[2015_KAGGLE CHALLENGE CASE II - CANCEL]

Diabetic Retinopathy Machine Learning

(장비구축->기본예제수행까지만 : 기록용 문서)

2015.6 ~ 7.4

MIN SANGSHIK

2015_kaggle_diabetic retinopathy 문제

You are provided with a large set of high-resolution retina images taken under a variety of imaging conditions. A left and right field is provided for every subject. Images are labeled with a subject id as well as either left or right (e.g. 1_left.jpeg is the left eye of patient id 1).
A clinician has rated the presence of diabetic retinopathy in each image on a scale of 0 to 4, according to the following scale:
0 - No DR
1 - Mild
2 - Moderate
3 - Severe
4 - Proliferative DR
Your task is to create an automated analysis system capable of assigning a score based on this scale.
The images in the dataset come from different models and types of cameras, which can affect the visual appearance of left vs. right. Some images are shown as one would see the retina anatomically (macula on the left, optic nerve on the right for the right eye). Others are shown as one would see through a microscope condensing lens (i.e. inverted, as one sees in a typical live eye exam). There are generally two ways to tell if an image is inverted:
It is inverted if the macula (the small dark central area) is slightly higher than the midline through the optic nerve. If the macula is lower than the midline of the optic nerve, it's not inverted.
If there is a notch on the side of the image (square, triangle, or circle) then it's not inverted. If there is no notch, it's inverted.
Like any real-world data set, you will encounter noise in both the images and labels. Images may contain artifacts, be out of focus, underexposed, or overexposed. A major aim of this competition is to develop robust algorithms that can function in the presence of noise and variation.

2015_kaggle_diabetic retinopathy 관련 제공데이터 (압축시 약 100G, 압축 해제시 500GB)

Train Data : File 수 : 35126, 용량 35.3G
Test Data : File 수 : 53576, 용량 53.7G

File Name	Available Formats
trainLabels.csv	.zip (69.40 kb)
sampleSubmission.csv	.zip (81.55 kb)
sample	.zip (10.40 mb)
train	train.zip.001 (7.81 gb) train.zip.002 (7.81 gb) train.zip.003 (7.81 gb) train.zip.004 (7.81 gb) train.zip.005 (1.34 gb)
test	test.zip.001 (7.81 gb) test.zip.002 (7.81 gb) test.zip.003 (7.81 gb) test.zip.004 (7.81 gb) test.zip.005 (7.81 gb) test.zip.006 (7.81 gb) test.zip.007 (2.75 gb)

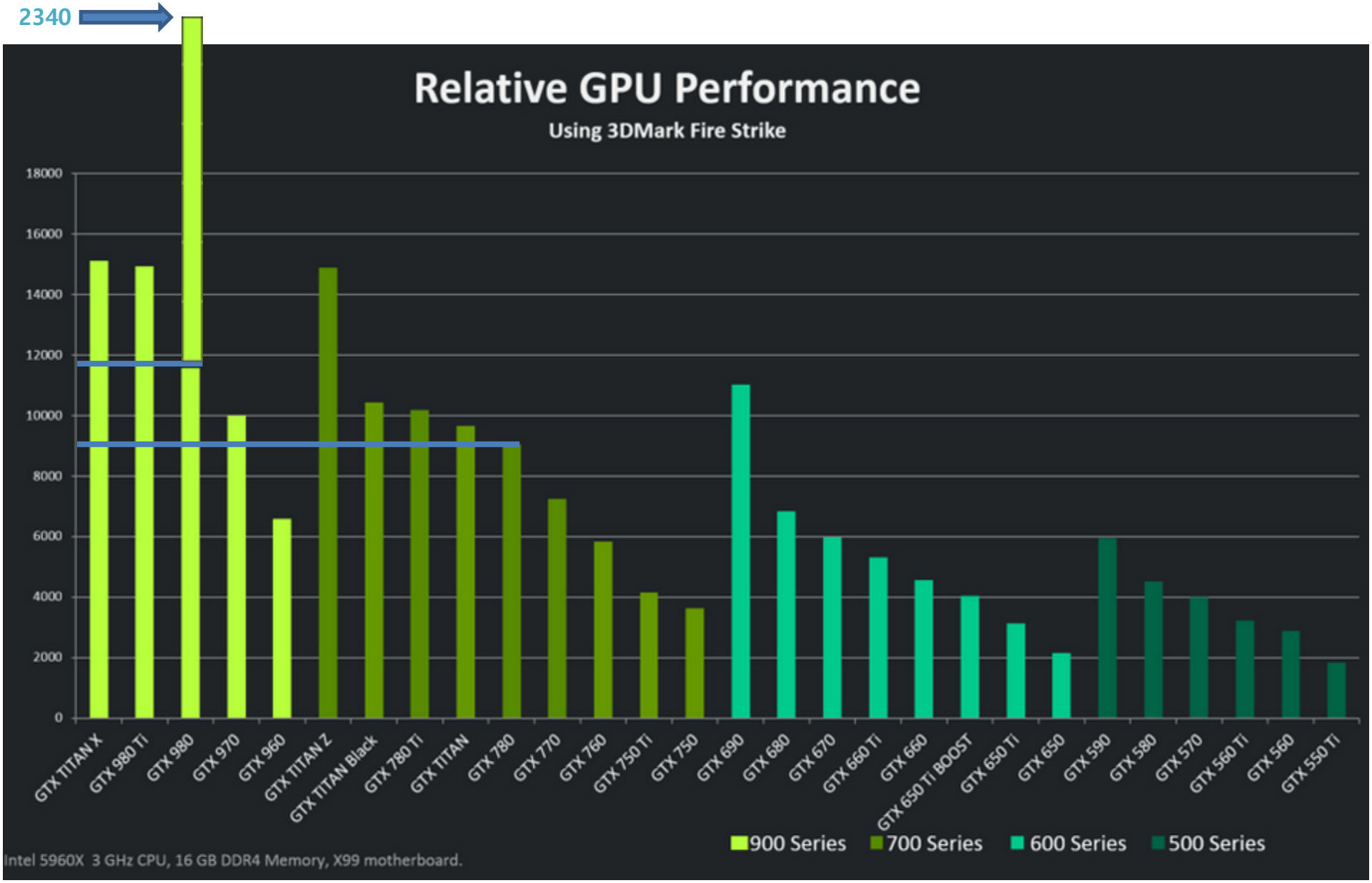
2015_kaggle_diabetic retinopathy 관련 H/W 준비

빅데이터 머신러닝을 위한 GPU 장비 구매 (3000$, 2015.6)

CPU	인텔 하스웰-E i7 5820K 오버클럭	3.3G -> 최대4.5G
메모리	삼성 DDR4 16GB 17000 쿼드채널	21300까지 오버클럭
메인보드	기가바이트 GA-X99-UD4 오버클럭	램뱅크8개 최대 64기가
그래픽카드	기가바이트 GTX980 UDV 윈드포스 메탈 4GB	무상3년
SSD	ADATA PremierPro SP900 256GB	읽기속도 550MB/s
DVD멀티	별매(옵션에서 추가가능)	SATA방식만가능
네트워크	인텔 기가비트랜(빠르고 안정적인 데이터전송/낮은 CPU 점유율)	
사운드	ALC1150 7.1채널 HIGH DEFINITION 오디오	
케이스	기본케이스(CX1100/ALPHA)	같은성능 다른디자인
파워서플라이	스카이디지털 PS3-700KO 액티브PFC	무상 3년 AS

2015_kaggle_diabetic retinopathy 관련 H/W 준비

GPU 성능 : 2340 (1170*2, GTX980*2, 향후 4개까지 확장가능)



2015_kaggle_diabetic retinopathy 관련 S/W 준비

빅데이터 머신러닝을 위한 S/W 셋팅

없음
(포팅 필요)

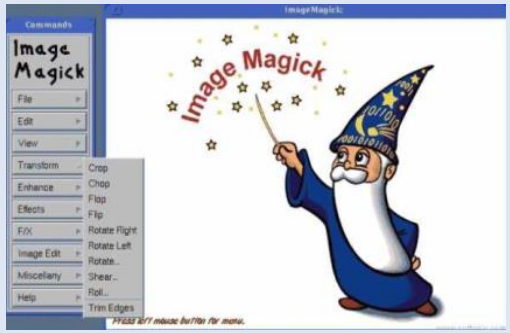
python PyDEV
eclipse
PyCUDA

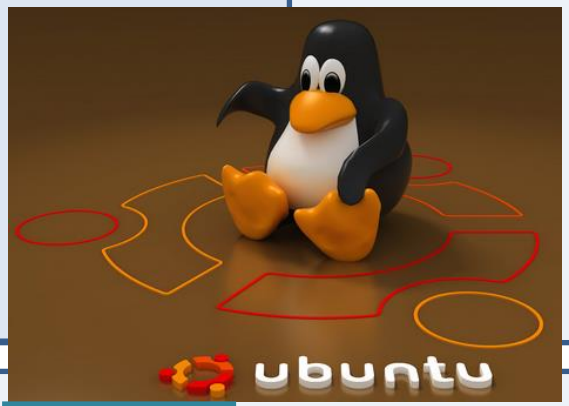
GraphLab
GPU Support
ML Platform

없음
(코딩 필요)

Parallel

이미지 프로세싱 프로그램





NVIDIA CUDA

NVIDIA CUDA

CUDA ("Compute Unified Device Architecture", 최초의 CUDA SDK는 2007년 2월 15일에 공개)

GPGPU(General-Purpose computing on Graphics Processing Units, **GPU 상의 범용 계산**)

2015_kaggle_diabetic retinopathy 기존 연구자 Approach 1

https://github.com/hoytak/diabetic-retinopathy-code

[1단계 : 기본 이미지 전처리]
- 툴(ImageMagick, http://www.imagemagick.org/)을 이용하여 빈공간 제거,
- 256*256 ??? 으로 사이즈 변환?
- 이미지 센터로 이동
  (NN 성능을 올리기 위해 이미지 스케일 변환은 안했다...)
- 이미지를 여러가지로 변환해서 데이터 추가 => Hue, Contrast, White Balance

[2단계 : ML 이용 레벨 구분함]
- 여러가지로 훈련시킴, 각 레벨 또는 0 vs 1-4, 0-1 vs. 2-4, 0-2 vs. 3-4, 0-3 vs. 4 등
- 각 Feature 를 Boosted Regression Tree 이용하여 Level 구분함

Main Ideas

Use ImageMagick's convert tool to trim off the blank space to the sides of the images, then pad them so that they are all 256x256. Thus the eye is always centered with edges against the edges of the image. I avoided scaling the images to improve the neural net performance.
Create multiple versions of each image varying by hue and contrast and white balance.
Duplicate each class so each class is represented equally, then shuffle the data.
Train several neural nets, one trained to predict the level membership, and another 4 to distinguish 0 vs 1-4, 0-1 vs. 2-4, 0-2 vs. 3-4, and 0-3 vs. 4.
For each image, extract both class predictions and the values of the final neural net layer. Pool all of these features across models and variations of the same images.
Train boosted regression trees on these pooled features to predict the level.
Round to the nearest integer as the class prediction.
Each of these steps could definitely be tuned for greater accuracy, and I am welcome to feedback.

1. Install the ImageMagick convert command line tool (named imagemagick) and the GNU parallel tool (named parallel). These are in all the major linux repositories.

1-2. Install the GPU version of Graphlab

Create with:

```
sudo pip install http://static.dato.com/files/graphlab-create-1.3.gpu.tar.gz
```

See https://dato.com/products/create/gpu_install.html and https://dato.com/products/create/quick-start-guide.html for more info.

2. Run prep_image.sh on each image to prepare the image variations and resized images. The instructions are at the top of the prep_image.sh file. Essentially, in the directory diabetic/, run::

```
ls train/*.jpeg test/*.jpeg | parallel ./prep_image.sh
```

This will preprocess all the images into diabetic/processed/train/ and diabetic/processed/test/.

3. Run create_image_sframes.py in diabetic/ to build the primary image SFrames. This will produce directories diabetic/image-sframes/train/ and diabetic/image-sframes/test/.

4. Run create_balanced_classes.py in diabetic/ to build a collection of balanced classes for training each of the neural nets above. This may take a while, and it produces roughly 150GB of data in the end. If an error occurs, the script can be restarted.

5. Set the which_model variable at the top of create_nn_model.py; this is one of [0,1,2,3,4]. Run this file on a machine with cuda installed to run the NN training code.
I get about 200 images / second on amazon's GPU instance type,
and about 320 images / second on a GTX 780.

It takes about a day to train one of the models, but once trained, it can be loaded and used for any future predictions. Models are saved to diabetic/nn_256x256/models/.

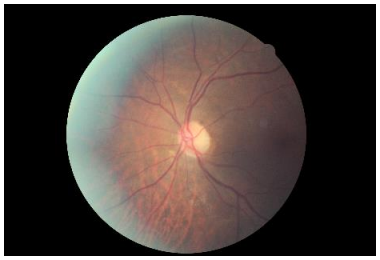
6. Once the model results are saved, run create_submission.py to build the final regression model and submission.

(1/5) prep_image.sh

[1단계 : 기본 이미지 전처리 1]

- 툴(ImageMagick, http://www.imagemagick.org/)을 이용하여 빈공간 제거, 센터맞춤, 사이즈256*256 으로 조정

```
size=256x256
convert -fuzz 10% -trim +repage -resize $size -gravity center -background black -extent $size -
equalize $1 $out
=>
[실제 실행]
convert -fuzz 10% -trim +repage -resize 256x256 -gravity center -background black -extent
256x256 -equalize 10_left.jpeg 10_left_step1.jpeg
```



4752*3168 => 256*256

```
convert -fuzz 10% -trim +repage -resize 256x256 -gravity center -background black -extent
256x256 -equalize 15_left.jpeg 15_left_step1.jpeg
```

3888*2592 => 256*256

[1단계 : 기본 이미지 전처리 2 processed/run-stretch]

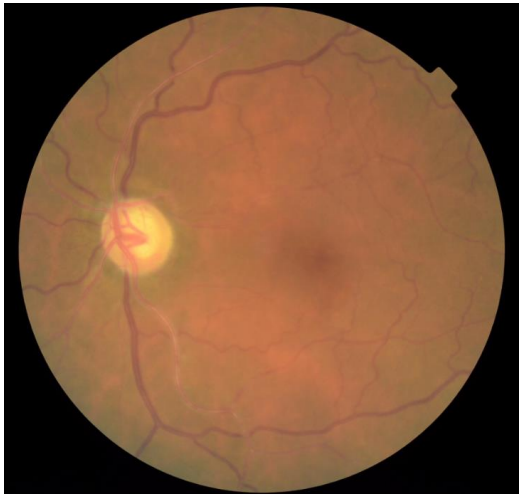
- 툴(ImageMagick, http://www.imagemagick.org/)을 이용

convert -fuzz 10% -trim +repage -transparent black -contrast-stretch 2x2% -resize $size -gravity center -background black -extent $size –equalize $1 $out

=>

convert -fuzz 10% -trim +repage -transparent black -contrast-stretch 2x2% -resize 256X256 -gravity center -background black -extent 256x256 –equalize 10_left.jpeg 10_left_step2.jpeg

convert -fuzz 10% -trim +repage -transparent black -contrast-stretch 2x2% -resize 256X256 -gravity center -background black -extent 256x256 –equalize 15_left.jpeg 15_left_step2.jpeg



[1단계 : 기본 이미지 전처리 3 processed/run-stretch]

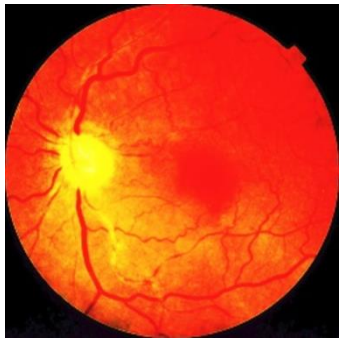
- 툴(ImageMagick, http://www.imagemagick.org/)을 이용

convert -fuzz 10% -trim +repage -transparent black -contrast-stretch 2x2% -resize $size -gravity center -background black -extent $size –equalize $1 $out

=>

convert -fuzz 10% -trim +repage -transparent black -contrast-stretch 2x2% -resize 256X256 -gravity center -background black -extent 256x256 –equalize 10_left.jpeg 10_left_step2.jpeg

convert -fuzz 10% -trim +repage -transparent black -contrast-stretch 2x2% -resize 256X256 -gravity center -background black -extent 256x256 –equalize 15_left.jpeg 15_left_step2.jpeg

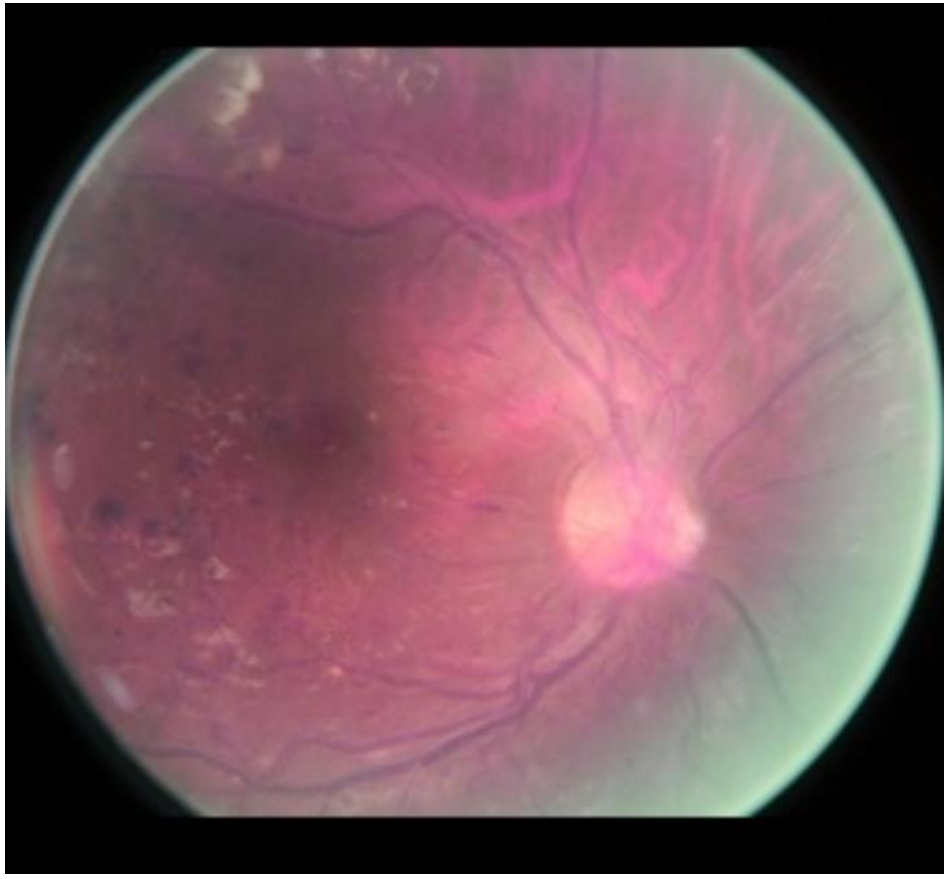



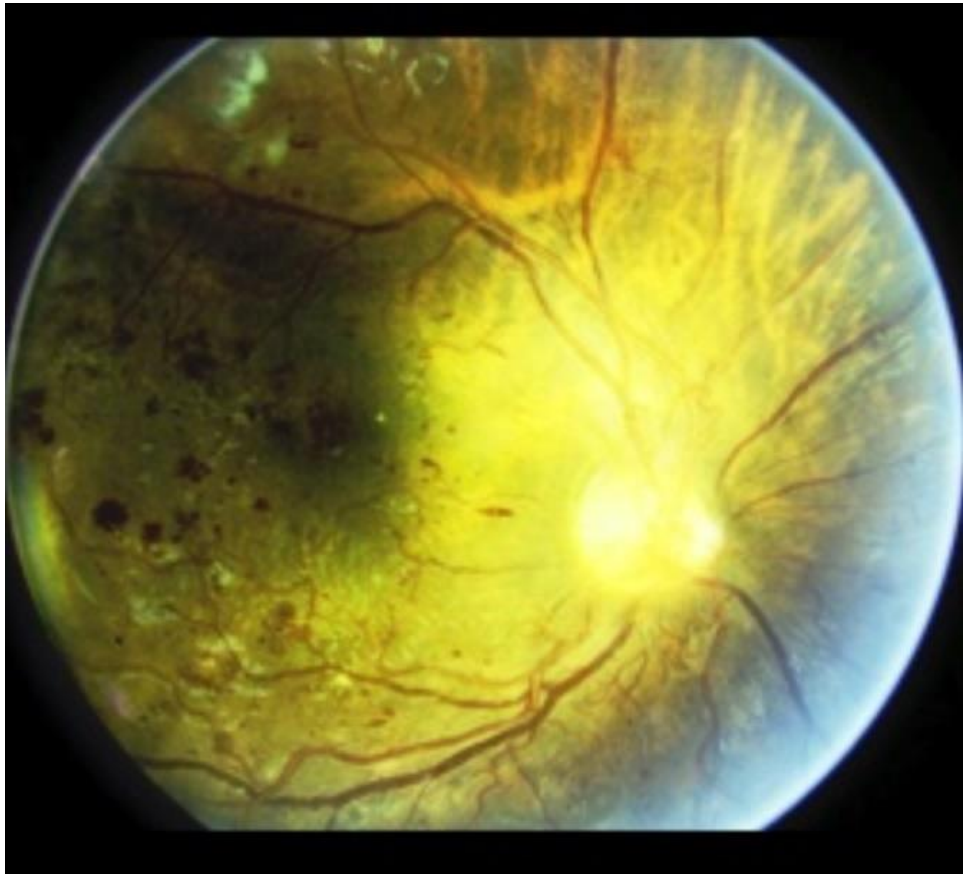


[1단계 : 기본 이미지 전처리 3 processed/run-hue-1, 2]

convert -fuzz 10% -trim +repage -modulate 100,100,80 -resize 256x256 -gravity center -background black -extent 256x256 –equalize 16_left.jpeg 16_left_step3_hue1.jpeg

convert -fuzz 10% -trim +repage -modulate 100,100,120 -resize 256x256 -gravity center -background black -extent 256x256 -equalize 16_left.jpeg 16_left_step3_hue2.jpeg



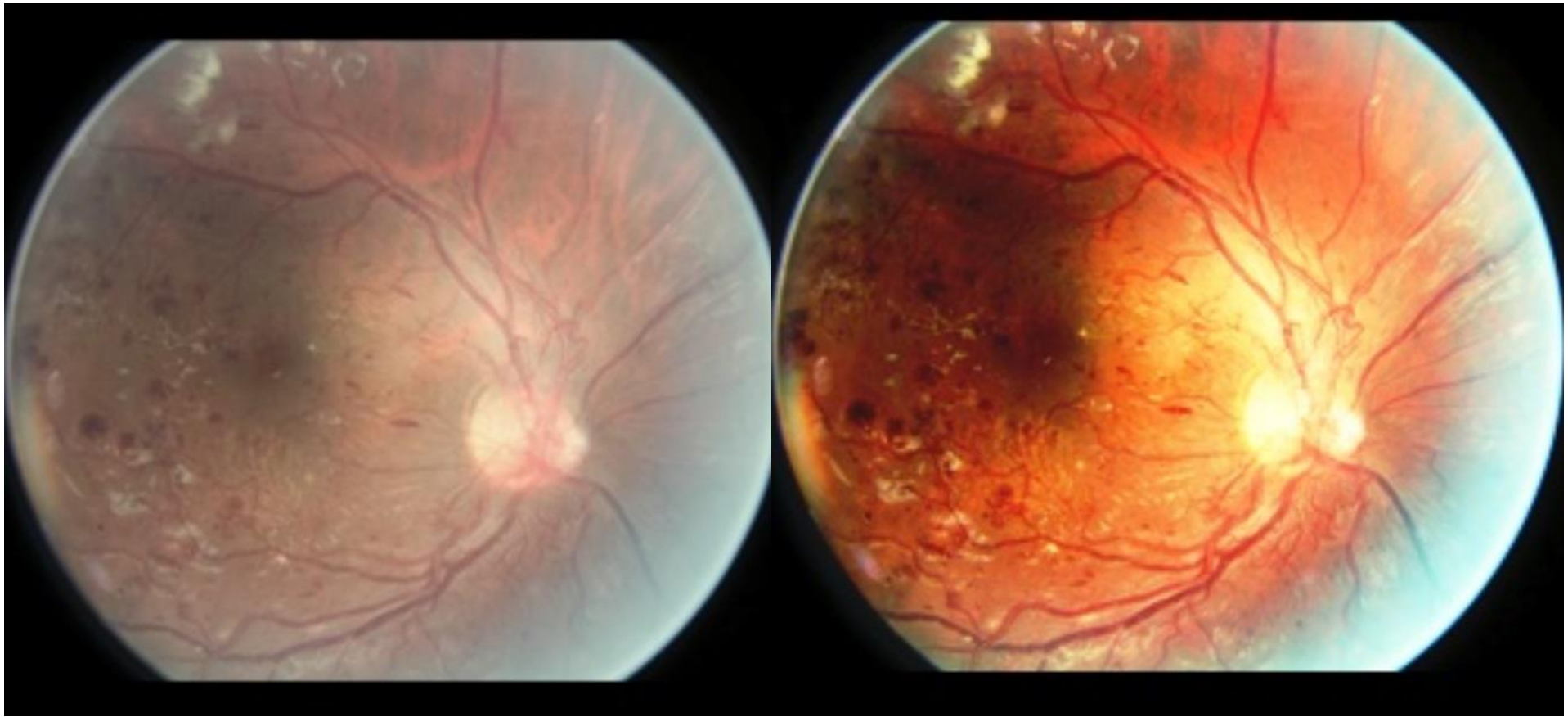


[1단계 : 기본 이미지 전처리 4 processed/run-sat-1, 2]

- 툴(ImageMagick, http://www.imagemagick.org/)을 이용

convert -fuzz 10% -trim +repage -modulate 100,80,100 -resize 256x256 -gravity center -background black -extent 256x256 –equalize 16_left.jpeg 16_left_step3_sat1.jpeg

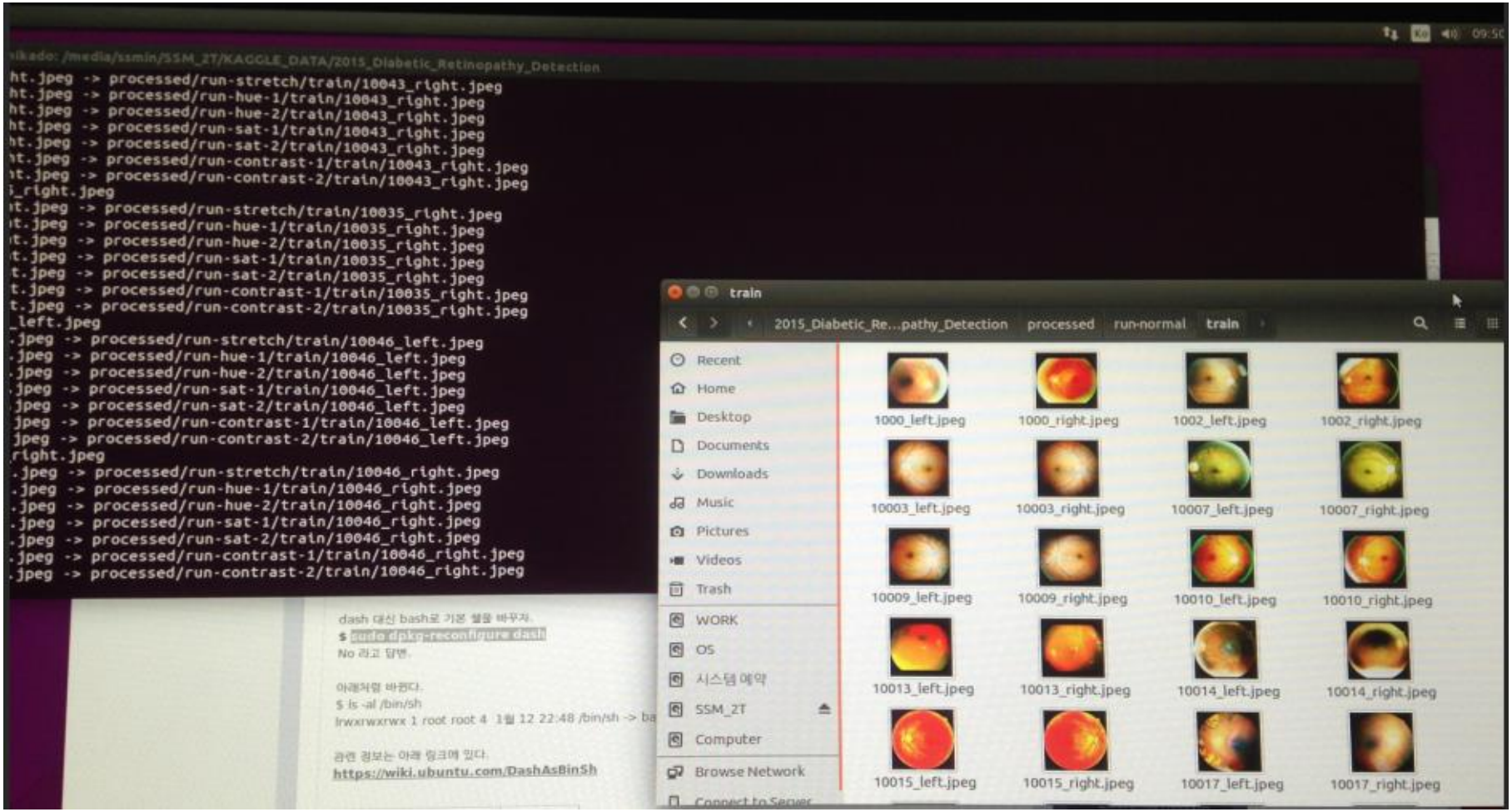
convert -fuzz 10% -trim +repage -modulate 100,120,100 -resize 256x256 -gravity center -background black -extent 256x256 -equalize 16_left.jpeg 16_left_step3_sat2.jpeg



[1단계 : 기본 이미지 전처리 실행]

- 툴(ImageMagick, http://www.imagemagick.org/)을 이용
- 빠른 이미지 전처리를 위해서 Windows에서는 Python등을 이용하여 멀티스레드 또는 멀티프로세스 프로그래밍을 하던지, linux에서 기본적으로 제공되는 GNU Parallel (멀티프로세스-CPU 코어 활용-지원) 명령어를 이용

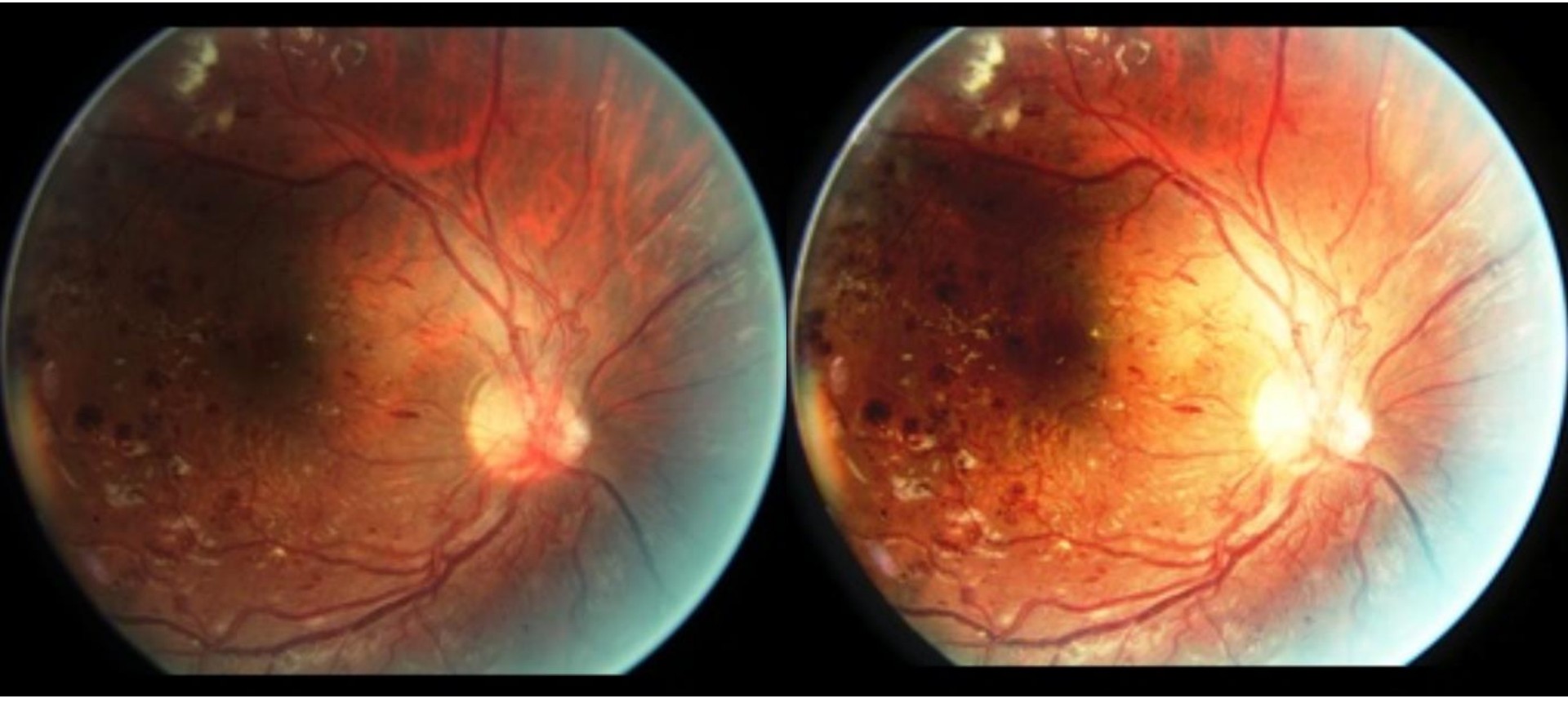
=> Serial한 변환보다 기본적으로 (4~6배, 아래 시스템에서는 6 Core i7 intel Processor이용)
But 9만개, 100G정도의 이미지파일을 처리하는데, 약 6~8시간 예상



[1단계 : 기본 이미지 전처리 5 processed/run-contrast-1, 2]
- 툴(ImageMagick, http://www.imagemagick.org/)을 이용

convert -fuzz 10% -trim +repage -sigmoidal-contrast 5,75% -resize 256x256 -gravity center -background black -extent 256x256 –equalize 16_left.jpeg 16_left_step3_cont1.jpeg

convert -fuzz 10% -trim +repage -sigmoidal-contrast 10,50% -resize 256x256 -gravity center -background black -extent 256x256 -equalize 16_left.jpeg 16_left_step3_cont2.jpeg



(2/5) create_image_sframes.py

GraphLab-Create 1.4.1

GraphLab Create enables developers and data scientists to apply machine learning to build state of the art data products.

```
import graphlab as gl
import re
import random
from copy import copy
import os

# Run this script in the same directory as the train/ test/ and
# processed/ directories -- where you ran the prep_image.sh. It will
# put a image-sframes/ directory with train and test SFrames in the
# save_path location below.

save_path = "./"

# gl.set_runtime_config("GRAPHLAB_CACHE_FILE_LOCATIONS", os.path.expanduser("~/data/tmp/"))
# shuffle the training images
X = gl.image_analysis.load_images("processed/")
X["is_train"] = X["path"].apply(lambda p: "train" in p)

# Add in all the relevant information in places
source_f = lambda p: re.search("run-(?P<source>[^/]+)", p).group("source")
X["source"] = X["path"].apply(source_f)
extract_name = lambda p: re.search("[0-9]+_(right|left)", p).group(0)
X["name"] = X["path"].apply(extract_name)
X_train = X[X["is_train"] == True]
X_test = X[X["is_train"] != True]
```

```
# Add in the training labels
labels_sf = gl.SFrame.read_csv("trainLabels.csv")
label_d = dict( (d["image"], d["level"]) for d in labels_sf)
X_train["level"] = X_train["name"].apply(lambda p: label_d[p])

# Get roughly equal class representation by duplicating the different levels.
X_train_levels = [X_train[X_train["level"] == lvl] for lvl in [1,2,3,4] ]
n_dups = [int(round((1.0/5) / (float(xtl.num_rows()) / X_train.num_rows()) )) for xtl in X_train_levels]
for nd, xtl_src in zip(n_dups, X_train_levels):
for i in range(nd):
X_train = X_train.append(xtl_src)

# Do a poor man's random shuffle
X_train["_random_"] = random.sample(xrange(X_train.num_rows()), X_train.num_rows())
X_train = X_train.sort("_random_")
del X_train["_random_"]

# Save sframes to a bucket
X_train.save(save_path + "image-sframes/train")
X_test.save(save_path + "image-sframes/test")
```

(3/5) create_balanced_classes.py

```
import graphlab as gl
import re
import random
from copy import copy
import os
import graphlab.aggregate as agg
import array
import numpy as np
import sys

# Run this script in the same directory as the
train_path = "image-sframes/train-%d/"
valid_path = "image-sframes/validation-%d/"
X_data = gl.SFrame("image-sframes/train/")
def save_as_train_and_test(X, train_loc, valid_loc):

# Can't just randomly sample the indices
all_names = list(X["name"].unique())
n_valid = (2 * len(all_names)) / 100
random.shuffle(all_names)
tr_names = gl.SArray(all_names[n_valid:])
valid_names = gl.SArray(all_names[:n_valid])
X_train = X.filter_by(tr_names, 'name')
X_valid = X.filter_by(valid_names, 'name')
X_train.save(train_loc)
X_valid.save(valid_loc)
```

```
# The classes were already balanced by create_image_sframe, so we
# don't need to balance them below.
if not (os.path.exists(train_path % 0) and os.path.exists(valid_path % 0)):
print "Skipping class 0; already present. If error, remove these directories and restart."
save_as_train_and_test(X_data, train_path % 0, valid_path % 0)


##################################################################
# Now do the other splitting parts
for mi in [1,2,3,4]:
if os.path.exists(train_path % mi) and os.path.exists(valid_path % mi):
print "Skipping class %d; already present. If error, remove these directories and restart." % mi
continue
print "Running class %d" % mi
X_data["class"] = (X_data["level"] >= mi)
X_data_local = copy(X_data)
n_class_0 = (X_data["class"] == 0).sum()
n_class_1 = (X_data["class"] == 1).sum()

if n_class_0 < n_class_1:
        num_to_sample = n_class_1 - n_class_0
```

```
            # Oversample the ones on the border
            level_to_sample = mi - 1
            class_to_sample = 0
else:
            num_to_sample = n_class_0 - n_class_1
            # Oversample the ones on the border
            level_to_sample = mi
            class_to_sample = 1

X_data_lvl = X_data[X_data["level"] == level_to_sample]

# Do one extra of the closest class to slightly oversample the hard examples.
n = min(X_data_lvl.num_rows(), num_to_sample)
X_data_local = X_data_local.append(X_data_lvl[:n])
num_to_sample -= n
if num_to_sample > 0:
X_data_class = X_data[X_data["class"] == class_to_sample]
while num_to_sample > 0:
n = min(X_data_class.num_rows(), num_to_sample)
X_data_local = X_data_local.append(X_data_class[:n])
num_to_sample -= n

# Sort the rows
X_data_local["_random_"] = np.random.uniform(size = X_data_local.num_rows())
X_data_local = X_data_local.sort("_random_")
del X_data_local["_random_"]
save_as_train_and_test(X_data_local, train_path % mi, valid_path % mi)
```

(4/5) create_nn_model.py

```
import graphlab as gl
import re
import random
from copy import copy
import os
import graphlab.aggregate as agg
import array
import sys

model_name = "full-inet-small"
which_model = 4

print "Running model %d, %s" % (which_model, model_name)
alt_path = os.path.expanduser("~/data/tmp/")
if os.path.exists(alt_path):
gl.set_runtime_config("GRAPHLAB_CACHE_FILE_LOCATIONS", alt_path)
model_path = "nn_256x256/models/"
X_train = gl.SFrame("image-sframes/train-%d/" % which_model)
X_valid = gl.SFrame("image-sframes/validation-%d/" % which_model)
X_test = gl.SFrame("image-sframes/test/")
...
...
...
...
```

(5/5) create_submission.py

```
import graphlab as gl
import re
import random
from copy import copy
import os
import graphlab.aggregate as agg
import array
import sys

base_path = os.getcwd()
model_path = base_path + "/nn_256x256/models/"
train_sf = []
test_sf = []
feature_names = []
for n in [0,1,2,3,4]:
try:
Xf_train = gl.SFrame(model_path + "/scores_train_%d" % n)
Xf_test = gl.SFrame(model_path + "/scores_test_%d" % n)
train_sf.append(Xf_train)
test_sf.append(Xf_test)
                                                    ...
                                                    ...
                                                    ...
with open('submission.csv', 'wb') as outfile:
fieldnames = ['image', 'level']
writer = csv.DictWriter(outfile, fieldnames=fieldnames)
writer.writeheader()
for d in X_out[['image', 'level']]:
writer.writerow(d)
```

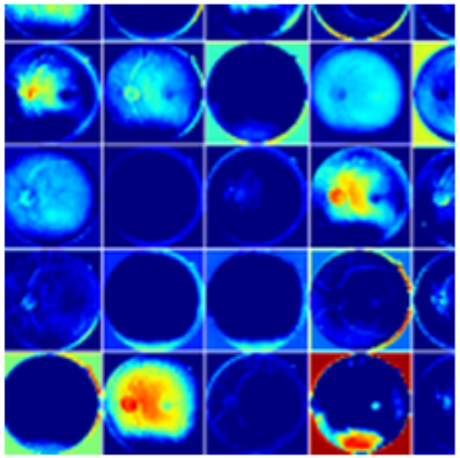
2015_kaggle_diabetic retinopathy 기존 연구자 Approach 2

http://www.joyofdata.de/blog/gpu-powered-deeplearning-with-nvidia-digits/

GPU Powered DeepLearning with NVIDIA DIGITS on EC2

Posted on 2015/04/25

In this tutorial I am going to show you how to set up CUDA 7, cuDNN, caffe and DIGITS on a g2.2xlarge EC2 instance (running Ubuntu 14.04 64 bit) and how to get started with DIGITS. For illustrating DIGITS' application I use a current Kaggle competition about detecting diabetic retinopathy and its state from fluorescein angiography.



[기타] GraphLab info - https://dato.com/download/install-gpu.html

Dato
Create Intelligence™

PRODUCTS SOLUTIONS LEARN EVENTS COMPANY BLOG

GraphLab Create™ with GPU Acceleration

Introduction

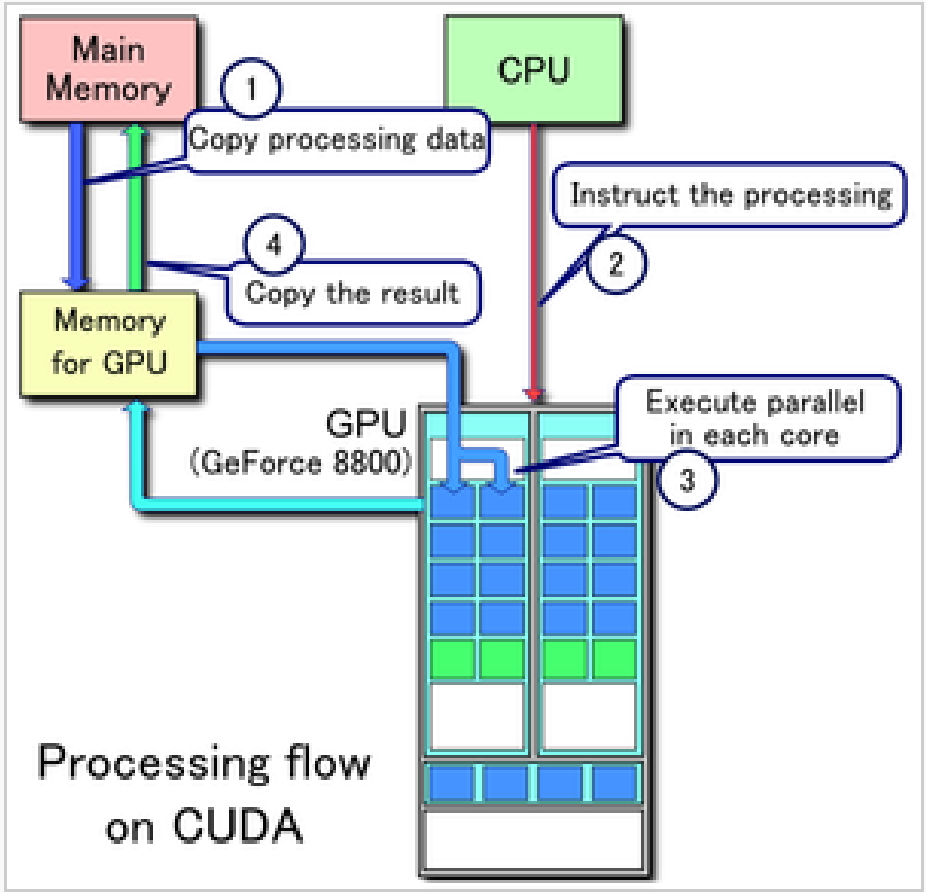
GraphLab Create™ with GPU Acceleration contains all of the capabilities of GraphLab Create™ with the additional support for NVIDIA CUDA GPUs (Graphics Processing Units). GPU acceleration significantly increases performance when using the Neural Network Classifier (boosted trees) toolkit.

During our benchmark, we clocked an ***18x speed improvement***. Using the Neural Network Classifier toolkit and a NVIDIA GTX-780 GPU on the MNIST image dataset we were able to process 28,150 images per second.

System Requirements:

- **Linux** : Any distribution with GLIBC >= 2.11
- A NVIDIA CUDA GPU: See NVIDIA CUDA Getting Started Guide for Linux for more details.
- NVIDIA driver version >=340

[기타] CUDA - https://ko.wikipedia.org/wiki/CUDA



Example of CUDA processing flow

1. Copy data from main mem to GPU mem
2. CPU instructs the process to GPU
3. GPU execute parallel in each core
4. Copy the result from GPU mem to main mem

https://ko.wikipedia.org/wiki/CUDA

Scientific Tools and Libraries
General Tools

Very general scientific computing
- NumPy
- SciPy

Visualization
- Matplotlib
- VisIt
- MayaVi
- Chaco
- VTK

High Performance Computing

Parallel Computing
- PETSc
- PyMPI
- Pypar
- mpi4py

GPGPU Computing
- PyCUDA
- PyOpenCL

Scientific Tools and Libraries
Domain Specific Tools

AI
- pyem
- ffnet
- pymorph
- Monte
- hcluster

Biology
- Brian
- SloppyCell
- NIPY
- PySAT

Molecular & Atomic Modeling
- PyMOL
- Biskit
- GPAW

Geo sciences
- GIS Python
- PyClimate
- ClimPy
- CDAT

Electromagnetics
- PyFemax

Astronomy
- AstroLib
- PySolar

Dynamic Systems
- Simpy
- PyDSTool

Finite Elements
- SfePy

Scientific Tools and Libraries
Special Topics

Wrapping other languages
- weave (C/C++)
- f2py (Fortran)
- Cython
- Ctypes (C)
- SWIG (C/C++)
- RPy / RSPython (R)
- MatPy (Matlab)
- Jython (Java)
- IronPython (.NET)

http://www.slideshare.net/onyame/python-for-high-performance-and-scientific-computing-8490899