizes. This work is interesting to us because it shows that different efforts have been made (and are made) to develop these types of applications.

3. Materials and Methods

3.1 Dataset

The nutritional values for the different types of desserts were extracted from [5]. With that information we created a CSV file containing information regarding calories, total fat, cholesterol, sodium and protein per a certain quantity, referred as the serving and measured in grams (g).
In terms of the pictures and their labels, we developed a function in Python to iterate through our dataset and retrieve the name of the folder to associate it with the corresponding image. Once, we established the association, we resized all the images to (150, 150,3). The choice of 150 pixels was arbitrary and just follows the reason of reducing the computational power required for processing. The next step involves vectorizing the images. By doing so we obtain 16039 arrays of dimensions (150,150,3) for the images. Finally, we used keras preprocess_input (this function subtracts the mean RGB channels of the ImageNet dataset), when training our models. Fig. 1 shows this function applied to a picture in our dataset.

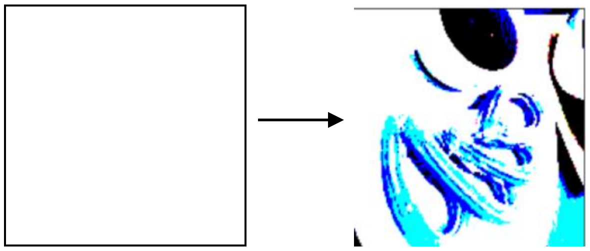


Figure 1- Demonstration of the preprocess_input function

3.2 Back-end

Firstly, we tried to apply an SVM to the pixels. In order to do this, we had to reduce our data's dimensionality so that it could be used as input for the SVM. With the images already resized to (150,150,3), we constructed a vector of dimension (1,67500) per image in the dataset. We then applied an SVM to the vectorized pixels. However, this method was not giving us good results we changed our approach and tried to fine-tune different transfer learning algorithms, by re-using pre-trained models with ImageNet weights, by adjusting their learning rate, by starting from 0.1 and dividing by 10 after each time we trained our model until we reached 0.0001. We also modified the optimizers (Adam and SGD). In this next part we will briefly describe the three transfer learning models we tried.

VGG16 Convolutional Neural Network proposed by K. Simonyan and A. Zisserman (Oxford University). When tested with ImageNet (image database containing over 15 million labelled high-resolution images, following a WordNet hierarchy [6]) images it is capable of obtaining 92.7 % accuracy. In Figure 2, we have depicted VGG16's architecture. VGG16 is composed by 13 convolutional layers, 5 pooling layers and 3 dense layers.

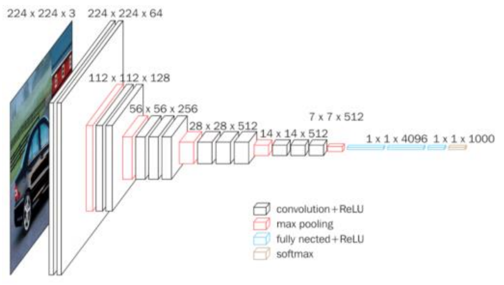


Figure 2 – Diagram illustrating VGG16's architecture.

ResNet50- It is a convolutional neural network that is 50 layers and is usually used for image classification. It deals with the "vanishing gradient" problem (small gradients resulting in poor conversion towards optimal weights).

The main curiosity is that it can skip connections during training, when it deems the connections are less relevant. It has an accuracy of 92.1 % when trained and tested on the ImageNet dataset.

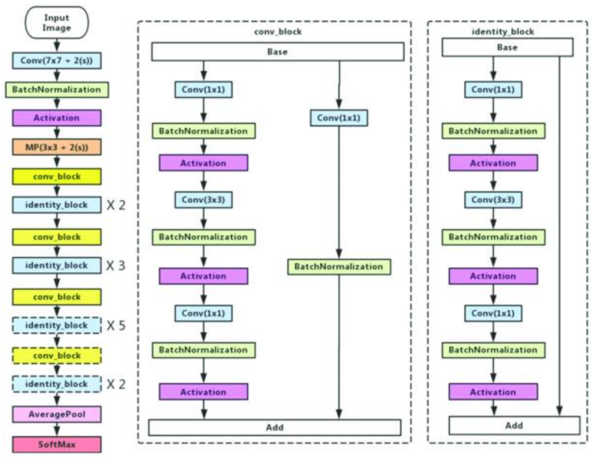


Figure 3 - Diagram illustrating ResNet50's architecture.

<u>InceptionV3</u>- This convolutional neural net was originally introduced during the ImageNet recognition challenge. InceptionV3 has been used in the past in the research of Leukemia. It has an accuracy of 93.7% when trained and tested on the ImageNet dataset.

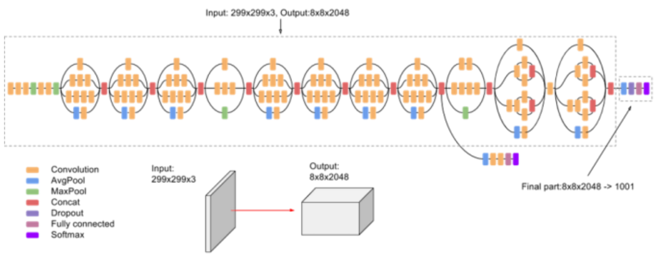
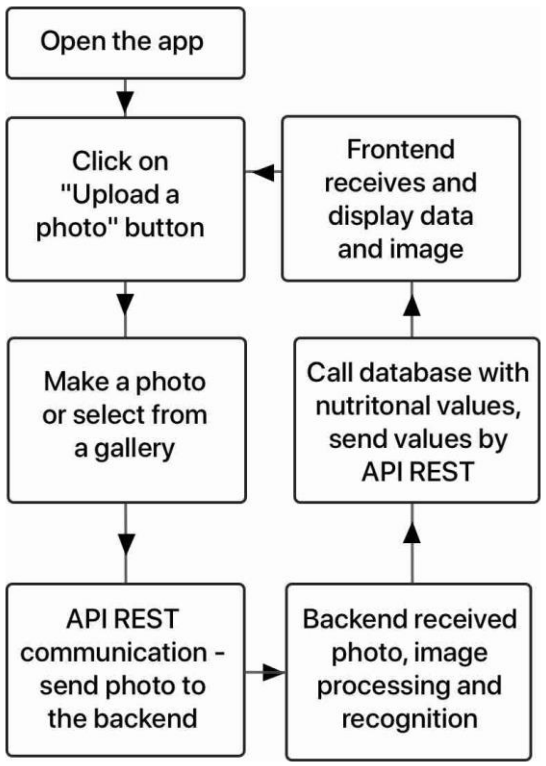


Figure 4 – Diagram illustrating InceptionV3's architecture.

Once we obtained a sufficiently good result, we saved our model to JSON format and our weights to h5 format so that it could be loaded later. This part is explained in 3.4.

3.3 Front-end

First of all, we had to decide what kind of device would be best. The most popular types of apps are nowadays web apps and mobile apps. Taking into consideration the versatility of use cases (by the end-users) better solutions are mobile devices. Then our solution would be also useful out of home - like for example for shopping. Then, taking into consideration the speed and convenience of usage we decide that our solution should be a mobile app, which may provide the best user experience. Also, worldwide data about the share of used devices shows that mobile devices are used by 52.1% of the population, compared to desktop market share of 44.2%. [7] After we decided on a mobile application, we had to choose an operating system. We considered iOS and Android. On the one hand, we hear that iOS applications might be easier to implement. But on the other hand, not every one of us is familiar with iOS development, so we decided on Android application and Java programming language. Also, we took into consideration international customers' preferences. Android is the dominant operating system for mobile devices (72,26% worldwide share) compared to iOS (27,03%) [8]. We also had to discuss user flows for our application. We decided on a very minimalistic approach. Every possible action and connection between them is presented in figure X. Figure X represents visuals of the app, in 2 possible states - waiting for upload and after uploading a picture.


Open the app
Click on "Upload a photo" button
Make a photo or select from a gallery
API REST communication - send photo to the backend
Backend received photo, image processing and recognition
Call database with nutritonal values, send values by API REST
Frontend receives and display data and image
Figure 5: User flow of our application.

The following figures represent visuals of the app, in 2 possible states - waiting for upload and after uploading a picture.

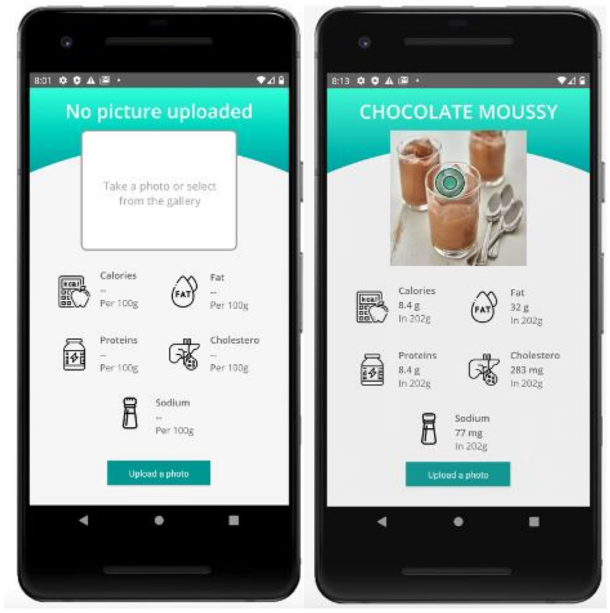


Figure 6: UI used for the application

We did have a lot of problems in the development of the Front End. The biggest difficulty was finding a good way to save the image obtained through the cell phone camera. The solution used was to use the library to use the Bitmap format.

3.4 Connection between front-end and back-end

We developed a REST API in Python that could be called from our Android application. In the API we created a function that was responsible for loading the model and running keras' predict_classes function when it was given a model and an image in array format. To retrieve the correct label and nutritional information we simply called the appropriate row in the dataframe containing the nutritional information. Since we were using an emulator to develop our application and we were running this on our localhost, we had to map the computer's localhost address to a global address. To do this we used a tunneling program known as ngrok. In this step we faced formatting problems. To solve these problems, we developed an additional function in our REST API, that formatted the nutritional values and the label of the food in dictionary form.
In the Android app, we called our localhost with a POST request containing the image and we retrieved the response to populate the corresponding fields dynamically. Below is a diagram of the connections.

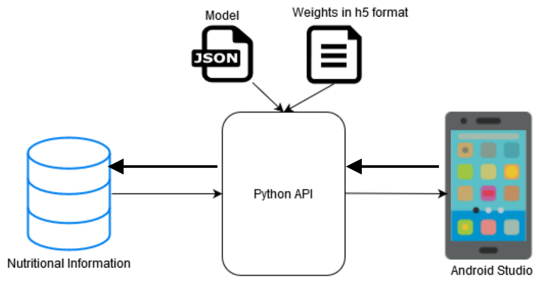


Figure 7- Structure of all the connected parts

4. Results

4.1 Transfer Learning results

Using an SVM with the vectors of flattened pixels we obtained and accuracy of 77.8% on the training set and an accuracy of 45.6% on the test set.
The table below summarizes the best accuracies we obtained (rounded to the closest decimal) by modifying the learning rates for both optimizers after training for 15 epochs, for each individual transfer learning algorithm.

Optimizer / Algorithm	Adam Optimizer	SGD Optimizer
ResNet50	Train Set: 96.1%	Train Set: 99.24%
	Test Set: 69.4%	Test Set: 71.5%
VGG16	Train Set: 82.1%	Train Set: 65.0%
	Test Set: 66.6%	Test Set: 31.9%
InceptionV3	Train Set: 86.6%	Train Set: 95.2%
	Test Set: 68.3%	Test Set: 58.0%

Table 1- Table showing the best recorded accuracies both in test and train set

4.2 Error Analysis

In order to understand and evaluate the functioning of the model, it is very important to analyze its accuracy and its errors. First, we made an analysis of the general accuracy of each food contained in the project. Through a horizontal bar graph, it was possible to see and compare the results obtained by food (Figure 8).

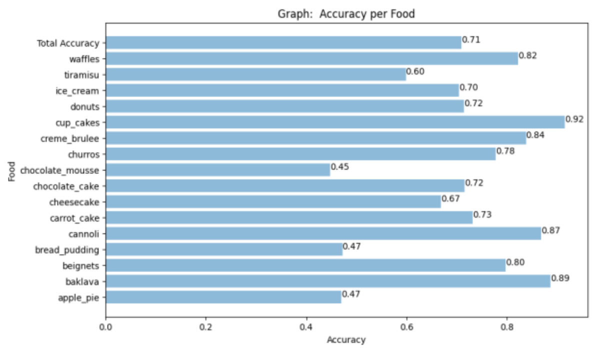


Figure 8 - Accuracy per food - This graph shows the total accuracy obtained by each food

As we can see, the model has a total accuracy of 71.5%. However, it is noticeable that some foods have better accuracy than others. For example: the chocolate mousse has the worst results, with 45% accuracy. In turn, the cupcake has an excellent 92% accuracy, the best result among all the foods studied.

To better evaluate the functioning of the model, the food with the worst results was analyzed: chocolate mousse. This analysis aims to understand a little of the justification for the low results obtained.

4.2.1. Chocolate Mousse

First, we analyzed how the images labeled as chocolate mousse were classified. We can see the results in the Figure 9:

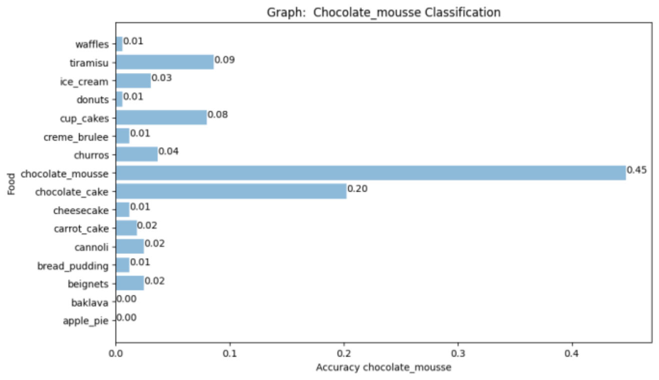


Figure 9 - Chocolate Mousse Classification – This graph shows us how the foods labeled as chocolate mousse were classified by our model.

We can see that only 45% of the time chocolate mousses are classified correctly. And that 20% of the time is classified as a chocolate cake, a food that can be very similar with a mousse, mainly due to the same ingredients and colors. Several factors can be the cause of the error. To search for a pattern, it is possible to make a general analysis of the images and how they are predicted. In the Image 1.3 we can see that there are some possible reasons to the missed classification. First, is the similarity of the images labeled as chocolate mousse with other foods, like the third picture of the image 1.3, that is very similar to a cheesecake or the fifth picture of this same image, that looks like an ice cream. In addition, other possible reasons for errors are: different colors, different shapes, and, lastly, an excess of colors can also cause some confusion.

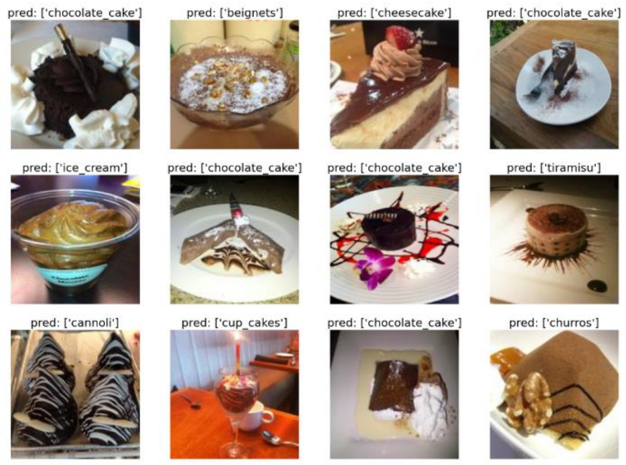


Figure 10 - This image shows the pictures and the predictions of the foods labeled as chocolate mousse

After that, it is possible to do a second analysis to answer the following question: What are the characteristics that determine that it will be predicted as chocolate mousse?

To answer this question, we can see how many times we predict food like chocolate mousse correctly. As we can see on the Figure 11 the result is better than the result obtained in Figure 8. We can see that 65% of the times that we predict an image as Chocolate Mousse we do it correctly.

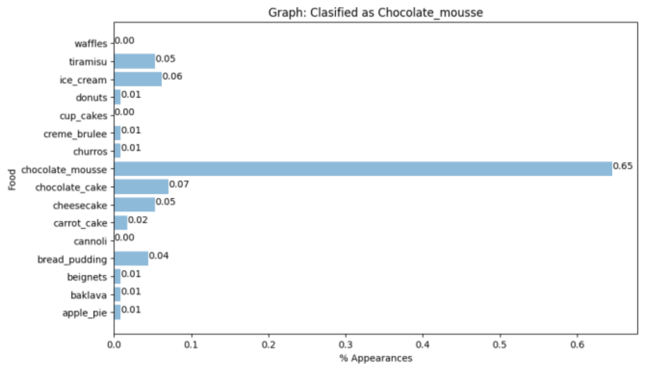


Figure 11 - Classified as Chocolate Mousse – This graph shows us the percentage of the true labels of the images that the model predicts as chocolate mousse

Another question that can help us checking the quality of the model is: What features does the model use to classify as chocolate mousse? To find a pattern, we can look at the images classified as chocolate mousse and see if there are any characteristics that bring them together. The Figure 12 shows us that there are two very clear patterns: the color (dark brown and white) and the cone format that explain some of the results obtained.

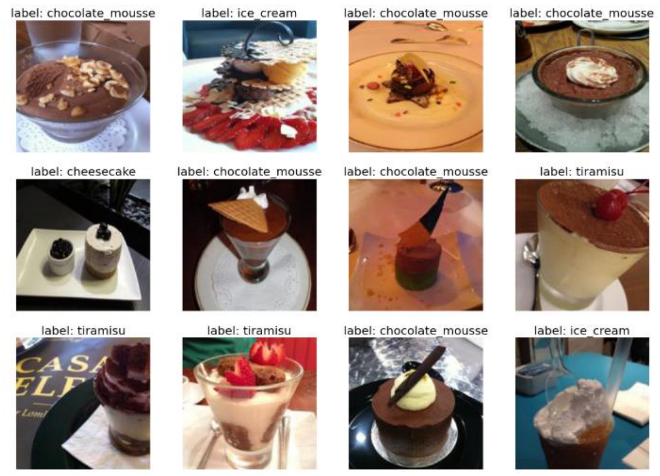


Figure 12 - This image shows us pictures of the foods that the model predicted as "Chocolate Mousse".

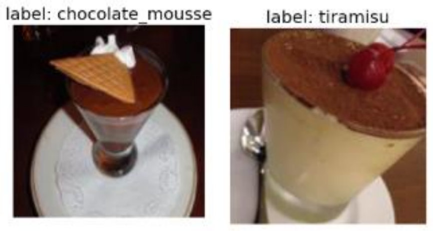
4.2.2. General Error Analysis

Looking at all misclassified images, we can see some reasons why they are being classified wrongly. Some of the reasons are:

- Non-food images

- Too much information

- Similar Formats



- Similar colors

- More than one food in the same image

- Very colorful images

- Backgrounds that are too large

Therefore, it is possible to conclude that, despite failures, our model is satisfactory. A change that could improve the results would be to clean the dataset more carefully, something difficult due to its large size of 16 thousand images. Another possibility that could present better results would be to select foods that are very different from each other, instead of choosing a category of foods as it was done (desserts). This would reduce errors due to similarity between foods and increase the accuracy of the model.

5 Conclusion

Despite problems related to the lack of computational power or the low speed Google Collaboratory has when processing images and integration problems we managed to develop a fully functional Android application.

Thanks to this work we have learnt about transfer learning and its big potential in solving difficult image classification tasks. For future work we suggest segmenting the images. In that way we would obtain only the desired section of the image, since one of our problem was our noisy images.

Other transfer learning algorithms could also be tried, and transfer learning in itself could be used to extract features from the images and then training a classifier (SVM, decision trees, k-nearest-neighbor…etc).

A comparison between different types of classifiers is interesting for future iterations. Also, once a better accuracy is reached it could be useful to train a classifier on food vs non-food objects to prevent it from classifying pictures that are not food. This is especially useful, if we want to deploy this application to reduce the amount of computational power needed, when calling the API on a remote server.

References

[1] Obesity and overweight WHO, https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight .

[2] Food image recognition using deep convolutional network with pre-training and fine-tuning by Keiji Yanai and Yoshiyuki Kawano.

[3] "Food Calorie Measurement Using Deep Learning Neural Network" by Parisa Pouladzadeh, Pallavi Kuhad , Sri Vijay Bharat Peddi, Abdulsalam Yassine, Shervin Shirmohammadi.

[4] "NutriNet: A Deep Learning Food and Drink Image Recognition System for Dietary Assessment", by Simon Mezgec and Barbara Koroušić Seljak

[5] Nutritional Information database available at https://www.nutritionix.com/
[6] ImageNet database, available at http://www.image-net.org/

[7] https://techjury.net/stats-about/mobile-vs-desktopusage/#gref

[8] https://gs.statcounter.com/os-marketshare/mobile/worldwide

[9]Link to our github: https://github.com/ESQ0001/ImageMining