

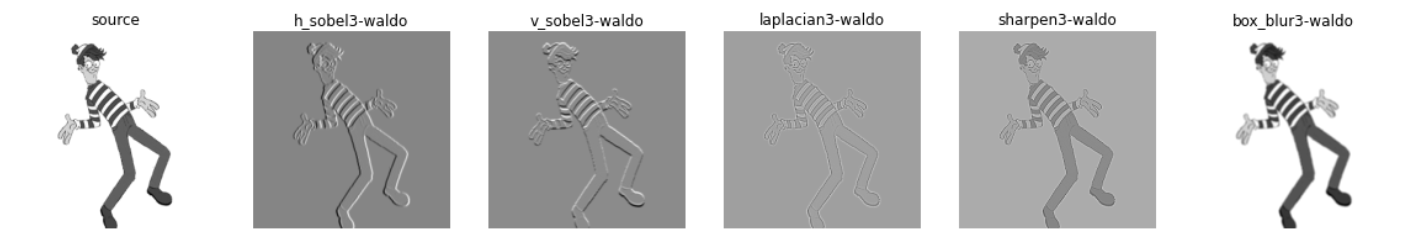
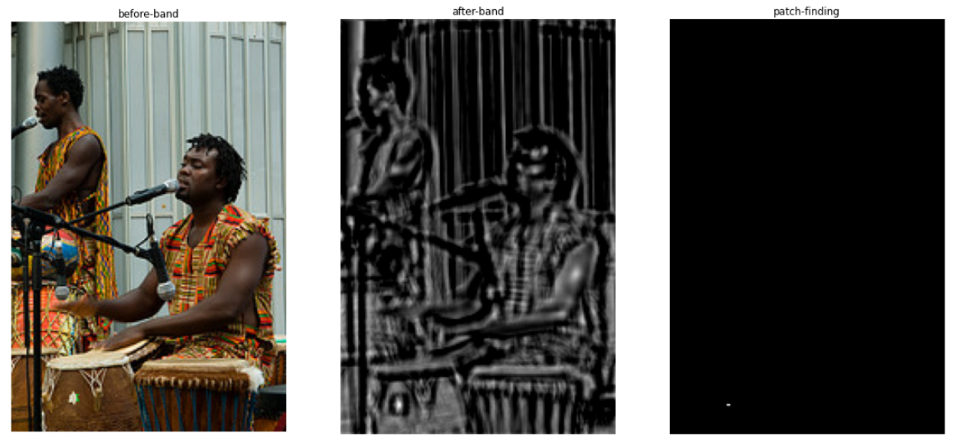
# **מבוא לראייה ממוחשבות 2020 -**

# **Midterm**



**שם: אילון מזרחי**

**ת.ז: \*\*\*\*\*\*\*\*\***

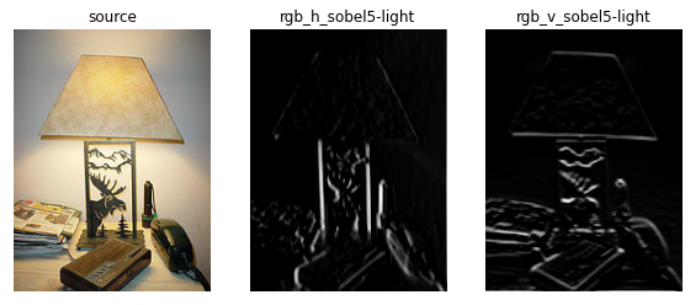
**תאריך הגשה: 15\05\2020**

# **Abstract:**

הגדרת הבעיה: מימוש פונקציה שתומכת בפעולות קונבולוציה וקורלציה, על ידי התחשבות בכל הפרמטרים השונים אשר פונקציות אלה מקבלות: מספר הפעולות שיש לעשות N, באיזו פעולה מדובר CORR, גודל הצעד S, גודל הריפוד P, פונקציית האקטיבציה ACTV והאם יש להפעיל normalized cross-correlation- ייצוג על ידי הארגומנט Norm, תמונת המקור ופילטר.

על הפונקציה לעיל לתמוך במודל שכבות, ולהיות מדויקת ברמת דיוק של הפונקציות השייכות לספריות המוכנות. בנוסף על הפונקציה להיות מסוגלת להפעיל template-matching, כפי שנלמד בהרצאה.

האלגוריתם המוצע: פעולה הנקראת myCorr כמו שהתקבלה בטופס הבחינה שמממשת את הנאמר לעיל. הפעולה משתמשת בכל היפר-הפרמטרים הרלוונטיים לקונבולוציה \ לקורלציה לפי בחירת המשתמש. הפונקציה מסוגלת בנוסף לבצע אקטיבציה לפי פונקציית עזר שהוגדרה עבור אקטיבציית relu, ועוד, מסוגלת לבצע normalized cross-correlation על מנת לבצע template-matching. ב- myCorr מתבצע מעבר על התמונה המרופדת כולה, הכפלת כל ערך של ה- patch בערך המתאים של הפילטר וסכימת כל המכפלות אל התוצאה, כפי שנעשה בפעולת קורלציה \ קונבולוציה. בנוסף הוצע מודל שתי שכבות אשר מבצע חידוד ומבליט את הקווים האנכיים בתמונת הקלט.

תוצאות האלגוריתם המוצע: תוצאת האלגוריתם המוצע הושוותה בכמה צורות שונות אל מול שתי תמונות, ביניהן תוצאת הפונקציה המוכרת של ספריית scipy והשגיאה המקסימלית (המרחק המקסימלי בין התוצאות) שהתקבלה היא 7.27 (באמצעות פילטר חידוד). בנוסף התקבלו תוצאות מדויקות עבור מיקום ה- match בתמונה המקורית בביצוע ה- template matching.



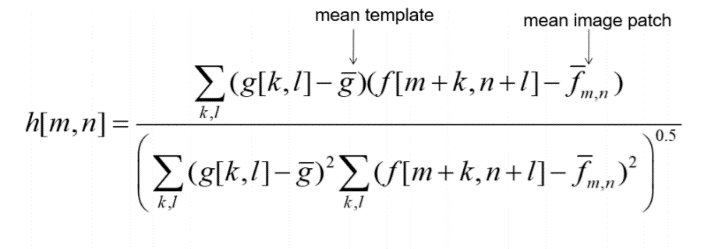
# **Introduction:**

אחד מהנושאים הפופולריים ביותר בעולם המודרני של הראייה הממוחשבת הוא למידה עמוקה באמצעות רשתות נוירונים. בבסיס תחום זה יש לבצע מניפולציות רבות ושונות על תמונות, על מנת שהמודל המשתמש במניפולציות אלה יהיה מסוגל לבצע את הפעולות שלשמן הוא הוגדר ואף יגיע לתוצאות טובות יותר בתחומים שונים: זיהוי \ חיזוי של תבניות בתמונות \ וידאו, חילול תמונות וכו'. כל תמונה מיוצגת על ידי מטריצה של פיקסלים וכל פיקסל מקבל ערך שונה המייצג את חוזקו \ עומקו.

מניפולציות אלה מושגות במידה רבה גם באמצעות 2 פעולות מתמטיות הנקראות קונבולוציה וקורלציה. בבסיס פעולות אלה, מועבר פילטר (בד"כ מיוצג על ידי מטריצת תבנית קטנה) על כל תת מטריצה השייכת לתמונת הקלט באמצעות צעדים שגודלם מוגדר בקלט. כל פיקסל השייך לפילטר שפוסע על תמונת המקור, כופל כל פיקסל שנמצא בתת מטריצה של המקור ותוצאת סכום מכפלות אלו מיוצגת על ידי פיקסל חדש בתמונת התוצאה. לבסוף מתקבלת תמונה חדשה בהתאם לפילטר שהוגדר (דוגמאות לפילטרים: חידוד, טשטוש, מציאת קצוות וכו'). על מנת לשמור \ לשנות את גודל תמונת התוצאה, ראשית יש לרפד את התמונה בפיקסלים נוספים, כיוון שפעולות מתמטיות אלה מקטינות את גודל התוצאה. בעיה זו היא מעניינת מפני שישנם המון סוגים של פרמטרים אליהם צריך להתייחס על מנת שהאלגוריתם יעבוד או כדי לשפרו: גודל הצעד של הפילטר על התמונה, גודל ריפוד התמונה לפני ביצוע בפעולה המתמטית עליה, מספר התמונות שיש לייצר בסופו של דבר, מספר הערוצים בתמונה ובפילטר וגודל הפילטר. כל הפרמטרים שהוצגו משפיעים על גודלה ומראה של תמונת התוצאה במידה רבה ויש להתאים את האלגוריתם לנוסחאות המתמטיות המיוצגות על ידי כל הפרמטרים הרלוונטיים:

פעולת קורלציה, מיקום התחלתי של patch בתוך תמונת המקור, לאחר ריפודה לפי הצורך (כדי לשמור על תוצאה בעלת גודל זהה לתמונת המקור, יש לבצע ריפוד בגודל 1. n, m הם מיקום ה- patch הנוכחי שיש לבצע עליו סכום מכפלות עם הפילטר ו- i, j הם מיקומי הפיקסלים בתוך ה- patch והפילטר:

פעולת קונבולוציה (שקולה להיפוך הפילטר ב- 180 מעלות ולאחר מכן פעולת קורלציה):

בנוסף, אתגר נוסף הוא להחליט באיזו פונקציית אקטיבציה כדאי להשתמש על מנת להגיע למטרה, כיצד לממש אותה, באיזו פונקציית נרמול כדאי להשתמש וכיצד לממש אותה. לדוגמה, עבור template matching כדאי להשתמש ב- normalized cross-correlation ולבצע אקטיבציית relu.

Normalized cross-correlation:

קיימים אלגוריתמים שונים לפתרון הבעיות הנ"ל, ביניהם פעולות הקונבולוציה והקורלציה של ספריית scipy, אך זו אינה תומכת בתמונת קלט עם ערוצי RGB. בנוסף פונקציות מוכנות אלה אינן תומכות בגודל הצעד S, אינן תומכות ב- normalized cross-correlation, והן מסוגלות לייצר תמונת פלט יחידה.

מתוך כל אלה, כדאי לחשוב על פתרון מעניין אשר מאפשר התחשבות בכל הפרמטרים האפשריים ובכל סוג של תמונה.

# **Solution:**

בפרויקט זה הוצע אלגוריתם אשר מממש פעולות של קונבולוציה \ קורלציה על פי בחירת המשתמש, לפי הדרישות שתוארו למעלה שאינן נלקחות בחשבון בפונקציות המוכנות של scipy.

פונקציית האם

def myCorr(Im, Ker, S=1, P=0, N=1, Norm=False, CORR=True, ACTV=None);

מסוגלת להוציא N פלטים של תמונות והיא ראשית קוראת לפונקציית העזר

def pre\_corr(Im, Ker, S, P, N, Norm, CORR, ACTV);

אשר מבצעת pre-processing עבור הבעיה הנתונה, כלומר בדיקות בסיסיות לתקינות הקלט, החלטה כיצד להתייחס אל הפילטר בהתאם לקביעת סוג הפעולה (קורלציה או קונבולוציה) על ידי הארגומנט CORR וריפוד תמונת הקלט אם יש צורך בכך, לפי הארגומנט P. לאחר מכן בתוך פונקציית האם, מוגדרים הגדלים הרלוונטיים של תמונות הפלט המפולטרות, והגדלים הרלוונטיים של הלולאות. כל אלה מתחשבים בתמונה המרופדת, בגודל הצעד ובגודל הפילטר הנתון Ker, מתוך הנוסחה הידועה:

גדלי הלולאות לוקחים בחשבון את התמונה המרופדת, והן רצות באינדקסים לפי הפיקסל השמאלי-עליון ביותר של ה- patch, אף על פי שניתן לממש את האינדקסצייה גם באמצעות ריצה על גודלה של תמונת המקור, לפי מרכז מטריצת patch. בתוך הלולאות, אם הפרמטר של Norm=False, מוכנס ה- patch הרלוונטי לפי האינדקסים אל פעולת עזר אשר מחשבת את סכום המכפלות הרצוי עבורו, עם תמיכה בכל ערוץ של תמונת הקלט. במידה שהפרמטר של Norm=False אז נקראת פונקצייה אשר מבצעת normalized cross-correlation על פי הנוסחה המתמטית שהוצגה בעמוד הקודם: ראשית, מכפלת הפרש הפילטר עם הערך הממוצע שלו בהפרש ה- patch עם הערך הממוצע שלו. לאחר מכן מעבר על כל הפיקסלים וחישוב ריבוע של אותם ההפרשים (ברמת הפיקסל) והוצאת שורש מהתוצאה. הפעולה מחזירה את תוצאת המכפלה הראשונה חלקי תוצאת השורש.

כל תוצאה כזו של תת מטריצה נכנסת לאינדקס הרלוונטי בתמונת התוצאה. לאחר מכן פונקציית האם קוראת לפונקציית האקטיבציה הרלוונטית relu הידועה, אשר מומשה גם היא, על ידי איפוס כל הפיקסלים בתמונת הקלט אשר ערכיהם שליליים. חתימות הפונקציות המוצגות:

def relu(img); def current\_corr2d(current\_mat, kernels);

def cross\_correlation\_norm(patch, f);

במידה שהוכנס N>1, אז התהליך חוזר על עצמו עד אשר מוכנסות לתוך הפלט N תמונות תוצאה.

על מנת לבצע template matching, ראשית יש לקבוע את הארגומנט הרלוונטי כ- Norm=True, ולאחר מכן להעביר את תמונת התוצאה בתוך פעולת threshold שמומשה גם היא, אשר מאפסת כל ערך מתחת לסף וממקסמת כל ערך מעל הסף:

def thresh(img, t);

בנוסף, נוצרה מחלקה של מודל לדוגמה של רשת פשוטה, אשר מממשת שתי שכבות קונבולוציה על פי הפונקציה הנ"ל, כל אלה מומשו גם הם בצורה בסיסית ביותר על מנת לבחון את שילוב האלגורים המוצע בתוך מודל של כמה שכבות. והשמות הרלוונטיים בקוד הם: Sobel\_Model, Convolutional\_Layer.

תרשים זרימת הפתרון המוצע:

**Pre-processing:** padding, conv/corr, checks, etc.

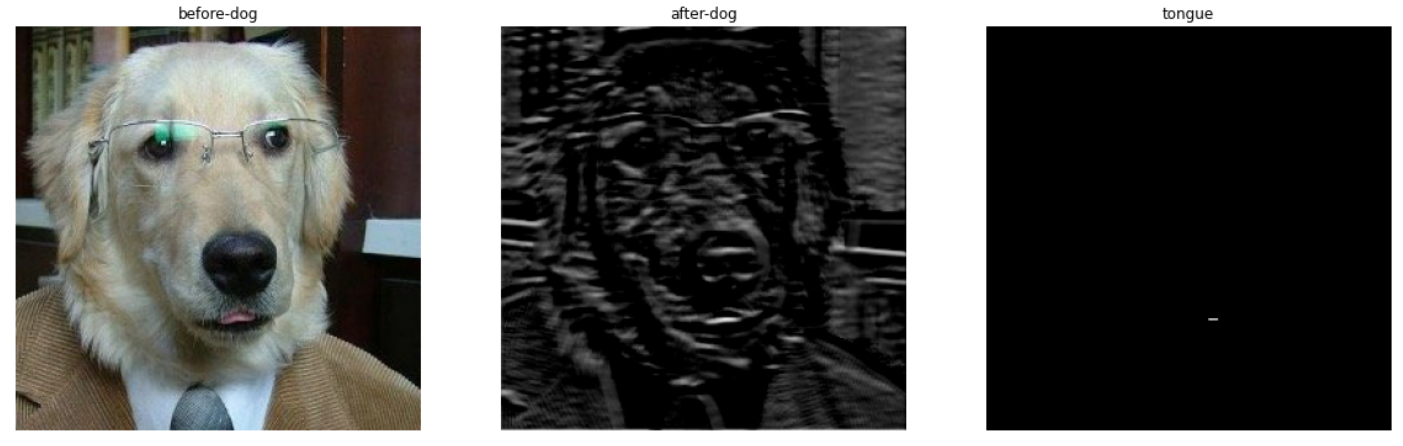
**Sizes' definition:** new\_shape, iterations' numbers, etc.

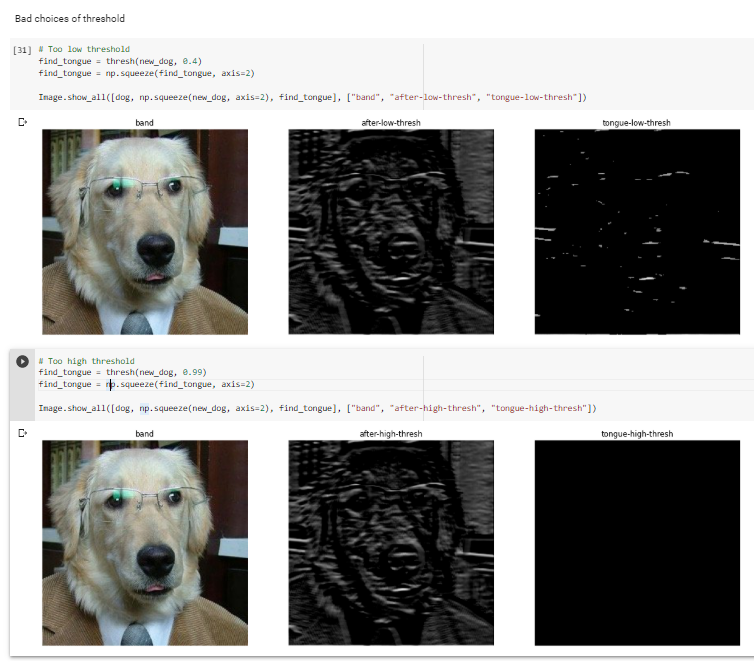
**One-step Correlation/Convolution/ Normalized Cross-Correlation**.

**Add current result to "results" and repeat until N times**

**Activation: relu.**

במציאת template matching, ישנה חשיבות בבחירת ערך הסף. ערך סף גבוה מדיי יוביל לתמונה שחורה ול"פספוס" ערך פיקסל המקסימום בתמונה לאחרת הקורלציה המנורמלת. ערך סף נמוך מדיי יוביל למציאתו של רעש לבן בתמונה לאחר הקורלציה במקום למצוא רק את ה- matching. לפיכך, ראשית יש למצוא את הערך המקסימלי של פיקסל בתמונת הביניים ובחירת ערך הסף בהתאם לערך זה- נמוך ממנו במקצת. בחירות שונות של ערך הסף:

בחירה טובה של ערך הסף להיות 0.8:

בחירות שגויות לערך הסף, כפי שתוארו:

תוצאות המודל שהוגדר בעל שתי השכבות:



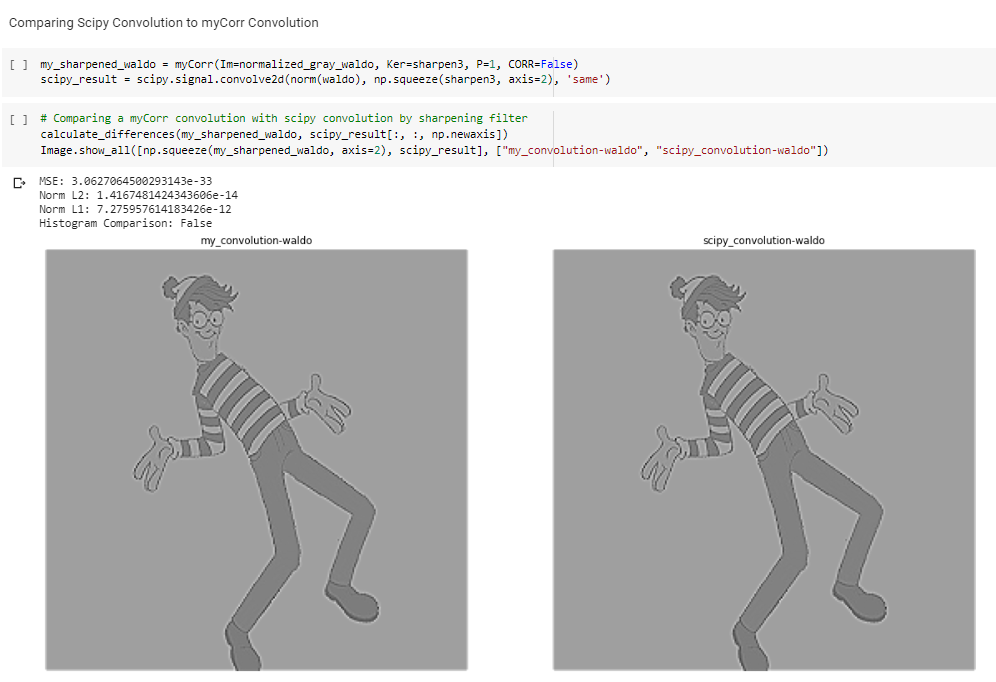
# **Results:**

האלגוריתם המוצע, נבדק על ידי פונקציה שהוגדרה, המבצעת לולאה של 10 איטרציות לכל תמונה. הפונקציה הורצה על 9 תמונות שונות, עם פרמטרים שונים ועם פילטרים שונים. חתימת הפונקציה הנ"ל:

def test\_myCorr(img, kernel\_lst, title\_lst, S=1, P=0, N=1, Norm=False, CORR=True, ACTV=None);

בנוסף האלגוריתם נבדק על ידי השוואת תוצאתו על ידי פילטר שלא מבצע כלום (משאיר את התמונה כפי שהיא) לתמונת המקור לפי מדידת מרחק ב- 3 שיטות: MSE, L1-distance, L2-distance. התוצאה שהתקבלה היא התוצאה הרצויה, כלומר, כל השגיאות התאפסו. עוד בדיקה שבוצעה, היא השוואת תוצאת האלגוריתם על ידי פילטר חידוד, אל תוצאת פונקציה מוכנה מהספרייה scipy, באותן שיטות מדידה. התוצאה שהתקבלה היא לא מדויקת, כלומר הקונבולוציה של scipy שונה במקצת מהקונבולוציה שהוגדרה ב- myCorr. התוצאות להלן:





הרצת בודדת של פונקציית הבדיקות שתוארה לעיל:



# **Discussions:**

כפי שצוין לעיל, תוצאת הקונבולוציה שמומשה בפתרון שונה במקצת מתוצאת הקונבולוציה שמבצעת פונקציה בספרייה scipy, לכן כדאי לחקור את המימוש של scipy ולשים לב אם ישנן דרכים בהן ניתן לשפר את הפתרון, או שמא זהו שוני רלוונטי.

בנוסף, ניתן בהמשך לממש תמיכה בערכים נוספים עבור הארגומנטים של הפונקציה myCorr:

1. מימוש פונקציות אקטיבציה נוספות אשר יתמכו בפונקציית האם שהוצעה.
2. מימוש סוגי padding שונים, בנוסף לריפוד באפסים כפי שבוצע באלגוריתם ה- padding שהוצע.

# **Summary and Conclusions:**

תהליך הקונבולוציה (או קורלציה) הינו תהליך חשוב מאד בעולם הראייה הממוחשבת, לו שימושים רבים. עם זאת הוא מהווה תהליך ארוך אשר דורש בחירת פרמטרים רבה ותכנון מושכל. אי לכך, יש לספק אלגוריתם יעיל ככל הניתן, אשר תומך בכלל הפרמטרים שהתהליך דורש, ומספק גמישות גבוהה בבחירת ערכיהם. כיום, כאשר תחום הלמידה העמוקה מתפתח בקצב אדיר, יש לספק גם תמיכה של האלגוריתם במימוש של מודל שכבות מורכב. קיימים פתרונות יעילים אשר מספקים שכבות קונבולוציה \ תהליך קונבולוציה, אך אלו לא בהכרח תומכים בכלל הפרמטרים שהוצגו, לדוגמה, האלגוריתם של scipy. לפיכך, בפרויקט זה הוצע פתרון אשר מספק את רוב הדרישות לעיל והוא נבדק על מעל 50 תמונות וסיפק תוצאות טובות, למשל, קורלציה על ידי פילטר שלא מבצע כלום ועל ידי בחירת כל הפרמטרים הרלוונטיים הובילה לתוצאה שקולה בדיוק לתמונת המקור. פתרון זה מספק תוצאות טובות, אך יש להמשיך לחקור בנושא על מנת לשפרו, מפני שזהו תחום בו ההתפתחות היא אדירה, והדרך אל האופטימום עלולה להיות אינסופית.

**בהמשך, צירוף המסמך מתוך Google Colab**

# **Imports**

In [0]:

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

**import** **math**

**import** **cv2**

**import** **scipy.signal**

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

**from** **skimage.metrics** **import** mean\_squared\_error

%**matplotlib** inline

# **Image Wrapper Class**

In [0]:

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

**def** \_\_init\_\_(self, path):

self.path = path

self.rgb\_image = **None**

self.bgr\_image = **None**

self.gray\_image = **None**

**def** read\_image(self, return\_image = **False**):

self.rgb\_image = plt.imread(self.path)

**if** return\_image:

**return** self.rgb\_image

**def** bgr(self, return\_image = **False**):

self.bgr\_image = np.flip(plt.imread(self.path), 2)

**if** return\_image:

**return** self.bgr\_image

**def** gray(self, return\_image = **False**):

self.gray\_image = cv2.cvtColor(plt.imread(self.path), cv2.COLOR\_RGB2GRAY)

**if** return\_image:

**return** self.gray\_image

@staticmethod

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

**if** len(image.shape) == 3:

plt.imshow(image)

**else**:

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

plt.axis('off')

plt.savefig("./data/" + title + ".png")

plt.title(title)

@staticmethod

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

**assert** len(image\_list) == len(title\_list), "Incompatible lengths of lists!"

N = len(image\_list)

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

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

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

Image.show(image\_list[i], title\_list[i])

plt.show()

Converts (N, N, 1) filter to (N, N, 3) filter

In [0]:

**def** add\_filter\_rgb\_channels(f):

rgb\_f = np.zeros((f.shape[0], f.shape[1], 3))

f = np.squeeze(f, axis=2)

**for** c **in** range(rgb\_f.shape[2]):

rgb\_f[:, :, c] = f

**return** rgb\_f

# **Filters' Definitions**

In [0]:

*# 3x3x1 filters*

h\_sobel3 = (np.asarray([[1, 0, -1], [2, 0, -2], [1, 0, -1]]) / 9)[:, :, np.newaxis]

v\_sobel3 = (np.asarray([[1, 2, 1], [0, 0, 0], [-1, -2, -1]]) / 9)[:, :, np.newaxis]

laplacian3 = np.asarray([[-1] \* 3, [-1, 8, -1], [-1] \* 3])[:, :, np.newaxis]

sharpen3 = np.asarray([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])[:, :, np.newaxis]

box\_blur3 = (np.asarray([[1] \* 3, [1] \* 3, [1] \* 3]) / 5)[:, :, np.newaxis]

gaus3 = (np.asarray([[1, 2, 1], [2, 4, 2], [1, 2, 1]]) / 5)[:, :, np.newaxis]

no\_effect3 = np.asarray([[0] \* 3, [0, 1, 0], [0] \* 3])[:, :, np.newaxis]

h\_blur3 = np.asarray([[0] \* 3, [1] \* 3, [0] \* 3])[:, :, np.newaxis]

v\_blur3 = np.asarray([[0, 1, 0], [0, 1, 0], [0, 1, 0]])[:, :, np.newaxis]

main\_diagonal\_sobel = (np.asarray([[0, 1, 2], [-1, 0, 1], [-2, -1, 0]]))[:, :, np.newaxis]

sec\_diagonal\_sobel = (np.asarray([[-2, -1, 0], [-1, 0, 1], [0, 1, 2]]))[:, :, np.newaxis]

kernels3 = [h\_sobel3, v\_sobel3, laplacian3, sharpen3, box\_blur3, gaus3,

no\_effect3, main\_diagonal\_sobel, sec\_diagonal\_sobel, h\_blur3, v\_blur3]

titles3 = ["h\_sobel3", "v\_sobel3", "laplacian3", "sharpen3", "box\_blur3", "gaus3",

"no\_effect3", "main\_diagonal\_sobel", "sec\_diagonal\_sobel", "h\_blur3", "v\_blur3"]

*# 5x5x1 filters*

h\_sobel5 = (np.asarray([[2, 1, 0, -1, -2], [2, 1, 0, -1, -2], [4, 2, 0, -2, -4],

[2, 1, 0, -1, -2], [2, 1, 0, -1, -2]]) / 25)[:, :, np.newaxis]

v\_sobel5 = (np.asarray([[2, 2, 4, 2, 2], [1, 1, 2, 1, 1], [0, 0, 0, 0, 0],

[-1, -1, -2, -1, -1], [-2, -2, -4, -2, -2]]) / 25)[:, :, np.newaxis]

laplacian5 = np.asarray([[-1] \* 5, [-1] \* 5, [-1, -1, 8, -1, -1], [-1] \* 5, [-1] \* 5])[:, :, np.newaxis]

sharpen5 = np.asarray([[0, 0, -1, 0, 0], [0, 0, -1, 0, 0], [-1, -1, 5, -1, -1],

[0, 0, -1, 0, 0], [0, 0, -1, 0, 0]])[:, :, np.newaxis]

box\_blur5 = (np.asarray([[1] \* 5, [1] \* 5, [1] \* 5, [1] \* 5, [1] \* 5]) / 9)[:, :, np.newaxis]

gaus5 = (np.asarray([[1, 4, 7, 4, 1], [4, 16, 26, 16, 4], [7, 26, 41, 26, 7],

[4, 16, 26, 16, 4], [1, 4, 7, 4, 1]]) / 273)[:, :, np.newaxis]

no\_effect5 = np.asarray([[0] \* 5, [0] \* 5, [0, 0, 1, 0, 0], [0] \* 5, [0] \* 5])[:, :, np.newaxis]

main\_diagonal\_blur = (np.asarray([[1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 1, 0, 0],

[0, 0, 0, 1, 0], [0, 0, 9, 0, 1]]) / 25)[:, :, np.newaxis]

h\_blur5 = np.asarray([[0] \* 5, [0] \* 5, [1] \* 5, [0] \* 5, [0] \* 5])[:, :, np.newaxis]

v\_blur5 = np.asarray([[0, 0, 1, 0, 0], [0, 0, 1, 0, 0], [0, 0, 1, 0, 0],

[0, 0, 1, 0, 0], [0, 0, 1, 0, 0]])[:, :, np.newaxis]

sec\_diagonal\_blur = (np.asarray([[0, 0, 0, 0, 1], [0, 0, 0, 1, 0], [0, 0, 1, 0, 0],

[0, 1, 0, 0, 0], [1, 0, 0, 0, 0]]) / 25)[:, :, np.newaxis]

kernels5 = [h\_sobel5, v\_sobel5, laplacian5, sharpen5, box\_blur5, gaus5,

no\_effect5, main\_diagonal\_blur, sec\_diagonal\_blur, h\_blur5, v\_blur5]

titles5 = ["h\_sobel5", "v\_sobel5", "laplacian5", "sharpen5", "box\_blur5", "gaus5",

"no\_effect5", "main\_diagonal\_blur", "sec\_diagonal\_blur", "h\_blur5", "v\_blur5"]

*# 3x3x3 filters*

rgb\_h\_sobel3 = add\_filter\_rgb\_channels(h\_sobel3)

rgb\_v\_sobel3 = add\_filter\_rgb\_channels(v\_sobel3)

rgb\_laplacian3 = add\_filter\_rgb\_channels(laplacian3)

rgb\_sharpen3 = add\_filter\_rgb\_channels(sharpen3)

rgb\_box\_blur3 = add\_filter\_rgb\_channels(box\_blur3)

rgb\_gaus3 = add\_filter\_rgb\_channels(gaus3)

rgb\_no\_effect3 = add\_filter\_rgb\_channels(no\_effect3)

rgb\_h\_blur3 = add\_filter\_rgb\_channels(h\_blur3)

rgb\_v\_blur3 = add\_filter\_rgb\_channels(v\_blur3)

rgb\_main\_diagonal\_sobel = add\_filter\_rgb\_channels(main\_diagonal\_sobel)

rgb\_sec\_diagonal\_sobel = add\_filter\_rgb\_channels(sec\_diagonal\_sobel)

rgb\_kernels3 = [rgb\_h\_sobel3, rgb\_v\_sobel3, rgb\_laplacian3, rgb\_sharpen3, rgb\_box\_blur3,

rgb\_gaus3, rgb\_no\_effect3, rgb\_main\_diagonal\_sobel, rgb\_sec\_diagonal\_sobel, rgb\_h\_blur3, rgb\_v\_blur3]

rgb\_titles3 = ["rgb\_h\_sobel3", "rgb\_v\_sobel3", "rgb\_laplacian3", "rgb\_sharpen3", "rgb\_sec\_diagonal3",

"rgb\_gaus3", "rgb\_no\_effect3", "rgb\_main\_diagonal\_sobel", "rgb\_sec\_diagonal\_sobel", "rgb\_h\_blur3", "rgb\_v\_blur3"]

*# 5x5x3 filters*

rgb\_h\_sobel5 = add\_filter\_rgb\_channels(h\_sobel5)

rgb\_v\_sobel5 = add\_filter\_rgb\_channels(v\_sobel5)

rgb\_laplacian5 = add\_filter\_rgb\_channels(sharpen5)

rgb\_sharpen5 = add\_filter\_rgb\_channels(sharpen5)

rgb\_box\_blur5 = add\_filter\_rgb\_channels(box\_blur5)

rgb\_gaus5 = add\_filter\_rgb\_channels(gaus5)

rgb\_no\_effect5 = add\_filter\_rgb\_channels(no\_effect5)

rgb\_h\_blur5 = add\_filter\_rgb\_channels(h\_blur5)

rgb\_v\_blur5 = add\_filter\_rgb\_channels(v\_blur5)

rgb\_main\_diagonal\_blur = add\_filter\_rgb\_channels(main\_diagonal\_blur)

rgb\_sec\_diagonal\_blur = add\_filter\_rgb\_channels(sec\_diagonal\_blur)

rgb\_kernels5 = [rgb\_h\_sobel5, rgb\_v\_sobel5, rgb\_laplacian5, rgb\_sharpen5, rgb\_box\_blur5,

rgb\_gaus5, rgb\_no\_effect5, rgb\_main\_diagonal\_blur, rgb\_sec\_diagonal\_blur, rgb\_h\_blur5, rgb\_v\_blur5]

rgb\_titles5 = ["rgb\_h\_sobel5", "rgb\_v\_sobel5", "rgb\_laplacian5", "rgb\_sharpen5", "rgb\_sec\_diagonal5",

"rgb\_gaus5", "rgb\_no\_effect5", "rgb\_main\_diagonal\_blur", "rgb\_sec\_diagonal\_blur", "rgb\_h\_blur5", "rgb\_v\_blur5"]

# **Show Input**

In [322]:

waldo = Image("./waldo.jpg").gray(**True**)

albert = Image("./albert.jpg").read\_image(**True**)

lena = Image("./lena.jpg").read\_image(**True**)

afeka = Image("./afeka.jpg").read\_image(**True**)

band = Image("./band.jpg").read\_image(**True**)

dog = Image("./dog.jpg").read\_image(**True**)

mnist = Image("./mnist.jpg").read\_image(**True**)[:, :, 0]

*# ImageNet photos*

cat = Image("./cat.jpg").read\_image(**True**)

carriage = Image("./carriage.jpg").read\_image(**True**)

light = Image("./light.jpg").read\_image(**True**)

sport = Image("./sport.jpg").read\_image(**True**)

green = Image("./green.jpg").read\_image(**True**)

Image.show\_all([waldo, albert, lena, afeka, mnist, cat, carriage, light, sport, green, band, dog],

["waldo", "albert", "lena", "afeka", "mnist", "cat", "carriage", "light", "sport", "green", "band", "dog"])



# **Relu Function Definition**

In [0]:

**def** relu(img):

*# Operates 'relu' activation function on a given image*

*# Returns the activated image*

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

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

**for** c **in** range(img.shape[2]):

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

img[i][j][c] = 0

**return** img

# **Max Absolute Normalization Definition**

In [0]:

**def** norm(img):

max\_pixel = np.max(np.abs(img))

**if** max\_pixel != 0:

**return** img / max\_pixel

**return** img

# **Threshold Function**

In [0]:

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

temp\_img = np.copy(img)

**for** i **in** range(temp\_img.shape[0]):

**for** j **in** range(temp\_img.shape[1]):

**for** c **in** range(temp\_img.shape[2]):

**if** temp\_img[i][j][c] < t:

temp\_img[i][j][c] = 0

**return** temp\_img

# **Correlation**

One-step Correlation

In [0]:

**def** current\_corr2d(current\_mat, kernels):

**assert** kernels.shape[0] == kernels.shape[1], "Incompatible size of kernel!"

**assert** current\_mat.shape[0] == kernels.shape[0] **and** current\_mat.shape[1] == kernels.shape[1], "Incompatible sizes of matrices!"

channels\_sums = 0

**for** i **in** range(current\_mat.shape[0]):

**for** j **in** range(current\_mat.shape[1]):

**for** c **in** range(current\_mat.shape[2]):

channels\_sums += current\_mat[i][j][c] \* kernels[i][j][c]

**return** channels\_sums

Normalized Cross-Correlation Definition

In [0]:

**def** cross\_correlation\_norm(patch, f):

mean\_patch = np.mean(patch)

mean\_filter = np.mean(f)

sub1 = patch - mean\_patch

sub2 = f - mean\_filter

sum1, sum2 = 0, 0

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

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

**for** c **in** range(f.shape[2]):

sum1 += np.power(patch[i][j][c] - mean\_patch, 2)

sum2 += np.power(f[i][j][c] - mean\_filter, 2)

sqrt\_ = math.sqrt(sum1 \* sum2)

**if** sqrt\_ != 0:

**return** sub1 / sqrt\_, sub2

**else**:

**return** patch, f

Checks And Preprocessing

In [0]:

**def** pre\_corr(Im, Ker, S, P, N, Norm, CORR, ACTV):

*# Validates the input arguments and commits preprocess of the correlation / convolution operations*

*# Returns the padded image and the kernels for the given operation*

*# Input checks*

**assert** len(Im.shape) == 3, str("Expected Im to has 3 dimensions, got " + str(len(Im.shape)))

**assert** len(Ker.shape) == 3, str("Expected Ker to has 3 dimensions, got " + str(len(Ker.shape)))

**assert** S > 0, "Invalid value for stride S!"

**assert** P >= 0, "Invalid value for padding P!"

**assert** N > 0, "Invalid value for kenels' number N!"

**assert** ACTV == **None** **or** ACTV == relu, "Invalid value for activation function ACTV!"

new\_Ker = list()

**if** **not** CORR: *# Convolution*

**for** k **in** Ker: *# Rotates kernels by 180 degrees*

new\_Ker.append(k[::-1])

**else**:

new\_Ker = Ker

*# Padd with zeroes the image if P > 0*

Im = np.pad(Im, pad\_width=((P, P), (P, P), (0, 0)), mode='constant', constant\_values=0)

**return** np.asarray(new\_Ker), Im

myCorr

In [0]:

**def** myCorr(Im, Ker, S=1, P=0, N=1, Norm=**False**, CORR=**True**, ACTV=**None**):

Ker, padded\_img = pre\_corr(Im, Ker, S, P, N, Norm, CORR, ACTV)

W, H, C = Im.shape

K = Ker[:][:][0].shape[0]

new\_shape = (((W - K + 2\*P) // S + 1), ((H - K + 2\*P) // S + 1), 1)

new\_img = np.zeros(new\_shape)

results = np.zeros((new\_shape[0], new\_shape[1], N))

*# Loop limits*

width = W + 2\*P - K + 1

height = H + 2\*P - K + 1

**for** k\_num **in** range(N):

**for** i **in** range(0, width, S):

**for** j **in** range(0, height, S):

current\_mat = padded\_img[i : i + K, j : j + K, :]

**if** Norm: *# Then do normalized cross correlation*

current\_mat, Ker = cross\_correlation\_norm(current\_mat, Ker)

current\_sum = current\_corr2d(current\_mat, Ker)

new\_img[i // S][j // S] = current\_sum

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

new\_img = ACTV(new\_img)

results[:, :, k\_num] = np.squeeze(new\_img, axis=2)

**return** results

Test function for myCorr on a given image

In [0]:

**def** test\_myCorr(img, kernel\_lst, title\_lst, S=1, P=0, N=1, Norm=**False**, CORR=**True**, ACTV=**None**):

output = list()

**if** img.shape[2] == 1:

output.append(np.squeeze(img, axis=2))

**elif** img.shape[2] == 3:

output.append(img)

title\_lst = np.insert(title\_lst, 0, "source")

**for** k **in** kernel\_lst:

output.append(np.squeeze(myCorr(norm(img), k, S, P, N, Norm, CORR, ACTV), axis=2))

Image.show\_all(output, title\_lst)

# **Finding items inside images**

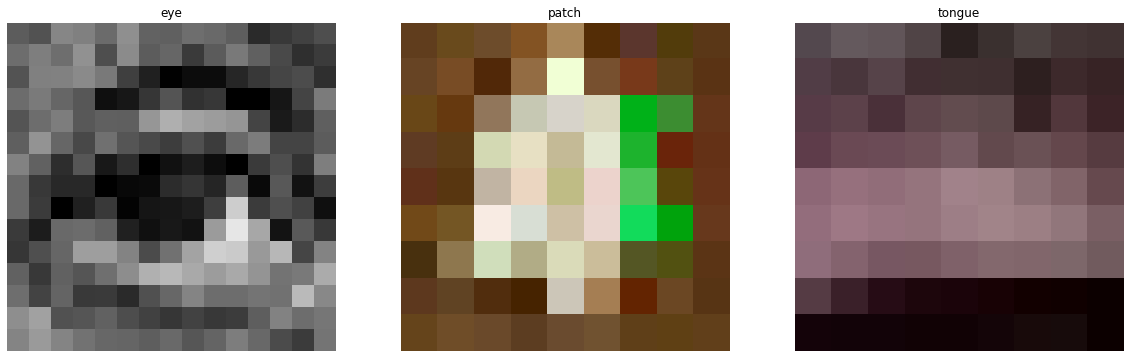
In [331]:

eye = albert[38:53, 37:52]

patch = band[216:225, 32:41, :]

tongue = dog[180:189, 137:146, :]

Image.show\_all([eye, patch, tongue], ["eye", "patch", "tongue"])



In [0]:

new\_albert = myCorr(Im=norm(albert), Ker=norm(eye), N=1, S=1, P=1, ACTV=relu, Norm=**True**, CORR=**False**)

In [0]:

new\_band = myCorr(Im=norm(band), Ker=norm(patch), N=1, S=1, P=1, ACTV=relu, Norm=**True**, CORR=**False**)

In [0]:

new\_dog = myCorr(Im=norm(dog), Ker=norm(tongue), N=1, S=1, P=1, ACTV=relu, Norm=**True**, CORR=**False**)

Good choices of threshold

In [0]:

find\_eye = thresh(new\_albert, 0.464)

find\_eye = np.squeeze(find\_eye, axis=2)

find\_patch = thresh(new\_band, 0.68)

find\_patch = np.squeeze(find\_patch, axis=2)

find\_tongue = thresh(new\_dog, 0.8)

find\_tongue = np.squeeze(find\_tongue, axis=2)

In [336]:

Image.show\_all([albert, np.squeeze(new\_albert, axis=2), find\_eye], ["before-albert", "after-albert", "eye-finding"])



In [337]:

Image.show\_all([band, np.squeeze(new\_band, axis=2), find\_patch], ["before-band", "after-band", "patch-finding"])



In [338]:

Image.show\_all([dog, np.squeeze(new\_dog, axis=2), find\_tongue], ["before-dog", "after-dog", "tongue"])



Bad choices of threshold

In [339]:

*# Too low threshold*

find\_tongue = thresh(new\_dog, 0.4)

find\_tongue = np.squeeze(find\_tongue, axis=2)

Image.show\_all([dog, np.squeeze(new\_dog, axis=2), find\_tongue], ["band", "after-low-thresh", "tongue-low-thresh"])



In [340]:

*# Too high threshold*

find\_tongue = thresh(new\_dog, 0.99)

find\_tongue = np.squeeze(find\_tongue, axis=2)

Image.show\_all([dog, np.squeeze(new\_dog, axis=2), find\_tongue], ["band", "after-high-thresh", "tongue-high-thresh"])



# **Calculates Distances Of myCorr Function**

In [0]:

**def** L1(img1, img2):

*# Calculates L1 norm between two images*

**assert** len(img1.shape) == 3, str("Expected img1 to has 3 dimensions, got " + str(len(img1.shape)))

**assert** len(img2.shape) == 3, str("Expected img2 to has 3 dimensions, got " + str(len(img2.shape)))

**assert** img1.shape[0] == img2.shape[0] **and** img1.shape[1] == img2.shape[1], "Incompatible sizes of images!"

**return** abs(np.sum(img1) - np.sum(img2))

**def** L2(img1, img2):

*# Calculates L2 norm between two images*

**assert** len(img1.shape) == 3, str("Expected img1 to has 3 dimensions, got " + str(len(img1.shape)))

**assert** len(img2.shape) == 3, str("Expected img2 to has 3 dimensions, got " + str(len(img2.shape)))

**assert** img1.shape[0] == img2.shape[0] **and** img1.shape[1] == img2.shape[1], "Incompatible sizes of images!"

sum = 0

**for** i **in** range(img1.shape[0]):

**for** j **in** range(img1.shape[1]):

**for** c **in** range(img1.shape[2]):

sum += math.pow((img1[i][j][c] - img2[i][j][c]), 2)

**return** math.sqrt(sum)

In [0]:

**def** calculate\_differences(img1, img2):

mse = mean\_squared\_error(img1, img2)

norm2 = L2(img1, img2)

norm1 = L1(img1, img2)

flag = np.array\_equal(img1, img2)

print('MSE: ' + str(mse) + '**\n**Norm L2: ' + str(norm2) + '**\n**Norm L1: ' + str(norm1) + '**\n**Histogram Comparison: ' + str(flag))

In [0]:

normalized\_gray\_waldo = norm(waldo)[:, :, np.newaxis]

Correlating with a no-effect filter to compare with source image

In [0]:

new\_gray\_waldo = myCorr(Im=normalized\_gray\_waldo, Ker=no\_effect3, P=1)

In [0]:

*# Comparing a gray image before correlation and the same image after a correlation with no-effect filter*

calculate\_differences(normalized\_gray\_waldo, new\_gray\_waldo)

Image.show\_all([np.squeeze(new\_gray\_waldo, axis=2), waldo], ["my\_correlation-waldo", "source-Waldo"])

MSE: 0.0

Norm L2: 0.0

Norm L1: 0.0

Histogram Comparison: True



Comparing Scipy Convolution to myCorr Convolution

In [0]:

my\_sharpened\_waldo = myCorr(Im=normalized\_gray\_waldo, Ker=sharpen3, P=1, CORR=**False**)

scipy\_result = scipy.signal.convolve2d(norm(waldo), np.squeeze(sharpen3, axis=2), 'same')

In [0]:

*# Comparing a myCorr convolution with scipy convolution by sharpening filter*

calculate\_differences(my\_sharpened\_waldo, scipy\_result[:, :, np.newaxis])

Image.show\_all([np.squeeze(my\_sharpened\_waldo, axis=2), scipy\_result], ["my\_convolution-waldo", "scipy\_convolution-waldo"])

MSE: 3.0627064500293143e-33

Norm L2: 1.4167481424343606e-14

Norm L1: 7.275957614183426e-12

Histogram Comparison: False



# **Cascade Two Layers**

In [0]:

**class** **Convolutional\_Layer**():

**def** \_\_init\_\_(self, data, f, act=**None**):

self.data = data

self.f = f

self.act = act

**def** forward(self):

**return** myCorr(self.data, self.f, P=1, N=1, ACTV=self.act)

In [0]:

**class** **Sobel\_Model**():

**def** \_\_init\_\_(self, data, f1, f2):

self.data = data

self.f1 = f1

self.f2 = f2

self.conv1 = **None**

self.conv2 = **None**

self.output = list()

self.output.append(data)

**def** forward(self):

self.conv2 = Convolutional\_Layer(self.data, self.f2)

o1 = self.conv2.forward()

self.conv1 = Convolutional\_Layer(o1, self.f1)

o2 = self.conv1.forward()

self.output.append(o1)

self.output.append(o2)

**return** o2

In [0]:

f1 = h\_sobel5

f2 = sharpen3

data = norm(Image('./afeka.jpg').gray(**True**))[:, :, np.newaxis]

In [0]:

model = Sobel\_Model(data, f1, f2)

output = model.forward()

output = [np.squeeze(o, axis=2) **for** o **in** model.output]

In [0]:

Image.show\_all(output, ["before-model-afeka", "mid-sharpened-afeka", "after-horizontal-sobel-afeka"])



In [0]:

*# Tests parameter N*

waldos = myCorr(normalized\_gray\_waldo, main\_diagonal\_sobel, N=3, ACTV=relu)

waldos = [waldos[:, :, 0], waldos[:, :, 1], waldos[:, :, 2]]

Image.show\_all(waldos, ["waldo-1", "waldo-2", "waldo-3"])

In [0]:

afeka\_titles = [t + "-afeka" **for** t **in** rgb\_titles3]

In [0]:

test\_myCorr(img=afeka, kernel\_lst=rgb\_kernels3[: len(rgb\_kernels3) // 2], title\_lst=afeka\_titles[: len(afeka\_titles) // 2], P=1)



In [0]:

test\_myCorr(img=afeka, kernel\_lst=rgb\_kernels5[len(rgb\_kernels5) // 2 :], title\_lst=afeka\_titles[len(afeka\_titles) // 2 :], P=10, S=4, CORR=**False**, ACTV=relu)

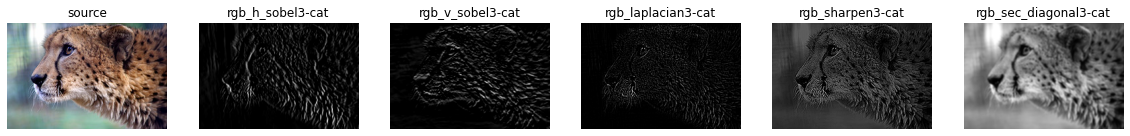


In [0]:

cat\_titles = [t + "-cat" **for** t **in** rgb\_titles3]

In [0]:

test\_myCorr(img=cat, kernel\_lst=rgb\_kernels3[: len(rgb\_kernels3) // 2], title\_lst=cat\_titles[: len(cat\_titles) // 2], P=0, S=1, ACTV=relu)



In [0]:

test\_myCorr(img=cat, kernel\_lst=rgb\_kernels5[len(rgb\_kernels5) // 2 :], title\_lst=cat\_titles[len(cat\_titles) // 2 :], P=1, S=1, CORR=**False**)

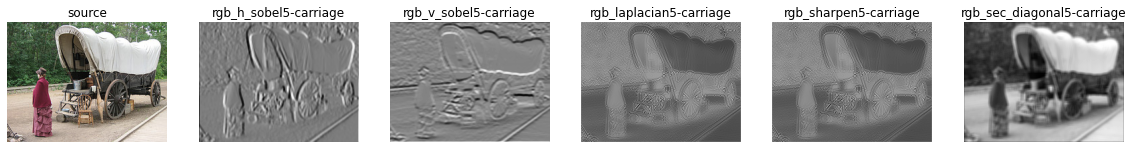


In [0]:

carriage\_titles = [t + "-carriage" **for** t **in** rgb\_titles5]

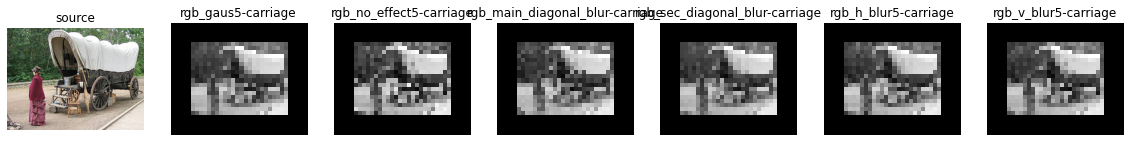
In [0]:

test\_myCorr(img=carriage, kernel\_lst=rgb\_kernels5[: len(rgb\_kernels5) // 2], title\_lst=carriage\_titles[: len(carriage\_titles) // 2], P=1, S=1, CORR=**True**)



In [0]:

test\_myCorr(img=carriage, kernel\_lst=rgb\_kernels5[len(rgb\_kernels5) // 2 :], title\_lst=carriage\_titles[len(carriage\_titles) // 2 :], P=50, S=10, CORR=**False**)

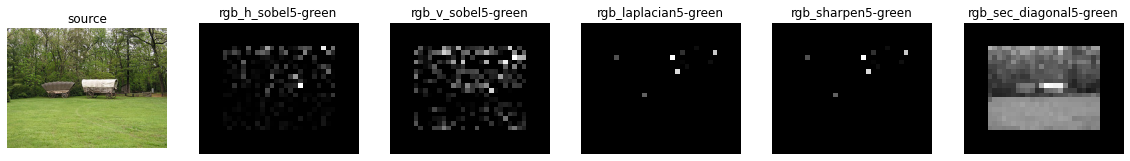


In [0]:

green\_titles = [t + "-green" **for** t **in** rgb\_titles5]

In [0]:

test\_myCorr(img=green, kernel\_lst=rgb\_kernels5[: len(rgb\_kernels5) // 2], title\_lst=green\_titles[: len(green\_titles) // 2], P=50, S=10, ACTV=relu)



In [0]:

test\_myCorr(img=green, kernel\_lst=rgb\_kernels3[len(rgb\_kernels3) // 2 :], title\_lst=green\_titles[len(green\_titles) // 2 :], P=20, S=5, CORR=**True**)



In [0]:

lena\_titles3 = [t + "-lena" **for** t **in** rgb\_titles3]

lena\_titles5 = [t + "-lena" **for** t **in** rgb\_titles5]

In [0]:

test\_myCorr(img=lena, kernel\_lst=rgb\_kernels5[: len(rgb\_kernels5) // 2], title\_lst=lena\_titles5[: len(lena\_titles5) // 2], P=1, S=1, ACTV=relu)



In [0]:

test\_myCorr(img=lena, kernel\_lst=rgb\_kernels3[len(rgb\_kernels3) // 2 :], title\_lst=lena\_titles3[len(lena\_titles3) // 2 :], P=1, S=2, ACTV=relu)



In [0]:

light\_titles3 = [t + "-light" **for** t **in** rgb\_titles3]

light\_titles5 = [t + "-light" **for** t **in** rgb\_titles5]

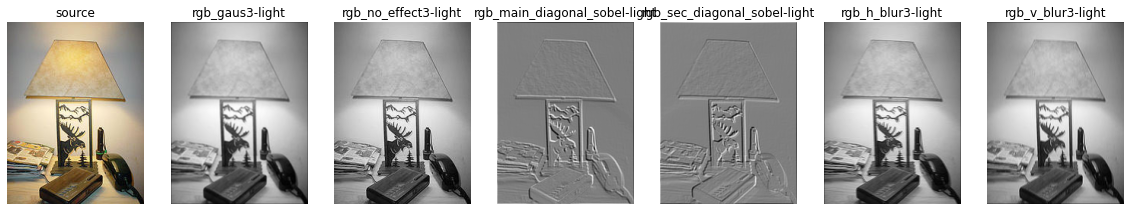
In [0]:

test\_myCorr(img=light, kernel\_lst=rgb\_kernels5[: len(rgb\_kernels5) // 2], title\_lst=light\_titles5[: len(light\_titles5) // 2], P=1, S=1, ACTV=relu)



In [0]:

test\_myCorr(img=light, kernel\_lst=rgb\_kernels3[len(rgb\_kernels3) // 2 :], title\_lst=light\_titles3[len(light\_titles3) // 2 :], P=1, S=1)



In [0]:

mnist\_titles3 = [t + "-mnist" **for** t **in** titles3]

mnist\_titles5 = [t + "-mnist" **for** t **in** titles5]

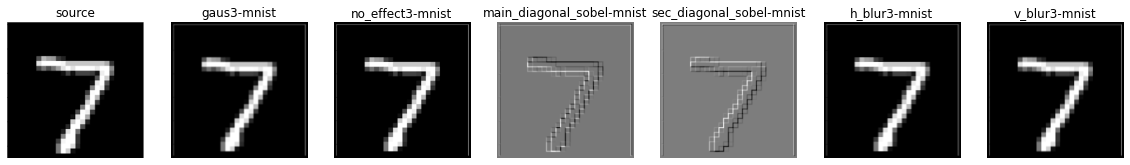
In [0]:

test\_myCorr(img=mnist[:, :, np.newaxis], kernel\_lst=kernels5[: len(kernels5) // 2], title\_lst=mnist\_titles5[: len(mnist\_titles5) // 2], P=1, S=1)



In [0]:

test\_myCorr(img=mnist[:, :, np.newaxis], kernel\_lst=kernels3[len(kernels3) // 2 :], title\_lst=mnist\_titles3[len(mnist\_titles3) // 2 :], P=5)



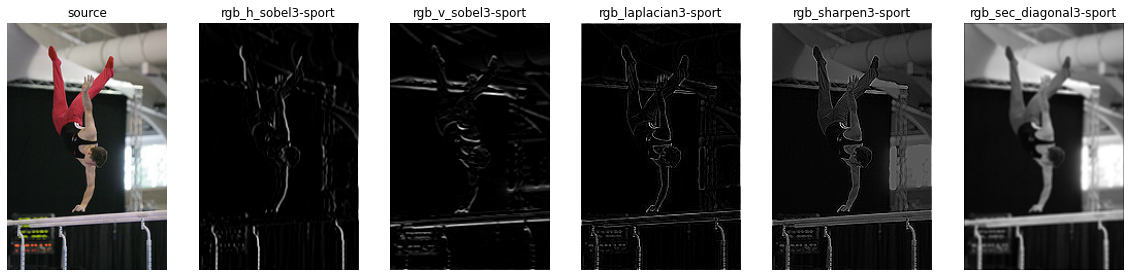
In [0]:

sport\_titles3 = [t + "-sport" **for** t **in** rgb\_titles3]

sport\_titles5 = [t + "-sport" **for** t **in** rgb\_titles5]

In [0]:

test\_myCorr(img=sport, kernel\_lst=rgb\_kernels3[: len(rgb\_kernels3) // 2], title\_lst=sport\_titles3[: len(sport\_titles3) // 2], P=1, S=1, ACTV=relu)



In [0]:

test\_myCorr(img=sport, kernel\_lst=rgb\_kernels5[len(rgb\_kernels5) // 2 :], title\_lst=sport\_titles5[len(sport\_titles5) // 2 :], P=1, S=1, CORR=**False**)



In [0]:

waldo\_titles3 = [t + "-waldo" **for** t **in** titles3]

In [0]:

test\_myCorr(img=waldo[:, :, np.newaxis], kernel\_lst=kernels3[: len(kernels3) // 2], title\_lst=waldo\_titles3[: len(waldo\_titles3) // 2], P=0, S=1, CORR=**False**)



In [0]:

test\_myCorr(img=waldo[:, :, np.newaxis], kernel\_lst=kernels3[len(kernels3) // 2 :], title\_lst=waldo\_titles3[len(waldo\_titles3) // 2 :], P=1, CORR=**True**)



In [341]:

!zip -r /content/data.zip /content/data