# Scalable system for machine learning applied to image rating

**Hongyu Tu**[1]

## Abstract

In the project, I aim to create a pipeline that can collect images from Instagram and related information including the number of likes and comments. The data then gets fed into a machine-learning model that will try to learn the correlation between a photo itself and its popularity. the whole system was designed to be online (i.e. always up to date on each run), so it is highly scalable when there's more data as time goes on. This paper shows the entire process of how the dataset was created, maintained, and used for the NN training.

## 1 Introduction

We are currently living in an era of social media, about 95 million images are posted to Instagram every day. As somehow who has a lot of interest in photography, I personally follow a lot of photographers who take amazing pictures. It might be obvious to tell the difference between the pictures taken by a professional verse an amateur, however, the popularity between different photographers or even different pictures taken by the same photographer can be huge and not as obvious as it seems. Therefore, I pose the idea to learn the pattern behind popular posts, and hopefully, I can learn some insights into what makes them popular.

## 2 Related Work

A major reason why I found this task interesting is that not much work has been done on rating images, especially those related to social media. One work that I have found is studying Aesthetics in Photographic Images Using a Computational Approach (Datta et al., 2006), their training data consists of around 3000 images collected from Photo.net. Since it's from 2006, the algorithm used in the paper is traditional CV algorithms like support vector machines and visual feature descriptors. Another related work is Automatic Rating and Selection of Digital Photographs (Kormann et al., 2009), similar to the prior one, as it's from 2009, the techniques used in the paper are SIFT keypoint descriptors and support vector machines, applied to 840 images. Both those papers are limited to the lack of more advanced big data techniques and my work will try to take a shot at it.

## 3 Approach

The whole approach can be simplified to three steps: collect data, process and store them, then feed them into a neural network. Details will be covered below.

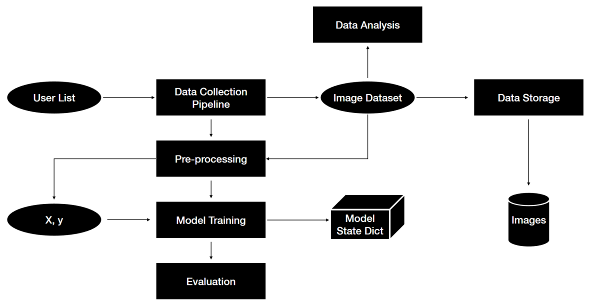


Figure 0. Overall Workflow

### 3.1 Data Collection

After going over the available datasets online, I couldn't find suitable existing ones that are suitable for this project. Therefore, I decided to collect the data I need myself. To avoid traffic attacks, Instagram has some protection protocol implemented that prohibits users to straight up scrawl data from the website, therefore, I used an open-sourced library called 'Instaloader' (`https://instaloader.github.io/index.html`). Even with the help of the API, I still encounter a lot of timeouts and errors during the process of collecting the data and had to constantly switch my IP address in order to keep the downloading process going. I would say this is one of the biggest challenges that I had to deal with during the entire project, as the process was boring, and not many results can be shown as it's only the first step.

Aside from that and back to the main topic, I essentially build a wrapper around the crawl function from Instaloader in this step. With a list of users that I want to study provided, the wrapper will download the number of specified posts (all by default) from all users in the list. For each post, besides the image itself, the number of likes was obtained, as an indicator of how good a picture is, as well as the date that particular post was made.

More specifically, in this step, I composed a list of 27 users, all of them are photographers that I follow personally. They have a combined follower count of 9,421,078, and a combined post count of 71,747 posts (as of December 10, 2022). To have a better understanding of the 27 users that I have chosen, here is a plot showing how many followers each user has. Notice that the y-axis is on log scale, and this graph, I believe, is a great demonstration of the wide range of distribution of the demographic I have chosen here.

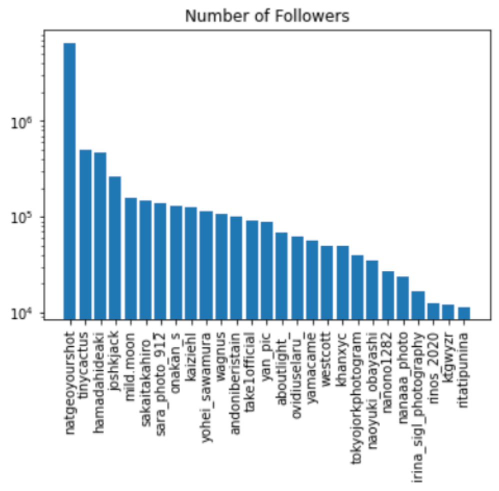


Figure 1. User follower info

Going back to the wrapper, for scalability, it was made in a polymorphic way. This means that no matter how many users we want to expand the study to, all we need is to put all their username into the text file and feed it into the wrapper, and the code will automatically go through the list one by one and have everything stored in the exact format that will be used for the steps later. Another advantage of using the API is that it keeps track of already-downloaded data. This allows the wrapper to not worry about re-downloading things when I try to get my dataset up to date, it will only download new posts from the user that comes after the last time the code was run. All the design choices mentioned allows the code to be reusable and guarantee scalability.

## 3.2 Data Analysis

With all the data in hand, I need to have an idea of how it looks and do some balancing so I can make adjustments and system design choices for the next steps. To see if it makes sense to use the number of likes as an indicator of how good a picture is, I tried to show the top 5 and bottom 5 posts with the least amount of likes. Since there are some posts containing more than one picture or some with videos that can be hard to display, I filtered the data to keep only images and if there is more than one in the same post, I will only keep one that was used as the thumbnail. Displayed on the right, we can see the images on the left are the most liked ones and the other half are least liked. From the aesthetic standpoint, I'd say it seems to be fair. As the most liked ones clearly have either better framing or better story-telling, whereas the ones on the right are more on the ordinary side when compared to the ones on the left.

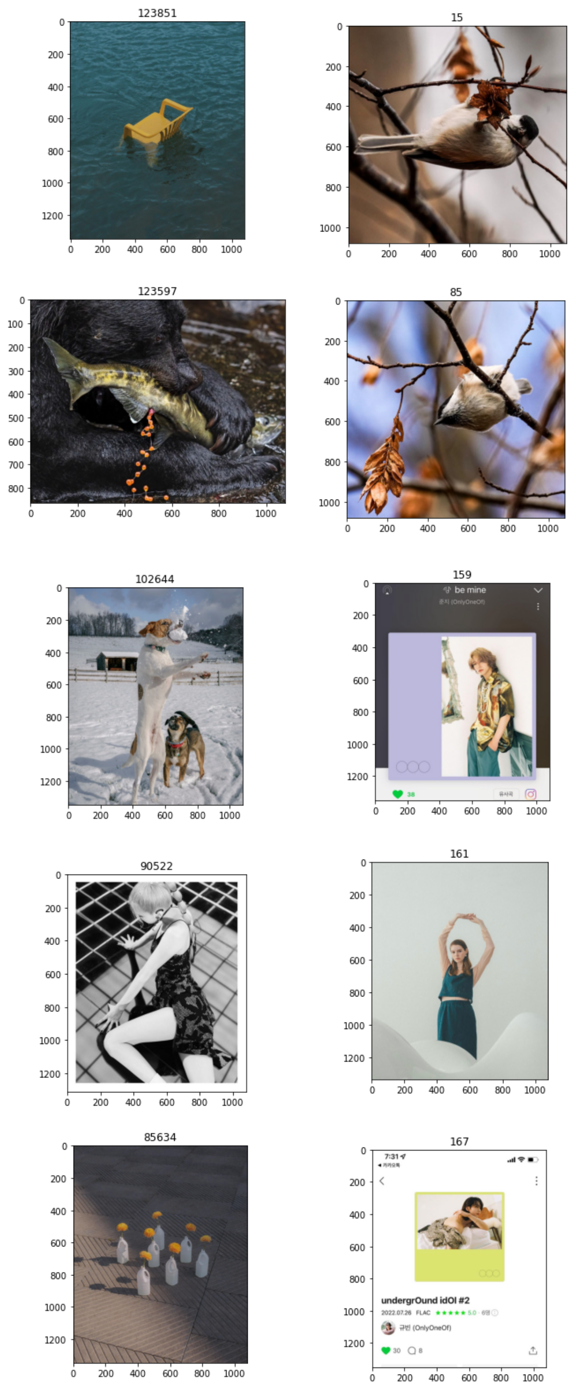


Figure 2. Top/Bottom 5 Liked posts

For further studies, I tried to standardize the number of likes of each post either based on the total number of followers each user has, or the post with the highest number of likes. Another attempt was to get the number of days each post has been posted by subtracting the original post date from today's date. These two attempts were to answer two questions that have a major effect on the model design later, which will be discussed in more depth in section 4.1.

## 3.3 Data Storage

Throughout the original development process, I used pickle (https://docs.python.org/3/library/pickle.html), a python library as my way to store data and have them shared between different codes. It was easy to use as it can pack list or NumPy array objects into a binary file and unpack later. However, the use of pickle requires code knowledge and is not straightforward and not as flexible when it comes to building a scalable system.

With all that said, for query purposes, I migrated all the data to MongoDB. Pymango (https://api.mongodb.com/python/1.6/index.html) was used to connect my script to the MongoDB server and use code to insert data into the database in batches. There are two tables, one for the posts, each row has fields including the number of likes, the image reshaped to be a vector, the number of days it has been posted, and lastly the user that posted it. Another table is for the users, each row has fields including the followers each user has, the number of posts they each have, and the key is their user name. Overall the process was super simple and the dataset can be exported to JSON files. All these are made available on Google Drive.

## 3.4 Pre-processing

With the data analysis done and stored in place, it's time to get them ready for the model training. The number of posts from each different user as balanced so they each have about the same amount of posts in the dataset. Then all the images were shuffled so the model won't get any extra information from the ordering because the only information that should use for the model to make their decision are the images themselves.

Since Instagram doesn't have a strict rule on the sizing of the posts, at this point, the images in the dataset have all likes of different sizes, which is not acceptable for a machine-learning model. I, therefore, rescaled all images to 50 by 50, a size that is pretty standard for image processing, where a decent amount of information can be preserved while keeping the computation less expensive.

Another standard procedure during the pre-processing stage is gray-scaling, which can reduce the data size by 67% as it keeps only one channel instead of 3 (red, green, blue). However, I made the decision not to do that and keep all three channels, as this is a project about photography, and colors are a huge part of it. There are photographers that only shot black and white photos, and some are known for the vibrant coloring in their work. I want to make sure my model makes use of this information and makes better predictions when wondering if someone will like a picture or not.

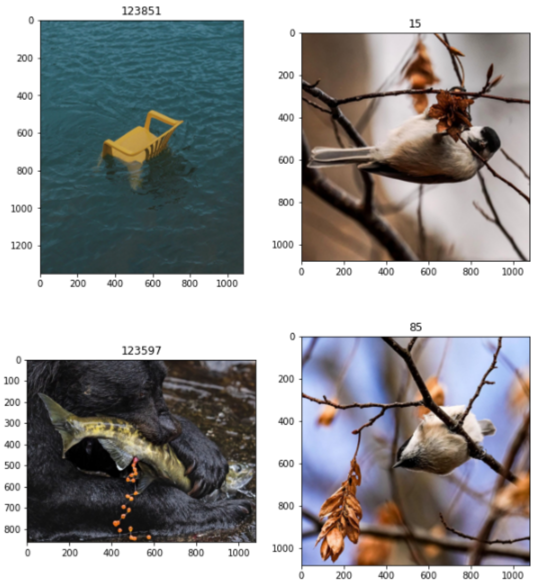


Figure 3. Example image before pre-processing

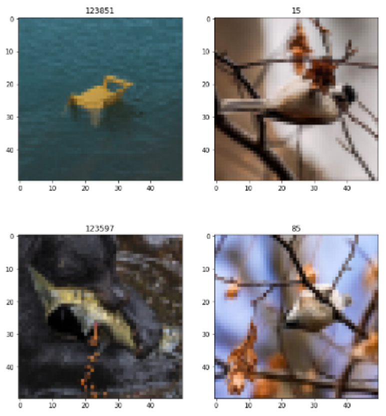


Figure 4. Example image after pre-processing

Finally, before feeding all the data into the model, I make an 8:1:1 split on the data for the train set, validation set, and test set. This is standard as well, allowing me to do parameter tuning for the model training and can prevent over-fitting.

### 3.5 Model Training

For this image rating task, I used a multi-layer neural network. It started with a 50-by-50 to 40-by-40 convolution layer with a filer size of 11 by 11, and an output channel of 10. It was followed by a 40-by-40 to 36-by-36 convolution layer with a filer size of 5 by 5, and an output channel of 5. To reduce the chance of having a model that is too large, a max pool layer with filter size 2 was added as well as a dropout layer with 0.25 probability. After all this, each post has a size of 18 by 18 and a channel number of 5. Then, a flatten layer and three dense layers were used to compress the number of vectors all the way from 1620 ($18 \times 18 \times 5$) to 500 and then to 1. LeakyRELU with 0.1 was used between all layers to add the non-linearity.

PyTorch v1.11.0 with GPU was used in this step. I've decided to use Adam as my optimizer so I can worry less about setting the right learning rate. Since we are looking to match the standardized number of likes each post gets, it's basically a float point comparison, so it makes sense to use Mean Squared Error as the loss function here. To perform the gradient descent, I used Batch Stochastic gradient descent with a size of 50 and an epoch number of 50.

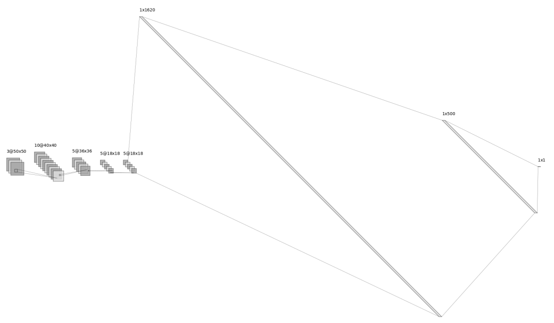


Figure 5. Model Structure

### 3.6 Evaluation

To have an idea of how well the model is performing, I used the linear regression model from scikit-learn (`https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression`) as my baseline here. Since there's no parameter tuning required here, I used an 8:2 split for only the train set and test set.

To compare how well the model is actually performing, I implemented a custom metric to rank images based on the score they get from both models and see how it compares to the actual ranking. The samples are randomly drawn from the dataset.

## 4 EXPERIMENT

In this section, I will be talking in more detail regarding the results I obtain from the above steps.

### 4.1 Data Analysis

As mentioned in section 3.2, there are two questions that wonder before the training process:

- Does days since post matter?
- Does the follow count of user matter?

To answer the first question, before I looked at the data, I believe there should be a positive correlation between the days one post has been posted with the number of likes they get, as more time online means more publicity (more views) and thus more likes. So I sorted the posts for each user based on the number of days and made a plot showing the trend of the number of likes with respect to the days.

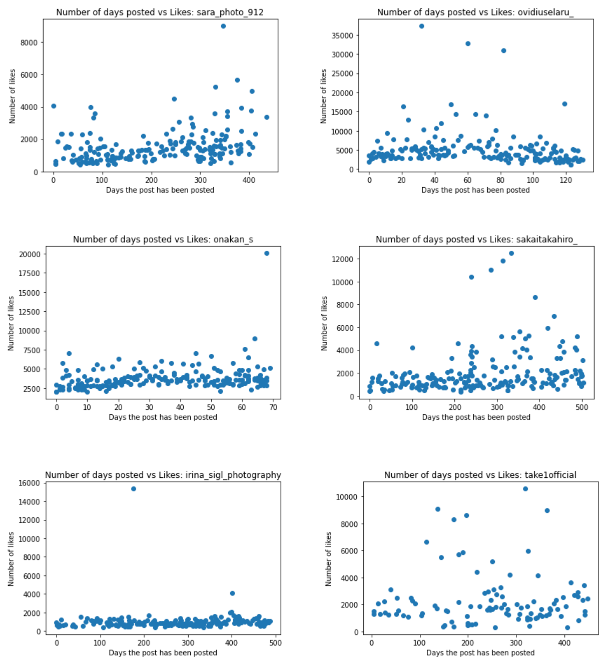


Figure 6. Days vs Likes

From the plots, it seems to be the number of likes doesn't change much as the number of days increases. This makes sense to me as well though, because, in real life, we only like the posts that are recent, we don't usually go to the bottom of someone we just followed and like their photos from way back. The number of followers each user has also only grows over time, the publicity can be a lot larger than what they can get when they just started the account. Lastly, when Instagram recommend posts, they are usually recent posts and not posts from 5 years ago, this also makes up the difference in view, so it is totally possible for a recent post to have more view count than a post that was posted a long time ago. With this figured out, I decided not to penalize the number of likes a post has during training as they don't really give an edge.

Continuing to the second question, with more followers, I would also suspect the difference in followers will give a less good image more likes, which is not helpful when the model is learning about the images themselves. If so, I should penalize the number of likes based on the number of followers each user has, and the score will be replaced by the average number of likes each post gets per user: if every follower likes a post, the score will be 1, and if no one likes the post, it will be 0. Since liking the post isn't restricted to followers, there can be cases with scores larger than, however, it should always be larger than 0. It makes sense for me to have this design until I, again, plotted out my hypothesis.

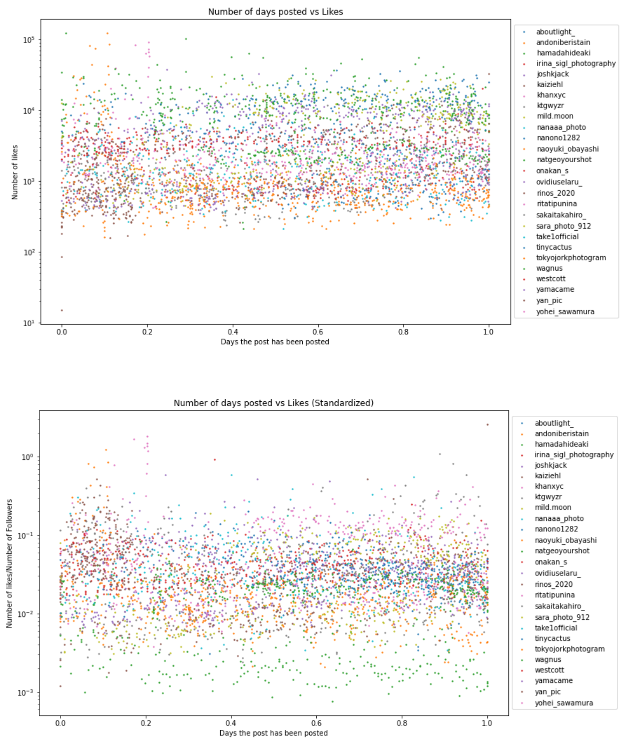


Figure 7. Non-standardized (Top) vs standardized (Bottom)

As shown above, before the standardization, the dots with different colors are mixed together, which is what I wanted as the different colors indicate different users. However, as we look at the plot at the bottom, the dots are more separated, and even worse, the dots at the bottom are largely the same color, the same when we look at each line vertically. This means that each user gets a stationary like/follower ratio over time, which is what we should avoid, cause this will cause information leaks and the model can simply learn to give some users low scores and other users high scores to beat the system, and this defeats the goal for the system to learn only the correlation between images vs likes, not ones posted them. With this figured out, I decided to simply use the most liked post as the standardizer so that all pictures will have a score between 0 and 1, which surprisingly helped during the training phase, allowing a faster convergence.

## 4.2 Model Training

To get the best performance, I started with 3 convolution layers instead of 2, and the filter sizes were a lot larger than what they were mentioned in section 3.5. The use of the validation set here is really important as it prevents me to tune the model based on the test set. After the structure is settled, here below is the Learning curve. As epochs increase, there was some fluctuation in the training loss, however, there's no sign of over-fitting when looking at the validation loss curve, as it doesn't increase, which is something that will happen if we used too many epochs.

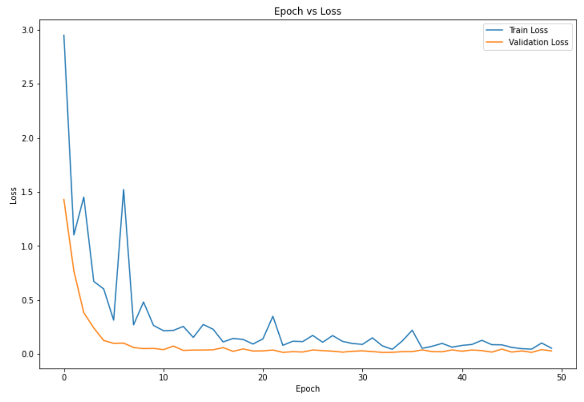
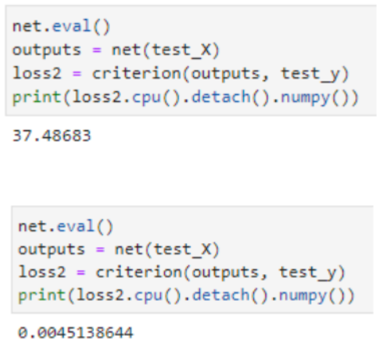


Figure 8. Loss vs Epoch



```
net.eval()
outputs = net(test_X)
loss2 = criterion(outputs, test_y)
print(loss2.cpu().detach().numpy())
37.48683
```

```
net.eval()
outputs = net(test_X)
loss2 = criterion(outputs, test_y)
print(loss2.cpu().detach().numpy())
0.0045138644
```

Figure 9. Change in Test Loss, before(Top), After (Bottom)

*Table 1.* Ranking accuracy

| ITEMS IN SET | NN MODEL | BASELINE |
|---|---|---|
| 2 | 0.78391 | 0.49964 |
| 5 | 0.52312 | 0.19981 |
| 10 | 0.24123 | 0.10041 |

The above curve would be meaningless if the test loss doesn't perform well. Luckily, that's not the case. As shown below, while left completely out of the training process, the test loss has a drastic decrease, which means the model is performing extremely well in terms of generalizability.

As for the baseline, the linear regression model got absolutely overwhelmed by the task. Not only did it take forever to train (since scikit-learn does not have support for GPU acceleration), but while having a reported accuracy of 0.9998 for training data, the mean squared error was calculated to be 50,220,844,083,300.55. This is a sign of a huge amount of over-fitting going on here.

Lastly, a custom function was used to show how good this model is at making the right predictions. It will be considered a correct ranking if an image is at the actual position. For example, if the correct order of five images is [0, 4, 2, 3, 1] and the actual ranking is [0, 1, 2, 3, 4], it will have a score of 0.6. The performance is shown above. The baseline is basically a random guess, with 2 being around 50%, 5 being 20%, and 10 being 10%. Compared to the results from the baseline, the nn model performed a lot better, despite there still being a lot of room for improvement.

## 5 CONCLUSION

Overall I think the model performed better than I was expecting, especially given the amount of time given to implement it. Throughout the project, not only did I collect my own database, but had some hands-on experience with storing data in a server-based database. However, there are still a lot of things that can be done to improve this project given more time though:

- More users can be added to improve how general the model is
- More custom scoring metrics so I can evaluate the model better
- Use YOLO to extract more information from the images or even just tags so we know the subject of the shot
- Use some decay model for standardizing the follower count

These are just some ideas but I'm sure they should all make the model better. With all that said, I'm glad that I have chosen and pushed through with this topic. I've learned a lot and this is a great learning process.

## 6 TEAM CONTRIBUTIONS

Hongyu Tu is the only member of Team 31 and all the work mentioned in this paper is done by Hongyu Tu alone.

## REFERENCES

Datta, R., Joshi, D., Li, J., and Wang, J. Z. Studying aesthetics in photographic images using a computational approach. In Leonardis, A., Bischof, H., and Pinz, A. (eds.), *Computer Vision – ECCV 2006*, pp. 288–301, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33837-6.

Kormann, D., Dunker, P., and Paduschek, R. Automatic rating and selection of digital photographs. In Chua, T.-S., Kompatsiaris, Y., Mérialdo, B., Haas, W., Thallinger, G., and Bailer, W. (eds.), *Semantic Multimedia*, pp. 192–195, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-10543-2.