

하용호
@kakao

백날 자습해도 이해 안 가던 딥러닝,
머리속에 인스톨 시켜드립니다.



오랜만이죠? 이 짤?

딥러닝이 뜨겁습니다.

딥러닝 가능케한 3대장

빅데이터

이제는 식상한 용어지만
하여간 있다!

GPU

다나와 가서
사면 된다!

알고리즘

아..

뭔가 옛날에 뉴럴넷은 배웠는데..

혼자서 공부하려고 인터넷을 뒤져보지만

빡셉니다...

예습, 자습, 복습 - 인간이 할 수 없는 세가지 일

일단 단어들부터 보통이 아닙니다.

DropOut

ReLU

MaxPooling

Convolution

**Stochastic
Gradient
Decent**

SoftMax

근데 이것도 한번 보시죠

French Fries

Hash
Browns

McFlurry

McMuffin

BigMac

근데 이것도 한번 보시죠

프렌치프라이

해시
브라운

맥플러리

맥머핀

빅맥

사실 별거 아닌건데 외국인 만난것처럼 당황해서 그렇습니다.

조선시대 조상님이 맥도날드 메뉴보면
얼마나 겁날까요?

근데 사실 그냥 먹는 이야기. 별거 없죠.
그분들도 금방 적응하실 겁니다.

이 시간을 통해, 딥러닝도
편안하게 느끼게 되셨으면 좋겠습니다.

딥러닝이 글케 좋아?

문제?

컴퓨터에게 그림을 주고

개인지 고양이인지 판단하게 하시오?

이 프로그램 어떻게 짜지?



(이제는 식상한 짤... 이 짤을 쓰는 내가 부끄럽네요)

딥러닝으로 만든 코드를 봅시다.

VGG16 모델 만들기

```
def ConvBlock(layers, model, filters):
    for i in range(layers):
        model.add(ZeroPadding2D((1,1)))
        model.add(Convolution2D(filters, 3, 3, activation='relu'))
    model.add(MaxPooling2D((2,2), strides=(2,2)))
```

```
def FCBLOCK(model):
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
```

```
# 약간의 전처리로, 0~255사이의 rgb값을 평균을 빼서 중간으로 맞춰줍니다.
vgg_mean = np.array([123.68, 116.779, 103.939]).reshape((3,1,1))
```

```
def vgg_preprocess(x):
    x = x - vgg_mean          # subtract mean
    return x[:, ::-1]           # reverse axis bgr->rgb
```

```
# 오리지널 VGG16 모델
def VGG_16():
    model = Sequential()
    model.add(Lambda(vgg_preprocess, input_shape=(3,224,224),
                    output_shape=(3,224,224)))
```

```
    ConvBlock(2, model, 64)
    ConvBlock(2, model, 128)
    ConvBlock(3, model, 256)
    ConvBlock(3, model, 512)
    ConvBlock(3, model, 512)
```

```
    model.add(Flatten())
    FCBLOCK(model)
    FCBLOCK(model)
    model.add(Dense(1000, activation='softmax'))
    return model
```

모델 만들고, ImageNet으로 훈련된 가중치 불러오기

```
model = VGG_16()
fpath = "/home/deploy/notebook/henry/vgg16.h5"
model.load_weights(fpath)
```

```
def get_batches(path, gen=image.ImageDataGenerator(), shuffle=True,
                batch_size=8, class_mode='categorical'):
    return gen.flow_from_directory(path, target_size=(224,224),
                                 class_mode=class_mode, shuffle=shuffle, batch_size=batch_size)
```

VGG모델을 개, 고양이를 알아볼 수 있게 finetune하자.

```
model.pop() # 말미에 ImageNet 1000가지 분류를 뺍니다.  
for layer in model.layers: layer.trainable=False # 가중치 안변하게 고정  
model.add(Dense(2, activation='softmax')) # 2가지를 분류할 수 있는 레이어를 추가  
model.compile(optimizer=Adam(lr=0.001),  
              loss='categorical_crossentropy', metrics=['accuracy'])
```

트레이닝 ¶

```
batch_size=64
```

```
path = "/home/deploy/notebook/henry/dogscats/"  
#path = "/home/deploy/notebook/henry/dogscats/sample/"
```

```
batches = get_batches(path+'train', batch_size=batch_size)  
val_batches = get_batches(path+'valid', batch_size=batch_size)
```

Found 23000 images belonging to 2 classes.

Found 2000 images belonging to 2 classes.

```
model.fit_generator(batches, samples_per_epoch=batches.nb_sample,  
                    validation_data=val_batches,  
                    nb_val_samples=val_batches.nb_sample,  
                    nb_epoch=2)
```

Epoch 1/2

23000/23000 [=====] - 723s - loss: 0.0896 - acc: 0.9800 - val_loss: 0.0847 - val_acc: 0.9810

Epoch 2/2

23000/23000 [=====] - 724s - loss: 0.1045 - acc: 0.9781 - val_loss: 0.0925 - val_acc: 0.9835

정확도 98.35% !!



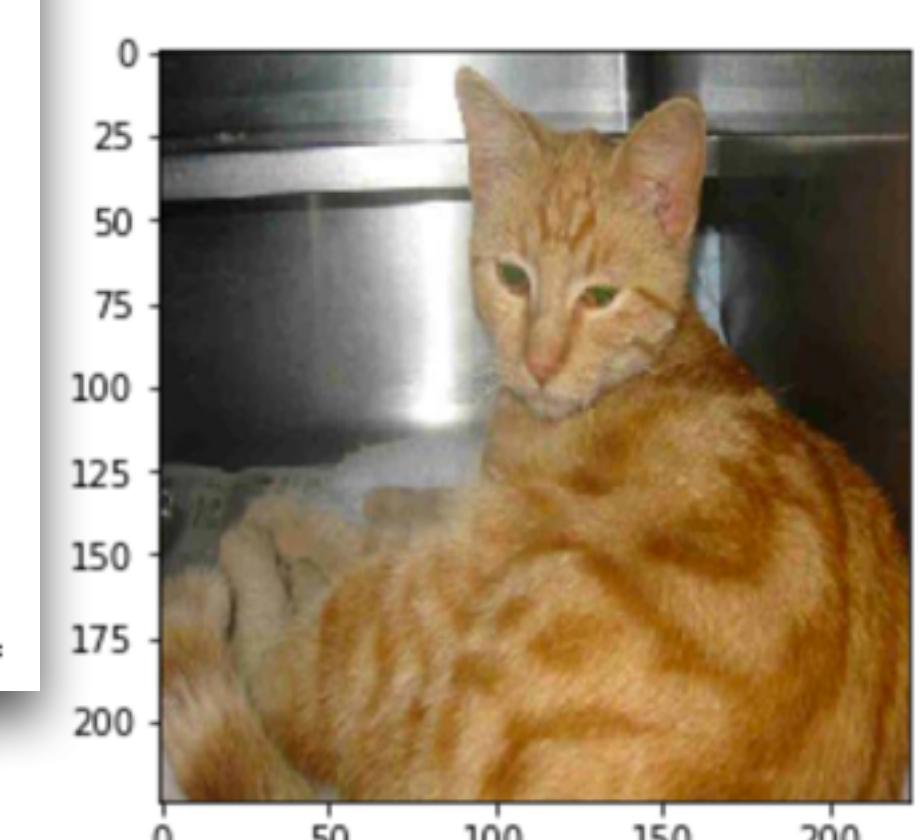
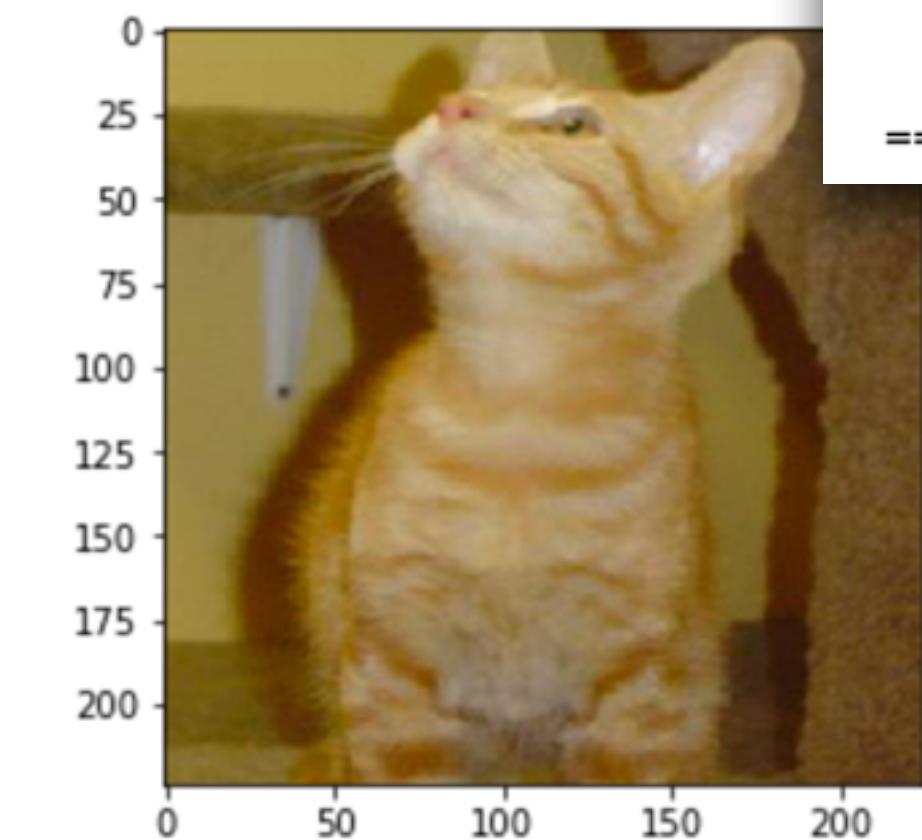
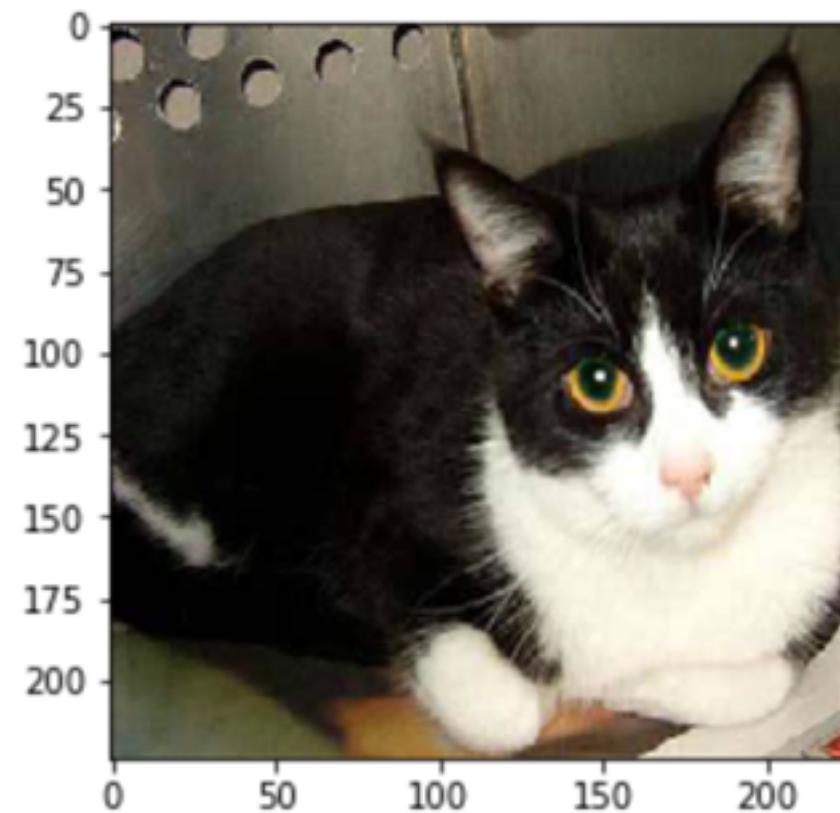
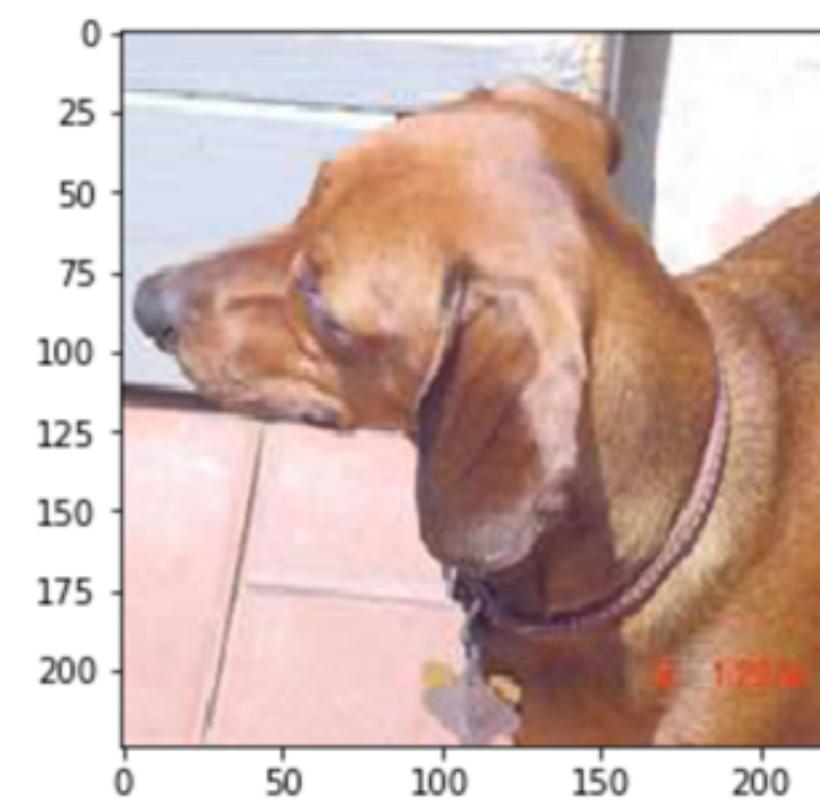
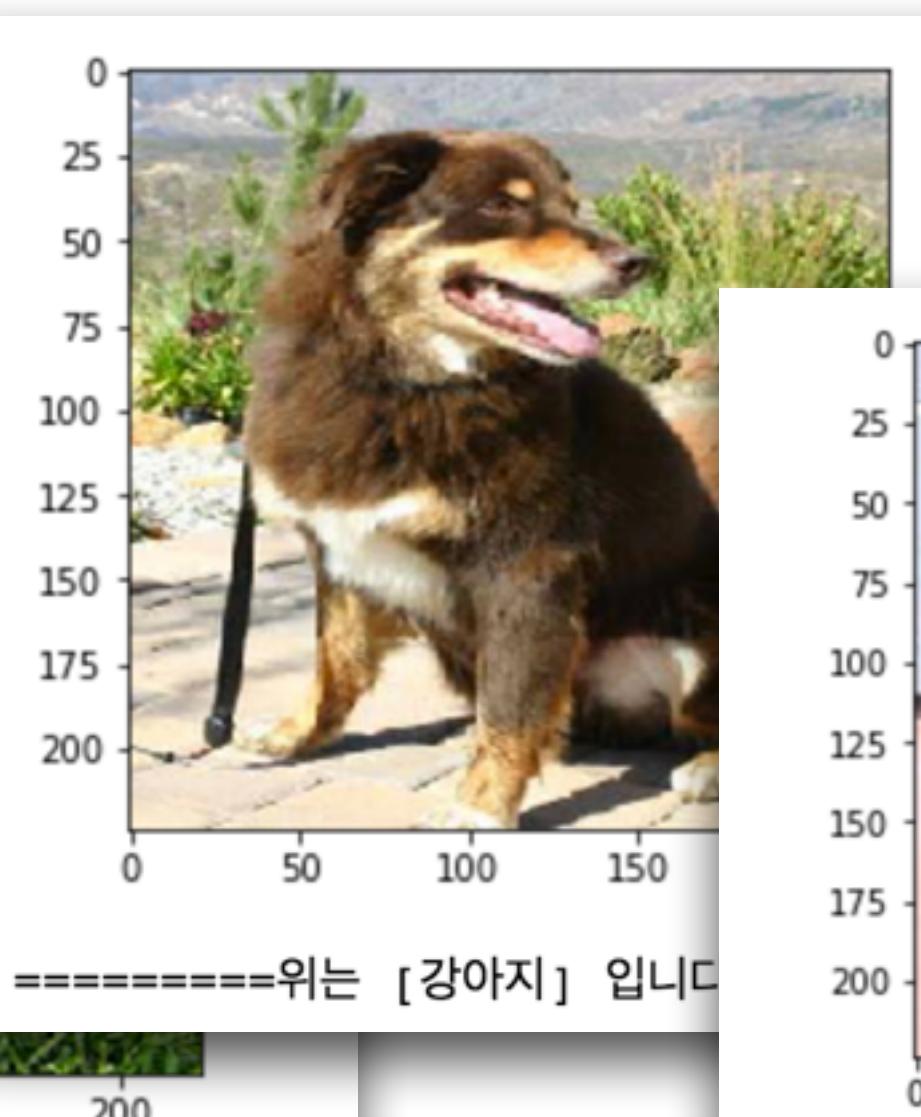
실제로 추측해봅시다!!

```
# 그림 불러오고, 딥러닝이 판단해서 결과 보여줌
```

```
def predict_catdog(img_path, model):
    img = image.load_img(img_path, target_size=(224, 224))
    plt.imshow(img)
    x = np.expand_dims(image.img_to_array(img), axis=0)
    doggy = model.predict_classes(x, verbose=0)[0] # 강아지면 1, 고양이면 0리턴
    name = u"강아지" if doggy else u"고양이"
    plt.show()
    print(u"=====위는 [%s] 입니다.===== %s")
```

```
# 랜덤으로 6개만 뽑아다가 추측해보자.
```

```
test_files = glob.glob(path+'valid/*/*.jpg')
for i in random.sample(test_files, 6):
    predict_catdog(i, model)
```



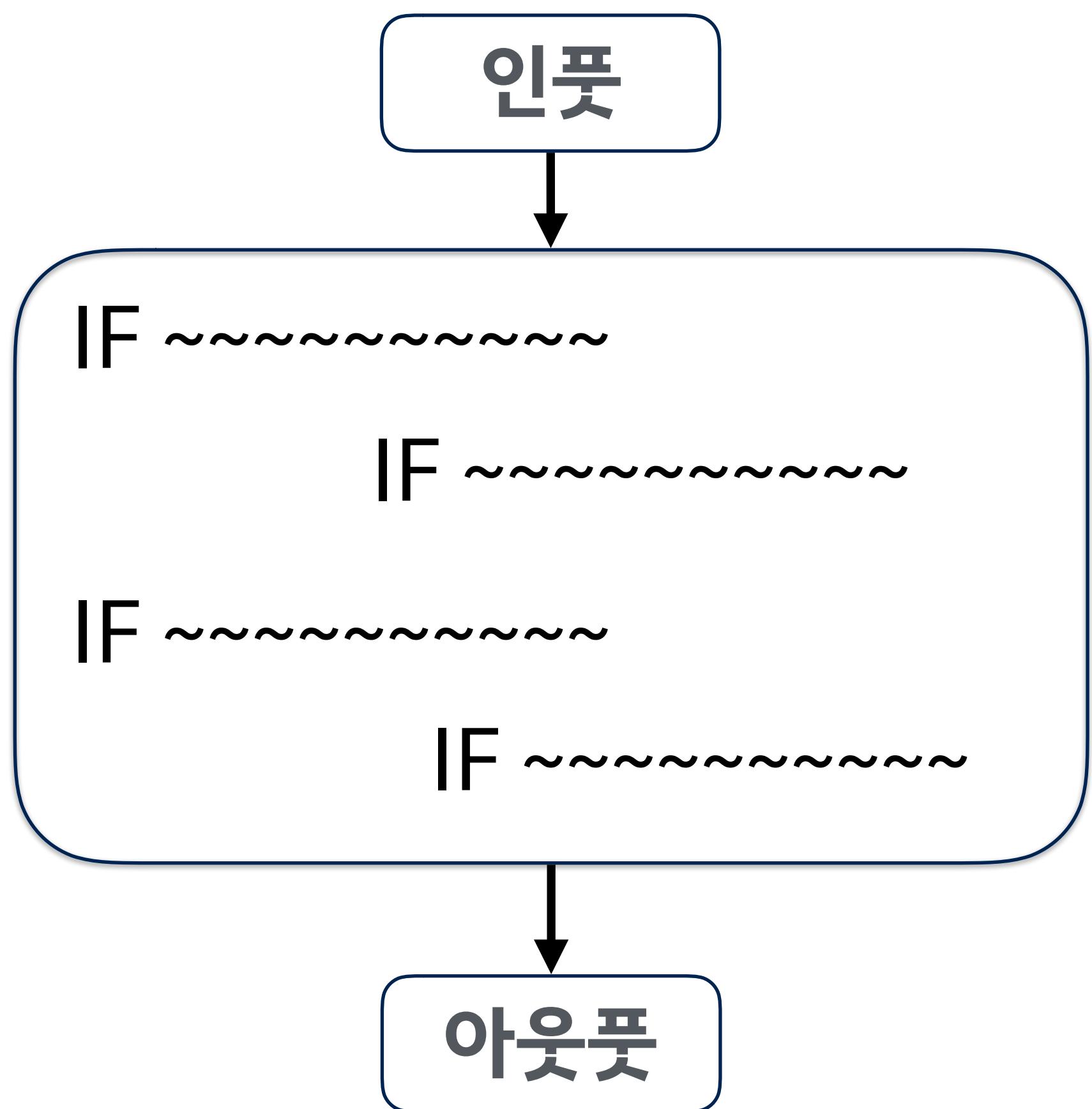
엄청 잘 맞춘다!!

**사진을 보고 개랑 고양이를 구분하는
신박한 일을, 달랑 저 짧은 코드로 할 수 있다!**

패러다임 쉬프트

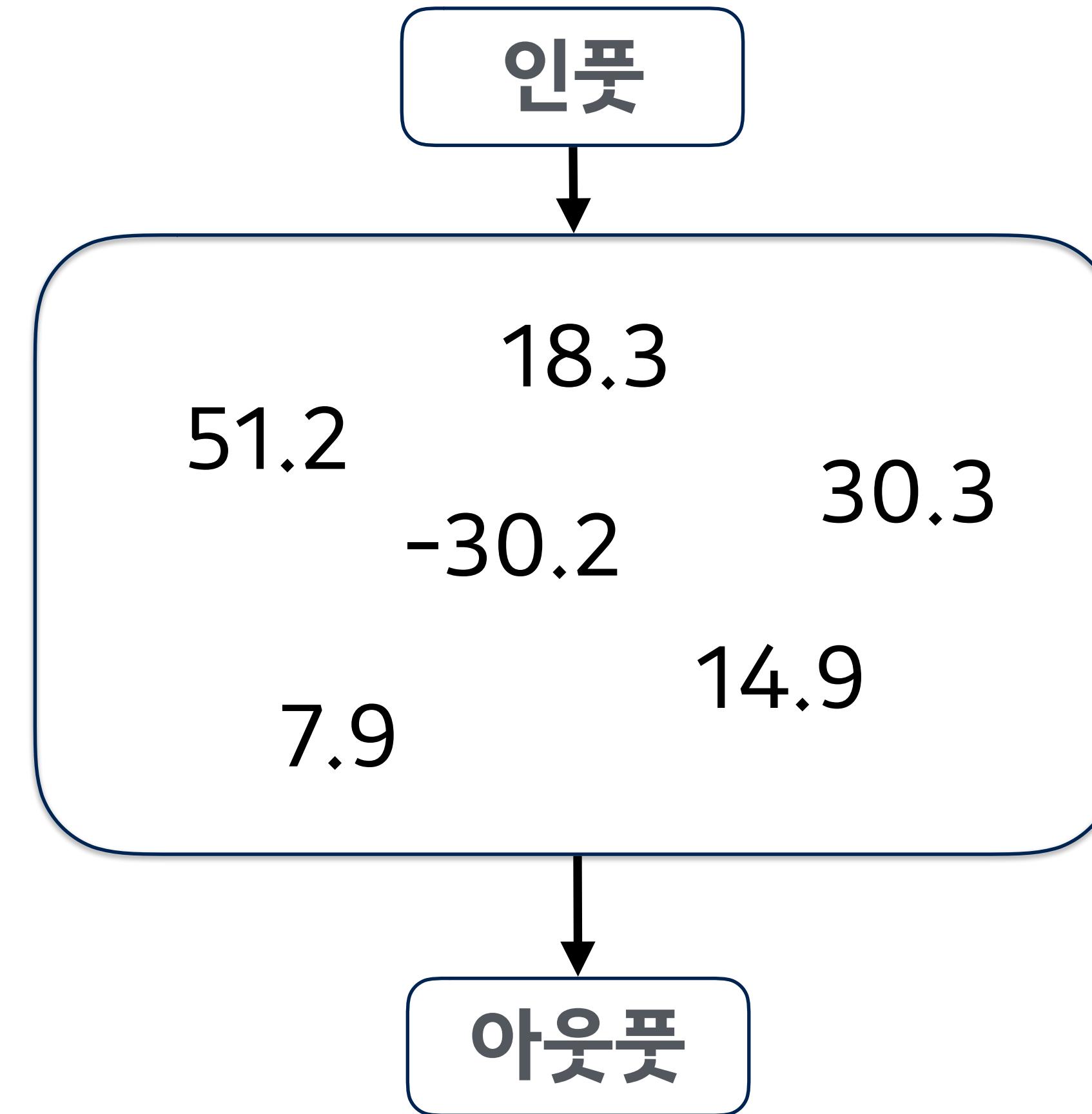
기존의 코딩:

조건을 라인바이라인으로 **긴 코드**로 써내려 가는 일



앞으로의 코딩:

조건을 학습 **모델의 여러 가중치로 변환**하는 일



딥러닝의 정수중의 하나: VGG16

```
# 오리지널 VGG16 모델
def VGG_16():
    model = Sequential()
    model.add(Lambda(vgg_preprocess, input_shape=(3,224,224)))

    ConvBlock(2, model, 64)
    ConvBlock(2, model, 128)
    ConvBlock(3, model, 256)
    ConvBlock(3, model, 512)
    ConvBlock(3, model, 512)

    model.add(Flatten())
    FCBLOCK(model)
    FCBLOCK(model)
    model.add(Dense(1000, activation='softmax'))

    return model

model = VGG_16()
model.compile(optimizer=Adam(lr=0.001),
              loss='categorical_crossentropy', metrics=[ 'accuracy' ])
```

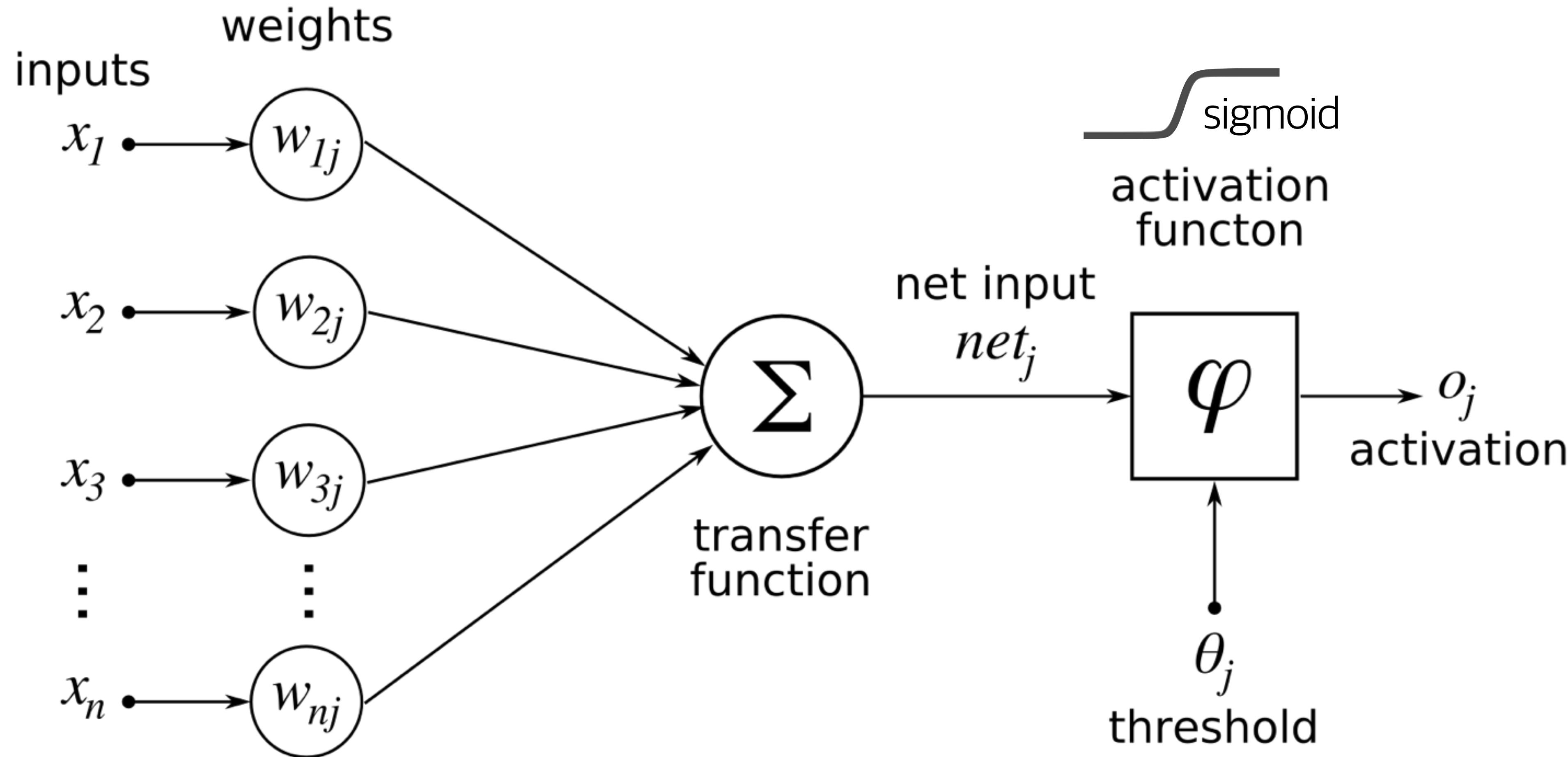
```
def ConvBlock(layers, model, filters):
    for i in range(layers):
        model.add(ZeroPadding2D((1,1)))
        model.add(Convolution2D(filters, 3, 3,
                               activation='relu'))
    model.add(MaxPooling2D((2,2), strides=(2,2)))
```

```
def FCBLOCK(model):
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
```

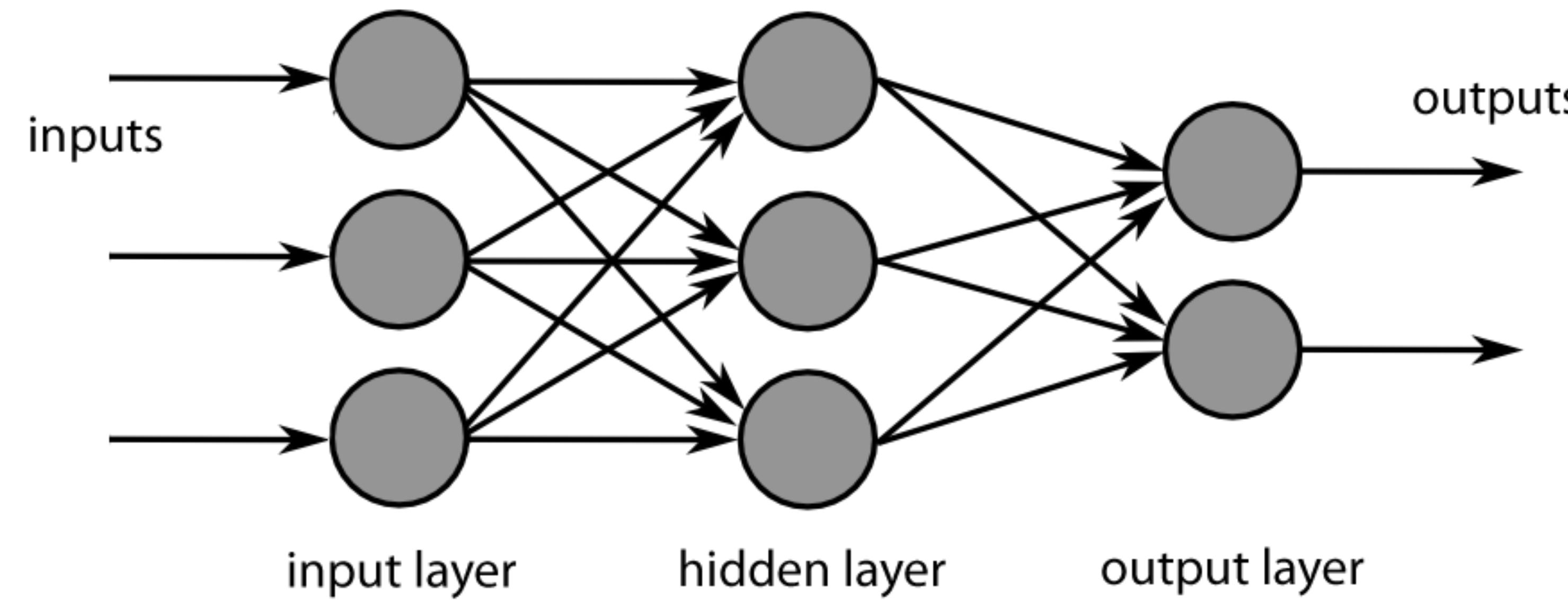
강의가 끝날 때쯤 다 이해 갈겁니다^^

일단 뉴얼넷 복습부터

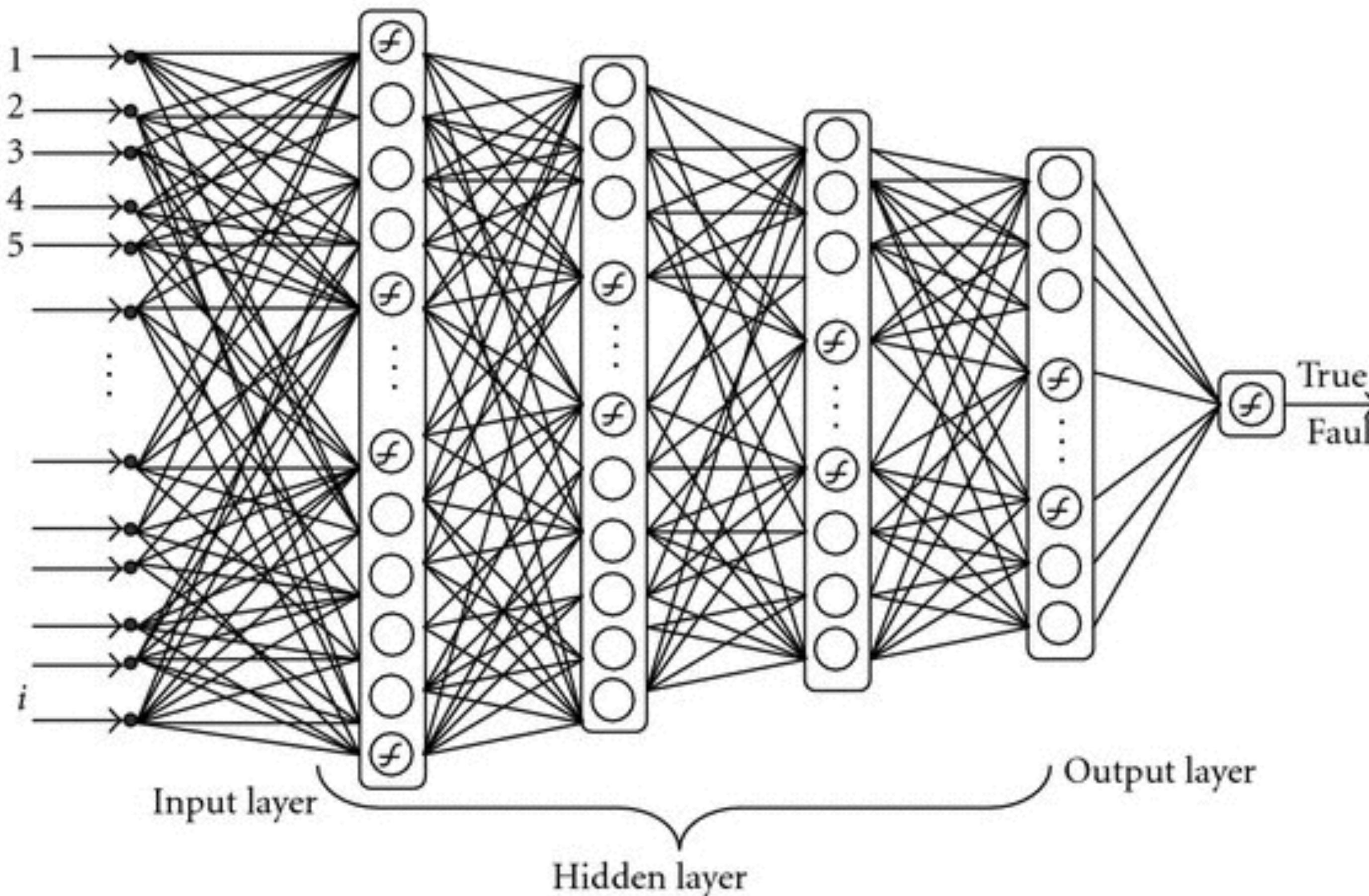
일단 뉴럴넷의 한 컴포넌트는 일케 생겼었죠



요런게 모여서 이런게 되었었습니다.



계속 층을 쌓아나갈 수도 있습니다.



이렇게 깊어지면 딥뉴럴넷(DNN)이라 하죠

**사람의 뇌신경을 닮은 뉴럴넷이니
사람이 하는거 다 할 수 있지 않을까?!**

그런데 잘 안됨

세가지 문제가 있었다.

덜하거나

Underfitting 학습이 잘 안돼!

느리거나

Slow 도대체 학습은 언제 끝나는 건가?

과하거나

Overfitting 겨우 되어도 융통성이 없다?!

(하여간 적당히가 없어요)

차근차근 해결해봅시다!

덜하거나

Underfitting

학습이 잘 안돼!

느리거나

Slow

도대체 학습은 언제 끝나는 건가?

과하거나

Overfitting

겨우 되어도 융통성이 없다?!

덜하거나
Underfitting

학습이 잘 안돼!

느리거나
Slow

도대체 학습은 언제 끝나는 건가?

과하거나
Overfitting

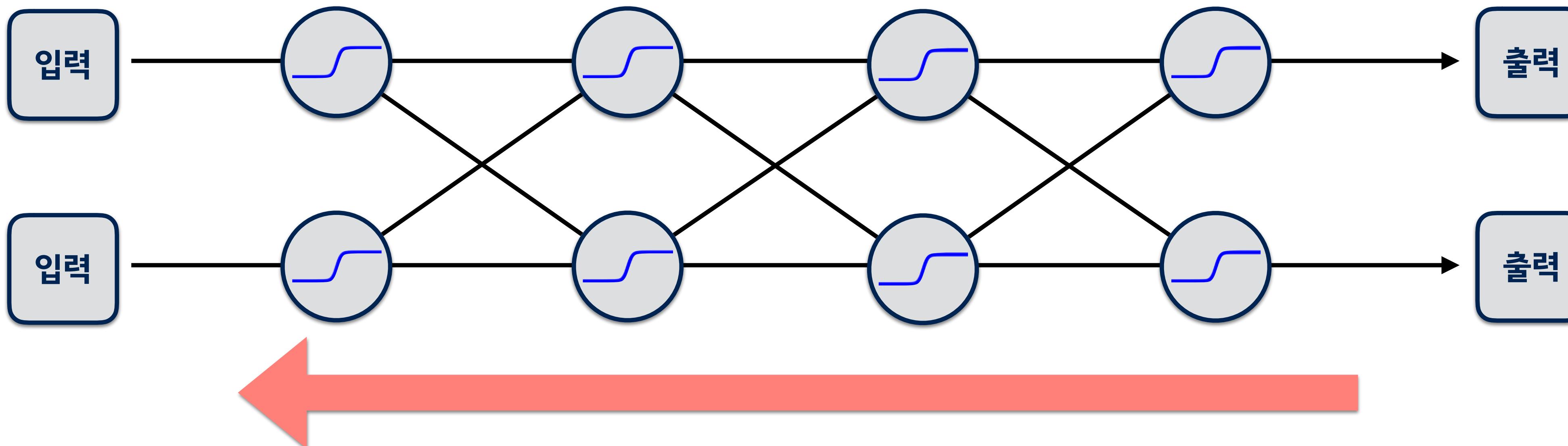
겨우 되어도 융통성이 없다?!

뉴럴넷의 학습방법 Back propagation

(사실 별거 없고 그냥 “뒤로 전달”)

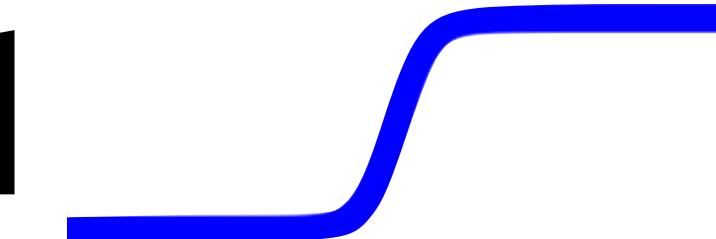
뭐를 전달하는가?

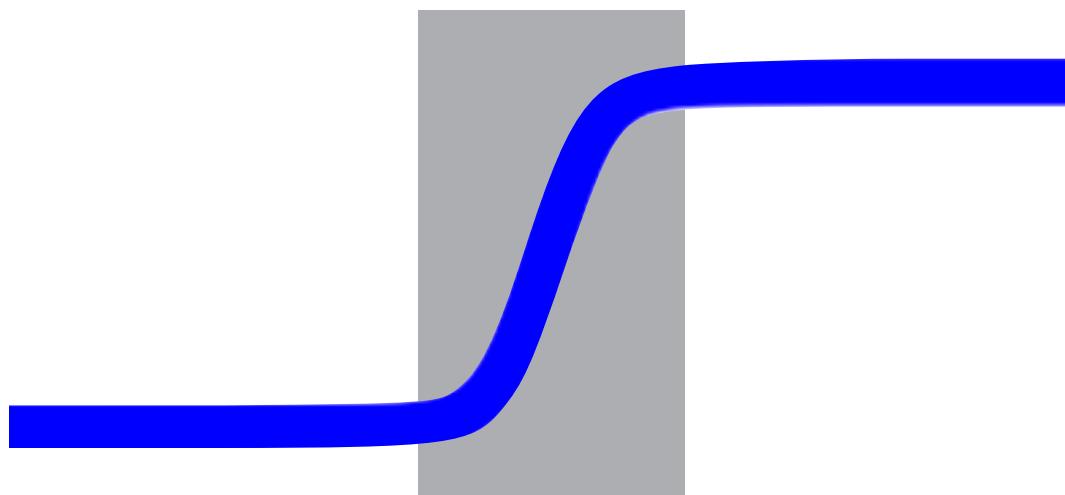
현재 내가 틀린정도를 ‘미분(기울기)’ 한 거



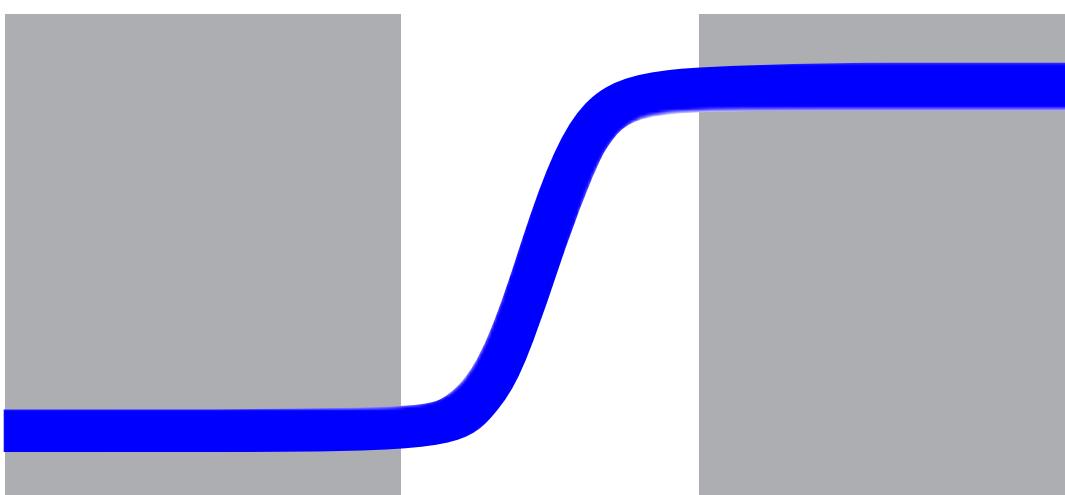
미분하고, 곱하고, 더하고를 역방향으로 반복하며 업데이트한다.

근데 문제는?

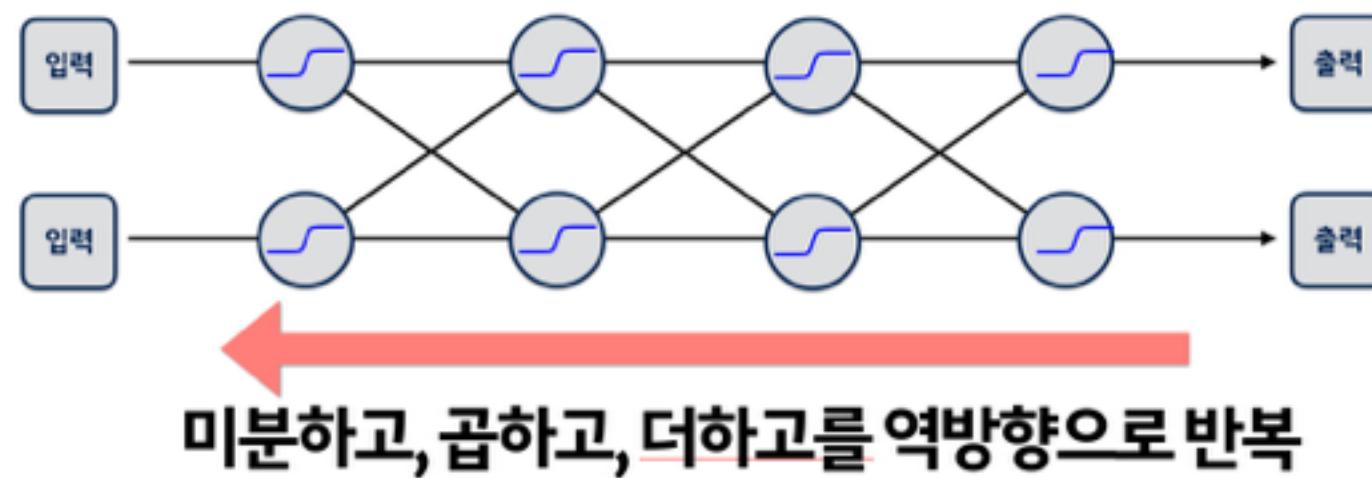
우리가 activation 함수로 sigmoid  를 썼다는 것



여기의 미분(기울기)는 뭐라도 있다. 다행



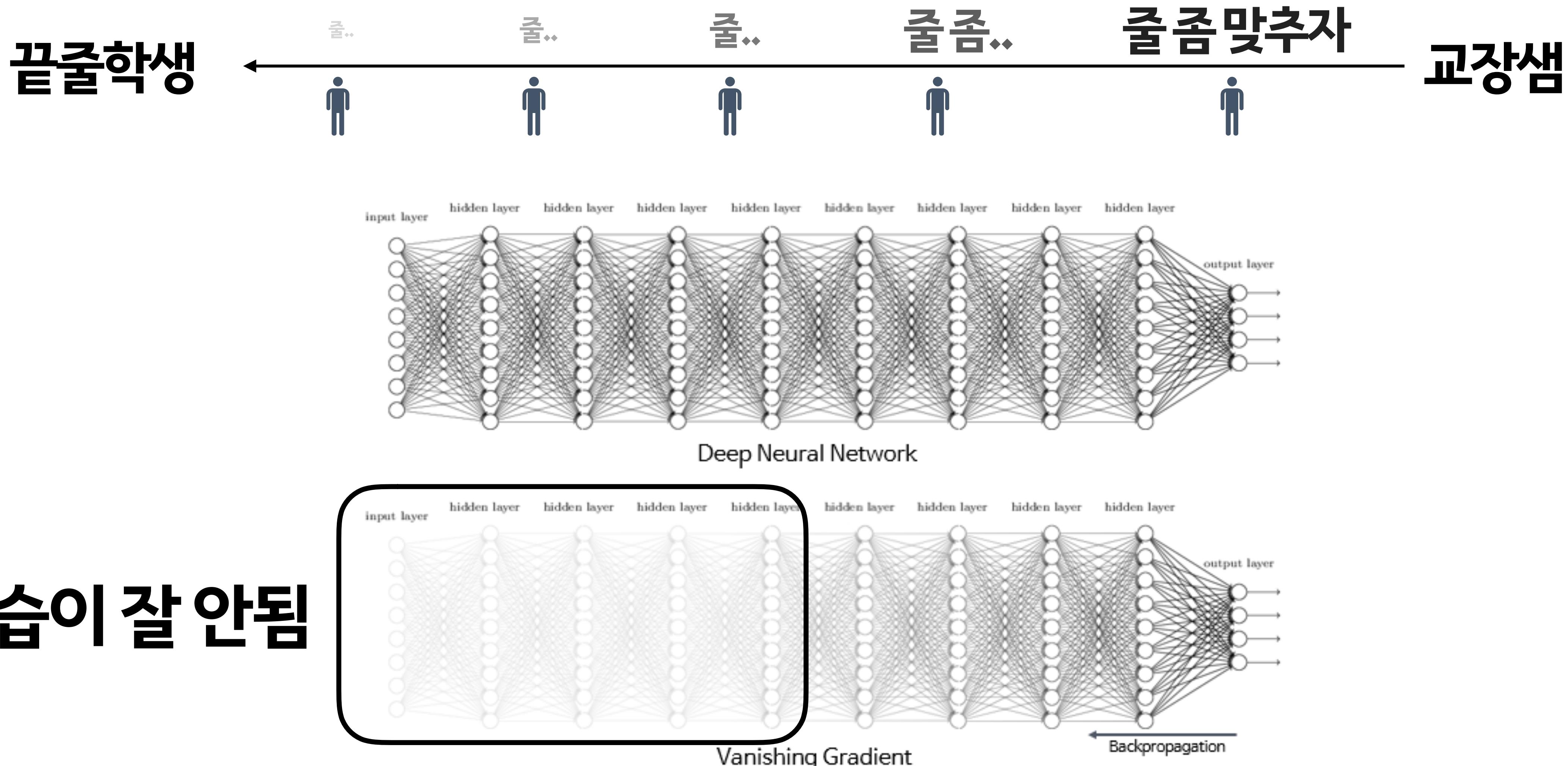
근데 여기는 기울기 0.. 이런거 중간에 곱하면 뭔가 뒤로 전달할게 없다?!



그런 상황에서 이걸 반복하면??????

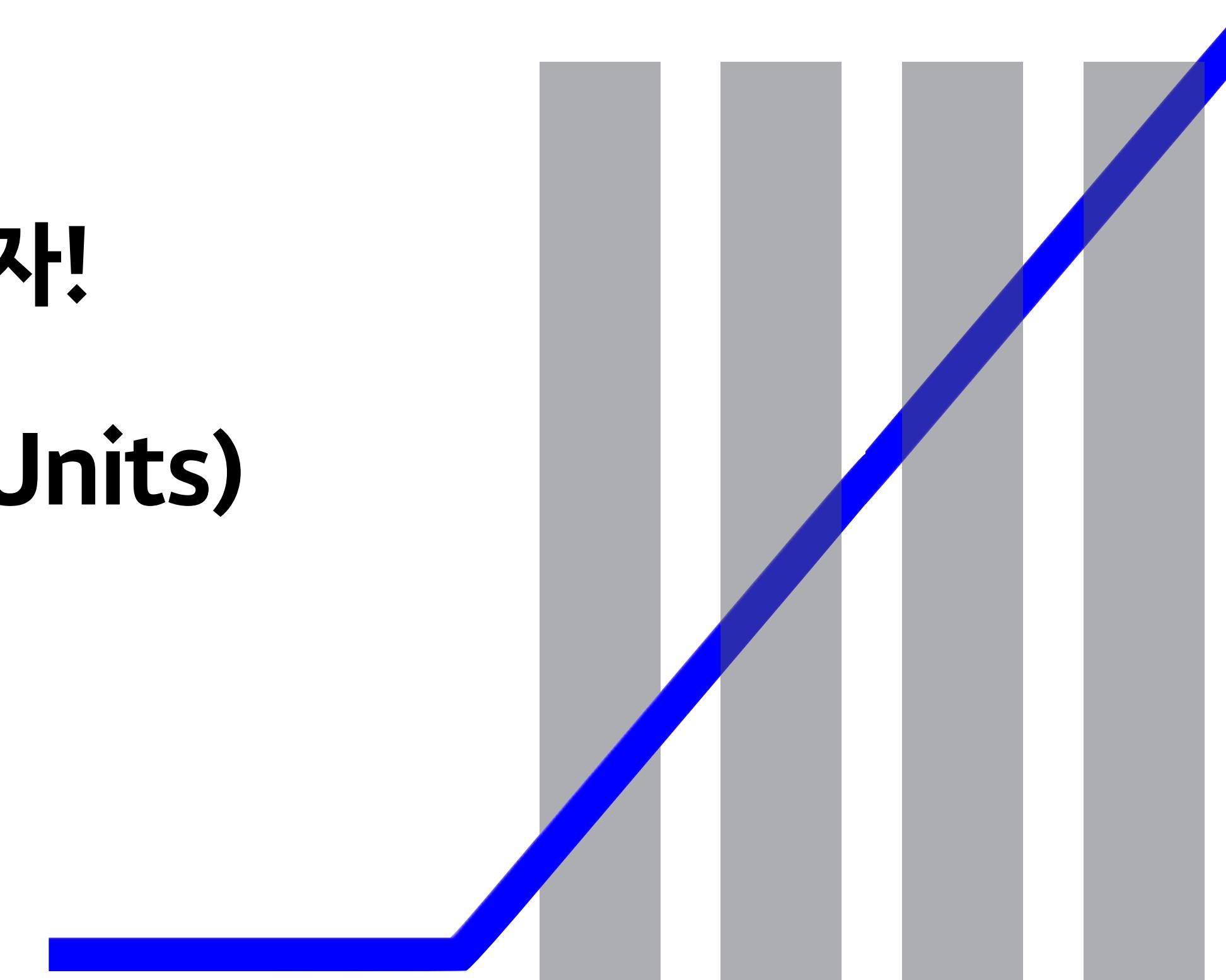
미분하고, 곱하고, 더하고를 역방향으로 반복

Vanishing gradient 현상 : 레이어가 깊을 수록 업데이트가 사라져간다. 그래서 fitting이 잘 안됨(underfitting)



사그라드는 sigmoid 대신
죽지 않는 activation func 을 쓰자!

→ **ReLU** (Rectified Linear Units)



이녀석은 양의 구간에서 전부 미분값(1)이 있다!



꼴 줄 학생까지 이야기가 전달이 잘 되고 위치를 고친다!

문제?

전달하다가, 사그라져 버린다(vanishing gradient)

해결!

sigmoid -> ReLU = 뒤로전달 오케이!

덜하거나
Underfitting

학습이 잘 안돼!

느리거나
Slow

도대체 학습은 언제 끝나는 건가?

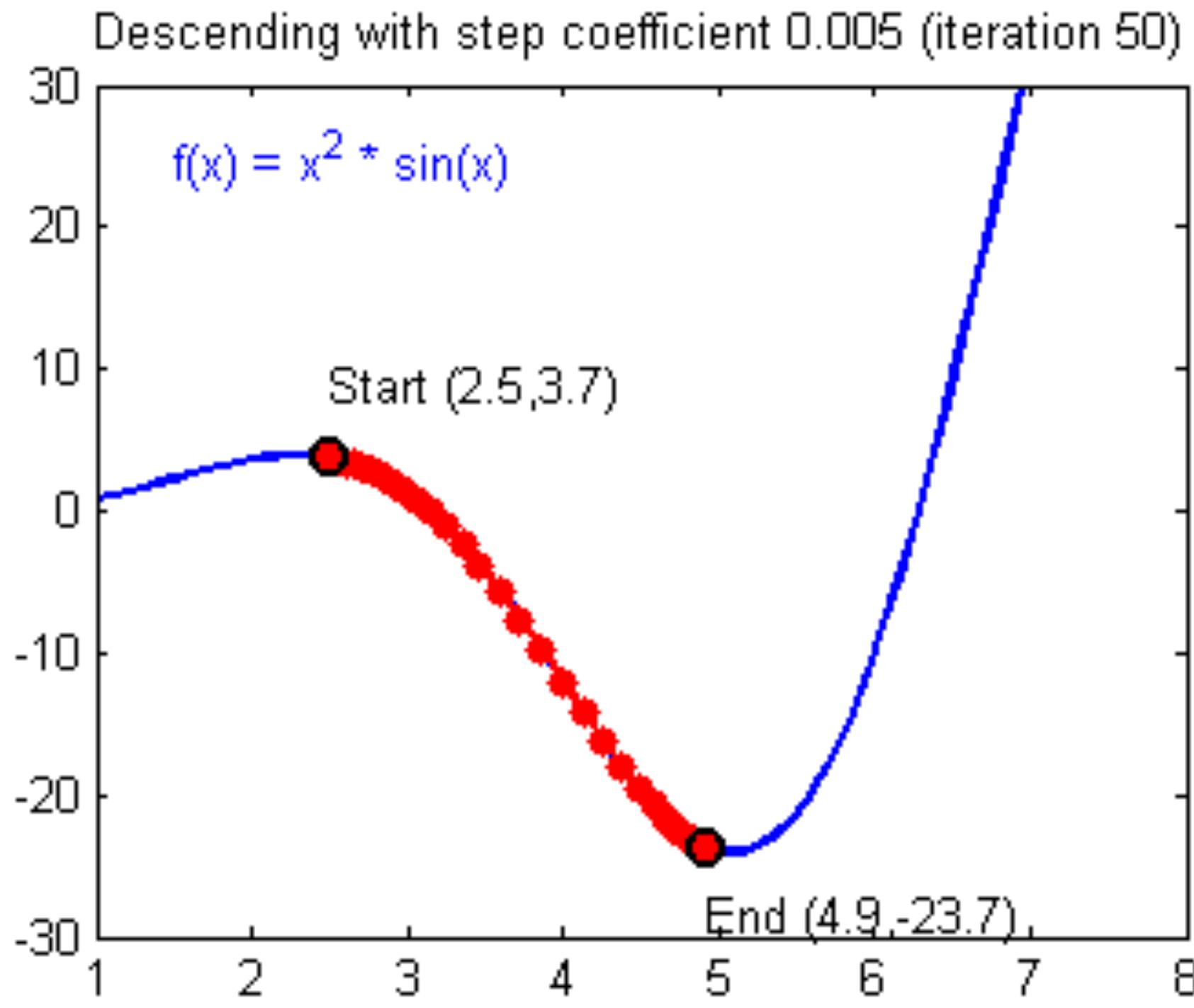
과하거나
Overfitting

겨우 되어도 융통성이 없다?!

기존 뉴럴넷이 가중치 parameter들을
최적화(optimize)하는 방법

Gradient Decent

loss function의 현 가중치에서의 기울기(gradient)를 구해서
loss를 줄이는 방향으로 업데이트해 나간다.



뉴럴넷은 loss(or cost) function을 가지고 있습니다. 쉽게 말하면 “틀린 정도”

현재 가진 weight 세팅(내 자리)에서,
내가 가진 데이터를 다 넣으면
전체 에러가 계산됩니다.

거기서 미분하면 에러를 줄이는 방향을 알 수 있습니다.
(내자리의 기울기 * 반대방향)

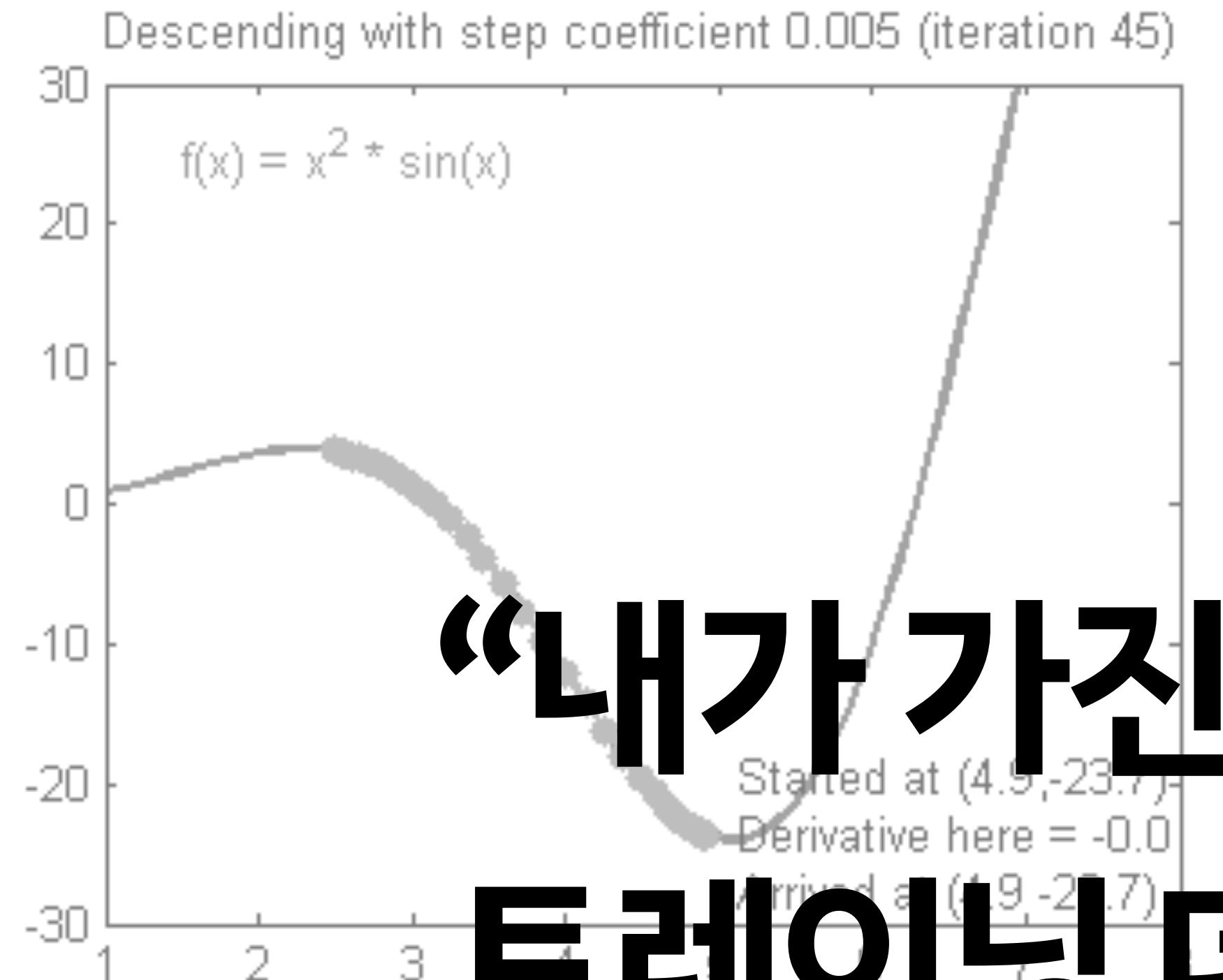
그 방향으로 정해진 스텝량(learning rate)을
곱해서 weight을 이동시킵니다. 이걸 반복~~

$$\text{weight의 업데이트} = \frac{\text{에러 낮추는 방향}}{\text{(decent)}} \times \frac{\text{한발자국 크기}}{\text{(learning rate)}} \times \frac{\text{현 지점의 기울기}}{\text{(gradient)}}$$

$$= -\gamma \nabla F(\mathbf{a}^n)$$

아 그렇구나.

..잠깐? 이상한게 지나갔는데?



“내가 가진 데이터를 다 넣으면?”
트레이닝데이터가 몇억건인데...
한발자국 갈때마다 몇억건을 넣어?
weight의 업데이트 = $- \gamma \nabla F(\mathbf{a}^n)$
 γ (learning rate) $\nabla F(\mathbf{a}^n)$
(gradient)

뉴럴넷 loss(or cost) function을
가지고 있습니다. 쉽게 말하면 “틀린 정도”
내가 가진 weight 세팅(내 자리)에서,
내가 가진 데이터를 다 넣으면
전체 에러가 계산됩니다.

어느천년에 다하는가.
GD보다 빠른 옵티마이저는 없을까?

Stochastic Gradient Decent!!

SGD의 컨셉: 느린 완벽보다 조금만 훗어보고 일단 빨리 가봅시다.



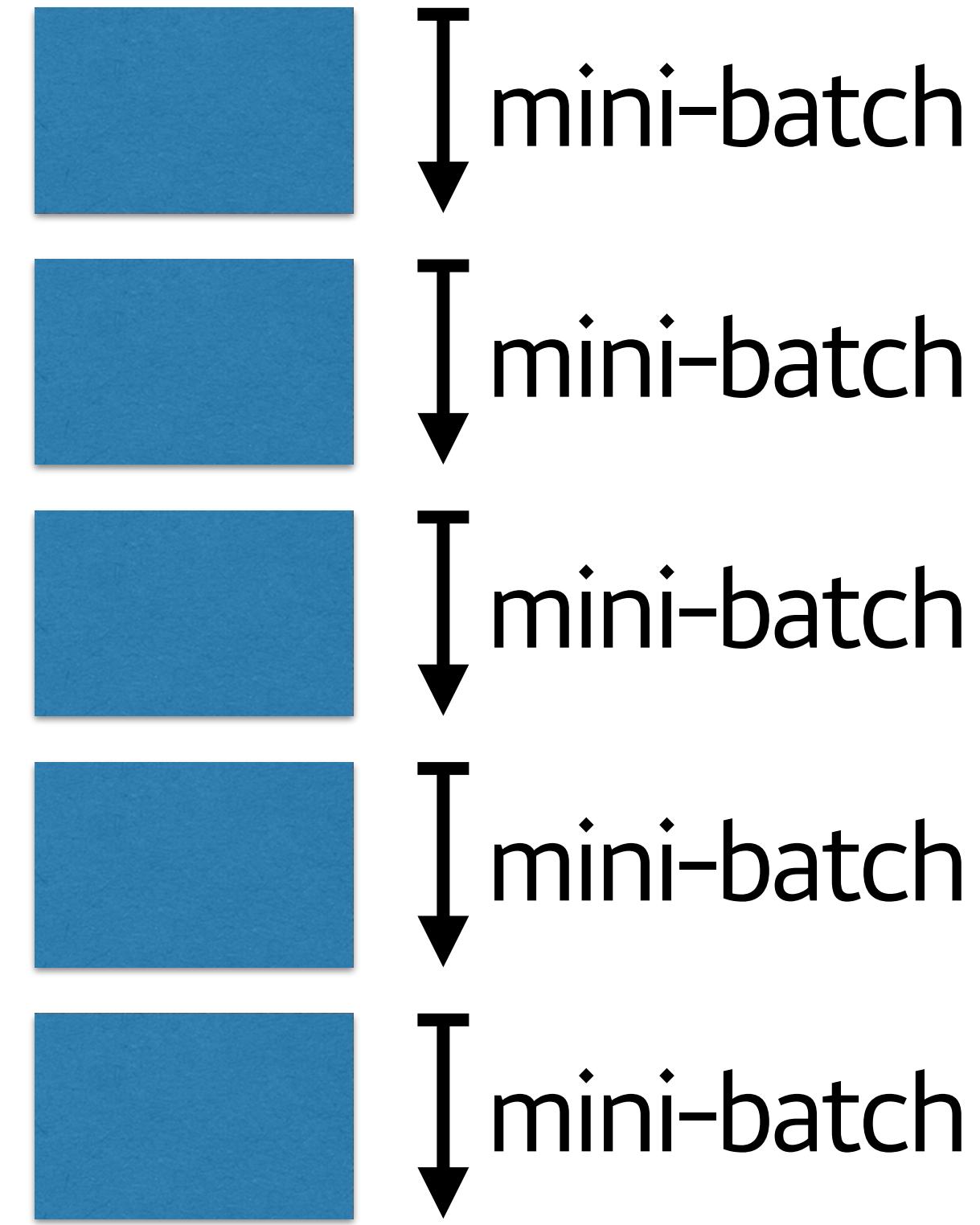
(이거랑 닮음. 하지만 done도 better도 perfect도 실패하는 우리네 삶..ㅠㅠ)

Gradient Decent

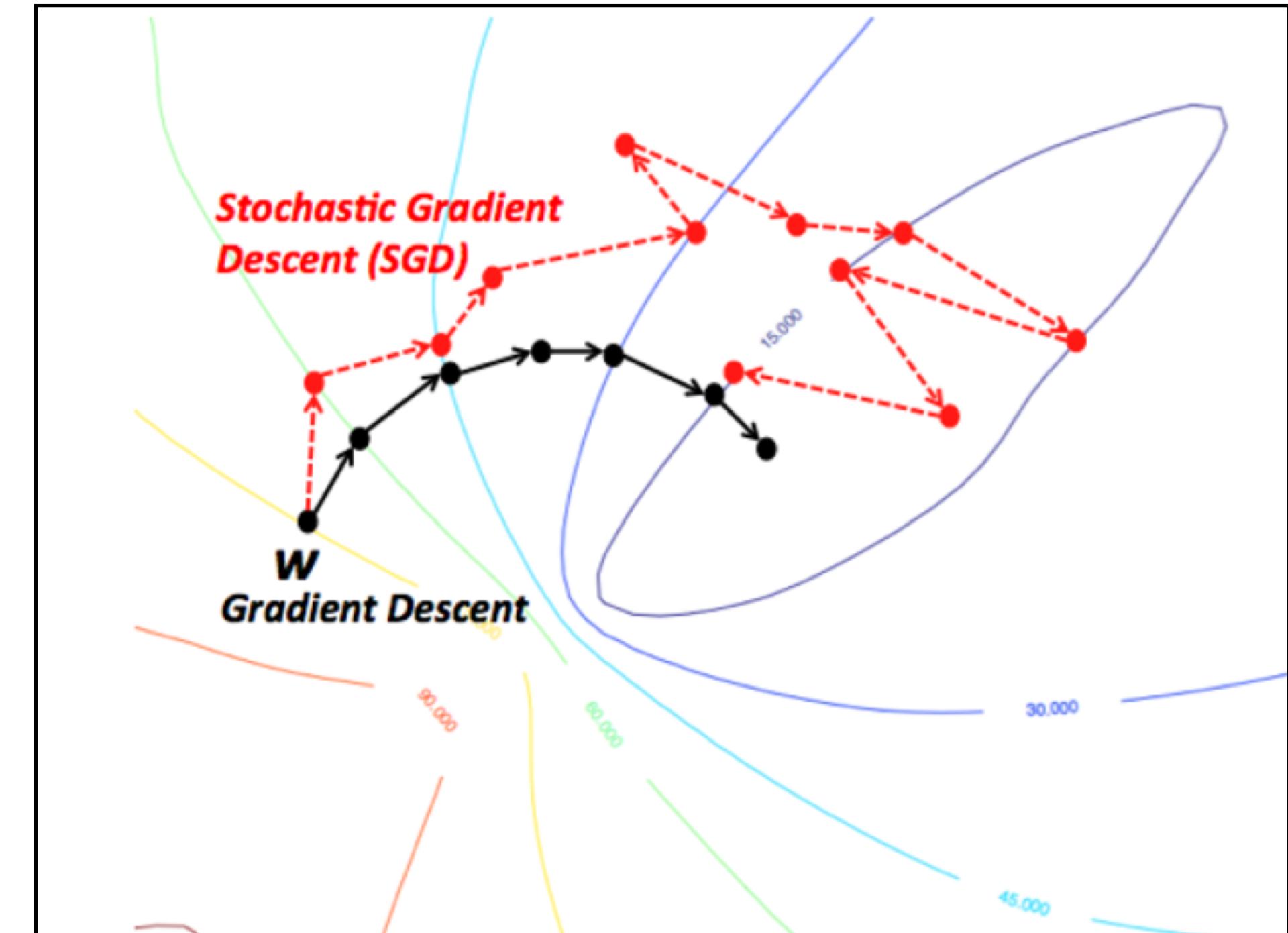


전부다 읽고나서
최적의 1스텝 간다.

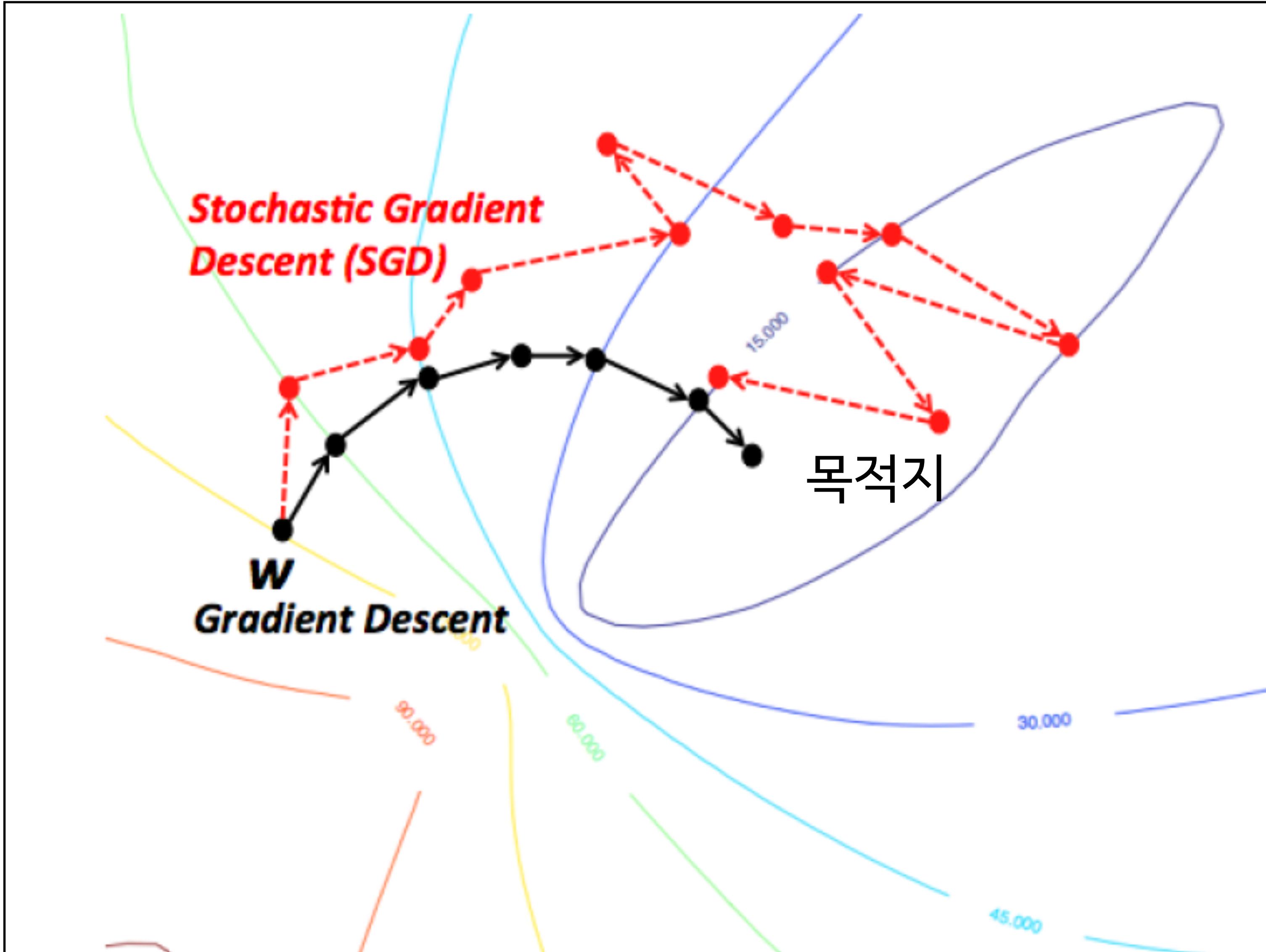
Stochastic Gradient Decent



작은 토막마다
일단 1스텝간다.



GD vs SGD



Gradient Decent

모든 걸 계산(1시간)후

최적의 한스텝

6스텝 * 1시간 = 6시간

최적인데 너무 느리다!

Stochastic Gradient Descent

일부만 검토(5분)

틀려도 일단 간다! 빠른 스텝!

11스텝 * 5분 = 55분 < 1시간

조금 헤매도 어쨌든 인근에
아주 빨리 갔다!

문제?

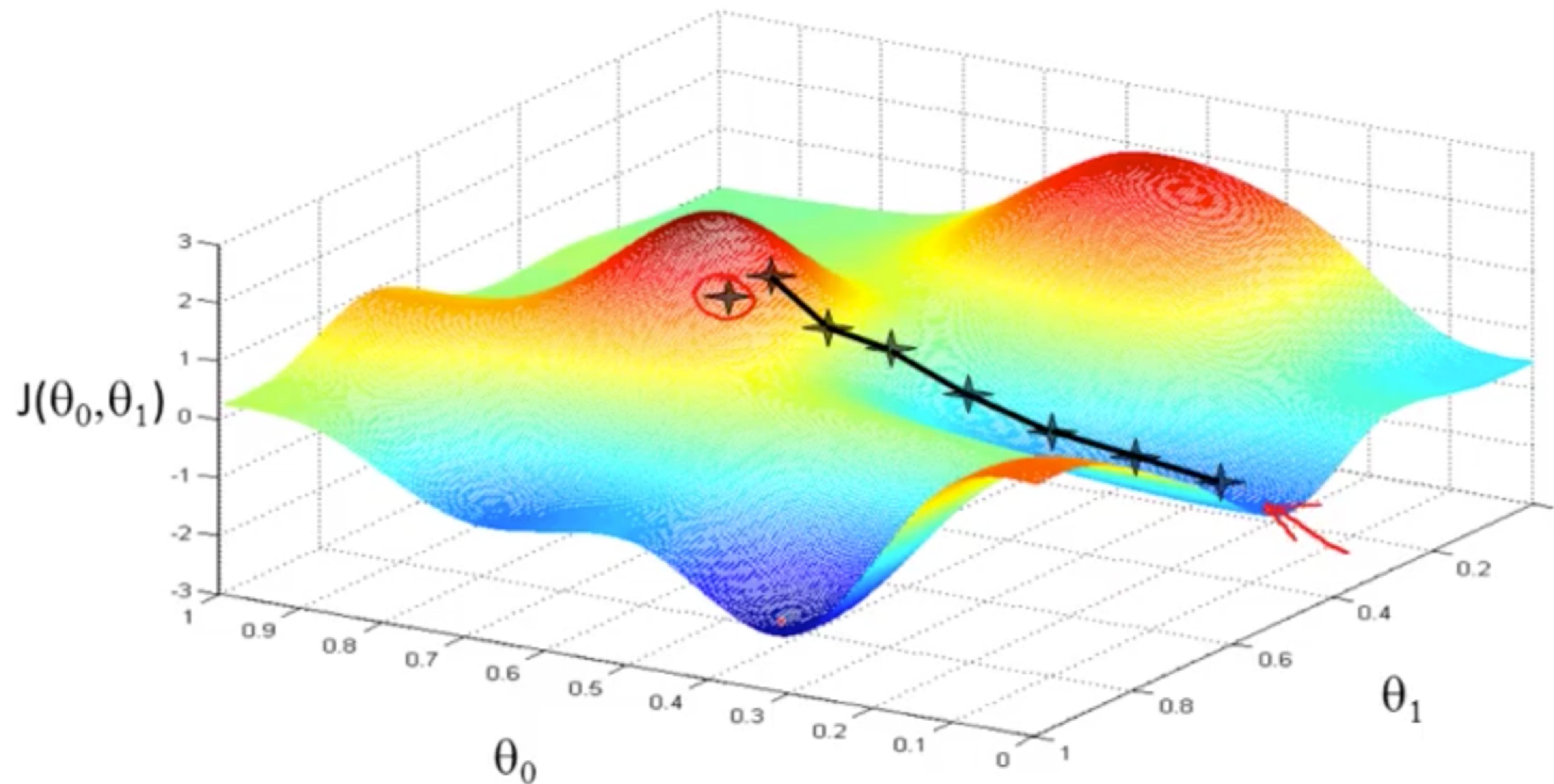
걸음마다 batch로 전부다 계산하려니
GD가 너무 오래 걸린다



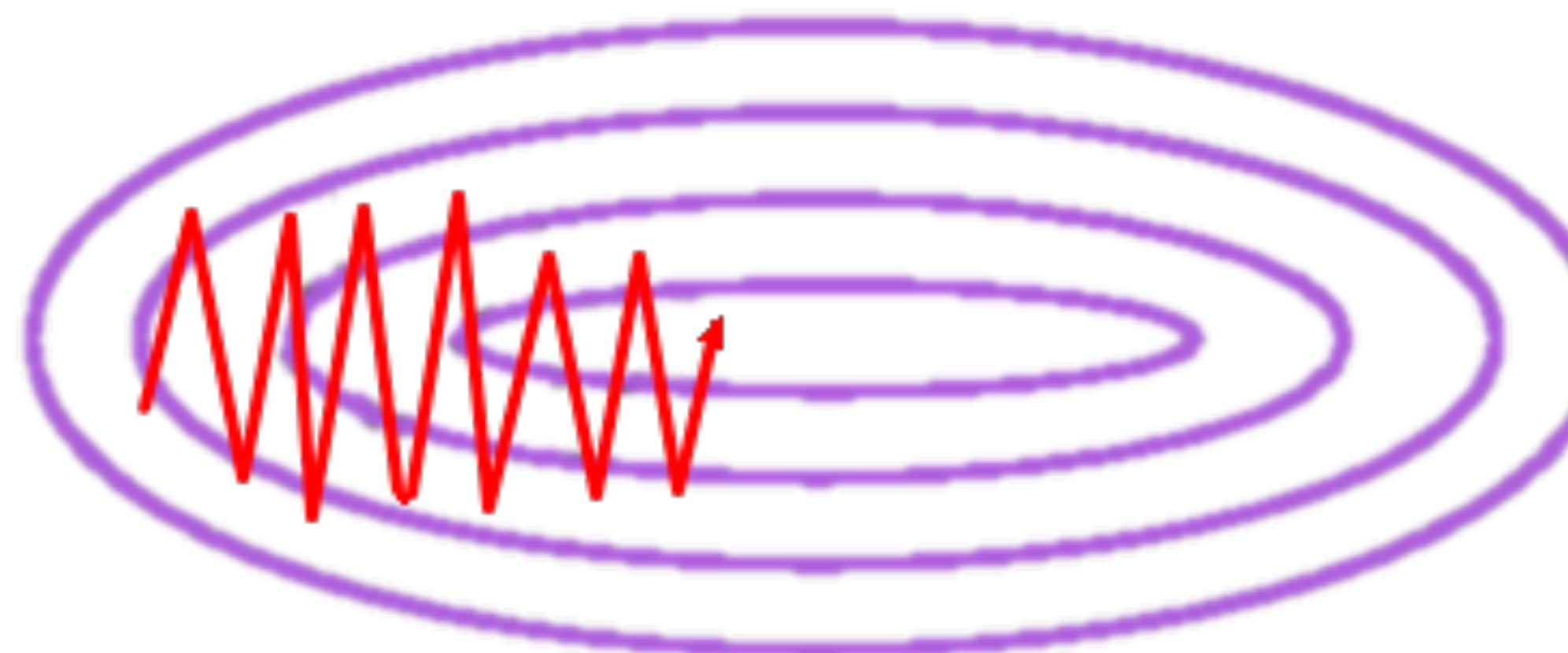
해결!

SGD로 mini-batch마다 움직여
같은 시간에 훨씬 더 많이 진행해서 해결!

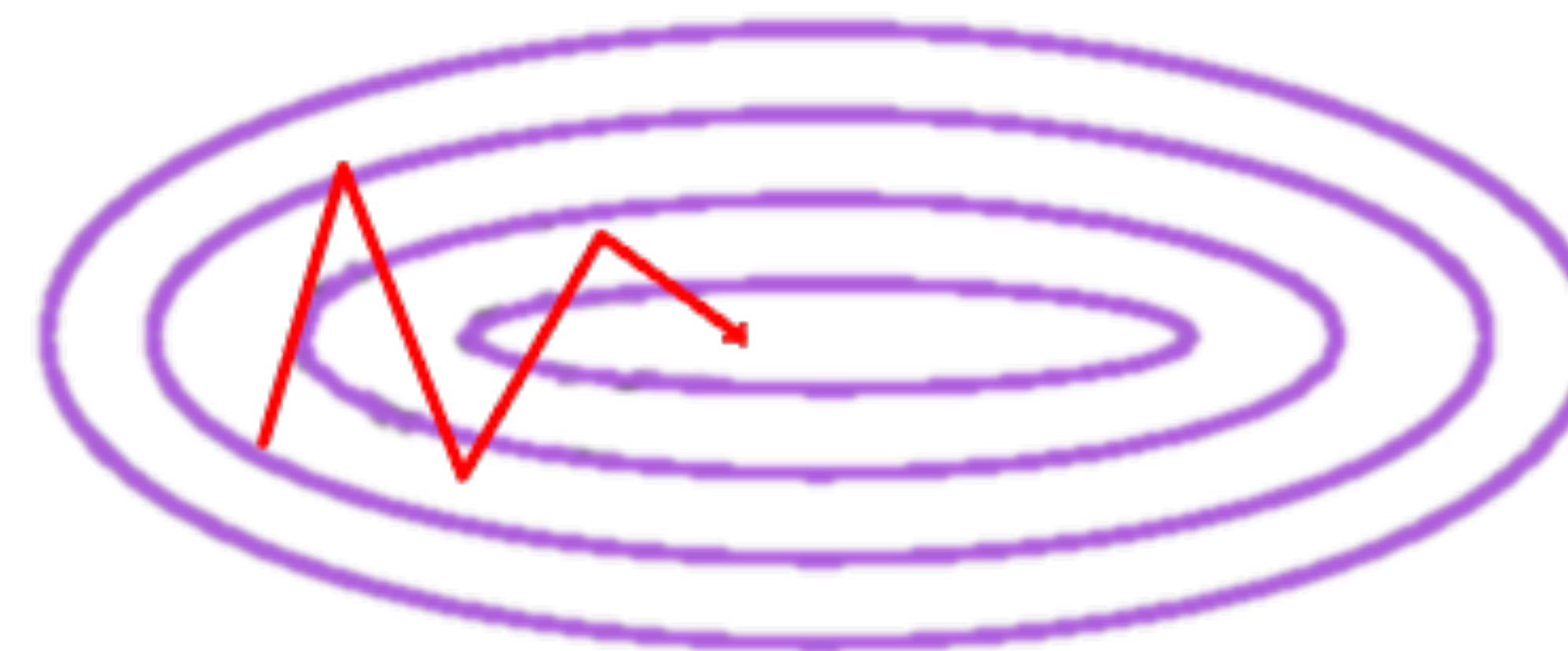
다시 생각해봐도 이건, 굴곡 많은 산을
좋은 오솔길을 찾아 잘 내려가는 일과 참 비슷



근데 미니배치를 하다보니 와리가리(?) 방향 문제가 있다.

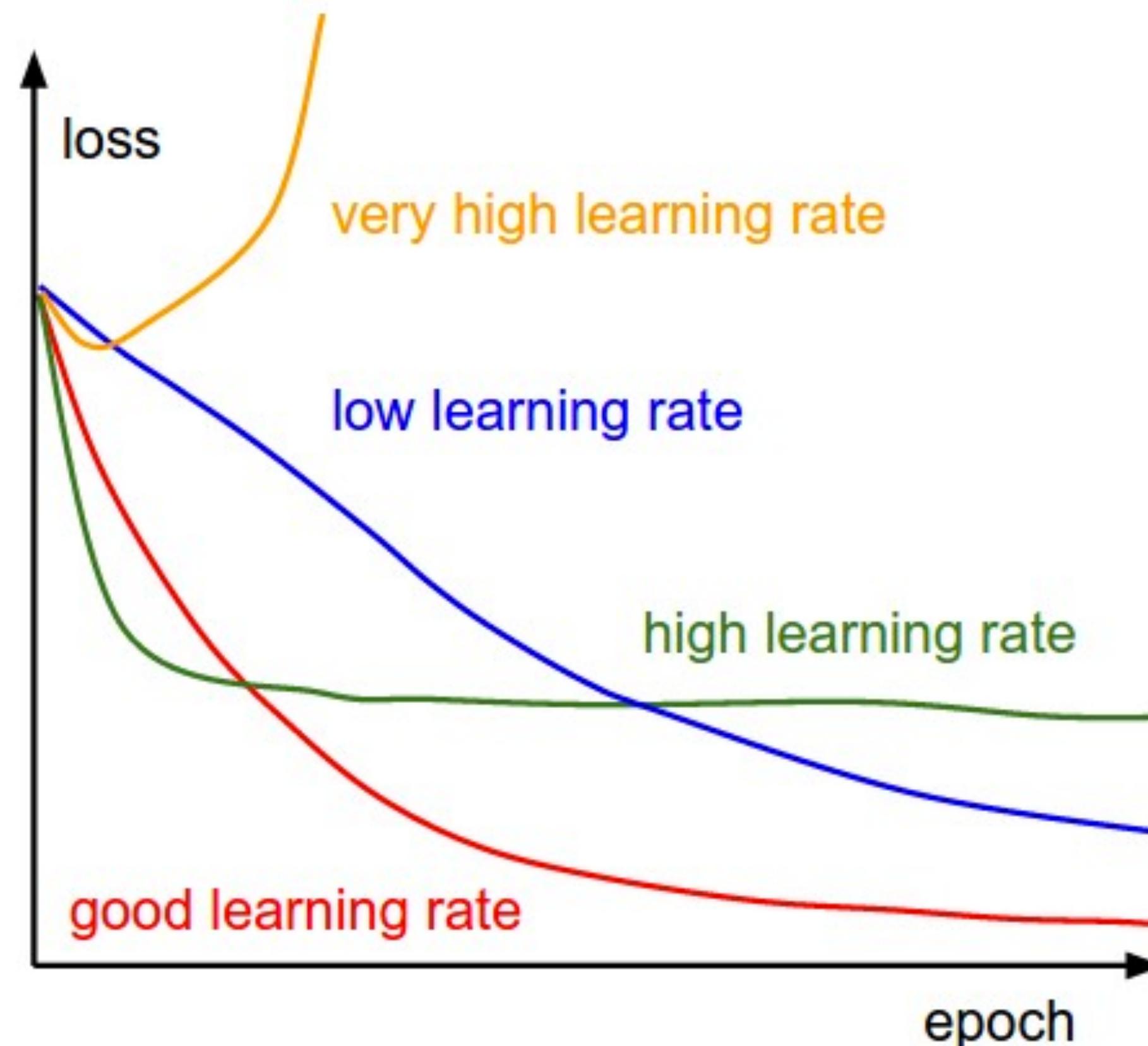


딱봐도 더 잘 갈 수 있는데
훨씬 더 헤매면서 간다.



훑기도 잘 훌으면서,
좀 더 휙휙 **더 좋은 방향**으로 갈 순 없을까?

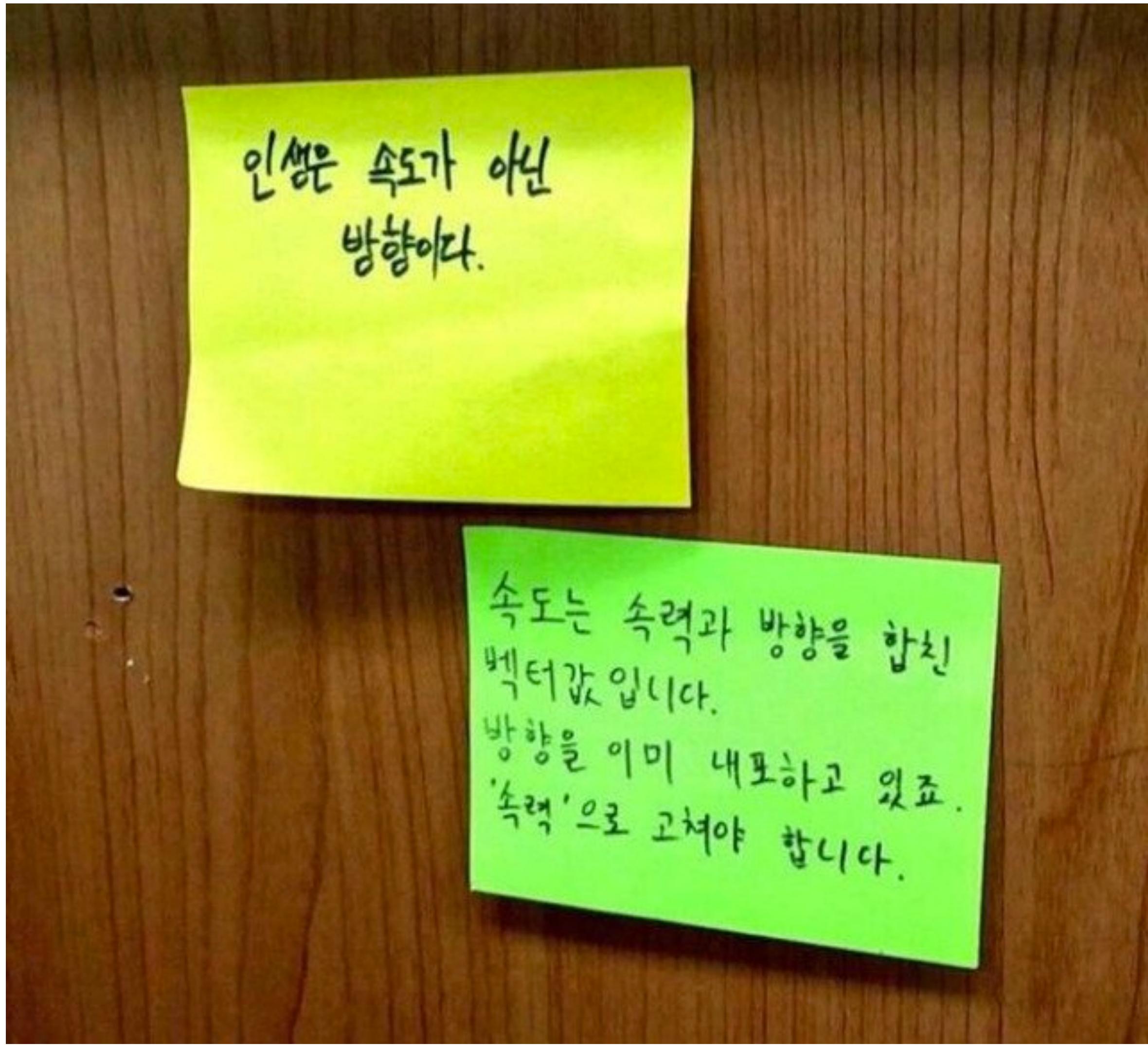
스텝사이즈(learning rate)도 문제가 된다.



보폭이 너무 작으면 오래 헤매고(파란라인)

보폭이 너무 크면, 오솔길을 지나친다(녹색라인)

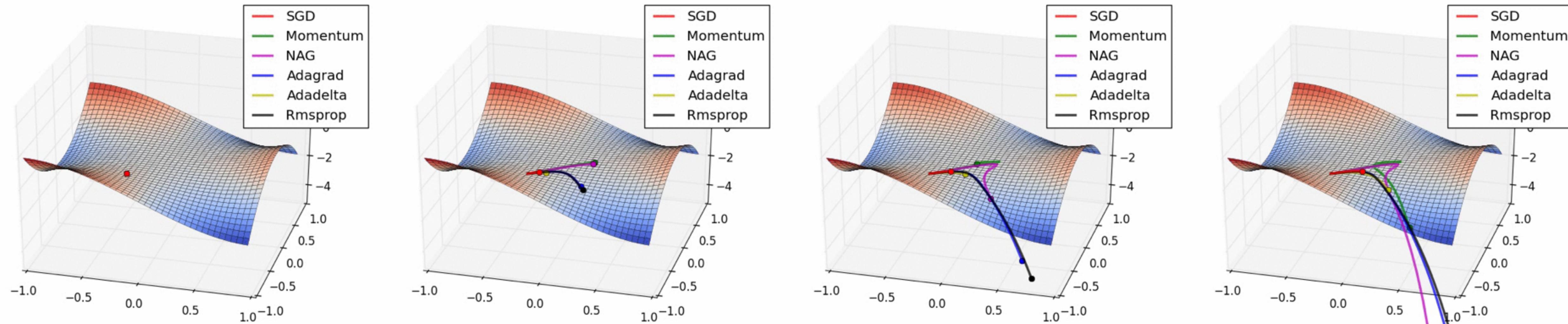
이과가 또.jpg



$-\gamma \nabla F(\mathbf{a}^n)$ 산을 잘 타고 내려오는 것은
 $\nabla F(\mathbf{a}^n)$ 어느 방향으로 발을 디딜지
 γ 얼마 보폭으로 발을 디딜지
두 가지를 잘 잡아야 빠르게 타고 내려온다.

SGD를 더 개선한 멋진 optimizer 많다!
SGD의 개선된 후계자들

여러가지 방법이 있다!



<http://imgur.com/NKsFHJb>

산내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보

모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.

GD

스텝방향

SGD

전부 다봐야 한걸음은
너무 오래 걸리니까
조금만 보고 빨리 판단한다
같은 시간에 더 많이 간다

Momentum

스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자

Nesterov Accelerated Gradient

NAG

일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라

Nadam

Adam에 Momentum
대신 NAG를 붙이자.

Adam

RMSProp + Momentum
방향도 스텝사이즈도 적절하게!

RMSProp

보폭을 줄이는 건 좋은데
이전 맥락 상황 봐가며 하자.

Adagrad

안가본 곳은 성큼 빠르게 걸어 훑고
많이 가본 곳은 잘아니까
갈수록 보폭을 줄여 세밀히 탐색

AdaDelta

종종 걸음 너무 작아져서
정지하는 걸 막아보자.

잘 모르겠으면 Adam!

문제?

SGD가 빠른데 좀 헤맨다



해결!

SGD의 개선된 버전을 골라
더 빠르고 더 정확하게!

덜하거나
Underfitting

학습이 잘 안돼!

느리거나
Slow

도대체 학습은 언제 끝나는 건가?

과하거나
Overfitting

겨우 되어도 융통성이 없다?!

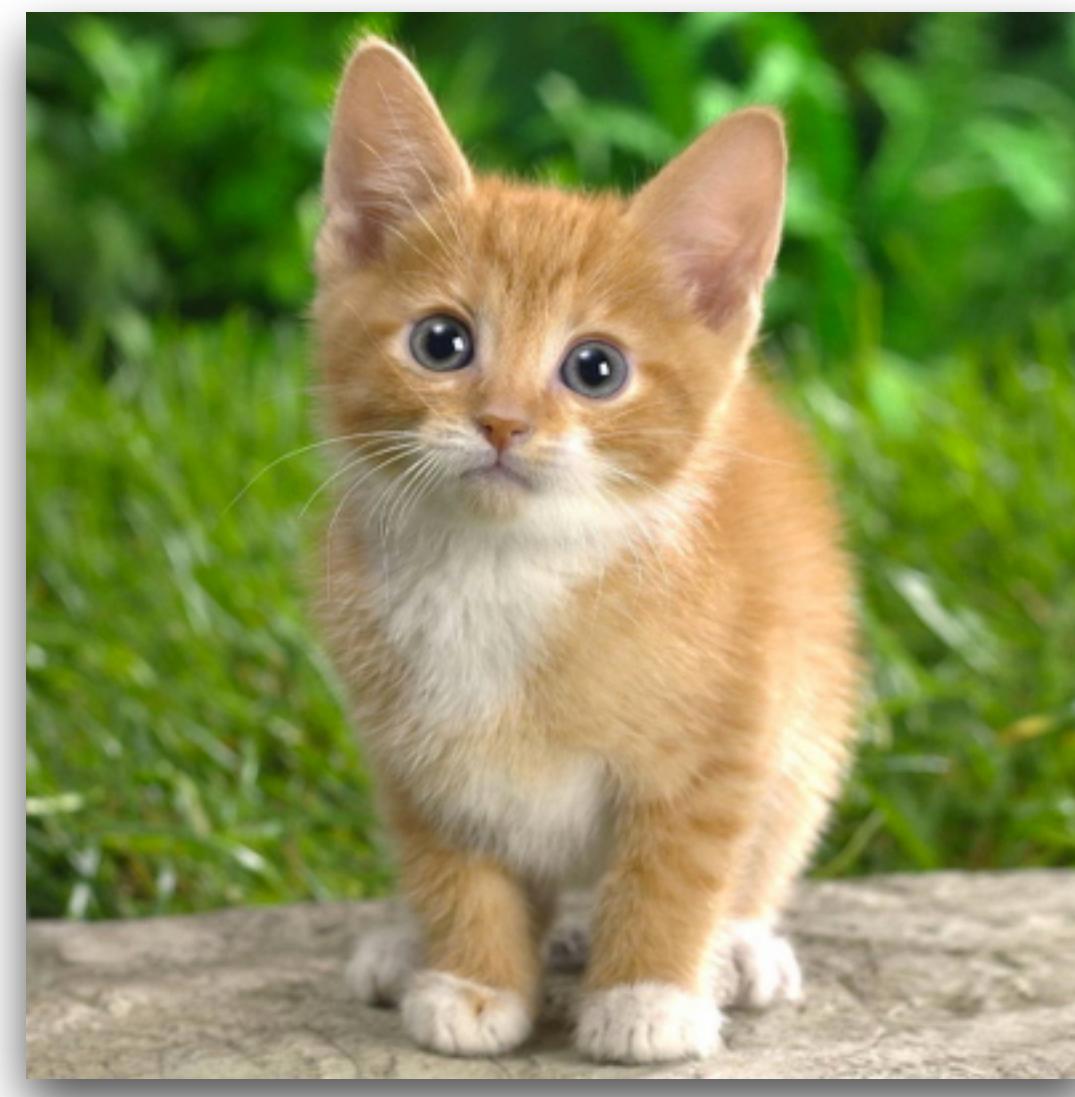
열심히 뉴럴넷에게 고양이



를 가르쳤더니..



뚱뚱하니까 고양이 아님



갈색이니까 고양이 아님



귀처졌으니까 고양이 아님

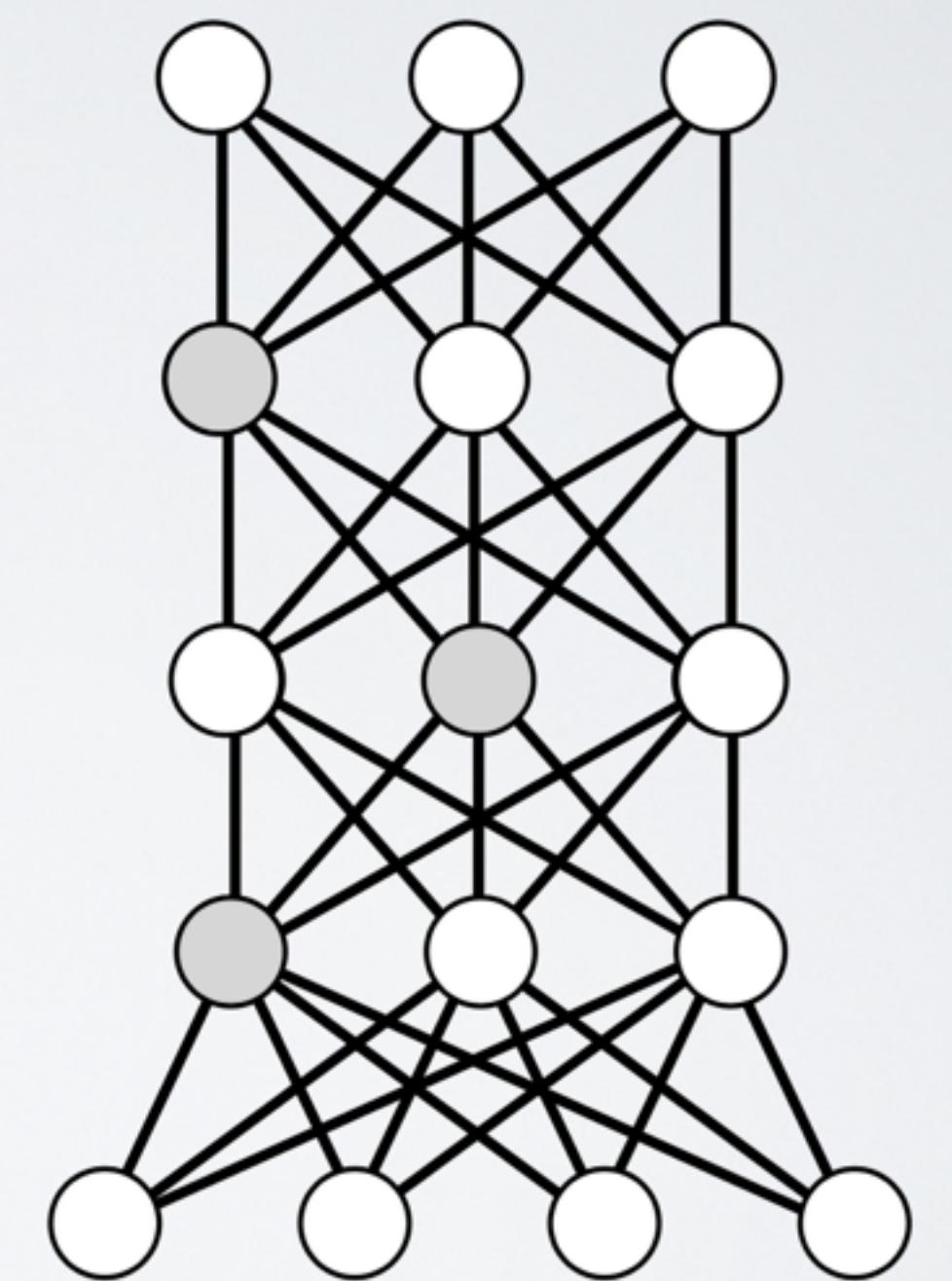
융통성이라곤 눈꼽만큼도 없다!

Overfitting

뉴럴넷에게 융통성을 기르는 방법은? 가르칠 때부터, 좀 가리면서 가르친다!

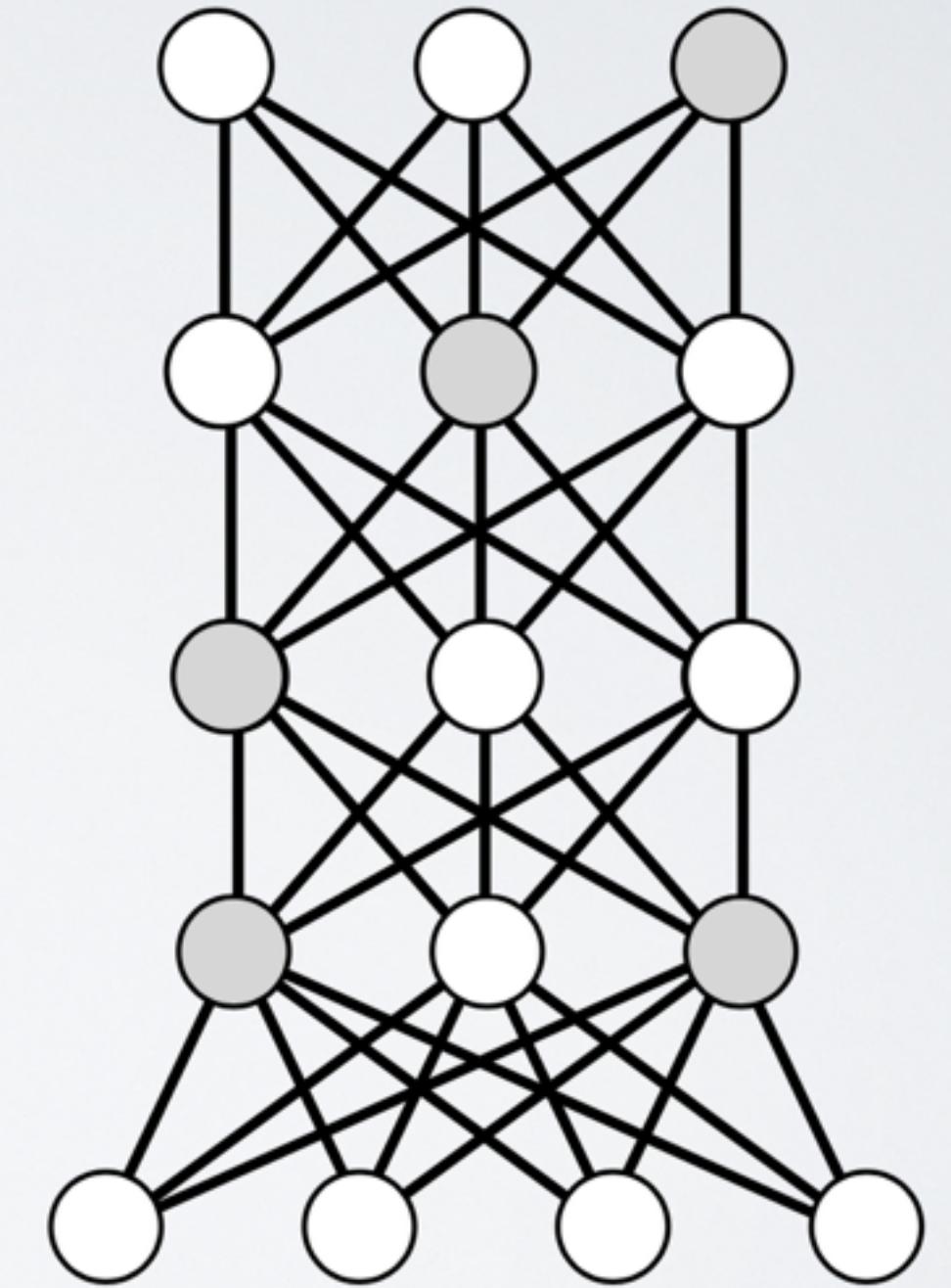
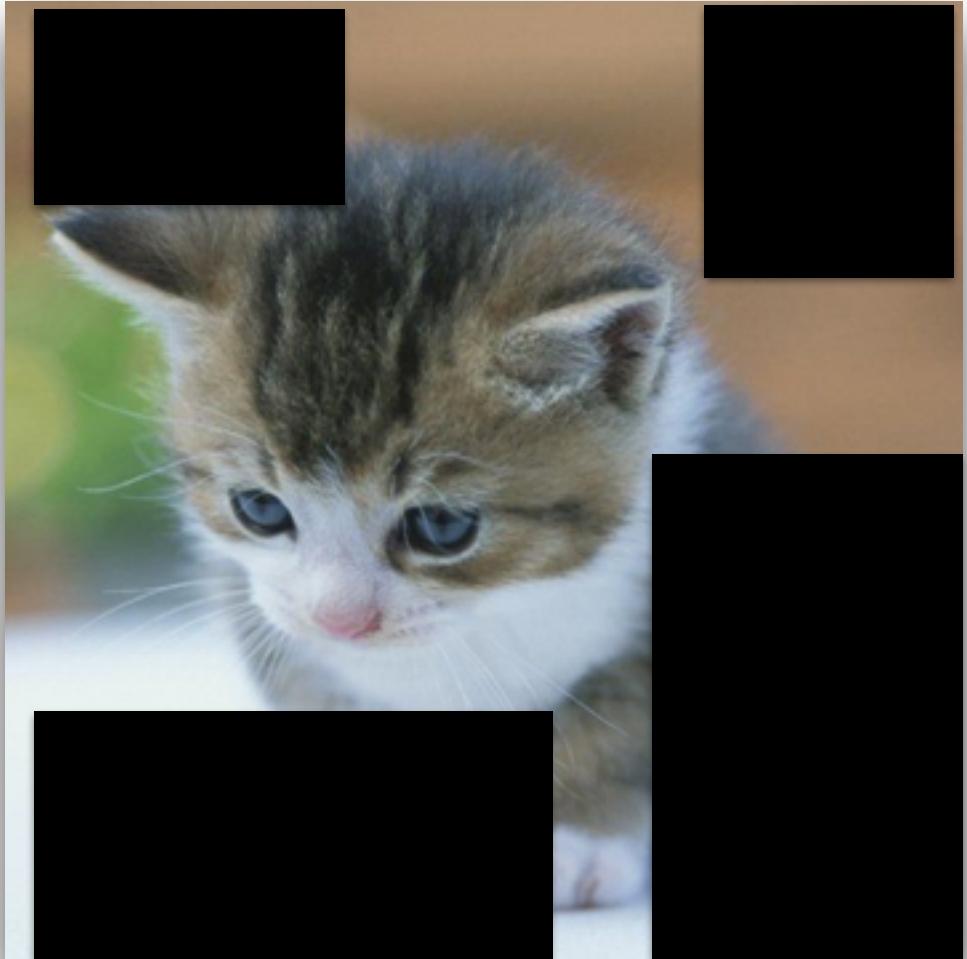
DropOut!

**학습 시킬 때,
일부러 정보를 누락시키거나
중간 중간 노드를 끈다.**



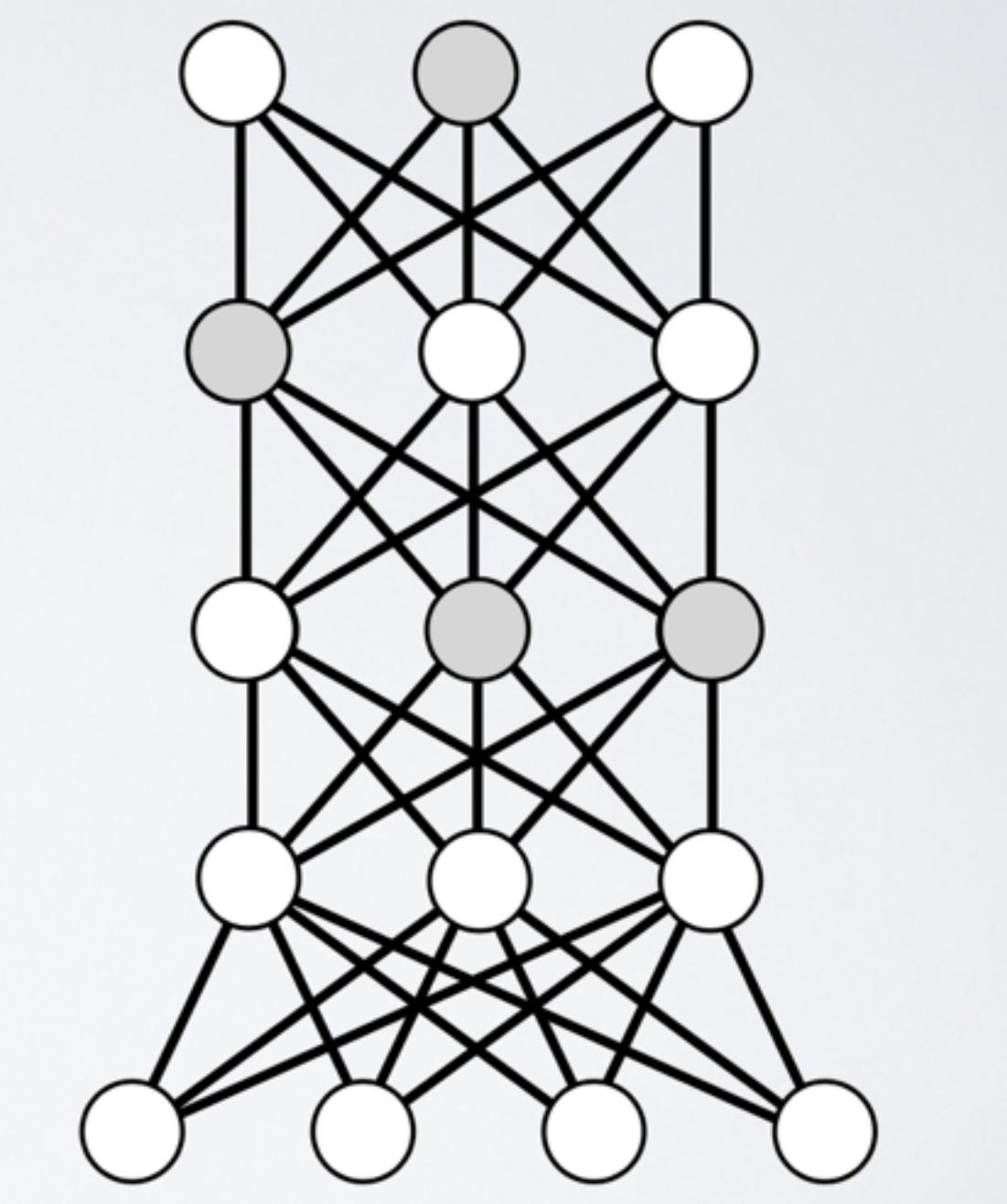
○ :units turned off

얼굴위주



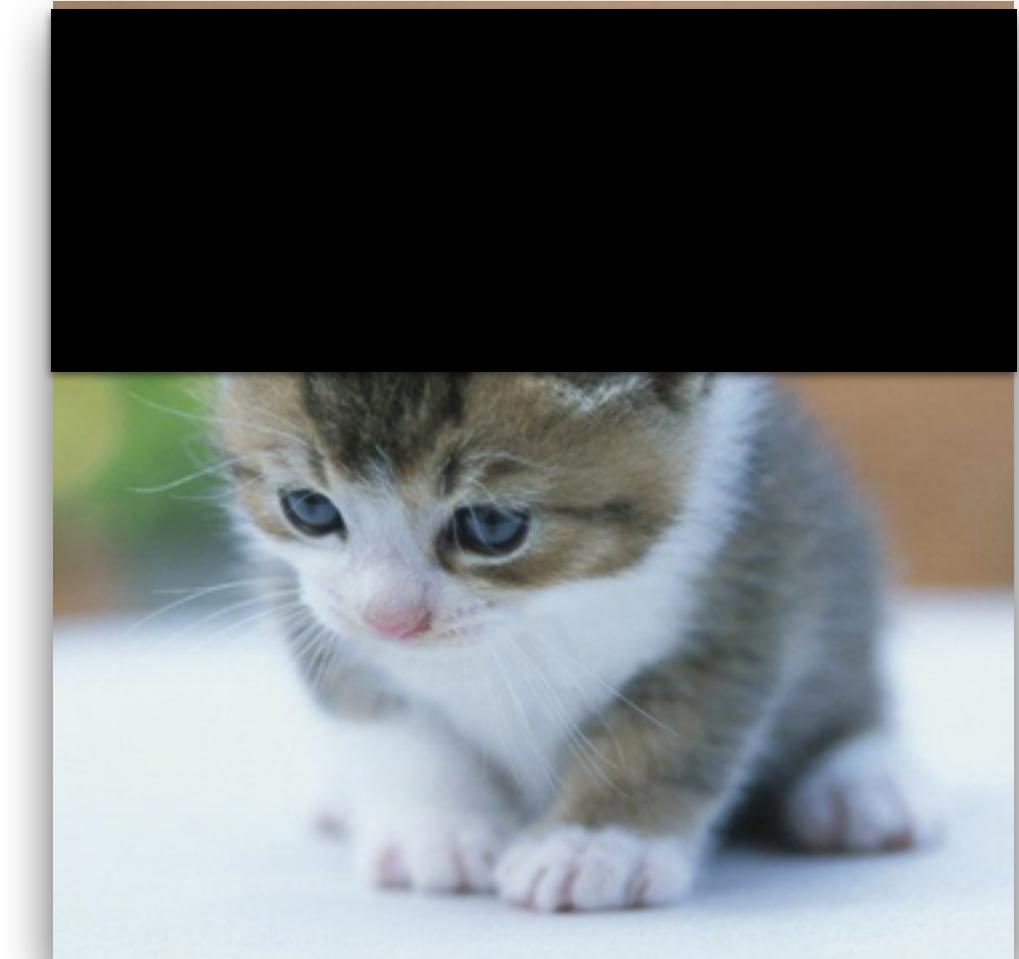
○ :units turned off

색지우고



○ :units turned off

귀 빼고



**dropout으로
일부에 집착하지 않고
중요한 요소가 무엇인지
터득해 나간다.**

문제?

과적합으로 융통성이 없다.



해결!

DropOut으로
유연성을 획득시킨다.

문제 해결 완료!

덜하거나

Underfitting

학습잘되고

느리거나

Slow

이젠 빠르고

과하거나

Overfitting

융통성도 생겼다!

딥러닝을 가능하게 한 개념들을 배웠다!

ReLU

SGD

RmsProp

mini-batch

DropOut

Adam

Adam

Vanishing gradient

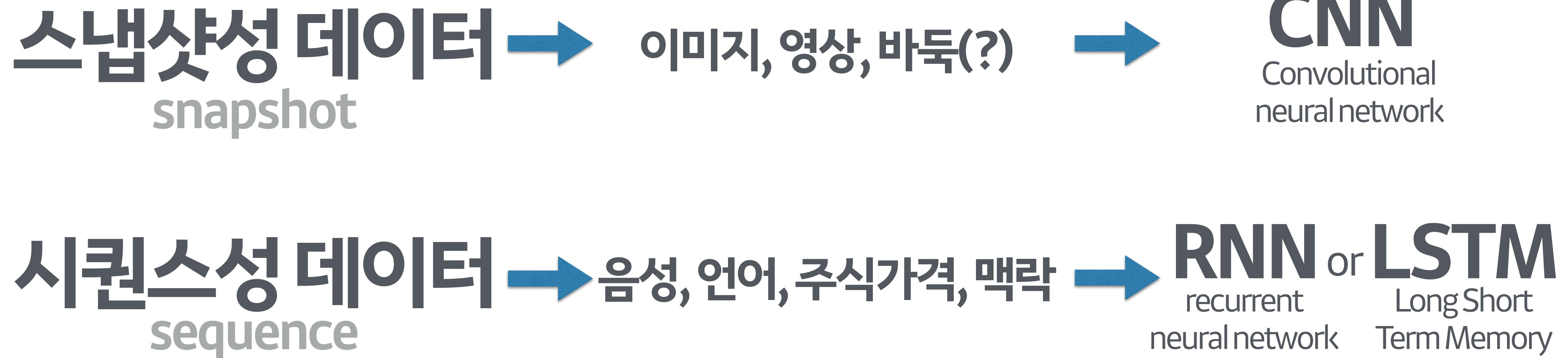
LearningRate

등등등~
000~

깊은 뉴럴넷의 학습이 가능해졌다.

이제 이것들을 잘 쌍아서 일을 해보자.

문제의 유형에 따라 적절한 아키텍쳐를!



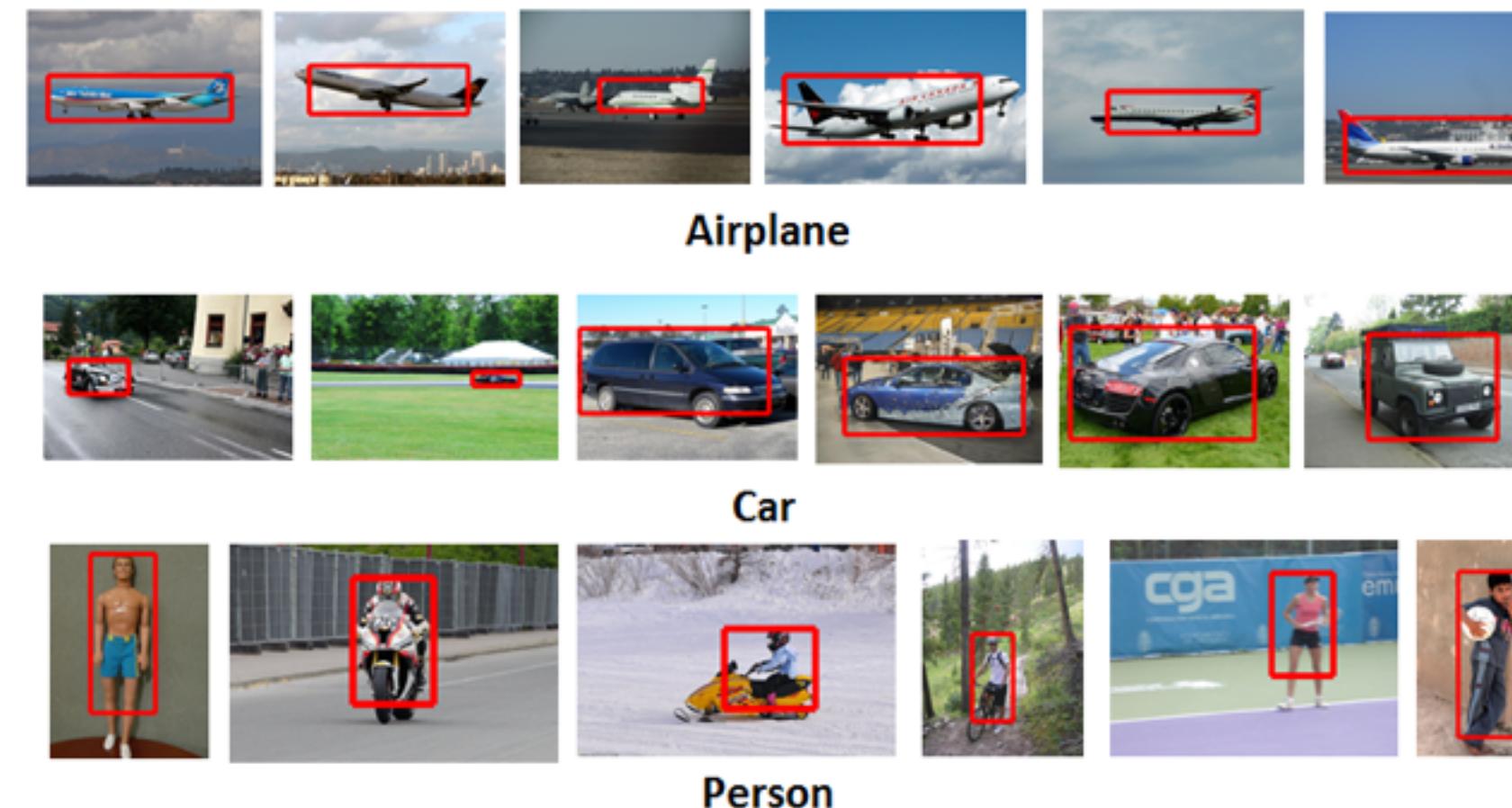
아까 봤던 VGG도
유명한 CNN구조 중의 하나.

이바닥에 유명한 녀석으로
AlexNet(구형),
VGG(인기 많고 많이 씀, 그냥 믿고 쓰는 허브 솔트 같은 느낌),
GoogleNet(첨에 인기 없고, 뒤에 Inception 버전업되고 좀 씀),
ResNet(레이어 짱 많음, 최근 많이 쓰임)

이런거 다 어디서 나오나요?

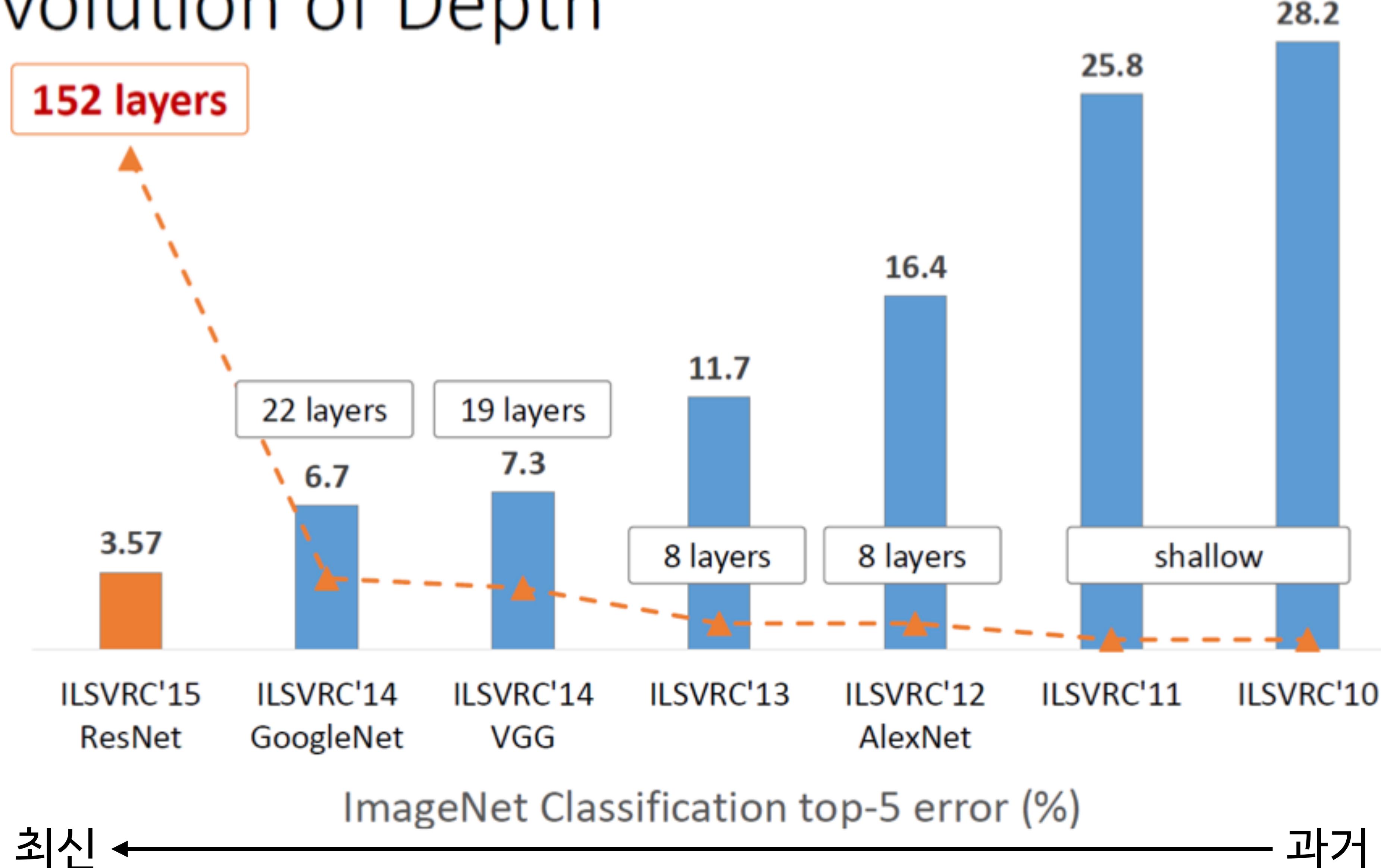


영상인식 관련 천하제일 무술 대회가 있습니다.
The ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
길어서 보통 ImageNet 대회라고 합니다. ㅎㅎ



연구자들이 자웅을 겨루며, 좋은 구조가 많이 나왔습니다.

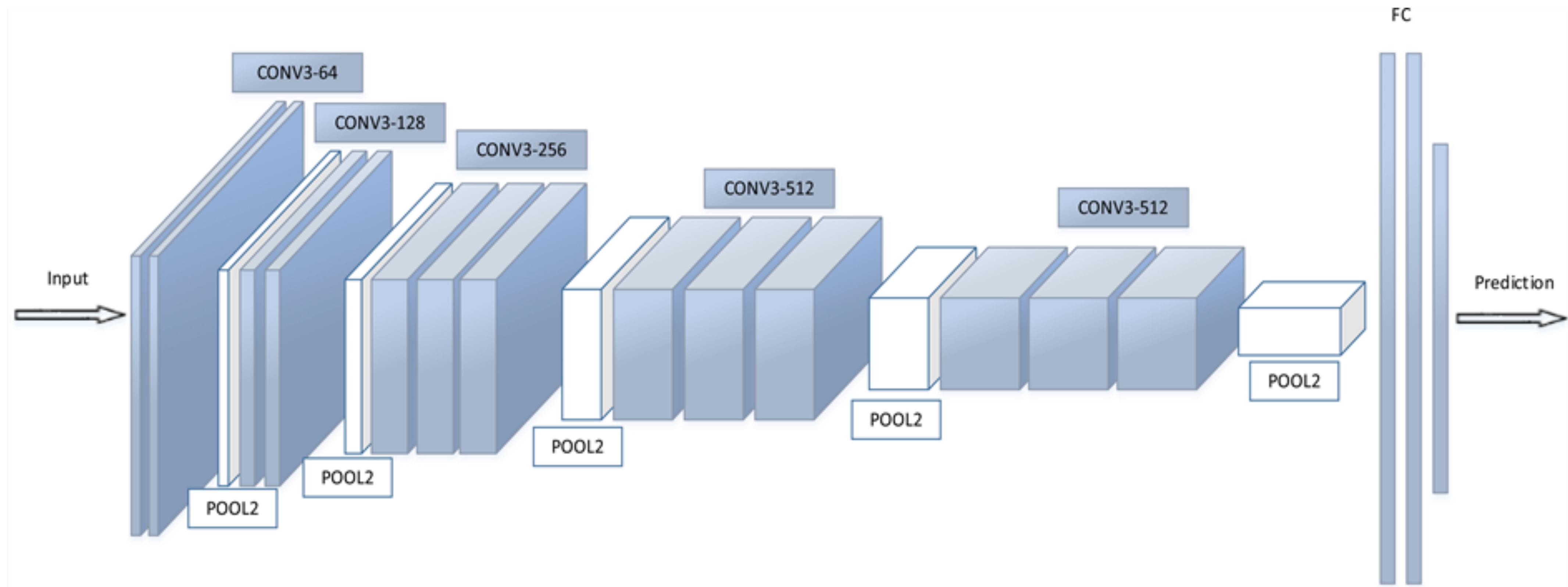
Revolution of Depth



**VGG는 구조가 직관적이고
성능도 상당히 쓸만해서
대중적으로 많이 쓰입니다.**

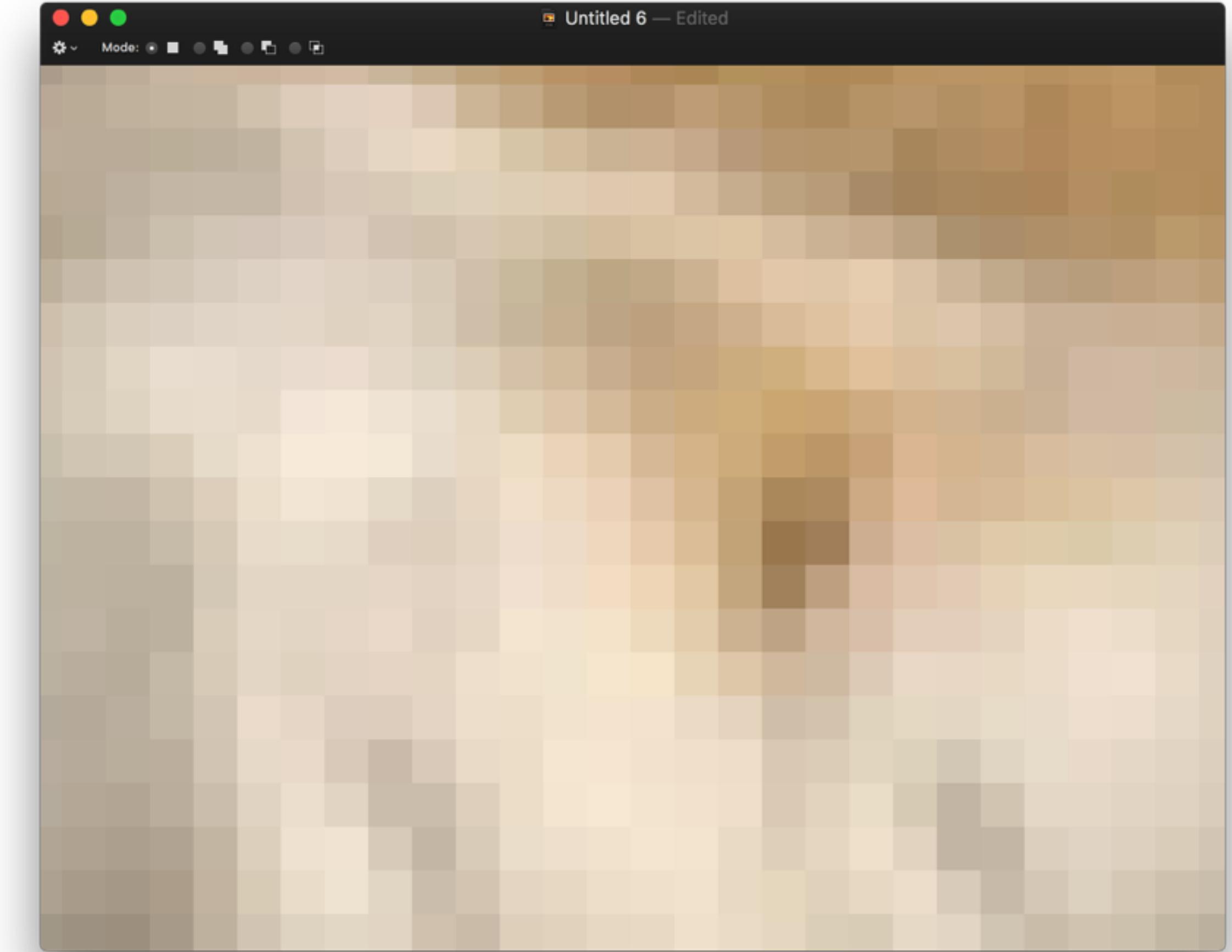
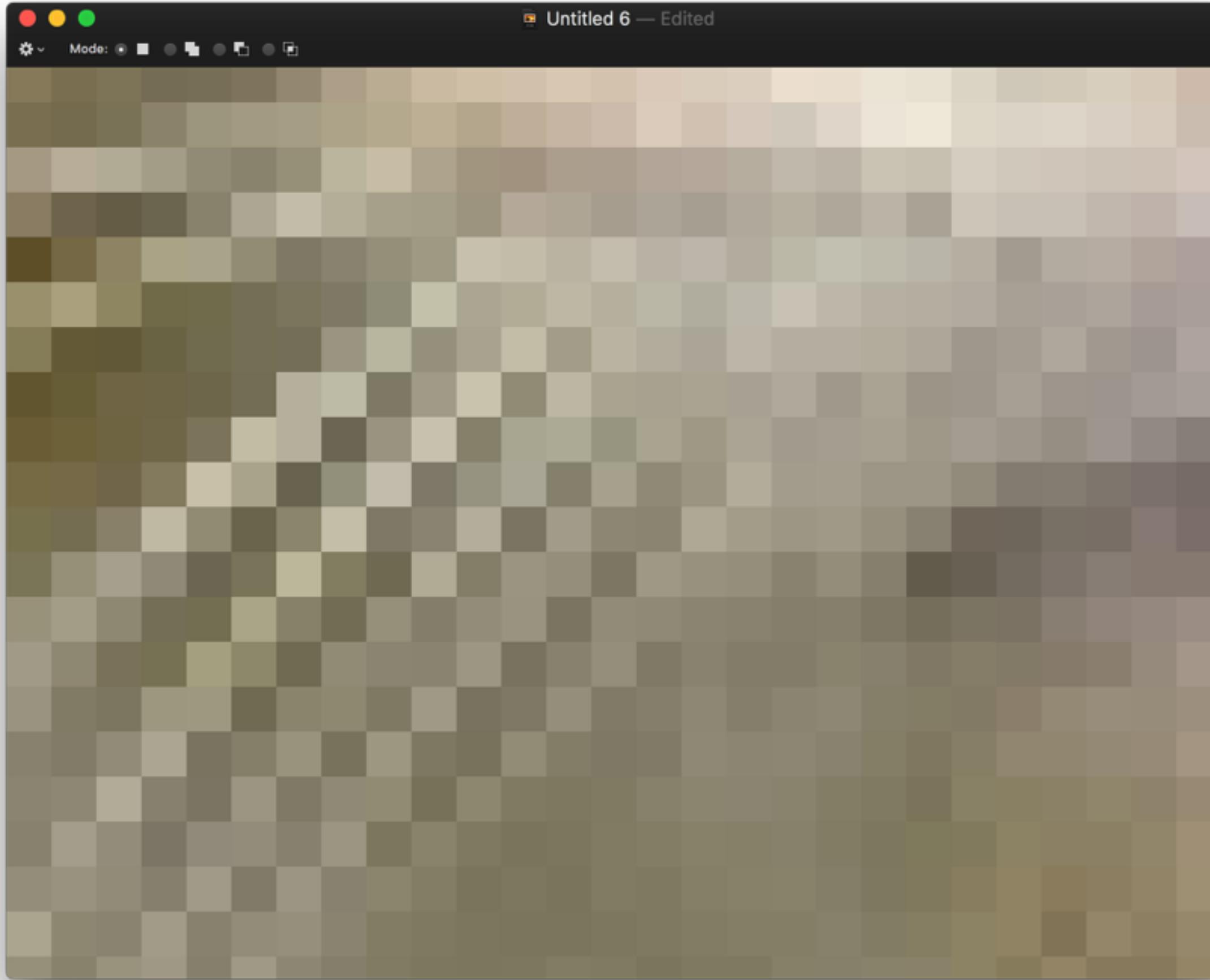
(다른 많은 구조의 재료가 되곤 합니다)

VGGNET의 구조



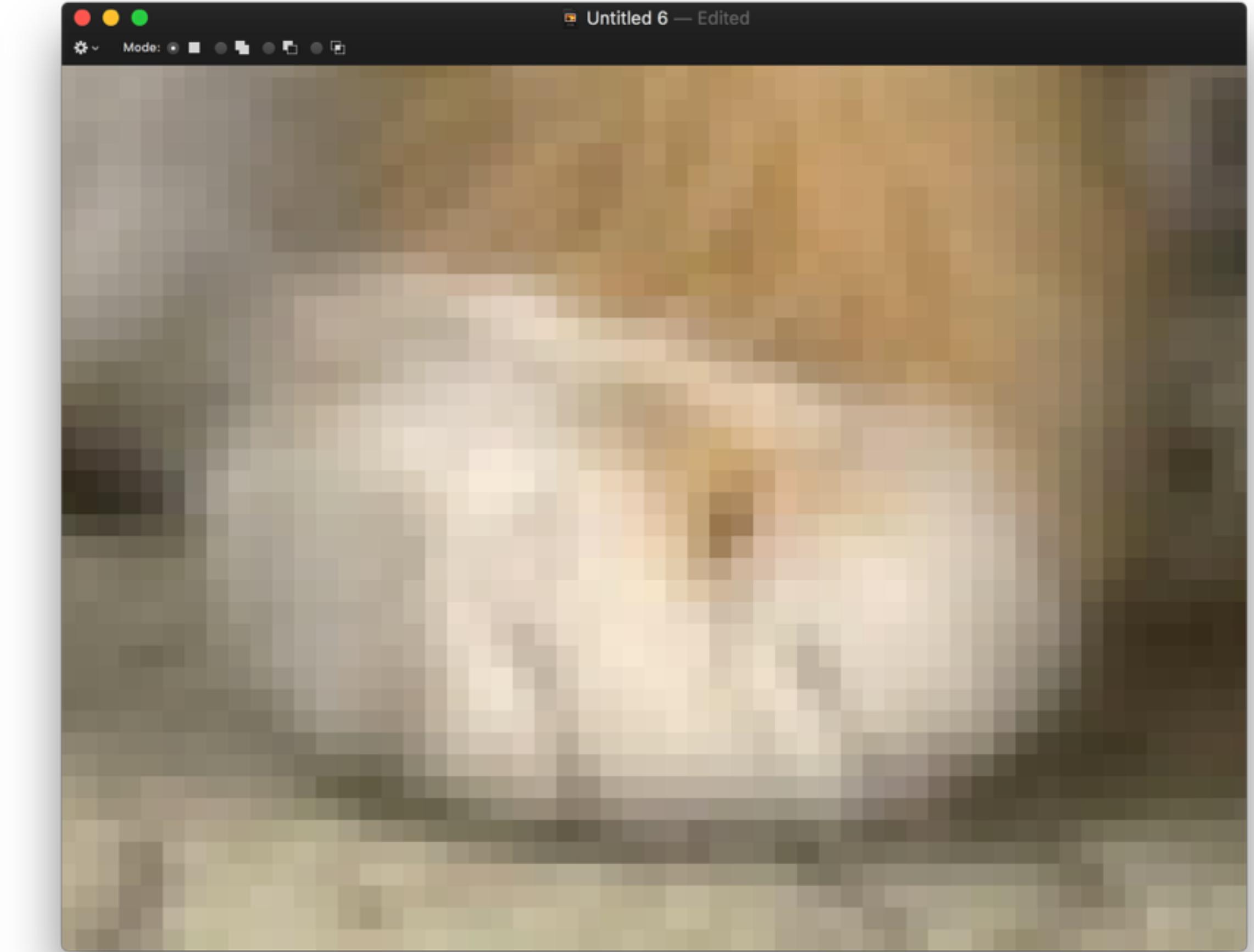
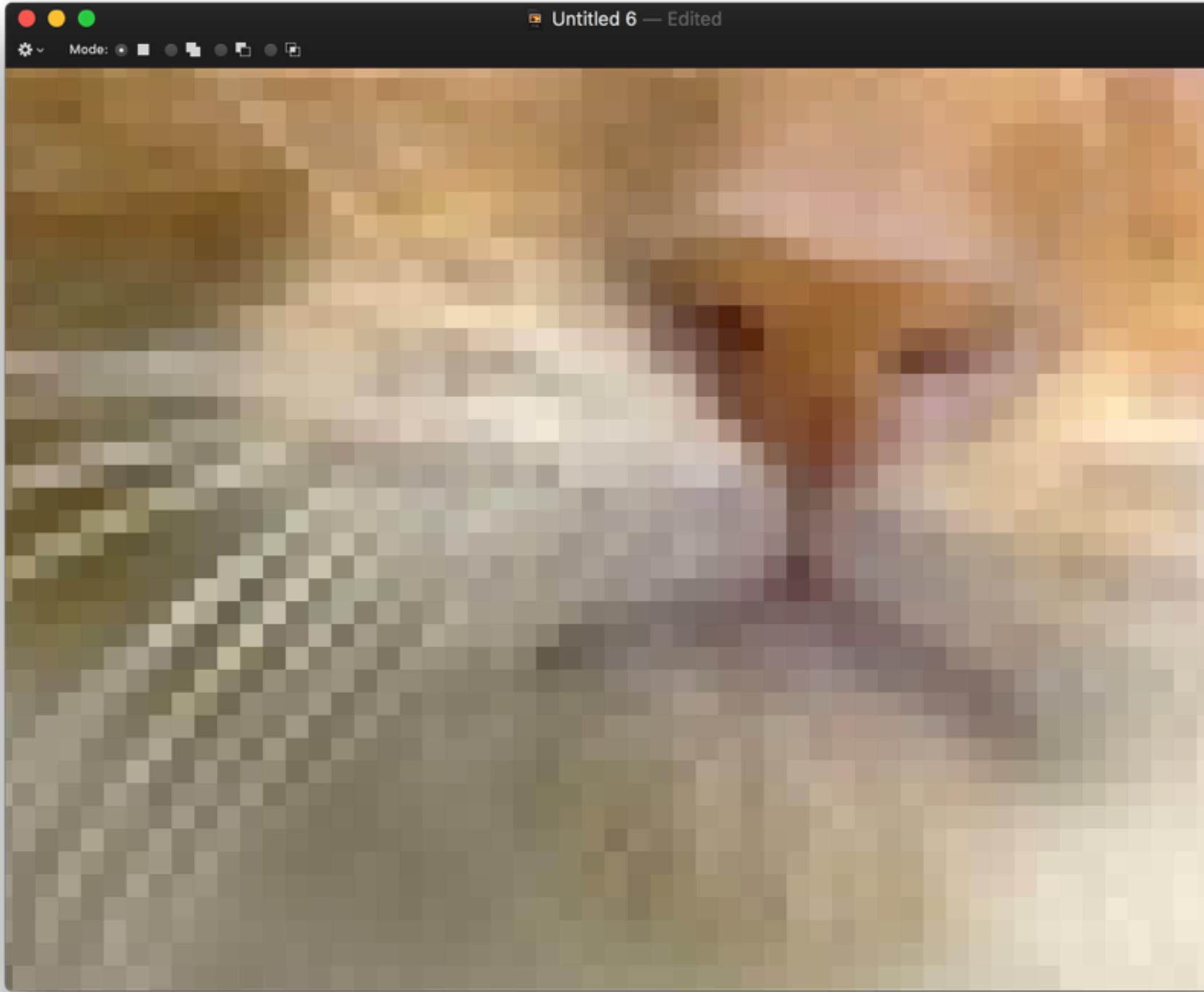
이해를 하기 위해, 우리가 그림을 본다고 상상해봅시다.

그림을 눈앞 1cm거리에서 본다고 생각해보자.



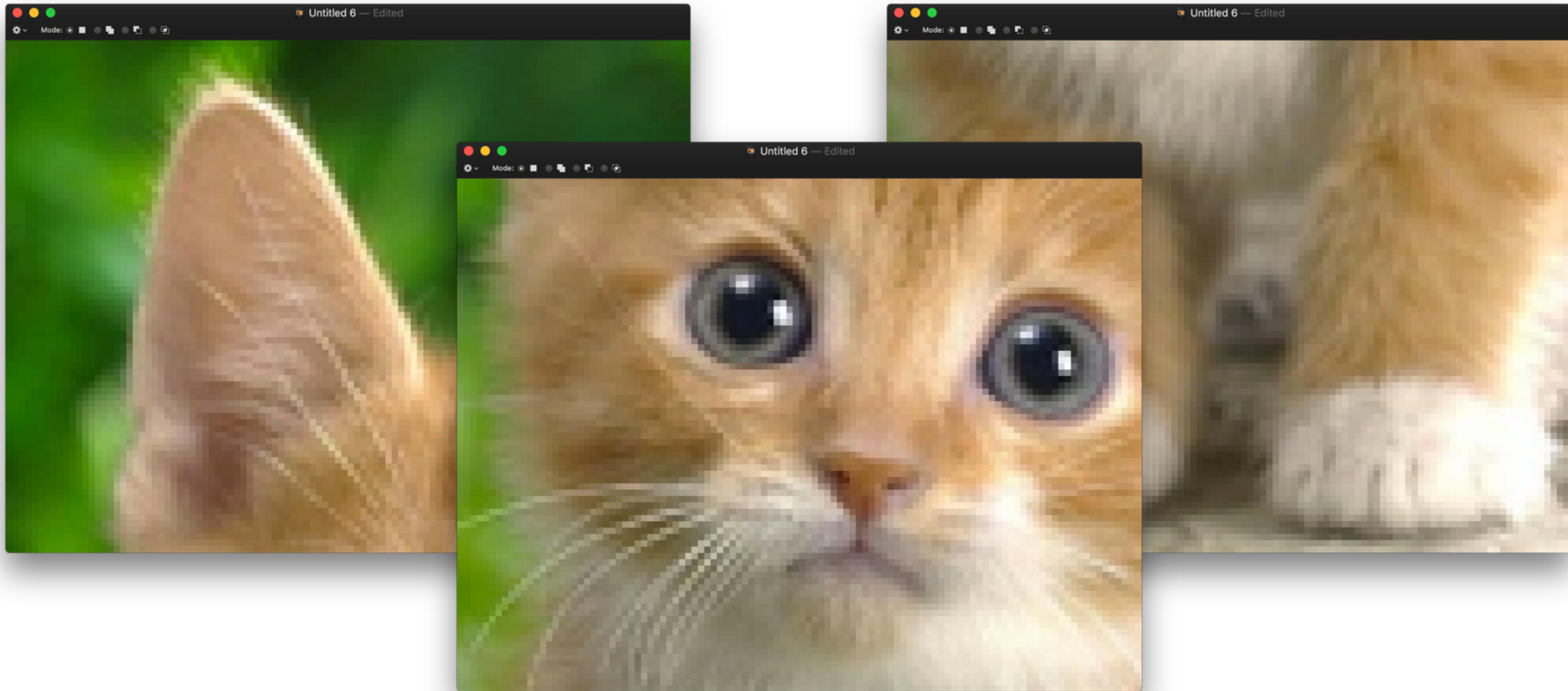
처음에는 점과 선, 이상한 질감 몇개 밖에 모르겠다.

점과 선, 질감을 충분히 배우고, 조금 떨어져서 보자.



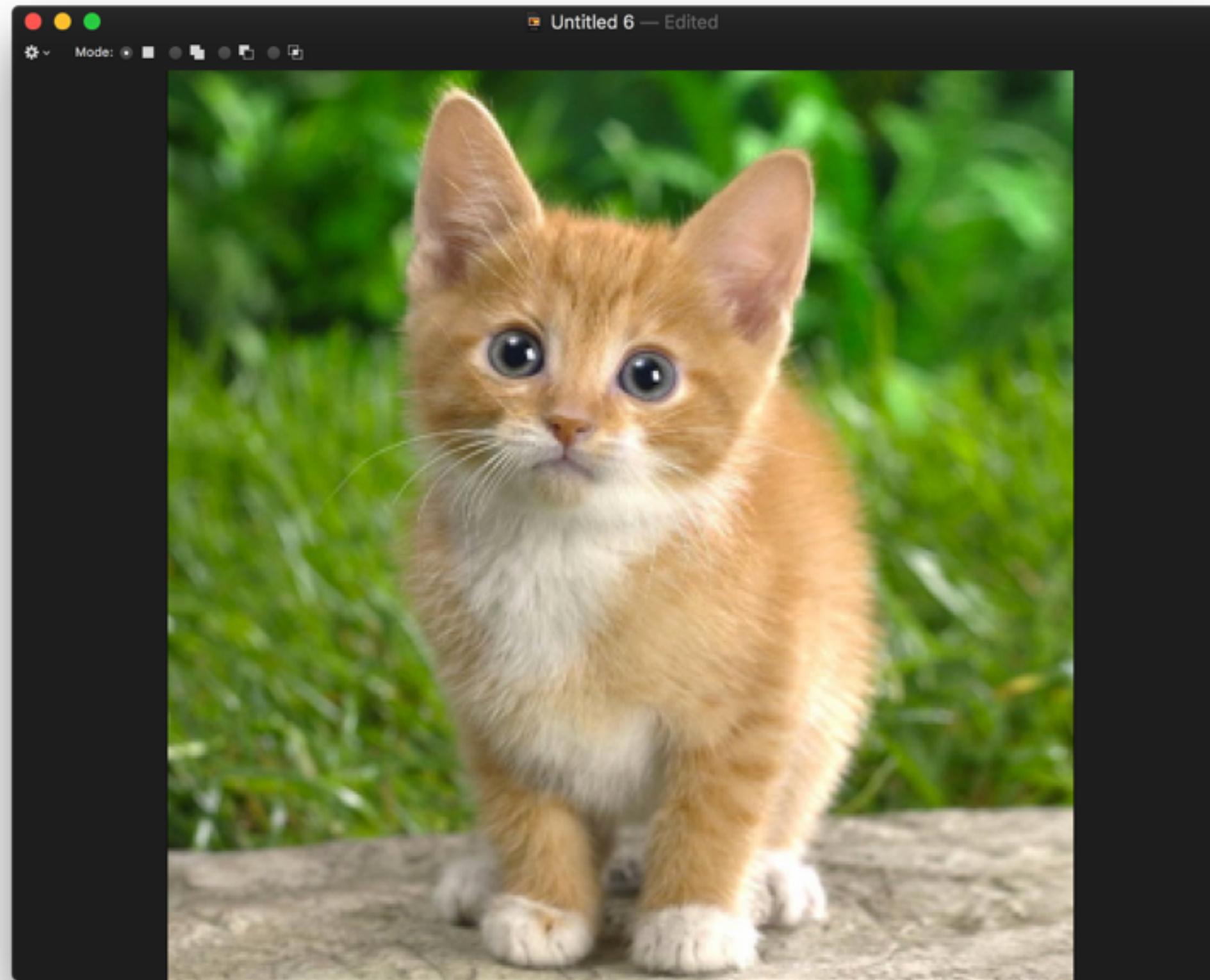
점과 선이 질감이 합쳐져 삼각형, 동그라미, 복실함이 보인다.

삼각형, 원, 사각형, 복실함등을 조합해서 보니



뾰족귀와 땡그란 눈과 복실한 발을 배웠다.

더 멀리서 보니, 그것들이 모아져있다. 이것은?

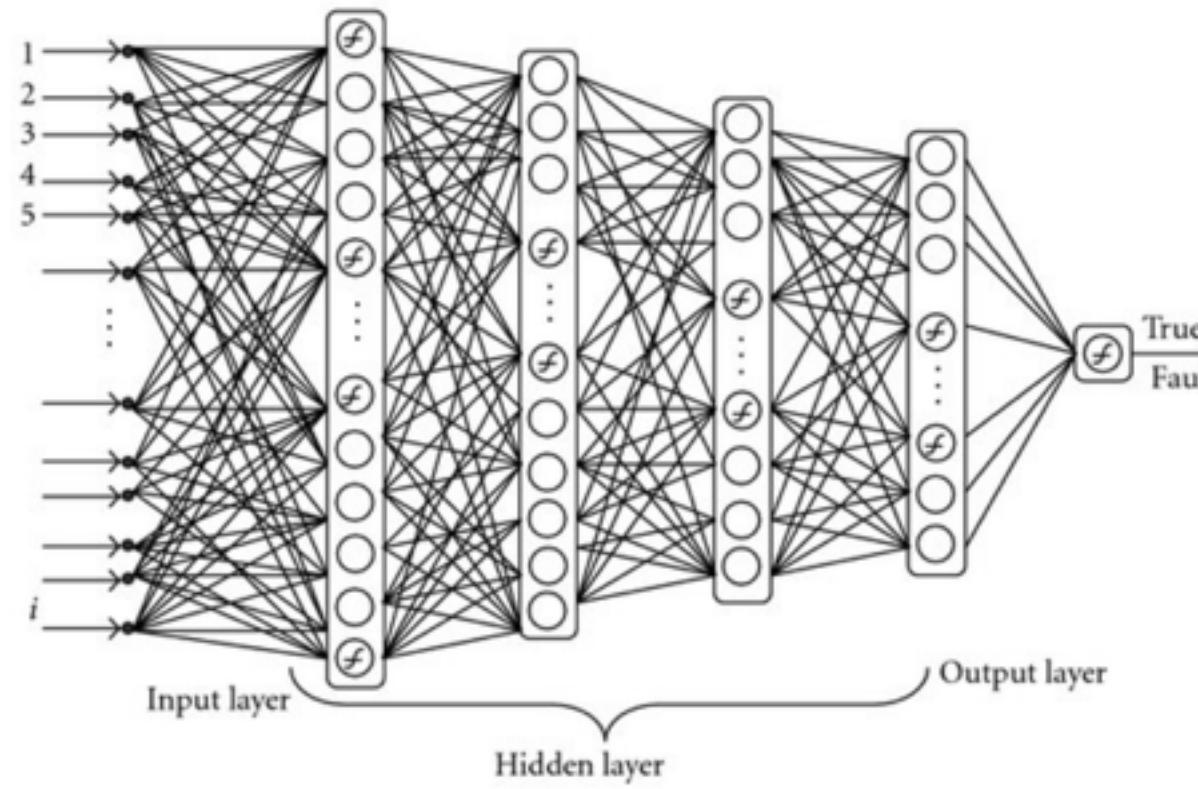


고양이!!

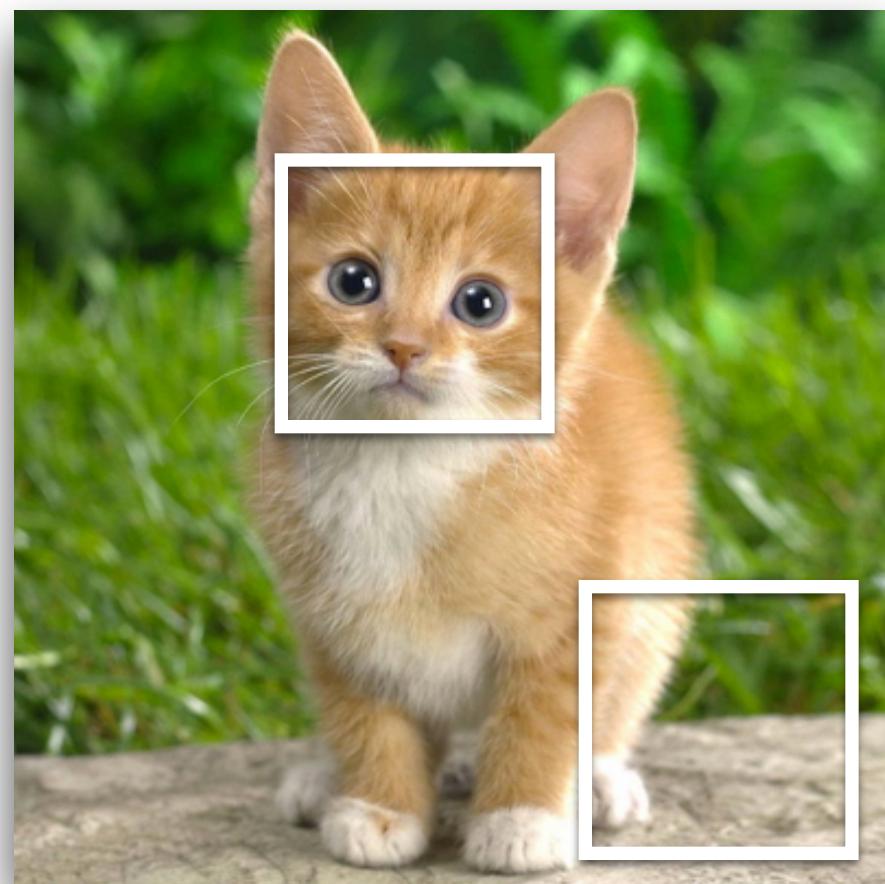
조각을 보고, 패턴을 익히고, 점점 멀리서 조합을 본다.

이 방식을 흉내내면 컴퓨터도 그림을 잘 볼 수 있겠지?

계산량 측면에서도!



이런 생각을 해봤다.
보통 뉴럴넷은 서로가 서로에게 전부다 연결되어 있는데
이라다보니 맞춰야 할 weight들도 많아..



이미지에서는 인근 픽셀끼리만 상관있지 않나?
가까운 것들끼리만 묶어서 계산하면 의미도 있고
계산량도 줄겠는데?

Convolution: 특정 패턴이 있는지 박스로 훑으며 마킹

위아래선필터, 좌우선필터, 대각선필터, 이런질감필터,
요런질감필터, 동그라미필터 등등 여러가지 “조각” 필터로
해당 패턴이 그림 위에 있는지 확인한다.

Convolution 박스로 밀고나면, 숫자가 나옴
그 숫자를 Activation(주로 ReLU)에 넣어나온 값
이걸로 이미지 지도를 새로 그린다.

이게 Conv필터한개예제

$$\left(\begin{array}{ccc} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{array} \right)$$

이런걸 밀고 다닌다.



input image

$$\left(\begin{array}{c} ? \\ \times -1 \end{array} + \begin{array}{c} 68 \\ \times -1 \end{array} + \begin{array}{c} 70 \\ \times -1 \end{array} \right. \\ \left. + \begin{array}{c} ? \\ \times -1 \end{array} + \begin{array}{c} 56 \\ \times 8 \end{array} + \begin{array}{c} 52 \\ \times -1 \end{array} \right. \\ \left. + \begin{array}{c} ? \\ \times -1 \end{array} + \begin{array}{c} 80 \\ \times -1 \end{array} + \begin{array}{c} 59 \\ \times -1 \end{array} \right) \\ = ?$$

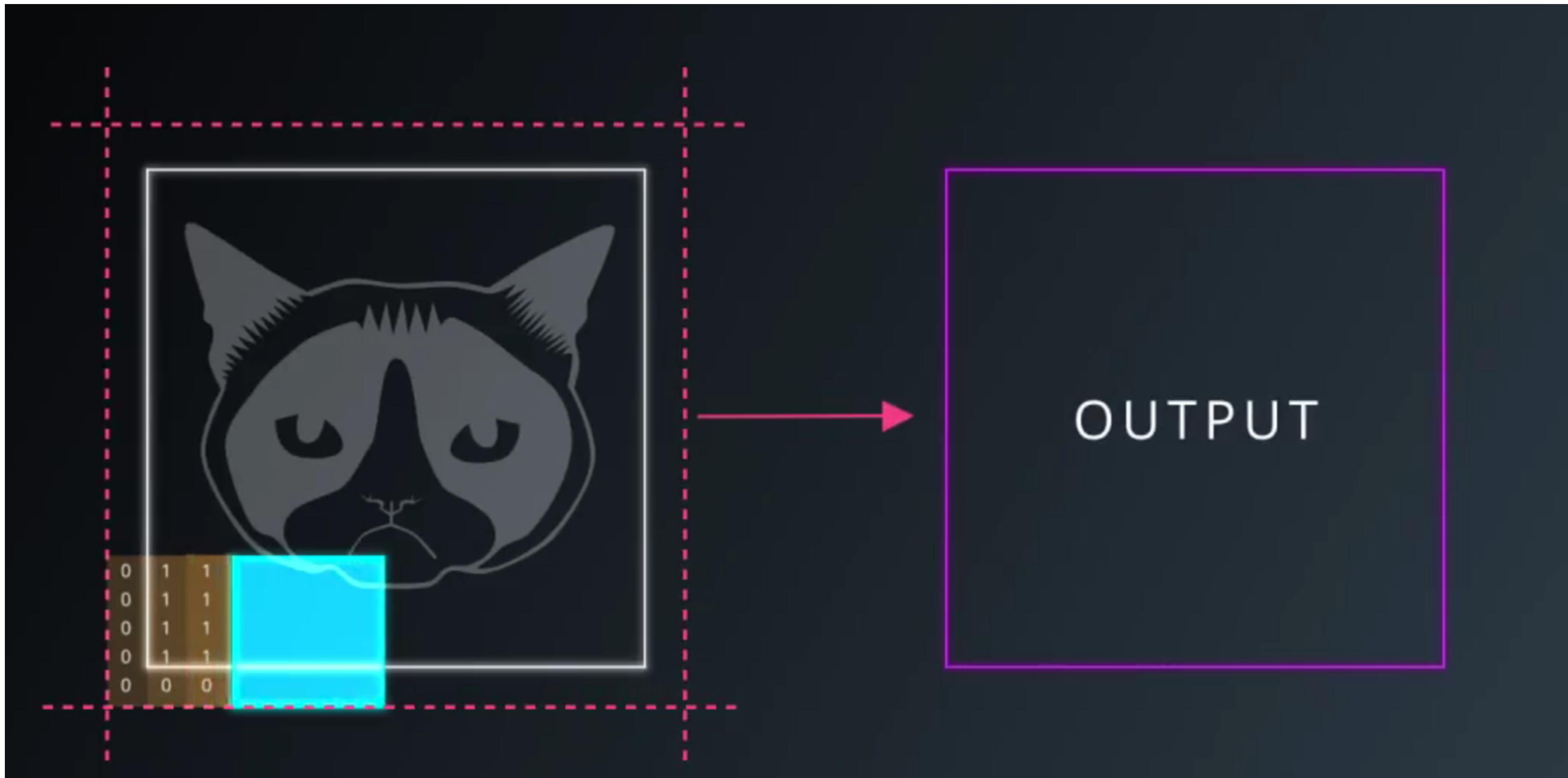
kernel:



output image

사소하지만 중요한거 하나. Zero padding

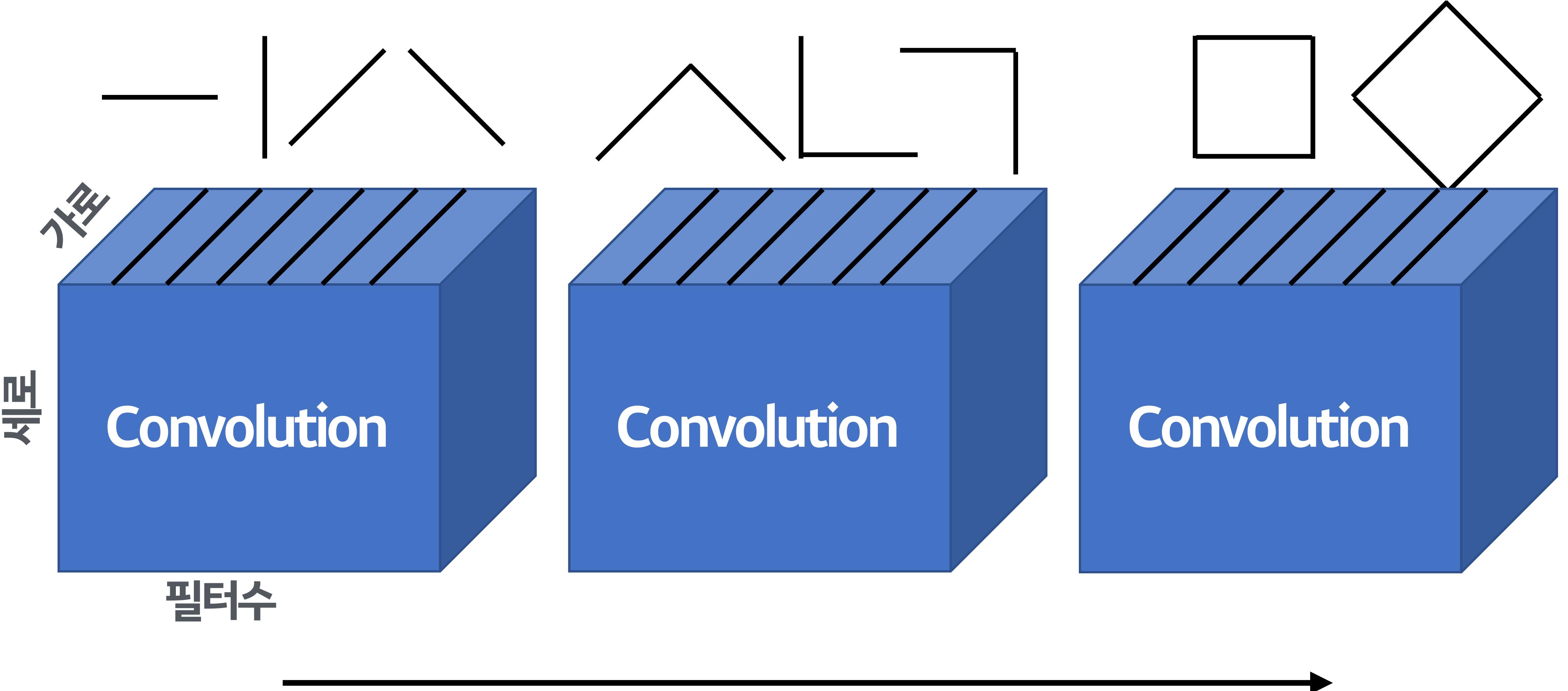
귀퉁이가 짤리다보니, 사이즈유지를 위해 conv 전에 0을 모서리에 보태고 한다.



Convolution의 멋진 점은

간단한 필터들이 쌓여가면서
엄청나게 복잡한 필터를 만들어나가는 것

이런 필터를 뉴럴넷이 알아서 찾아주는 것

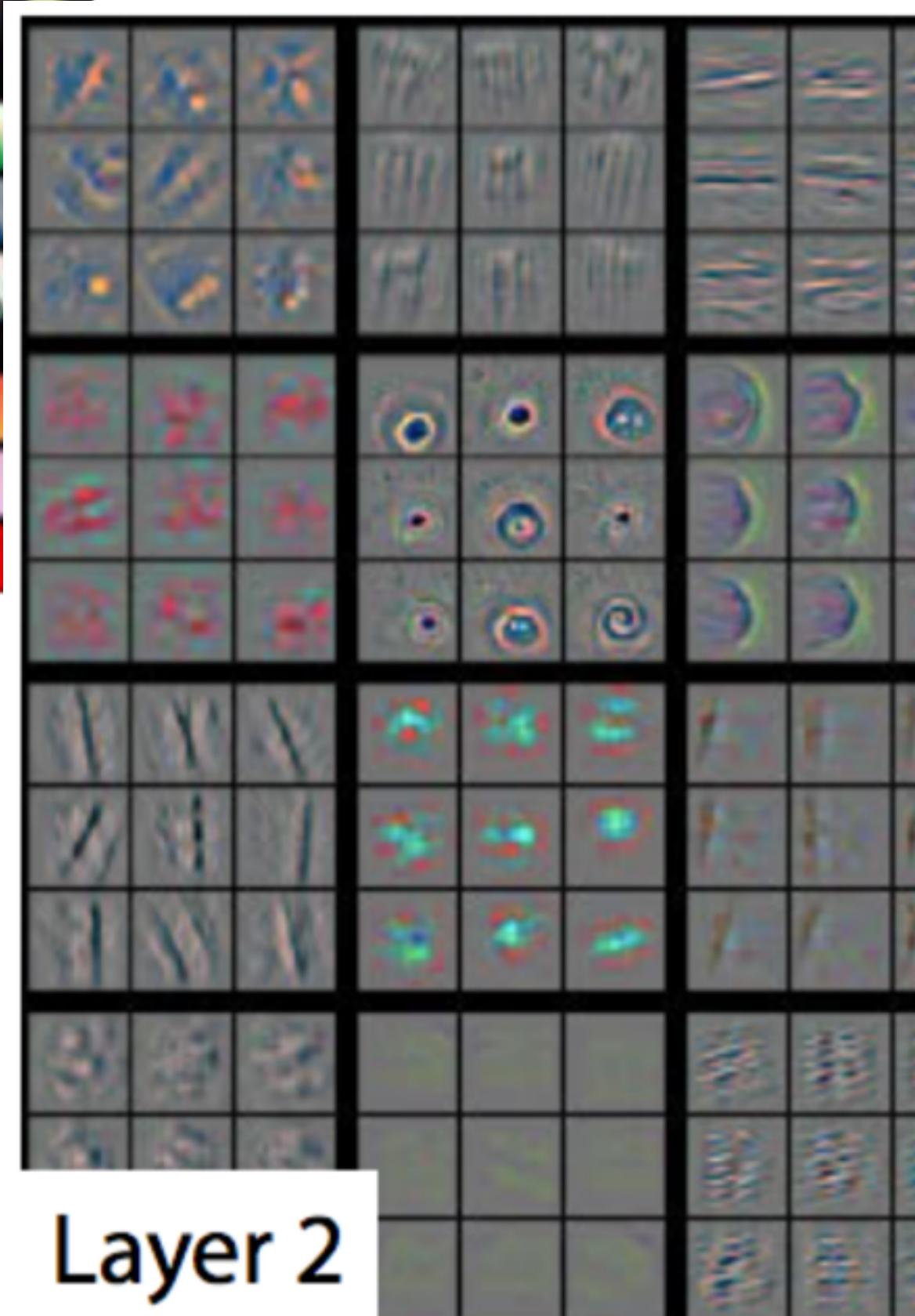


Convolution의 좋은점

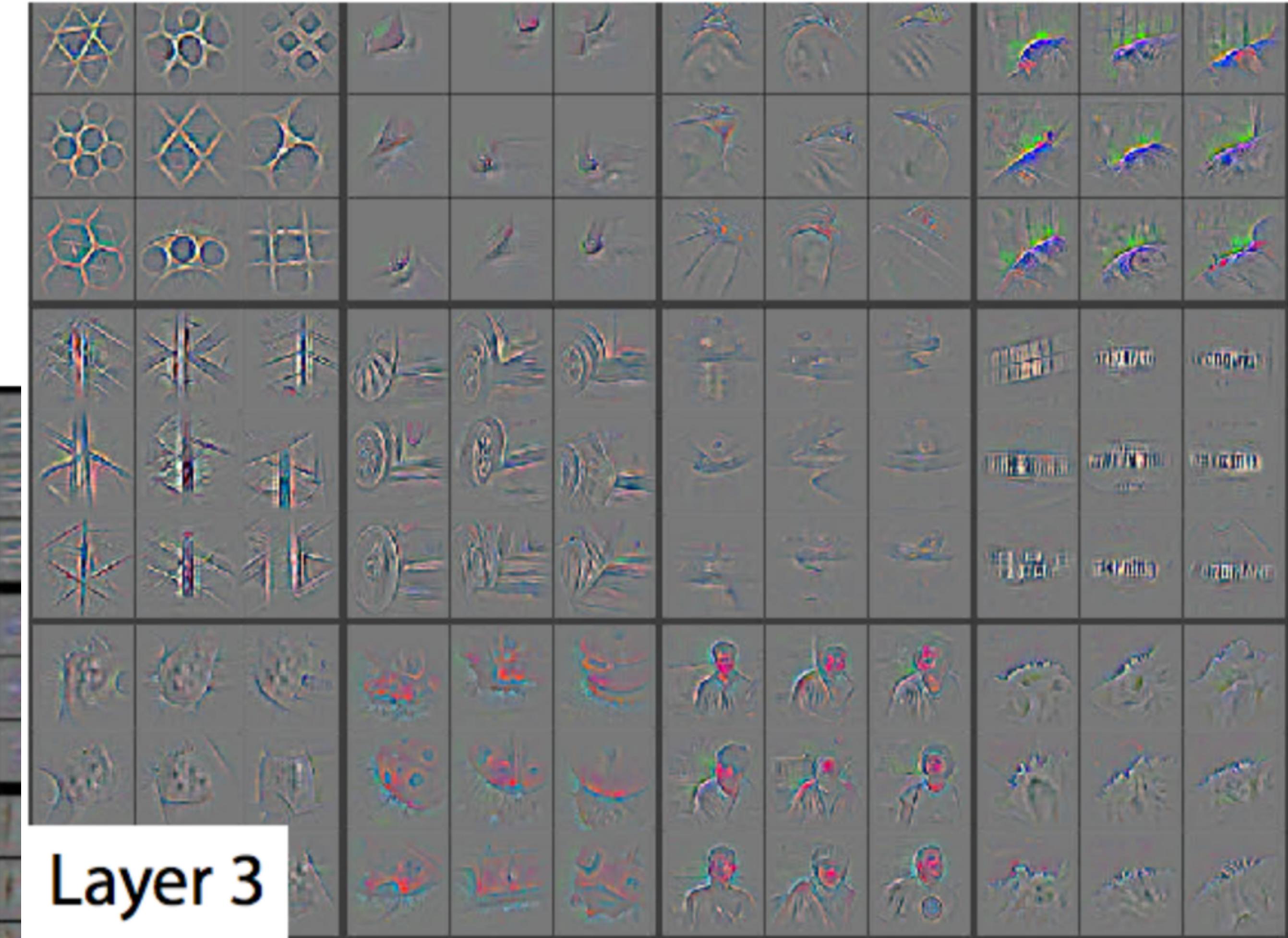
부품을 조립해 더 복잡한 부품을 만든다



Layer 1



Layer 2

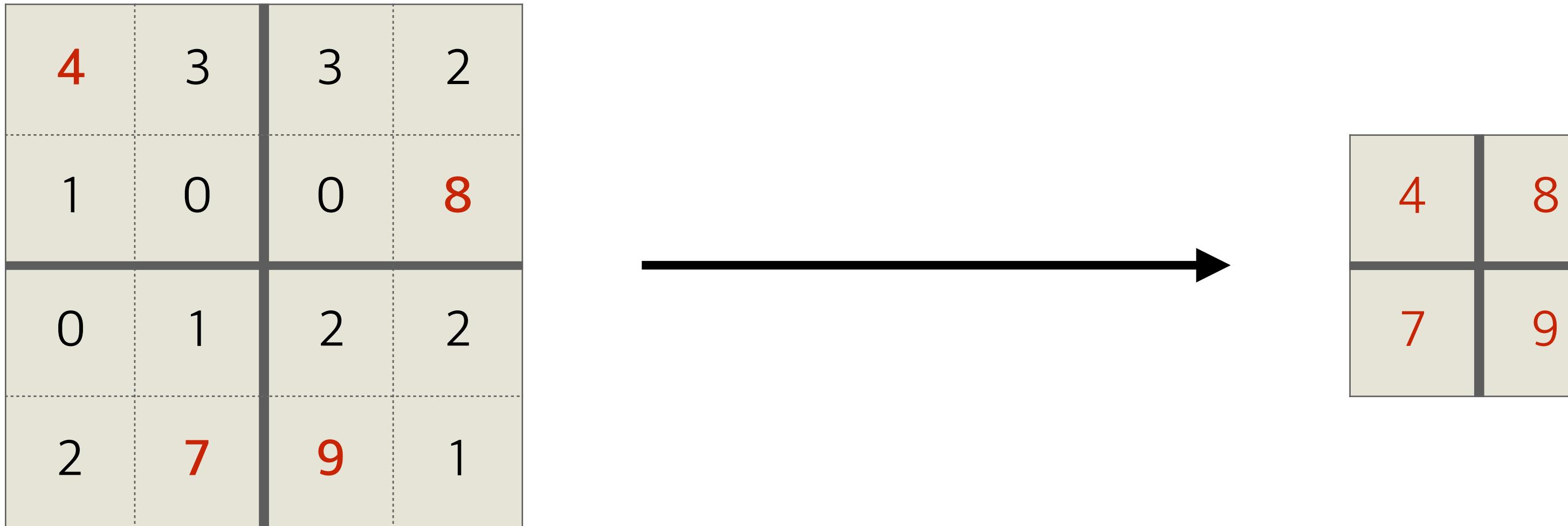


Layer 3

점점 더 멀리서 보는 법?

우리가 멀어져도 되지만
그림을 줄여도 되겠구나?

사이즈를 점진적으로 줄이는 법 MaxPooling



$n \times n$ (Pool)을 중요한 정보(Max)한개로 줄인다.
선명한 정보만 남겨서, 판단과 학습이 쉬워지고
노이즈가 줄면서, 덤으로 융통성도 확보된다.

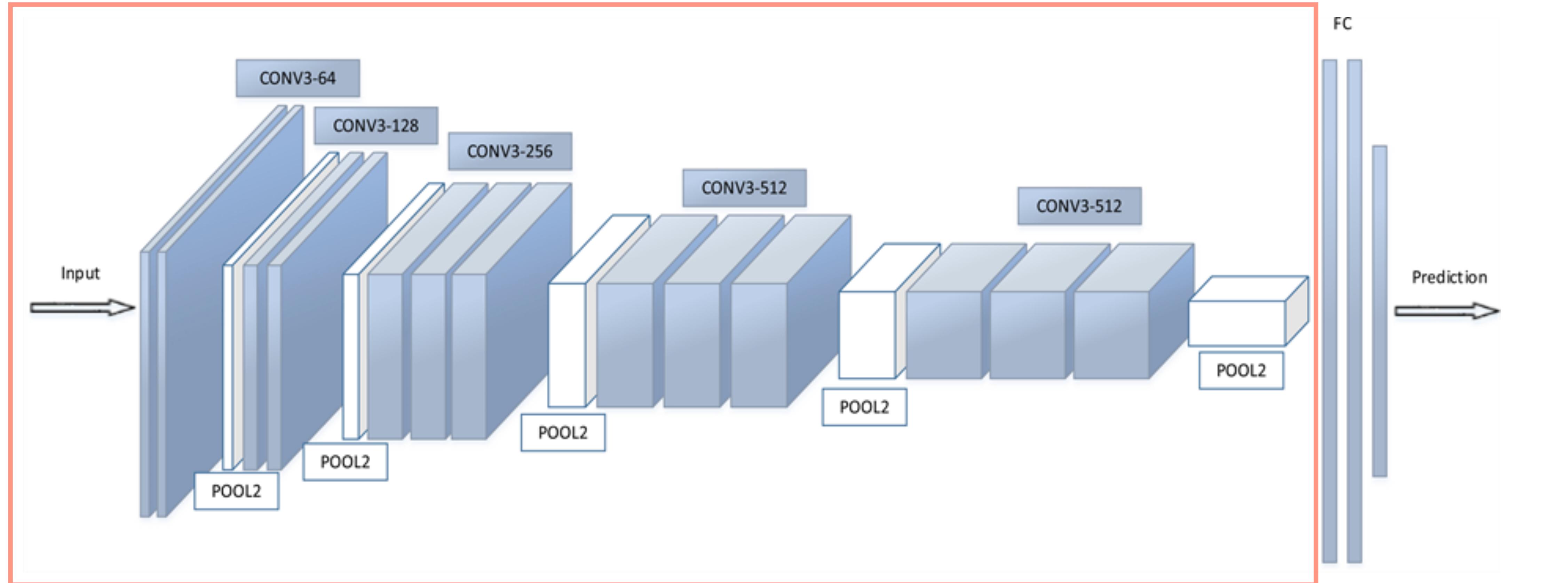
stride라고 해서

보통 2×2 로 화면 전역에 적용한다

좌우로 몇칸씩 뛸지 설정. 보통(2×2)

그러면 절반짜리
이미지가 완성!

패턴들을 쌓아가며 점차 복잡한 패턴을 인식한다(conv)



사이즈를 줄여가며, 더욱 추상화 해나간다(maxpooling)

Deep Visualization Toolbox

yosinski.com/deepvis

#deepvis



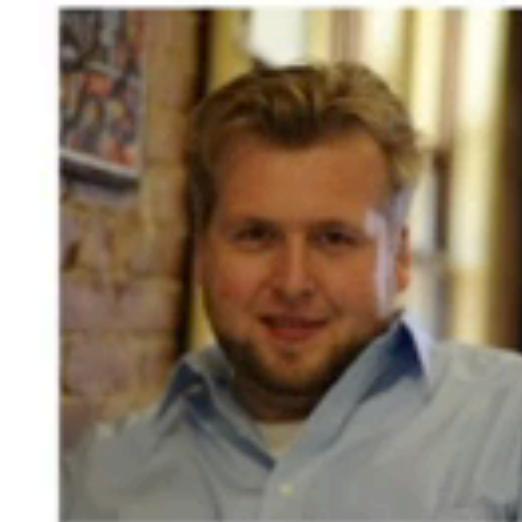
Jason Yosinski



Jeff Clune



Anh Nguyen



Thomas Fuchs



Hod Lipson



Cornell University

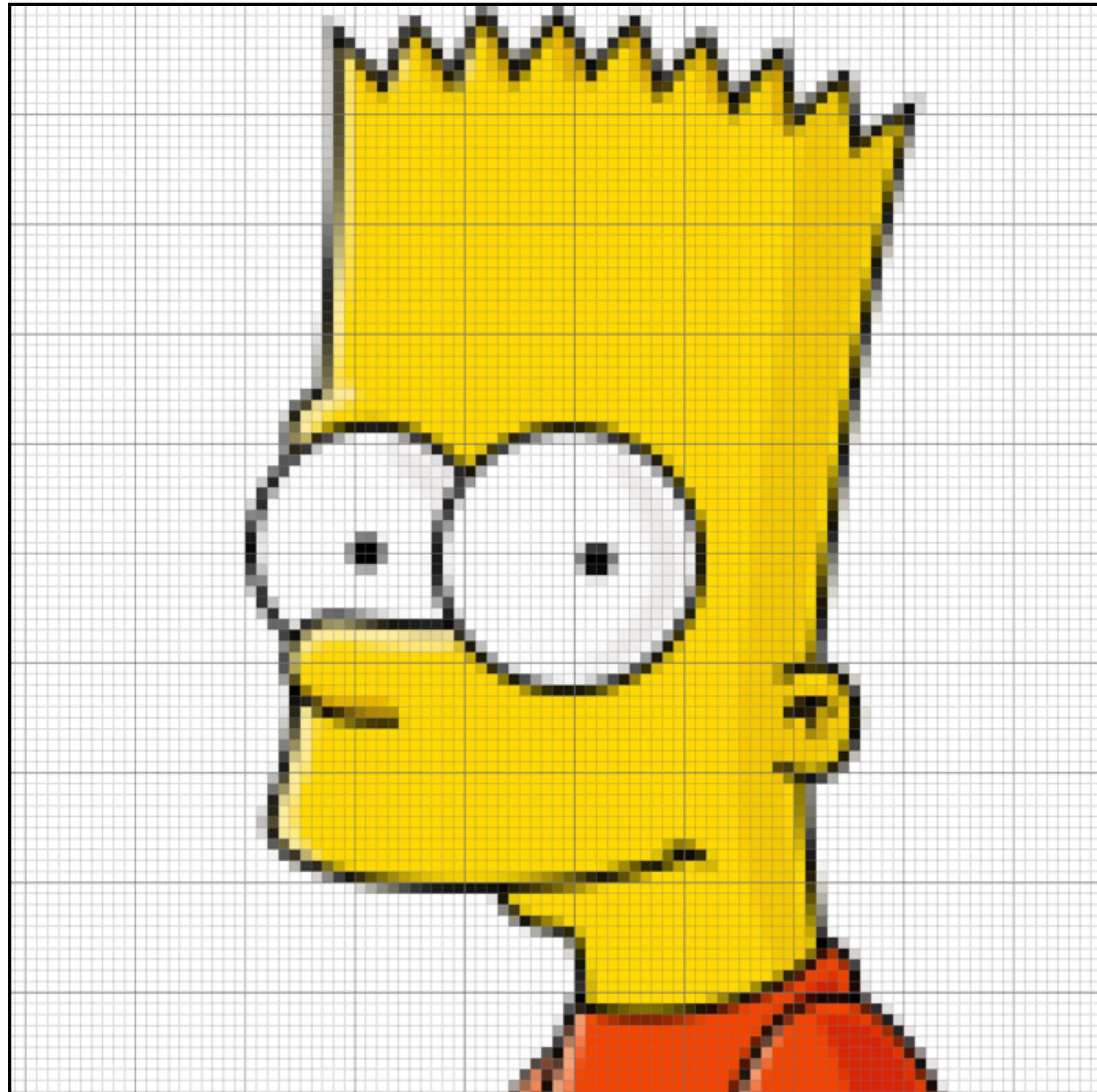


UNIVERSITY
OF WYOMING



Jet Propulsion Laboratory
California Institute of Technology

후반부에는 추상화부품으로 남는다.



Conv와
MaxPooling
의 반복

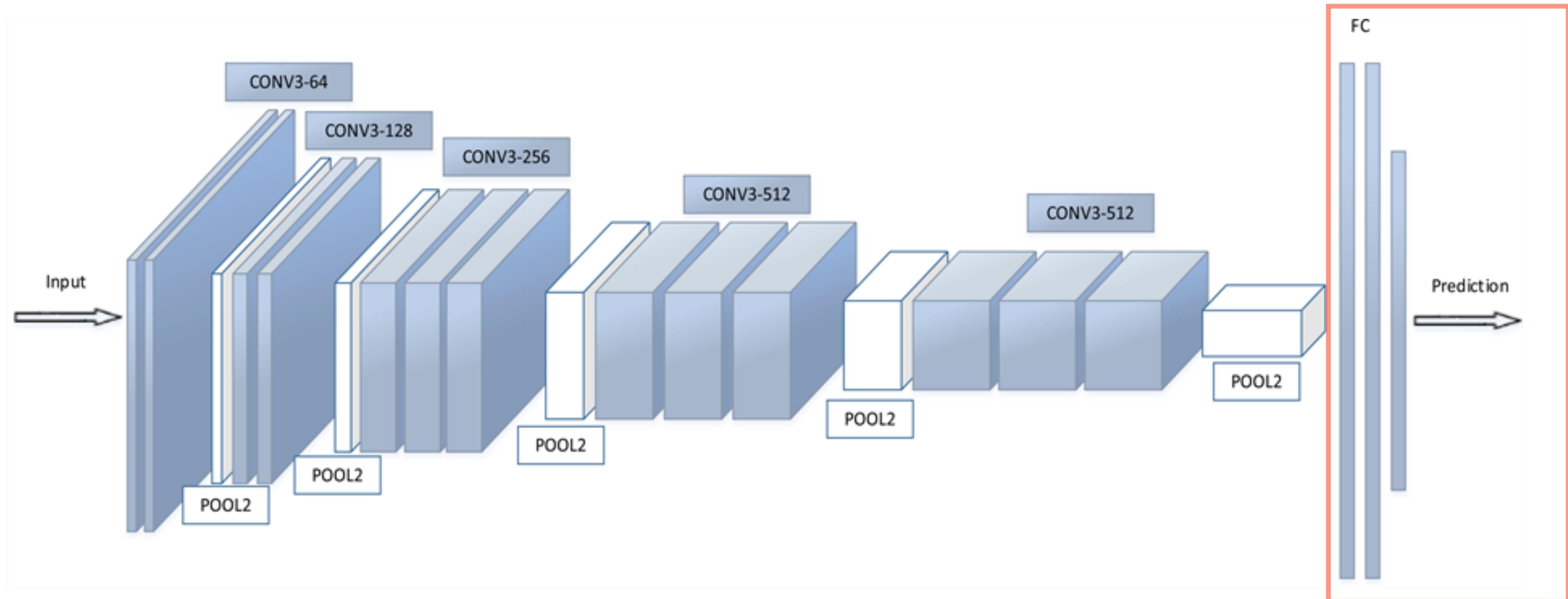


우리는 궁극적으로
이런 녀석을 가지게 된다.

시작은 이렇게 256*256
픽셀을 다 보았어야 해도

막판, 추상화가 끝난 데이터를 FC에 넣어 판단한다

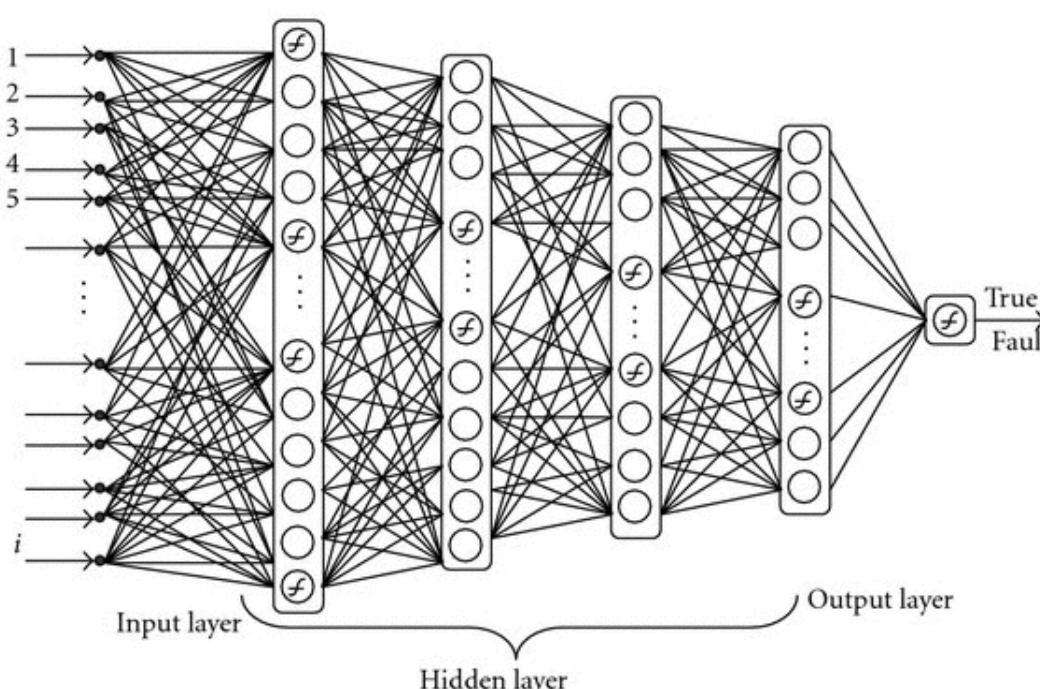
(fully connected layer)



최종판단은 Fully Connected Layer에게 먹여서 하게 한다.



눈과 코와 귀가 있고
티를 입고 있으니



“개” X
“고양이” X
“사람” O
“말” X

뉴런에게 답을 회신받는 3가지 방법

Value

이게 얼마가 될거 같니?

output을
그냥 받는다.

0/X

기냐? 아니냐?

output에
sigmoid를 먹인다.

Category

종류중에 요건 뭐냐?

output에
softmax를 먹인다.

그래서 1000개 종류 분류하는
아키텍쳐는
끝이 SoftMax로 되어있다.

이제 다시 처음의 코드를 봅시다!

```
# 오리지널 VGG16 모델
def VGG_16():
    model = Sequential()
    model.add(Lambda(vgg_preprocess, input_shape=(3,224,224)))

    ConvBlock(2, model, 64)
    ConvBlock(2, model, 128)
    ConvBlock(3, model, 256)
    ConvBlock(3, model, 512)
    ConvBlock(3, model, 512)

    model.add(Flatten())
    FCBLOCK(model)
    FCBLOCK(model)
    model.add(Dense(1000, activation='softmax'))
    return model

model = VGG_16()
model.compile(optimizer=Adam(lr=0.001),
              loss='categorical_crossentropy', metrics=[ 'accuracy' ])
```

```
def ConvBlock(layers, model, filters):
    for i in range(layers):
        model.add(ZeroPadding2D((1,1)))
        model.add(Convolution2D(filters, 3, 3,
                               activation='relu'))
        model.add(MaxPooling2D((2,2), strides=(2,2)))

def FCBLOCK(model):
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
```

Conv블럭은

```
def ConvBlock(layers, model, filters):
    for i in range(layers): 3) 그런 Conv를 layers 수만큼 겹겹히 쌓는구나.
        model.add(ZeroPadding2D((1,1))) 1) 사이즈 안줄어들게 가장자리 채워주고
        model.add(Convolution2D(filters, 3, 3,
                                activation='relu'))
    model.add(MaxPooling2D((2,2), strides=(2,2)))
4) 그렇지. 충분히 했으면 사이즈 줄여줘야지
```

3) 그런 Conv를 layers 수만큼 겹겹히 쌓는구나.
1) 사이즈 안줄어들게 가장자리 채워주고
2) 3x3 사이즈 패턴으로,
filter갯수만큼 패턴을 찾겠구나.
activation은 ReLU를 쓰네?

```
# 오리지널 VGG16 모델
def VGG_16():
    model = Sequential()
    model.add(Lambda(vgg_preprocess, input_shape=(3,224,224)))

    ConvBlock(2, model, 64)
    ConvBlock(2, model, 128)
    ConvBlock(3, model, 256)
    ConvBlock(3, model, 512)
    ConvBlock(3, model, 512)

    model.add(Flatten())
    model.add(FCBlock(model))
    model.add(FCBlock(model))
    model.add(Dense(1000, activation='softmax'))
    return model

model = VGG_16()
model.compile(optimizer=Adam(lr=0.001),
              loss='categorical_crossentropy', metrics=[ 'accuracy' ])
```

Conv+MaxPooling 덩어리를
필터 갯수 늘리면서
그림사이즈는 점점 줄어들게
쌓아 올리는 구만.

이걸 FC에 먹이려고 좌악 펴주고

FC블럭은

```
def FCBlock(model):
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
```

- 1) 출력 4096짜리,
Fully Connected NN이군
- 2) ReLU썼고

3) 그렇지. 오버피팅 방지로 DropOut 들어가야지

```
# 오리지널 VGG16 모델
```

```
def VGG_16():
    model = Sequential()
    model.add(Lambda(vgg_preprocess, input_shape=(3,224,224)))
```

```
    ConvBlock(2, model, 64)
```

```
    ConvBlock(2, model, 128)
```

```
    ConvBlock(3, model, 256)
```

```
    ConvBlock(3, model, 512)
```

```
    ConvBlock(3, model, 512)
```

```
    model.add(Flatten())
```

```
    FCBlock(model)
```

FCBlock(model) **마지막엔 1000개 출력인데,**

```
    model.add(Dense(1000, activation='softmax'))
```

return model **그중 하나 고르는(Category)니까 softmax구만**

```
model = VGG_16()
```

model.compile(optimizer=Adam(lr=0.001), **그렇지. 길찾는법은 믿고 쓰는 Adam이지.**
loss='categorical_crossentropy', metrics=['accuracy'])

```
# 오리지널 VGG16 모델
def VGG_16():
    model = Sequential()
    model.add(Lambda(vgg_preprocess, input_shape=(3,224,224)))

    ConvBlock(2, model, 64)
    ConvBlock(2, model, 128)
    ConvBlock(3, model, 256)
    ConvBlock(3, model, 512)
    ConvBlock(3, model, 512)

    model.add(Flatten())
    FCBLOCK(model)
    FCBLOCK(model)
    model.add(Dense(1000, activation='softmax'))
return model

model = VGG_16()
model.compile(optimizer=Adam(lr=0.001),
              loss='categorical_crossentropy', metrics=['accuracy'])
```

Clear!!!!!!

축하드립니다!

여러분은 지금 실제 돌아가는 코드를
왜 그런지를 알면서
읽어 낼 수 있게 되었어요!

인스톨 완료!

부분부분만 공부하면 어렵게 느껴지던 딥러닝

왜 그렇게 만들어졌는지를 따져보면 구조가 그려집니다.

구조를 알면 어렵지 않아요.

**(LSTM편과 강화학습편은 다음에 또 선보일게요
그리고 저희 사람 뽑습니다. 곧 공고 할게요~~)**