Exploratory Research of Object Recognition Based on Deep Learning and Image Perceptual Hashing

Lisha Han
han.lis@husky.neu.edu
INFO7390 , Fall 2017, Northeastern University

1. Abstract

This research was inspired by the work I did about perceptual hashing for image during undergraduate. When we encounter images with lower resolution will it still keep the high accuracy using original CNN model? Perceptual hashing in image recognition keeps its robust when it encounter low distorted images. And it is pretty efficient in image recognition. In this research I will do a little investigation on the combination of perceptual hashing algorithm and deep learning to reach better accuracy and efficient computation.

2. Introduction

Images recognition has become an important function of search engine and shopping apps. With this function people can search item with images they uploaded. How to recognize object and allocate a proper label is in demand. In the past few years, convolutional neural networks is found an efficient way to object recognition. Deep learning models have achieved great success on image classification tasks [5].

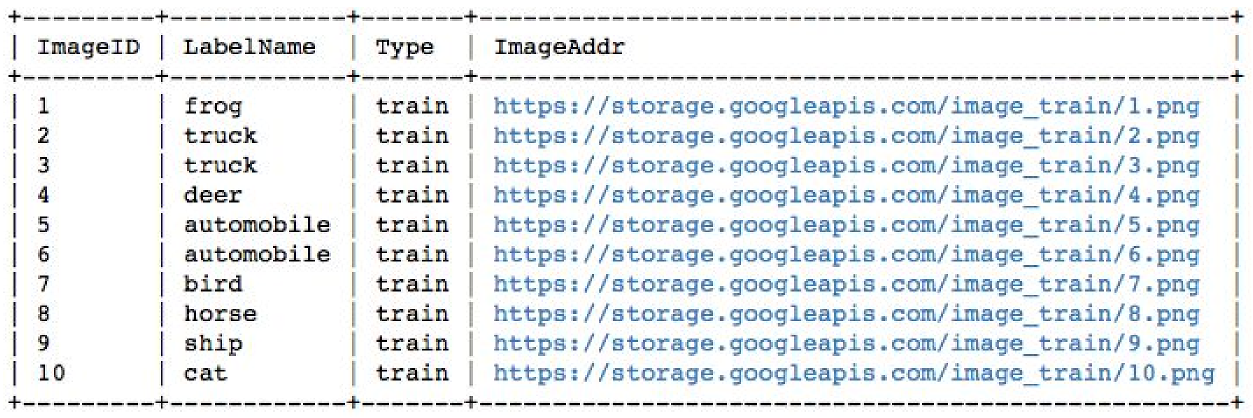
Perceptual hashes are hashing functions that map source data into hashes while maintaining correlation. These types of functions allow us to make meaningful comparisons between hashes in order to indirectly measure the similarity between the source data. Image perceptual hashing is a proved effective way of comparing similar images. It is widely used in images search. This technique is quick and simple and has good robustness and accuracy. The fingerprint created by perceptual image hashing technique is unique and it wouldn't change with its color, shape or resolution ratio.

3. Dataset

1. I am using cifar10[1] 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 2.1 is a brief preview of the database image and label table.

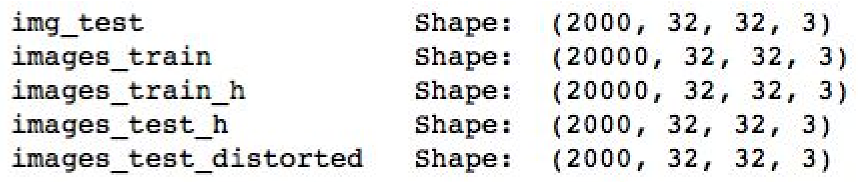


```
+---------+------------+-------+------------------------------------------------+
| ImageID | LabelName  | Type  | ImageAddr                                      |
+---------+------------+-------+------------------------------------------------+
| 1       | frog       | train | https://storage.googleapis.com/image_train/1.png  |
| 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 5 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. “Images_train_h” and “images_test_h” are numpy arrays made up with 1 and 0s(finger print of images). They are used for training and testing on the new CNN model.



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

Fig 3.2 Shapes of datasets

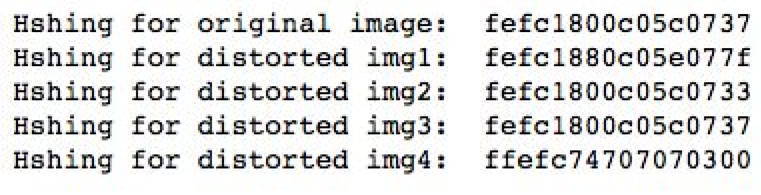
4. Overview

4.1 Image perceptual hashing and robustness

Perceptual hashing is the use of an algorithm that produces fingerprint of various forms of multimedia.[2][3] A perceptual hash is a fingerprint based on the image input, that can be used to compare images by calculating the Hamming distance (which basically means counting the number of different individual bits).

There are a couple of different perceptual image hashing algorithms, but they all use similar steps for generating the media fingerprint. The easiest one to explain is the Average Hash (also called aHash). Let's take the following image and see how it works. Returning to our query image we can compare it against modified versions of itself in order to test the robustness of our perceptual hash comparison.[4]

Fig 4.1 shows hashing values calculated using average hashing. We can get a brief concept that different images generated distorted from same image have almost the same hashings.

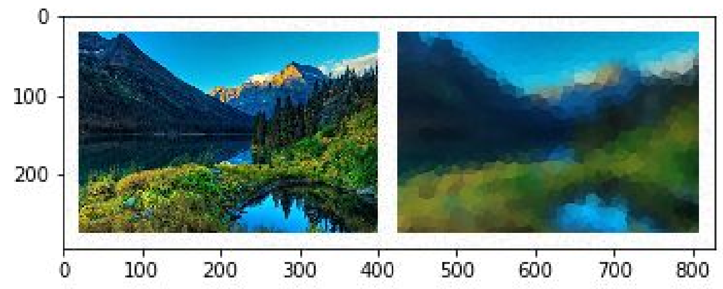


```
Hshing for original image:  fefc1800c05c0737
Hshing for distorted img1:  fefc1880c05e077f
Hshing for distorted img2:  fefc1800c05c0733
Hshing for distorted img3:  fefc1800c05c0737
Hshing for distorted img4:  ffefc74707070300
```

Fig 4.1Hashing values of original image and distorted ones

1. Compare hamming distance between unmodified query image and a distorted version of the query image created using the crystalize filter in Photoshop. The hamming distance is 4.

```
imgplot = plt.imshow(Image.open('/Users/lisha/Desktop/show_images/1.jpg'))
plt.show()
print('hamming distance: ',hamming(h0,h1))
```

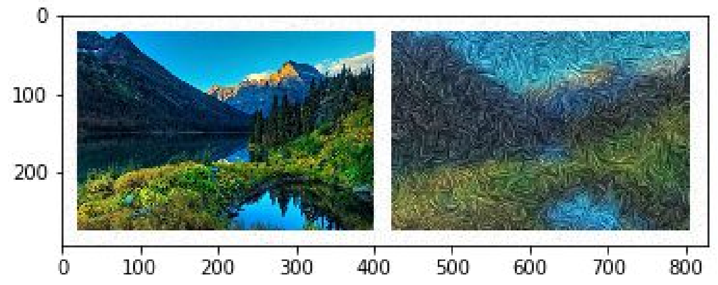


```
hamming distance:  4
```

Fig 4.2 Hamming distance between crystalize image and original one

2. Compare hamming distance between unmodified query image and a distorted version of the query image created by adding a visual swirl over the image. The hamming distance is 0.

```
imgplot = plt.imshow(Image.open('/Users/lisha/Desktop/show_images/3.jpg'))
plt.show()
print('hamming distance: ',hamming(h0,h3))
```

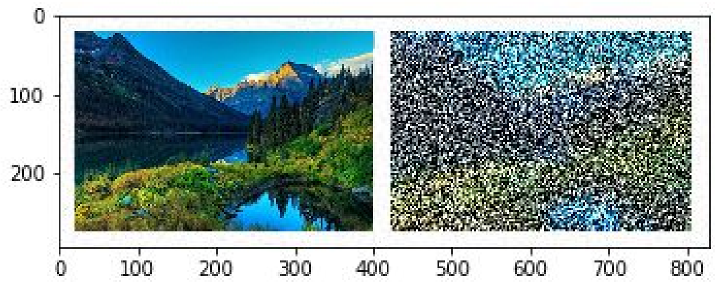


```
hamming distance:  0
```

Fig 4.3 Hamming distance between swirl image and original one

3. Compare hamming distance between unmodified query image and a distorted version of the query image created by increasing the contrast and adding significant high frequency noise. The hamming distance is 1.

```
imgplot = plt.imshow(Image.open('/Users/lisha/Desktop/show_images/2.jpg'))
plt.show()
print('hamming distance: ',hamming(h0,h2))
```

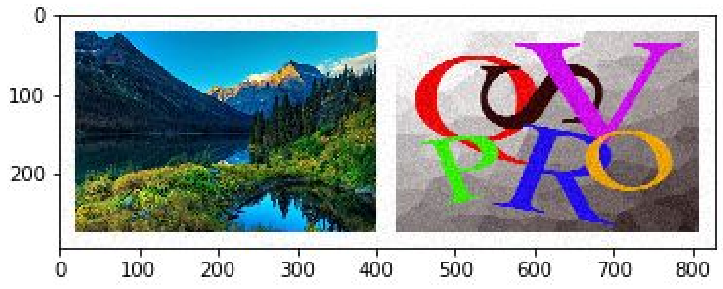


```
hamming distance:  1
```

Fig 4.4 Hamming distance between noise added image and original one

4. Compare hamming distance between unmodified query image and a simple generated image that is deliberately different from our query image. We got a hamming distance of 14.

```
imgplot = plt.imshow(Image.open('/Users/lisha/Desktop/show_images/4.jpg'))
plt.show()
print('hamming distance: ',hamming(h0,h4))
```



```
hamming distance:  14
```

Fig 4.5 Hamming distance between original image and a random one

The greatest hamming distance is 16. Based on investigation above, we can see different images generated distorted from same image got very small hamming distance. And Fig 4.5 shows that different images have big hamming distance.

4.2 Hamming distance of cifar10 dataset and comparison of different hashing algorithms

I chose images from dog, cat and truck catalog for comparison. I calculated the hamming distances between similar images whose images have same label and different images come from different catalogs.

1. Average Hashing

Plot hamming distances between different catalogs and same catalog. Most white spots stay up high.

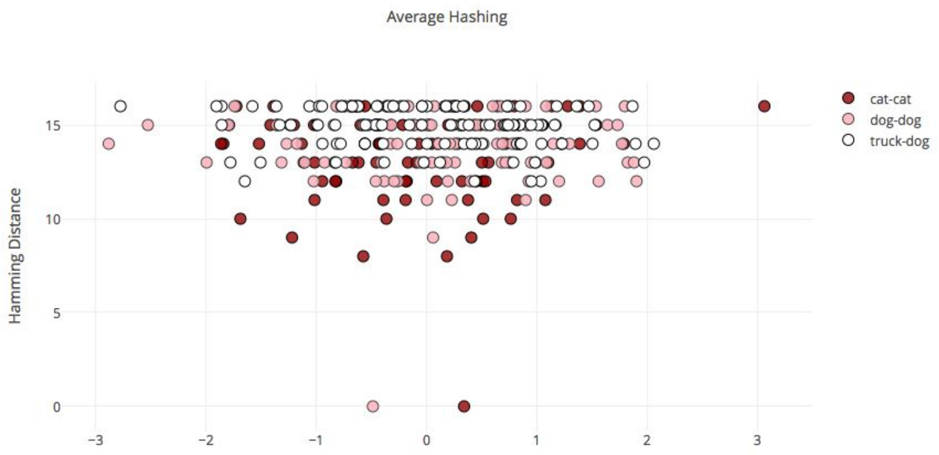


Fig 4.6 Scatter plot of hamming distance between different catalogs using average hashing

2. P-Hashing

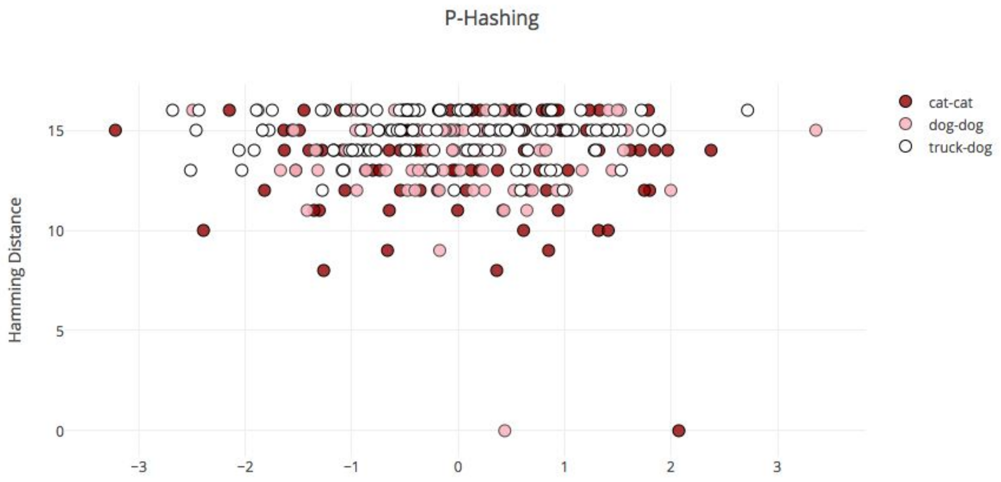


Fig 4.7 Scatter plot of hamming distance between different catalogs using p-hashing

3. D-Hashing

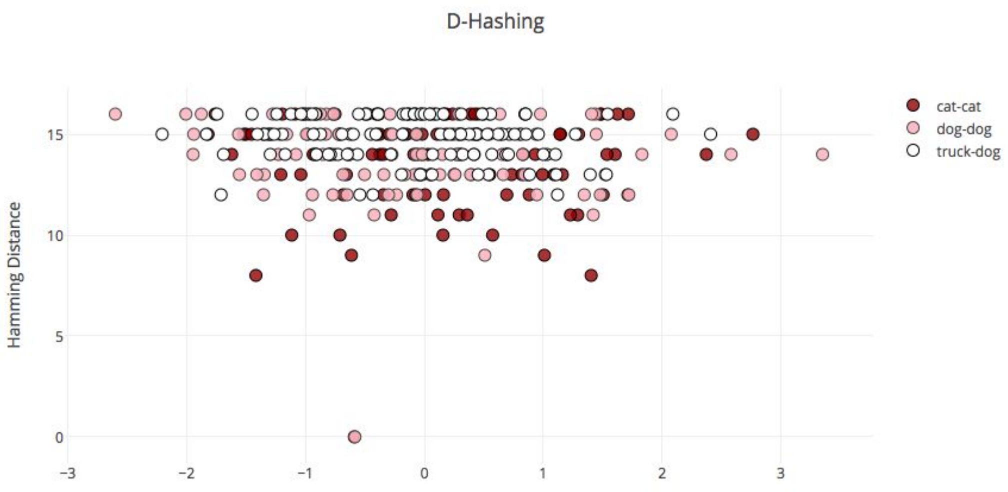
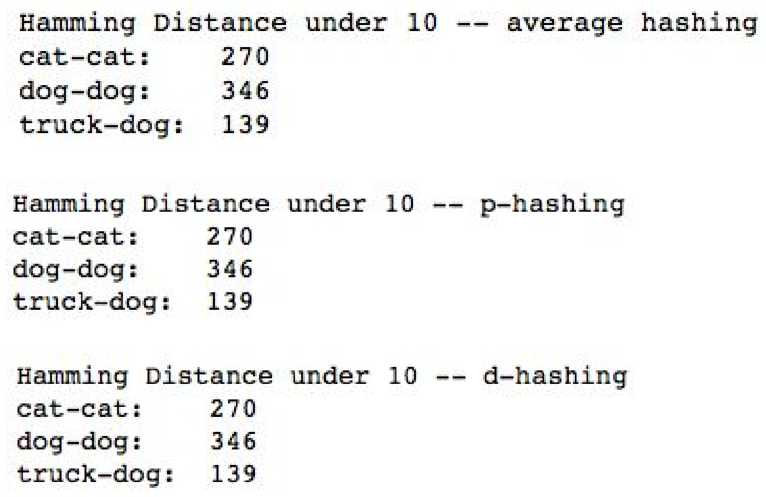


Fig 4.8 Scatter plot of hamming distance between different catalogs using d-hashing

4. Comparison of 3 perceptual hashing algorithms

I calculated images hamming distances under 10 with 3 types of hashing algorithm. The result indicated that they all have the same performance. So I decided to use the most common hashing-- average hashing for future investigation.



```
Hamming Distance under 10 -- average hashing
cat-cat:    270
dog-dog:    346
truck-dog:  139

Hamming Distance under 10 -- p-hashing
cat-cat:    270
dog-dog:    346
truck-dog:  139

Hamming Distance under 10 -- d-hashing
cat-cat:    270
dog-dog:    346
truck-dog:  139
```

Fig 4.9 Count hamming distance under 10

5. Experiment

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.

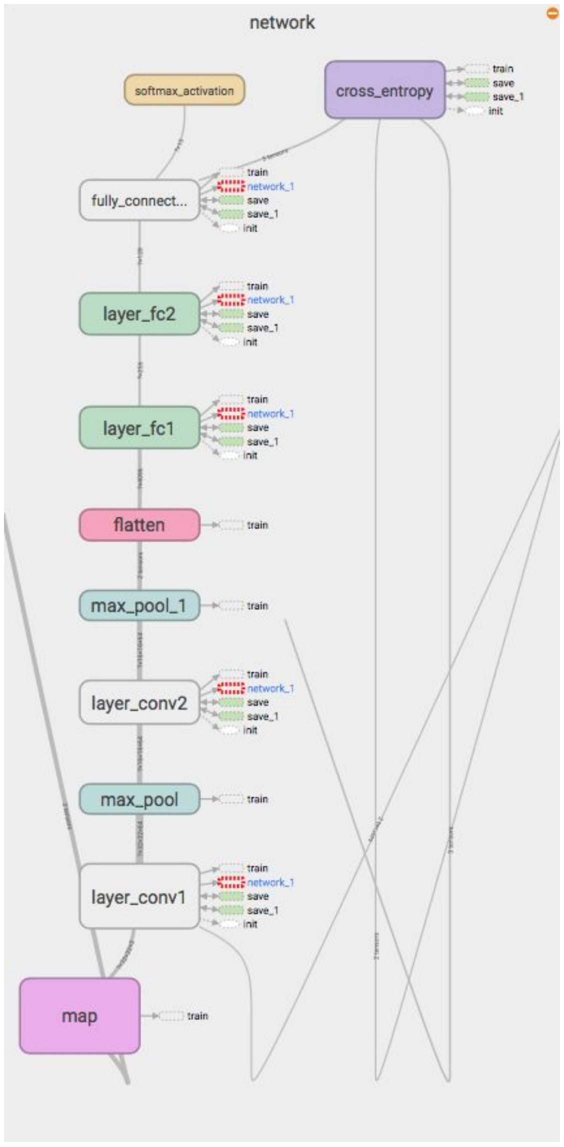


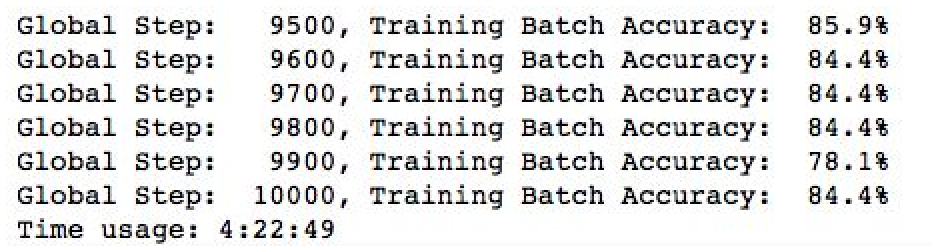
Fig 5.1 Network Structure generated by tensorboard

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.

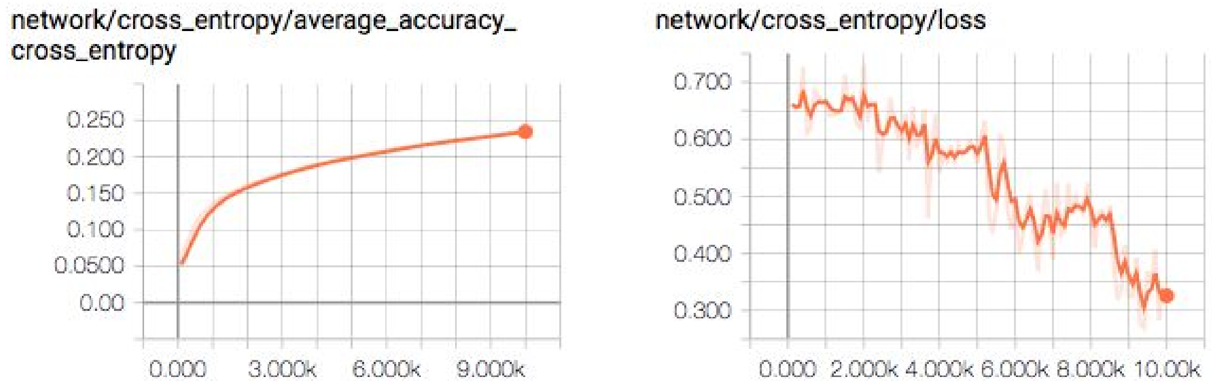
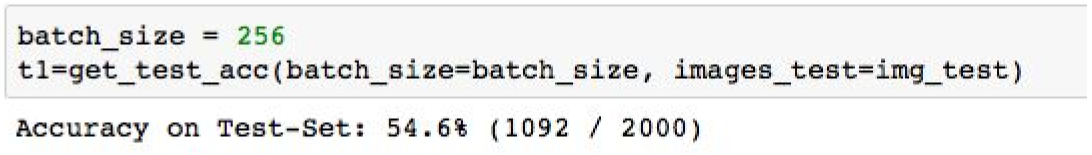


Fig 5.3 average accuracy and loss monitored on tensorboard

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



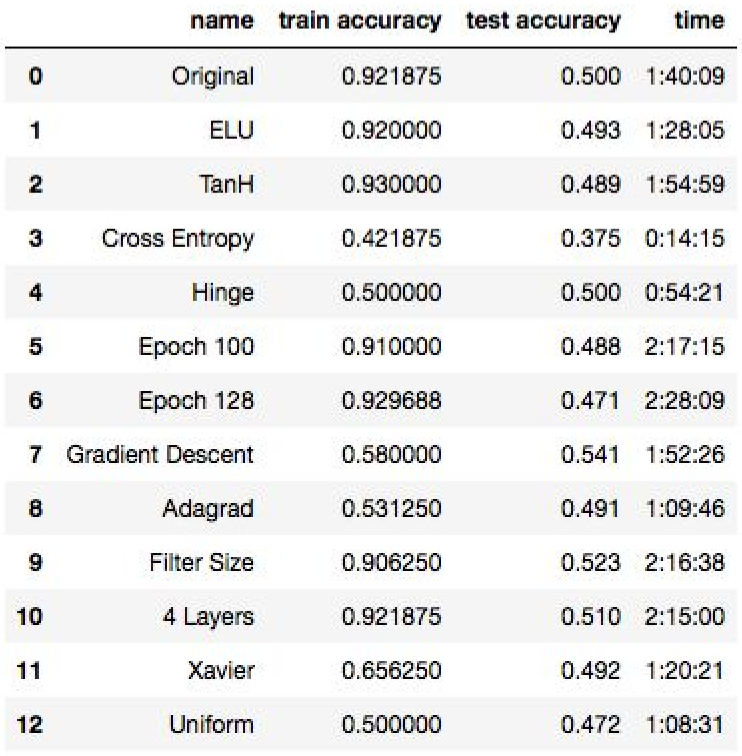
```
batch_size = 256
t1=get_test_acc(batch_size=batch_size, images_test=img_test)

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.



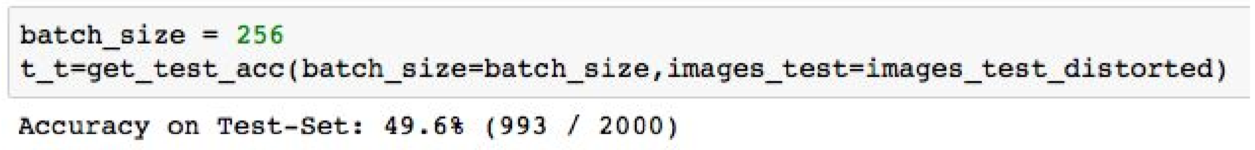
	name	train accuracy	test accuracy	time
0	Original	0.921875	0.500	1:40:09
1	ELU	0.920000	0.493	1:28:05
2	TanH	0.930000	0.489	1:54:59
3	Cross Entropy	0.421875	0.375	0:14:15
4	Hinge	0.500000	0.500	0:54:21
5	Epoch 100	0.910000	0.488	2:17:15
6	Epoch 128	0.929688	0.471	2:28:09
7	Gradient Descent	0.580000	0.541	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 Use images pre-processed by average hashing to train the model

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%. The distorted images were just set contrast of 200 from the original ones. There was 5% drop of the accuracy. Is there any way to make the model more robust?

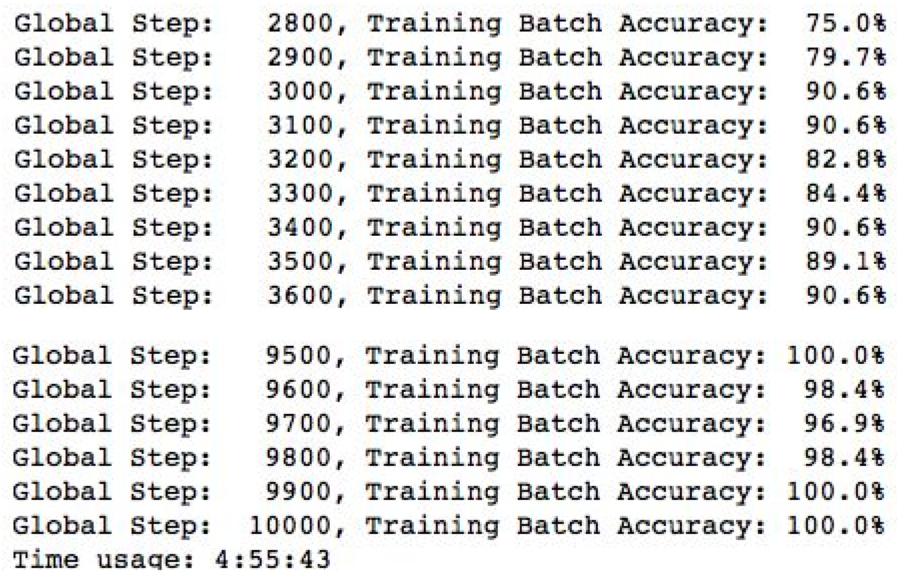


```
batch_size = 256
t_t=get_test_acc(batch_size=batch_size,images_test=images_test_distorted)
```
```
Accuracy on Test-Set: 49.6% (993 / 2000)
```

Fig 5.6 Test accuracy of distorted images

"Images_train_h" is the dataset I generate from cifar2. I calculated the average pixels of each image. And then I compare each pixel with the average. If the pixel is above the average, the value of this pixel should be 1. Otherwise the value will be 0. And then reshape the dataset to (20000, 32, 32, 3) which is suitable for the model.

The training batch was set to 64. And the training steps was set to 10,000. After Running the training for 3600 steps. The training accuracy already reached the same as the original model. See Fig 5.6. This is a big improvement on the training efficiency.



```
Global Step:   2800, Training Batch Accuracy:  75.0%
Global Step:   2900, Training Batch Accuracy:  79.7%
Global Step:   3000, Training Batch Accuracy:  90.6%
Global Step:   3100, Training Batch Accuracy:  90.6%
Global Step:   3200, Training Batch Accuracy:  82.8%
Global Step:   3300, Training Batch Accuracy:  84.4%
Global Step:   3400, Training Batch Accuracy:  90.6%
Global Step:   3500, Training Batch Accuracy:  89.1%
Global Step:   3600, Training Batch Accuracy:  90.6%

Global Step:   9500, Training Batch Accuracy: 100.0%
Global Step:   9600, Training Batch Accuracy:  98.4%
Global Step:   9700, Training Batch Accuracy:  96.9%
Global Step:   9800, Training Batch Accuracy:  98.4%
Global Step:   9900, Training Batch Accuracy: 100.0%
Global Step:  10000, Training Batch Accuracy: 100.0%
Time usage: 4:55:43
```

Fig 5.7 Training accuracy and steps took

After training for 10,000 steps. I checked the tensorboard. The average_accuracy_cross_entropy got close to 0.3. And the loss cost is almost 0. Some improvement after training using perceptual hashing pre-processed images.

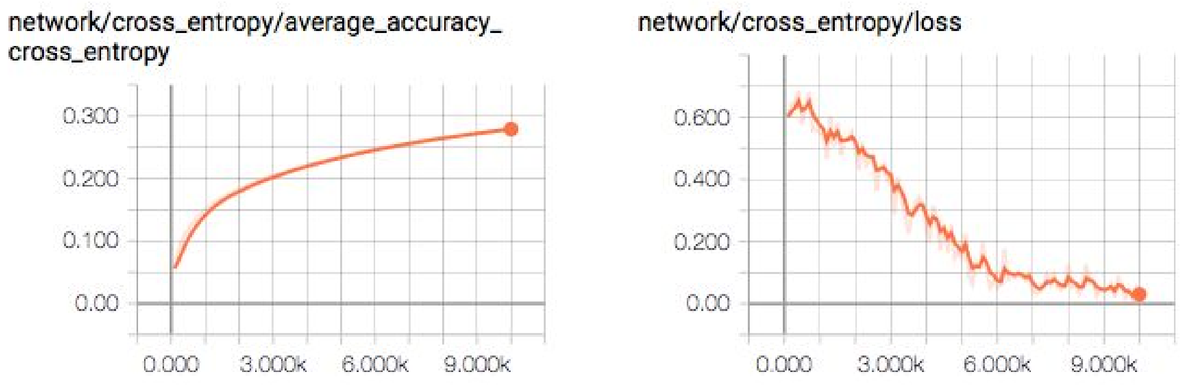
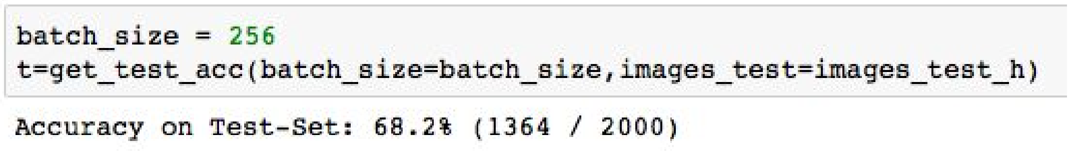


Fig 5.8 Average accuracy and loss monitored from tensorboard

Let’s check the testing accuracy out. The training accuracy improved from 54.6% to 68.2%.



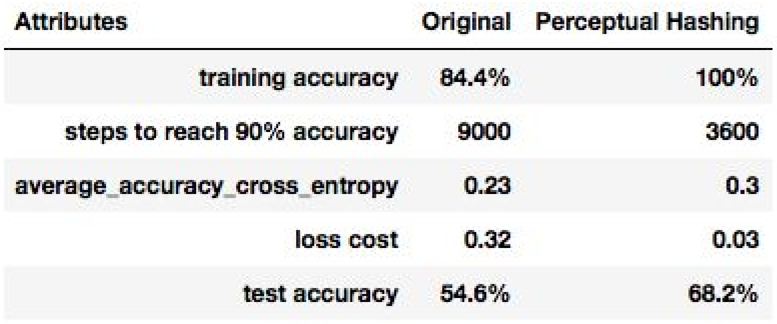
```
batch_size = 256
t=get_test_acc(batch_size=batch_size,images_test=images_test_h)

Accuracy on Test-Set: 68.2% (1364 / 2000)
```

Fig 5.9 Test accuracy of new model

6. Conclusions

In this research the model got improved in testing accuracy by 13.6%. And the loss cost reduced by 0.29. The training steps need be taken deduced by 64%. The result is in Fig 6.1.



Attributes	Original	Perceptual Hashing
training accuracy	84.4%	100%
steps to reach 90% accuracy	9000	3600
average_accuracy_cross_entropy	0.23	0.3
loss cost	0.32	0.03
test accuracy	54.6%	68.2%

Fig 6.1 Experiment result and comparison

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. And The data collection is limited. 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. Based on good testing accuracy training with perceptual hashing processed images may get even better accuracy.

7. References:

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

[2] *Buldas, Ahto, Andres Kroonmaa, and Risto Laanoja. "Keyless Signatures' Infrastructure: How to Build Global Distributed Hash-Trees - Springer." Keyless Signatures' Infrastructure: How to Build Global Distributed Hash-Trees - Springer. Springer Link, 18 Oct. 2013. Web. 03 Nov. 2014.*

[3] *Klinger, Evan, and David Starkweather. "PHash." .org: Home of , the Open Source Perceptual Hash Library. N.p., n.d. Web. 02 Nov. 2014.*

[4] *Perceptual Hashing by Joe Bertolami, May 28, 2014 from bertolami.com*

[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] *Similar images detection: Perceptual hashing algorithm -- implementation dHash by python, Erum, Aug 4, 2015 from Jianshu*

[8] *Three ways of similar images detection based on perceptual hashing algorithm, Together_CZ, Sep 26, 2014, from CSDN*

Code of this research is available on my github.

https://github.com/lishahan/INFO7390