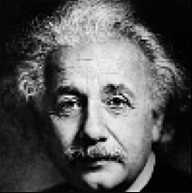


**Mid Term Project Report**

**Introduction to Computer Vision 2020**

**Daniel Ivkovich: 316421262**

**15/05/2020**





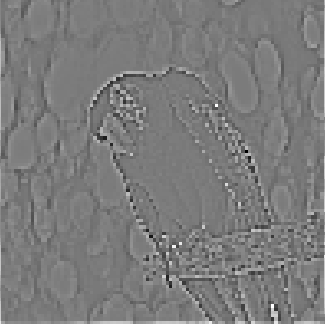
**Abstract**

Convolutional Networks are based on the correlation operation (convolution without 180 degrees rotation of the filter kernel).

In this work I have wrote and used the basic convolution operation, including Stride, Padding and normalized cross correlation.

The objective is to create the forward path of a convolutional layer (activation map), by adding to the convolution (correlation) Stride, Padding and Normalization (Normalized Cross Correlation to enable template matching), followed by an activation non linearity. Then to apply multiple kernels to a single input image / activation map.

The output image should have the correct size / dimensions, based in the input size and hyper parameters and the operation will work on all types of images: grey level / RGB / Float / UINT8.



**Introduction**

Motivation: The Convolution/Correlation function is a very important feature in Computer Vision. It constitutes the basis for a Convolutional Neural Network which is used to analyze, categorize and create images.

It is very important to implement the Convolution in a precise manner because inside a neural network we can't always keep track on what's going on and it is important to forward correct and precise information forward.

We can use the function in order to extract vital information about an image for future use, this information is called features and we can do this while ignoring the things we don't want in the image.

These Features will then move forward in the Neural network and will be used to do what the network does.

We can also use the Convolution/Correlation function in order to find specific patches of information in a picture, we can use a small sub-image as the Filter to which we will use on the original image using a special normalization called Normalized cross validation – which is a form of similarity between the filter and the original image in each iteration.

The Convolution will result in a new image with higher frequencies in the most matching areas. This output can be threshholded and we will be left with white dots in the areas that Match the pattern almost correctly or prefectly.

scipy has a function called conv2d() which calculates the Convolution and a function called correlate2d() which calculates the Correlation.

These functions do not work on all types of images and need improvement working with RGB Float / RGB UINT8.

Moreover, they do not have the option to use Normalized Cross Correlation which I have Implemented in this project.

Hyperparameters include: stride, padding, Norm, ACTV.

stride and padding influence the output size by the following template:

**H1 = (H – F + 2\*p)//stride + 1.**

**W1 = (W – F + 2\*p)//stride + 1.**

**Proposed Solution**

**Stage One:**

Define shapes and Output Dimention according. ( W1, H1, N)

Check the input shapes and deal with them accordingly.

If they are GRAYSCALE then add a third axis to them.

**Stage Two:**

if the boolean value CORR is False: flip all kernels by 180 degrees.

**Stage Three:**

Calculate N times:

Convolve Each channel of the Image with the corresponding channel of the filter, taking into account the new hyper-parameter Norm.

If Norm is activated the function will Normilize Cross corelate every patch in each iteration.

If the ACTV is present in the input as another hyper-parameter the function shall then change the output according to the activation function presented.

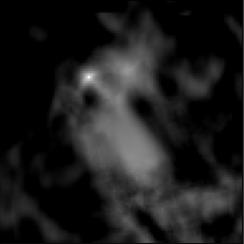
In Each Channel Convolution between the Image and the Filter the output will be of shape (W1, H1, 1). After performing them all and adding to a numpy array the result Output will be (W1, H1, N).

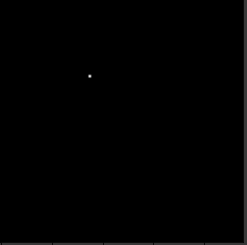
Note: if we wish to find a patch in an image we will use

norm= normalized\_cross\_correlation as an hyperparameter and pass it through a threshhold.

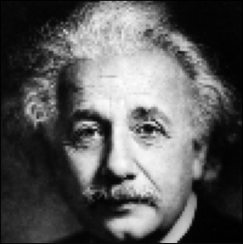
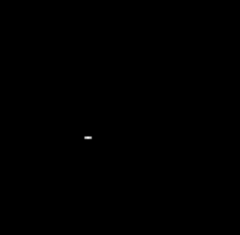
**Results**

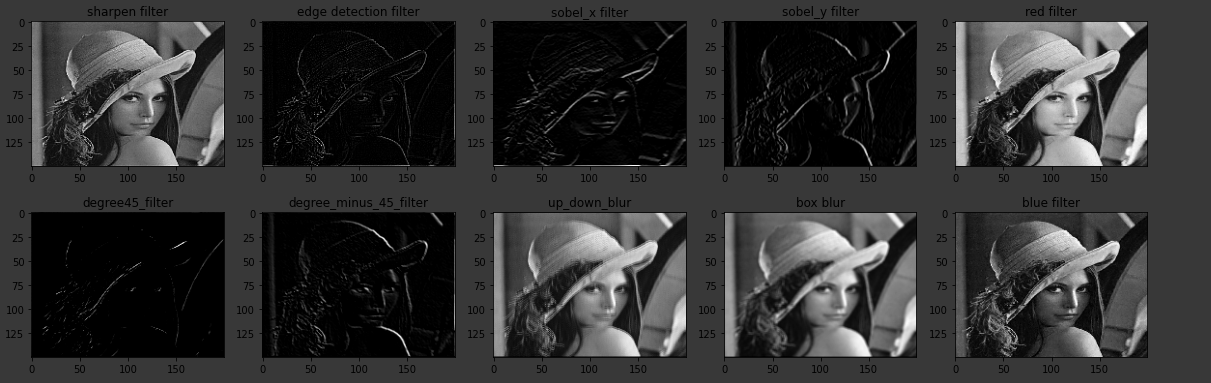
Detect the parrot Eye using normalized cross validation:



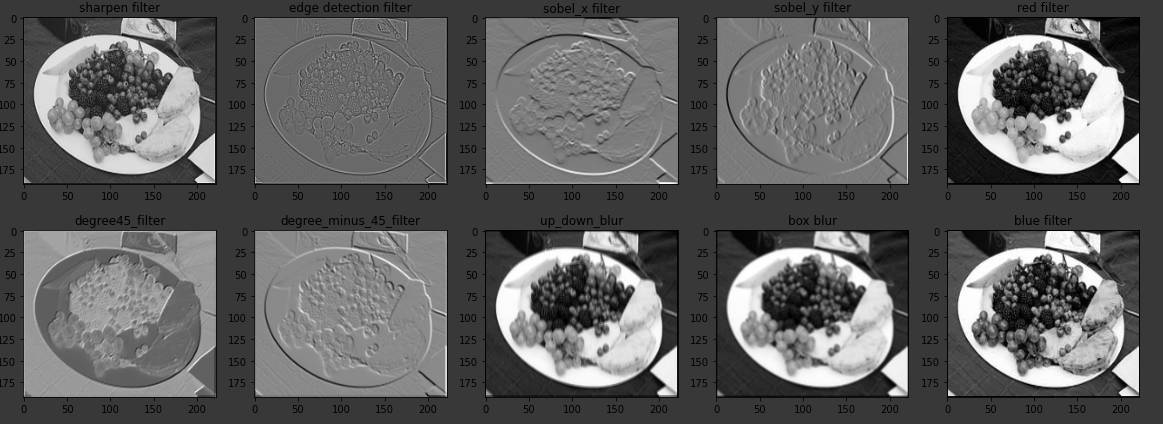


Detect the Einstein's Eye using normalized cross validation:

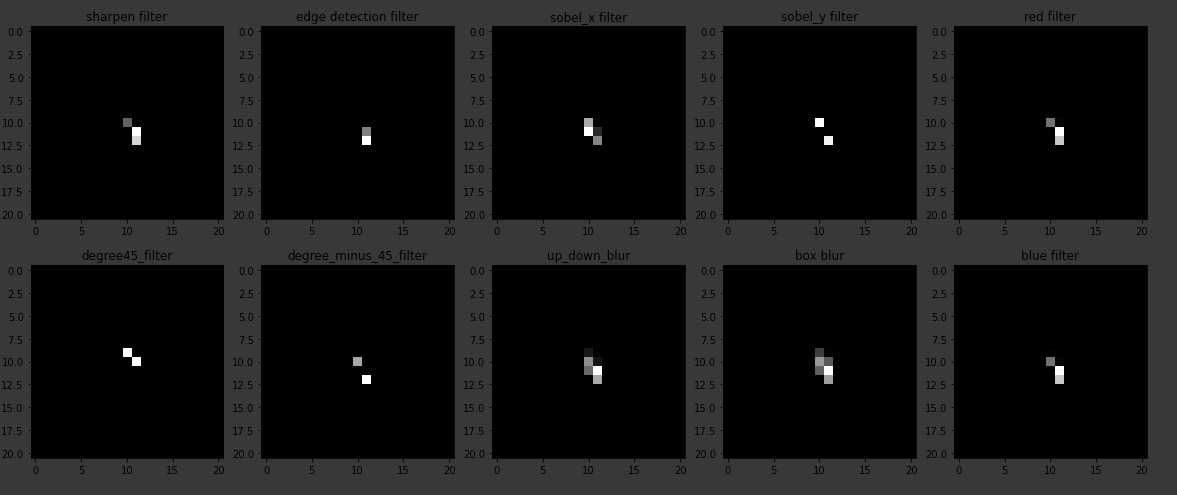


10 Filters on Lena:

10 Filters on ImageNet grapes Images:



An Example of Bad Hyper-parameters for the Mnist Digit:



**Summary And Conclusion**

In conclusion to this project, the problem was to create an improved and argument versatile Convolution/Correlation function and to create a basic Convolutional Layer resulting with better feature extraction.

In addition I have imlemented a RelU activation function as well as a normalized cross validation option to the myconv function which allows the possibility to search for objects in the image and detecting their location in the image after thresholding the normalized image.

I have also proved that my convolution works precisely in the following check:

I have convolved an original image with 3x3x3, 5x5x3, 7x7x3 identity filters with the corresponding hyper parameter P = 1, 2, 3 for each filter size to get the exact same image dimentions as the input of the original image.

I have gotten a difference of **zero** between each convolved image and the original image, showing that the convolution is precise.

Using the convolution I have cascaded 2 layers of the Afeka Logo and thus combined 2 types of filters in a row resulting in more complex feature selection.

I have also Found Einstein's eye using the normalized cross validation as shown in the results section by convolving the image with an 15x15 eye patch of the original image.

All of the things I have implemented in this project will surely be used in the near future to implement more complex Convolutional Neural Networks for better feature extraction and better results.

Thanks for reading,

Daniel Ivkovich.

**CODE IN READABLE FORMAT**

ignore this if already read.

**Imports**[**¶**](file:///C:\Users\unact\Downloads\MidTerm_Daniel_Ivkovich.html#Imports)

In [0]:

**import** **numpy** **as** **np**

**import** **math**

**from** **matplotlib** **import** pyplot **as** plt

**import** **cv2**

**from** **urllib.request** **import** urlopen

**Basic Image Converts and Imshow**

In [0]:

**class** **Image**(object):

@staticmethod

**def** show(img, title='image'):

plt.imshow(img, cmap='gray')

plt.show()

@staticmethod

**def** show\_all(image\_list, title\_list):

plt.figure(figsize=[20, 20])

**assert** len(image\_list) == len(title\_list), "Houston we've got a problem"

N = len(image\_list)

**for** index, (img, title) **in** enumerate(zip(image\_list, title\_list)):

plt.subplot(1, N, index+1)

**if** len(img.shape) != 3:

plt.imshow(img, cmap='gray')

**else**:

plt.imshow(img)

plt.title(title)

plt.show()

**Defining Helper Functions for myconv and Normalized Cross Validation**

In [0]:

**def** show\_after\_conv(convolved\_, labels):

convolved\_gray = [norm\_abs(convolved\_[:, :, i]) **for** i **in** range(convolved\_.shape[2])]

Image.show\_all(convolved\_gray, labels)

**def** equals(img1, img2):

img1 = normUINT8(img1)

img2 = normUINT8(img2)

boolx = np.linalg.norm((img1 - img2))

**if** boolx == 0:

print('The Images are equal')

**else**:

print('The Images are NOT equal')

**def** threshhold(img, thresh):

result = np.copy(img)

**for** i **in** range(img.shape[0]):

**for** j **in** range(img.shape[1]):

**if** img[i][j] < thresh:

result[i][j] = 0

**return** result

**def** norm(img):

*'''*

*converts to Float and between [0, 1]*

*'''*

img = img.astype(float)

img -= img.min()

max\_val = img.max()

**if** max\_val > 0:

img /= img.max()

**return** img

**def** normUINT8(img):

img = img.astype(float)

img -= img.min()

img /= img.max()

img \*= 255

**return** img.astype('uint8')

**def** read\_image\_from\_link(link, size):

data = urlopen(link).read()

data = np.frombuffer(data, np.uint8)

rv = cv2.imdecode(data, cv2.IMREAD\_COLOR)

rv = cv2.cvtColor(rv, cv2.COLOR\_BGR2RGB)

rv = cv2.resize(rv, (size[0],size[1]))

**return** rv

**def** relu(img):

**for** i **in** range(img.shape[0]):

**for** j **in** range(img.shape[1]):

**if** img[i][j] < 0:

img[i][j] = 0

**def** norm\_abs(img):

**return** (img / np.max(np.abs(img)))

**def** normalized\_cross\_validation(filt, patch):

F = filt.shape[0]

C = filt.shape[2]

**assert** F == patch.shape[0], 'Filter is: ' + str(filt.shape) +' != ' + 'sub Image is: ' + str(patch.shape)

**assert** F == patch.shape[1], 'Filter is: ' + str(filt.shape) +' != ' + 'sub Image is: ' + str(patch.shape)

patch\_mean = np.mean(patch)

filter\_mean = np.mean(filt)

patch\_mean\_substracted = patch-patch\_mean

new\_filter\_mean\_substracted = filt-filter\_mean

squared\_sum\_patch = 0

squared\_sum\_filter = 0

**for** i **in** range(F):

**for** j **in** range(F):

**for** c **in** range(C):

squared\_sum\_patch += (patch\_mean\_substracted[i][j][c]\*\*2)

squared\_sum\_filter += (new\_filter\_mean\_substracted[i][j][c]\*\*2)

total\_div = math.sqrt(squared\_sum\_patch \* squared\_sum\_filter)

normalized\_patch = patch\_mean\_substracted/total\_div

**return** normalized\_patch

**def** filter\_iteration(filt, patch):

*"""*

*Filter = FxFxC*

*patch = FxFxC*

*returns the sum of the Iteration.*

*"""*

F = filt.shape[0]

C = filt.shape[2]

**assert** F == patch.shape[0], 'Filter is: ' + str(filt.shape) +' != ' + 'sub Image is: ' + str(patch.shape)

**assert** F == patch.shape[1], 'Filter is: ' + str(filt.shape) +' != ' + 'sub Image is: ' + str(patch.shape)

result = 0.0

**for** i **in** range(F):

**for** j **in** range(F):

result += (filt[i, j, :] \* patch[i, j, :]).sum()

**return** result

**Defining myconv Function**

In [0]:

**def** myconv(img, ker, stride=1, p=0, N=1, Norm=**None**, CORR=**False**, ACTV=**None**):

*"""*

*ker = FxFxC*

*img = WxHxC*

*output = W1xH1x1*

*RETURNS outputs = W1xH1xN*

*"""*

ker = ker.astype(float)

**if** len(img.shape) == 2:

img = img[:, :, np.newaxis]

**if** len(ker.shape) == 2:

ker = ker[:, :, np.newaxis]

F = ker.shape[0] *# kernel dimentions are FxF*

C = ker.shape[2] *# Filter channels*

W = img.shape[0]

H = img.shape[1]

H1 = (H - F + 2\*p)//stride + 1

W1 = (W - F + 2\*p)//stride + 1

outputs = np.zeros((W1, H1, N))

*# pad image with mirror mode.*

PADDED\_IMG = np.pad(norm(img), pad\_width=((p, p), (p, p), (0,0)), mode='constant', constant\_values=0)

*# Flip Kernels if Convolution mode is enabled.*

**if** **not** CORR:

**for** c **in** range(C):

kernel = ker[:,:, c]

kernel = np.flip(np.flip(kernel, axis=0),axis=1)

*# Apply N filters on the Image.*

**for** i **in** range(N):

output\_shape = (W1, H1)

output = np.zeros(output\_shape) *# (W1,H1,C)*

width\_end = W - (F//2 + 1) + 2\*p - (F//2 - 1)

height\_end = H - (F//2 + 1) + 2\*p - (F//2 - 1)

*# Convolution Operation Loops*

**for** y **in** range(0, width\_end, stride):

**for** x **in** range(0, height\_end, stride):

start\_y = y

end\_y = start\_y + F

start\_x = x

end\_x = start\_x + F

sub\_image = PADDED\_IMG[start\_y:end\_y, start\_x:end\_x, :] *# (F, F, C)*

**if** Norm **is** **not** **None**:

sub\_image = normalized\_cross\_validation(ker, sub\_image)

result\_pixel = filter\_iteration(ker, sub\_image) *# (F, F, C) , # (F, F, C)*

output[y//stride][x//stride] = result\_pixel *# shape (1) being put W1,H1 times. total: (W1, H1, 1)*

**if** ACTV **is** **not** **None**:

ACTV(output) *# Now a single output is W1,H1,1*

outputs[:, :, i] = output

**return** outputs

**Get Desired Images from the Web**

In [0]:

lena\_url = 'https://www.researchgate.net/profile/Adelrahman\_Idries/publication/232815017/figure/fig3/AS:300463506378754@1448647490218/Original-Lena-image.png'

afeka\_url = 'https://www.afeka.ac.il/images/default.jpg'

mnist\_url = 'https://datawrangling.s3.amazonaws.com/sample\_digit.png'

imageNet\_url = 'http://www.image-net.org/nodes/4/12492106/d2/d2b59602c09e7a5931b8cc2defe738211e015e73.thumb'

waldo\_url = 'https://static.boredpanda.com/blog/wp-content/uploads/2020/03/where-is-waldo-coronavirus-edition-book-fb26.png'

parrot\_url = 'https://spca.bc.ca/wp-content/uploads/bird-parrot-scarlet-macaw-on-branch-in-wild.jpg'

albert\_url = 'https://www.biography.com/.image/t\_share/MTE5NDg0MDU0OTU2OTAxOTAz/albert-einstein-9285408-1-402.jpg'

grapes\_url = 'http://www.image-net.org/nodes/4/07745940/25/256dac393e9ac7e52d236d83f7f90fc52711756d.thumb'

grapes2\_url = 'http://www.image-net.org/nodes/8/07744811/9a/9a6e15cc9f8903a9d6d663a6f156d9a17fb8a6b0.thumb'

grapes3\_url = 'http://www.image-net.org/nodes/8/07744811/b9/b90c1d496a9957103f1234480f3ed0bbf30c5a59.thumb'

grapes4\_url = 'http://www.image-net.org/nodes/8/07744811/ba/bae4759e81d998b5145d6979d6d47e7e75b48181.thumb'

In [170]:

lena\_img = read\_image\_from\_link(link=lena\_url, size=(200, 150))

afeka\_img = read\_image\_from\_link(link=afeka\_url, size=(400, 200))

mnist\_img = read\_image\_from\_link(link=mnist\_url, size=(28, 28))

imageNet\_img = read\_image\_from\_link(link=imageNet\_url, size=(200, 250))

waldo\_img = read\_image\_from\_link(link=waldo\_url, size=(150, 150))

albert\_img = read\_image\_from\_link(link=albert\_url, size=(100, 100))

parrot\_img = read\_image\_from\_link(link=parrot\_url, size=(100,100))

grapes\_img = read\_image\_from\_link(link=grapes\_url, size=(250,300))

grapes2\_img = read\_image\_from\_link(link=grapes2\_url, size=(220,190))

grapes3\_img = read\_image\_from\_link(link=grapes3\_url, size=(250,200))

grapes4\_img = read\_image\_from\_link(link=grapes4\_url, size=(200,240))

imgs = [lena\_img, afeka\_img, mnist\_img, imageNet\_img, waldo\_img, albert\_img, parrot\_img, grapes\_img, grapes2\_img, grapes3\_img, grapes4\_img]

Image.show\_all(imgs, ['' **for** i **in** range(len(imgs))])



**Define Convolutional Layer using myconv function**

In [0]:

**class** **Convolution\_layer**():

**def** \_\_init\_\_(self, data, ker, s, p, n, norm=**None**, actv=**None**):

*'''*

*FILTER = FxFxC*

*INPUT = HxWxC*

*NORM = None or normalized cross validation*

*ACTV = None or RelU*

*'''*

self.data = data

self.ker = ker

self.s = s

self.p = p

self.n = n

self.norm = norm

self.actv = actv

**def** forward(self):

*'''*

*RETURNS output of shape: W1xH1xN*

*'''*

**return** myconv(img=self.data, ker=self.ker, stride=self.s, p=self.p, N=self.n, Norm=self.norm, ACTV=self.actv)

**Kernels and Filters for later**

In [0]:

*""" KERNELS AND FILTERS"""*

*# kernels FxF*

nothing3D = np.array([[0, 0, 0],[0, 1, 0], [0, 0, 0]])

nothing5D = np.zeros((5,5))

nothing5D[2][2] = 1

nothing7D = np.zeros((7,7))

nothing7D[3][3] = 1

sobel\_x = np.array([[1, 2, 1],[0, 0, 0],[-1, -2, -1]])/5

sobel\_y = np.array([[1, 0, -1],[2, 0, -2],[1, 0, -1]])/5

blur = np.array([[1, 1, 1],[1, 1, 1],[1, 1, 1]])/9

blur\_filter = np.ones((3, 3, 3))/27

sharpen = np.array([[-1/9,-1/9,-1/9],[-1/9,3,-1/9],[-1/9,-1/9,-1/9]])

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

up\_down = np.array([[0, 1, 0],[0, 0, 0],[0, 1, 0]])

degree45\_line\_detector = np.array([[0, -2, -1],

[2, 0, -2],

[1, 2, 0]])/5

degree\_minus45\_line\_detector = np.array([[1, 2, 0],

[2, 0, -2],

[0, -2, 1]])/(-5)

red\_kernel = np.zeros((3,3))

blue\_kernel = np.zeros((3,3))

*# Filters FxFx3 and FxFx5 and FxFx7*

blur\_5D = np.ones((3, 3, 5))

identity\_5D = np.zeros((3, 3, 5))

identity\_5D[1][1][:] = np.ones((5))

degree\_minus\_45\_filter = np.repeat(degree45\_line\_detector[:, :, np.newaxis], 3, axis=2)

degree45\_filter = np.repeat(degree\_minus45\_line\_detector[:, :, np.newaxis], 3, axis=2)

red\_filter = np.repeat(red\_kernel[:, :, np.newaxis], 3, axis=2)

red\_filter[1][1][0] = 1

blue\_filter = np.repeat(blue\_kernel[:, :, np.newaxis], 3, axis=2)

blue\_filter[1][1][2] = 1

identity = np.repeat(nothing3D[:, :, np.newaxis], 3, axis=2)

up\_down\_filter = np.repeat(up\_down[:, :, np.newaxis], 3, axis=2)

sharpen\_filter = np.repeat(sharpen[:, :, np.newaxis], 3, axis=2)

edge\_detection\_filter = np.repeat(edge\_detection[:, :, np.newaxis], 3, axis=2)

blur\_filter = np.repeat(blur[:, :, np.newaxis], 3, axis=2)

sobel\_y\_filter = np.repeat(sobel\_y[:, :, np.newaxis], 3, axis=2)

sobel\_x\_filter = np.repeat(sobel\_x[:, :, np.newaxis], 3, axis=2)

blacken\_filter = np.zeros((3, 3, 3))

filters1 = [sharpen\_filter, edge\_detection\_filter, sobel\_x\_filter, sobel\_y\_filter, red\_filter]

filters2 = [degree45\_filter, degree\_minus\_45\_filter, up\_down\_filter, blur\_filter, blue\_filter]

labels1 = ["sharpen filter", "edge detection filter", "sobel\_x filter", "sobel\_y filter", "red filter"]

labels2 = ["degree45\_filter", "degree\_minus\_45\_filter", "up\_down\_blur", "box blur", "blue filter"]

**def** forward\_10\_filters\_and\_show(img, s, p, n, actv):

**global** filters1

**global** filters2

**global** labels1

**global** labels2

image\_set1 = []

image\_set2 = []

**for** filt **in** FILTERS1:

conv\_layer = Convolution\_layer(data=img, ker=filt, s=s, p=p, n=n, actv=actv)

image\_set1.append(conv\_layer.forward()[:, :, 0])

**for** filt **in** FILTERS2:

conv\_layer = Convolution\_layer(data=img, ker=filt, s=s, p=p, n=n, actv=actv)

image\_set2.append(conv\_layer.forward()[:, :, 0])

Image.show\_all(image\_set1, LABELS1)

Image.show\_all(image\_set2, LABELS2)

**Cascading 2 Convolutional Layers and Demonstrating the results.**

Note: the first layer produces W1xH1x3 Therefore, the second filter should be FxFx3 and then goes through the 2nd layer and finaly outputs W1xH1x1

In [173]:

**def** cascade(img):

conv\_layer1 = Convolution\_layer(data=img, ker=blur, s=1, p=1, n=3, norm=**None**, actv=**None**)

x = conv\_layer1.forward()

rv10 = x[:,:,0]

conv\_layer2 = Convolution\_layer(data=x, ker=sobel\_x\_filter, s=1, p=1, n=1, norm=**None**, actv=**None**)

x = conv\_layer2.forward()

rv2 = x[:,:,0]

Image.show\_all([rv10, rv2], ['first layer','second layer'])

cascade(afeka\_img)



**Convolving gray parrot Image with 3x3x3 5x5x3 7x7x3 identity filters only adjusting the padding so the output will be of the same dimention.**

**All the Images are Equal which proves that the Convolution is working.**

In [174]:

gray\_parrot = cv2.cvtColor(parrot\_img, cv2.COLOR\_RGB2GRAY)

labels = ['Original', 'nothing3D', 'nothing5D', 'nothing7D']

convolved1 = myconv(img=gray\_parrot, ker=nothing3D, stride=1, p=1)[:,:,0]

convolved2 = myconv(img=gray\_parrot, ker=nothing5D, stride=1, p=2)[:,:,0]

convolved3 = myconv(img=gray\_parrot, ker=nothing7D, stride=1, p=3)[:,:,0]

convolved = [convolved1, convolved2, convolved3]

**for** conv **in** convolved:

equals(gray\_parrot, conv)

Image.show\_all([gray\_parrot, convolved1, convolved2, convolved3], labels)

The Images are equal

The Images are equal

The Images are equal



**Using Alberts Eye Pattern as Filter with Normalized Cross Validation and Threshholding Image to detect Matching patterns**

**Note: the padding is 7 because the filter is 9x9 and I wanted the Result to be 'same' dimentions.**

In [175]:

albert\_gray = cv2.cvtColor(albert\_img, cv2.COLOR\_RGB2GRAY)

albert\_eye\_filter = albert\_gray[50:65, 30:45]

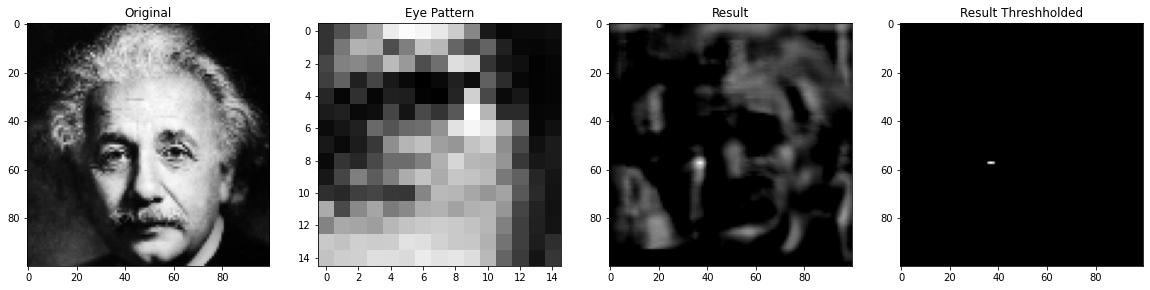
convolved = myconv(img=albert\_gray, ker=albert\_eye\_filter, stride=1, p=7, N=1, Norm=normalized\_cross\_validation, ACTV=relu)

convolved\_gray = convolved[:, :, 0]

result = convolved\_gray

thresh = threshhold(result, 0.8)

Image.show\_all([albert\_gray, albert\_eye\_filter, result, thresh], ['Original', 'Eye Pattern', 'Result', 'Result Threshholded'])



**Using Parrot Eye Pattern as Filter and Threshholding Image to detect Matching patterns**

**Note: padding is 5 because the filter is 5x5 and I wanted the result to be of 'same' dimentions.**

In [176]:

parrot\_eye\_filter = parrot\_img[25:40, 30:45]

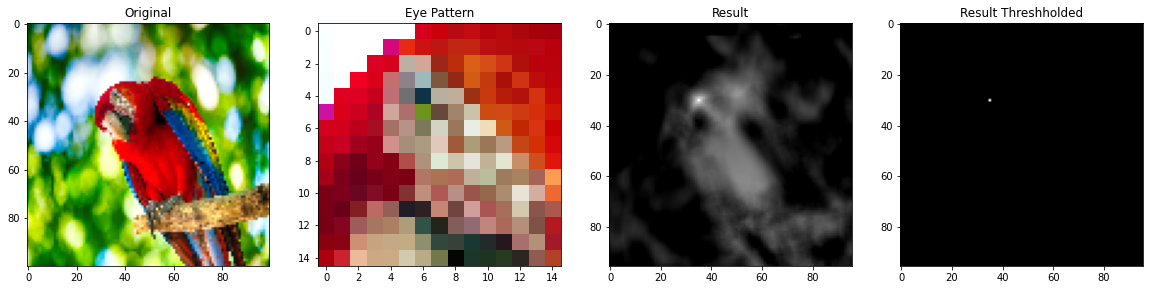
convolved = myconv(img=parrot\_img, ker=parrot\_eye\_filter, stride=1, p=5, N=1, Norm=normalized\_cross\_validation, ACTV=relu)

convolved\_gray = convolved[:, :, 0]

result = convolved\_gray

thresh = threshhold(result, 0.8)

Image.show\_all([parrot\_img, parrot\_eye\_filter, result, thresh], ['Original', 'Eye Pattern', 'Result', 'Result Threshholded'])



**An Example of Bad Hyper-parameters**

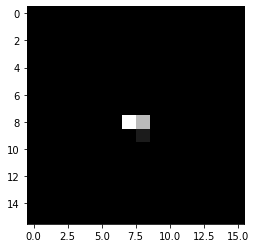
We can't even see that the digit is '4'

In [177]:

convolved = myconv(img=mnist\_img, ker=sharpen\_filter, stride=8, p=50, N=1, Norm=**None**, ACTV=**None**)

convolved\_gray = convolved[:, :, 0]

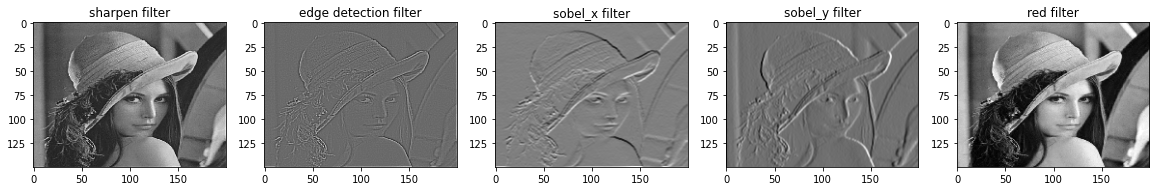
Image.show(convolved\_gray)



**Operating N=10 Filters on the same Image with different hyper-parameters using the Convolutional Layer I made.**

In [178]:

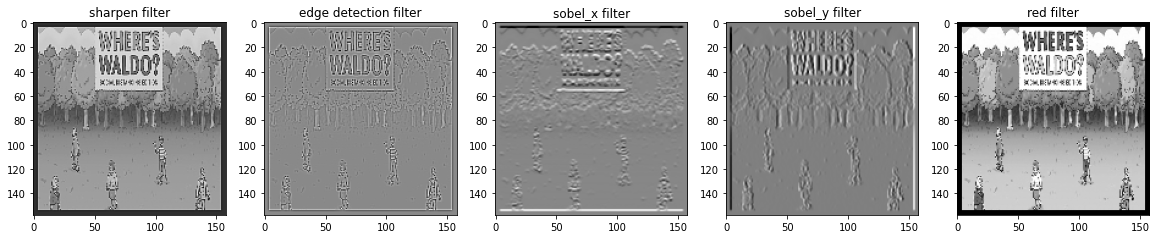
forward\_10\_filters\_and\_show(img=lena\_img, s=1, p=1, n=1, actv=**None**)





In [179]:

forward\_10\_filters\_and\_show(img=waldo\_img, s=1, p=5, n=1, actv=**None**)





In [180]:

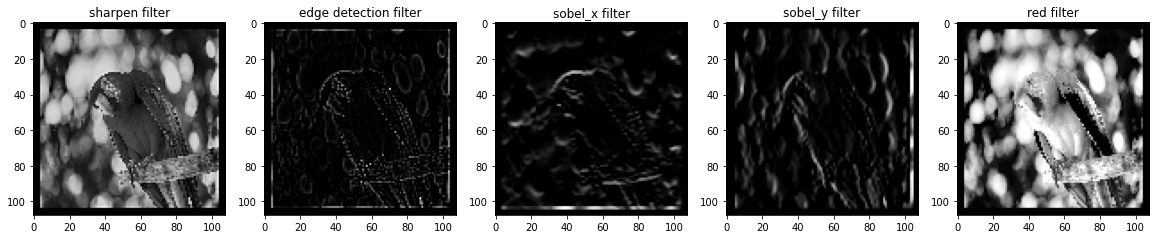
forward\_10\_filters\_and\_show(img=albert\_img, s=2, p=3, n=1, actv=relu)





In [181]:

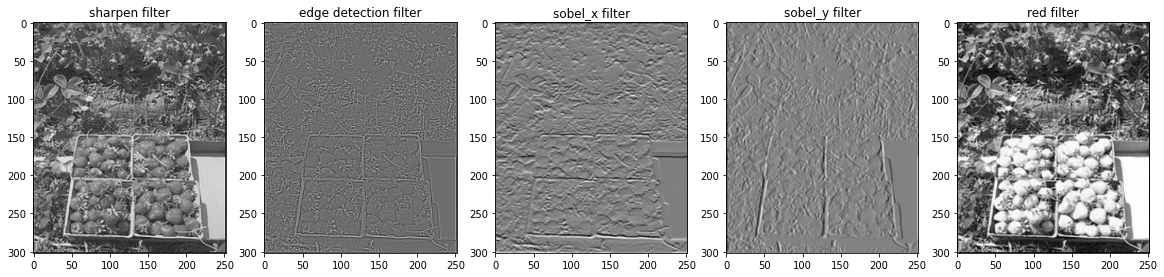
forward\_10\_filters\_and\_show(img=parrot\_img, s=1, p=5, n=1, actv=relu)

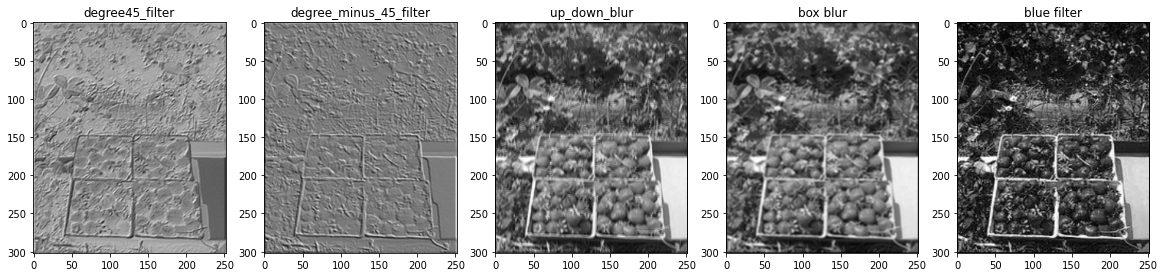




In [182]:

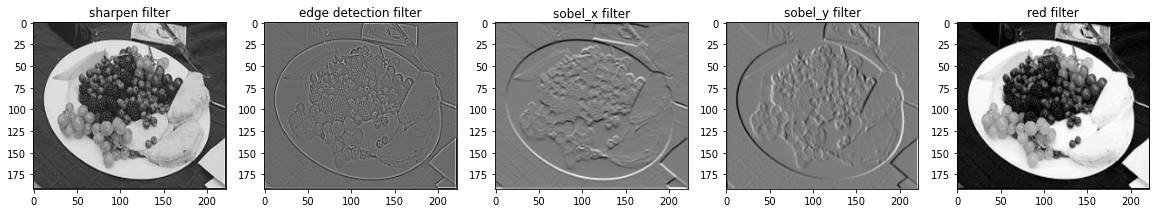
forward\_10\_filters\_and\_show(img=grapes\_img, s=1, p=2, n=1, actv=**None**)





In [183]:

forward\_10\_filters\_and\_show(img=grapes2\_img, s=1, p=2, n=1, actv=**None**)





In [184]:

forward\_10\_filters\_and\_show(img=grapes3\_img, s=1, p=2, n=1, actv=**None**)

