Pacific Graphics 2018
H. Fu, A. Ghosh, and J. Kopf
(Guest Editors)

Volume 37 (2018), Number 7

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

SUBMISSION ID 1086

Abstract

*We present ToonNet: a cartoon-style image recognition dataset. We construct our basal 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 screenshots of 3D models with a cartoon shader, a 2D-3D-2D procedure using cartoon-modeling method and a hand-drawing stylization filter. We also describe how to build an effective convolutional neural network on the cartoon image dataset. We present three techniques for building the deep neural network (DNN) (**IUS**: Inputs Unified Stylization, stylizing the inputs to reduce the complexity of hand-drawn cartoon images ; **FIN**: Feature Inserted Network, inserting some intuitionistic and valuable global features into the network; **NPN**: Network Plus Network). By utilizing these techniques, the classification accuracy can reach to 78% (top-1) and 93%(top-3), which has an improvement of about 5% (top-1) on classical DNNs. This work provides a cartoon image dataset and presents a targeted strategy for building a DNN-based classification system, which extends the architecture of state-of-the-art networks and has a satisfactory performance.*

CCS Concepts

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

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. But few applications or tools try to deal with this style of pictures, let alone build a cartoon-style dataset. Such a dataset can serve as a training or evaluation benchmark for some applications, like 2D cartoon picture modeling system. So, we try to construct a basic dataset, and build an efficient classification neural network.

Some large-scale image datasets are widely used for object detection and recognition, like MNIST [LC10] and imagenet [DDS*09] [RDS*14], and also a few well-labeled small datasets, like Caltech [GHP07], PASCAL [EGW*10], MSRC [SWRC06], CIFAR10/100 [KH09], 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 some challenges because of the pioneering of this work. Cartoon images are not the snapshots of real-world objects, so it is difficult to make use of existing resources to generate cartoon pictures. A number of cartoon images from the Internet are available, but most of them have the similar cartoon-style: one big color block connected with another. 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 put the models into Unity3D and implement a novel shader on them. Then we take snapshots of them from several angles of view. The second strategy is a 2D-3D-2D procedure. We take advantage of MagicToon [FYX17], a lightweight 2D-3D cartoon picture modeling method, to generate multiple 3D cartoon models, and do similar operations as first strategy. Another policy is an image processing method, by using a hand-drawing-style filter to change a common cartoon image to hand-drawing style. 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 an deep neural network model targets the constructed recognition dataset. Convolutional neural networks have shown excellent performance in solving visual recognition problems in past few year. However, some most state-of-the-art DNN models like GoogleNet [SLJ*15] and ResNet [HZRS16] focus on the recognition problem of real-world-style images. Notwithstanding the formidable ability of these models may still suitable for a cartoon picture classification system, we intend to obtain an appropriate classification system to deal with the cartoon picture dataset. And we present three neoteric techniques for building the model. The Inputs Unified Stylization (**IUS**) technique, we make the stylization of the inputs unified on the premise that do not destroy the principal sematic information of the input. IUS reduces the complexity of inputs and enhance classification accuracy. The second technique is Feature Inserted Network namely **FIN**. Since traditional neural network inputs lack intuitionistic statistical information, like the RGB

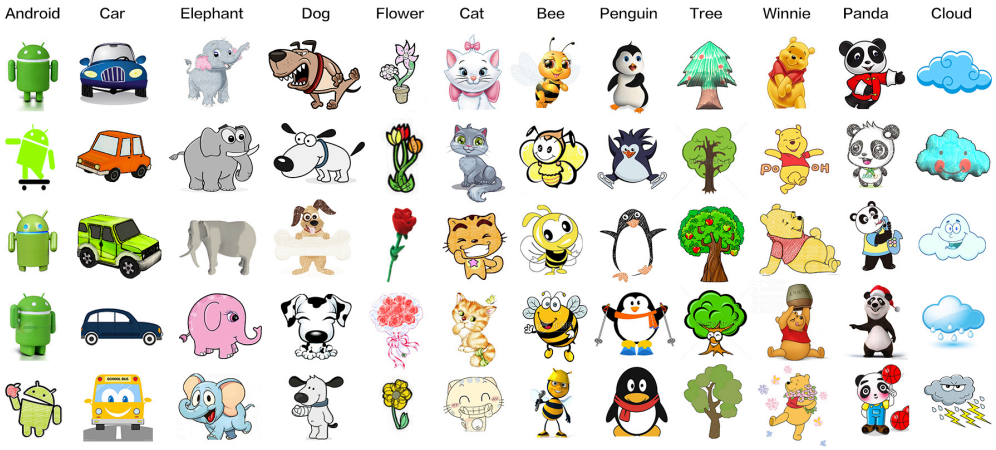


Figure 1: *Overview of ToonNet. We define 12 different classes and get original data from the Internet. We then do manual filtration and expand our dataset using several methods which will be explained in Section 3*

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) [LCY13] and [GR06]. We pre-trained several traditional DNNs on our dataset, and collect their pre-logits layers (the layer before the final 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 the DNN-based system, we can enhance the functionality of applications like image skeleton extraction method or some other AR applications like MagicToon [FYX17], 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 screenshots of 3D models with a novel shader, a 2D-3D-2D procedure using cartoon-modeling method and a hand-drawing stylization filter to enrich our dataset, to gain a picture set with multiple cartoon-styles.
- We introduce a cartoon-style-image-targeted strategy on building a DNN-based classification system, which involves **IUS**, **FIN** and **NPN** techniques, and performs better than traditional state-of-the-art DNN models.

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. Then we give an application concept using our system. 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 [LC10] is one of the most widely-used dataset 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 [GHP07]: a challenging set of 256 object categories containing 30607 images, PASCAL [EGW*10]: a developing image database since 2015 and contains image information for Classification, Detection, Segmentation and Person Layout Taster, MSRC [SWRC06] published by Microsoft, CIFAR10/100 [KH09], etc. Other datasets like ImageNet [DDS*09], a database with millions of images, is so widely used that has almost been the "standard" dataset for measurement and competition [RDS*14] of algorithm performance in the field of computer vision. However, all these datasets mentioned above are in terms of collections of real-world-style image. 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. [Bag12] provides a cartoon dataset, but it does not meet our need because it is mainly used for object detection. Yu et al. [YS11] 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 generating cartoon image data. Ha et al. [HE17] 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 cat, bus, penguin 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. [LQLW17] use GAN to paint black-white sketches automatically, and achieve satisfactory result on Japanimation. However, it cannot be used to paint simple cartoon image. The main reason is that Japanimation usu-

ally contains lots of complex colors but the cartoon image we need just contains several simple colors, so the method they propose does not achieve good result. Gatys et al. [GEB16] use 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. Besides, 3D models can also help generating data because of their rich information. Mitchell et al. [MFE07] 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) [RDS*14], such as AlexNet [KSH12], VggNet [SZ14],GoogleNet [SLJ*15] and ResNet [HZRS16]. [CPC16] 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 [BWDS17] 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, [LCY13] propose a deep network structure called Network In Network to enhance model discriminability. However, since the cartoon dataset is pioneering, few studies focus on developing a generative 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 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. Screenshots 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 human, Non-Photorealistic Rendering methods have to be applied to models in advance. Our method first apply a cel shader (or toon shader) to the model, then depict contours using Sobel operator, and finally capture several screenshots of the model. 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] 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 highlight. In practice, we only use ambient light and one directional light, since cartoon images usually do not care about complex light conditions.

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.*

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 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 screenshots of the model from different angles, getting final data. Using this method, we can easily generate a mounts of cartoon images data from a single model. Figure 3 showshow 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 [FYX17], a 2D-to-3D creative cartoon modeling system to apply transformation. 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 to obtain new cartoon images. We do not need to apply NPR methods to the model again because MagicToon has already done the toon shading step. All we need is to model the image input, and capture screenshots. 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-drawing cartoon images. Figure 4 shows how a cartoon image generates new images using a 2D picture-modeling system.

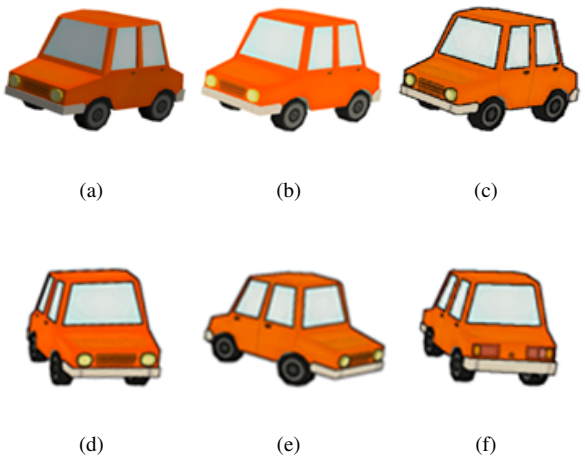


Figure 3: *Procedure of generating images from a 3D model. (a) is the input model. (b) is the model added cel shader. (c) is the model depicted contours using Sobel operator. (d,e,f) are the screenshots of the model from different angles.*

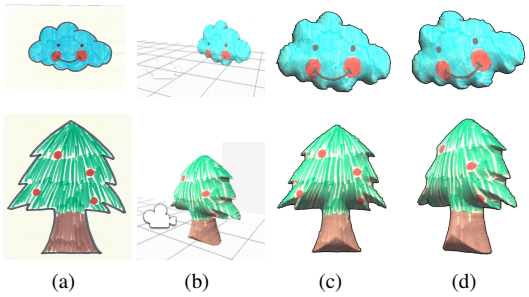


Figure 4: *Generating images using MagicToon. (a) are the input images. (b) show the models generated in Unity3D. (c,d) are the images captured buy the camera.*

3.4. Hand-drawing 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 [LXJ12] to change a common cartoon image to a grayscale pencil-drawing-style image. We first convert the cartoon image to grayscale, and compute gradient of the image, using it as the edge image. Then we do tone mapping to the grayscale image, changing it to a pencil-drawing-style tone, and apply texture to it. We finally merge the edge image and texture image, which generates the grayscale pencil-drawing-style image.

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

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

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 in 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 see 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 [CM02] 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.

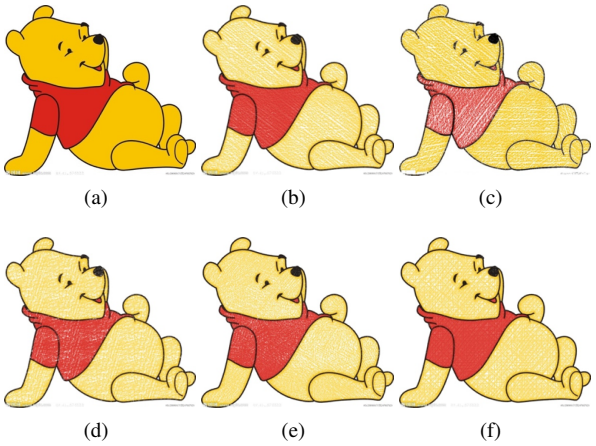


Figure 5: *Generating images using pencil-drawing-style filter. (a) is the input image. (b) and (c) are the images added pencil texture with and without segmentation. (d) and (e) are the images added two different crayon textures. (f) is the image added customized texture.*

Using this method, we can actually apply any textures to the cartoon image besides pencil texture. Some textures like crayon texture also achieve satisfactory result. We can even design textures manually to customize cartoon effect. 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 ~~will show~~ the three main techniques ~~used~~ to build a generative neural network model to recognize images in our dataset. 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, ~~and~~ 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)

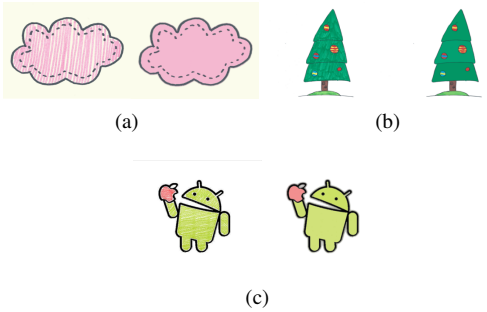
Data preprocessing always plays an important role in image-processing field. A suitable pre-processed input image will have a good effect on neural networks learning tasks. As to our classification system, since there will be various patterns or styles of cartoon image in our dataset, we need to reduce 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 [BTTS13] 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 and dark colors rather than exactly 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 saturation close to minimum and a high value near maximum, while the outline pixel has a low value 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 drawing images have a fe that colors are painted irregularly. Therefore, we do some pre ssing to the images with Mean Shift [CM02] filter, ~~replacing~~ each pixel with the mean of the pixels in a certain range neighborhood having 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 pixel in its eight neighborhood is checked, and if it either has a close hue 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 breadth-first search until no pixel is filled and then restart from another seed pixel. Figure 6 shows the result of filling a hand drawing image.

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 real world contain much more complicated information of both backgrounds and foregrounds, and leads to ~~the~~ trouble ~~for~~ reducing the complexity of inputs without loss of obbligato details. As a result, Inputs Unified Stylization prep-rocessing strategy for real image recognition problem is circumscribed.



(a) (b) (c)

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

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 ~~an~~ end-to-end model, and always begin with convolution or some local region targeted operation, and result in ~~a lack~~ of some intuitionistic statistical information that may be valuable. Although the strong learning ability of efficient DNNs can get a bit of abstract features after long time training, inserting valuable features into networks directly may still enhance the analytical ability.

After IUS pre-processing, the inputs we get are images with simple background and a unified style ~~that~~ one big color block connected with another. ~~It's~~ easy to think of using the color histograms 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. 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 [HZRS16] and GoogleNet_v3 [SVI*15]; 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 the feature inserted layer and the final output layer, as the updated pre-logits layer.

The features that can be inserted into network are not limited to the RGB histogram information, which may be inadaptable for real-world style images since some of them have complicated background and more detailed texture than cartoon images we trained in this article. We can choose other local or global valuable features of an input image such as the sift feature, local color contrast, HSV color histograms, et

4.3. Network Plus Network (NPN)

We propose a new multiple-network structure ~~called~~ NPN. The overall structure of NPN is a paratactic connection of several pre-

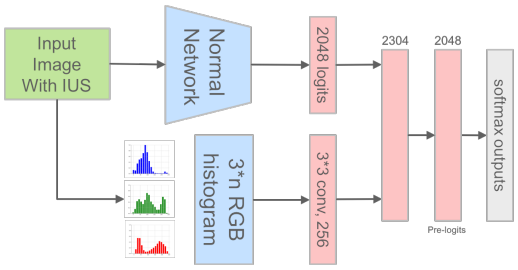


Figure 7: *An example of a Feature Inserted Network structure, with resized (151×151) RGB image, which has preprocessed by IUS. The global feature used is the color histogram in RGB channels, and each histogram merged into n bins.*

logits layers of normal networks, and a final convolutional layer like a simple CNN model. Figure 8 is an overview of a sample NPN. NPN structure ~~can be divided into~~ two parts. The first ~~part~~ is a pre-training structure. There could be m networks used here (N_1,N_2,$\cdots$, N_m). These m networks are pre-trained using the same database, so actually each network already has some ability of image recognition, and they may have their unique features for a same input image. We reserve the part of these networks that the layers before the final fc layer, and retrain them as a mixed ~~network.~~ The other part 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 in the end like traditional DNNs.

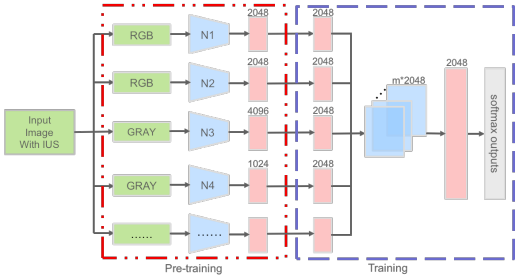


Figure 8: *Overview of a Network Plus Network structure, with a resized RGB image, which has preprocessed by IUS. The left part is the pre-training part of NPN, which contains m normal networks, and these networks have already been trained for the same input dataset. And the right part is the training part, whose inputs are the pre-logits layers of the m networks.*

~~**How to train a NPN model**~~ Training a NPN model is ~~easy but~~ time-consuming. We should train m normal networks on the dataset at the beginning. These m networks ~~can~~ 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.

~~It is worth noting that~~ in m=1, ~~that is to say~~ we only choose one normal network, and ~~in~~ 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 pre-train [GR06], 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 front end, utilizing a server with Nvidia GeForce GTX 980 with CuDNN 6. 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 test, and we use the same data (the random 70%) in all experiments. All these traditional network use Dropout [HSK*12] and Batch-Normalization [IS15] as regularizers to improve the generalization and inference ability and prevents overfitting. We train use a 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 5.1 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, much less than the cost of the total network (see Figure 13).

	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 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.*

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 real world may have more complicated details. The results in Figure 9 show the difference: RGB hisogram figure of cartoon-style images may have more distinct peaks, especially the images processed by IUS.

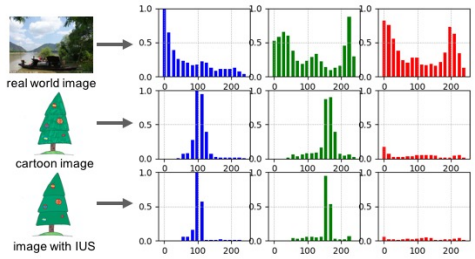


Figure 9: *An example of RGB histogram. Right side are the normalized RGB histogram of real world image, cartoon image and the cartoon image with IUS.*

In our experiments, FIN can improve the evaluation accuracy as well, as table 5.2 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.

	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

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.*

5.3. Experiments on NPN

NPN is the main structure of our system. In our experiments, NPN show several fantastic features on 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 of networks, 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 have less 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 expeirment. And figure 10 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 vert 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 network. 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.

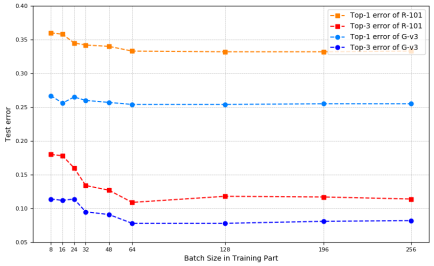


Figure 10: *Different batch size used in the training part of NPN-1 structure. When batch size is up to 64, it is not trivial to keep the accuracy of network.*

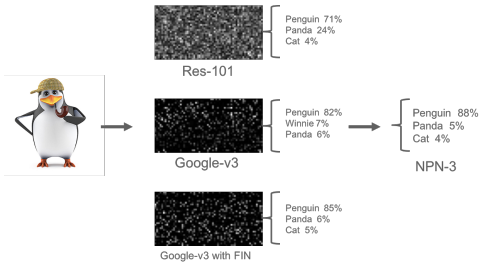


Figure 11: *Visualization of pre-logits layers of three different networks for the same "Penguin" input. The 1×2048-size prelogits layer is resized to 32×64 in this figure. Besides them are the top-3 inference of the networks and the NPN-3 structure, which mixed these three networks.*

Second, NPN-m ($m > 1$) structure has much better one-crop accuracies than traditional state-of-the-art networks. Figure 11 is a convincing prove 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 network used are similar like G-v3 and G-v3 with FIN, the feature sof pre-them will be analogous as well but still have som differences. NPN-m structure is created to combine these features of different networks, and gain a feature more valuable. Table 5.3 shows the outstanding inference performance of NPN-m on our dataset. 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 chosen for NPN, the better performance will 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.

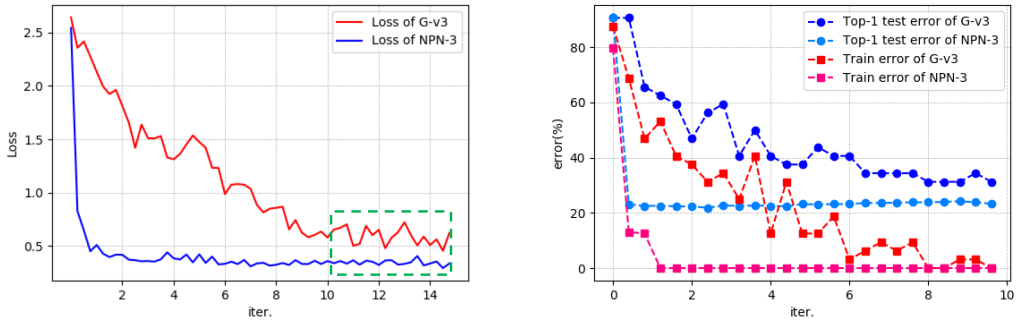


Figure 12: *The comparison of training status between NPN-3 (use R-101, G-v3 and G-v3-FIN) and G-v3. Left: training loss of NPN-3 and G-v3. Right: training error and top-1 error of two networks.*

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

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.*

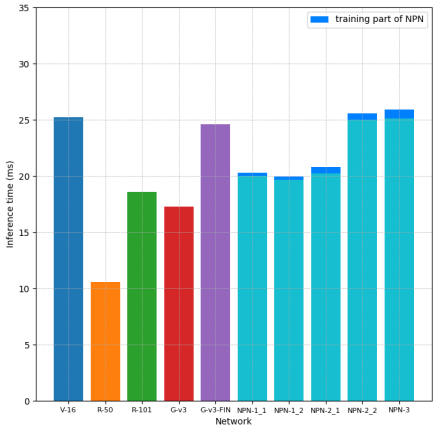


Figure 13: *Inference time per image of different networks with batch of 1 image. Note that the time cost results of networks using FIN involves the time cost of RGB histogram calculation. NPN-1_1: NPN with R-101; NPN-1_2: NPN with G-v3; NPN-2_1: NPN with R-101 and G-v3; NPN-2_2: NPN with G-v3 and G-v3-FIN; NPN-3: NPN with with R-101, G-v3 and G-v3-FIN. And the results of NPNs have been divided into pre-training part and training part.*

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 12 shows, NPN-3 can convergence to an ideal state in 2 iterations. And the convergence will be more smooth and less loss shocks than normal networks (the green frame in subfigure left 12).

Another amazing feature is that NPN structure is born for parallel procession. The FLOPs in pre-traing part can be completely parallel-calculated. As a result, evaluation time of NPN is close to normal networks, although [CPC16] shows "the number of operations is a reliable estimate of the inference time". Figure 13 shows that the evaluation time of NPN is related to the slowest networks it used in pre-training part because of parallel-calculation. And the time cost in training part is negligible since the CNN in this part is shallow.

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.

6. Application

ToonNet provides rich semantic classification information, which can be applied to various problems about cartoon. For instance, we can easily conduct region segmentation to cartoon images. Since we have known the structure of the image using semantic information, we can do segmentation more efficiently and precisely compared to segmentation using only color and edge information. Also, we can extract the skeleton of the cartoon character in the image using semantic information similarly. Given the information, we can define the number of nodes and the topology structure of the skeleton in advance, thus we can extract a more accurate skeleton. Furthermore, when we model the cartoon image, the information

is still very useful, because different types of cartoon characters should be modelled differently. Added semantic information, the modelled character will be more vivid.

As for industry cartoon applications, developers can take advantage of ToonNet as well. Consider an interactive cartoon product. Developers can pre-define some effects like audio or skeleton animation. When a user scans a cartoon image, the character in the image will be recognized by ToonNet, and a corresponding skeleton of the character will be fitted to the image. Then the character will dance and sing using pre-defined resources. Therefore, ToonNet is perfect for creative cartoon applications.

7. 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 4000 images after manual filtration), we introduce several methods to expand our dataset, including snapshots of 3D models, 2D-3D-2D procedure and hand-drawing 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 [CPK*18].

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 an 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 an universal strategy on different situations. At the next stage, we will train NPN on other real-world style image databases, like CIFAR and ImageNet.

References

[Bag12] BAGDANOV A. D.: Color attributes for object detection. In *Computer Vision and Pattern Recognition* (2012), pp. 3306–3313.

[BTTS13] BHARGAVA N., TRIVEDI P., TOSHNIWAL A., SWARNKAR H.: 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* (2013), pp. 157–160. 5

[BWDS17] BOLUKBASI T., WANG J., DEKEL O., SALIGRAMA V.: Adaptive neural networks for efficient inference. 527–536. 3

[CM02] COMANICIU D., MEER P.: Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence 24*, 5 (2002), 603–619.

[CPC16] CANZIANI A., PASZKE A., CULURCIELLO E.: An analysis of deep neural network models for practical applications. *CoRR abs/1605.07678* (2016). URL: http://arxiv.org/abs/1605.07678, arXiv:1605.07678. 3, 8

[CPK*18] CHEN L. C., PAPANDREOU G., KOKKINOS I., MURPHY K., YUILLE A. L.: 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. 9

[DDS*09] DENG J., DONG W., SOCHER R., LI L. J., LI K., LI F. F.: Imagenet: A large-scale hierarchical image database. *Proc of IEEE Computer Vision & Pattern Recognition* (2009), 248–255. 1, 2

[EGW*10] EVERINGHAM M., GOOL L. V., WILLIAMS C. K. I., WINN J., ZISSERMAN A.: The pascal visual object classes (voc) challenge. *International Journal of Computer Vision 88*, 2 (2010), 303–338. 1, 2

[FYX17] FENG L., YANG X., XIAO S.: Magictoon: A 2d-to-3d creative cartoon modeling system with mobile ar. In *Virtual Reality* (2017), pp. 195–204. 1, 2

[GEB16] GATYS L. A., ECKER A. S., BETHGE M.: Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 2414–2423.

[GHP07] GRIFFIN G., HOLUB A., PERONA P.: Caltech-256 object category dataset. *California Institute of Technology* (2007). 1, 2

[GR06] GE H., RR S.: Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507. 2, 6

[HE17] HA D., ECK D.: A neural representation of sketch drawings. *CoRR abs/1704.03477* (2017). URL: http://arxiv.org/abs/1704.03477, arXiv:1704.03477.

[HSK*12] HINTON G. E., SRIVASTAVA N., KRIZHEVSKY A., SUTSKEVER I., SALAKHUTDINOV R. R.: Improving neural networks by preventing co-adaptation of feature detectors. *Computer Science 3*, 4 (2012), págs. 212–223. 6

[HZRS16] HE K., ZHANG X., REN S., SUN J.: Deep residual learning for image recognition. 770–778. 1, 3, 5

[IS15] IOFFE S., SZEGEDY C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR abs/1502.03167* (2015). URL: http://arxiv.org/abs/1502.03167, arXiv:1502.03167. 6

[KH09] KRIZHEVSKY A., HINTON G.: Learning multiple layers of features from tiny images. 1, 2

[KSH12] KRIZHEVSKY A., SUTSKEVER I., HINTON G. E.: Imagenet classification with deep convolutional neural networks. In *International Conference on Neural Information Processing Systems* (2012), pp. 1097–1105. 3

[LC10] LECUN Y., CORTES C.: The mnist database of handwritten digits. 1, 2

[LCY13] LIN M., CHEN Q., YAN S.: Network in network. *Computer Science* (2013). 2, 3

[LQLW17] LIU Y., QIN Z., LUO Z., WANG H.: Auto-painter: Cartoon image generation from sketch by using conditional generative adversarial networks.

[LXJ12] LU C., XU L., JIA J.: Combining sketch and tone for pencil drawing production. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (Goslar Germany, Germany, 2012), NPAR '12, Eurographics Association, pp. 65–73. URL: http://dl.acm.org/citation.cfm?id=2330147.2330161.

[MFE07] MITCHELL J., FRANCKE M., ENG D.: Illustrative rendering in team fortress 2. In *International Symposium on Non-Photorealistic Animation and Rendering* (2007), pp. 71–76.

[RDS*14] RUSSAKOVSKY O., DENG J., SU H., KRAUSE J., SATHEESH S., MA S., HUANG Z., KARPATHY A., KHOSLA A., BERNSTEIN M.: Imagenet large scale visual recognition challenge. *International Journal of Computer Vision 115*, 3 (2014), 211–252. 1, 2

[SLJ*15] SZEGEDY C., LIU W., JIA Y., SERMANET P., REED S., ANGUELOV D., ERHAN D., VANHOUCKE V., RABINOVICH A.: Going deeper with convolutions. 1–9. URL: doi.ieeecomputersociety.org/10.1109/CVPR.2015.7298594, doi:10.1109/CVPR.2015.7298594. 1, 3

[SVI*15] SZEGEDY C., VANHOUCKE V., IOFFE S., SHLENS J., WOJNA Z.: Rethinking the inception architecture for computer vision. *Computer Science* (2015), 2818–2826. 5

[SWRC06] SHOTTON J., WINN J., ROTHER C., CRIMINISI A.: Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *European Conference on Computer Vision* (2006), pp. 1–15. 1, 2

[SZ14] SIMONYAN K., ZISSERMAN A.: Very deep convolutional networks for large-scale image recognition. *CoRR abs/1409.1556* (2014). URL: http://arxiv.org/abs/1409.1556, arXiv:1409.1556. 3

[val] Half lambert. https://developer.valvesoftware.com/wiki/Half_Lambert.

[YS11] YU J., SEAH H.-S.: *Fuzzy diffusion distance learning for cartoon similarity estimation*, vol. 26. Springer, 2011.