Exploratory Research of Object Recognition Based on Deep Learning and Single image Super Resolution

Lisha Han
han.lis@husky.neu.edu
INFO7245 , Spring 2018, Northeastern University

1. Abstract

This research was inspired by the image enhance technique I am learning recently. I did an exploratory research on image classification last semester. I tried to use perceptual hashing to process images before training to reach a better accuracy. The training efficiency and testing accuracy got some increase. However the accuracy is still not ideal. One of many ways to improve classification accuracy is to improve image augmentation. In this research, I will do a little investigation applying SISR technique to add training images more details to improve accuracy.

2. Introduction

In the past few years, deep convolutional neural networks is found a good way to object recognition. CNNs are trained using large collections of diverse images[5]. From these large collections. With this technique, the accuracy of image catalog recognition will reach 95%[3].

Single Image Super-Resolution is used in image enhance and recover. Most algorithms are learning-based methods that learn a mapping between the lo-resolution and hi-resolution image spaces. Among them, the Super-Resolution Convolutional Neural Network (SRCNN) has drawn considerable attention due to its simple network structure and excellent restoration quality[2].

The network contains all convolution layers, thus the size of the output is the same as that of the input image. The overall structure consists of three parts that are analogous to the main steps of the sparse-coding-based methods. The patch extraction and representation part refers to the first layer, which extracts patches from the input and represents each patch as a high-dimensional feature vector. The non-linear mapping part refers to the middle layer, which maps the feature

vectors non-linearly to another set of feature vectors, or namely hi-resolution features. Then the last reconstruction part aggregates these features to form the final output image.

3. Dataset

1. I am using cifar10 downloaded from website(https://www.cs.toronto.edu/~kriz/cifar.html). The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Images document was backed up on Google Cloud Platform. The images are set to public. I created a database to store image type image URL and image labels. User can query from this bucket(https://console.cloud.google.com/storage/browser/image_train?project=image-category-184805). Fig 3.1 is a brief preview of the database image and label table.

ImageID	LabelName	Type	ImageAddr
1	frog	train	https://storage.googleapis.com/image_train/1.png
2	truck	train	https://storage.googleapis.com/image_train/2.png
3	truck	train	https://storage.googleapis.com/image_train/3.png
4	deer	train	https://storage.googleapis.com/image_train/4.png
5	automobile	train	https://storage.googleapis.com/image_train/5.png
6	automobile	train	https://storage.googleapis.com/image_train/6.png
7	bird	train	https://storage.googleapis.com/image_train/7.png
8	horse	train	https://storage.googleapis.com/image_train/8.png
9	ship	train	https://storage.googleapis.com/image_train/9.png
10	cat	train	https://storage.googleapis.com/image_train/10.png

Fig 3.1 Table of images and labels

2. When I use cifar10 training my model I got accuracy around 10%. This means my model is not learning anything. I changed my training dataset to cifar2. cifar2 includes 2 classes of images -- dog and cat. cifar2 was extracted from local image data file. It only have 2 classes. There are

20000 images for training and 2000 for testing. For training and testing purpose I generated 3 subset of data suitable for tensorflow model. I listed their shapes below in Fig 2.2. “Img_test” is made up with numbers under 1. It is for testing accuracy of original CNN model. “Images_train” is for training of original CNN model. “Images_test_distorted” is a set of images was distorted from the “img_test”. It will be used to compare the accuracy of distorted images and non-distorted images.

```
img_test                Shape:  (2000, 32, 32, 3)
images_train            Shape:  (20000, 32, 32, 3)

images_test_distorted   Shape:  (2000, 32, 32, 3)
```

Fig 3.2 Shapes of datasets

4. Overview

4.1 Single image resolution

The goal of Super-Resolution (SR) methods is to recover a high resolution image from one or more low resolution input images. The application of SR can be shown in fig4.1.

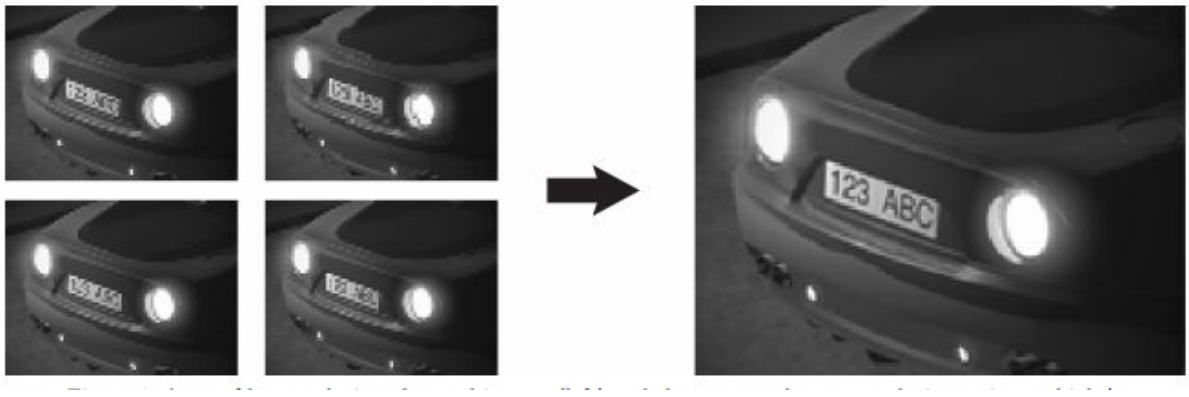

Fig 4.1Hashing values of original image and distorted ones. This image is from internet.

There are many models widely used nowadays. Bicubic Interpolation, Deep Convolutional Neural Networks, Generative Adversarial Neural Networks and Pixel Recursive Super Resolution are most commonly used models. There are a few metrics to evaluate the model. MOS confirmed the superior perceptual performance of SRGAN using mean opinion score. PSNR standard quantitative measures peak signal-to-noise ratio. SSIM structural similarity.

SSIM is used for measuring the similarity between two images. The SSIM index is a full reference metric; in other words, the measurement or prediction of image quality is based on an initial uncompressed or distortion-free image as reference. The most basic and the focus of this research is classification accuracy and training efficiency. Fig 4.1 shows a comparison of different SR models. We can see image processed by bicubic model is blurry. PSNR of SRResNet and SRGAN are both high. And the SSIM of SRResNet and SRGAN are both good.

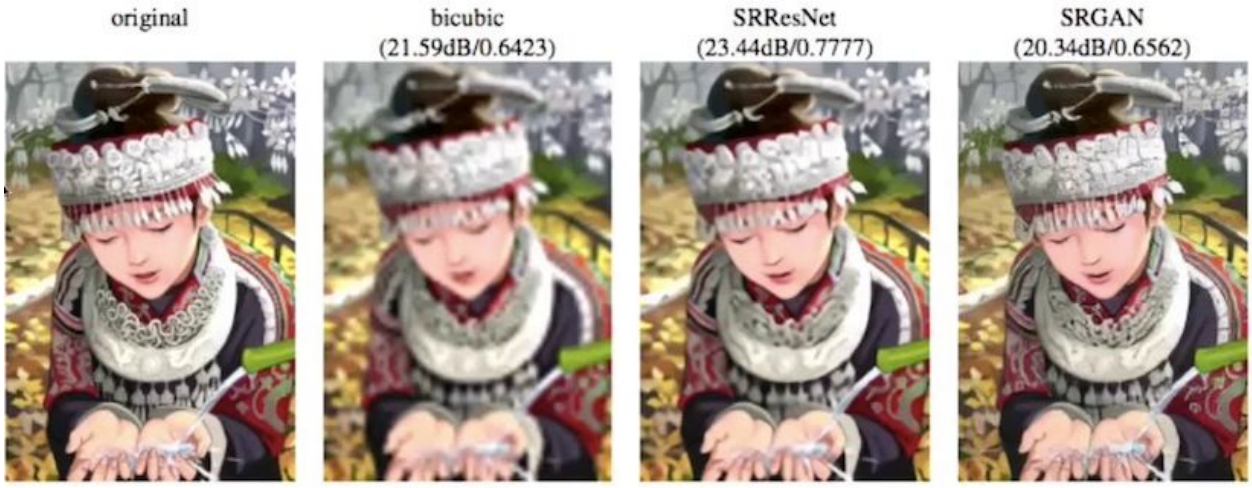

Fig 4.1[2] Hashing values of original image and distorted ones

4.2 SRGAN super resolution generative adversarial neural network

In this model, the model generates images using generator G and random noise. The model performs a batch update of weights in discriminator A given generated images, real images, and labels.

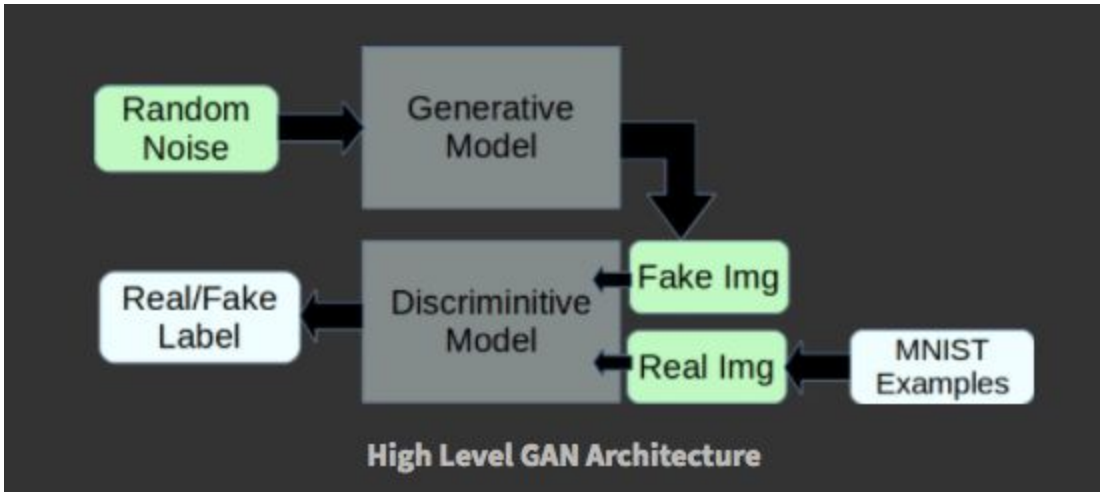

Fig 4.2 Workflow of SRGAN. This image is from internet.

Then the model performs a Batch update of weights in G given noise and forced "real" labels in the full GAN. During the training session, this flow will repeat till the end just as fig 4.2 shown.

5. Experiment and results

5.1 Build a CNN model and evaluate it.

I built a 3-layer CNN model using prettytensor. Fig 5.1 is the architecture of my network.

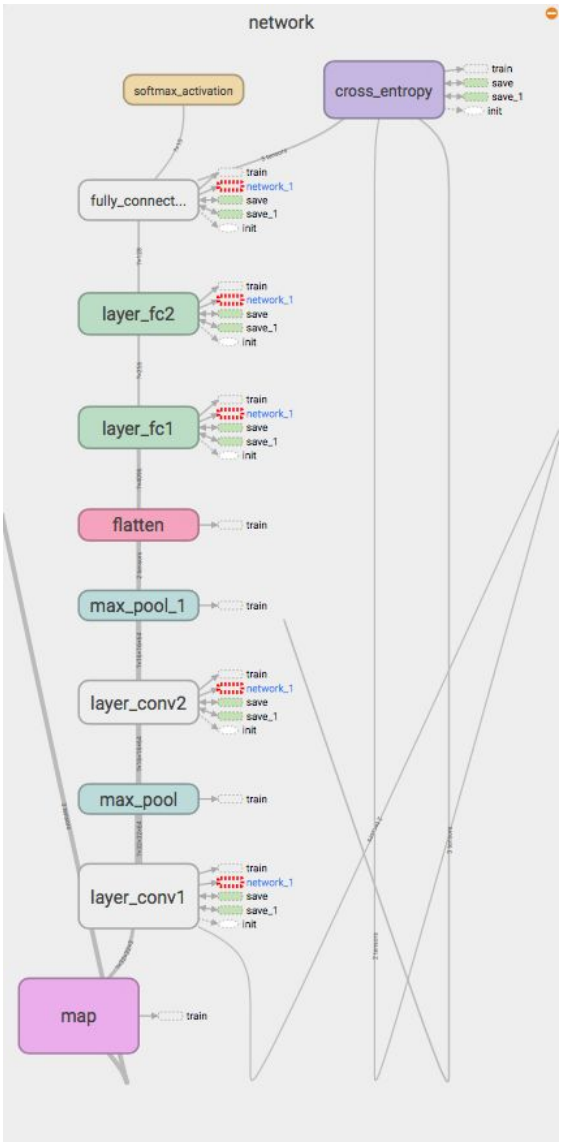

Fig 5.1 Network Structure generated by tensorboard

The activation function is Relu. To improve the accuracy I initialize bias and weights with:

w = tf.Variable(tf.truncated_normal([5,5,3,64], stddev=0.1),name="w")

b = tf.Variable(tf.constant(0.1,shape=[64]), name="b")

I set the random batch size to 64 and the training steps to 10,000. After running for 4 hours, the training accuracy reached 84.4% in 10,000 steps(Fig 5.2).

```
Global Step:   9500, Training Batch Accuracy:  85.9%
Global Step:   9600, Training Batch Accuracy:  84.4%
Global Step:   9700, Training Batch Accuracy:  84.4%
Global Step:   9800, Training Batch Accuracy:  84.4%
Global Step:   9900, Training Batch Accuracy:  78.1%
Global Step:  10000, Training Batch Accuracy:  84.4%
Time usage: 4:22:49
```

Fig 5.2 Training accuracy and steps took

I used tensorboard for monitor. Fig 5.3 shows changes of accuracy and loss. The average_accuracy_cross_entropy got close to 0.25 after training. And the loss cost approached 0.3.

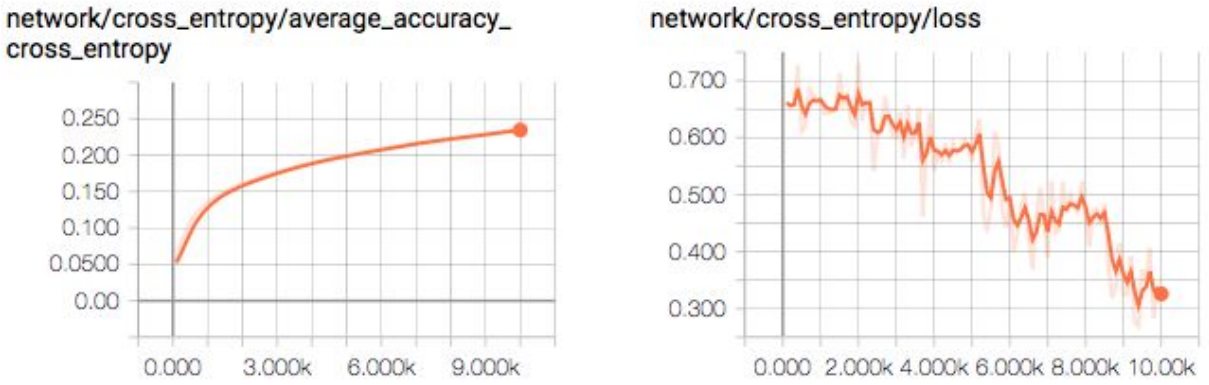

Fig 5.3 average accuracy and loss monitored on tensorboard

After 10,000 steps of training, the accuracy of testing set is 54.6% which is no better than just guessing.

```
batch_size = 256
t1=get_test_acc(batch_size=batch_size, images_test=img_test)
```

```
Accuracy on Test-Set: 54.6% (1092 / 2000)
```

Fig 5.4 Test accuracy of the original model

5.2 Explore of model accuracy improvement

I did some experiments on the original model by changing different settings. I changed activation function to ELU and TanH. Using ELU takes less time than using ReLU to reach network plateaus. But the accuracy of testing set is no better. I changed the cost function to cross-entropy and hinge. Their training accuracy were hanging around 50%. They should at least reach 90% to be ready for testing. Then I tried to run the training with different epochs. They both took longer to reach the same training accuracy. Plus there was no improvement on the model. Changing gradient estimation, network architecture and initialization are not helping in either training efficiency nor the accuracy improvement.

	name	train accuracy	test accuracy	time
0	Original	0.921875	0.500	1:40:09
1	ELU	0.920000	0.493	1:28:05
2	TanH	0.930000	0.489	1:54:59
3	Cross Entropy	0.421875	0.375	0:14:15
4	Hinge	0.500000	0.500	0:54:21
5	Epoch 100	0.910000	0.488	2:17:15
6	Epoch 128	0.929688	0.471	2:28:09
7	Gradient Descent	0.580000	0.541	1:52:26
8	Adagrad	0.531250	0.491	1:09:46
9	Filter Size	0.906250	0.523	2:16:38
10	4 Layers	0.921875	0.510	2:15:00
11	Xavier	0.656250	0.492	1:20:21
12	Uniform	0.500000	0.472	1:08:31

Fig 5.5 Result comparison of different changes made on the original model

I decided to keep using the original model I've been using for future investigation. Fig 5.5 is a comparison of different changes of the model.

5.3 Improvement proposal

Deep learning models have achieved great success on image classification tasks [5]. What if we use distorted images as our test set. I used distorted image set to evaluate the model I got accuracy of 49.6% as in fig5.6.

```
batch_size = 256
t_t=get_test_acc(batch_size=batch_size,images_test=images_test_distorted)
```

```
Accuracy on Test-Set: 49.6% (993 / 2000)
```

Fig 5.6 Test accuracy of distorted images

The distorted images were just set contrast of 200 from the original ones. There was 5% drop of the accuracy. Is there anyway to improve the image resolution of the training dataset? For the next part I will build SRGAN model and train it. Then I will use the trained model to pre-process the cifar2 image dataset. And I will use the new dataset to train the CNN model I built.

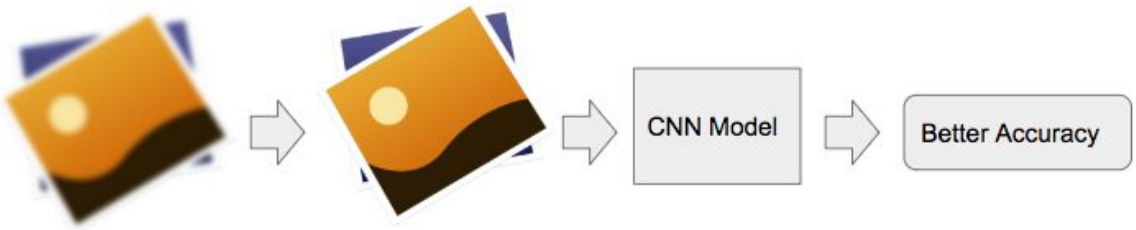

Fig 5.7 Improvement proposal

5.4 Build SRGAN model and train

In this part, I will keep using cifar2 to train and get processed data during training.

Layer (type)	Output Shape	Param #
dense_11 (Dense)	(None, 256)	25856
leaky_re_lu_8 (LeakyReLU)	(None, 256)	0
batch_normalization_4 (Batch	(None, 256)	1024
dense_12 (Dense)	(None, 512)	131584
leaky_re_lu_9 (LeakyReLU)	(None, 512)	0
batch_normalization_5 (Batch	(None, 512)	2048
dense_13 (Dense)	(None, 1024)	525312
leaky_re_lu_10 (LeakyReLU)	(None, 1024)	0
batch_normalization_6 (Batch	(None, 1024)	4096
dense_14 (Dense)	(None, 3072)	3148800
reshape_2 (Reshape)	(None, 32, 32, 3)	0

Total params: 3,838,720
Trainable params: 3,835,136
Non-trainable params: 3,584

Fig 5.8.1 The configuration of generator and discriminator

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               1573376
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 257
=================================================================
Total params: 1,704,961
Trainable params: 1,704,961
Non-trainable params: 0
_________________________________________________________________
```

Fig 5.8.2 The configuration of generator and discriminator

After training for 41 minutes and 10k steps. The training accuracy is around 90% as shown in fig 5.9.

```
train_gan(epochs=10000, batch_size=32, save_interval=200)
8900 [D loss: 0.395069, acc.: 84.38%] [G loss: 2.901890]
9000 [D loss: 0.251203, acc.: 89.06%] [G loss: 4.574396]
9100 [D loss: 0.228056, acc.: 89.06%] [G loss: 4.837804]
9200 [D loss: 0.337465, acc.: 89.06%] [G loss: 2.739313]
9300 [D loss: 0.427032, acc.: 79.69%] [G loss: 4.829577]
9400 [D loss: 0.187012, acc.: 92.19%] [G loss: 4.654322]
9500 [D loss: 0.366167, acc.: 78.12%] [G loss: 3.982394]
9600 [D loss: 0.176786, acc.: 92.19%] [G loss: 4.840919]
9700 [D loss: 0.397459, acc.: 82.81%] [G loss: 5.224366]
9800 [D loss: 0.216768, acc.: 92.19%] [G loss: 5.225894]
9900 [D loss: 0.192573, acc.: 90.62%] [G loss: 3.722124]
```

Fig 5.9 Training result of SRGAN model

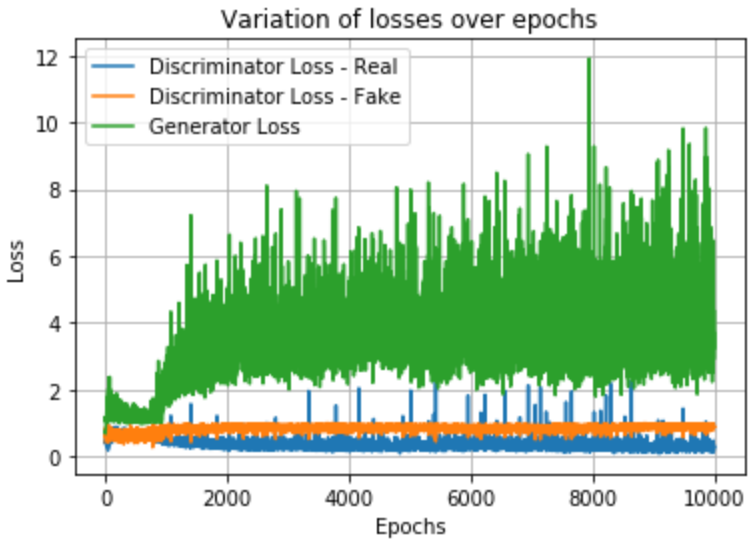

Fig 5.10 Loss of Discriminator and Generator

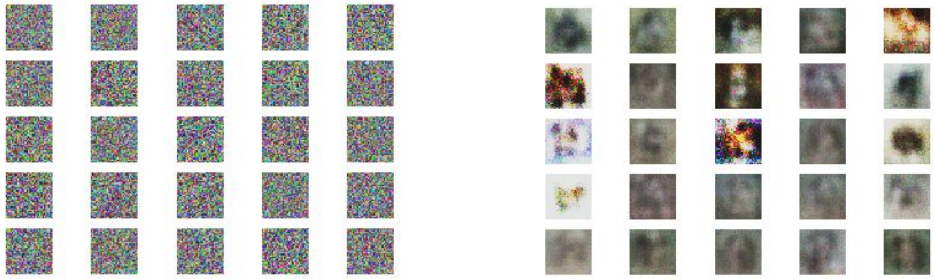

Fig 5.10 Outcome images after the training

Fig 5.10 shows the original image and the final image generated using SRGAN model. We can see from the final image there is some images showed up. But still we still cannot tell what is in the images. The model needs more training.

```
train_gan(epochs=50000, batch_size=32, save_interval=200)
48900 [D loss: 0.437374, acc.: 87.50%] [G loss: 2.931694]
49000 [D loss: 0.347780, acc.: 81.25%] [G loss: 4.499424]
49100 [D loss: 0.263484, acc.: 87.50%] [G loss: 4.019207]
49200 [D loss: 0.173457, acc.: 92.19%] [G loss: 4.601897]
49300 [D loss: 0.203441, acc.: 92.19%] [G loss: 4.358111]
49400 [D loss: 0.307258, acc.: 89.06%] [G loss: 3.500326]
49500 [D loss: 0.280723, acc.: 82.81%] [G loss: 3.311149]
49600 [D loss: 0.250430, acc.: 87.50%] [G loss: 5.636917]
49700 [D loss: 0.276778, acc.: 87.50%] [G loss: 3.249075]
49800 [D loss: 0.408126, acc.: 82.81%] [G loss: 4.308856]
49900 [D loss: 0.378515, acc.: 82.81%] [G loss: 3.709414]
Time usage: 3:03:36
```

Fig 5.11 Training result of SRGAN model

Than I trained the model with 50k steps. The training took about 3 hours. The training accuracy is around 87%.

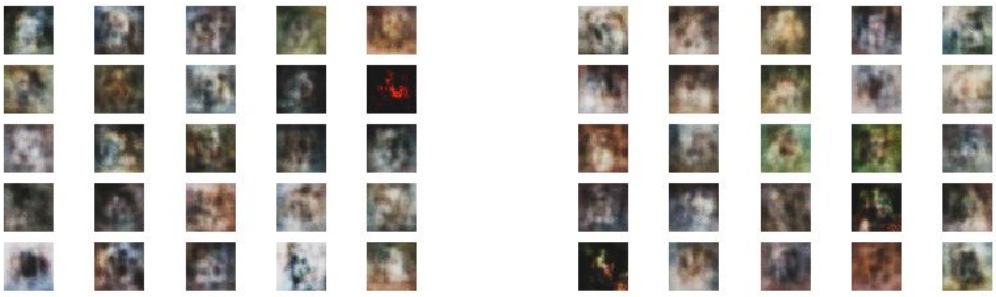

Fig 5.12 Outcome images after the training

Fig 5.12 shows the beginning image and the final image from step 10001 and 69800. The figures in the images become much clearer. If we train the model more we will get high resolution images as expected.

5.5 Use pre-processed data to train CNN model

The dataset got processed during training of SRGAN model. The shape of it is (10000, 32, 32, 3). This is the same as the input of CNN model.

```
Global Step:   9200, Training Batch Accuracy:  87.5%
Global Step:   9300, Training Batch Accuracy:  67.2%
Global Step:   9400, Training Batch Accuracy:  81.2%
Global Step:   9500, Training Batch Accuracy:  81.2%
Global Step:   9600, Training Batch Accuracy:  89.1%
Global Step:   9700, Training Batch Accuracy:  76.6%
Global Step:   9800, Training Batch Accuracy:  82.8%
Global Step:   9900, Training Batch Accuracy:  85.9%
Global Step:  10000, Training Batch Accuracy:  84.4%
Time usage: 2:27:24
```

Fig 5.13 Training accuracy and time of CNN model

As Fig 5.15 shows. The average_accuracy_cross_entropy got close to 0.22 after training. And the loss cost approached 0.3.

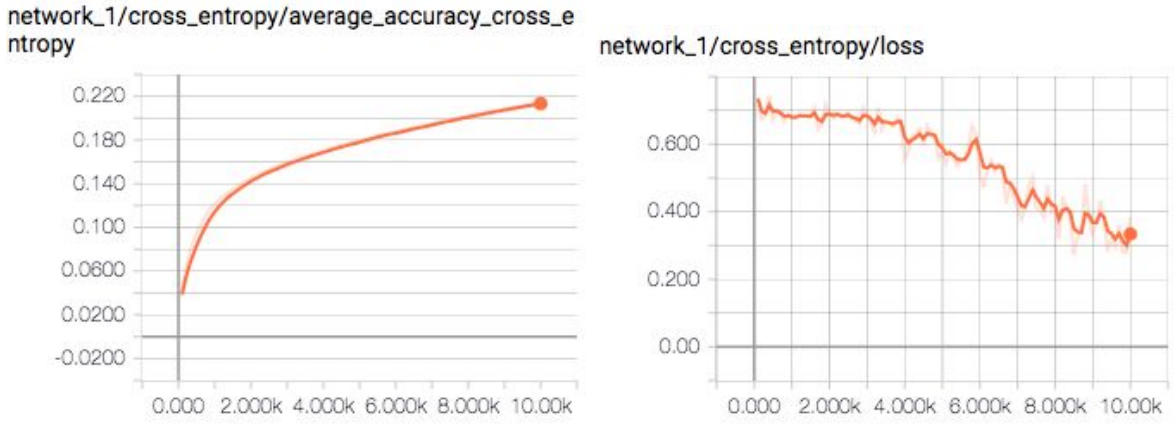

Fig 5.15 Training accuracy and time of CNN model

After 10k steps and 2.5 hours training. The training accuracy reached similar as the original model. Then I tested the model with pre-processed testing dataset. The accuracy dropped to 51.2% as shown in Fig 5.15.

```
batch_size = 256
t_gan=get_test_acc(batch_size=batch_size, images_test=test_gan)
```

```
Accuracy on Test-Set: 51.2% (1025 / 2000)
```

Fig 5.15 Testing accuracy of CNN model

6. Conclusion & Discussion

In this research so far, I got the image classification testing accuracy of 54%. It took 4 hours and 10k steps to train the model. The accuracy of SRGAN model is 90%. It took around 1 hour and 10k steps to train it. I tried to explore the way to improve accuracy and efficiency by changing activation functions, model architectures, training epochs, loss function and so on. They are not g helping very much with the improvement. But I got a better configuration. Using relu as activation function and initialise the model with bias and weights of below values.

w = tf.Variable(tf.truncated_normal([5,5,3,64], stddev=0.1),name="w")

b = tf.Variable(tf.constant(0.1,shape=[64]), name="b")

And set the batch size to 64.

Attributes	Original	SRGAN
training accuracy	84.40%	84.40%
steps to reach 90% accuracy	9000	more than 10000
training time for 10000 steps	4 h	2.5h
average_accuracy_cross_entropy	0.23	0.22
loss cost	0.32	0.3
test accuracy	54.60%	51.20%

Fig 6.1 Experiment result and comparison

The experiment results are shown in Fig 6.1. There is a 37.5% improvement of training efficiency. And the test accuracy dropped from 54.6% to 51.2%. There are not very big difference for the other attributes. However it took a long time to train the SRGAN model. If you want to get better photo quality you will need to train more. And this will take longer. If I take the training of SRGAN model into consideration, the training efficiency of orginal CNN model is better.

There is still room to make improvement. Unfortunately, I didn't run the training on the cloud. It took me lots of time to train my model. It's not easy to monitor and compare the result if I want to make any change to the model. The original model still needs more improvement. There are some ways to improve the accuracy. Such as training the CNN models more. And also we can improve the quality of dataset by training SRGAN model more. There are lots of SRGAN

models. I can try other models. I can also change the activation function, architecture, loss function or initialization of the model. Till now, the CNN model based on SRGAN has little improvement from original CNN. There is still long way to go.

7. Code with documentation

Code of this research is available on my github.
https://github.com/lishahan/CSYE7245_2018Spring/tree/master/FinalProject

8. References

[1] *Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.*

[2] *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, ´ Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi Twitter, 2016*

[3] *Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising, Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang, 2017*

[4] *Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network, Wenzhe Shi, Jose Caballero, Ferenc Huszar´ , Johannes Totz, Andrew P. Aitken, Rob Bishop1 , Daniel Rueckert, Zehan Wang, 2016*

[5] *A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, pages 1106–1114, 2012*

[6] *Learning Fine-grained Image Similarity with Deep Ranking by Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, Ying Wu1*

[7] *Accelerating the Super-Resolution Convolutional Neural Network, Chao Dong, Chen Change Loy, and Xiaoou Tang, 2016*

[8] *Is the deconvolution layer the same as a convolutional layer? Wenzhe Shi, Jose Caballero, Lucas Theis, Ferenc Huszar, Andrew Aitken, Christian Ledig, Zehan Wang,2016*