Image Classification using Convolutional Neural Network

Jonathan Bhaskar
jbhaskar@sfu.ca

Ankit Patel
ampatel@sfu.ca

Abstract

Convolutional Neural Networks (CNNs) have been established as a powerful class of models for image recognition problems. Inspired by a blog post [1], we tried to predict the probability of an image getting a high number of likes on Instagram. We modified a pre-trained AlexNet ImageNet CNN model using Caffe on a new dataset of Instagram images with hashtag 'me' to predict the likability of photos. We achieved a cross validation accuracy of 60% and a test accuracy of 57% using different approaches. Even though this task is difficult because of the inherent noise in the data, we were able to train the model to identify certain characteristics of photos which result in more likes.

Problem Definition

The popularity of a photo uploaded on social media like Facebook or Instagram is quantified by the number of likes it gets. In this project, we try to explore the notion that the characteristics of a photo might make it more "likeable". Lately Convolutional Neural Networks are being used vastly in Computer Vision space to recognize things, places and people in personal photos, signs, people and lights in self-driving cars, crops, forests and traffic in aerial imagery, various anomalies in medical images and all kinds of other useful things. We train a CNN classification model on Instagram images, grouping the images into two sets of "good" and "bad" based on a normalized count of the number of likes an image got. We used Caffe to perform this. It is one of the popular packages that is widely used for developing deep convolutional neural network models for image and video processing applications.

Methodology

Data Collection

The first script we wrote was to connect to the Instagram API through an access token and download information for images uploaded. We restricted ourselves to images that had the hash tag #me, which we assumed would contain photos of a person. We also restricted ourselves to images that are online for at least a month to allow it to have enough likes. Unfortunately, this metadata did not have user information – most importantly, the number of followers a user had. We needed this information to normalize the likes of a photo. Instead of just the number of likes, if we could get the ratio of the no. of likes to the no. of views, that would give us a better estimate of how popular an image was. The no. of views could roughly be quantified by the no. of followers a user has.

Next, we got a unique list of users from the image information. We had to get the user information for each of these users from another endpoint which contained the no. of followers a user had. For this endpoint, Instagram permits 5000 API hits per access token. In order to parallelize the process, we collected multiple access tokens by setting up the OAuth protocol and asking our friends to login. Our first attempt used the requests library in Python to interact with the API. But requests in python is synchronous which means the execution is blocked until the server responds. So if you want to make thousand requests, you will have to do one request after the other one. Each request was taking approximately 1 second, which means it would take days to collect enough data.

Considering this limitation, we started exploring non-blocking I/O libraries like Node.js. Node.js provides an event-driven architecture and a non-blocking I/O. Using Node.js, we were able to hit the API much faster. We collected information of 100000 users.

Data Processing

These users had uploaded over 200000 images. Celebrities tend to have large number of followers and a new user will have less number of followers and therefore will likely have almost no likes. To make our training unbiased to such extreme cases, we decided to filter out users with very high and low number of followers. On observing the distribution of number of followers, we found that Standard deviation of the distribution was 5 times the mean. This implies that the data still had many outliers and the SD is not a good estimate of the central tendency. We then found the median and used the Median Absolute Deviation (MAD) estimate instead of SD in order to filter the images. This brought down the number of images to 120000.

The image information had the URL. We used node.js to download all 120000 images. Here, we hit the limit on the number of files that can be open at the same time, as node.js was continuously opening files and waiting for the response from the URL to write to those files. We had to change the ulimit of the OS to get over this. Another problem we faced was node.js starting too many workers that caused the OS to stop responding. To overcome this, we wrote a loop in our code that restricted the no. of workers to a value we could select. We also had a few problems with programming for concurrency but we were able to work around it. All of our scripts had a computational complexity of O(n) and a space complexity of O(n).

Image filtering

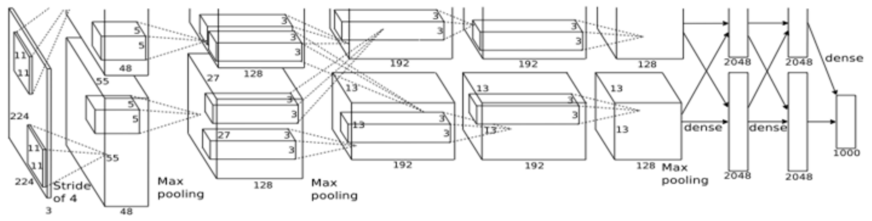
On eye-balling the collected images, we found that most of the images weren't photos of people. There were a lot of pictures of quotes and the like. We certainly wanted to filter these images to make the dataset less noisy. We applied OpenCV [1] face recognition algorithm to detect human face/eye in the image and narrowed down to just photos of people. In spite of running on multiple cores, OpenCV took 10 minute for 1000 images. It would take a day or more to classify all 120000 images. Therefore, we looked for alternatives and found CCV [2], a modern computer vision library. CCV uses a 2014 algorithm to detect faces rather than the decade old one used by OpenCV and has a 30% greater accuracy. It was able to classify 1000 images in less than a minute.

CCV is written in C and does not have a python wrapper. Instead, it has a HTTP API. We ran it on an 8 core machine and using synchronous python requests did not make use of the full computational power of the machine. So we used python's GRequests. GRequests is the requests library with gevent which allows making asynchronous HTTP Requests. We processed 20 images in a batch.

CCV filtered 60000 good images from the 120000 provided. We ran the steps mentioned after this but found that even this was too noisy with a lot of false positives. The machine learning model did not work very well with the false positives. CCV had a confidence rating for it's classification. As we had enough data, we filtered the images further by using a higher confidence rating even though this resulted in false negatives. Our final dataset had 35000 good images with very few false positives. The space complexity for this part was O(n).

Modelling

We took the ratio of number of likes and followers for each image, sorted the images based on this ratio our data into two halves, one above 50 percentile of this ratio as good images (assigned binary 1 as label) and other half as bad images (assigned binary 0 as label).



We attempted to transfer knowledge from AlexNet ImageNet CNN [3]. As shown in figure above, it has 5 convolutional layers and 3 fully connected layers. We applied this model to our cleaned image dataset and extracted FC6 and FC7 activation features, each of 4096 dimensions. We distributed the Caffe computation on CPUs available on Spark framework to speed up the feature extraction process. We performed data augmentation on collected features by considering horizontally flipped (mirrored) images.

The Scikit-learn python package contains a handful of basic machine learning algorithms ranging from regression/classification to unsupervised clustering/dimensionality reduction models. The package contains many algorithms, however, the algorithms proposed were not designed for scalable solutions. Spark's Machine Learning Library (MLlib) provides a wide range of scalable machine learning models. The main difference between Spark MLlib and Scikit-learn is its ability of parallelism over a commodity cluster. With Spark MLlib, we are not restricted to datasets that can fit into one machine's memory or its computation power. We can extend it to a cluster of computers and speed up the training process of a machine learning model. We applied algorithms from both scikit learn and MLlib libraries to our dataset.

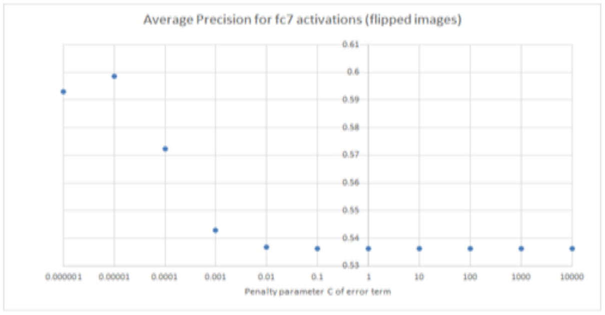
Firstly, we tried Linear SVM classifier. We cross validated regularization parameter C from 10^{-6} to 10^{5} and evaluated the classifier's performance using average precision. Then we applied one of the ensemble methods, Random Forest Classifier. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. We took 500 estimators to train random forest.

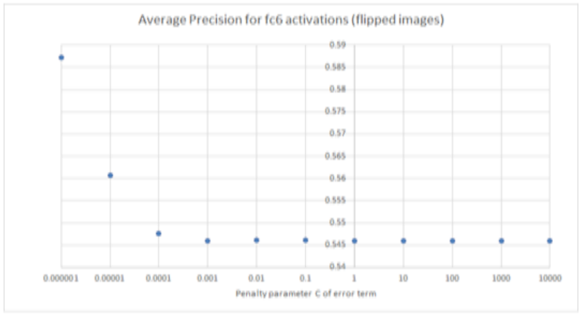
As our dataset grew, we applied MLlib SVM and let it run on the cluster. It trains a SVM using Stochastic Gradient Descent. We cross validated regularization parameter and step size and chose the one that gave the best cross validation accuracy. The results we got are as follows:

FC7 activations average precision		
	(flipped+normal) image activations	normal activations
sklearn - LinearSVC	0.599	0.597
sklearn - RandomForestClassifier	0.587	0.579
Mllib - SVMWithSGD	-	0.6

FC6 activations average precision		
	(flipped+normal) image activations	normal activations
sklearn - LinearSVC	0.587	0.603
sklearn - RandomForestClassifier	0.578	0.579
Mllib - SVMWithSGD	-	0.598

For linear SVM using scikit learn, we cross validated different values of C and the plot for both FC6 and FC7 flipped + normal features are as follows.





For MLlib SVM with SGD, we got the best precision with regularizer = 10^{-4} and step-size = 1.

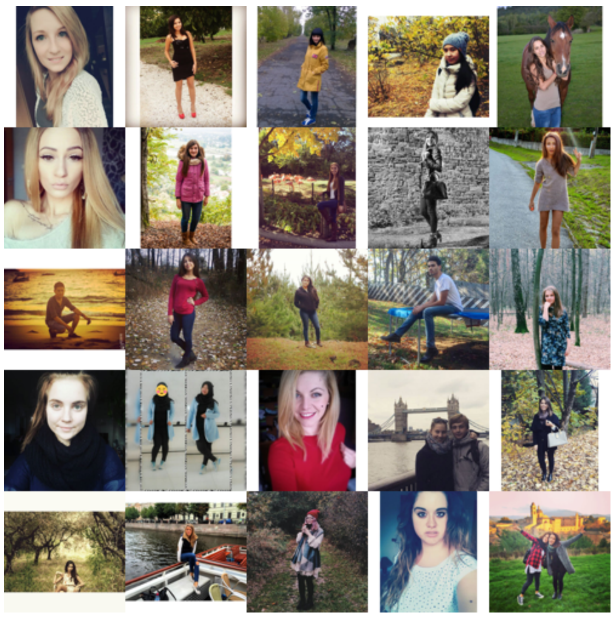
Fine tuning

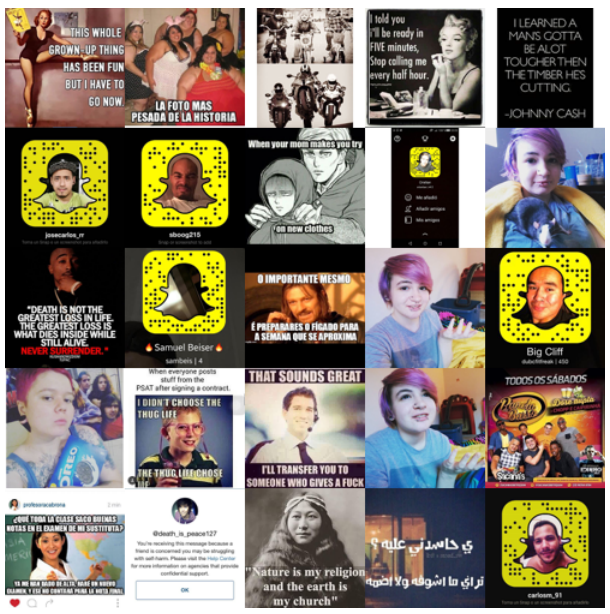
We then tried fine-tuning the AlexNet model with Caffe to our data. Fine-tuning takes an already learned model, adapts the architecture, and resumes training from the already learned model weights. The benefit of such approach is that, since pre-trained networks are learned on a large set of images, the intermediate layers capture the "semantics" of the general visual appearance.

We divided our dataset into train and test in the ratio 3:1. We changed the *train_val.prototxt* to read images from our directory instead of the default LMDB Caffe uses for input. We trained the model for 35000 iterations with a starting learning rate of 0.001. We set the learning rate to decrease by a factor of 0.1 every 5000 iterations i.e. after the first 5000 iterations, the learning rate would decrease to 0.0001, 0.00001 after the next 5000 iterations and so on. This way we decrease the learning rate after the loss stagnates after a number of iterations. We also changed the fc8 layer to work with 2 classes instead of the 20 classes the default model outputs. With an input of 7.5 GB, the RAM used did not cross 12 GB at any point of time, so the space complexity was O(n).

We trained the Caffe model on the train data and once it is trained we passed test images through the model, sorted these images based on the output probability, and took the top and bottom 25 images to see if there is any trend that ConvNet model has learned from training.

Results




THIS WHOLE GROWN UP THING HAS BEEN FUN BUT I HAVE TO GO NOW
LA FOTO MAS PESADA DE LA HISTORIA
I told you I'll be ready in FIVE minutes, Stop calling me every half hour.
I LEARNED A MANS GOTTA BE ALOT TOUGHER THEN THE TIMBER HES CUTTING -JOHNNY CASH
When your mom makes you try on new clothes
"DEATH IS NOT THE GREATEST LOSS IN LIFE. THE GREATEST LOSS IS WHAT DIES INSIDE WHILE STILL ALIVE. NEVER SURRENDER."
O IMPORTANTE MESMO É PREPARARES O FÍGADO PARA A SEMANA QUE SE APROXIMA
Big Cliff
I DIDN'T CHOOSE THE THUG LIFE THE THUG LIFE CHOSE
THAT SOUNDS GREAT I'LL TRANSFER YOU TO SOMEONE WHO GIVES A FUCK
TODOS OS SÁBADOS
"Nature is my religion and the earth is my church"

The best 25 images are to the left, and the worst to the right. The differences are easy to see. Almost all of the top 25 images have women, and almost all of them are Caucasian. But there seem to be two groups here. The first group are close up photos of the person. The second group is more interesting. They are photos taking in locations traditionally considered to be beautiful, either with a lot of nature (greenery, water, etc.) or with famous landmarks. The model seems to have learned that photos taken in beautiful locations get more likes. This was a surprising outcome we did not expect at all. It suggests that even with the noisy data and not so impressing accuracies, the model actually seems to work.

The bottom 25 images are mostly memes which I guess are not that funny. The model also seems to hate one particular person for some reason.

Web Interface

Caffe provides a default web interface. We had to modify it to work with our model. The HTML also needed to be modified as we only output two classes. The default web application also uses bet_file weights which helps when there are a large number of classes. We had to remove it from the web application.

The web interface is accessible at http://54.175.110.234:5000/

Future Work:

1. Filtering of images with persons using ImageNet itself. A major problem was dealing with the false negatives and positives for which I'm sure a properly trained CNN would do a much better job.
2. Obtaining more training data. We had a hard time because of Instagram's rate limiting. Given more time, we could get more data which would result in a better model because of the noise.
3. Using a better base CNN model. We used the default Caffe CNN as the base model, but there are other models like VGG which are better by more than 10% accuracy.

References:

[1] Andrej Karpathy. "What a Deep Neural Network thinks about your #selfie." http://karpathy.github.io/2015/10/25/selfie/ (2008).
[2] Bradski, Gary R., and Adrian Kaehler. "Learning OpenCV: computer vision with the OpenCV library/." (2008).
[3] Li, Jianguo, and Yimin Zhang. "Learning surf cascade for fast and accurate object detection." *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE, 2013.
[4] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.