Are Anime Cartoons?

Chai Jia Xun, Haritha Ramesh, Justin Yeo
Stanford University
450 Serra Mall, Stanford, CA 94305
chaijiaxun@gmail.com, haritha1211@gmail.com, yeozx@u.nus.edu

Abstract

The realm of popular culture is full of debate: Are ninjas or pirates better? Is the Flash faster than Superman? Which character pairing is the best? This project targets the age-old debate: Are anime cartoons? More specifically, we wish to see if a machine can learn to differentiate between anime and cartoon.

For this project, we collected 3600 screenshots of popular cartoons and anime. We then attempted 2 methods of classifying them. First, we used a rudimentary neural network, training on 1200 images and testing on 400. This got us test accuracy averaging at about 58%, which was significantly low. Realizing the critical flaws in our method, we later used a convolutional neural network, training on 2700 images and testing on 900. This brought our test accuracy up to about 92%, a number significantly higher than a random guess. This accuracy, being the rough average across multiple sets of hyperparameters, shows that there are significant, machine-learnable differentiating features between anime and cartoon. In other words, we can now provide popular culture with some evidence that anime are different from cartoons.

1. Introduction

Image classification is a common problem in the field of Computer Vision and Machine Learning. For many different reasons, we want to see computers becoming more able to do the things that humans can do – and perhaps do them even better. Yet, through this process, we actually discover that computers can tell us things about our datasets that we may not have known or consciously realized. For example, neural networks can discover new features in training, and in extreme cases like Google Brain, even create cryptographic algorithms we do not understand.

For this project, we wanted to explore this characteristic of neural networks, and do so in the fun context of pop culture references. If a machine could accurately classify images into anime or cartoon, it would mean that there are sufficient features that identify each category of animation. To be more specific: if our classifier succeeded, we provide the internet with some proof that anime and cartoons are different. On the other hand, if it fails, we conclude nothing because another method may succeed (and the debate carries on). We started the project with a desired accuracy of 90% - although this number is arbitrary, we believe that achieving this level of accuracy should be convincing enough for our goal.

While our initial implementation using 1600 images and a basic neural network only obtained about 58% test accuracy, our later implementation using 3600 images and a convolutional neural network managed to obtain about 92% accuracy. Since we managed to reach our goal of 90% accuracy, we can convincingly say that anime have significant differentiating features from cartoons.

2. Related Work

Although there are no existing studies matching our subject topic, there exist similar research papers on image classification that include identifying or classifying traditional paintings through extraction of visual features.

For example, a paper by Karayev et. al. [5] describes the problem of identifying image or art style. The paper notes that several different types of image styles exist, including photographic techniques, composition styles, moods, genres and so on. However, as the study describes, these need not be mutually exclusive and are simply attributes to various styles. One key takeaway from this paper is that while artistic style analysis of images commonly use features such as color histograms, convolutional neural networks (CNNs) also seem to provide good results for style analyses.

Similarly, a paper by past CS229 students Blessing and Wen [1] describes the various methods they used to identify paintings by famous artists. These methods involved several types of image features such as HOG, SIFT, and SSIM, run on kernels such as the linear kernel and Radial Basis Function kernel. We will be looking to these as potential alternatives to the convolutional neural network, which we have less control over.

We used these papers as rough guides when exploring ways to improve our classifier. We also made use of several references in understanding and implementing a

convolutional neural network.

More specifically, we referred to a video by Brandon Rohrer [9] for an exemplary introduction to convolutional neural networks, and later used other references such as this page by Adit Deshpande [8] to reinforce our understanding. We also used tutorials to aid us in implementing our project. One of these was by Francois Chollet [2], aptly titled "Building powerful image classification models using very little data". The full list of references is in section 7, at the end of this paper.

3. Data Collection

We created our own dataset by downloading batches of screenshots of different cartoons and anime from Google Images and websites that aggregate such screenshots. We found a chrome extension called "Fatkun Batch Download Image" [10] that was incredibly useful for this. For consistency, we scaled each collected image to 120 by 80 pixels.

In the process of gathering these images, we had a few main criteria. First, the image had to be immediately recognizable to us as cartoon or anime – this encouraged a wide classification margin, at least to humans. Second, the image had to be a screen taken from a televised show, because other media (e.g. posters) varied much more drastically in art style. Lastly, we focused on images with at least one character in them, because we felt that backgrounds were often not immediately classifiable, even to humans.

Figure 1: Image selection criteria

Keeping these criteria in mind, we initially collected approximately 800 screenshots of anime and 800 screenshots of cartoons. Our initial classification method, the rudimentary neural network, was run on this dataset. We continued to gather more samples as we later transited to a convolutional neural network, bringing the numbers up to 1800 screenshots of each class.

Despite the increase in numbers, it was likely that our dataset was still too small. From our research, we found that convolutional neural networks needed large datasets, with many applications using about 5000 images per class. As there were no time-efficient ways for us to reach that number, we used data augmentation on our collected images – we flipped, zoomed and sheared them to create new (marginally different) training data. This allowed us to artificially increase our training set size while mitigating overfitting.

Figure 2: Data Augmentation

4. Implementation and Results

Over the course of our project, we attempted to use two different models to classify images into our two classes. We began with the rudimentary Multi-Layer Perceptron (MLP) neural network to get a basic understanding of how neural networks work. After realizing the limitations of basic neural networks with regards to image classification, we moved on to Convolutional Neural Networks (CNNs).

4.1. MLP Approach

For our initial attempts with the MLP neural network, we decided to begin with testing 2 simple approaches. The first was to just enter the raw pixel data into the neural network, and the second method was to preprocess the image by subtracting its k-means counterpart and then enter the processed data into the neural network. The intuition behind this was that cartoons generally have fewer details (or gradients) in the coloring of the characters, so the difference between the k-means image and the original image should be much less than those of the anime.

To go into the technicalities, we used the scikit-learn Python library (along with numpy for scientific computing and pillow for image processing) extensively throughout our code. We used the library to carry out our k-means pre-processing, as well as training and testing the neural networks. We ran each method on the initial dataset of 1600 images multiple times, testing with a randomly generated hold-out set of 400 images (or 25%) each time.

Figure 3: Obtaining and Subtracting K-means colors

4.2. MLP Results

The results of the tests were inconclusive. Our preliminary results gave us an average of 58% and 50% test accuracy for the raw and processed methods respectively, despite near-perfect training accuracy. Contrary to our initial belief, the k-means difference method did worse than the naive method.

Figure 4: Accuracy for the Raw RGB and Processed data

After some reflection, we realized a critical flaw in our method: it did not capture spatial relations. Using a basic neural network, we compared each individual test pixel with the matrix formed from the corresponding pixels of the training set. However, the identifying features of anime and cartoon are not just pixel values or the vector of the image as a whole - they would be the shapes of faces, gradients, or perhaps the number of colors in the image. Moreover, these features are likely to be at different positions in different images, so the basic neural network was likely to not perform well on them without a significant amount of image processing. Through this initial experimentation, we realized why convolutional neural networks are popular in computer vision - by convolving the trained features onto the test examples, we can properly extract features that can relate to our labels.

4.3. CNN Approach

Having realized that we need a spatially-aware algorithm, we went on to learn about convolutional neural networks. The first thing we had to understand was convolution, and why it was important.

4.3.1 Understanding CNN

What we came to learn was that the CNN uses nodes that contain filters of predetermined sizes. There are several such nodes in a single convolution layer, each containing a filter formed by the training data. When facing new data, the filters are convolved onto the image to give each pixel a value based on its surroundings. For example, if we had a 3 by 3 filter that sought out a diagonal, applied on a 5 by 5 image, we would get something like the following:

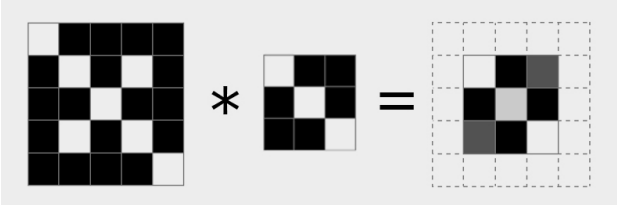

Figure 5: Visualizing Convolution

Note that the resulting "image" is smaller than the original because the filter cannot be applied at the edges. The cells of the resulting image represent how well the filter fits at that position of the image – for example, we have a perfect match at positions (1,1) and (3,3) of the original image (following 0-based indexing). This is particularly important because it means that features can be detected at any position in the image – something our basic neural network could not do!

In addition to the convolution layers, CNNs have activation and pooling layers. The activation layers modify the layer's values to increase the weightage of closely matching data, while the pooling layers reduce the spatial size of the results and make features less dependent on space. The pooling layers reduce size by taking the maximum of each small area (e.g. 2 by 2) in the representation, and retaining only that cell in the new representation (discarding 3 cells in this case).

The combination of these three types of layers, as well as the end layers that aggregate the results, allow a trained CNN to detect features in images.

4.3.2 Applying CNN

For our application, we used a framework called keras, with theano as our neural network backend. Our code referenced a blog post by the Francois Chollet on "Building powerful image classification models using very little data".

Our implementation uses 3 convolution layers, each accompanied by their activation and pooling layers. Our initial set of hyperparameters were 3 by 3 filters, 2 by 2 pool size, and convolution layers with 32, 32, and 64 features respectively.

4.4. CNN Results

The results of the tests were positive. Our preliminary run gave us a test accuracy of 91.74%. We noted that removing data augmentation brought the initial test accuracy of 91.74% down to 73%. This shows that data augmentation was significant.

We also experimented with the training set size and number of epochs to see how it affected test accuracy:

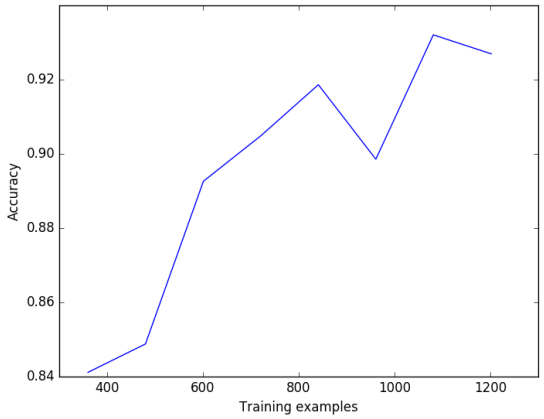

Figure 6: Accuracy vs. Training Set Size

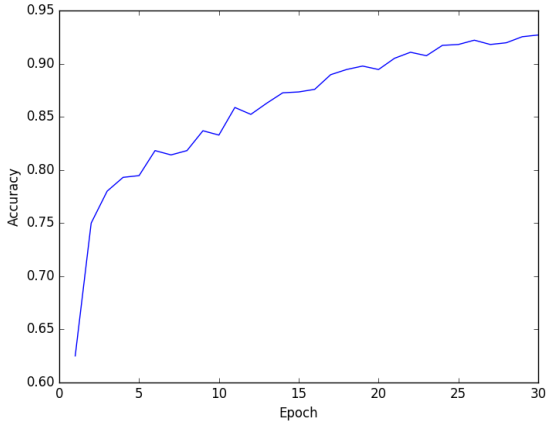

Figure 7: Accuracy vs. Epochs (Iterations)

From these results, we see that greater training set size improves test accuracy. For the number of epochs, we see accuracy improving and then tapering off at about 30. Following these tests, we decided to modify the CNN hyperparameters, to see if we could increase accuracy.

Filter Size	Pool Size	Accuracy
3	2	**91.74**
5	2	**88.87**
7	2	**82.46**
9	2	**81.80**
3	3	**92.10**
5	3	**92.33**

Table 1: Experimenting with Filters and Pooling

We modified the filter size and pool size, but only managed to get a slight increase in test accuracy when increasing the pool size.

5. Visualizations

While not immediately relevant to our results, we also spent some time attempting to visualize the layers in our CNN. We thought it might help us – and anyone reading this paper – better understand how a CNN gets to its results. These visualizations were run on an anime screenshot, and in grayscale for visual clarity.

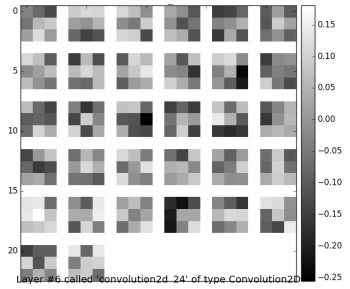

Figure 8: Trained Features of Convolution Layer 1

This first image shows the features of the first convolution layer of our trained model. Each of these 3 by 3 features are applied to the initial image – noting that later layers are features of these features, meaning that the CNN captures the spatial relations between features as well.

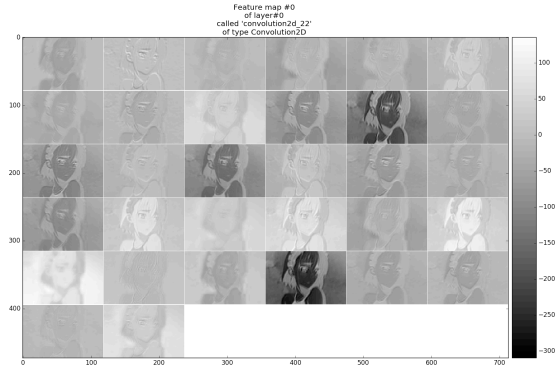

Figure 9: Features of Convolution Layer 1 applied to an image

In Figure 9, we see the effect of convolving each of the 32 feature filters on the input image. The lighter regions indicate a better match between feature and image.

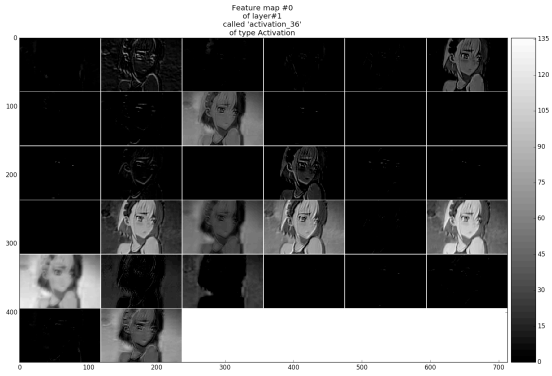

Figure 10: Activation Layer after Convolution Layer 1

In Figure 10, we see the activation layer mostly eliminating the slightly darker grays and only retaining the much lighter areas.

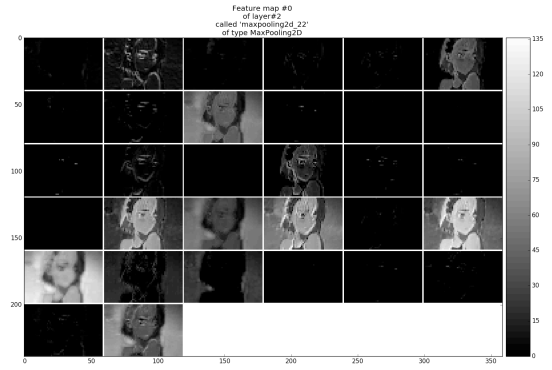

Figure 11: Pooling Layer after Convolution Layer 1

Lastly, in Figure 11, we see the pooling layer reducing each image's length and width by half, while retaining most of the image information. In fact, the down-sized images look almost similar previous images.

Through this visualization process, we see several 'images', or nodes, turning almost completely black. Intuitively, this means that the CNN is not detecting these features in the image, and while we cannot fully understand it without scrutinizing all layers, we know that this influences the final result – whether the CNN classifies the image as anime or cartoon.

6. Conclusion and Future Work

In this project, we attempted to use 2 types of neural networks to classify images as anime or cartoon, to see if a machine could tell the difference between the two genres of pop culture.

While our basic MLP neural network obtained a low test accuracy of 58%, our CNN managed to obtain a high test accuracy of about 92%. This tells us that there are significant differentiating features between anime and cartoon – features that can even be detected by a machine.

While this should give us enough evidence to end the debate on whether anime are cartoons, we found an issue that popular culture would criticize our results for: our model classifies a popular cartoon series as anime. When given 20 screenshots of Avatar (the Last Air-bender), the model classified 85% of them as anime. Hence, there is still room for improvement in our model, especially for more ambiguous cases like Avatar, where characters were humanoid and art styles were closer to those of anime.

One thing that we believe could serve as a better differentiating feature is the way the eyes are drawn. After all, anime eyes are quite distinctly drawn. Other possible methods we would consider would be to use HOG, GIST and/or SIFT as features. We would also consider other algorithms such as SVM and linear kernels, to compare the accuracies of each.

7. References

[1] Blessing, A., & Wen, K. (n.d.). *Using Machine Learning for Identification of Art Paintings.* Retrieved December 13, 2016 from http://cs229.stanford.edu/proj2010/BlessingWen-UsingMachineLearningForIdentificationOfArtPaintings.pdf

[2] Chollet, F. (2016, June 5). *Building powerful image classification models using very little data.* Retrieved December 13, 2016, from https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

[3] Elgammal, A. (2016). *Creating Computer Vision and Machine Learning Algorithms That Can Analyze Works of Art.* Retrieved December 13, 2016, from https://www.mathworks.com/company/newsletters/articles/creating-computer-vision-and-machine-learning-algorithms-that-can-analyze-works-of-art.html?requestedDomain=www.mathworks.com

[4] Gatys, L. A., Ecker, A. S., & Bethge, M. (n.d.). *Image Style Transfer Using Convolutional Neural Networks.*

[5] Karayev, S., Trentacoste, M., & Han, H. et al. (2014). *Recognizing Image Style. Recognizing Image Style.* Retrieved December 13, 2016, from https://sergeykarayev.com/files/1311.3715v3.pdf.

[6] Redi, M., & Merialdo, B. (n.d.). *Enhancing Semantic Features with Compositional Analysis for Scene Recognition.*

[7] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (n.d.). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Journal of Machine Learning Research.

[8] Deshpande, A. (2016). *A Beginner's Guide To Understanding Convolutional Neural Networks*. *Adeshpande3.github.io*. Retrieved 13 December 2016, from https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

[9] Rohrer, B. (2016). *How Convolutional Neural Networks work*. Retrieved from https://www.youtube.com/watch?v=FmpDIaiMIeA

[10] *Fatkun Batch Download Image*. (2016). *Chrome.google.com*. Retrieved 13 December 2016, from https://chrome.google.com/webstore/detail/fatkun-batch-download-ima/nnjjahlikiabnchcpehcpkdeckfgnohf?hl=en