**Image Processing**

* **한번에 image dataset 올리는 방법**

import tensorflow as tf  
import cv2

from keras.preprocessing.image import ImageDataGenerator

#load data set  
data\_generator = ImageDataGenerator(rescale=1./255)  
train\_generator = data\_generator.flow\_from\_directory("C:\\Users\\nise\Desktop\\images\\train", target\_size=(200,200),  
 batch\_size=1, class\_mode='categorical')  
x\_train, y\_train = train\_generator.next()  
x\_train = x\_train.squeeze()  
y\_train=y\_train.squeeze()

Squeeze는 굳이 할 필요 x

* **Image Filtering**

이미지도 음성신호처럼 주파수로 표현할 수 있음. 일반적으로 고주파는 밝기의 변화가 많은 곳, 즉 경계선 영역에서 나타나고 저주파는 일반적인 배경에서 나타나게 됨. 따라서 고주파를 제거하면 Blur 처리가 되고, 저주파를 제거하면 대상의 영역을 확인할 수 있다(선명 해진다)

Low-Pass filter(LPF)와 High-Pass filter(HDF)를 이용하여 LDF를 적용하여 노이즈를 제거하거나 blur처리를 할 수 있으며, HPF를 적용하면 경계선을 찾을 수가 있다.

OpenCV에서는 cv2.filter2D() 함수를 이용하여 이미지의 kernel(filter)를 적용하여 이미지를 Filtering 할 수 있다. Kernel은 특정 행렬을 의미하는데 Kernel size가 크면 이미지 전체가 blur처리가 많이 됨.

* Filter가 적용되는 방법은
* 이미지의 각 pixel에 kernel을 적용함
* 각 pixel에 NxN 윈도우를 올려놓고, 그 영역안에 포함되는 값을 sum을 한 뒤에 1/N\*N을 곱해준다
* 그 결과는 해당 윈도우 영역안의 평균값이 되고, 그 값을 해당 pixel에 적용함

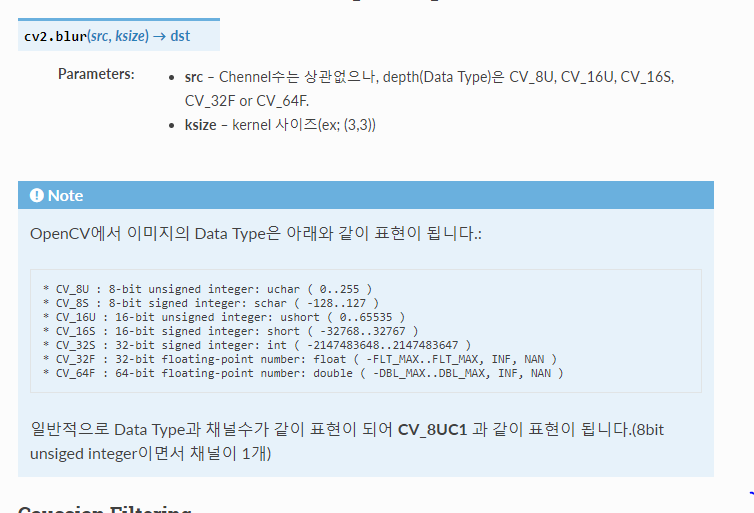


* **Image Blurring**

Image Blurring은 low-pass filter를 이미지에 적용하여 얻을 수 있음. 고주파영역을 제거함으로써 노이즈를 제거하거나 경계선을 흐리게 할 수 있음. OpenCV에는 4가지 형태의 blurring 방법을 제공함

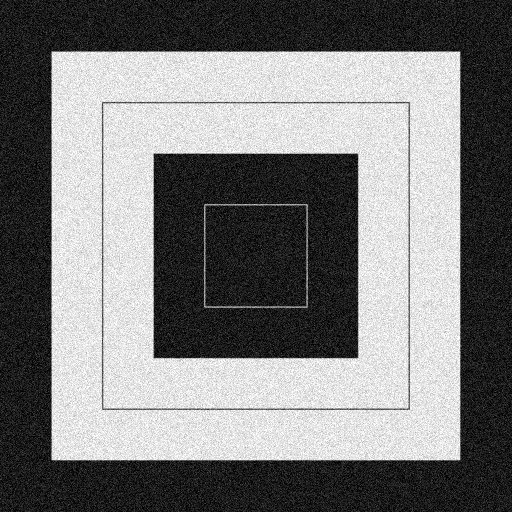
* Averaging

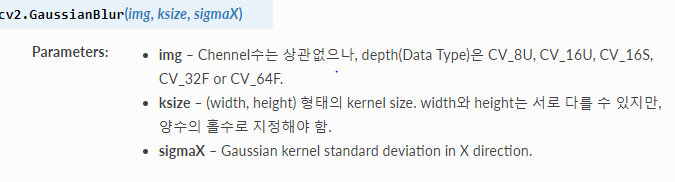
Box형태의 Kernel을 이미지에 적용한 후 평균값을 box의 중심점에 적용하는 형태임. 이는 cv2.blur() 또는 cv2.boxFilter() 함수를 통해 구현 가능



* Gaussian Filtering

Box filter는 동일한 값으로 구성된 kernel을 사용하지만, Gaussian Filter는 Gaussian 함수를 이용한 Kernel을 적용함. 즉, Kernel 행렬의 값을 Gaussian 함수를 통해서 수학적으로 생성하여 적용함. Kernel의 사이즈는 양수이며 홀수로 지정해야 함. 이미지가 가지는 Gaussian Noise(전체적으로 밀도가 동일한 노이즈, 백색 노이즈)를 제거하는데 효과적



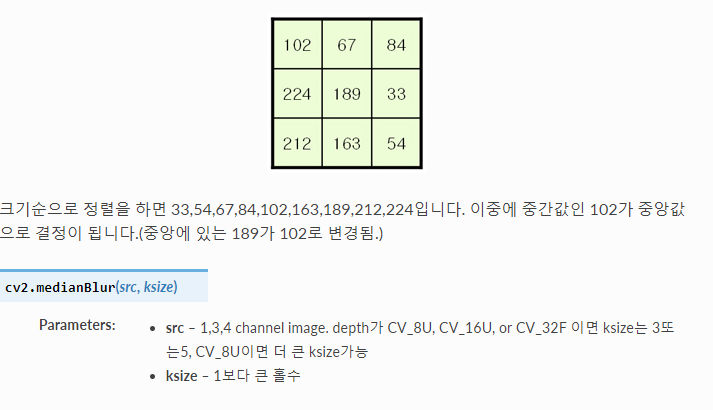


* Median Filtering

Kernel window와 pixel의 값들을 정렬한 후에 중간 값을 선택하여 적용함. Salt-and-pepper noise(점 잡음) 제거에 매우 유용함



EX)



* Bilateral Filtering

위의 Blur처리는 경계선까지 Blur처리가 되어, 경계선이 흐려지게 됨. 이 양방향 필터는 경계선을 유지하면서 Gaussian Blur처리를 해주는 방법임



dst\_x\_0=cv2.bilateralFilter(x\_train,9,75,75)  
dst\_y\_0=cv2.bilateralFilter(y\_train,9,75,75)  
  
  
  
print(x\_train[0].shape)  
plt.imshow(dst\_x\_0)  
cv2.waitKey(0)  
cv2.destroyAllWindows()  
plt.show()

import os  
from datetime import datetime  
from time import sleep  
import numpy as np  
import tensorflow as tf  
import cv2  
from constants import \*  
from imageai.Detection import ObjectDetection  
from keras.preprocessing.image import ImageDataGenerator  
  
  
#Emotion labeling  
Emotion = ['angry', 'disgusted', 'fearful', 'happy', 'sad', 'surprised', 'neutral']  
  
#load data set  
data\_generator = ImageDataGenerator(rescale=1./255)  
train\_generator = data\_generator.flow\_from\_directory("C:\\Users\\nise\Desktop\\images\\train", target\_size=(200,200),  
 batch\_size=1, class\_mode='categorical')  
x\_train, y\_train = train\_generator.next()  
x\_train = x\_train.squeeze()  
y\_train=y\_train.squeeze()  
  
  
#BilateralBlur  
dst\_x\_0=cv2.bilateralFilter(x\_train,9,75,75)  
dst\_y\_0=cv2.bilateralFilter(y\_train,9,75,75)  
  
#------------------------------------------------------------- 다시 설계해  
  
x\_imge=tf.reshape(dst\_x\_0, [-1,200,200,1])  
  
w1= tf.Variable(tf.random\_normal([5,5,1,32], stddev=0.01))  
b1=tf.Variable(tf.random\_normal([32]))  
h1=tf.nn.conv2d(x\_imge,w1, strides=[1,1,1,1], padding='SAME')+b1  
l1=tf.nn.relu(h1)  
l1=tf.nn.max\_pool(l1, ksize=[1,2,2,1],  
 strides=[1,2,2,1], padding='SAME')  
l1=tf.nn.dropout(l1, keep\_prob=0.5)  
  
w2=tf.Variable(tf.random\_normal([5,5,32,64], stddev=0.01))  
b2=tf.Variable(tf.random\_normal([64]))  
h2=tf.nn.conv2d(l1,w2, strides=[1,1,1,1], padding='SAME')+b2  
l2=tf.nn.relu(h2)  
l2=tf.nn.max\_pool(l2, ksize=[1,2,2,1],  
 strides=[1,2,2,1], padding='SAME')  
l2=tf.nn.dropout(l2, keep\_prob=0.5 )  
  
w3=tf.Variable(tf.random\_normal([5,5,64,128], stddev=0.01))  
b3=tf.Variable(tf.random\_normal([128]))  
h3=tf.nn.conv2d(l2,w3, strides=[1,1,1,1], padding='SAME')+b3  
l3=tf.nn.relu(h3)  
l3=tf.nn.max\_pool(l3,ksize=[1,2,2,1],  
 strides=[1,2,2,1], padding='SAME')  
l3=tf.nn.dropout(l3,keep\_prob=0.5)  
  
w4=tf.Variable(tf.random\_normal([5,5,128,256], stddev=0.01))  
b4=tf.Variable(tf.random\_normal([256]))  
h4=tf.nn.conv2d(l3,w4, strides=[1,1,1,1], padding='SAME')+b4  
l4=tf.nn.relu(h4)  
l4=tf.nn.max\_pool(l4, ksize=[1,2,2,1],  
 strides=[1,2,2,1], padding='SAME')  
l4=tf.nn.dropout(l4, keep\_prob=0.5)  
  
w5=tf.Variable(tf.random\_normal([5,5,256,512], stddev=0.01))  
b5=tf.Variable(tf.random\_normal([512]))  
h5=tf.nn.conv2d(l4,w5, strides=[1,1,1,1], padding='SAME')  
l5=tf.nn.relu(h5)  
l5=tf.nn.max\_pool(l5, ksize=[1,2,2,1],  
 strides=[1,1,1,1], padding='SAME')  
l5=tf.nn.dropout(l5, keep\_prob=0.5)  
  
 #fully connected layer  
l5=tf.reshape(l5, [-1, 3\*13\*13\*512])  
w\_fc=tf.get\_variable("w\_fc",[3\*13\*13\*512,7], initializer=tf.contrib.layers.xavier\_initializer())  
b\_fc=tf.Variable(tf.random\_normal([7]))  
hypothesis=tf.matmul(l5, w\_fc)+b\_fc  
  
#--------------------------------------------------------------------------------------------------------  
#X,Y  
train\_steps = 32000  
  
X=tf.placeholder(tf.float32,[None, 28821])  
Y=tf.placeholder(tf.float32,[None,7])  
batch\_size=50  
learning\_rate=0.01  
  
cost=tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=hypothesis, labels=Y))  
optimizer=tf.train.AdamOptimizer(learning\_rate).minimize(cost)  
correct\_prediction = tf.equal(tf.argmax(hypothesis,1),tf.argmax(Y,1))  
accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,tf.float32))