ToonNet: A cartoon image dataset and a DNN-based semantic classification system*

Yanqing Zhou
Shanghai Jiao Tong University
Shanghai
zhouyq@sjtu.edu.cn

Yongxu Jin
Shanghai Jiao Tong University
Shanghai
dowbee@sjtu.edu.cn

Anqi Luo
Shanghai Jiao Tong University
Shanghai
miracledestiny@sjtu.edu.cn

Szeyu Chan
Shanghai Jiao Tong University
Shanghai
szeyu1121@gmail.com

Xiangyun Xiao
Shanghai Jiao Tong University
Shanghai
xiaoxiangyun@sjtu.edu.cn

Xubo Yang
Shanghai Jiao Tong University
Shanghai
yangxubo@sjtu.edu.cn

ABSTRACT

Cartoon-style pictures can be seen almost everywhere in our daily life. Numerous applications try to deal with cartoon pictures, a dataset of cartoon pictures will be valuable for these applications. In this paper, we first present ToonNet: a cartoon-style image recognition dataset. We construct our benchmark set by 4000 images in 12 different classes collected from the Internet with little manual filtration. We extend the basal dataset to 10000 images by adopting several methods, including snapshots of rendered 3D models with a cartoon shader, a 2D-3D-2D converting procedure using a cartoon-modeling method and a hand-drawing stylization filter. Then, we describe how to build an effective neural network for image semantic classification based on ToonNet. We present three techniques for building the Deep Neural Network (DNN), namely, **IUS**: Inputs Unified Stylization, stylizing the inputs to reduce the complexity of hand-drawn cartoon images ; **FIN**: Feature Inserted Network, inserting intuitionistic and valuable global features into the network; **NPN**: Network Plus Network, using multiple single networks as a new mixed network. We show the efficacy and generality of our network strategies in our experiments. By utilizing these techniques, the classification accuracy can reach 78% (top-1) and 93%(top-3), which has an improvement of about 5% (top-1) compared with classical DNNs.

CCS CONCEPTS

• **Computing methodologies** → *Neural networks*; *Image processing*; Supervised learning by classification;

KEYWORDS

Image Dataset, Cartoon Image recognition, Machine Learning

*Produces the permission block, and copyright information

VRCAI '18, December 2–3, 2018, Hachioji, Japan

ACM ISBN 978-1-4503-6087-6/18/12…$15.00
https://doi.org/10.1145/3284398.3284403

ACM Reference Format:
Yanqing Zhou, Yongxu Jin, Anqi Luo, Szeyu Chan, Xiangyun Xiao, and Xubo Yang. 2018. ToonNet: A cartoon image dataset and a DNN-based semantic classification system. In *International Conference on Virtual Reality Continuum and its Applications in Industry (VRCAI '18), December 2–3, 2018, Hachioji, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3284398.3284403

1 INTRODUCTION

Cartoons have always been very popular among children even adults for many years. Cartoon-style pictures can be seen almost everywhere in our daily life. More and more applications try to deal with this style of pictures, so building a cartoon-style dataset will be valuable. Such a dataset can serve as a training or evaluation benchmark for applications, like semantic classification systems, skeleton extraction methods, cartoon picture modeling systems, etc. It will be be beneficial to various cartoon image applications, like Cartoon-face maker, Art-teaching apps, etc. As a result, we construct a basic cartoon dataset, and build an efficient semantic classification neural network for it.

Many large-scale image datasets are widely used for object detection and recognition, like MNIST [Lecun and Cortes 2010] and Imagenet[Deng et al. 2009] [Russakovsky et al. 2014], and also a few well-labeled small datasets, like Caltech[Griffin et al. 2007], PASCAL[Everingham et al. 2010], MSRC[Shotton et al. 2006], CIFAR10/100[Krizhevsky and Hinton 2009], etc. Recently, many academics focus on the extension of these well-known datasets. However, almost all the images that these datasets mentioned above contained are real-world-style, which have a great difference with cartoon-style images in visual perception.

Constructing a cartoon image dataset will be faced with challenges. Cartoon images are not the snapshots of real-world objects, so it is difficult to make use of existing resources to generate cartoon pictures. Although a number of cartoon images from the Internet are available, most of them have the similar cartoon-style: one big color block connected with another. What we need is a dataset with multiple cartoon styles. Therefore we focus on generating other genres of cartoon images like color-penciled style and crayon style, and present three strategies to extend our basal dataset. First, we make use of 3D cartoon models. We implement a cel-shader on the models. Then we take snapshots of them from several angles of view automatically. The second strategy is a 2D-3D-2D procedure.

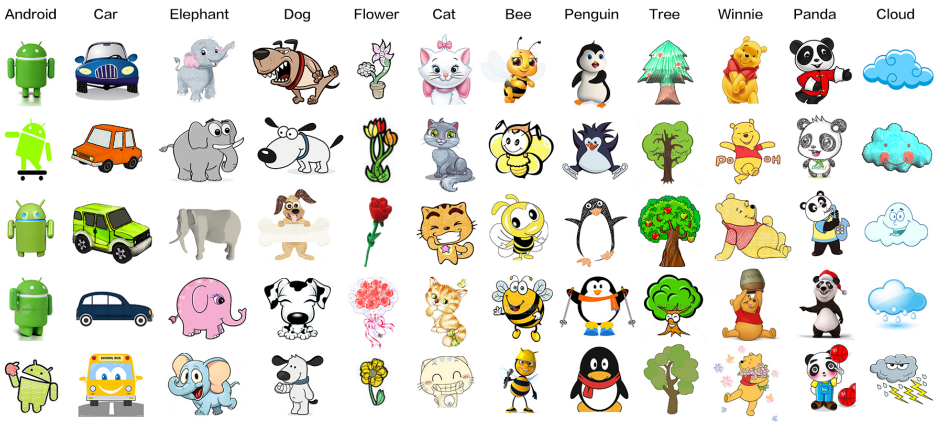


Figure 1: Overview of ToonNet. We define 12 different classes and get original data from the Internet. We then do manual filtration and expand our dataset using several methods which will be explained in Section 3

We take advantage of [Feng et al. 2017], a lightweight 2D-3D cartoon picture modeling method, to generate multiple 3D cartoon models, and do similar operations as the first strategy. Another policy is an image processing method, by using a hand-drawn style filter to change a common cartoon image to hand-drawn styles. We segment the image by color and location, apply randomly rotated grayscale texture to each part, and finally map the color back.

Another major component of this paper explains how to develop a deep neural network model targets the constructed recognition dataset. Convolutional neural networks have shown excellent performance in solving visual recognition problems in past few years. However, some most state-of-the-art DNN models like GoogleNet[Szegedy et al. 2015a] and ResNet[He et al. 2016] focus on the recognition problem of real-world-style images. Notwithstanding the formidable ability of these models can be used for a cartoon picture semantic classification system, the semantic classification results are not good enough on ToonNet in our experiments. We intend to obtain an appropriate semantic classification system to deal with the cartoon picture dataset. And we present three novel techniques for building the model. The Inputs Unified Stylization (**IUS**) technique, we make the stylization of the inputs unified on the premise that does not destroy the principal semantic information of the input. IUS reduces the complexity of inputs and enhances classification accuracy. The second technique is Feature Inserted Network namely **FIN**. Since traditional neural network inputs lack intuitionistic statistical information, like the RGB color histogram which may be valuable of cartoon images, we insert such information into the network and gain better performance. We also apply **NPN** (Network Plus Network) technique, which is a kind of fractional trained network and a derivant of NIN(Network In Network)[Lin et al. 2013] and [GE and RR 2006]. We pre-trained several traditional DNNs on our dataset, and collect their pre-logits layers (the layer before the final full-connected(FC) layer), and retrain these connected collected FC layers as a new plus-network, we find that NPN obtains a higher stability and recognition accuracy than single DNNs.

With the semantic classification information gained by our DNN-based system, we can enhance the functionality of applications like image skeleton extraction method or some other applications and achieve better user experiences.

The main contributions of this work are:

- We construct a cartoon-style image dataset ToonNet, besides collecting from the internet, we use snapshots of 3D models with a cartoon shader, a 2D-3D-2D procedure using a cartoon-modeling method and a hand-drawn stylization filter to enrich our dataset, to gain a picture set with multiple cartoon-styles.
- We introduce strategies on building a DNN-based semantic classification system for cartoon style images, which involves **IUS**, **FIN** and **NPN** techniques, and performs better than traditional state-of-the-art DNN models, like GoogleNet, ResNet, etc.

The rest of the paper is organized as follows. The next section describes an overview of related work. Section 3 shows the technologies when constructing ToonNet. In section 4 we present the three key techniques of our DNN model and shows some experiments on it. Finally we summarize this article and outline some future works.

2 RELATED WORK

Image datasets There are a number of well-known image datasets. MNIST[Lecun and Cortes 2010] is one of the most widely-used datasets in simple image recognition field using machine learning. It is a database of handwritten digits, consists of 60000 training samples and 10000 test samples. There are small datasets like Caltech256[Griffin et al. 2007]: a challenging set of 256 object categories containing 30607 images, PASCAL[Everingham et al. 2010]: a developing image database since 2015 and contains image information for Classification, Detection, Segmentation and Person Layout Taster, MSRC[Shotton et al. 2006] published by Microsoft, CIFAR10/100[Krizhevsky and Hinton 2009], etc. Other datasets like Imagenet[Deng et al. 2009], a database with millions of images, is so widely used that has almost been the "standard" dataset for

measurement and competition [Russakovsky et al. 2014] of algorithm performance in the field of computer vision. However, all these datasets mentioned above are in terms of collections of real-world-style images. So it is necessary and pioneering to construct a cartoon-style image database.

Cartoon image datasets We need a dataset with many tags and lots of data for recognition. However, there are not enough cartoon datasets available on the Internet. Bagdanov et al. [Bagdanov 2012] provides a cartoon dataset, but it does not meet our need because it is mainly used for object detection. Yu et al.[Yu and Seah 2011] also provides a small cartoon dataset including Tom and Jerry for cartoon similarity estimation. Since we cannot find a practical dataset to use, we finally decide to construct our own dataset.

Generating cartoon data There are lots of methods to help to generate cartoon image data. Ha et al. [Ha and Eck 2017] develop a method to automatically generate sketch drawings based on a neural representation. They use sketch-RNN to achieve both conditional and unconditional sketch generation, which can draw simple cartoon sketch drawings like cartoon cats, buses, penguins and so on. This method can provide lots of sketch drawings in a reasonable amount of time, but the quality is not good enough because the dataset they use contains just doodles drawn by human. Liu et al. [Liu et al. 2017] use GAN to paint black-white sketches automatically, and achieve satisfactory results on Japanimation. However, it cannot be used to auto-paint simple cartoon images. The main reason is that Japanimation usually contains lots of complex colors but the cartoon image we need just contains several simple colors. Gatys et al. [Gatys et al. 2016] use Convolutional Neural Network(CNN) to transfer the style of one image to another one, which can be used to transfer cartoon styles. Given a source cartoon image, it can change the target image to the cartoon style, but the result is not stable. Huang et al. [Huang and Belongie 2017] furtherly optimize the style transfer method to realtime. Dong et al.[Dong et al. 2017] propose a method to synthesize realistic images directly with natural language description. Given an image and text descriptions, it can apply the style that the text described to the image, but it does not achieve satisfactory results on cartoon images as well. Besides, 3D models can also help to generate data because of their rich information. Mitchell et al. [Mitchell et al. 2007] propose a practical Non-Photorealistic Rendering method which can be used to cartoonize 3D models.

Neural Network Models Different from traditional methods of image classification, neural networks achieved amazing performance in recent years. Especially these impressive winning models in Imagenet Large Scale Visual Recognition Challenge (ILSVRC)[Russakovsky et al. 2014], such as AlexNet[Krizhevsky et al. 2012], VGGNet[Simonyan and Zisserman 2014],GoogleNet[Szegedy et al. 2015a] and ResNet[He et al. 2016]. [Canziani et al. 2016] gives a comprehensive analysis of these state-of-the-art models of some important metrics in practical applications, such as the relationship between accuracy and inference time. Some academics[Bolukbasi et al. 2017] take use of some of these networks to reduce the evaluation time without loss of accuracy by adaptively utilizing them. For the structure of network models, [Lin et al. 2013] propose a deep network structure called Network In Network to enhance model discriminability. And MCDNN[Schmidhuber 2012] calculate the average results of each DNN, while NPN makes use of pre-logits layers of DNNs and uses a CNN to combine them. However, since the cartoon dataset is pioneering, few studies focus on developing a discriminative neural network model used for cartoon-style picture classification. We present three niche-targeting technologies and extend the architecture of the famous network models to a multiple-plus fractional trained network. The goal of our network is to achieve better recognition performance on our dataset.

3 CONSTRUCTING TOONNET

We use various methods to create and enlarge our cartoon database. Cartoon images have many different styles, which requires the dataset to include different styles of data as many as possible. However, the data on the Internet is far less than we need, so we introduce several methods to generate new data using existing data. All of these methods are explained below in detail.

3.1 Data Collection

We first use a web crawler to build our initial dataset. We define 12 different tags (android robot, car, elephant, dog, flower, cat, bee, penguin, tree, Winnie bear, panda, cloud) and crawl about 1000 images for each tag from the Internet. However, the data we get is crude and noisy, so we have to filter it manually. We finally obtain about 4400 images in total, which is not sufficient to construct a practical dataset. Thus, we introduce several methods to replenish the dataset.

3.2 Snapshots of 3D models

3D models are applicable to generate new data because we can get large amounts of different snapshots of models from any positions and angles of view. To make those snapshots look like being drawn by a human, Non-Photorealistic Rendering methods have to be applied to models in advance. Our method first applys a cel-shader (or toon shader) to the model, then depict contours using Sobel operator, and finally capture several snapshots of the model automatically. The light model of our cel-shader is computed as

$$I_c = k_a i_a + \sum_{m \in lights} (k_d(\alpha(L_m \cdot N) + \beta) i_{m,d} \times Ramp(\alpha(L_m \cdot N) + \beta)) \quad (1)$$

Where I_c is the output color, i_a controls the ambient lighting, $i_{m,d}$ controls the diffuse lighting of light m, k_a and $k_d$are the ambient and diffuse terms of the 3D model, L_m is the light vector, N is the normal vector, α and β are coefficients of Half Lambert[val [n. d.]] model and are usually set to 0.5. We ignore the exponent term of Half Lambert for simplicity. *Ramp*() function refers to sampling a 1D ramp texture by diffuse light intensity, and returns the sampled value. Figure 2 shows the ramp texture we use. We use Half Lambert lighting technique to ensure plenty of light when viewing from different angles, and a ramp texture to obtain cartoon-style tone. The specular term is ignored because few cartoon images consider the highlight. In practice, we only use ambient light and one directional light, since cartoon images usually do not care about complex light conditions.

We postprocess the scene using Sobel operator to depict the contour of the cartoon model. We only apply Sobel operator to the color buffer because the white background and the ramp texture

Figure 2: The ramp texture we use is a 3-level grayscale image. Using this texture, the cel shaded model will have up to three diffuse intensity levels.

(a) (b) (c) (d)

Figure 3: (a) is the input model. (b) is the model added cel shader. (c) is the model depicted contours using Sobel operator. (d) is the snapshot of the model from a different angle.

ensure high contrast ratio of the color buffer image to let Sobel operator work properly. Thus we do not need help from depth buffer or normal buffer.

After applying cel shader and Sobel operator, we finally capture snapshots of the model from different angles, getting the generated data. Using this method, we can easily generate a mounts of cartoon images data from a single model. In our dataset, we collect 200 models and generate 2000 images. Figure 3 shows how to convert a 3D model to 2D cartoon images.

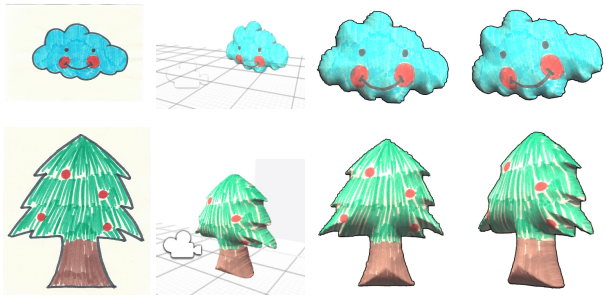
3.3 2D-3D-2D procedure

Besides generating cartoon images from a 3D model, we also consider generating multiple images from a 2D image. This requires applying transformation technique to the original image. We use MagicToon [Feng et al. 2017], a 2D-to-3D creative cartoon modeling system as our transformation method. First, we use MagicToon to convert the 2D cartoon image to a 3D model, then capture snapshots of the models from different angles of view automatically to obtain new cartoon images. We do not need to apply NPR methods to the model again because this method has already done the toon shading step. All we need is to model the image input, and capture snapshots. Using this method, one single cartoon image can generate many new cartoon images easily. And we also use a black stroke style shader to make the images captured might look more like hand-drawn cartoon images. We get about 1500 new images by the procedure. Figure 4 shows how a cartoon image generates new images using a 2D picture-modeling system.

3.4 Hand-drawn stylization filter

Cartoon images usually have different styles, like pencil sketch style, crayon style, CG style, etc. A cartoon dataset should have different styles of cartoon images for variety. To achieve this goal, we use a filter similar to Lu's method[Lu et al. 2012] to change a common cartoon image to a grayscale hand-drawn image. We first convert the cartoon image to grayscale, and compute the gradient of the image, using it as the edge image. Then we do tone mapping to the grayscale image, changing it to a hand-drawn tone, and apply a texture to it. We finally merge the edge image and texture image, which generates the grayscale hand-drawn image.

To change the grayscale image to a color one, we apply the following equation to the original cartoon image:



(a) (b) (c) (d)

Figure 4: Generating images using MagicToon. (a) are the input images. (b) show the models generated in Unity3D. (c,d) are the images captured by the camera.

$$I = I + G - \frac{G \circ I}{255} \tag{2}$$

Where I is the original cartoon image and G is the grayscale pencil-drawing-style image with texture. I and G are all $w \times h \times 3$ matrix, where three channels in G are the same (grayscale value). Operator $\circ$ refers to Hadamard product. This equation will apply the texture of the grayscale image to the original image. Consider a pixel in G. When the pixel value is 0 (black pixel), the corresponding pixel value of I does not change. When the pixel value is 255 (white pixel), the corresponding pixel value in I is also 255 (white pixel). This means that the black strokes of the texture are mapped to color strokes, and the white blank remains the same. Therefore, the grayscale texture can be directly applied to the cartoon image according to the color. In practice, we first apply histogram equalization to G, darkening the strokes, otherwise, the color of the final output image will be too light.

Furthermore, some pencil textures are so ordered that when they are directly applied to the image, the image may seem weird. This is because the ordered texture makes the image seem to be drawn by a whole, and human definitely cannot draw different areas and colors with only several strokes, like Figure 5(b) shows. To solve the problem, we first use Mean Shift[Comaniciu and Meer 2002] to segment the cartoon image, and apply the texture to each segmented area separately. When applying the texture, we rotate it by a random angle in advance, so that each area seems to be drawn separately, instead of being drawn by a whole, as shown in Figure 5(c). After this transformation, the pencil-style cartoon image seems more realistic.

Using this method, we can actually apply any textures to the cartoon image beside pencil texture. Some textures like crayon texture also achieve satisfactory results. And we gain about 2000 images by the filter with different styles. We can even design textures manually to customize cartoon effect and generate more cartoon images. Figure 5 shows the result of different textures applying to the cartoon image. These generated cartoon images are suitable to replenish our dataset.

4 LEARNING A DNN MODEL

In this section, we introduce the three main techniques to build a discriminative neural network model to recognize images in our

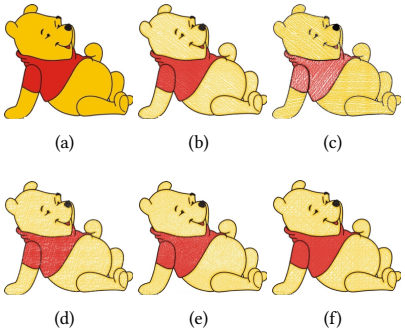


Figure 5: Generating images using hand-drawn stylization filter. (a) is the input image. (b) and (c) are the images added pencil texture without and with segmentation. (d) and (e) are the images added two different crayon textures. (f) is the image added customized texture.

dataset. The task is to learn the relationship tags and input images. The model is end-to-end. Input: one cartoon image; Output: semantic classification result of it. 4.1 describes the pre-processing strategy "Inputs Unified Stylization" called IUS, 4.2 gives the idea to utilize the global features of images "Feature Inserted Network" called FIN. In 4.3, we will show a fractional trained network structure using multiple networks "Network Plus Network", we called NPN.

4.1 Inputs Unified Stylization (IUS)

Data preprocessing always plays an important role in image-processing field. A suitable pre-processed input image will have a good effect on neural networks learning tasks. As to our classification system, since there will be various patterns or styles of cartoon image in our dataset, we need to reduce the complexity of our image inputs without loss of classification information. Thus we propose IUS method, to change these cartoon images to a unified and simple style.

The basic idea is to apply a filling algorithm analogous to Flood Fill [Bhargava et al. 2013] that replaces similar colors with one color and fills the image.

Firstly, we need to find and mask background and outline of the image, which should not be filled. They are always light or dark colors rather than absolute white or black. Thus, we convert the image from RGB color space to HSV color space for better recognition. In practice, the background pixel has a lower saturation(S) close to the minimum and a higher value(V), while the outline pixel has a low value(V) which is actually a little bigger than minimum because the hand-drawn outline is not dark and homogeneous enough. Besides, some small light color regions are left due to arbitrary painting, which should not be masked as background.

In addition, hand-drawn images have the feature that colors are painted irregularly. Therefore, we do preprocessings to the images like Mean Shift [Comaniciu and Meer 2002] filter, replaces each pixel with the mean of the pixels in a certain range neighborhood with similar colors. To make our result more colorful, we separate an image into several small regions, and in each region, pick up the pixel with the highest saturation as a seed pixel. Now we can start to fill the image from a seed pixel. Each neighborhood pixel of the seed will be checked, and if it either has a close hue(H) to the seed pixel or is the light pixel not masked as background, it will be filled with the same color as the seed pixel's and its eight neighborhood pixels will be checked soon. This procedure will repeat by using the strategy of the breadth-first search until no pixel is filled and then restart from another seed pixel. Figure 6 shows the results of filling hand-drawn images.

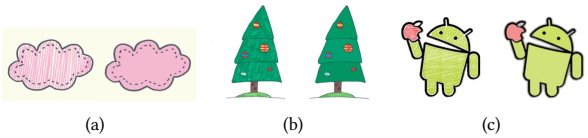


Figure 6: The examples of images parsed by IUS .(left) Input images and (right) the filled images by our algorithm . The white among strokes are filled and the colors are more homogeneous.

Finding a stylization method to make inputs unified may be suitable for cartoon-style images, but it is more difficult for real-world-style images. Pictures captured in the real world contain much more complicated information of both backgrounds and foregrounds, which leads to the trouble reducing the complexity of inputs without loss of obbligato details. As a result, Inputs Unified Stylization preprocessing strategy for real image recognition problem may have limitations.

4.2 Feature Inserted Network (FIN)

Global features of images are very useful to a classification system, especially targeted on characteristic data like cartoon images. Traditional neural networks usually use end-to-end models, and always begin with convolution or some local region targeted operations. These operations might result in a lack of some intuitionistic statistical information that may be valuable. Although the strong learning ability of efficient DNNs can get a few abstract features after long time training, inserting valuable features into networks directly may still enhance the analytical ability.

After IUS preprocessing, the inputs we get are images with simple backgrounds and a unified style that one big color block connected with another. The colors of these blocks are valuable, so we determine to use the color histograms of the cartoon image as our inserted feature. We count up the frequency of occurrence of pixel values from 0 to 255 in each RGB channel, and merge the 256×3 cells into n×3 bins(n=19 in our experiments). The histogram information will be inserted into a normal network after normalization on mixed-channels. We divide a normal network into several parts: the input layer with a resized image; the core of a normal network, which contains all the layers before the final pre-logits layer; the final pre-logits layer, which is layer before final FC layers, the size of the layer is 4096 in VGGNet and 2048 in ResNet50/101/152[He et al. 2016] and GoogleNet_v3[Szegedy et al. 2015b]; the final FC layer and the softmax outputs layer. As figure 7 shows, we insert our global feature into the final pre-logits layer, and use a convolution layer to convert the RGB-features into one-dimensional feature logits, then an additional full-connected layer appended, between

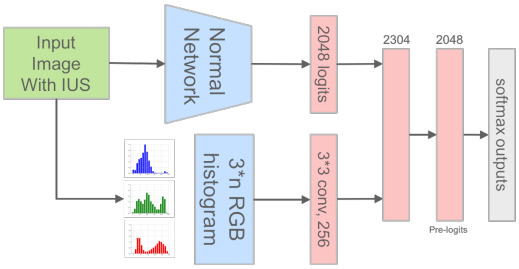


Figure 7: An example of a Feature Inserted Network

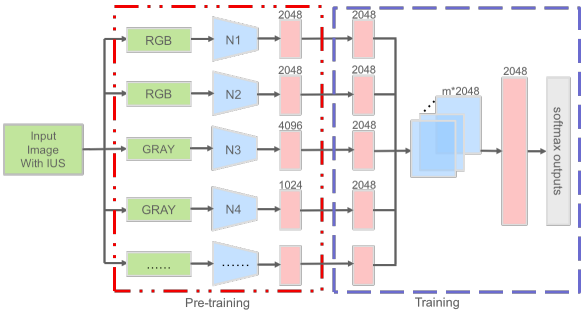


Figure 8: Overview of a Network Plus Network structure

the feature inserted layer and the final output layer, as the updated pre-logits layer.

In our work, we use RGB histogram information as the feature for FIN in experiments, but the features that can be inserted into networks are not limited to it. The RGB information may be inadaptable for real-world style images since some of them have complicated backgrounds and more detailed texture than cartoon images. We can choose other local or global valuable features of an input image such as the SIFT features, local color contrast, HSV color histograms, etc.

4.3 Network Plus Network (NPN)

We propose a new multiple-network structure Network Plus Network(NPN). The overall structure of NPN is a paratactic connection of several pre-logits layers of normal networks, and a shallow CNN in the end. Figure 8 is the overview of a sample NPN. The NPN structure contains two parts. The first subpart is a pre-training structure. There could be m networks used here (N_1,N_2,$\cdots$, N_m), thus we call the structure NPN-m. These m networks are pre-trained using the same database.Actually, each network already has the ability of image recognition, and they will have their unique features on the same input image. We reserve the part of these networks that the layers before the final FC layer, and retrain them as a mixed system. The other subpart of NPN is the new training part, whose inputs are the m FC layers of the pre-training part. For each FC layer, we implement a new FC layer on it to transform them into the same dimension, e.g. 1×2048, and mix them as a m×2048 layer. Then a m×1-size convolution layer is applied and a final FC layer at the end.

Training a NPN model Training a NPN model is a time-consuming task. We should train m normal networks on the dataset at the beginning. These m networks may have different types or color spaces of images, such as RGB, HSV, GRAY, etc. And the models may have different structures such as networks with FIN and networks without FIN, and they can even be same models just with different training parameters. Then we start training the multiple-network. We reload these m-models and pick-up their pre-logits outputs as the new input, and train the new simple CNN.

In particular, when we only choose one normal network ($m = 1$), and at the training part, we reserve most parts of the pre-trained network, and retrain the rest layers. The NPN-1 structure is a simple fractional trained network and similar to the traditional networks using [GE and RR 2006], with the ability of reducing the dimensionality of data.

5 EXPERIMENTS

We evaluate our three network strategies (IUS,FIN,NPN) on our constructed ToonNet that consists of 12 cartoon classes. We measure network using TensorFlow library for core computations and Python for the front end, utilizing a server with Nvidia GeForce GTX 980 with CuDNN 6. The batchsize used is 16 and AdamOptimizer as the optimizer. It costs about 1 hour for each DNN of NPN and 15 minutes for the last CNN part. The model is trained from scratch, since pre-trained models on ImageNet are tailored for ordinary images.

We evaluate our method on several aspects, like evaluation time, test accuracy etc. The results show that each learning strategy can enhance the classification accuracy, especially the NPN structure. Furthermore, our evaluation time cost is close to the cost of tradition networks in spite of the m-times FLOPs of NPN-m structure for the reason that NPN structure is born for parallel procession.

Note that the data we used for training is a random 70% of our dataset, and the rest for tests, and we use the same data (the random 70%) in all experiments. All these traditional networks use Dropout[Hinton et al. 2012] and Batch-Normalization[Ioffe and Szegedy 2015] as regularizers to improve the generalization and inference ability and prevents overfitting. Our training use an initial learning rate 2e-3, and divided by 10 every 5 iterations.

5.1 Experiments on IUS

We evaluate the evaluation accuracy on traditional networks and the networks with IUS. The results in table 1 show that IUS has the ability of enhancing test accuracy, especially for these shallow networks with plain structure. Although the improvement is limited for deep networks like ResNet and GoogleNet_v3, IUS is still useful in most situations for its trivial time cost: 5ms per image

Table 1: Top-1 test error on ToonNet constructed. The "With IUS" means the custom networks only with IUS structure, that is to say the inputs for them are pre-processed.

	Custom	With IUS
AlexNet	45.4	44.1
VGG-16	40.4	39.4
ResNet-50	36.8	36.5
ResNet-101	34.7	34.4
ResNet-152	32.8	32.6
GoogleNet_v3	**27.1**	**26.7**

Table 2: Top-1 and Top-3 test error on ToonNet constructed. The networks tested here are R-101: ResNet of 101 layers and G-v3: GoogleNet Inception-v3.

	Color Space	IF With FIN	Top-1 err.	Top-3 err.
R-101	GRAY	-	37.6	15.6
		√	35.4	13.5
	RGB	-	34.4	12.8
		√	**33.0**	**11.8**
G-v3	GRAY	-	30.3	12.8
		√	28.4	11.5
	RGB	-	26.7	9.6
		√	**25.8**	**8.7**

5.2 Experiments on FIN

In our work, we insert the RGB histogram of the input image as our feature in FIN structure. The modality of RGB histogram is quite different between real-world-style images and cartoon-style, because images captured in the real world may have more complicated details. In our experiments, FIN can improve the evaluation accuracy as well, as table 2 show. For gray image inputs, FIN can improve accuracy much more than that with RGB images inputs, for the reason that the lack of color information is inherent for gray images. FIN enhance the performance even though the custom powerful networks can learn a bit of color information.

5.3 Experiments on NPN

NPN is the main structure of our system. In our experiments, NPN show several fantastic features in machine learning tasks.

First, NPN is suitable for machine learning tasks on computers or servers with common specifications, which may have the limitation of parameter chosen, like batch size. We unable to use a very large batch size when training a traditional network because of high video memory cost. Since NPN structure uses a fractional training strategy and has fewer FLOPs (Floating Point Operations Per Second) in the training part, we can choose a small batch size for the pre-training part of NPN and a much bigger one for the training part. We use NPN-1 structure in the batch-size experiment. And figure 9 shows the results. We pretrain ResNet-101 and GoogleNet-v3 using batch size of 24, and use different batch size for the training part. We argue that a vary small batch size (16, 8, e.g.)may have poor ability to converge to a good global optimum of the training set despite that normal batch size is not trivial to keep the accuracy of networks. The NPN structure makes us able to use different batch size for different network parts (pre-training part and training part), and achieve a higher accuracy.

Second, NPN-m ($m > 1$) structure has much better one-crop accuracies than traditional state-of-the-art networks. Figure 10 is a convincing proof that different networks will have different pre-logits layers for the same input, and it is no doubt that each pre-logits layer of these networks has its unique feature of the input image. If the networks used are similar like G-v3 and G-v3 with FIN, the features of them will be analogous as well but still have some differences. NPN-m structure is created to combine these features of different networks, and gain a feature more valuable. Table 3 shows the outstanding inference performance of NPN-m on our dataset.

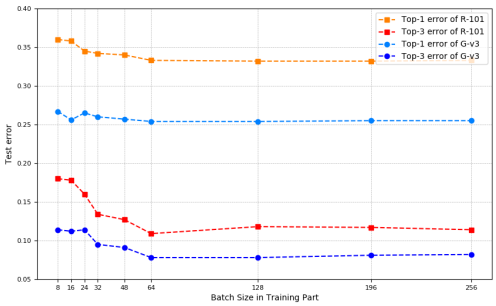


Figure 9: Different batch size used in the training part of NPN-1 structure. When batch size is up to 64, it is not trivial to keep the accuracy of network.

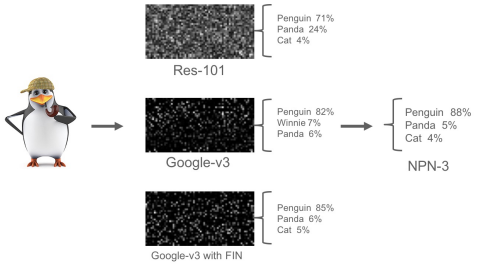


Figure 10: Visualization of pre-logits layers

Table 3: The inference performance of NPN-m structure. The column m shows how many networks we used in NPN. The second column is the networks used, G-v3-FIN means GoogleNet Inception-v3 with FIN.

m	Network used	Top-1 err.	Top-3 err.
1	R-101	33.3	11.8
1	G-v3	25.4	8.1
2	R-101 & G-v3	24.7	7.2
2	R-101 & G-v3-FIN	24.0	7.2
2	G-v3 & G-v3-FIN	23.3	7.3
3	R-101 & G-v3 & G-v3-FIN	**22.4**	**7.0**
4	R-50 & R-101 & G-v3 & G-v3-FIN	**22.4**	7.3

We test different group with R-50, R-101, G-v3 and G-v3-FIN. We find that a mixed network (NPN) outperforms a good single model. And the better single nets chose for NPN, the better performance will be achieved, such as the difference between NPN-2 with R-101,G-v3 and NPN-2 with G-v3,G-v3-FIN. Moreover, m is not linear with the inference performance as results show.

The training part of NPN-m can have a much faster convergence since the networks in the pre-training part have been trained already. Like the training status in figure 11 shows, NPN-3 can convergence to an ideal state in 2 iterations. And the convergence will have more smooth loss shocks than normal networks (the green frame in subfigure left 11).

An appropriate selection of m is significant, since m is linear with the total training time cost, we should pre-trained m useful single networks for a NPN-m network. So it may become a trade-off between larger m and shorter training time.

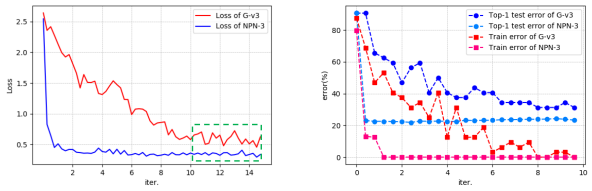


Figure 11: The comparison of training status between NPN-3 (use R-101, G-v3 and G-v3-FIN) and G-v3.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we present ToonNet, a cartoon-style image recognition dataset. Since we cannot get enough data from the Internet (about 4400 images after manual filtration), we introduce several methods to expand our dataset, including snapshots of rendered 3D models, 2D-3D-2D procedure and hand-drawn stylization filter. Using these methods, we can easily generate new data based on other resources.

In the future, besides going on enlarging the database(the number of tags and the images in each tag) , we can make our dataset support not only recognition, but also new features like semantic segmentation. By adding segmentation information to our data, users can train neural networks like DeepLab[Chen et al. 2018].

We also provide a targeted strategy for building a DNN-based classification system using IUS, FIN and NPN, and enhance the performance compared with state-of-the-art networks. IUS makes the inputs have a unified style with little time cost and FIN takes advantage of global features of cartoon images. Both make use of the characteristics of cartoon style inputs and may only suitable for cartoon-image training tasks. NPN extends the architecture of classic networks and may be a universal strategy in different situations. It make sense that NPN is able to gain a better performance than single models by mixing their features. At the next stage, we will train NPN on other real-world style image databases, like CIFAR and Imagenet.

ACKNOWLEDGEMENTS

This work was partially supported by the Natural Key Research and Development Program of China (2018YFB1004902), Natural Science Foundation of China (61772329, 61373085). We thank all the reviewers for their valuable comments.

REFERENCES

[n. d.]. Half Lambert. https://developer.valvesoftware.com/wiki/Half_Lambert.

Andrew D. Bagdanov. 2012. Color Attributes for Object Detection. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (CVPR '12)*. IEEE Computer Society, Washington, DC, USA, 3306–3313. http://dl.acm.org/citation.cfm?id=2354409.2354951

Neeraj Bhargava, Prakriti Trivedi, Akanksha Toshniwal, and Himanshu Swarnkar. 2013. Iterative Region Merging and Object Retrieval Method Using Mean Shift Segmentation and Flood Fill Algorithm. In *Third International Conference on Advances in Computing and Communications*. 157–160.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for efficient inference. (2017), 527–536.

Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. 2016. An Analysis of Deep Neural Network Models for Practical Applications. *CoRR* abs/1605.07678 (2016). arXiv:1605.07678 http://arxiv.org/abs/1605.07678

L. C. Chen, G Papandreou, I Kokkinos, K Murphy, and A. L. Yuille. 2018. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 40, 4 (2018), 834–848.

Dorin Comaniciu and Peter Meer. 2002. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence* 24, 5 (2002), 603–619.

Jia Deng, Wei Dong, R. Socher, Li Jia Li, Kai Li, and Fei Fei Li. 2009. ImageNet: A large-scale hierarchical image database. *Proc of IEEE Computer Vision & Pattern Recognition* (2009), 248–255.

Hao Dong, Simiao Yu, Chao Wu, and Yike Guo. 2017. Semantic Image Synthesis via Adversarial Learning. *CoRR* abs/1707.06873 (2017). arXiv:1707.06873 http://arxiv.org/abs/1707.06873

Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. 2010. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision* 88, 2 (2010), 303–338.

Lele Feng, Xubo Yang, and Shuangjiu Xiao. 2017. MagicToon: A 2D-to-3D creative cartoon modeling system with mobile AR. In *Virtual Reality*. 195–204.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2016. Image Style Transfer Using Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*. 2414–2423.

Hinton GE and Salakhutdinov RR. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507.

Gregory Griffin, Alex Holub, and Pietro Perona. 2007. Caltech-256 Object Category Dataset. *California Institute of Technology* (2007).

David Ha and Douglas Eck. 2017. A Neural Representation of Sketch Drawings. *CoRR* abs/1704.03477 (2017). arXiv:1704.03477 http://arxiv.org/abs/1704.03477

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. (2016), 770–778.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *Computer Science* 3, 4 (2012), págs. 212–223.

Xun Huang and Serge J. Belongie. 2017. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. *CoRR* abs/1703.06868 (2017). arXiv:1703.06868 http://arxiv.org/abs/1703.06868

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR* abs/1502.03167 (2015). arXiv:1502.03167 http://arxiv.org/abs/1502.03167

Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *International Conference on Neural Information Processing Systems*. 1097–1105.

Yann Lecun and Corinna Cortes. 2010. The mnist database of handwritten digits. (2010).

Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network In Network. *Computer Science* (2013).

Yifan Liu, Zengchang Qin, Zhenbo Luo, and Hua Wang. 2017. Auto-painter: Cartoon Image Generation from Sketch by Using Conditional Generative Adversarial Networks. (2017).

Cewu Lu, Li Xu, and Jiaya Jia. 2012. Combining Sketch and Tone for Pencil Drawing Production. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering (NPAR '12)*. Eurographics Association, Goslar Germany, Germany, 65–73. http://dl.acm.org/citation.cfm?id=2330147.2330161

Jason Mitchell, Moby Francke, and Dhabih Eng. 2007. Illustrative rendering in Team Fortress 2. In *International Symposium on Non-Photorealistic Animation and Rendering*. 71–76.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, and Michael Bernstein. 2014. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115, 3 (2014), 211–252.

Jurgen Schmidhuber. 2012. Multi-column Deep Neural Networks for Image Classification. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (CVPR '12)*. IEEE Computer Society, Washington, DC, USA, 3642–3649. http://dl.acm.org/citation.cfm?id=2354409.2354694

Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. 2006. TextonBoost: Joint Appearance, Shape and Context Modeling for Multi-class Object Recognition and Segmentation. In *European Conference on Computer Vision*. 1–15.

Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556 http://arxiv.org/abs/1409.1556

C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. 2015a. Going deeper with convolutions. 00 (June 2015), 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015b. Rethinking the Inception Architecture for Computer Vision. *Computer Science* (2015), 2818–2826.

Jun Yu and Hock-Soon Seah. 2011. Fuzzy Diffusion Distance Learning for Cartoon Similarity Estimation. *J. Comput. Sci. Technol.* 26, 2 (March 2011), 203–216. https://doi.org/10.1007/s11390-011-1123-x