

האוניברסיטה העברית בירושלים

בית הספר להנדסה ומדעי המחשב ע"ש רחל וסלים בנין

סדנת תכנות בשפת C++

תרגיל בית 5

תאריך הגשה: 07/01/2021 בשעה 23:55

נושאי התרגיל: היכרות עם C++, מחלקות, const, operators overloading

1 רקע

בתרגיל בית זה נקבל הצצה לאחד התחומים ה"לוהטים" באקדמיה ובתעשייה כיום – עיבוד תמונות. עיבוד תמונה נעשה לשתי מטרות עיקריות: שיפור תמונות (הפחתת רעש, חידוד, אפקטים) והסקת מסקנות (מדידות, זיהוי צורות, סיווג אובייקטים) ויש לו שימושים רבים בתעשיות שונות ובעולם האקדמי.

1.2 ייצוג תמונות כמטריצות

בתמונה יכולים להיות מספר רב של גווני צבע. בתרגיל זה נתמקד בגווני שחור לבן, הנקראים גם גווני אפור. ישנם 256 גוונים כאלו ומקובל לתאר אותם בעזרת מספרים טבעיים בין 0 ל-255 כאשר 255 מייצג בהירות מוחלטת (לבן) ו-0 מייצג כהות מוחלטת (שחור). לכן, ניתן להסתכל על תמונות כמטריצות המורכבות מתאים, וניתן לכנות כל תא כ"פיקסל". לכל פיקסל יהיה גוון צבע שיתאר אותו. כלומר ניתן לתרגם כל תמונה בגווני אפור (שחור-לבן) למטריצה של מספרים עם ערכים טבעיים בין 0 ל-255. *** שימו לב – לא נתעסק בתרגיל זה בתמונות צבעוניות כלל.

2 מימוש

במסגרת התרגיל תדרשו לממש שני חלקים. בחלק הראשון, תממשו מחלקת מטריצה (כל תמונה תיוצג כמטריצה של מספרים בין 0 ל-255). בחלק השני, תכתבו קובץ שיכלול פונקציות אותם נרצה להפעיל על המטריצות. חישובו היטב מהם ערכי ההחזרה לכל מתודה/אופרטור ואילו מתודות/אופרטורים משנים את האובייקט הנוכחי. אין להרחיב את ה-API המפורט, כלומר, אין להוסיף מתודות public למחלקה (ניתן להוסיף מתודות פרטיות כמה שתמצאו).

2.1 חלק ראשון - המחלקה Matrix (Matrix.h, Matrix.cc)

קבצים: בחלק זה עליכם לממש בשני קבצים: Matrix.h, Matrix.cc (עליכם להגיש את שניהם). באמצעות מחלקה זו נבנה את המטריצות הדרושות לשם הפעלת האופרטורים על התמונות. טיפוס הנתונים שתכיל המטריצה יהיה מסוג float. הסיבה שטיפוס הנתונים שנשמר הוא float הוא כדי להתמודד בקלות עם פעולות חילוק וכפל במטריצות וסקלרים שונים. שימו לב – קובץ ה-header נממש את "שלד" המחלקה (כלומר – את ה-API) ובקובץ הקוד (Matrix.cc) יהיו המימושים של המתודות/פונקציות. מבחינת תיעוד הפונקציות והמחלקה, עליכם לכתוב תיעוד מפורט ב-header ולשים בקובץ הקוד תיעוד מינימלי (ניתן ואף רצוי לכתוב תיעוד מפורט ב-header ולהעתיק אותו לקובץ המימוש).

2.1.1 ה-API הדרוש (לא ניתן להרחיב את ה-API בשום צורה!)

Method type	Description	Comments	Error Handling
Constructor	Matrix(int rows, int cols)	Constructs matrix rows * cols (need to make sure rows, cols are non negative). Initiates all elements to 0.	
Default constructor	Matrix()	Constructs 1*1 matrix, where the single element is initiated to 0.	
Copy constructor	Matrix(Matrix &m)	Constructs matrix from another matrix.	
Destructor	~Matrix()	Destroys the matrix.	
Getter	getRows()	Returns the amount of rows (int).	
Getter	getCols()	Returns the amount of columns (int).	
	vectorize()	<p>Transforms a matrix into a column vector. Supports function calling. I.E:</p> <pre>Matrix m(5,4); m.vectorize(); int r = m.getRows(); // suppose to return 20 int c = m.getCols(); // suppose to return 1</pre> <p>[1, 2, 3] [4, 5, 6] ⇒ transpose ([1, 2, 3, 4, 5, 6, 7, 8, 9]) [7, 8, 9]</p> <p>*should support m.vectorize().getRows();</p>	
	print()	Prints matrix elements, no return value (void). Prints space after each element (not including the last element in the row). Prints new line after each row (not including the last row).	

=	assignment	Matrix a, b; ... a = b;	
*	Matrix multiplication	Matrix a, b; ... Matrix c = a * b ** do not forget algebra rules for matrix multiplication.	Check dimensions valid for operation.
*	Scalar mult. On the right	Matrix m; float c; ... Matrix m2 = m * c;	
*	Scalar mult. In the left	Matrix m; float c; ... Matrix m2 = c * m;	
*=	Matrix multiplication	Matrix a, b; ... a *= b; ** it is equivalent to a = a * b	Check dimensions valid for operation.
*=	Scalar mult. accumulation	Matrix a; float c; ... a *= c;	
/	Scalar division on the right	Matrix a; float c; ... Matrix b = a / c; ** check to scalar is not 0.	Check division by zero.
/=	Scalar division	Matrix a; float c; a /= c; ** check to scalar is not 0.	Check division by zero.

+	Matrix addition	Matrix a, b; Matrix c = a + b;	Check dimensions valid for operation.
+=	Matrix addition accumulation	Matrix a, b; a += b;	Check dimensions valid for operation.
+=	Matrix scalar addition.	Matrix a; float c; ... a += c; *adding c to each cell of the matrix.	
()	Parenthesis indexing	int i, j; Matrix m; ... float val = m(i, j); m(i,j) = 5.6; ** check the indexes are in the right range.	Check indexes are in valid ranges.
[]	Brackets indexing	int i; Matrix m; ... float val = m[i]; m[k] = 6.7; ** read section 2.1.2	Check index is in valid range.
==	Equality	Matrix a, b; ... bool t = (a == b);	
!=	Not equal	Matrix a, b; ... bool t = (a != b);	

>>	Input stream	Fills matrix elements. Reads from given input stream. I.E: istream is; Matrix m(3, 5); is >> m; I.E: The input is: 1 2 3 4 5 6 Is >> m; // m is [1 2 3] // [4 5 6]	Check input stream validity. * <u>No need</u> to check if the content of the stream is valid (size, spaces..).
<<	Output stream	Matrix m; ... std::cout << m << std::endl; // OR file << m;	

התמודדות עם שגיאות – בחלק מהמתודות מצוינת בדיקה הכרחית שעליכם לעשות כחלק מהמתודה. במידה וזו נכשלת, עליכם לזרוק חריגה מסוג MatrixException אשר מצורף כקובץ h במודל - אין להגיש קובץ זה. החריגה צריכה להכיל את ההודעה המתאימה לשגיאה.

ההודעות המתאימות לכל סוג שגיאה:

	Description	Message
1	Check dimensions valid for operation.	"Invalid matrix dimensions.\n"
2	Check division by zero.	"Division by zero.\n"
3	Check indexes are in valid ranges.	"Index out of range.\n"
4	Check input stream validity.	"Error loading from input stream.\n"

2.1.2 גישה למטריצה לפי פרמטר יחיד

מטריצה הינה אובייקט דו מימדי (נדרשים שני אינדקסים על מנת לזהות תא באופן חד-חד ערכי). למימושים שונים נוח יותר לגשת לכל איבר במטריצה באמצעות אינדקס יחד. נבצע את המיפוי באופן הבא:
בהינתן מטריצה A , i אינדקס שורה, r אורך שורה במטריצה ו- j אינדקס עמודה:

$$A(i,j) = A[i*r+j]$$

2.1.3 output stream כתיבה

כאשר נכתוב ל-stream כלשהו את המטריצה היא תכתב באופן הבא:
לאחר כל ערך במטריצה יופיע תו רווח בודד (למעט האחרון האחרון בשורה).
בסוף כל שורה, תהיה ירידה שורה (למעט השורה האחרונה במטריצה).
כלומר, בהינתן מטריצה בגודל $3*3$ של אפסים בלבד, הדפסה שלה תהיה כך:

```
0 0 0\n0 0 0\n0 0 0
```

2.2 חלק שני – מימוש האופרטורים – Filters.cc

קבצים: בחלק זה עליכם לממש בקובץ Filters.cc אשר מסתמך על קובץ Filters.h אשר נמצא במודל, רק קובץ Filters.cc הוא להגשה - אין להגיש את קובץ Filters.h.

בחלק זה נממש 3 אופרטורים שונים מהעולם של עיבוד תמונה. הייצוג של תמונה בחלק זה של התרגיל יהיה בעזרת המחלקה מטריצה אותה אתם נדרשים לממש בחלק 2.1 של התרגיל.
*בחלק זה ניתן ומומלץ לכתוב לעצמכם פונקציות עזר.

האופרטורים אותם נממש (החתימות נמצאות ב Filters.h):

Operator	Signature	Comments
Quantization	Matrix quantization (const Matrix &image, int levels)	Performs quantization on the input image by the given number of levels. Returns new matrix which is the result of running the operator on the image.
Gaussian Blurring	Matrix blur (const Matrix &image);	Performs gaussian blurring on the input image. Returns new matrix which is the result of running the operator on the image.

Sobel operator (edge detection)	Matrix sobel (const Matrix &image)	Performs sobel edge detection on the input image. Returns new matrix which is the result of running the operator on the image.
---------------------------------	------------------------------------	--

2.2.1 הקדמה - קונבולוציה (convolution)

קונבולוציה זו פעולה בינארית בין שני אופרנדים (בדומה לחיבור, חיסור וכו'). מקובל להשתמש בה בתחומים רבים במתמטיקה, עיבוד אותות, סטטיסטיקה וכו'. בהקשר שלנו נסתכל על קונבולוציה על מטריצות (תמונות). יהיה לנו צורך בפעולה זו בפעולות הטשטוש ובאופרטור sobel. בהקשר שלנו, ניתן לדמיין שנתונה מטריצה כלשהי ונתונה לנו עוד מטריצה קטנה בגודל 3×3 (בתרגיל זה נעבוד עם קונבולוציות בגודל 3×3 בלבד) ואנחנו זזים עם המטריצה הקטנה כאילו הייתה חלון על המטריצה הגדולה ומבצעים חישוב ערכים מחודש.

ההסבר הטוב ביותר יהיה דרך דוגמא.

דו': בהינתן המטריצה A בגודל 5×5 הבאה והמטריצה B בגודל 3×3 :

130	45	2	254	34
9	17	65	190	54
23	56	175	178	178
56	78	90	67	45
67	49	30	29	28

1	1	2
3	4	1
0	9	8

ניצור מטריצה חדשה בגודל 5×5 (המימדים של A) ונאתחל אותה עם אפסים. כעת נרצה לחשב את התא $0,0$ במטריצה החדשה ע"י קונבולוציה. זה יהיה באופן הבא: נציב את מטריצת הקונבולוציה B "על" המטריצה A כאשר התא האמצעי $(1,1)$ במטריצה B נמצא על התא התואם לתא אותו אנו רוצים לחשב במטריצה החדשה – במקרה זה, התא $(0,0)$. זה יראה כך:

1	1	2			
3	140	45	2	254	34
0	99	17	65	190	54
	23	56	175	178	178
	56	78	90	67	45
	67	49	30	29	28

נחשב את הערך בתא $(0,0)$ במטריצה החדשה ע"י כפל של כל מספר במספר שנמצא "עליו" ונסכם, כאשר המספרים שנמצאים "באוויר" מוכפלים באפסים (זה נקרא padding). התוצאה של התא $0,0$ במטריצה החדשה יהיה:

$$8 \cdot 17 + 9 \cdot 9 + 0 \cdot 0 + 1 \cdot 45 + 4 \cdot 130 + 3 \cdot 0 + 2 \cdot 0 + 1 \cdot 0 + 1 \cdot 0$$

כאשר המספרים האדומים מגיעים מהמטריצה B (מטריצת הקונבולוציה) והמספרים השחורים מגיעים מהמטריצה A (המטריצה עליה אנחנו מבצעים את הפעולה).

לאחר מכן, נזוז עם החלון צעד אחד ימינה ונחשב את הערך של התא $(0,1)$ באותו אופן:

1	1	2		
130	44	21	254	34
90	19	65	190	54
23	56	175	176	178
56	78	90	67	45
67	49	30	29	28

וכך הלאה.
כאשר נרצה לחשב ערך של תא במרכז התמונה, למשל התא (3,2):

130	45	2	254	34
9	17	65	190	54
23	56	175	176	178
56	78	90	67	45
67	49	30	29	28

ההצבה תהיה כך ואין צורך בכפל באפסים (מה שקראנו לו למעלה padding).
באופן זה בסופו של תהליך ניתן לקבל מטריצה חדשה, שתהיה הקונבולוציה של A ו-B.
***** שימו לב – על מנת לשמור על אחידות בתרגיל – לאחר הקונבולוציה יכולים להיות ערכים לא שלמים. לכן, בכל פעם שתחשבו ערך קונבולוציה של תא מסוים במטריצה, בבקשה השתמשו בפונקציה rintf (מהספריה cmath) ורק אז שמרו את ערכו.**
***** שימו לב 2 – מופיע במודל מצגת המסבירה ומדגימה כיצד לבצע את הקונבולוציה. היעזרו בה!**

2.2.2 קוואנטיזציה (quantization)

בהקשר של עיבוד תמונה, פעולה זו נועדה על מנת לצמצם את מספר הצבעים בתמונה. כפי שציינו בתחילת התרגיל, מספר הגוונים האפשריים בתמונה הוא 256 (כל מספר טבעי בין 0 ל-255), פעולת הקוואנטיזציה מאפשרת לנו להוריד את מספר הגוונים לכל מספר שנרצה. בתרגיל זה נממש גרסה קלה יותר של קוואנטיזציה, שתבצע באופן הבא:

בהינתן תמונה כלשהי ומספר גוונים אשר נרצה שיהיה בתמונה החדשה, נחלק את סקלת הגוונים בין 0 ל-255 למספר הגוונים החדש שנרצה, נחשב את הגוון הממוצע בכל תחום חדש שנוצר, ובתמונה החדשה – כל גוון מקבל את הגוון הממוצע שנבחר עבור התחום שלו. דוגמא: נניח כי מספר הצבעים החדש אותו נרצה הוא 8.

נחלק את 256 הגוונים ל-8 חלקים ונקבל את סקאלת הצבעים הבאה:
 $32 = 256/8$ ולכן בכל רמה יהיו 32 צבעים.

* שימו לב, מספר הצבעים החדש יהיה חזקה של 2 ולא גדול מ-256 (אין צורך לוודא).
נבנה את המערך הבא:

0	32	64	96	128	160	192	224	256
---	----	----	----	-----	-----	-----	-----	-----

כעת, מערך זה מגדיר לנו את גבולות הצבעים:

גבול תחתון	גבול עליון
0	31
32	63
64	95
96	127
128	159
160	191
192	223
224	255

לכל קבוצת צבעים כזו נחשב ממוצע (מעגלים למטה אם יצא מספר לא עגול)

והתוצאה המתקבלת היא:

גבול תחתון	גבול עליון	צבע חדש
0	31	15
32	63	47
64	95	79
96	127	111
128	159	143

175	191	160
207	223	192
239	255	224

ובתמונה החדשה נחליף את הצבעים כך ש-:
 כל הפיקסלים בתמונה עם צבעים בין 0 ל31 הופכים ל15.
 כל הפיקסלים בתמונה עם צבעים בין 32 ל63 הופכים ל47.
 וכך הלאה.
 בסוף, התמונה שנעדין תהפוך מ256 גוונים אפשריים, ל8 גוונים אפשריים.

2.2.3 טשטוש (blurring)

בחלק זה נבנה אופרטור המבצע פעולת טשטוש. פעולה זו יחסית פשוטה, בהינתן תמונה (מטריצה), כל שנצטרך לעשות הוא להפעיל קונבולוציה עם המטריצה:

$$\left(\frac{1}{16}\right) * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

ולהחזיר את המטריצה המתקבלת. פעולה זו נקראת "טשטוש גאוסיאני".

**** העשרה – שימו לב מה קורה בפעולת הקונבולוציה עם מטריצה זו, אנו בעצם הופכים את הערך של כל תא, להיות תלוי לינארית בתאים הקרובים אליו כאשר אנחנו נותנים דגשים שונים לכל תא מסביבו. פעולה זו, גורמת ל"ערבוב" של הגוונים בתמונה יחד עם אלו הקרובים אליהם ולכן זה נראה כמו טשטוש.**

2.2.4 זיהוי גבולות (edges detection)

בחלק זה נבנה אופרטור המבצע הדגשה של הגבולות בתמונה. אנחנו נבנה אופרטור שנקרא "סובל" (sobel). הרעיון באופרטור זה הוא שהוא פועל על גבי שני צירי התמונה, ציר ה-X וציר ה-Y, לחוד ואז מחבר את התוצאה יחדיו. כלומר הפעולה כאן מתחלקת ל-3 שלבים:

(1) הפעלת קונבולוציה עם המטריצה:

$$G_x = \frac{1}{8} * \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

(2) הפעלת קונבולוציה עם המטריצה:

$$G_y = \frac{1}{8} * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

(3) החזרת התוצאה:

$$(G_x * \text{Img}) + (G_y * \text{Img})$$

כאשר הסימן * מסמן קונבולוציה.
כלומר, מבצעים קונבולוציה של התמונה עם G_x .
מבצעים קונבולוציה של התמונה עם G_y .
ומחברים את שתי התוצאות.

***** שימו לב – במידה ובאחד האופרטורים ערכי התמונה המתקבלת גולשים מעבר לגבולות 0 ו-255, עליכם לצמצם אותם חזרה באופן הבא: אם תא כלשהו עם ערך גדול מ-255, הוא הופך ל-255. אם תא כשלהו קטן מ-0, הוא הופך לאפס.**

******* סרטון המדגים איך sobel עובד: <https://www.youtube.com/watch?v=iendD-lqoog> (הסרטון לשם אינטואיציה בלבד ואין לראות בו כהסבר לשום פרט בתרגיל).

3 קלט - פלט

במסגרת הקבצים שסיפקנו לכם עבור התרגיל מופיע קובץ main אשר מקבל כקלט שלושה ארגומנטים:
(1) נתיב (אבסולוטי) לקובץ המייצג תמונה (נוצר בעזרת התוכנית image2file שסופקה לכם, ראו סעיף 4).
(2) שם של פילטר שיהיה אחד מהבאים: "sobel", "blur", "quant".
(3) נתיב לקובץ פלט שיכלול הדפסה של מטריצת התמונה המתקבלת (אם תרצו לראות אותו, העזרו בקובץ file2image, ראו סעיף 4).

התוכנית שלכם צריכה לחשב את הפילטר המתאים על גבי התמונה המתאימה ולהחזיר את מטריצת התוצאה המתאימה. (קראו את הקובץ main.cc בזהירות וודאו שהנכם מבינים אותו היטב).

3.1 הנחות לגבי התמונות

בתרגיל זה נעבוד עם תמונות בגודל $128 * 128$ בגווי שחור-לבן בלבד. את התמונות ניתן לייצר בעזרת תוכניות הפייתון שסופקו לכם (ראו סעיף 4). ניתן להניח כי כל הערכים בתמונה הללו יהיו מספרים בין 0 ל-255 ואין צורך לוודא זאת.

4 קבצי עזר

(1) לנוחיותכם, סופקה לכם תוכנית main (הקובץ main.cc) אשר מבצע את הקריאה לפונקציות אותן אתם נדרשים לממש. קובץ זה איננו להגשה. הוא כולל את פקודת הmain איתה נריץ את התוכנית שלכם לצורך ייצור תמונות. וודאו כי הקוד שלכם מתקמפל עם תוכנית זו בהצלחה.
(2) כפי שנכתב בחלק הנוגע לשגיאות, ניתן לכם קובץ בשם MatrixException.h אשר מכיל מחלקה של חריגה. אין להגיש קובץ זה. שימוש בקובץ זה ייעשה על ידי זריקה של חריגות בקובץ Matrix.cc.
(3) בנוסף, בתרגיל זה נספק לכם שתי תוכניות python שיוכלו לסייע לכם על מנת לייצר טסטים ולבחון את הקוד שלכם.

התוכנית הראשונה:

* **image2file.py** – תוכנית זו מקבלת כקלט קובץ תמונה וממירה אותה לקובץ. ניתן יהיה להשתמש בה על מנת לבחון את התוכנית שלכם עם כל תמונה שתמצאו. שימו לב לאופן בו התוכנית מקבלת קלט (מופיע בקובץ עצמו). שורת ההרצה:

<FILE>=f- <SHOW>=s- <IMG_PATH>=python3 image2file.py -p

כאשר:

IMG_PATH - נתיב אבסולוטי לקובץ התמונה (נסו לעבוד עם jpg).

SHOW - מקבל 1 או 0 באינדיקטור בוליאני האם להציג את התמונה או לא.

FILE - אינדיקטור בוליאני גם כן המקבל 1 או 0, האם לשמור את הקובץ הנוצר (נשמר לתיקיה הנוכחית).

* **file2image.py** - תוכנית זו מקבלת נתיב לקובץ בפורמט זהה לזה שפולטת התוכנית image2file (ראו קבצים לדוגמא) ומחזירה קובץ תמונה. בעזרת שתי התוכניות האלו תוכלו ליצור קבצי מבחן ולבדוק שהתוצאות של התוכנית שלכם תואמות למצופה. שורת ההרצה:

<FILE_PATH>=python3 file2image.py -p

כאשר:

FILE_PATH - נתיב לקובץ המתאר תמונה בפורמט שנפלט ע"י הקובץ image2file.py ופולט קובץ תמונה שיקרא image.jpg בתיקיה הנוכחית.

5 הערות ודגשים

* כלל קבצי הקוד שניתנו לכם: MatrixException.h Filters.h main.cc Matrix.cc Matrix.h וקבצי פייתון לעזר כפי שפורטו בסעיף 4.

(1 עליכם להגיש קובץ tar שייקרא ex5.tar והוא יכלול את הקבצים הבאים ואותם בלבד:
Filters.cc Matrix.cc Matrix.h

הפקודה ליצירת קובץ tar היא: **tar -cvf ex5.tar Matrix.h Matrix.cc Filters.cc**

(2 בתרגיל זה נשתמש כמצוין בתקנון הקורס בגרסת **c++14**.

(3 אין להשתמש בשום מבנה נתונים מספריית STL או בקונטיינרים דינמיים.

(4 אין להשתמש בשום מילה שמורה/פונקציה משפת C.

(5 הקפידו על שימוש נכון ב-const!!!

(6 השתמשו ב noexcept היכן שצריך.

(7 הקפידו לא לשלוח משתנים by value אם אין בכך צורך.

(8 ספריות מותרות לתרגיל: cmath, iostream, fstream, string

6 הגשת התרגיל

• וודאו כי התרגיל שלכם מתקמפל ועובד על מחשבי בית הספר בעזרת הפקודה:

o compiledProgram- <code files> Werror -g -lm -std=c++14- g++ -Wall -Wvla -Wextra

• וודאו כי התוכנית שלכם עוברת בדיקת ולגריינד ללא שום שגיאה או הערה בעזרת הפקודה:

<Command to debug> valgrind -leak-check=full

• קראו בקפידה את הוראות התרגיל וודאו כי הנכם משתמשים בגרסה המתאימה של השפה.

• אנא וודאו כי התרגיל שלכם עובר את סקריפט ה-submission-pre- ללא שגיאות או אזהרות. הסקריפט זמין בנתיב:

~labcc/presubmit/ex5/run <path_to_submission>

שימו לב כי בניגוד לתרגילים קודמים, כעת הסקריפט ירוץ בעת הגשת התרגיל ותוצאותיו ישלחו למייל. כמו כן, התוצאות מהרצה ידנית מהנתיב הנתון לא ישלחו למייל.

• כתבו את הודעות המופיעות בתרגיל בעצמכם ואל תעתיקו. העתקת הודעות מהקובץ יכולה להוסיף תווים מיותרים ולפגוע בבדיקה האוטומטית המנקדת את עבודתכם.

- **פתרון בית הספר** – במסגרת תרגיל זה לא יסופק פתרון בית ספר. סיפקנו לכם קבצים במודל שיאפשרו לכם לבצע diff אל מול תוצאות האופרטורים שלנו וכך תוכלו לדעת שביצעתם את הפעולות השונות באופרטורים בהצלחה. קריאה והבנה מעמיקה של החומר שהועבר בהרצאות ובתרגולים תסייע לכם לכתוב את ספריית המטריצה בהצלחה ולכן אין צורך בפתרון כזה.

בהצלחה!



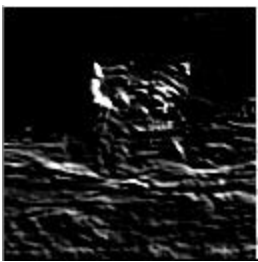

נספח – דוגמאות לפלטים

התמונות שסיפקנו לכם במודל הן (קיבלתם גם את התמונות וגם את "קבצי המטריצה" שיוצרו בעזרת image2file.py).

לנה	גבעת רם	כלב
		

אופרטור	קלט	פלט
---------	-----	-----

		קוונטיזציה עם 16 רמות
		קוונטיזציה עם 8 רמות
		קוונטיזציה עם 4 רמות
		קוונטיזציה עם 2 רמות
		טשטוש גאוסיאני
		טשטוש גאוסיאני

		<p>סובל</p>
		<p>סובל</p>