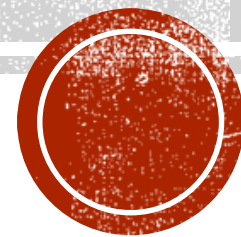


# **D**IGITAL **I**MAGE **P**ROCESSING

**-APPLIED MATHEMATICS CAPSTONE DESIGN-**



**D.I.P** 응용수학과  
김하현, 최우빈

# 목차

- 주제 ; cats and dogs binary classification
- (1) CNN(Convolutional Neural Network , 합성곱 신경망)을 이용한 직접 모델링 후, cats 와 dogs 의 각각 data set 을 training 한 후 accuracy 측정 - Python
- (2) 기존 data set의 image 들을 fourier transform의 한 종류인 DWT(Discrete Wavelet Transform)한 image 들로 training 한 후, accuracy 측정 - DWT : Matlab 으로 RGB 변환
- (3) 검증된 알고리즘을 통해, 기존 data set과 DWT data set 의 accuracy 측정 ( Alex net, Google net)
- -Matlab R2017b의 toolbox

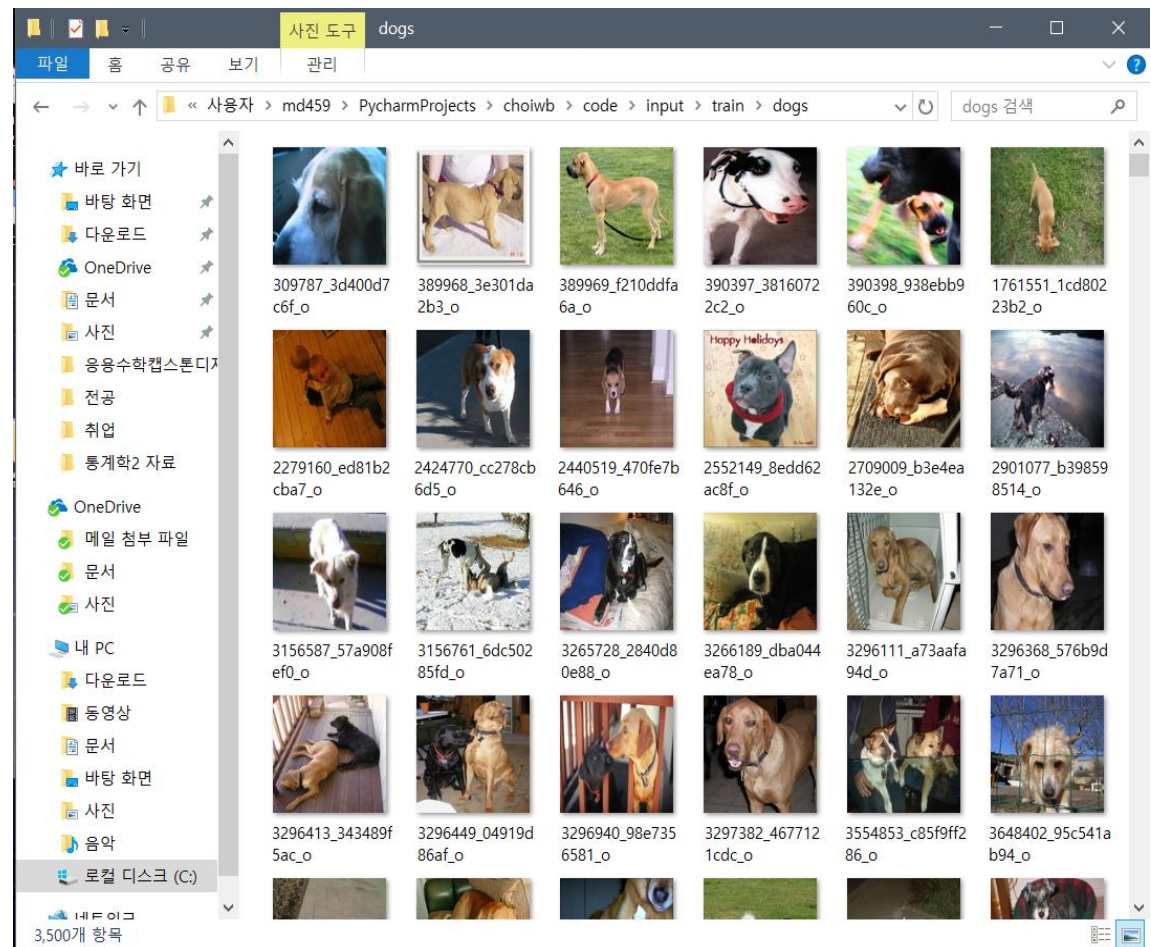
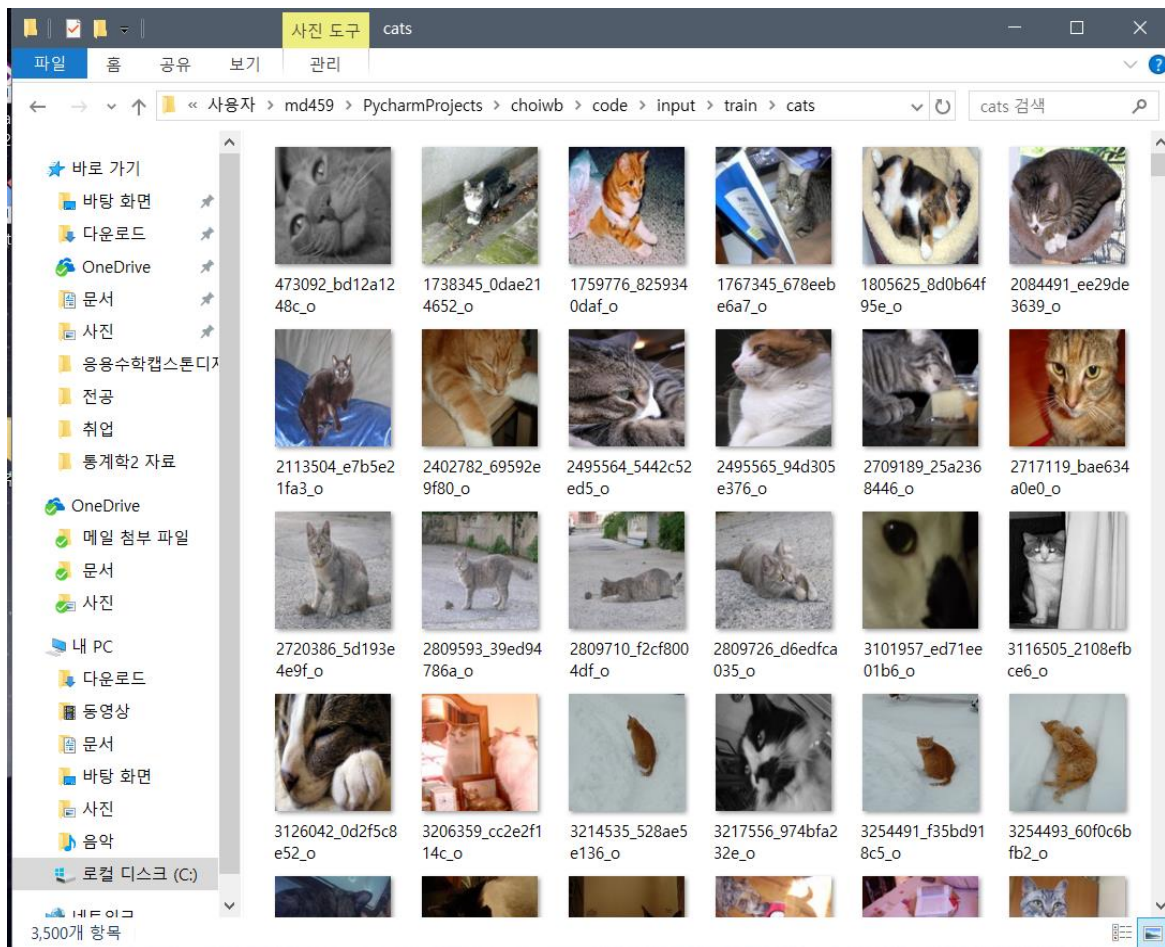


# CATS AND DOGS TRAINING , TEST , AUGMENTATION SET <FILE PATH>

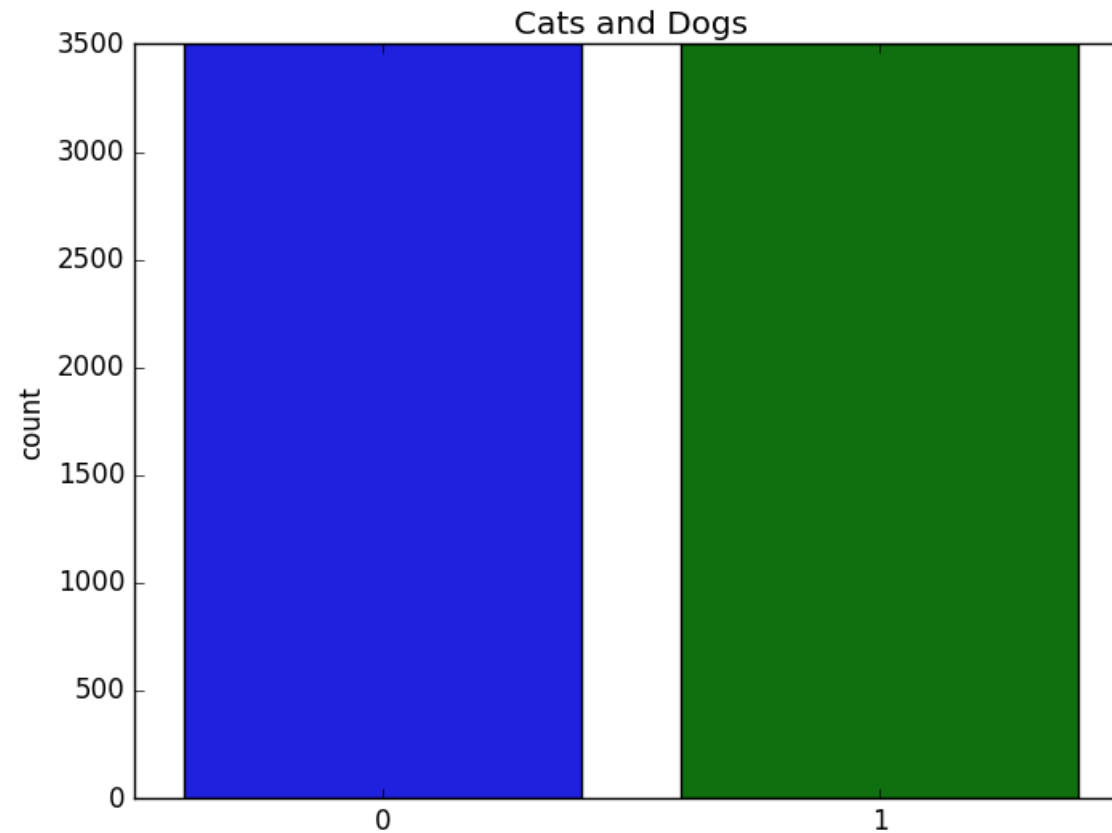
```
TRAIN_CATS_DIR = 'input/train/cats/' +  
TRAIN_DOGS_DIR = 'input/train/dogs/' +  
TEST_DIR = 'input/test/' +  
VALD_CATS_DIR = 'input/validation/cats/' +  
VALD_DOGS_DIR = 'input/validation/dogs/' +  
AUG_DIR = 'input/train/aug/' +
```

Image : 128 x 128 pixels





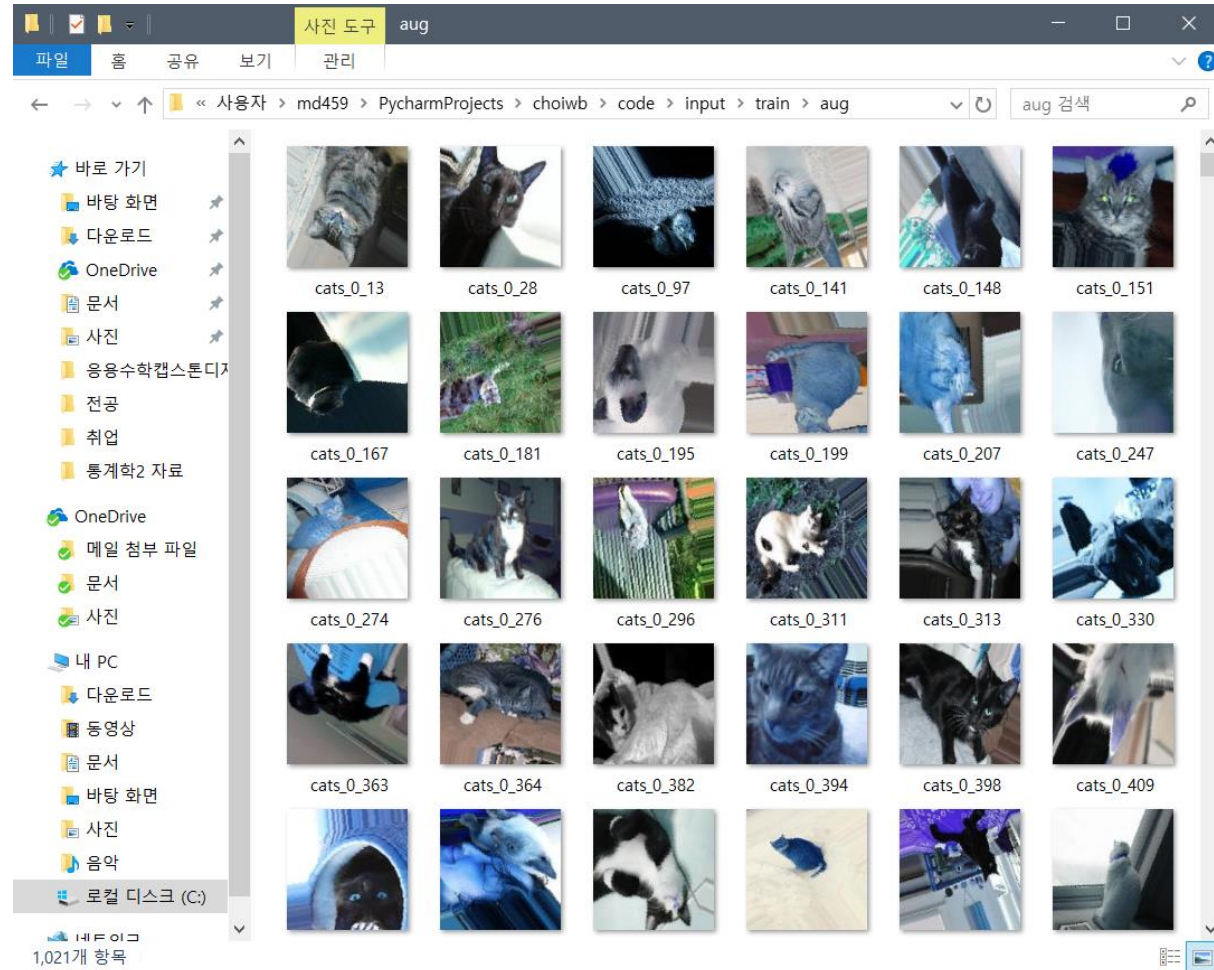
# TRAINING SET (CATS:: 3500 , DOGS : 3500),





# AUGMENTATION 과정

-  $\text{round}(\text{len}(\text{train}) / 50)$



**TRANSPOSE(대각화) : [COUNT , ROW , COLUMN , CHANNEL] -> [COUNT , CHANNEL , ROW , COLUMN]**

teosorflow : [count , row , column , channel] default(기본값) 시작 과정 (CPU)

row major order(행 우선 순위)



속도 때문 !

1	2	3
4	5	6
7	8	9

→

row-major								
1	2	3	4	5	6	7	8	9

NVIDIA cuDNN : [count , channel , row , column] default(기본값) 모델링 과정 (GPU)

- gray scale : 1 channel
- RGB scale : 3 channel



- Train set 형태 : (7000, 3, 128, 128)

augmentation 적용



- Train set 형태 : (7890, 3, 128, 128)

- Validation set 형태 : (1964, 3, 128, 128)

Validation\_split : 20%

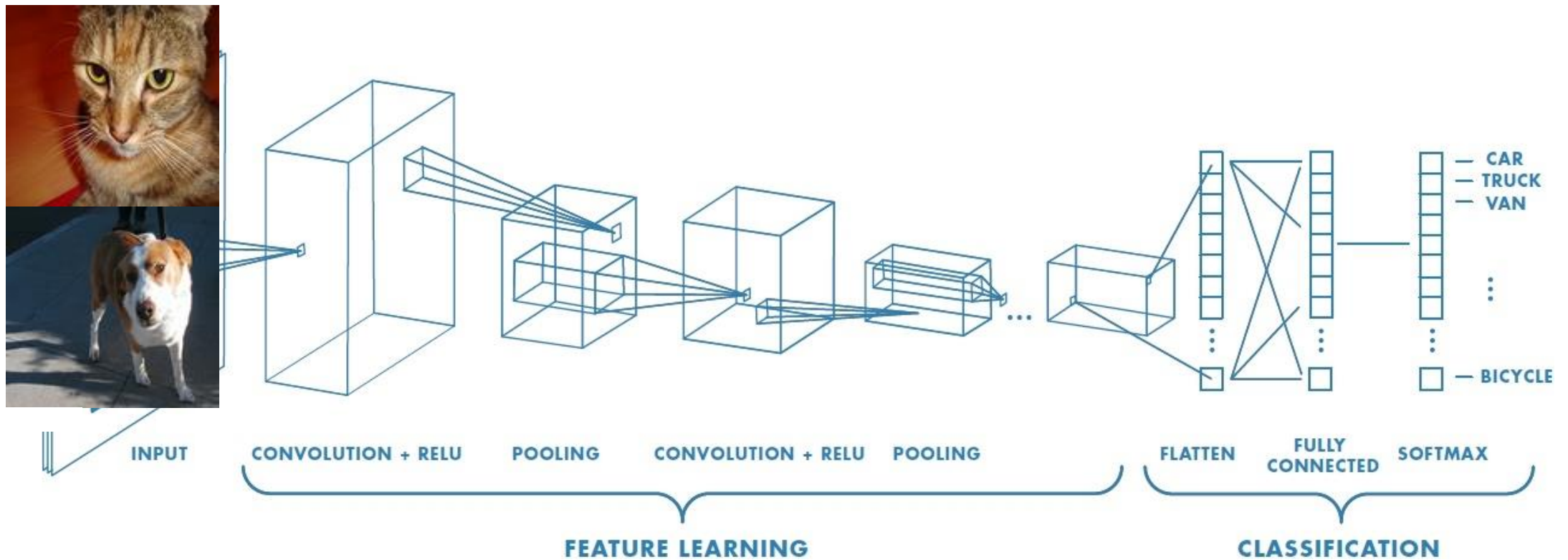


Train on 6312 samples, validate on 1578 samples





# CNN(CONVOLUTIONAL NEURAL NETWORK) 모델링 과정



- CNN 모델링 과정

Input layer(128 X 128 픽셀) : 16 neurons, 4 x 4 , ReLU , max\_norm = 2

Max pooling layer 3 x 3

Hidden layer : 32 neurons 4 x 4 , ReLU , max\_norm = 2

Max pooling layer 3 x 3

Hidden layer 64 neurons 4 x 4 , ReLU , max\_norm = 2

Max pooling layer 3 x 3

- Model flatten

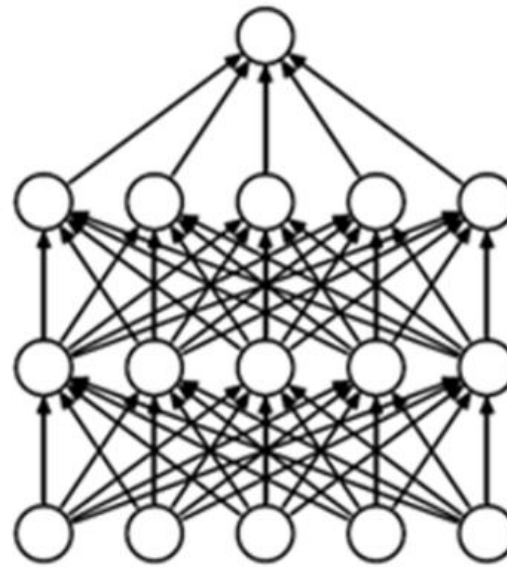
- Output Layer 128 neurons , Sigmoid , Dropout : 0.3



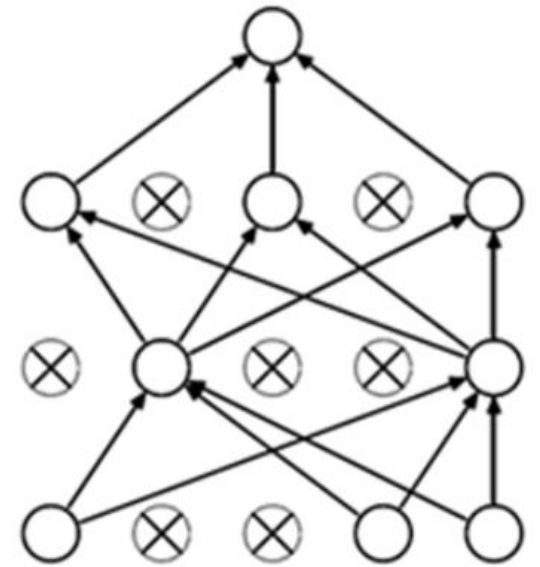
- **max\_norm** : 정규화(normalization)의 형태, 모든 뉴런에 대한 **weight vector**의 크기에 **upper bound**를 적용하고 **gradient descent**를 사용

- **flatten layer** : 2차원 행렬 형태를 1차원 벡터 형태로 변환

- **dropout** :  
**overfitting** 피하기 위함



(a) Standard Neural Net



(b) After applying dropout.



# 모델 연산과정

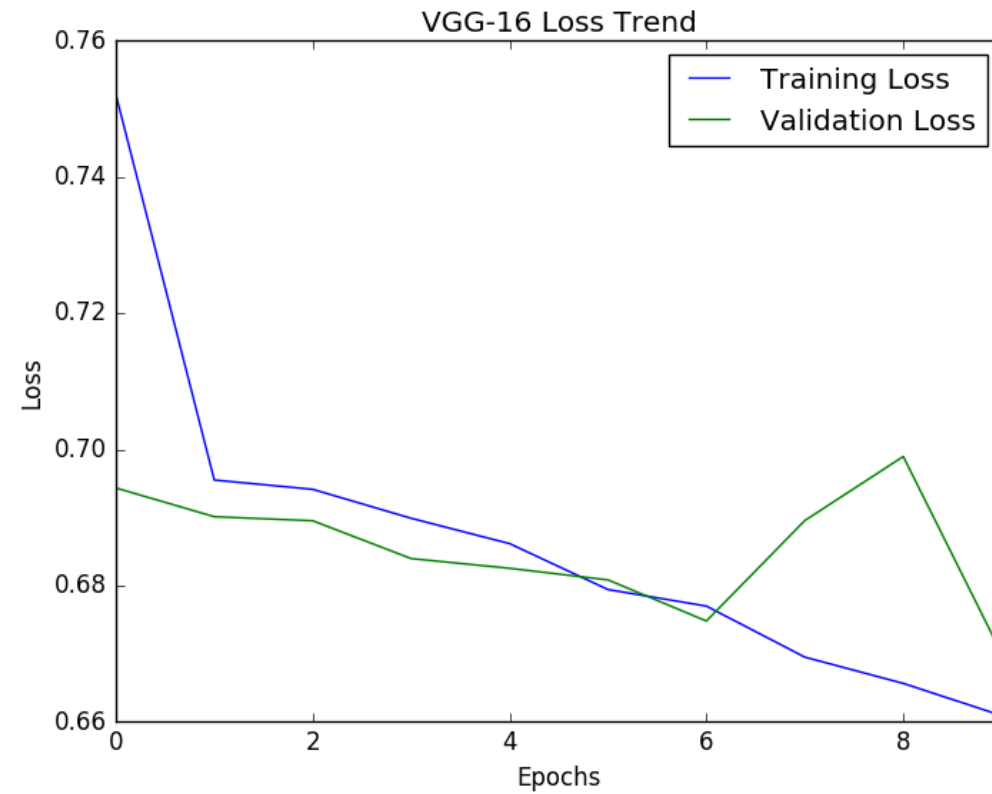
- Epoch 1/10
  - 80s - loss: 0.7521 - acc: 0.5021 - val\_loss: 0.6943 - val\_acc: 0.5006
- Epoch 2/10
  - 89s - loss: 0.6955 - acc: 0.5120 - val\_loss: 0.6901 - val\_acc: 0.5406
- Epoch 3/10
  - 84s - loss: 0.6941 - acc: 0.5119 - val\_loss: 0.6895 - val\_acc: 0.5393
- Epoch 4/10
  - 82s - loss: 0.6899 - acc: 0.5368 - val\_loss: 0.6840 - val\_acc: 0.5507
- Epoch 5/10
  - 74s - loss: 0.6862 - acc: 0.5433 - val\_loss: 0.6825 - val\_acc: 0.5634



- Epoch 6/10
- 80s - loss: 0.6794 - acc: 0.5700 - val\_loss: 0.6808 - val\_acc: 0.5653
- Epoch 7/10
- 74s - loss: 0.6770 - acc: 0.5762 - val\_loss: 0.6748 - val\_acc: 0.5659
- Epoch 8/10
- 78s - loss: 0.6695 - acc: 0.5848 - val\_loss: 0.6896 - val\_acc: 0.5406
- Epoch 9/10
- 74s - loss: 0.6656 - acc: 0.6008 - val\_loss: 0.6990 - val\_acc: 0.5640
- Epoch 10/10
- 79s - loss: 0.6609 - acc: 0.6006 - val\_loss: 0.6700 - val\_acc: 0.5875
- Baseline Accuracy: 60.08% , 연산시간: 987.7676200866699 , 약 16분 28초



# LOSS TREND






# <개선방향>

1. Train set과 augmentation set 데이터 증가

2. Googlenet 알고리즘 적용

3. Model parameter / hyper parameter 조정

\*Hyperparameter : Bayesian statistics에서 prior distribution의 parameter이다.  Posterior distribution만으로 추정을 하는 것보다 prior distribution을 알 때, 훨씬 더 잘 추정

\*Neural Network에서 hyperparameter

신경망 학습을 통해서 tuning 또는 optimization 해야 하는 주 변수가 아니라 사람들이 prior 설정을 하거나, 외부 모델 메커니즘을 통해 자동으로 설정이 되는 변수



Neural Network 에서 Hyperparameter 이란 ( = free parameter)

Ex) learning rate , cost function(cross entropy , least square) ,  
mini batch size , training 회수, hidden layer 개수,  
weights initialization



### -cross entropy

$p, m$  : 확률 분포

$$H(p, m) = -\sum_i p(x_i) \log(m(x_i))$$

기대 값과 예측 값의 차이가 클수록 결과가 크게 나온다.

결과는 항상 양수 이다.

### -cross entropy cost function

$y$  : 기대 값


$a$  : 신경망에서 출력된 값

$n$  : 훈련 데이터의 개수

$$C = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln(1 - a)]$$



# 중간결과에서 개선점

- Training set 증가 (7000  15724)
- Batch Normalization 적용
- neuron 개수 증가 (GPU 연산)

[input – hidden1 – hidden2 – output]

[16 – 32 – 64 – 128]  [64 – 128 – 256 – 512]



# BATCH NORMALIZATION

Gradient Vanishing / Gradient Exploding 이 일어나지 않도록 하는 아이디어 중의 하나. 지금까지는 이 문제를 Activation 함수의 변화 (ReLU 등), Careful Initialization, small learning rate 등으로 해결하였지만, 이런 간접적인 방법보다는 training 하는 과정 자체를 전체적으로 안정화 하여 학습 속도를 가속시킬 수 있는 근본적인 방법.

 **Layer Normalization !**

\*\*\* Internal Covariate Shift : Network의 각 층이나 Activation 마다 input의 distribution이 달라지는 현상.



각 층의 input의 distribution을 평균 0, 표준편차 1인 input으로 normalize 시키는 방법 고안.



# LAYER NORMALIZATION

- State of the art한 DNN 학습시간 단축.
- 중간 층의 출력을 정규화 함으로써 실현.

\*\*\*Covariate Shift(공 변량 변화)

: 입출력 규칙(주어진 입력에 대하여 출력의 생성규칙)은 **training** 할 때와, **test** 할 때 다르지 않지만, 입력(공 변량)의 분포가 **training** 할 때와, **test** 할 때 다른 상황.





# INTERNAL COVARIATE SHIFT

<CNN에서의 각 층마다 입력분포>

(1)입력 층 으로의 입력분포

평균 0 , 분산 1로 선형변환

(2)중간 층 으로의 입력분포

-정해진 입력분포를 유지하지 못함!

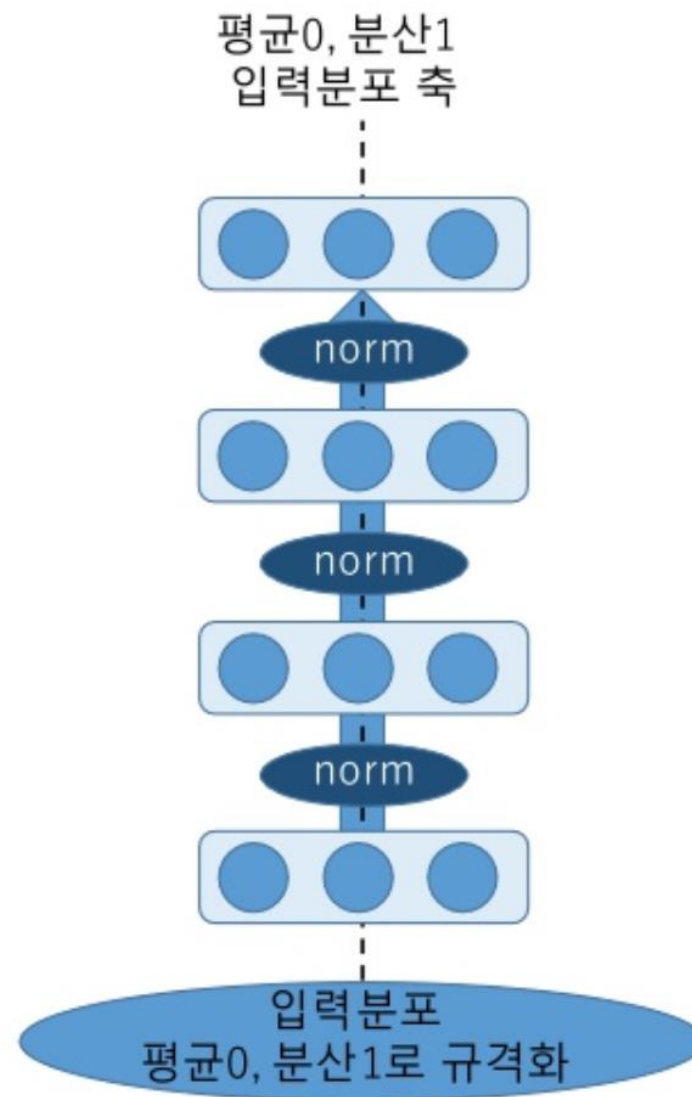
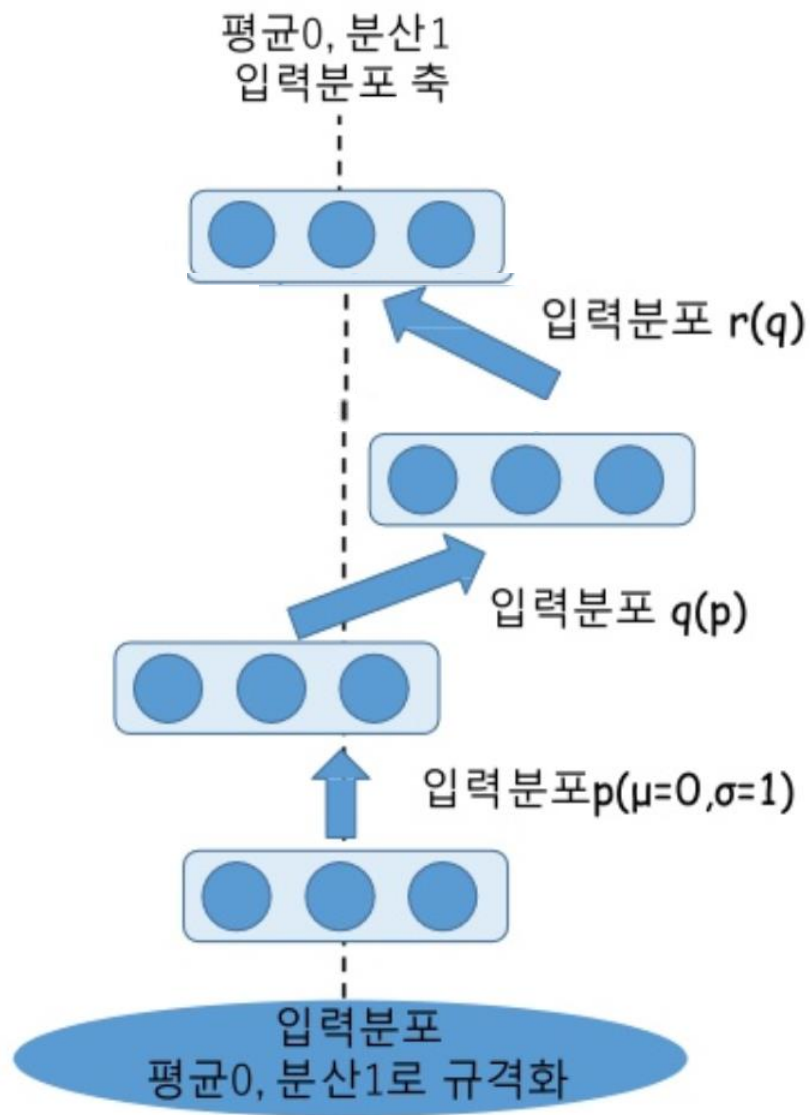
---



학습할 때 마다, 중간층으로의  
입력분포가 변화  
(Internal Covariate Shift)



# NORMALIZATION PROCESS



# TRAINING SET (CATS:; 7853 , DOGS : 7871),

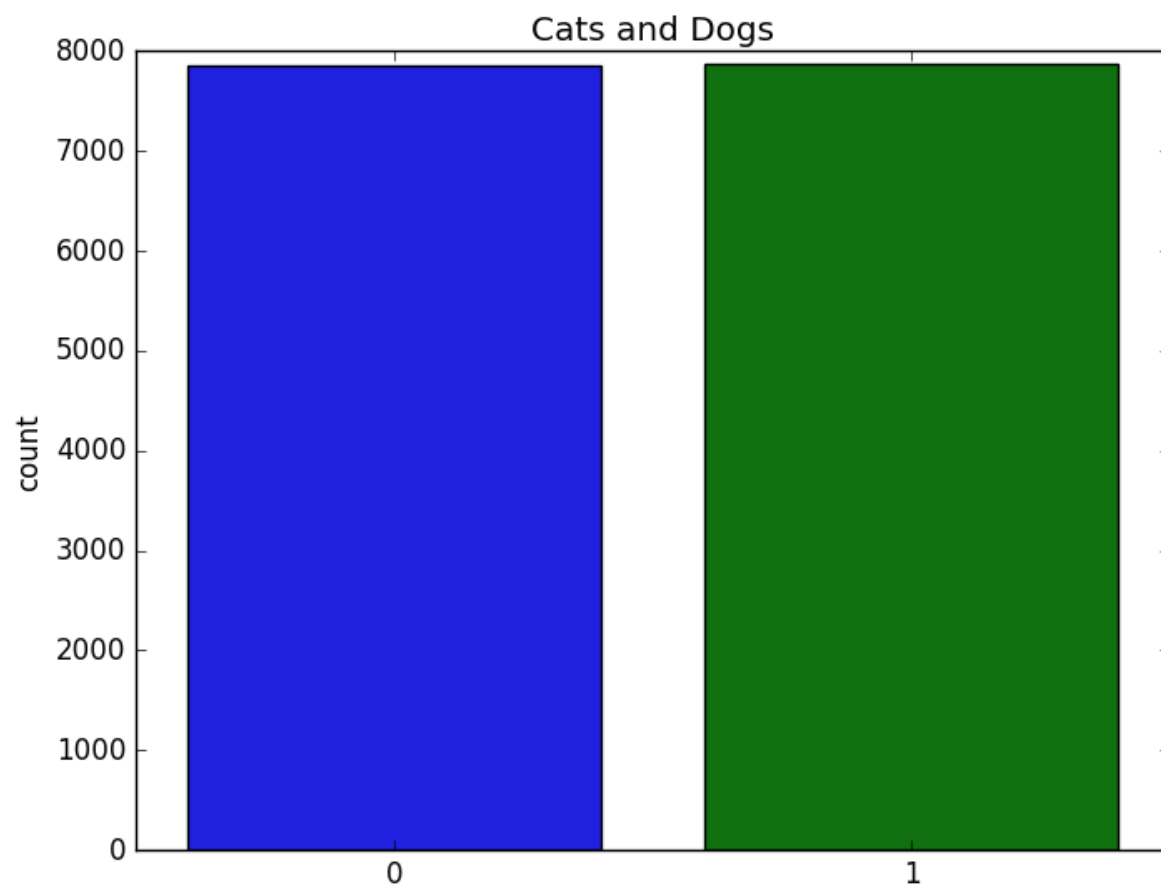


사진 도구

cats

파일 | 홈 | 공유 | 보기 | 관리

← → ↕ ↑

choiwooseok > PycharmProjects > choiwb > code > input > train > cats

cats 검색

바로 가기

바탕 화면

다운로드

문서

사진

code

input

음악

카카오톡 받은 파일

OneDrive

내 PC

다운로드

동영상

문서

바탕 화면

사진

음악

로컬 디스크 (C:)

네트워크

97417\_99fb243f  
ce\_o

473092\_bd12a1  
248c\_o

537488\_2b591b  
a078\_o

756031\_7a9fb1b  
928\_o

756035\_b426cb  
3bb6\_o

756120\_5af365c  
d92\_o

756156\_e684a2a  
0a4\_o

832424\_0a2fb39  
f1f\_o

993775\_d06d86  
da8e\_o

1738345\_0dae2  
14652\_o

1759776\_82593  
40daf\_o

1767345\_678ee  
be6a7\_o

1805625\_8d0b6  
4f95e\_o

2084491\_ee29de  
3639\_o

2113504\_e7b5e  
21fa3\_o

2402782\_69592  
e9f80\_o

2495564\_5442c  
52ed5\_o

2495565\_94d30  
5e376\_o

2709189\_25a23  
68446\_o

2717119\_bae63  
4a0e0\_o

2720386\_5d193  
e4e9f\_o

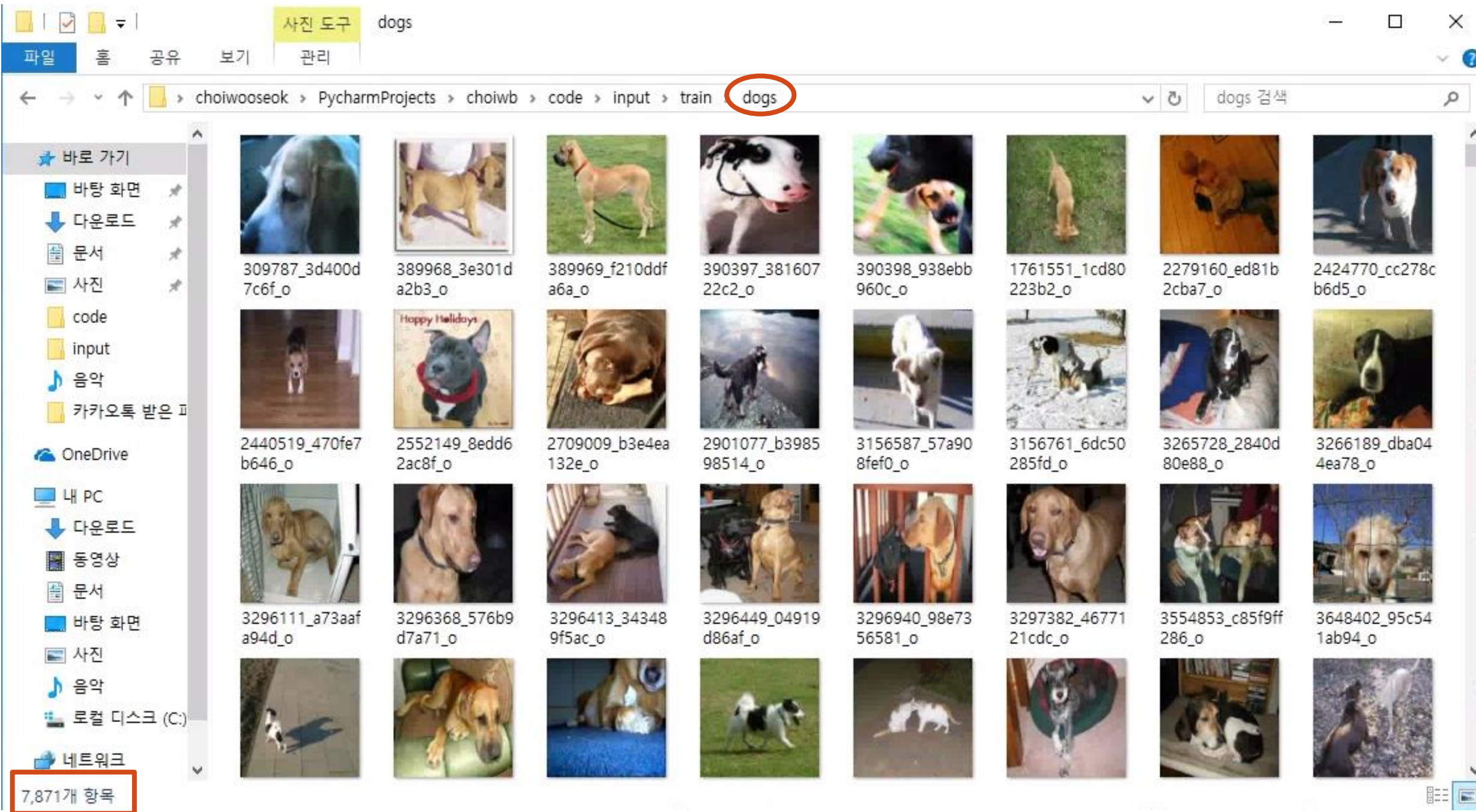
2809593\_39ed9  
4786a\_o

2809710\_f2cf80  
04df\_o

2809726\_d6edfc  
a035\_o

7,853개 항목





# MODELING

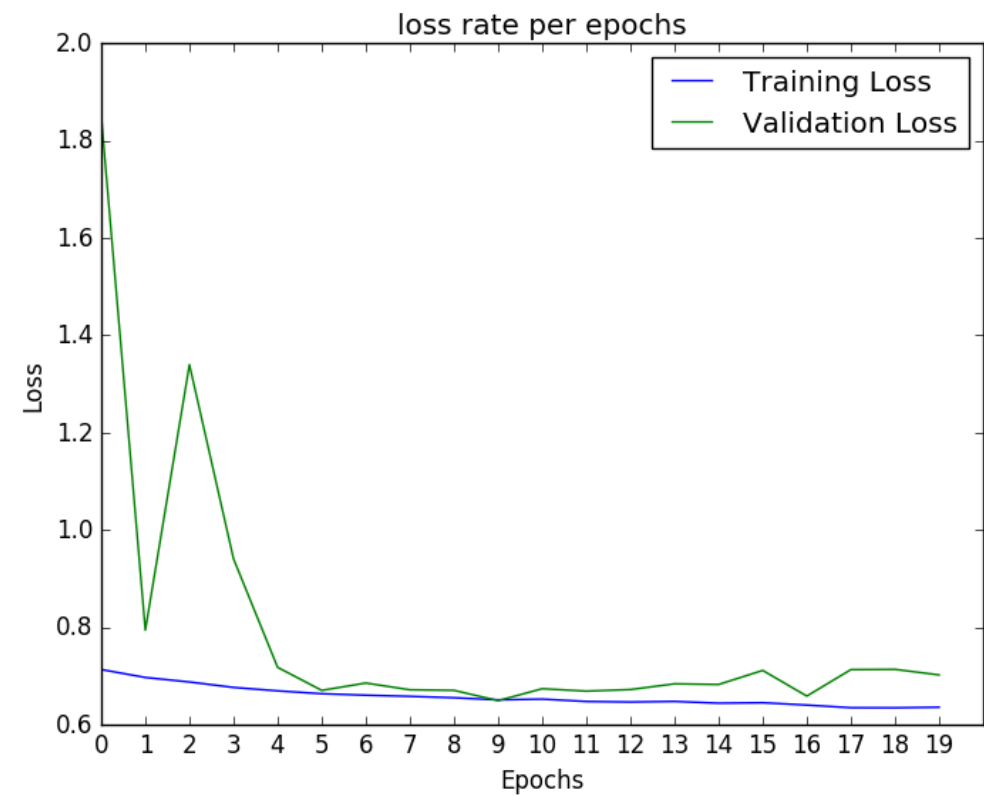
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 3, 128, 64)	131136
batch_normalization_1 (Batch Normalization)	(None, 3, 128, 64)	256
activation_1 (Activation)	(None, 3, 128, 64)	0
conv2d_2 (Conv2D)	(None, 3, 128, 64)	65600
batch_normalization_2 (Batch Normalization)	(None, 3, 128, 64)	256
activation_2 (Activation)	(None, 3, 128, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 3, 42, 21)	0
conv2d_3 (Conv2D)	(None, 3, 42, 128)	43136
batch_normalization_3 (Batch Normalization)	(None, 3, 42, 128)	512
activation_3 (Activation)	(None, 3, 42, 128)	0
conv2d_4 (Conv2D)	(None, 3, 42, 128)	262272
batch_normalization_4 (Batch Normalization)	(None, 3, 42, 128)	512
activation_4 (Activation)	(None, 3, 42, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 3, 14, 42)	0
conv2d_5 (Conv2D)	(None, 3, 14, 256)	172288
batch_normalization_5 (Batch Normalization)	(None, 3, 14, 256)	1024

activation_5 (Activation)	(None, 3, 14, 256)	0
conv2d_6 (Conv2D)	(None, 3, 14, 256)	1048832
batch_normalization_6 (Batch Normalization)	(None, 3, 14, 256)	1024
activation_6 (Activation)	(None, 3, 14, 256)	0
max_pooling2d_3 (MaxPooling2D)	(None, 3, 4, 85)	0
conv2d_7 (Conv2D)	(None, 3, 4, 512)	696832
batch_normalization_7 (Batch Normalization)	(None, 3, 4, 512)	2048
activation_7 (Activation)	(None, 3, 4, 512)	0
conv2d_8 (Conv2D)	(None, 3, 4, 512)	4194816
batch_normalization_8 (Batch Normalization)	(None, 3, 4, 512)	2048
activation_8 (Activation)	(None, 3, 4, 512)	0
max_pooling2d_4 (MaxPooling2D)	(None, 3, 1, 170)	0
flatten_1 (Flatten)	(None, 510)	0
dense_1 (Dense)	(None, 512)	261632
batch_normalization_9 (Batch Normalization)	(None, 512)	2048
activation_9 (Activation)	(None, 512)	0





dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 1)	513
batch_normalization_11 (Batch Normalization)	(None, 1)	4
activation_11 (Activation)	(None, 1)	0
=====		
Total params: 7,151,493		
Trainable params: 7,145,603		
Non-trainable params: 5,890		
-----		



# 하드웨어 사양, 연산시간, **ACCURACY**

- 프로세서 : Intel(R) core TM i5 – 6500 CPU @ 3.20GHz
- RAM : 8.00 GB
- GPU : NVIDIA GeForce GTX 1060 3 GB
- 연산시간 : 약 28분 4초
- **Accuracy : 70.67%**

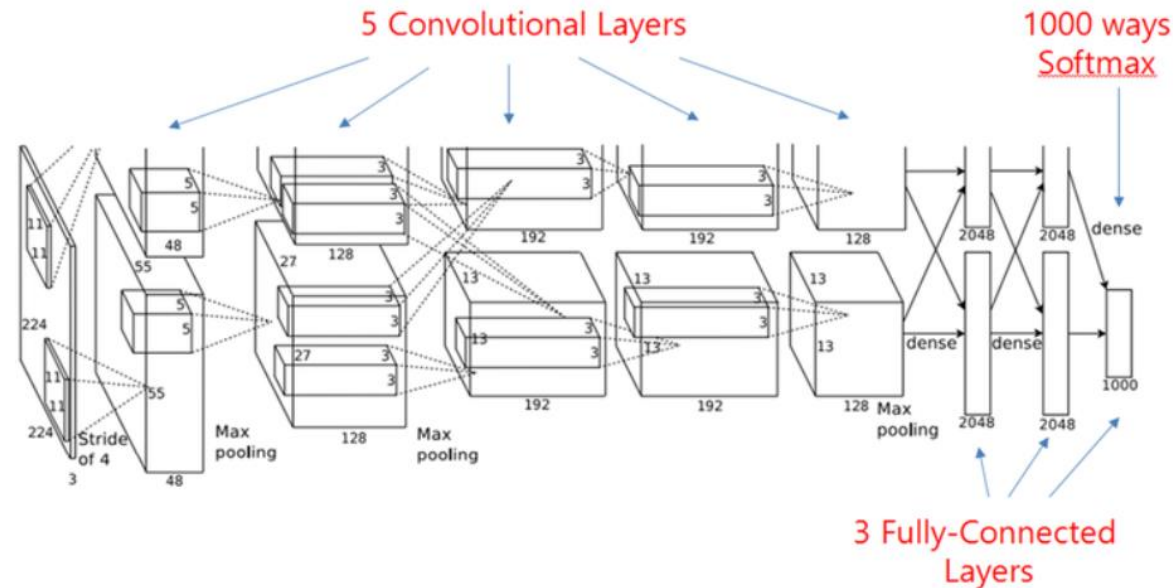
```
Accuracy: 70.67%  
[INFO] Saving the model and weights...  
Saved model to disk  
연산시간: 1684.4383420944214
```



# ALEX NET , GOOGLE NET

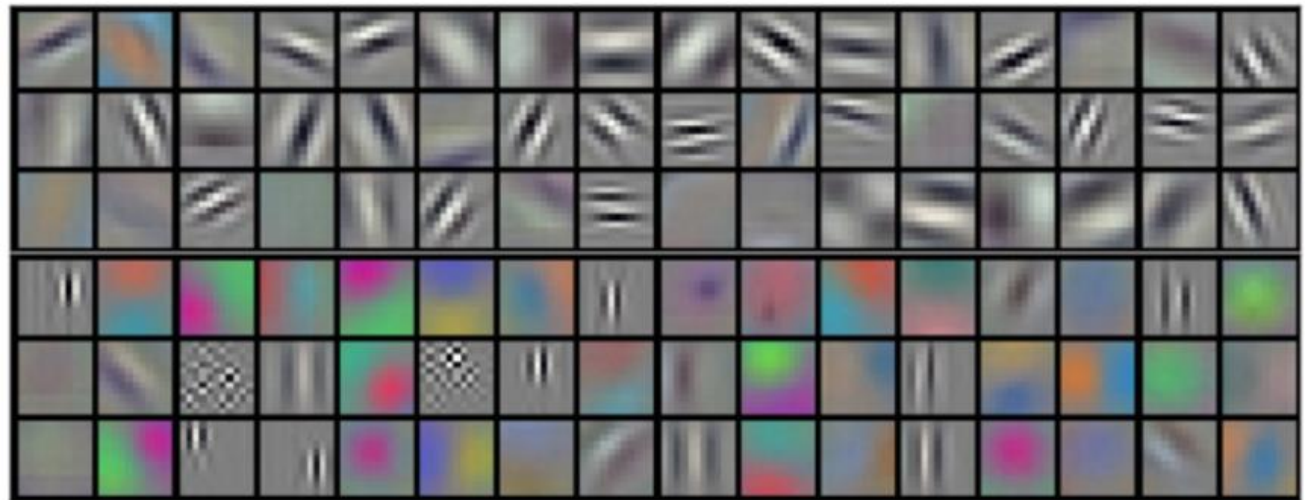
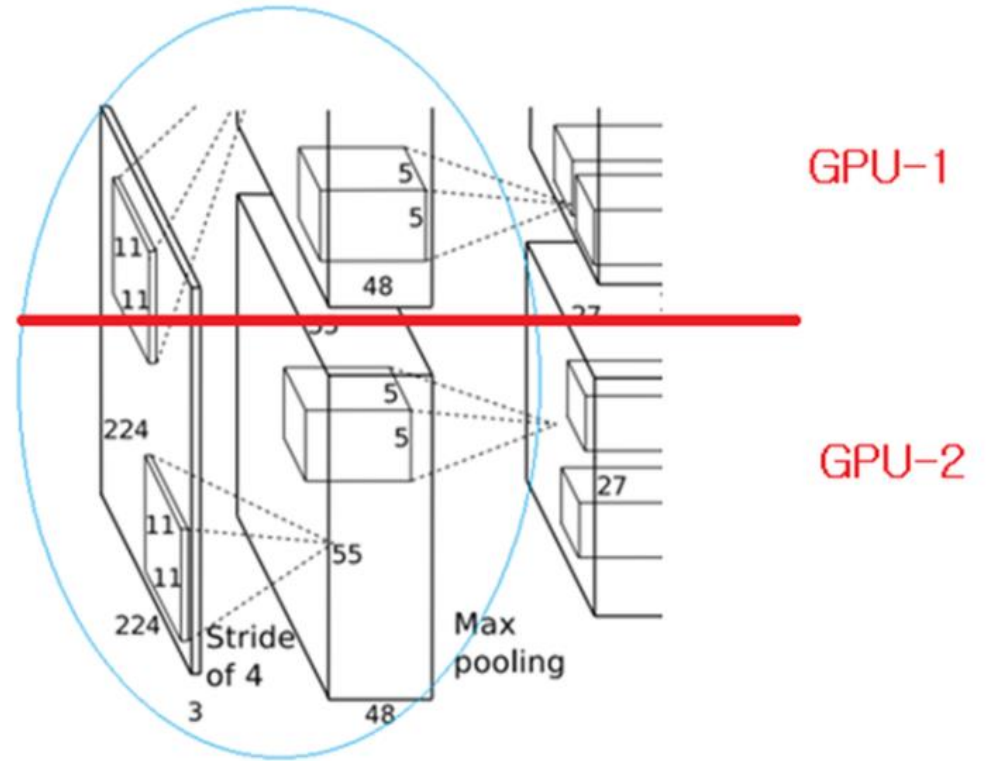
- Alex net 이란?
- ImageNet 영상 데이터 베이스를 기반으로 한, ILSVRC(ImageNet Large Scale Visual Recognition Challenge) – 2012 우승 한 CNN 알고리즘

➡ 2개의 GPU를 기반으로 한 병렬 구조



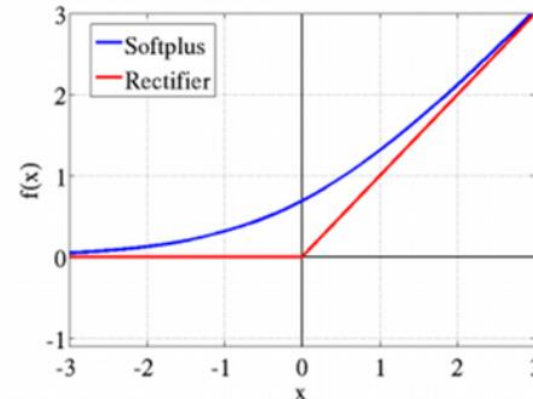
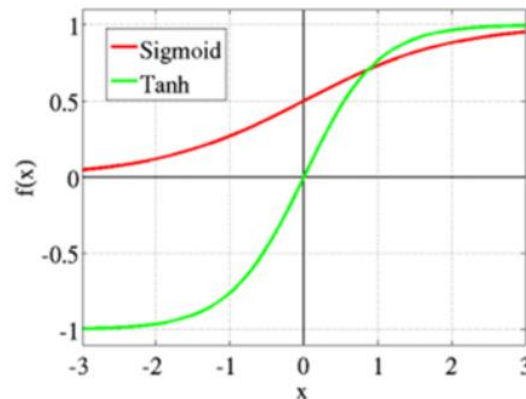
## Alex net 구조

- GPU-1 : **color**과 상관 없는 정보를 추출 하기 위 Kernel 이 학습
- GPU-2 : 주로 **color**과 관련된 정보를 추출하기 Kernel이 학습



- Alex net 성능 향상

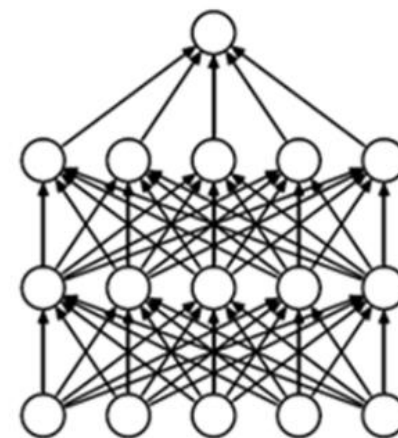
(1) ReLU :  $x = 0$ 에서 미분이 안되지만, Sigmoid나 Tanh보다 학습 속도 탁월, back propagation 결과도 단순



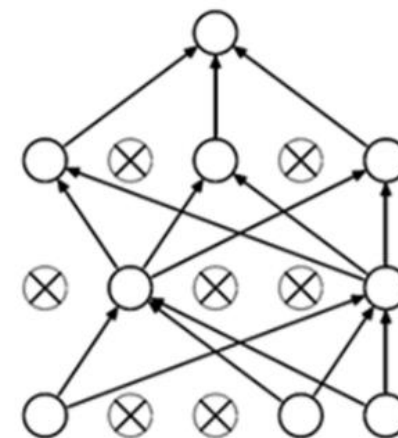
(2) Max pooling : pooling란 convolution에서 Feature map 영상의 크기를 줄이기 위한 용도이며, max pooling는 Window에서 최대값을 갖는 픽셀을 선택 ( Alex net에서는 3 x 3)

(3) Dropout : overfitting 해결

일정한 mini batch 구간 동안 생략된 망에 대한 학습을 끝내면 다시 무작위로 다른 뉴런들을 생략하면서 반복적인 학습 수행




(a) Standard Neural Net



(b) After applying dropout.



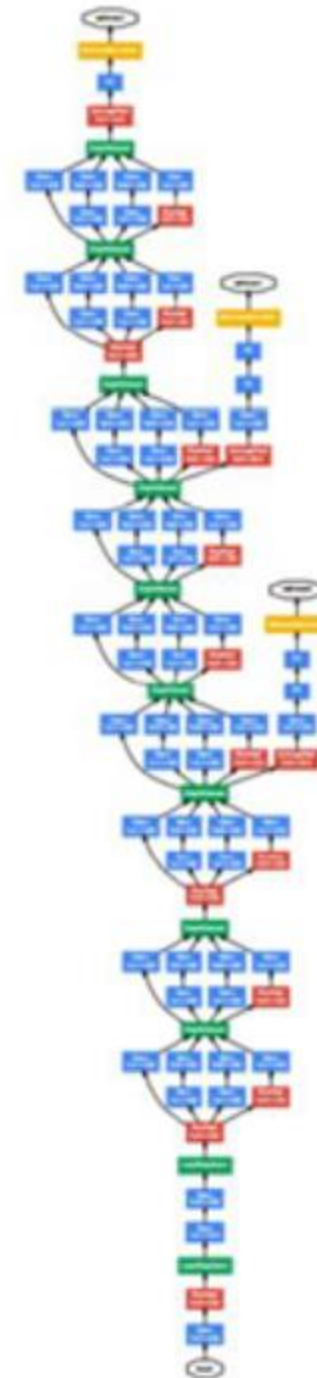
- Google net 이란?
- ILSVRC 2014 우승한 CNN 알고리즘
- 핵심 : **Networks are deeper and deeper !!!**
- 2013년도까지는 10 layers 미만이었지만, Google net은 22 layers
- \*\* 망 증가 시 free parameter의 수, 연산량 증가  overfitting 문제 발생
- 따라서, 문제 해결을 위한 **Inception model** 적용





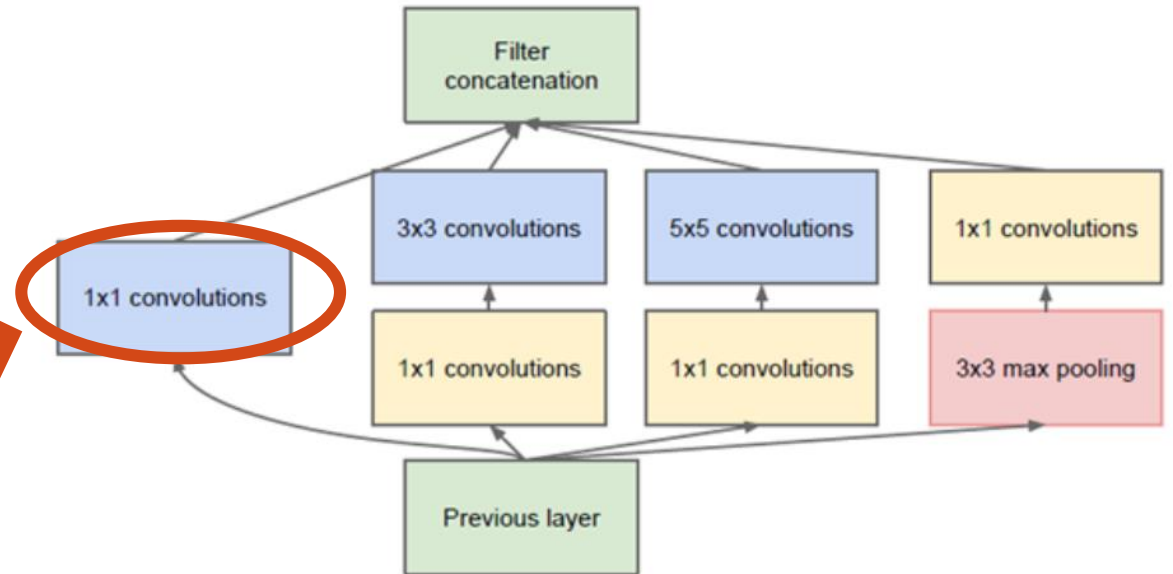
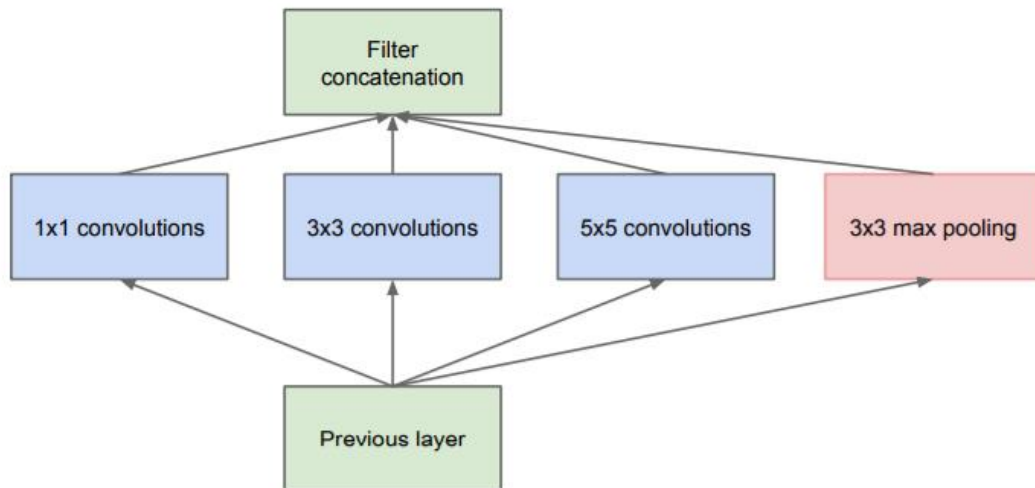


C. Szegedy et al, "Going Deeper with Convolutions" (CVPR 2015)



# INCEPTION MODEL

- 같은 layer에 서로 다른 크기를 갖는 convolution filter를 적용하여, 다른 scale의 feature를 획득
- 1 x 1 convolution을 사용하여 차원(dimension)을
- 적절히 줄이고(reduce), 망이 깊어 졌을 때,
- 연산 량 증가 문제 해결



# 22 LAYER

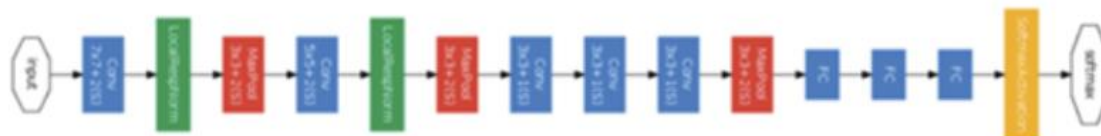
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture.



# ALEX NET 과 GOOGLE NET의 파라미터(PARAMS)의 수와 연산 량(FLOP'S) 비교

	<i>params</i>	<i>FLOPs</i>
Krizhevsky, Suskever and Hinton (2012)	60M	2B



Szegedy et al (2014)	5M	1.5B
----------------------	----	------



# ALEX NET , GOOGLE NET APPLICATION

- Alex net 검증 (validation set : cats 976 images , dogs 976 images)

23x1 Layer array with layers:

1	'input'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'fc7'	Fully Connected	4096 fully connected layer
20	'relu7'	ReLU	ReLU
21	'fc8'	Fully Connected	1000 fully connected layer
22	'prob'	Softmax	softmax
23	'classificationLayer'	Classification Output	crossentropyex with 'n01440764', 'n01443537', and 998 other classes

ans =

2x2 table

Label	Count
cats	976
dogs	976

confMat =

0.9268	0.0732
0.1098	0.8902

%	cats	dogs
cats	92.68	7.32
dogs	10.98	89.02



■ Google net 검증 (validation set : cats 77 images , dogs 77 images)

mini batch size : 10  
cross validation : 3  
validation split : 0.7 , 0.3

Layers: [144×1 nnet.cnn.layer.Layer]

Connections: [170×2 table]

Training on single GPU.

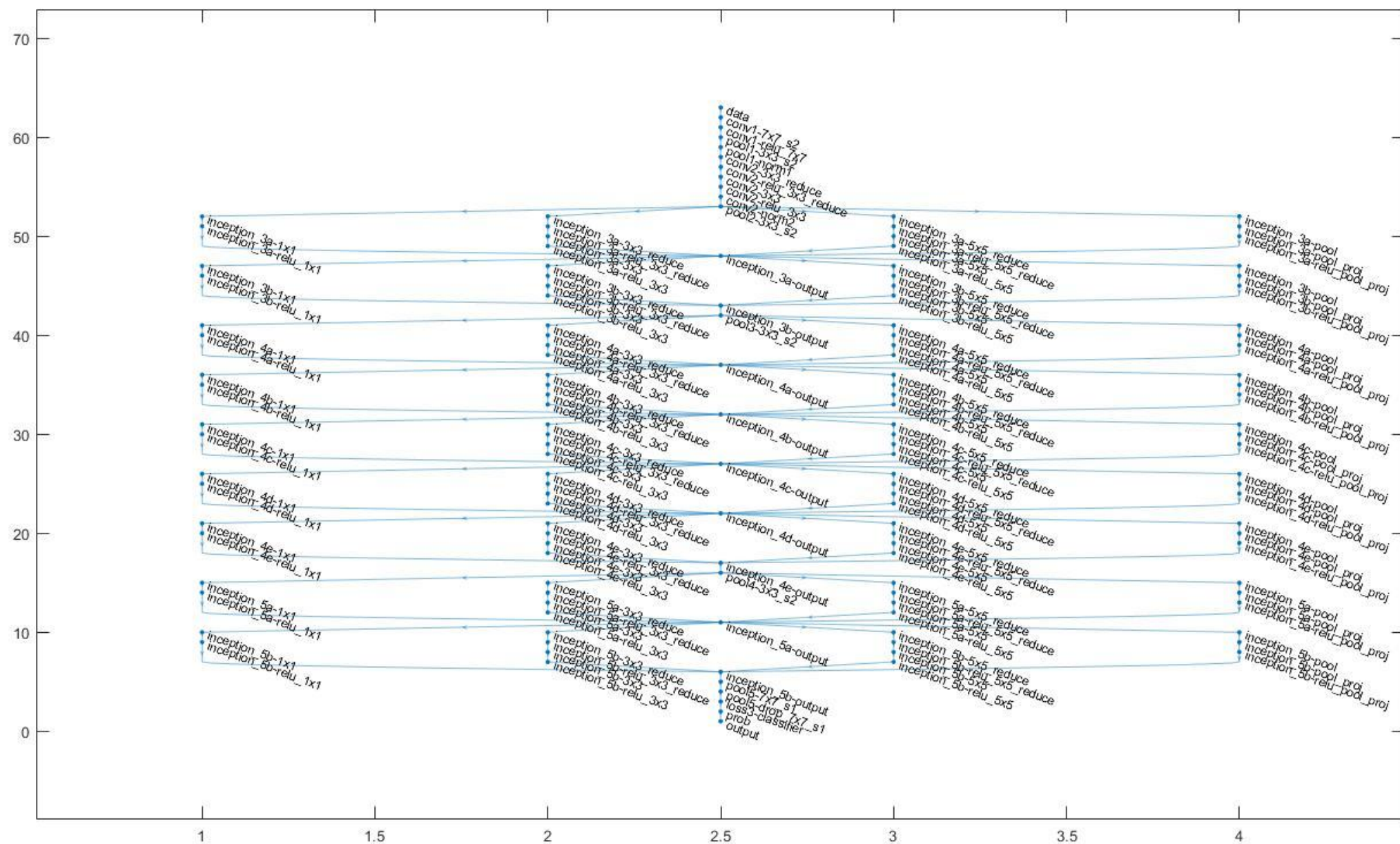
Initializing image normalization.

accuracy =

0.9348

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Validation Loss	Mini-batch Accuracy	Validation Accuracy	Base Learning Rate
1	1	0.43	0.6682	0.5906	60.00%	65.22%	1.00e-04
1	2	2.39	0.7177		40.00%		1.00e-04
1	3	3.08	0.4452	0.5106	80.00%	78.26%	1.00e-04
1	4	5.68	0.6661		60.00%		1.00e-04
1	5	6.00	0.9095		60.00%		1.00e-04
1	6	6.29	0.5651	0.2347	60.00%	93.48%	1.00e-04
1	7	8.09	0.2109		90.00%		1.00e-04
1	8	8.42	0.3843		80.00%		1.00e-04
1	9	8.75	0.1850	0.2542	100.00%	93.48%	1.00e-04
1	10	10.13	0.0413		100.00%		1.00e-04
2	11	10.43	0.3277		90.00%		1.00e-04
2	12	10.72	0.2489	0.1380	90.00%	93.48%	1.00e-04
2	13	11.82	0.1378		90.00%		1.00e-04
2	14	12.11	0.0070		100.00%		1.00e-04
2	15	12.39	0.1494	0.1026	90.00%	93.48%	1.00e-04
2	16	13.58	0.0633		100.00%		1.00e-04
2	17	13.88	0.0345		100.00%		1.00e-04
2	18	14.17	0.0098	0.0956	100.00%	93.48%	1.00e-04
2	19	15.08	0.0344		100.00%		1.00e-04
2	20	15.41	0.0010		100.00%		1.00e-04
3	21	15.71	0.0760	0.0925	100.00%	93.48%	1.00e-04
3	22	16.89	0.0149		100.00%		1.00e-04
3	23	17.16	0.0557		100.00%		1.00e-04
3	24	17.46	0.0047	0.0905	100.00%	93.48%	1.00e-04
3	25	18.46	0.1053		90.00%		1.00e-04
3	26	18.74	0.0140		100.00%		1.00e-04
3	27	19.03	0.0090	0.0889	100.00%	93.48%	1.00e-04
3	28	19.89	0.0048		100.00%		1.00e-04
3	29	20.18	0.0342		100.00%		1.00e-04
3	30	20.47	0.0009	0.0904	100.00%	93.48%	1.00e-04









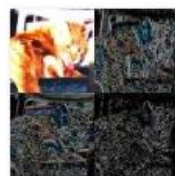
# DWT(DISCRETE WAVELET TRANSFORM) CATS ,DOGS)



165837\_c1dd297  
8e3\_o\_wav



1457998\_eabdfefeb\_o\_wav



1553668\_e1d6b16a77\_o\_wav



1583638\_d514daa1ec\_o\_wav



1586840\_2387a4e79a\_o\_wav



1594113\_4645f125b4\_o\_wav



14391096\_a15524de66\_o\_wav



14603492\_191be47594\_o\_wav



14779422\_9e72e8b412\_o\_wav



15148909\_df97bcd066\_o\_wav



15219412\_b638ac1d30\_o\_wav



15243063\_5b2cbc8c01\_o\_wav



15437110\_a461deeee6\_o\_wav



16202909\_77d8e9998b\_o\_wav



16382726\_670ab8f1aa\_o\_wav



16387910\_ca14e17f7d\_o\_wav



16432573\_2b183be805\_o\_wav



16452635\_948327d65c\_o\_wav



16560575\_5ed29382c5\_o\_wav



143564659\_8202cfc638\_o\_wav



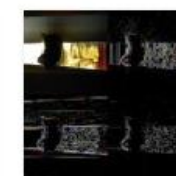
145956022\_bc8542c182\_o\_wav



146221410\_1d041638f4\_o\_wav



146369754\_03ba960a59\_o\_wav



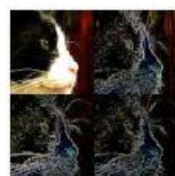
146607644\_ac2f09e533\_o\_wav



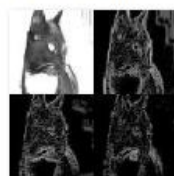
146685637\_1cb308a7e0\_o\_wav



146916868\_80d31b08ec\_o\_wav



146927634\_700f884b1b\_o\_wav



147530354\_0940eed5f3\_o\_wav



147579697\_708d312551\_o\_wav



148061496\_a3016aa17a\_o\_wav



148771017\_146de656ce\_o\_wav



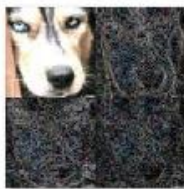
149154800\_146b4d4d12\_o\_wav







1499022\_5bead7  
4eca\_o\_wav



14324892\_59a7fb  
7926\_o\_wav



15332687\_a1d7f7  
35f4\_o\_wav



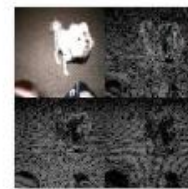
15520587\_dffd1c  
b12a\_o\_wav



15531917\_701ea  
a6a9d\_o\_wav



15679347\_7920ce  
9d76\_o\_wav



15729566\_bf6027  
d097\_o\_wav



15869719\_fdf750  
f41b\_o\_wav



143345262\_7dec  
087211\_o\_wav



143345264\_8dafa  
641a3\_o\_wav



143357497\_ab41  
a83a73\_o\_wav



144600279\_d85b  
1e8e68\_o\_wav



144824399\_fc442  
d576f\_o\_wav



145894818\_539d  
924acc\_o\_wav



146567642\_8153  
0c461e\_o\_wav



146809396\_da3e  
913ad4\_o\_wav



146838310\_e825f  
0be2c\_o\_wav



147686126\_3403  
4a5bd0\_o\_wav



148013534\_748c5  
039c9\_o\_wav



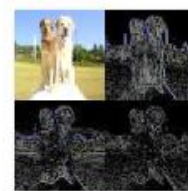
148818579\_e6f3f  
86992\_o\_wav



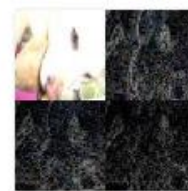
148818772\_bf7f6  
fa398\_o\_wav



148830727\_3b51  
a9c051\_o\_wav



149455812\_8dd6  
b1f1e0\_o\_wav



149798019\_28a8  
223ab9\_o\_wav



149894383\_996a  
ade99a\_o\_wav



149895119\_64d9  
7d08e2\_o\_wav



149964964\_7df42  
190db\_o\_wav



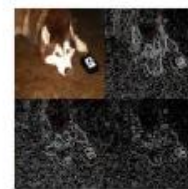
150071395\_6d7b  
ebbd90\_o\_wav



150135258\_50d1  
59b2a1\_o\_wav



150367532\_c5895  
8bf99\_o\_wav



150794181\_f8274  
9b66a\_o\_wav



151469509\_9084  
2fa09c\_o\_wav



■ DWT(Discrete Wavelet Transform) validation set( cats : 77 images , dogs : 77 images)

Epoch	Iteration	Time Elapsed	Mini-batch	Validation	Mini-batch	Validation	Base Learning
		(seconds)	Loss	Loss	Accuracy	Accuracy	Rate
1	1	3.75	0.6794	0.7729	60.00%	52.17%	1.00e-04
1	2	6.18	0.7394		60.00%		1.00e-04
1	3	6.88	0.8574	0.7031	30.00%	56.52%	1.00e-04
1	4	9.55	0.8722		40.00%		1.00e-04
1	5	9.92	0.9498		50.00%		1.00e-04
1	6	10.22	0.4789	0.5751	70.00%	65.22%	1.00e-04
1	7	11.79	0.4361		80.00%		1.00e-04
1	8	12.10	0.8257		60.00%		1.00e-04
1	9	12.41	0.5685	0.5274	80.00%	71.74%	1.00e-04
1	10	13.66	0.5026		60.00%		1.00e-04
2	11	13.96	0.1524		100.00%		1.00e-04
2	12	14.25	0.3132	0.5333	80.00%	73.91%	1.00e-04
2	13	15.41	0.3985		80.00%		1.00e-04
2	14	15.70	0.4820		70.00%		1.00e-04
2	15	16.01	0.4234	0.5384	80.00%	73.91%	1.00e-04
2	16	17.25	0.5126		80.00%		1.00e-04
2	17	17.55	0.0912		100.00%		1.00e-04
2	18	17.88	0.7960	0.5496	80.00%	78.26%	1.00e-04
2	19	19.06	0.4647		90.00%		1.00e-04
2	20	19.37	0.2492		80.00%		1.00e-04
3	21	19.66	0.0324	0.5252	100.00%	80.43%	1.00e-04
3	22	20.49	0.2599		90.00%		1.00e-04
3	23	20.79	0.2644		90.00%		1.00e-04
3	24	21.09	0.2600	0.5385	90.00%	80.43%	1.00e-04
3	25	22.19	0.2810		90.00%		1.00e-04
3	26	22.48	0.2412		90.00%		1.00e-04
3	27	22.76	0.1104	0.5378	100.00%	76.09%	1.00e-04
3	28	23.84	0.2868		90.00%		1.00e-04
3	29	24.12	0.4389		80.00%		1.00e-04
3	30	24.41	0.1122	0.5462	100.00%	76.09%	1.00e-04

accuracy =

0.7609



# GOOGLE NET 성능 향상을 위한 파라미터 조정

mini batch size : 32  
cross validation : 5  
validation split : 0.8 , 0.2

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Validation Loss	Mini-batch Accuracy	Validation Accuracy	Base Learning Rate
1	1	0.90	0.8037	0.5993	53.13%	70.00%	1.00e-04
1	2	2.30	0.6441		62.50%		1.00e-04
1	3	3.28	0.5509		84.38%		1.00e-04
2	4	4.24	0.3990		81.25%		1.00e-04
2	5	5.23	0.3082	0.2285	93.75%	90.00%	1.00e-04
2	6	6.86	0.2100		96.88%		1.00e-04
3	7	7.93	0.2087		90.63%		1.00e-04
3	8	9.08	0.1065		100.00%		1.00e-04
3	9	10.11	0.1262		96.88%		1.00e-04

accuracy =

0.9333

mini batch size : 32  
cross validation : 4  
validation split : 0.75 , 0.25

=====								
Epoch	Iteration	Time Elapsed	Mini-batch	Validation	Mini-batch	Validation	Base Learning	
		(seconds)	Loss	Loss	Accuracy	Accuracy	Rate	
=====								
1	1	1.08	0.7959	0.6687	43.75%	57.89%	1.00e-04	
1	2	4.63	0.7294		56.25%		1.00e-04	
1	3	5.63	0.5745		71.88%		1.00e-04	
2	4	6.63	0.3609	0.2693	90.63%	94.74%	1.00e-04	
2	5	8.26	0.2540		96.88%		1.00e-04	
2	6	9.16	0.2788		90.63%		1.00e-04	
3	7	10.09	0.1105		96.88%		1.00e-04	
3	8	11.13	0.1820	0.1063	90.63%	94.74%	1.00e-04	
3	9	12.62	0.2245		87.50%		1.00e-04	
=====								

accuracy =

0.9474





# Applied\_Mathematics\_Capstone\_Design

---

-Matlab : 응용수학캡스톤디자인 전공수업 (Alexnet , Googlenet) : cats vs dogs binary classification

- CNN(Convolutional Neural Network) 사용

<프로젝트 진행 과정>

- (1) Python 으로 직접 모델링 (Contests 폴더 kaggle\_cats\_dogs 참조)
- (2) image augmentation , training image wavelet transform 하였으나 accuracy 가 평균 65%
- DWT\_RGB.m : discrete wavelet transform code
- training set의 부족임을 알게되어 googlenet , alexnet 적용 ,
- At least, Matlab r2017b version, Using Computer Vision System Toolbox, Image Processing Toolbox, Neural Network Toolbox. Parallel Computing Toolbox, Statistics and Machine Learning Toolbox
- (3) Alexnet accuracy : cats : 87.04% , dogs : 83.33% , validation set : cats(77 images) , dogs(77 images)
- accuracy : cats : 92.68% , dogs : 89.02% , validation set : cats(976 images) , dogs(976 images)
- (4) Googlenet (I recommend using GPU better than CPU, because of time spending.)
- accuracy : 93.33% , validation set : cats(77 images) , dogs(77 images) -cross validation : 5, minibatch size : 32, validation : 0.2
- accuracy : 93.48% , validation set : cats(77 images) , dogs(77 images) -cross validation : 3, minibatch size : 10, validation : 0.3
- accuracy : 94.74% , validation set : cats(77 images) , dogs(77 images) -cross validation : 4, minibatch size : 32, validation : 0.25
- mini batch size : 64 -> GPU out of memory
- accuracy : 76.09% , DWT(Discrete Wavelet Transform) validation set : cats(77 images) , dogs(77 images)



# RESULT

- (1) Python 과 Matlab 의 사용을 통한 Deep Learning 에서 알고리즘의 한 종류인 CNN 직접 모델링.
- (2) 검증된 알고리즘인 Alex net , Google net 을 이용한 적용.
- <추가적인 진행 방향>
- Google net 의 내부 알고리즘 구조를 완벽히 이해 한 후, binary classification 뿐만 아니라, 실생활에 더욱더
- 밀접한 문제 해결 능력 향상.
- < Github portfolio URL (MATLAB , PYTHON related code) >
- <https://github.com/choiwb>



# 참고 문헌

- Batch Normalization: Accelerating Deep Network Training by

Reducing Internal Covariate Shift : ICML 2015

<https://pdfs.semanticscholar.org/c1ba/ed41e4bc9401b1b2ec8ef55ba45543f7a1a3.pdf>

- Classifying cats and dogs from images using deep learning

: university of Alberta

[https://github.com/shrobon/Classifying-Dogs-and-Cats-using-CNN/blob/master/MM803\\_Project\\_Reportf.pdf](https://github.com/shrobon/Classifying-Dogs-and-Cats-using-CNN/blob/master/MM803_Project_Reportf.pdf)

- <https://www.slideshare.net/ssuser06e0c5/normalization-72539464>





- Google net

<https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf>

- Alex net

<https://www.nvidia.cn/content/tesla/pdf/machine-learning/imagenet-classification-with-deep-convolutional-nn.pdf>

- Discrete Fourier Transform

[http://web.csulb.edu/~jchang9/m521/m695\\_sp10\\_FinalReport\\_Minh\\_Angela.pdf](http://web.csulb.edu/~jchang9/m521/m695_sp10_FinalReport_Minh_Angela.pdf)

