# Deeplearning

# Azure

Face detection, emotion, verify …

<https://azure.microsoft.com/en-in/services/cognitive-services/face/>

# Convolution Filter

import cv2

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

def crop\_center(img,cropx,cropy):

y,x = img.shape

startx = x//2-(cropx//2)

starty = y//2-(cropy//2)

return img[starty:starty+cropy,startx:startx+cropx]

def convolution(image, kernel):

Im, In = image.shape

Km, Kn = kernal.shape

output = np.zeros\_like(image)

for y in range(In - Kn + 1):

for x in range(Im - Km + 1):

output[y,x] = (kernel\*image[y:y+Km, x:x+Kn]).sum()

return output

# Read image

img = cv2.imread("image.jpg", 0)

img = crop\_center(img, 160, 160)

plt.imshow(img)

# Read kernal

kernel = np.array([[1, 0, -1],

[0, 0, 0],

[-1, 0, 1]])

# Add padding to img

padd\_img = np.pad(img, ((1,1),(1,1)), 'constant')

# Apply convolution

kernel = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])

edge\_kernel = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])

blur\_kernel = np.array([[1,1,1],[1,1,1],[1,1,1]])/9.0;

output = convolution(padd\_img,blur\_kernel)

plt.imshow(output)

plt.show()

# CNN only using Numpy

<https://www.kdnuggets.com/2018/04/building-convolutional-neural-network-numpy-scratch.html>

1. Reading the input image.
2. Preparing filters.
3. Conv layer: Convolving each filter with the input image.
4. ReLU layer: Applying ReLU activation function on the feature maps (output of conv layer).
5. Max Pooling layer: Applying the pooling operation on the output of ReLU layer.
6. Stacking conv, ReLU, and max pooling layers.
7. Reading input image

import skimage.data

# Reading the image

img = skimage.data.chelsea()

# Converting the image into gray.

img = skimage.color.rgb2gray(img)

1. Preparing filters

The following code prepares the filters bank for the first conv layer (l1 for short):

The values of the kernel filters are learned automatically by the neural network through the training process, and the filters kernels which results in the features that are most efficient for the particular classification or the detection are automatically learned.

l1\_filter = numpy.zeros((2,3,3)) # (2=num\_filters, 3=num\_rows\_filter, 3=num\_columns\_filter)

# If the image is RGB with 3 channels, the filter size must be (2, 3, 3=depth).

The size of the filters bank is specified by the above zero array but not the actual values of the filters. It is possible to override such values as follows to detect vertical and horizontal edges.

l1\_filter[0, :, :] = numpy.array([[[-1, 0, 1],

[-1, 0, 1],

[-1, 0, 1]]])

l1\_filter[1, :, :] = numpy.array([[[1, 1, 1],

[0, 0, 0],

[-1, -1, -1]]])

3. Conv Layer

def conv(img, conv\_filter):

if len(img.shape) > 2 or len(conv\_filter.shape) > 3: # Check if number of image channels matches the filter depth.

if img.shape[-1] != conv\_filter.shape[-1]:

print("Error: Number of channels in both image and filter must match.")

sys.exit()

if conv\_filter.shape[1] != conv\_filter.shape[2]: # Check if filter dimensions are equal.

print('Error: Filter must be a square matrix. I.e. number of rows and columns must match.')

sys.exit()

if conv\_filter.shape[1]%2==0: # Check if filter diemnsions are odd.

print('Error: Filter must have an odd size. I.e. number of rows and columns must be odd.')

sys.exit()

# An empty feature map to hold the output of convolving the filter(s) with the image.

feature\_maps = np.zeros((img.shape[0]-conv\_filter.shape[1]+1,

img.shape[1]-conv\_filter.shape[1]+1,

conv\_filter.shape[0]))

# Convolving the image by the filter(s).

for filter\_num in range(conv\_filter.shape[0]):

print("Filter ", filter\_num + 1)

curr\_filter = conv\_filter[filter\_num, :] # getting a filter from the bank.

"""

Checking if there are mutliple channels for the single filter.

If so, then each channel will convolve the image.

The result of all convolutions are summed to return a single feature map.

"""

if len(curr\_filter.shape) > 2:

conv\_map = conv\_(img[:, :, 0], curr\_filter[:, :, 0]) # Array holding the sum of all feature maps.

for ch\_num in range(1, curr\_filter.shape[-1]): # Convolving each channel with the image and summing the results.

conv\_map = conv\_map + conv\_(img[:, :, ch\_num],

curr\_filter[:, :, ch\_num])

else: # There is just a single channel in the filter.

conv\_map = conv\_(img, curr\_filter)

feature\_maps[:, :, filter\_num] = conv\_map # Holding feature map with the current filter.

return feature\_maps # Returning all feature maps.

def conv\_(img, conv\_filter):

filter\_size = conv\_filter.shape[0]

result = np.zeros((img.shape))

#Looping through the image to apply the convolution operation.

for r in np.uint16(np.arange(filter\_size/2,

img.shape[0]-filter\_size/2-2)):

for c in np.uint16(np.arange(filter\_size/2, img.shape[1]-filter\_size/2-2)):

#Getting the current region to get multiplied with the filter.

curr\_region = img[r:r+filter\_size, c:c+filter\_size]

#Element-wise multipliplication between the current region and the filter.

curr\_result = curr\_region \* conv\_filter

conv\_sum = np.sum(curr\_result) #Summing the result of multiplication.

result[r, c] = conv\_sum #Saving the summation in the convolution layer feature map.

#Clipping the outliers of the result matrix.

final\_result = result[np.uint16(filter\_size/2):result.shape[0]-np.uint16(filter\_size/2),

np.uint16(filter\_size/2):result.shape[1]-np.uint16(filter\_size/2)]

return final\_result

l1\_feature\_map = conv(img, l1\_filter)

4. ReLU Layer

def relu(feature\_map):

#Preparing the output of the ReLU activation function.

relu\_out = np.zeros(feature\_map.shape)

for map\_num in range(feature\_map.shape[-1]):

for r in np.arange(0,feature\_map.shape[0]):

for c in np.arange(0, feature\_map.shape[1]):

relu\_out[r, c, map\_num] = np.max(feature\_map[r, c, map\_num], 0)

return relu\_out

l1\_feature\_map\_relu = relu(l1\_feature\_map)

5. Max Pooling Layer

def pooling(feature\_map, size=2, stride=2):

#Preparing the output of the pooling operation.

pool\_out = np.zeros((np.uint16((feature\_map.shape[0]-size+1)/stride),

np.uint16((feature\_map.shape[1]-size+1)/stride),

feature\_map.shape[-1]))

for map\_num in range(feature\_map.shape[-1]):

r2 = 0

for r in np.arange(0,feature\_map.shape[0]-size-1, stride):

c2 = 0

for c in np.arange(0, feature\_map.shape[1]-size-1, stride):

pool\_out[r2, c2, map\_num] = np.max(feature\_map[r:r+size, c:c+size])

c2 = c2 + 1

r2 = r2 +1

return pool\_out

l1\_feature\_map\_relu\_pool = pooling(l1\_feature\_map\_relu, 2, 2)

6. Stacking Layers

Up to this point, the CNN architecture with conv, ReLU, and max pooling layers is complete. There might be some other layers to be stacked in addition to the previous ones as below.

# Second conv layer

l2\_filter = np.random.rand(3, 5, 5, l1\_feature\_map\_relu\_pool.shape[-1])

print("\n\*\*Working with conv layer 2\*\*")

l2\_feature\_map = conv(l1\_feature\_map\_relu\_pool, l2\_filter)

print("\n\*\*ReLU\*\*")

l2\_feature\_map\_relu = relu(l2\_feature\_map)

print("\n\*\*Pooling\*\*")

l2\_feature\_map\_relu\_pool = pooling(l2\_feature\_map\_relu, 2, 2)

print("\*\*End of conv layer 2\*\*\n")

# Reference

<https://deeplizard.com/learn/playlist/PLZbbT5o_s2xrfNyHZsM6ufI0iZENK9xgG>