Training a neural network for image classification using Google Image Search

Aucke Bos

s4591496

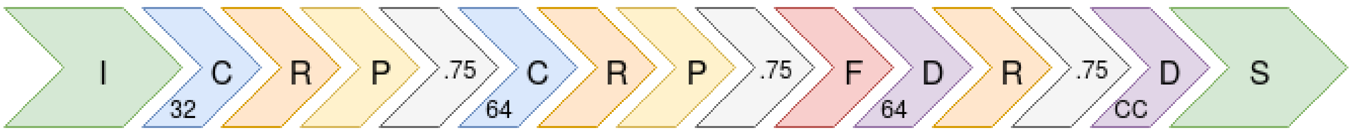


Figure 1. Structure of the neural network used in this project. C_{XX} is a convolutional layer with XX filters, R is a Rectified Linear Unit, P is a pooling layer with poolsize (3,3), .75 is a dropout layer of rate 25%, F is a flatten layer, D_{XX} is a dense layer with output size XX, CC is classcount, S is a sigmoid output layer.

Abstract

A problem for many data science related tasks is the need for a suitable dataset. For many research projects it is hard to find such a set, and sometimes a set that fits the needs is not even publicly available. This project is focused purely on datasets containing images. Its goal is to investigate whether it is possible to create and train a neural network by using images crawled from Google Image Search. As will be elaborated upon, this goal is too general for most cases, resulting in a network with low accuracy. When the context would be narrowed down, e.g. by trying to create a network for some specific image types, it might be possible to get a network with high enough accuracy.

1 Introduction

Neural networks can be trained on arbitrarily sized datasets. However, their performance generally increases with the size of training data they are given [3]. This is a big problem in their suitability for certain image classification tasks. They might only perform well enough on datasets too large to be found on the public internet.

This problem was encountered while performing research for this project on the original subject. A short summary of this project is not otiose; the idea consists of two parts. Firstly a neural network would need to be trained to recognise alcoholic beverages in pictures. These pictures would be retrieved from a dataset containing tweets with corresponding pictures. Secondly an algorithm would be built to estimate someones age based on the textual content of what they tweet. These two parts are then combined to investigate alcohol usage under different age groups.

While this original idea seemed interesting though challenging, one of the encountered problems set out a new topic, resulting in this paper: very large datasets containing tweeted pictures are hard to find on the internet. The main reasons for this are ethical and privacy issues. After some research I decided this research would not be feasible given the scope of the course.

Therefor I have decided to change the subject of the project completely. I have tried to find an answer to the question: "*Would a neural network trained solely on pictures queried from Google Image Search be able to classify images accurately enough so that the network would be usable in practice?*". If this would be possible, such a network could be used for the first part of the original research. This would make the complete project a lot more likely to succeed, since only textual tweets are needed for the second part. Such a dataset is easier to find, and a lot of them can be found online [1]. For the project I have used a Python library called iCrawler [2] to create 'datasets', where the queries are used as 'ground truth' labels. The network is built using Keras [3], where after Matplotlib [4] and Plotly [5] render visual results.

In Section 2 I discuss background and related work regarding this subject. This will cover why this research is important, and why I have chosen the approach the way I did. Thereafter, in Section 3, I describe in detail how I have implemented the system. Note that the code for this project can be found on GitHub [6]. The results of my findings, including an answer to the research question, are discussed in Section 5. The paper is finalized with a conclusion in Section 6.

2 Related work

Image classification is a really hard task to perform for computer programs, whereas human are very good at it. Part of

[1] https://data.world/datasets/twitter
[2] https://pypi.org/project/icrawler/
[3] https://keras.io/
[4] https://matplotlib.org/
[5] https://plot.ly/
[6] https://github.com/AuckeBos/neural-net-by-gis

the difficulties for automatic classification is the high class variability within classes [2]. These variabilities mainly occur due to surroundings and picture angles. For computers, it is hard to distinguish between information that is characteristic for the class an image belongs to, and information that is 'noise' and should not play a major role in classifying an image. One solution to this problem is to feed the program more samples to train on. With many samples, a computer should be able to find the general characteristic that hold for most of the pictures of a class, and discard the noisy features that it finds.

Obviously it is not always possible to get more samples to train on. A solution that partly solves this problem is *data augmentation*, which increases the size of the data by creating augmented duplicates, by using for example rotation or shifting of images. Data augmentation is used a lot in research about image classification, and has proven to be a good tool to increase classification accuracy [6]. Data augmentation is therefor also applied in this project.

Although data augmentation helps a neural network a lot, there still needs to be a reasonably large dataset to begin with. Many research papers use the ImageNet dataset [7], on which publicly available neural networks have been trained a lot. This is a good and easy way to get access to a large but general dataset or network. However this approach might be too general for some specific classification tasks.

Here we find the core of the motivation for this research. Searching for images on the web is a labour intesive task for humans [4], and Google Image Search (GIS) is for a lot a people a good tool for this. GIS does not store its own images, but rather provides a database of images that are retrieved from across the internet. It hereby provides a good source for the creation of datasets consisting of images. This project therefor uses a scraper that submits queries to GIS, and downloads large numbers of images for each query, where each query has its own 'cluster'. Using this crawler one can create his own dataset containing only images needed and suitable for their specific research.

An interesting view of what we are doing here arises when we look at how the input of our network is retrieved. Since Google crawls images from the internet, and it is able to provide images matching a textual description, GIS is in itself a type of image classifier. We are thus using the output of one image classifier as the input of our own classifier, with some preprocessing in between. This might feel like an odd thing to do at first, but this is actually quite common in the task of classification. An example of this classifier combination is the usage of different classifiers for face recognition, as described in [5]. As they explain, different classifiers might be sensitive to different features of images an clusters. This is the case at a lower level in a convolutional neural network as well: layers deeper in the network are sensitive to very different aspects of pictures than earlier layers of the network. Ofcourse we do not have access to the internal workings of GIS, and thus we do not know how image features are built up and recognised. It is however entirely possible that our network finds very different image features, hereby giving a good result if the two classifiers are 'combined' like this.

If then a suitable dataset is created using GISw, a system should be chosen to perform the task of classification. As mentioned, this image classification is a hard task for computers, while, although labour intensive, it is relatively easy for humans. Neural networks are a good approach for this task, since they work like the human brain in some way [1]. It is shown in literature that these networks perform well in practice [7]. Keras is a Python module that provides an easy to use framework to build such networks with little effort. This project therefor uses this framework to create the image classifier explained in the next section.

3 Approach

The result of this project is a command-line tool built in Python 3.6, and is able to perform four different tasks regarding the training of a neural network. Each of these commands is discussed in detail below. For each of them, I elaborate on both why and how these commands have been implemented.

3.1 Crawl terms

As mentioned in Section 1, the original idea of this paper concerned itself with alcoholic beverage recognition. In that section I also explained why and how I have changed the goal of the project. To keep a little bit of the original goal, I wanted to measure my network's performance on beer type classification. I found an article on Wikipedia that describes what beer types exist[8]. I presumed that if one would want to use my network to train on other clustering types later on, it would be useful to be able to extract these 'clusters' from a Wikipedia article. Therefor the classifier is able to parse a webpage, extract its tables, and output a file containing a user selected column of a user selected table.

To create a file `beertypes.txt` containing each beer type on a separate row, one should execute

```
$ python3 classifier.py crawl_terms
        https://en.wikipedia.org/wiki/
        List_of_beer_styles –o beertypes.txt
```

and select the first column of the first table.

3.2 Get images

After a list of terms is created, these terms need to be submitted to GIS. The user can define how many images need to be retrieved per query. GIS restricts a limit of a thousand

[7] http://www.image-net.org/

[8] https://en.wikipedia.org/wiki/List_of_beer_styles

images per query. Therefor the program adds a daterange filter to the query. If the user wants more than a thousand images per cluster, the program queries the term per thousand items, where each query has a date filter of a year, starting at the current date going backwards. Each term then gets its own sub directory in the output folder, where the images are saved in the original format. This way of storing the data is compatible with the ImageGenerator[9] of Keras, which is used when training the network in the next step.

To retrieve 250 images per beertype and save them in the folder `beertypes`, one should execute

$ python3 classifier.py get_images beertypes.txt beertypes −c 250

3.3 Create network

Once the dataset is created, one can start training the neural network. The program uses the base class NeuralNetwork for this, which currently has two specific implementations: SimpleCnn and CopiedCnn. It uses SimpleCnn by default, since it is built and tweaked to provide better results for images crawled from GIS. Visual representation of this rather simple network is given in Figure 1. The network is kept simple because more complex models showed to give less accurate outputs while needing longer training time. CopiedCnn was used as a basis for this implementation, which is an network copied from a blog post found online[10]. One can easily extend the program with a new neural network, as it only requires one definition to be defined: `build_network`.

When this command is ran, the network will be created and trained on the provided input parameters. The parameters define both how the input should be parsed, and with which parameters the network should be trained:

- *Image width, height, and depth.* The network needs to have a format to which all images will be reshaped, as the network needs all input to be of the same format. If images are of another format, they will be reshaped automatically.
- *Input.* The directory holding the data. Each image should have its own subdirectory. The network will create a class for each of the folders, and uses one-hot-encoding to encode the true labels. It uses 80% of the data for training; the rest is used for validation and updating the weights of the network accordingly.
- *Learning rate.* The learning rate that is applied when updating the network weights during the training step. The network will however reduce the rate when it notices that the accuracy is not increasing anymore.
- *Epochs.* The number of epochs used during the training step. One epoch is one pass forward and backwards of the entiry dataset through the network. In general, more epochs means higher accuracy but longer training time. The network stops early if it notices that the accuracy is no longer increasing.
- *Batch size and image generator batch size.* The batch size defines the sizes of the batches that are passed through the network. In practice, it showed longer training time but significant increase in accuracy for smaller batch sizes. The image generator batch size defines how many images are passed to the network as one 'item' for the training step. The larger this value, the more images each batch will contain, the higher the accuracy and slower the training.
- *Patience for early stopping and reducing the learning rate.* As mentioned, when the network notices that accuracy is not increasing anymore for a certain period, the learning rate will be reduced and eventually the training will be terminated. These two values define the patience of the network before that happens.

For many usecases the default parameters will give fine results, but the user is given the option to tweak the neural network to his dataset. When the command is run, the network is created and trained, and saved in a .h5 file. The history of accuracy while training is also saved in json format.

To create a network for the created beertypes using the default parameters, one should execute

$ python3 classifier.py create_network beertypes

3.4 Create plots

To give the user visual feedback on the performance of the trained network, the program is able to provide two different plots.

- The accuracy plot shows the accuracy of the network measured after each epoch. It uses the .json file that is saved upon creation. The plot can be used to measure performance during the training step.
- The scatter plot can be used to see whether the network is able to distinguish the different elements correctly. The accuracy is a better proof of performance, but the scatter plot can help significantly to visualize its performance. Because the network outputs vectors of size equal to the number of dimensions, dimension reduction is performed if needed. This reduction is done using Linear discriminant analysis[11] with the package `sklearn`[12]. The user can define the number of dimensions of the plot (2 or 3) and the number of elements to plot. One can define the network as 'good' if intra cluster distance in low, and inter cluster distance is high. If this is the case, a clustering algorithm like for example kmeans would

[9] https://keras.io/preprocessing/image/
[10] https://towardsdatascience.com/classify-butterfly-images-with-deep-learning-in-keras-b3101fe0f98
[11] https://en.wikipedia.org/wiki/Linear_discriminant_analysis
[12] https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

be able to cluster the output vectors quite good, which is an indication that the network did well.

To create these plots for the beertype network, one should execute

```
$ python3 classifier.py create_plots beertypes.h5
    beertypes
```

Provided that the .h5 en .json files have been renamed to 'beertypes'.

4 Results

When training a network on different datasets, it showed that it was really hard to classify the clusters correctly. Accuracy levels were disappointing, and these values we substantiated by the scatter plots. The results for different datasets are discussed in this section. In the next section, I will state some explanations for the disappointing results.

- I started by training the network on the dataset of 45 different beer types. The first problem I encountered: training such a network takes very long. But after the training was completed I also noticed that the network was unable to get its accuracy above ~5%. This is obviously too low to be usable, which also became clear when looking at the scatter plot. It almost seemed like the network had created random output for each picture.
- Next up was to train a network on very simple data, to check whether the network worked at all. Two clusters containing each about fifty pictures of the query 'red' and 'blue' respectively. These queries should result in images so different in both classes, that the network had to be able to classify these correctly. Fortunately this happened to be true. The plots scan be found in Figure 2. These show that the network reaches validation accuracy of a 100% after about 17 epochs, and the cluster plot shows a high inter cluster distance.
- With the next dataset I wanted to test how the network performed on more than two clusters. It contains ~100 images of seven different colors. The plots are shown in Figure 3. It is clear that again the performance is not very good: an accuracy of 16% is achieved. When we look at the scatter plot we see that the output is not very good, but also definitely not random. The intra cluster distance seems reasonably low, although the clusters seem to overlap a lot.
- This next test has again only two clusters, but these are less easy to distinguish; the queries are 'beer beverage' and 'wine' (The word 'beverage' is added to the first query because of the meaning of 'beer' in Dutch, one can imagine the noise this results in). The plots are show in Figure 4. The accuracy is again disappointing for a clustering with only two clusters: less than 64%. The cluster plot however shows quite OK separations. There are some images that are clustered 'wrongly': they live near the images of the other cluster. But many of the images are clustered close to other images of that cluster. So this network might still be useful for some cases.
- Now to still perform some beer clustering, I crawled around a thousand images of each of the first five beer types found on the Wikipedia page. The plots are shown in Figure 5. An accuracy of ~30% seems quite bad at first, but looking at the type of images this might be quite OK; more on this in the next section. The scatter plot again shows quite low intra class distance, however the space of the classes overlaps a lot.

5 Discussion

In general, the network performs worse than I had hoped beforehand. I therefor tried different, more complex networks on the same data. However all of them either needed an unusable amount of time to train, gave much worse results, or both. I have found some possible causes for these bad results. I state them below with decreasing influence:

1. The goal of this project might be too general. Training a neural network is an expensive task that generally needs a lot of data. Creating a system that can create a neural network for any number of any query type might simply be an infeasible task. On top of that, it is better to create a network for each task specifically, such that it can be tweaked to work best on the dataset at hand. One can for example make a network focused on detecting shapes, to recognize buildings in pictures. A classifier with the goal to recognize different fruits would need to be focused more on colors than on shapes.
2. The results of GIS are quite varying. When retrieving more than 50 pictures for a query, the 'noise' starts to increase quite rapidly. In my test datasets I have found a lot of pictures that show the queried item only a little bit, very vague, in the background, or only in text. This problem is quite hard to overcome, since we cannot influency the way Google retrieves its images. Simply adding more images might help a little bit, but more images will also mean relatively more noise, since the lower the results in the returned list, the lower their 'similarity' with the query.
3. Some queries are simply too hard to cluster. Even two very different beer types can result in very alike pictures. Beer is still poured in the same glass type or bottle, and still has quite a similar colour.
4. The network might simply need a lot more data to train on, or need a lot more epochs. I trained the network on my personal laptop, using Keras in GPU mode on a Nvidia GeForce GTX 1050. If one uses a more powerful computer or graphical card, it might be feasible to train the network on a lot more data, or propagate the dataset more times, or perform more data augmentation.

6 Conclusion

The plots in Appendix A show that the accuracy of the neural network is in many cases not high enough to be usable in practice. However, when one makes sure the input does not consist of too many classes, and the classes are 'different' enough, you might get a usable result. Next to that, it makes sense that it's not feasible to build a system that can build a well working neural network in only little time on any input data. Neural networks simply need a lot of computational power, which has always been one of their big bottlenecks [3]. So unfortunately the answer to the research question is 'no' for most of the cases I've tested the network upon. If one would create a network for each dataset specifically, the results will probably increase significantly. Fortunately the current program allows extension of new classes very easily, making it still useful for any future work on this subject.

References

[1] D. Ciregan, U. Meier, and J. Schmidhuber. "Multi-column deep neural networks for image classification". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. June 2012, pp. 3642–3649. DOI: 10.1109/CVPR.2012.6248110.

[2] Dan C. Cireundefinedan et al. "Flexible, High Performance Convolutional Neural Networks for Image Classification". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*. IJCAI'11. Barcelona, Catalonia, Spain: AAAI Press, 2011, pp. 1237–1242. ISBN: 9781577355144.

[3] Andrew Howard. "Some Improvements on Deep Convolutional Neural Network Based Image Classification". In: (Dec. 2013).

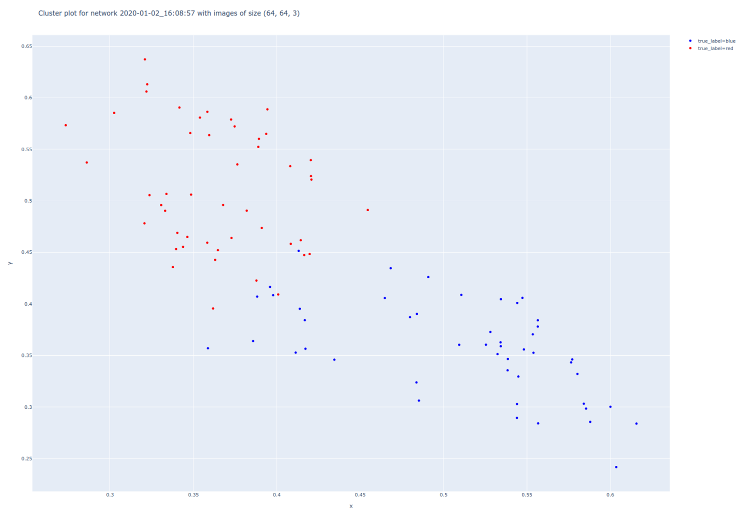
[4] Hao Liu et al. "Effective browsing of web image search results". In: Jan. 2004, pp. 84–90. DOI: 10.1145/1026711.1026726.

[5] Xiaoguang Lu, Yunhong Wang, and Anil K. Jain. "Combining classifiers for face recognition". In: *2003 International Conference on Multimedia and Expo. ICME '03. Proceedings (Cat. No.03TH8698)* 3 (2003), pp. III–13.

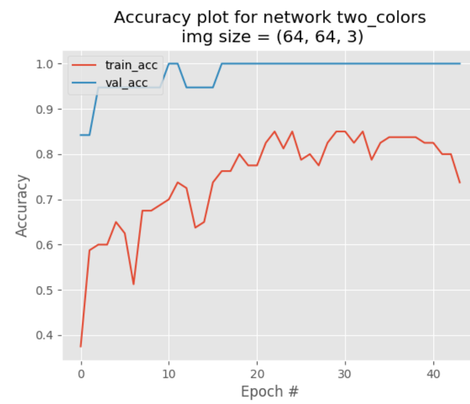
[6] Luis Perez and Jason Wang. "The Effectiveness of Data Augmentation in Image Classification using Deep Learning". In: (Dec. 2017).

[7] James A. Shine and Daniel B. Carr. "A COMPARISON OF CLASSIFICATION METHODS FOR LARGE IMAGERY DATA SETS". In: 2002.

A Scatter and accuracy plots for different test datasets

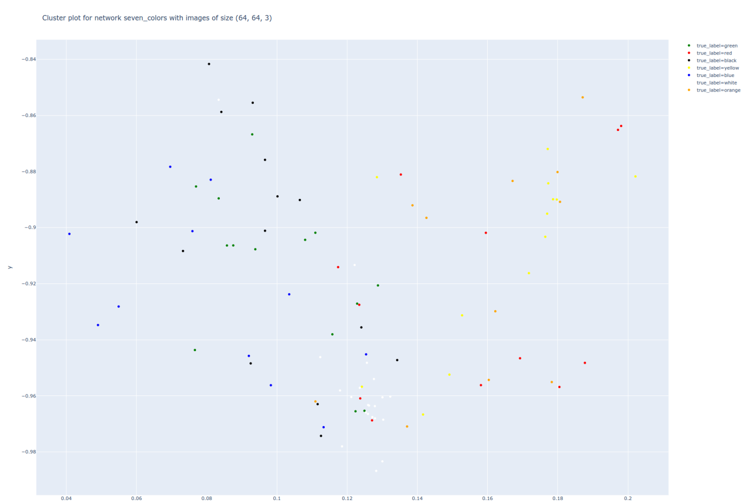


(a) Scatter plot

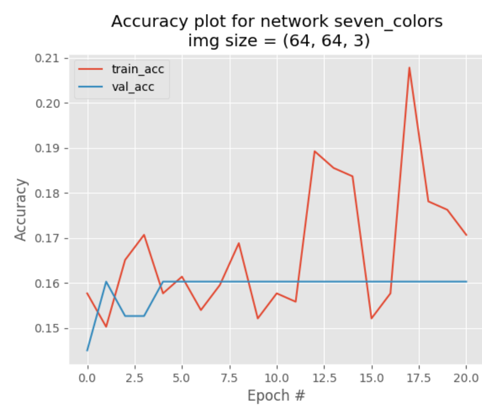


(b) Accuracy plot

Figure 2. Scatter and accuracy plots for a dataset of two colors of ~50 pictures each.

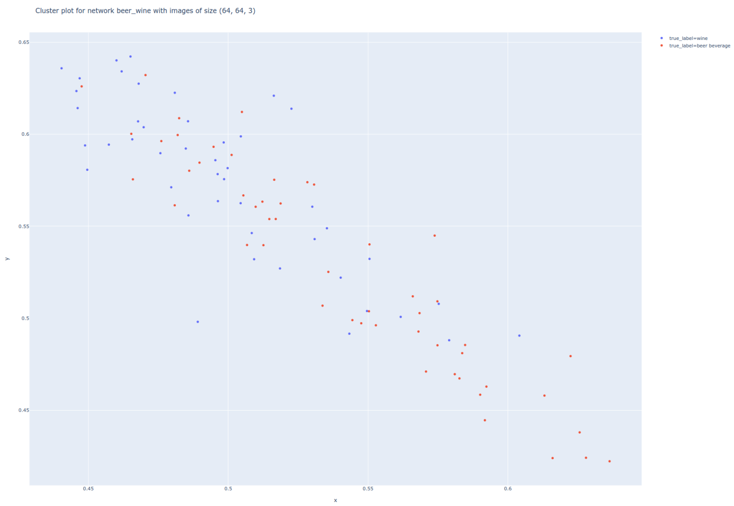


(a) Scatter plot

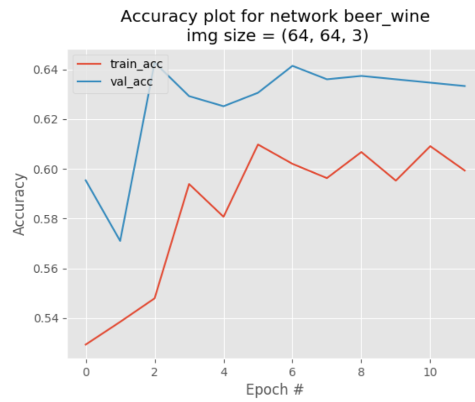


(b) Accuracy plot

Figure 3. Scatter and accuracy plots for a dataset of seven colors of ~100 pictures each.

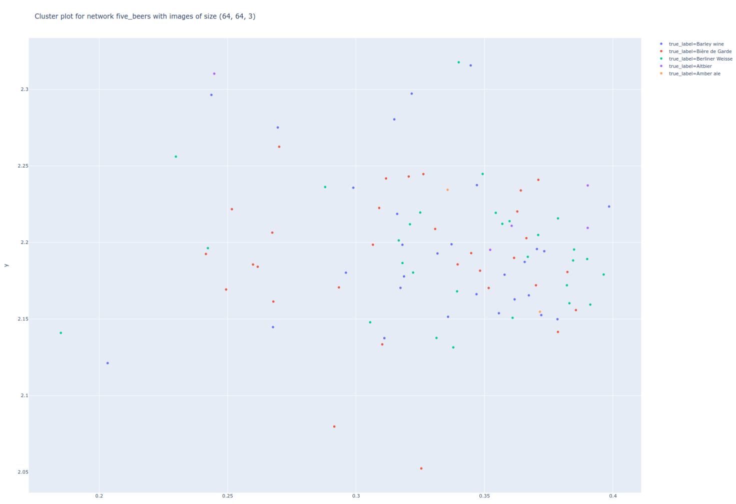


(a) Scatter plot

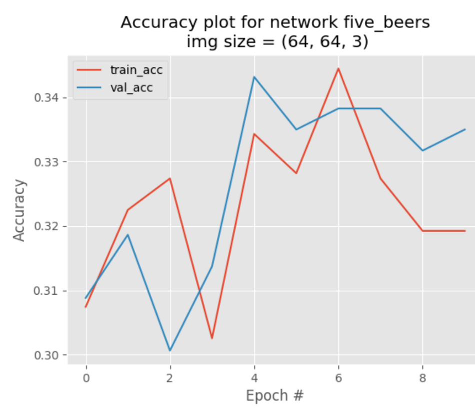


(b) Accuracy plot

Figure 4. Scatter and accuracy plots for a dataset of two clusters of ~2000 pictures each.



(a) Scatter plot



(b) Accuracy plot

Figure 5. Scatter and accuracy plots for a dataset of five beer types of ~100 pictures each.