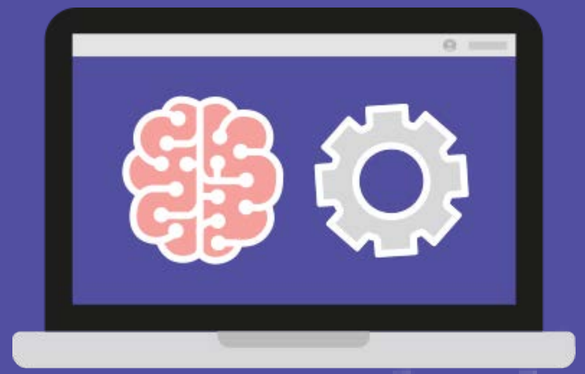
/* elice */

양재 AI School 인공지능 캠프

Lecture 15

데이터 변형(Augmentation), 과적합(Overfitting) 방지(드랍아웃, 정규화)



김도경 선생님

수업 목표

1 데이터 변형(Augmentation)

1. 데이터 변형의 개요와 종류
2. Keras를 이용한 Augmentation
3. OpenCV를 이용한 Augmentation

2 과적합(Overfitting) 방지(드랍아웃, 정규화)

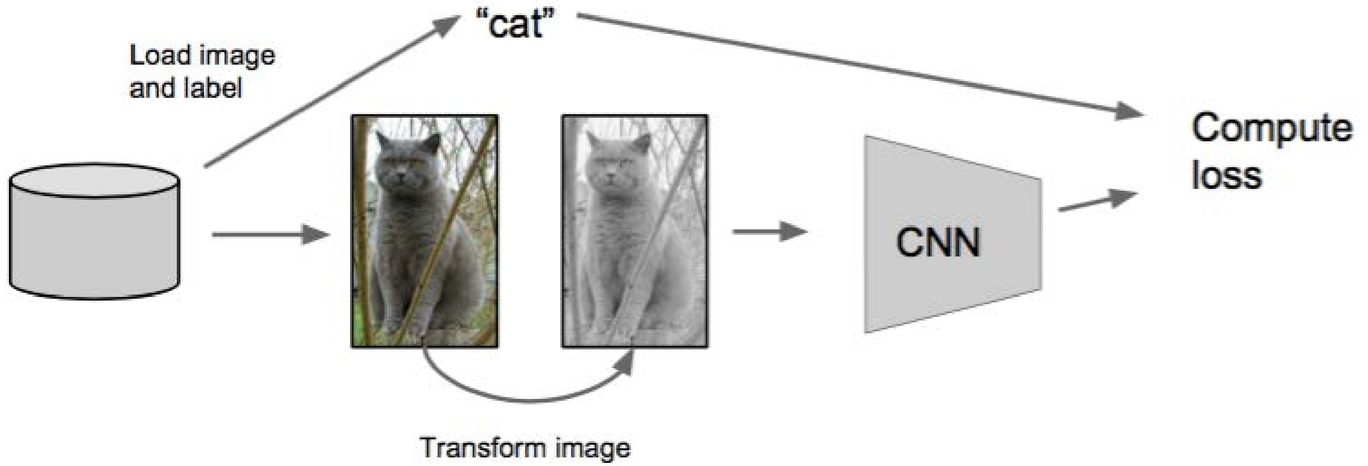
1. 과적합(Overfitting)
2. 드랍아웃(Dropout)
3. 정규화(Regularization)
   L1, L2 정규화
4. 배치 정규화(Batch Normalization) (추후에)

1. 데이터 변형(Augmentation)

1-1. 데이터 변형의 개요와 종류

Data augmentation

- 데이터를 늘려 네트워크(CNN 등)의 성능을 높이기 위해 사용하는 방법
  특히 데이터가 적을 때 사용하면 매우 효과적
- 이미지를 여러 방법을 통해 변형(transform) 한 뒤에
  네트워크의 입력 이미지로 사용하는 방식
- 일종의 정규화(Regularization) 작업으로 과적합을 막는 효과도 있음



1-1. 데이터 변형의 개요와 종류

평행이동(Translation)

- Ex) 이미지에서 모든 픽셀을 오른쪽으로 1픽셀 이동

  1. 사람의 눈에는 같은 이미지로 보임
  2. 컴퓨터는 이미지를 픽셀 벡터의 형태로 표현하고 인식

     =>원본 이미지와 다른 것으로 인식

1-1. 데이터 변형의 개요와 종류

좌우대칭(Horizontal Flip)

- 왼쪽만 바라보는 고양이 사진 70개를 넣어주면
  오른쪽을 보는 고양이는 못맞추게 됨
- 좌우대칭을 시켜주면 어느쪽을 보더라도 맞출수 있음

1-1. 데이터 변형의 개요와 종류

랜덤 크랍(Random Crop)

- 확률적으로 고양이를 꼬리를 보고 50%, 귀를 보고 30%로 판단한다고 할 때, 고양이가 상자속에 들어가서 꼬리만 있는 사진을 사람은 꼬리만 보고도 고양이라고 판단할 수 있음
  즉, 가려짐(Occlusion)에 대응 가능
- Random crop을 해서 NN에 넣어주면 각 부분만 보고도 고양이로 판단 가능
- **주의**) 오검출률(False positive rate)이 높아질 수 있으므로 상황에 따라 적절한 조절이 필요

1-1. 데이터 변형의 개요와 종류

밝기 조절(Brightness Change)

- 조명이나 빛의 반사 등에 의해 밝기가 변해도 NN이 인식 가능

1-1. 데이터 변형의 개요와 종류

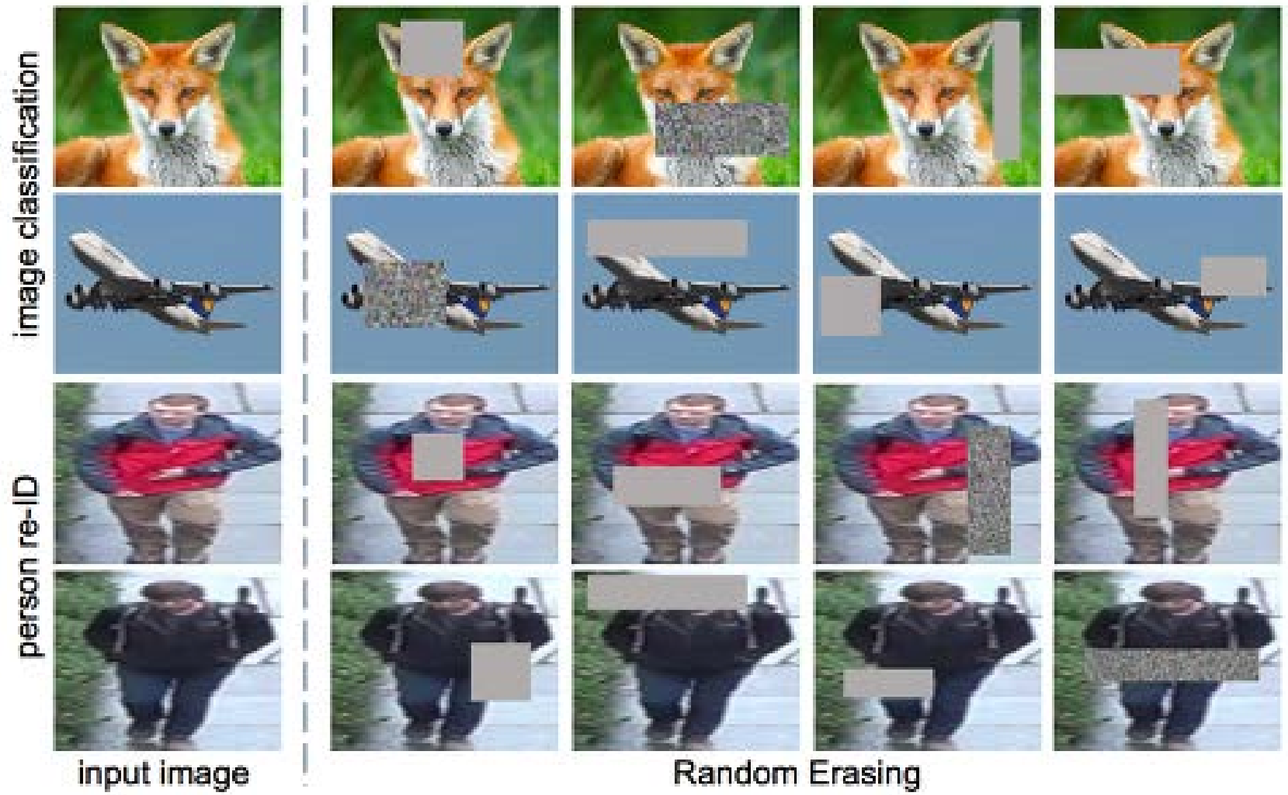
크기변경(Rescale)

- 이미지의 크기가 바뀌어도 NN이 같은 이미지로 인식 가능
- **확대축소(Zoom)**는 Crop후 Rescale하면 가능

1-1. 데이터 변형의 개요와 종류

일부 지우기(Random Erasing)

- 가려짐(Occlusion)에 대응 가능



1-1. 데이터 변형의 개요와 종류

블러(Blurring)

- Gaussian blur, Bilateral blur, Median blur 등 사진을 흐리게 하는 많은 기법들도 NN의 학습에 도움이 됨(아래는 유명한 Lena 사진)

1-1. 데이터 변형의 개요와 종류

컬러 노이즈(Color Noise)

- Alex Krizhevsky가 알렉스넷에서 Overfitting을 방지하기 위해 사용
- 각 RGB 칼럼의 principal vector, principal eigenvalue에 약간의 random 요소를 더해서 RGB를 미세하게 같은 방향, 크기로 변화시키는 방법 (PCA 사용)

1-1. 데이터 변형의 개요와 종류

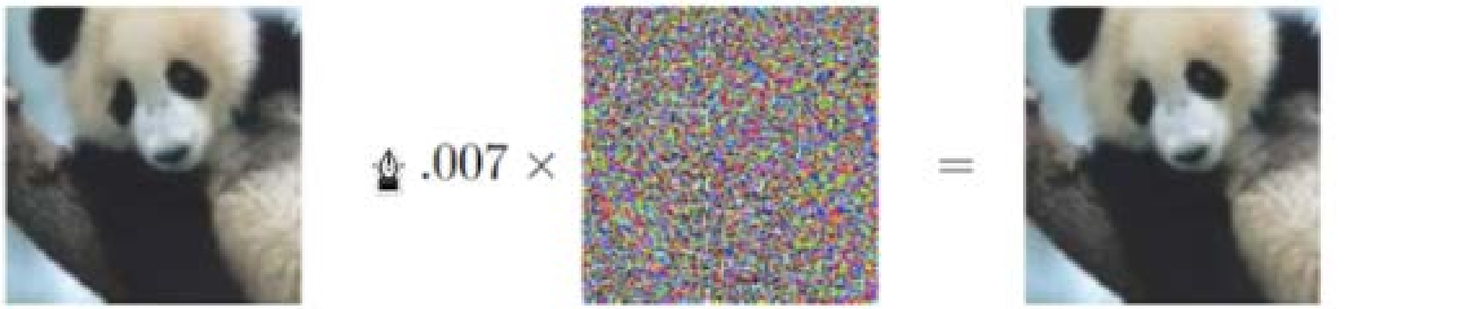
랜덤 노이즈(Random Noise)

- Gaussian noise 등의 노이즈를 입힐 수도 있음

1-1. 데이터 변형의 개요와 종류

랜덤 노이즈(Random Noise)

- Adversarial attack와 같은 현상도 있으므로 주의가 필요



1-2. Keras를 이용한 Augmentation

```
# example of horizontal shift image augmentation
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
# load the image
img = load_img('bird.jpg')
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)

# create image data augmentation generator
# 아래의 6개 옵션 중 하나를 고르거나 여러 옵션을 동시에 주시면 됩니다.
datagen = ImageDataGenerator(width_shift_range=[-200,200])
# 나머지 5개 옵션
# height_shift_range=0.5, horizontal_flip=True, rotation_range=90, brightness_range=[0.2,1.0], zoom_range=[0.5,1.0]
```

1-2. Keras를 이용한 Augmentation

```
# prepare iterator
it = datagen.flow(samples, batch_size=1)
# generate samples and plot
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    plt.imshow(image)
# show the figure
plt.show()
```

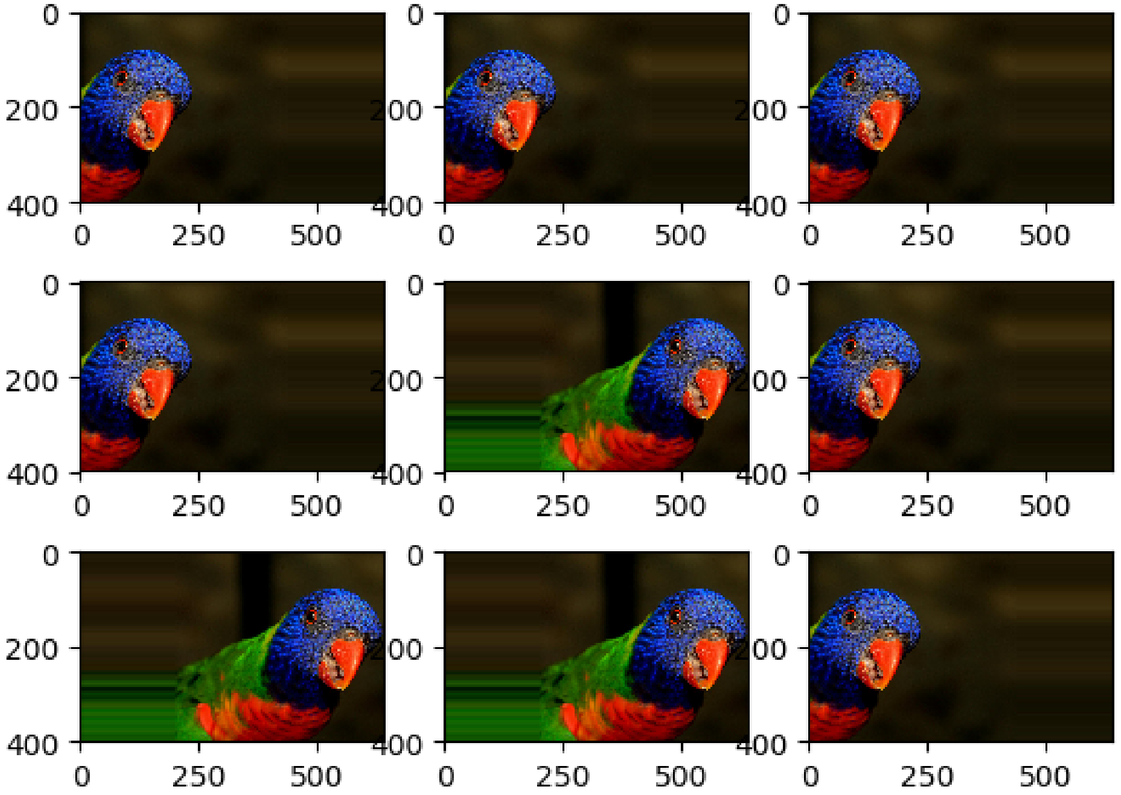
1-2. Keras를 이용한 Augmentation

- 원본 이미지

1-2. Keras를 이용한 Augmentation

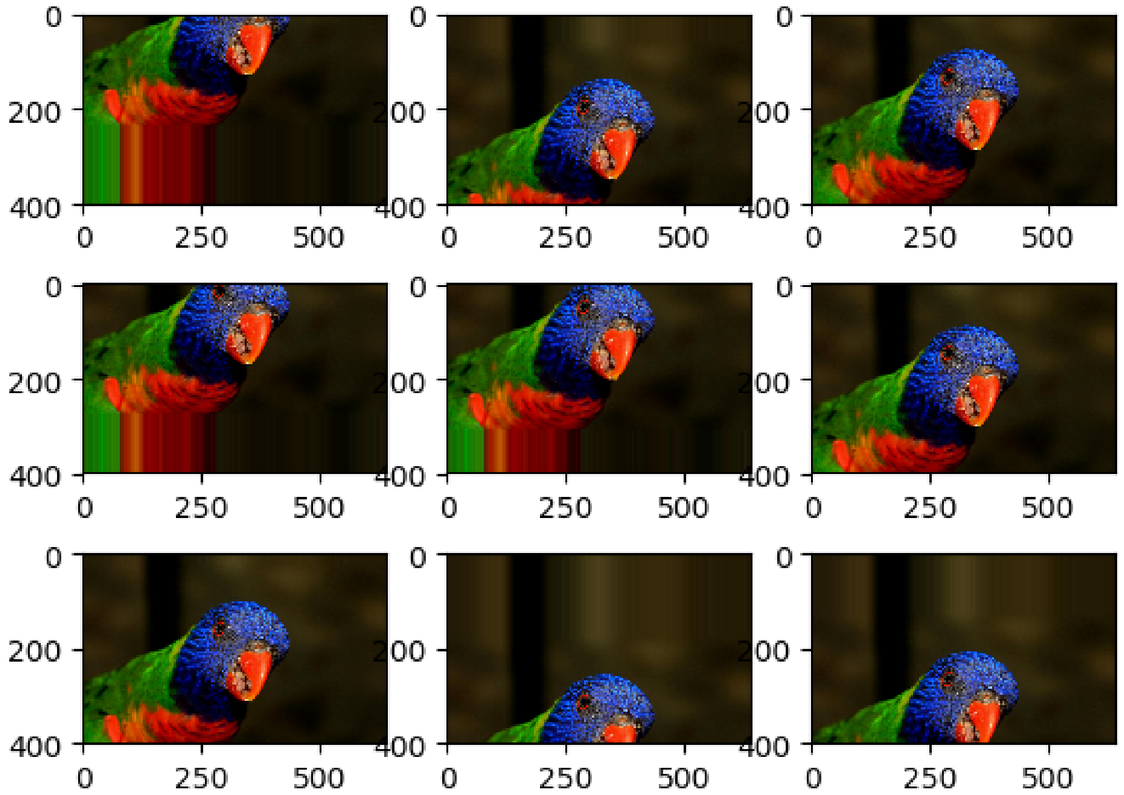
- width_shift_range=[-200,200]

가로방향 평행이동



- height_shift_range=0.5

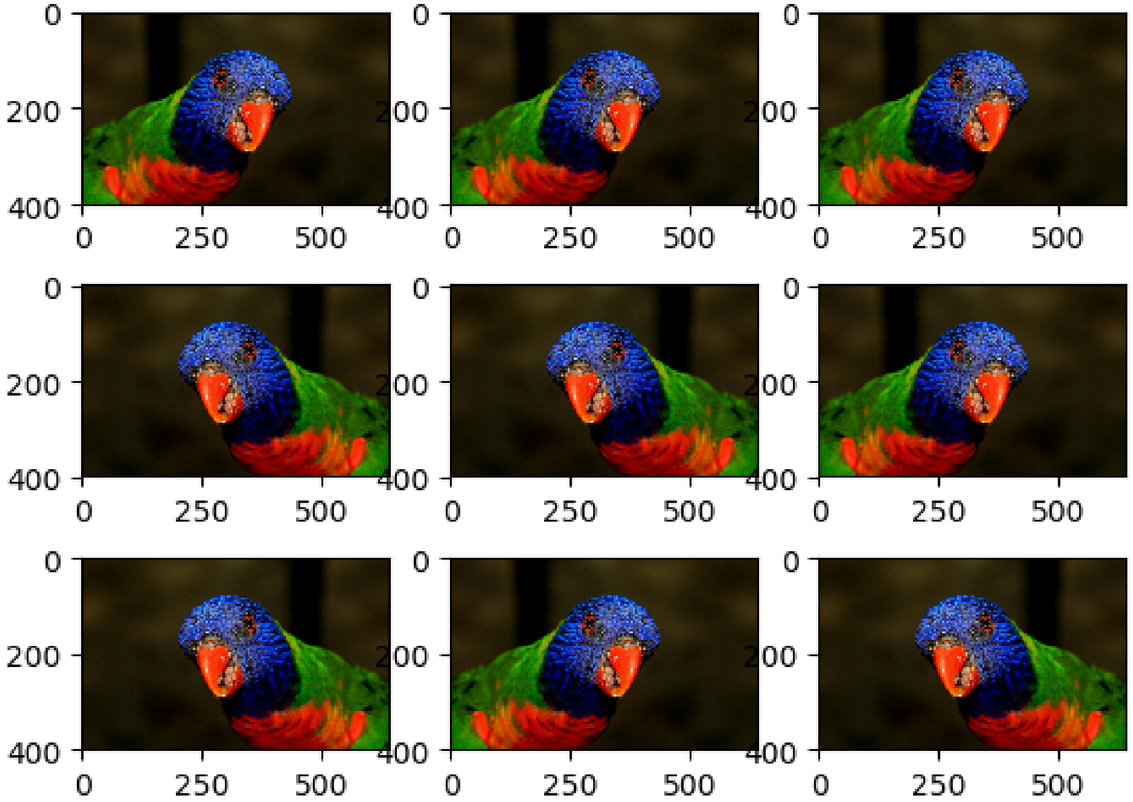
세로방향 평행이동



1-2. Keras를 이용한 Augmentation

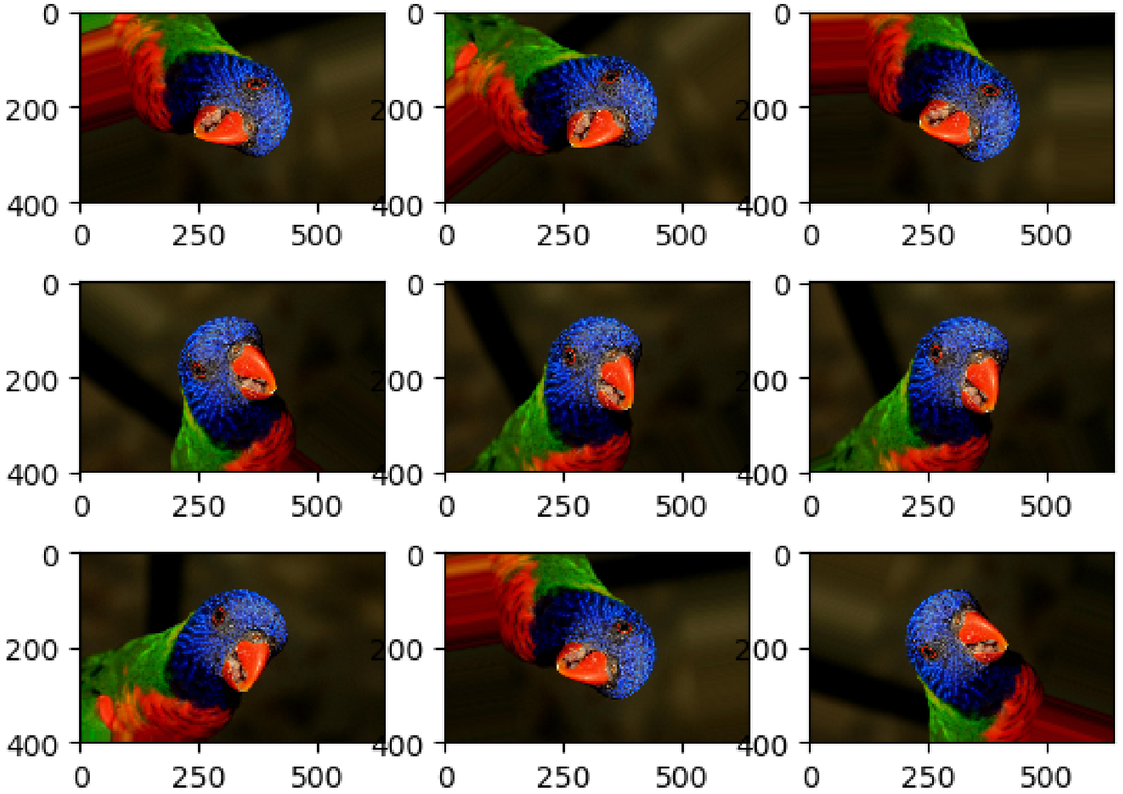
- horizontal_flip=True

가로방향 대칭이동



- rotation_range=90

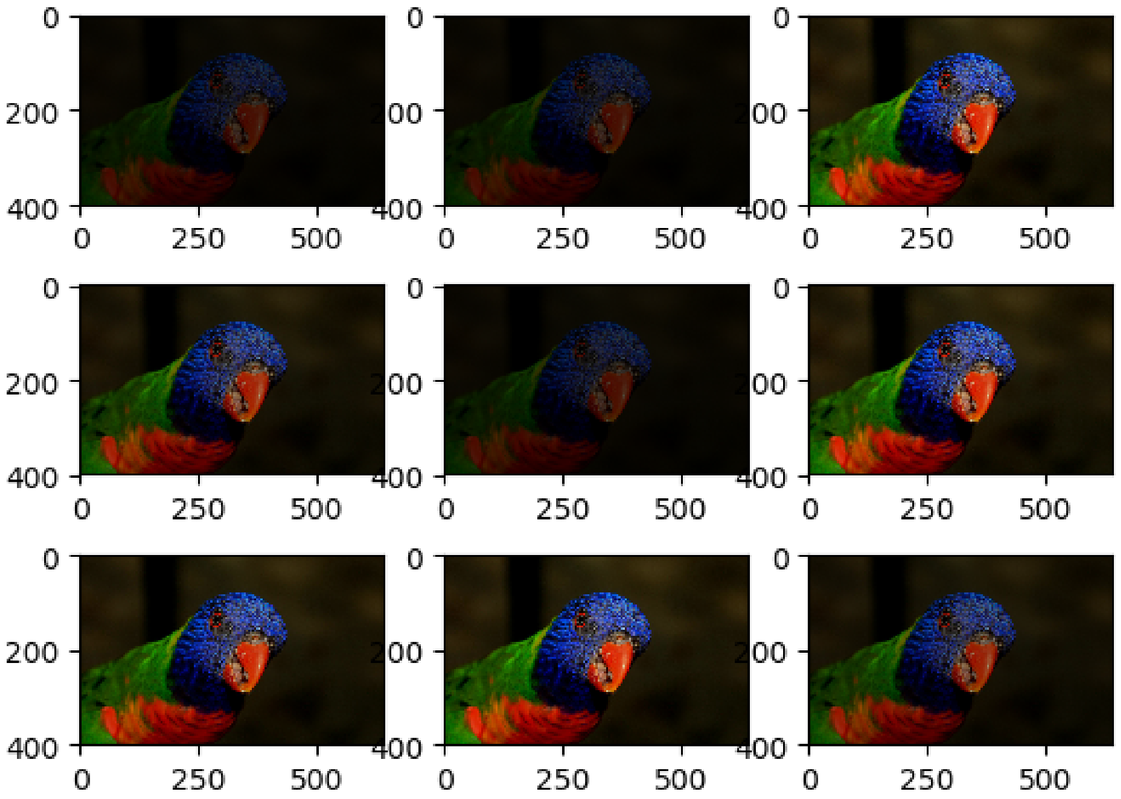
시계방향으로 90도 회전



1-2. Keras를 이용한 Augmentation

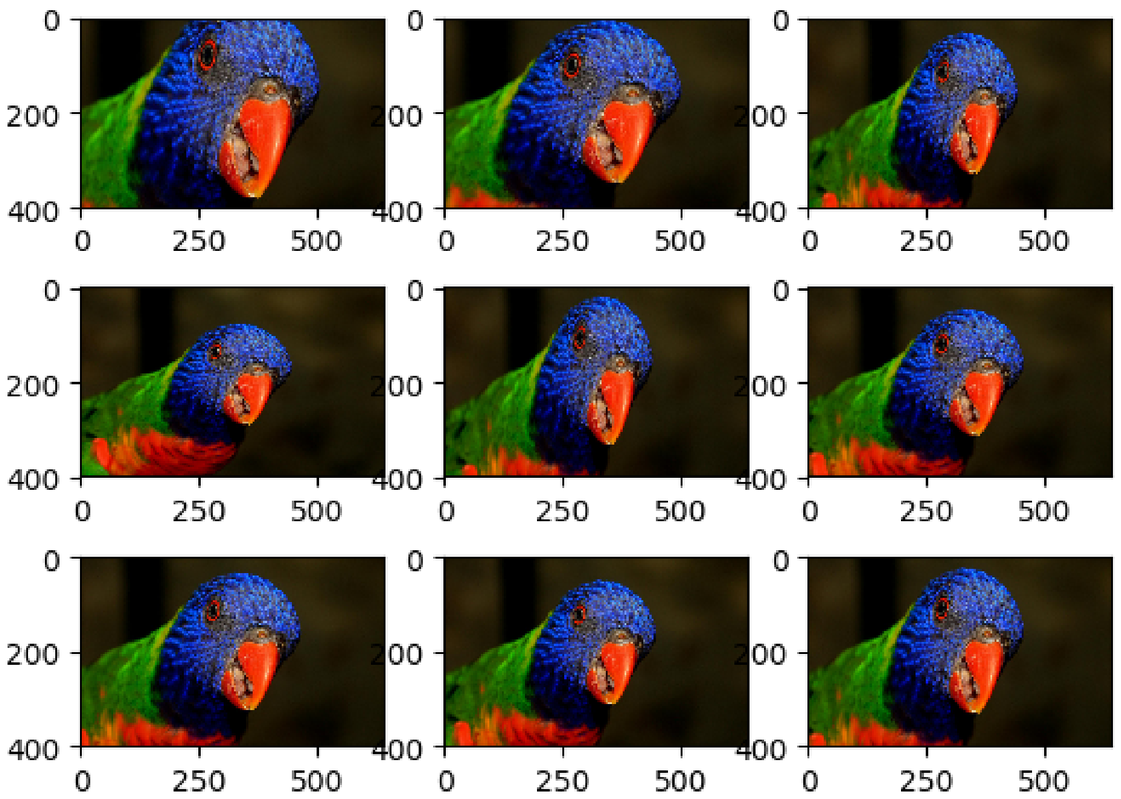
- brightness_range=[0.2,1.0]

밝기 조절



- zoom_range=[0.5,1.0]

가로, 세로 따로 확대



1-3. OpenCV를 이용한 Augmentation

OpenCV

- 인텔에서 개발한 Computer Vision 관련 프로그래밍을 쉽게 할 수 있도록 도와주는 Open Library
- 웬만큼 특수한 상황이 아니면 OpenCV만으로도 원하는 영상 처리가 가능
- **주요 알고리즘**

  이진화(binarization), 노이즈 제거, 외곽선 검출(edge detection), 패턴인식, 기계학습(machine learning), ROI(Region Of Interest) 설정, 이미지 변환(image warping), 하드웨어 가속

1-3. OpenCV를 이용한 Augmentation

```
    #RESIZE
    def resize_image(image,w,h):
        image=cv2.resize(image,(w,h))
        cv2.imwrite(Folder_name+"/Resize-"+str(w)+"*"+str(h)+Extension, image)

    #crop
    def crop_image(image,y1,y2,x1,x2):
        image=image[y1:y2,x1:x2]
        cv2.imwrite(Folder_name+"/Crop-"+str(x1)+str(x2)+"*"+str(y1)+str(y2)+Extension, image)

    def padding_image(image,topBorder,bottomBorder,leftBorder,rightBorder,color_of_border=[0,0,0]):
        image = cv2.copyMakeBorder(image,topBorder,bottomBorder,leftBorder,
            rightBorder,cv2.BORDER_CONSTANT,value=color_of_border)
        cv2.imwrite(Folder_name + "/padd-" + str(topBorder) + str(bottomBorder) + "*" + str(leftBorder) +
str(rightBorder) + Extension, image)

    def flip_image(image,dir):
        image = cv2.flip(image, dir)
        cv2.imwrite(Folder_name + "/flip-" + str(dir)+Extension, image)
```

1-3. OpenCV를 이용한 Augmentation

```
def invert_image(image,channel):
    # image=cv2.bitwise_not(image)
    image=(channel-image)
    cv2.imwrite(Folder_name + "/invert-"+str(channel)+Extension, image)

def add_light(image, gamma=1.0):
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
                      for i in np.arange(0, 256)]).astype("uint8")

    image=cv2.LUT(image, table)
    if gamma>=1:
        cv2.imwrite(Folder_name + "/light-"+str(gamma)+Extension, image)
    else:
        cv2.imwrite(Folder_name + "/dark-" + str(gamma) + Extension, image)
```

1-3. OpenCV를 이용한 Augmentation

```
def saturation_image(image,saturation):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    v = image[:, :, 2]
    v = np.where(v <= 255 - saturation, v + saturation, 255)
    image[:, :, 2] = v

    image = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)
    cv2.imwrite(Folder_name + "/saturation-" + str(saturation) + Extension,
image)
```

1-3. OpenCV를 이용한 Augmentation

```
def scale_image(image,fx,fy):
    image = cv2.resize(image,None,fx=fx, fy=fy, interpolation = cv2.INTER_CUBIC)
    cv2.imwrite(Folder_name+"/Scale-"+str(fx)+str(fy)+Extension, image)

def translation_image(image,x,y):
    rows, cols ,c= image.shape
    M = np.float32([[1, 0, x], [0, 1, y]])
    image = cv2.warpAffine(image, M, (cols, rows))
    cv2.imwrite(Folder_name + "/Translation-" + str(x) + str(y) + Extension, image)

def rotate_image(image,deg):
    rows, cols,c = image.shape
    M = cv2.getRotationMatrix2D((cols/2,rows/2), deg, 1)
    image = cv2.warpAffine(image, M, (cols, rows))
    cv2.imwrite(Folder_name + "/Rotate-" + str(deg) + Extension, image)
```

1-3. OpenCV를 이용한 Augmentation

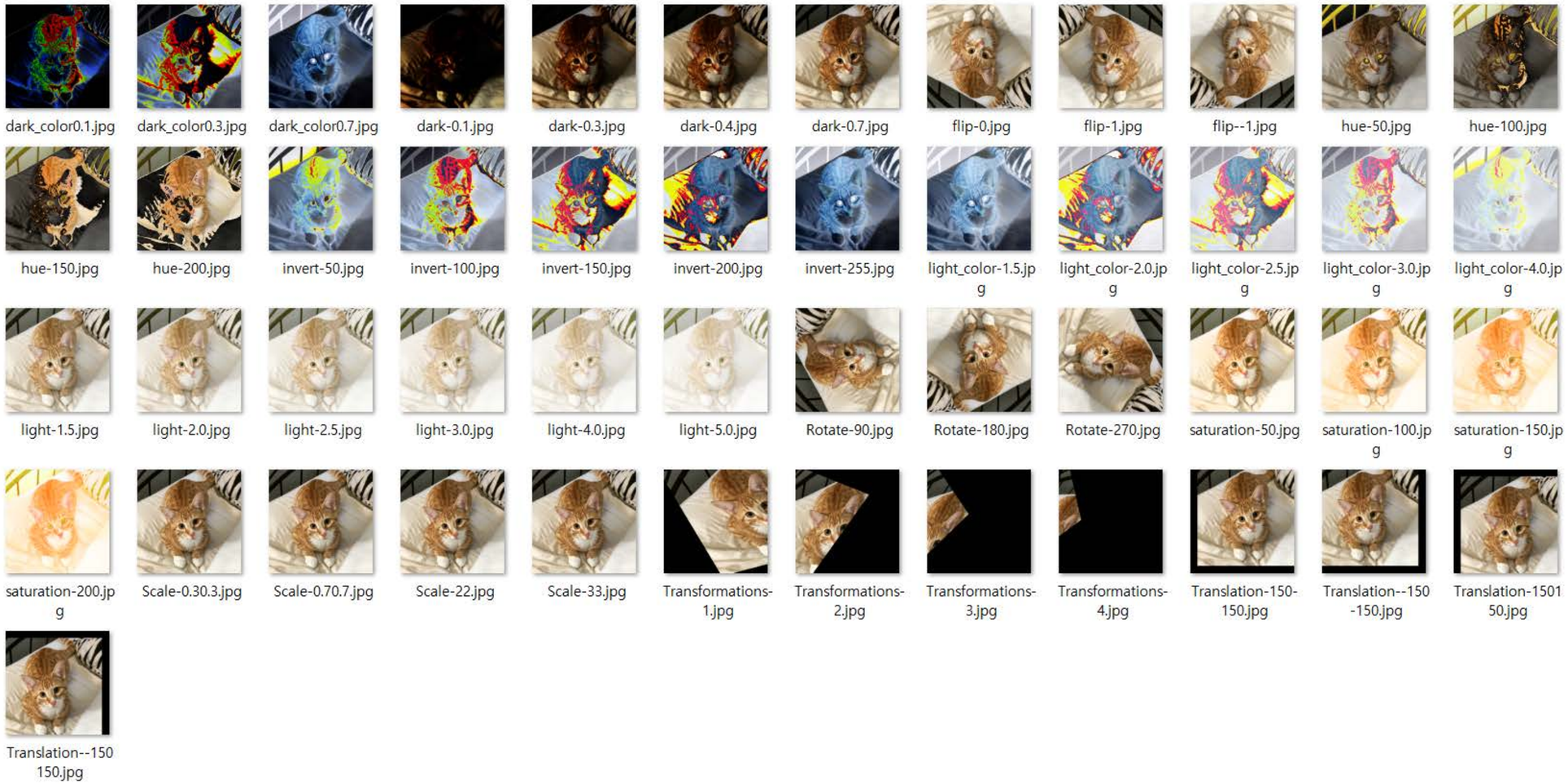
```
image_file="cat.jpg"
image=cv2.imread(image_file)
resize_image(image,450,400)


crop_image(image,100,400,0,350)
padding_image(image,100,0,0,0)
flip_image(image,0)
invert_image(image,255)
add_light(image,1.5)
add_light_color(image,255,1.5)
saturation_image(image,50)
hue_image(image,50)
translation_image(image,150,150)
rotate_image(image,90)
transformation_image(image)
```

1-3. OpenCV를 이용한 Augmentation

- 원본 이미지

1-3. OpenCV를 이용한 Augmentation

- 실행결과

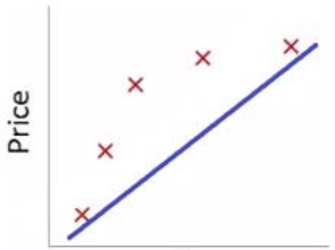


2. 과적합(Overfitting) 방지(드랍아웃, 정규화)

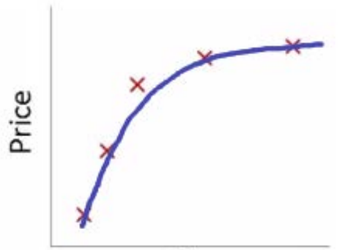
2-1. 과적합(Overfitting)

과적합

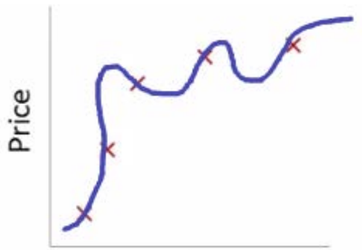
- Training 데이터의 많은 공통특성 이외에 지엽적인 특성까지 반영해 high variance로 훈련되어, Test 데이터에 대해서는 제대로 예측하지 못하는 현상
- 주로 파라미터가 많은 모델에 발생 (표현력이 높은 모델)



$\theta_0 + \theta_1 x$



$\theta_0 + \theta_1 x + \theta_2 x^2$

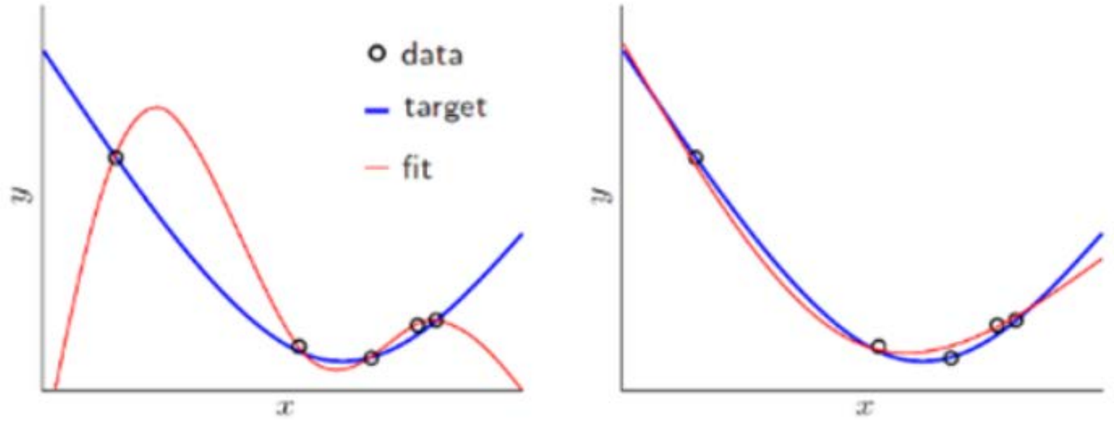


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

2-2. 정규화(Regularization)

정규화(Regularization)

- 과적합을 억제하기 위해서 사용하는 기법, 일반화 라고도 번역
- 손실함수에 가중치의 크기를 포함
- 가중치가 작아지도록 학습한다는 것은 노이즈에 영향을 덜 받도록 하겠다는 것 => Outlier의 영향을 적게 받음



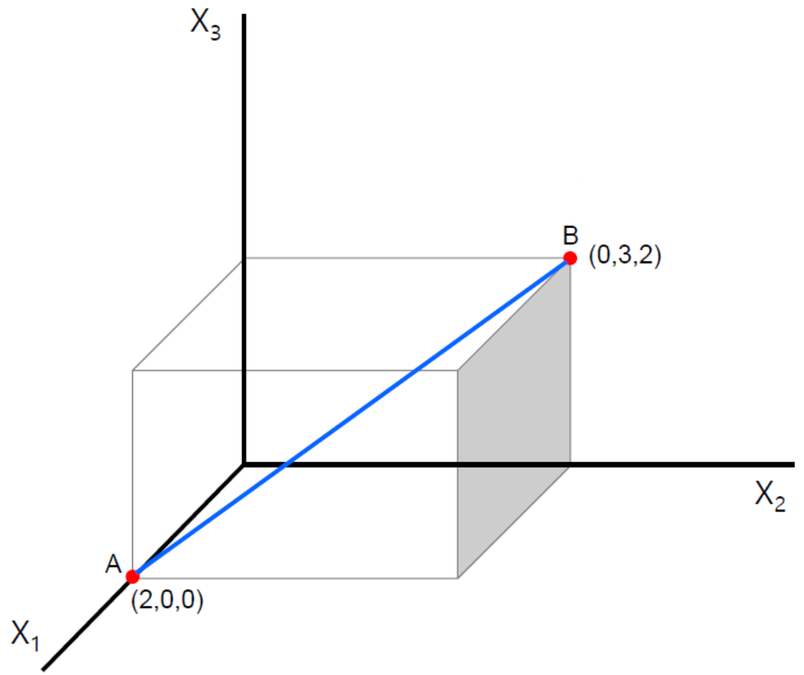
(a) without regularization (b) with regularization

2-2. 정규화(Regularization)

- L2 norm(유클리드 거리)

$$d_{(X,Y)} = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2}$$

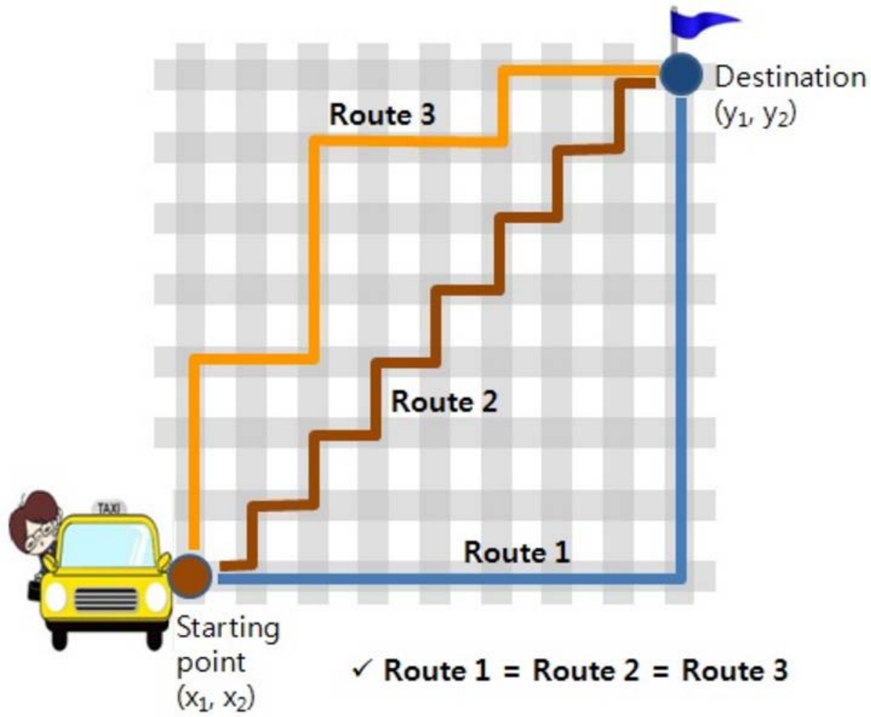
$$= \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$



2-2. 정규화(Regularization)

- L1 norm(맨해튼 거리)

$$d_{Manhattan(X,Y)} = \sum_{i=1}^{n} |x_i - y_i|$$



2-2. 정규화(Regularization)

- **L2 정규화**

$$Cost = \frac{1}{n}\sum_{i=1}^{n}\{L(y_i, \hat{y_i}) + \frac{\lambda}{2}|w|^2\}$$

예) Ridge Regression

- **L1 정규화**

$$Cost = \frac{1}{n}\sum_{i=1}^{n}\{L(y_i, \hat{y_i}) + \frac{\lambda}{2}|w|\}$$

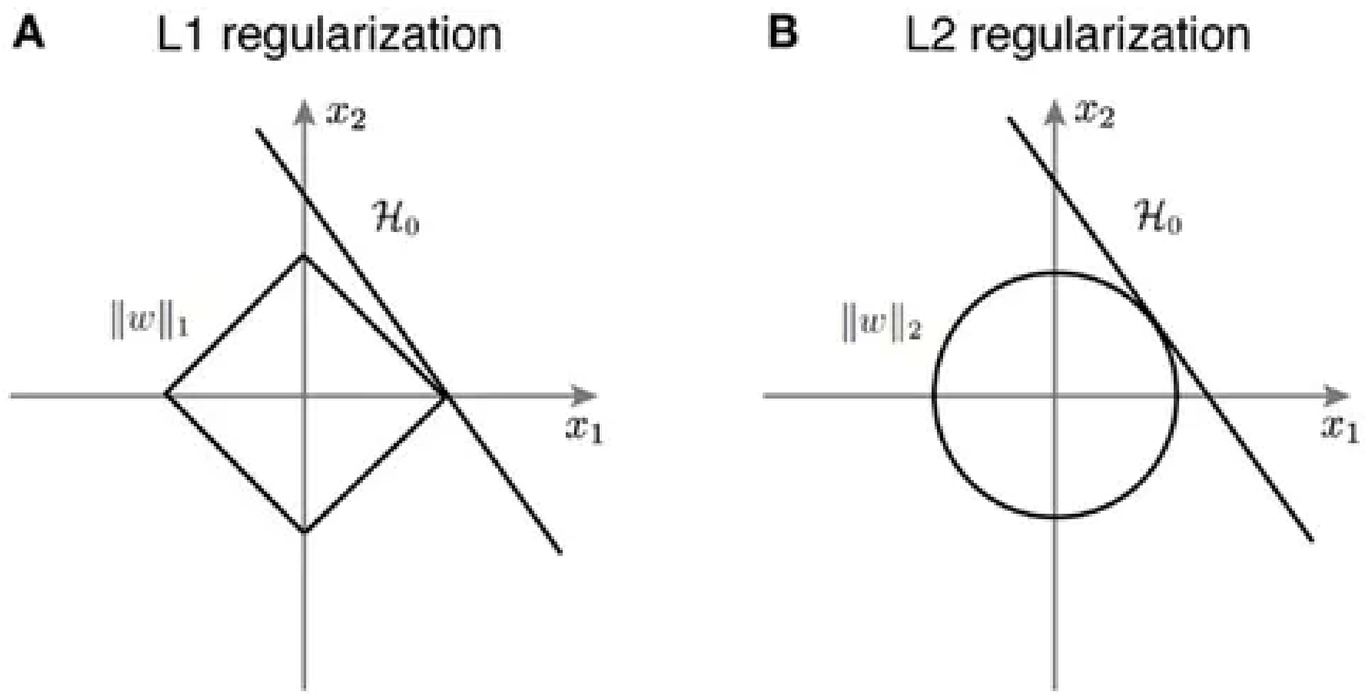
작은 가중치들이 거의 0으로 수렴하여 몇개의 중요한 가중치들만 남는 경향이 있어서 Sparse model에 적합

=> 컨벡스(Convex) 최적화에 유용하게 쓰임

예) Lasso Regression

2-2. 정규화(Regularization)

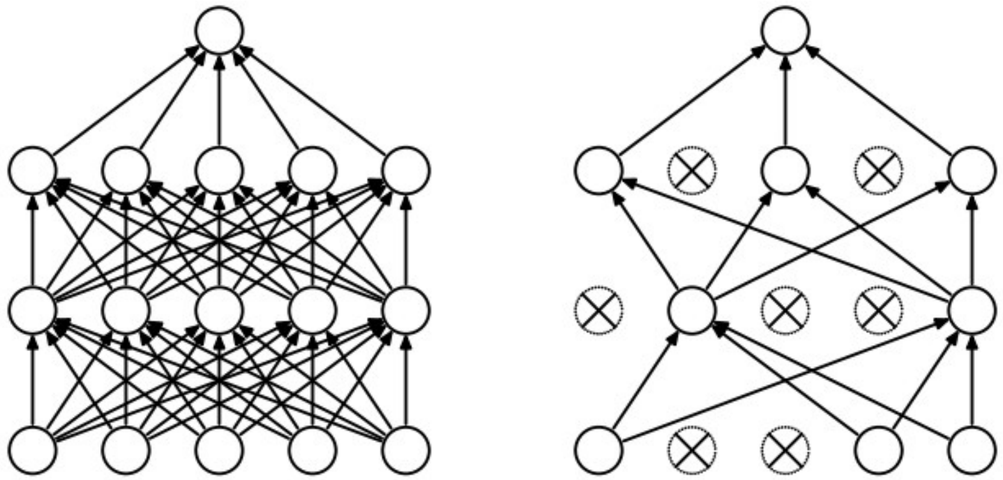
- 단, L1 Regularization 의 경우 그림처럼 미분 불가능한 점이 있기 때문에 Gradient-base learning에는 주의가 필요



2-3. 드랍아웃(Dropout)

드랍아웃(Dropout)

- 각 계층마다 일정 비율의 뉴런을 임의로 drop시켜 나머지 뉴런들만 학습하는 방법
- 드롭아웃을 적용하면 학습되는 노드와 가중치들이 매번 달라져, 과적합을 효과적으로 예방 (망 내부의 앙상블 학습으로 볼수 있음)



(a) Standard Neural Net

(b) After applying dropout.

2-3. 드랍아웃(Dropout)

드랍아웃(Dropout)

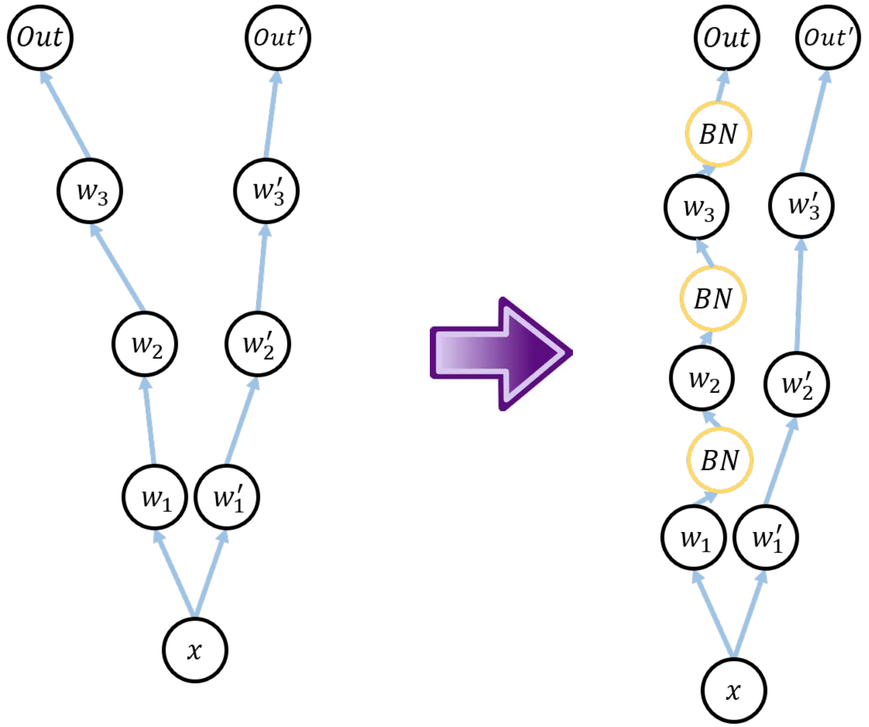
- 드롭아웃 비율은 은닉층 50%, 입력층 26% 정도가 일반적
- 다른 정규화 기법들과 상호 보완적으로 사용 가능
- 역전파는 ReLU처럼 동작

  순전파 때 신호를 통과시킨 뉴런은 역전파 때도 통과시키고,

  drop된 뉴런은 역전파 때도 신호를 차단
- Test 때는 모든 뉴런에 신호를 전달한다는 것에 주의

2-4. 배치 정규화(Batch Normalization)

- 학습시 현재 층의 입력은 모든 이전 층의 파라미터의 변화에 영향을 받음
- 망이 깊어짐에 따라 이전 층의 작은 파라미터 변화가 증폭되어 뒷단에 큰 영향을 끼칠수 있음



2-4. 배치 정규화(Batch Normalization)

- 학습하는 이전 층의 파라미터 변화로 현재 층의 입력의 분포가 바뀌는 현상을 내부 공변량 변화(Internal Covariate Shift)라고 함
- TV 오락 프로에서 귀마개를 하고 상대방의 입모양을 보고 무슨 말인지 알아내는 게임과 비슷함

2-4. 배치 정규화(Batch Normalization)

배치 정규화(BN, 2015)

- 기울기 소실, 기울기 폭발(Exploding)이 일어나지 않도록 한 대표적인 아이디어
- 활성화 함수 변경, Careful한 초기화, 작은 학습률 설정 등의 간접적인 방법이 아님
- 훈련 과정 자체를 안정화시켜 학습속도를 가속화한 근본적인 방법
- 배치 정규화(BN)는 평균과 분산을 조절하는 과정이 별도의 프로세스로 있는 것이 아니라 신경망 안에 포함되어 있다는 것이 가장 핵심적인 차이

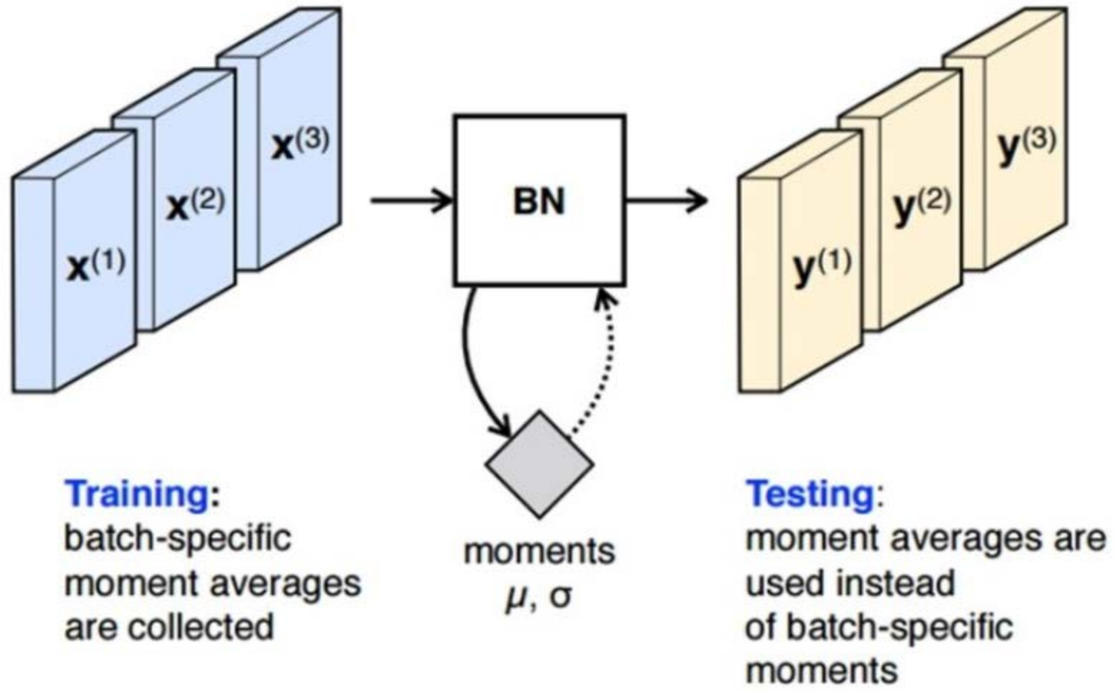
2-4. 배치 정규화(Batch Normalization)

- **Training 시**

  각 미니배치 마다 γ와 β를 구하고 저장해 둠

- **Test 시**

  구했던 γ와 β의 평균을 사용



참고자료

1. Data Augmentation
   1. https://nittaku.tistory.com/272
   2. (Keras 코드)
   https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/
   3. (OpenCV 코드)
   https://github.com/AISangam/Image-Augmentation-Using-OpenCV-and-Python
2. 과적합(Regularization, Dropout)
   1. https://umbum.tistory.com/222
   2. https://light-tree.tistory.com/125
3. 배치 정규화(BN)
   1. https://gomguard.tistory.com/186
   2. https://m.blog.naver.com/laonple/220808903260

/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

medium.com/elice