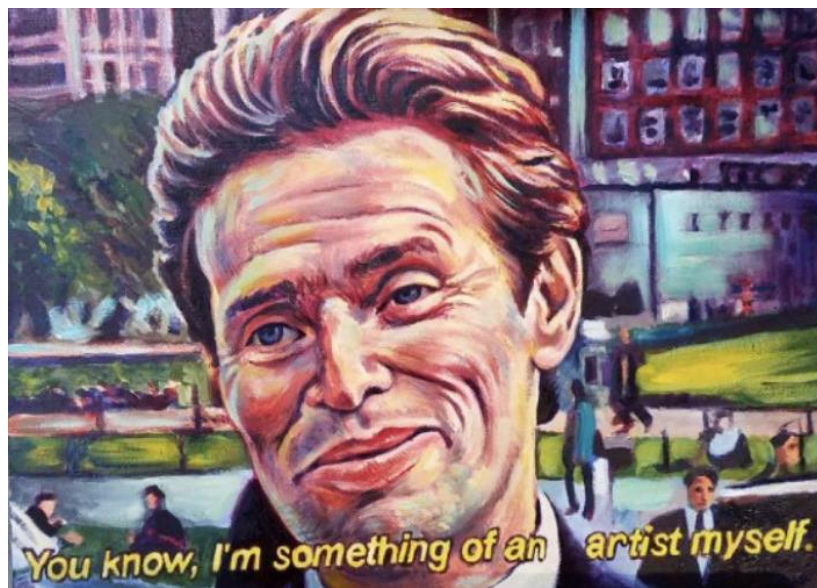


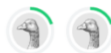
# I'm Something of a Painter Myself

Use GANs to create art - will you be the next Monet?



17

Vincent GAN Gogh



38.55316

Team: Orel Ben Zaken  
Omer Luxembourg

<https://github.com/omerlux/Something-of-a-Painter>

## מטרת ה-Challenge:

בניית מודל שלוקח תמונה שצולמה ע"י מצלמה ויוצר ציור של אותה תמונה בסגנון הצייר מונה (Monet).

### תיאור מתמטי:

נסמן: ב-  $X$  - תחום המכיל תמונות שצולמו ע"י מצלמה.  
ב-  $Y$  - תחום המכיל ציורי מונה.

אנו צריכים למצוא פונקציה  $G: X \rightarrow Y$  כך שעבור כל תמונה  $x \in X$ , שצולמה ע"י מצלמה, נקבל את הציור מונה  $y \in Y$  המתאים לה, כלומר, נרצה שיתקיים  $y = G(x)$ .

ב-Fig 1 ניתן לראות תיאור ויזואלי של משימה:  
מערכת שה-input שלה תמונה שצולמה ע"י מצלמה ו-output שלה הוא ציור בסגנון מונה של התמונה.

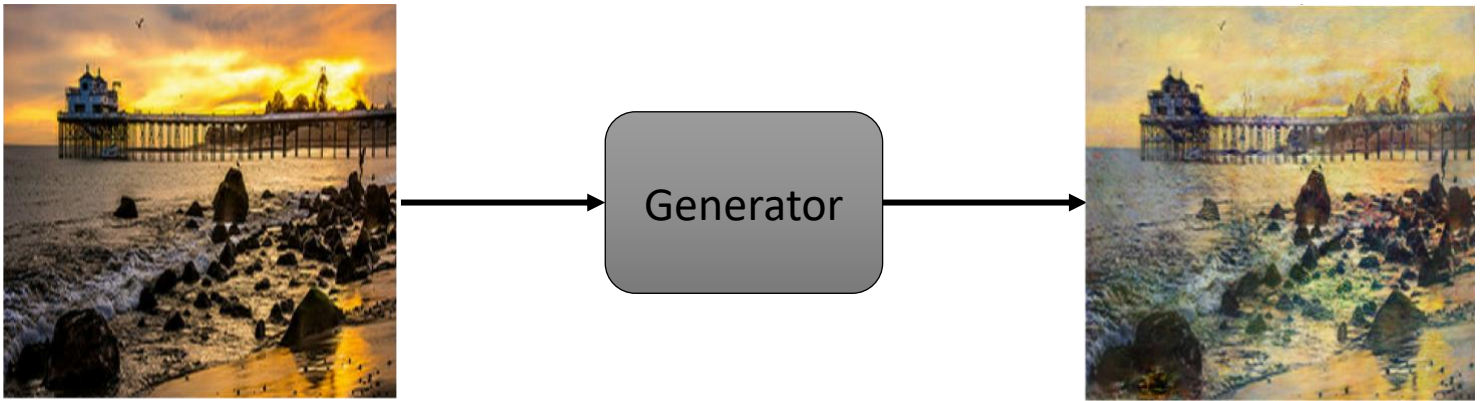


Fig 1: visual description of the challenge goal

## מדד להערכת הביצועים:

אנו רוצים ליצור ציורים "שנראים" כמו ציורים בסגנון מונה (Monet) ולכן, אנו צריכים מדד שיוכל להגיד לנו עד כמה הציורים שיצרנו "קרובים" לציורי Monet. המדד נקרא **MiFID** (Memorization-informed Fréchet Inception Distance) שמבוסס על המדד FID (Fréchet Inception Distance).

**FID:** מדד המשמש להערכת איכות התמונות שנוצרו על ידי מודל יוצר (generative model).

- בשלב הראשון משתמשים ברשת pretrained שנקראת Inception network כדי לחלץ פיצרים עבור כל תמונה (התמונות האמיתיות והתמונות שנוצרו מה-generator).
- ממדלים את הסתברות הדאטא של הפיצרים שחולצו באמצעות מודל גאוסי רב מיימדי.
- נסמן:  $\mathcal{N}(\mu_r, \Sigma_r)$  את ההסתברות הגאוסית של הפיצרים המתאימים לתמונות האמיתיות וב-  $\mathcal{N}(\mu_g, \Sigma_g)$  את ההסתברות הגאוסית של הפיצרים המתאימים לתמונות שיוצרו ע"י ה-generator.
- ולבסוף מחשבים את המדד FID לפי הנוסחא הבאה:

$$(1) FID = \|\mu_r - \mu_g\|^2 + Tr\left(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{\frac{1}{2}}\right)$$

\*נרצה שה-FID יהיה כמה שיותר קטן.

**MiFID:** מדד מבוסס על ה-FID ומעניש במקרה שבו יש שינוי של התמונות האמיתיות מהסט אימון. כלומר, יש גורם תיקון שמטרתו היא "להעניש" ולהעלות את המדד במקרה שבו ה-generator משנן את הסט אימון ולומד לייצר רק את התמונות האמיתיות שמופיעות באימון. הנוסחא ל-MiFID נתונה ע"י:

$$(2) MiFID = FID \cdot \frac{1}{d_{thr}}$$

\*ה-FID הוא מדד של מרחק בין הסתברויות והתוספת של הגורם "ענישה" באה למנוע מצב שבו אנשים מגישים ל-Kaggle את הציורי Monet מה-train כדי לקבל FID נמוך.

\*התמונות האמיתיות, ציורי Monet במקרה זה, המשמשות לחישוב ה-FID ב-Kaggle אלה לא רק הציורי Monet הנתונים בסט אימון, כלומר, יש סט ציורים שונה מזה של האימון המשמש להערכת המדד ולכן, לא נוכל לחשב אותו בצורה מדויקת במהלך האימון ונוכל לחשב אותו רק כאשר נבצע הגשה ל-Kaggle.

## דרך החישוב של $d_{thr}$ :

נסמן: ב-  $f_{rj}$  – את וקטור הפיצורים שחולצו ע"י ה-Inception network עבור התמונה האמיתית (ציור Monet אמיתי) ה- $j$ .

ב-  $f_{gi}$  – את וקטור הפיצורים שחולצו ע"י ה-Inception network עבור התמונה ה- $i$  שנוצרה ע"י ה-generator (ציור Monet שנוצר ע"י ה-generator).

שלב 1: עבור כל זוג שמורכב מתמונה אמיתית  $j$  ומתמונה שנוצרה ע"י  $i$  generator מחושב המרחק בין התמונות בצורה הבאה:

$$(3) d_{ij} = 1 - \frac{f_{gi} \cdot f_{rj}}{|f_{gi}| |f_{rj}|}$$

כאשר, הביטוי  $\frac{f_{gi} \cdot f_{rj}}{|f_{gi}| |f_{rj}|}$  שווה לקוסינוס הזווית בין שני הוקטורים (מהנוסחא של מכפלה פנימית) כלומר, המרחק בין שתי התמונות יותר קטן ככל שהזווית שבין שני וקטורי הפיצורים יותר קטנה.

שלב 2: עבור כל תמונה  $i$  שנוצרה ע"י ה-generator מוצאים את התמונה האמיתית  $j$  שהמרחק בינה הוא המינימלי בהשוואה למרחק משאר התמונות האמיתיות ואז ממצעים על כל המרחקים המינימלים על פני כל התמונות שנוצרו ע"י ה-generator. מתמטית אנו מחשבים את המחק המינימלי הממוצע על פני כל התמונות שנוצרו ע"י ה-generator הצורה הבאה:

$$(4) d = \frac{1}{N} \sum_i \min_j d_{ij}$$

כאשר,  $N$  מייצג את מספר התמונות שנוצרו ע"י ה-generator.

שלב 3: מגדירים ערך סף  $0 < \varepsilon < 1$  וכאשר המרחק המינימלי הממוצע  $d$  קטן ממנו זה אומר שהתמונות שנוצרו ע"י ה-generator דומות מדי לתמונות האמיתיות. מבחינה מתמטית  $d_{thr}$  מחושב כך:

$$(5) d_{thr} = \begin{cases} d, & \text{if } d < \varepsilon \\ 1, & \text{else} \end{cases}$$

## המודל CycleGAN [1]:

מה זה CycleGAN:

סוג של GAN שמאפשר ללמוד טרנספורמציה של תמונה מתחום  $X$  לתמונה מתחום  $Y$  כאשר לא נתונים הזוגות המתאימים. כלומר לכל תמונה  $x_i \in X$  לא נתון  $y_i \in Y$  המתאימה לה.

המטרה שלנו היא למצוא פונקציית מיפוי  $G_Y: X \rightarrow Y$  כך שהתמונות מ- $G_Y(X)$  יהיו מההתפלגות של  $Y$ .  
אם נשתמש במודל GAN קלאסי עם adversarial loss בלבד, אז לכל תמונה  $x_i \in X$  מיפוי זה יכול לתת תמונה כלשהי מתוך מתחום  $Y$  ואנו רוצים את התמונה  $y_i \in Y$  המסוימת שמתאימה לתמונה  $x_i$ .  
לכן, כדי להגביל את המיפוי של הפונקציה  $G_Y$  ה-CycleGAN לומד את המיפוי ההופכי  $G_X: Y \rightarrow X$  ובאמצעות שימוש ב-cycle consistency loss (שיוסבר בהמשך) "מכריחים" את הפונקציה  $G_Y, G_X$  לשמור תכונות של ה-input שלהן כך שיתקיים:  
 $G_X(G_Y(x)) \approx x, G_Y(G_X(y)) \approx y$

הקשר בין המודל לבעיה שלנו:

- במקרה שלנו  $X$  זה התחום של התמונות שצולמו ע"י מצלמה ו- $Y$  זה התחום של הצילומי Monet.
- אנו רוצים למצוא פונקציה  $G_Y: X \rightarrow Y$  שלוקחת תמונה שצולמה ע"י מצלמה והופכת אותה לצילום בסגנון Monet המתאים. כלומר, פונקציית המיפוי צריכה לשמור ולזהות תכונות של input על מנת שתוכל להציג אותם כצילום.
- אם נשתמש במודל GAN הקלאסי עם ה-adversarial loss בלבד, פונקציית המיפוי יכולה לתת לנו במוצא צילום כלשהו של Monet שלא קשור ל-input שהוא תמונה ממצלמה ואנו רוצים שהמערכת תיצור צילום ספציפי בסגנון Monet של ה-input.

הסיבה לשימוש ב-CycleGAN בבעיה שלנו:

באמצעות שימוש ב-CycleGAN ניתן "להכריח" את פונקציית המיפוי  $G_Y$  ללמוד תכונות של ה-input כך שבמוצא נקבל תמונה שנראית כמו צילום בסגנון Monet של ה-input (תמונה שצולמה ע"י מצלמה).

הארכיטקטורה של ה-CycleGAN:

כפי שמתואר ב-Fig 2 המודל כולל שתי פונקציות מיפוי (generators)  $G_Y: X \rightarrow Y, G_X: Y \rightarrow X$  ובנוסף שני discriminators  $D_X, D_Y$ .  
כאשר,  $D_X$  מטרתו להבחין בין תמונה אמיתית שצולמה ע"י מצלמה לבין "זיוף" ו- $D_Y$  מטרתו להבחין בין צילום Monet אמיתי לבין "זיוף".

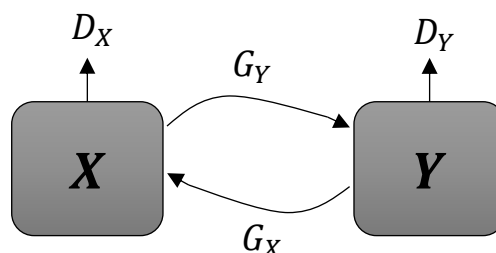


Fig 2: CycleGAN model

ארכיטקטורה של ה-*generators*:  
 בכל אחד מה-*generators* השתמשנו בארכיטקטורה שמבוססת על *U-Net*.

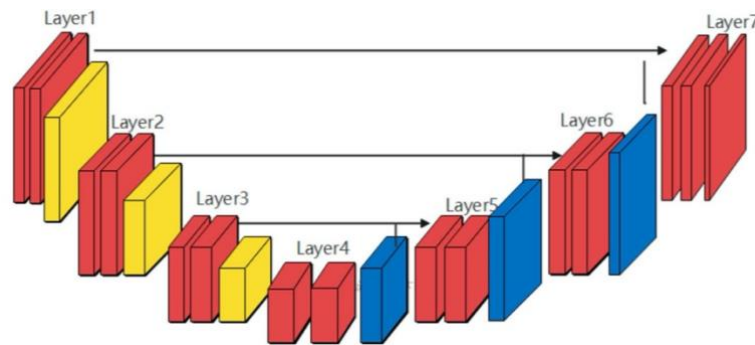


Fig 3: example for U-Net architecture

ארכיטקטורה של ה-*discriminators*:  
 אנו משתמשים בארכיטקטורה שמתאמה לגישת *PatchGAN* [2], שכבות קונבולוציה ו-*down samples* כאשר במוצא כל *discriminator* מקבלים *patch* בגודל  $30 \times 30$  כפי שמתואר ב-  
 Fig 4  
 כל פיקסל  $ij$  ב-*patch* בגודל  $30 \times 30$  מייצג האם ה-*patch*  $ij$  בתמונה (ה-*input*) הוא *real* או *fake*.

\* כדי לדעת איזה *patch* מהתמונה המקורית מתאים לפיקסל  $ij$  במוצא ה-*discriminator*, אפשר להתחקות אחרי ה-*receptive field* שמתאים לו.

\* מתמטית שימוש ב-*discriminator* בגישת *PatchGAN* שקול למצב שבו היינו "חותכים ידנית" את התמונה ל- $30 \times 30$  *patch*-ים חופפים, ונריץ *discriminator* "רגיל" (שבמוצא שלו יש נירון יחיד) על כל *patch* ונמצע על התוצאות.

\* ה-*discriminator* המושלם יוציא *patch* של 1-ים עבור תמונות שהן *real* ו-0-ים עבור תמונות שהן *fake*.

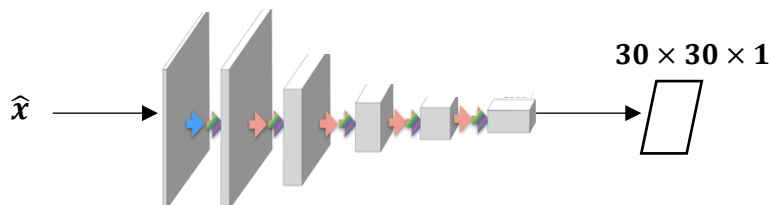


Fig 4: example for U-Net architecture

## פונקציות Loss:

הפונקציית loss הכוללת מורכבת ממספר פונקציות loss. נעבור על כל פונקציית loss ובסוף נציג את הפונקציית loss הכוללת.

### 1. Adversarial Loss:

ה-CycleGAN מורכב משני GAN-ים ולכן עבור כל GAN אנו מקבלים ה-adversarial loss.

$$(6) \mathcal{L}_{Adv}(G_Y, D_Y) = \mathbb{E}_{y \sim p_Y(y)} [\log(D_Y(y))] + \mathbb{E}_{x \sim p_X(x)} [\log(1 - D_Y(G_Y(x)))]$$

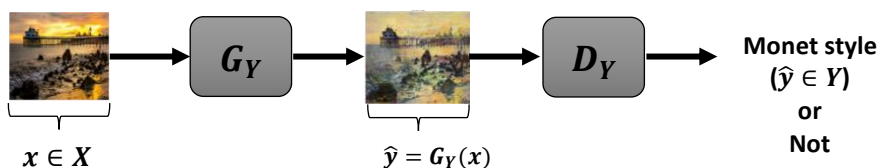


Fig 5: GAN that create Monet style images (Eq (6))

$$(7) \mathcal{L}_{Adv}(G_X, D_X) = \mathbb{E}_{x \sim p_X(x)} [\log(D_X(x))] + \mathbb{E}_{y \sim p_Y(y)} [\log(1 - D_X(G_X(y)))]$$

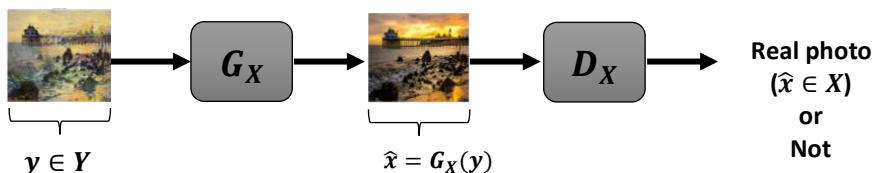


Fig 6: GAN that create real photos (Eq (7))

### 2. Cycle Consistency Loss:

כדי לצמצם את המרחב האפשרי אליו פונקציות המיפוי (generators) יכולים להתכנס וכדי "להכריח" את ה-generators ללמוד תכונות חשובות מ-input כך שיהיה ניתן לשחזר אותו.

$$(8) \mathcal{L}_{cyc}(G_X, G_Y) = \mathbb{E}_{x \sim p_X(x)} [\|G_X(G_Y(x)) - x\|_1] + \mathbb{E}_{y \sim p_Y(y)} [\|G_Y(G_X(y)) - y\|_1]$$

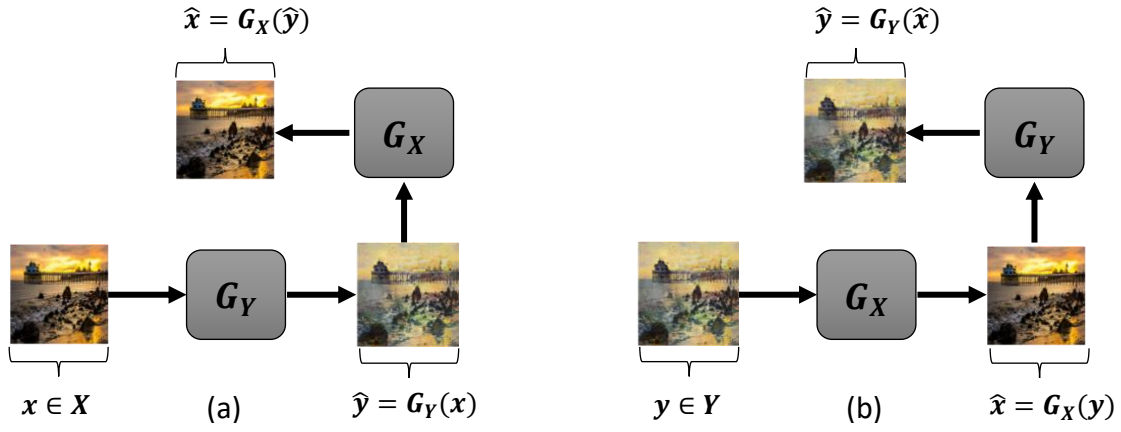


Fig 7: image translation cycles for cycle-consistency losses.

(a) reconstruction of Monet image and (b) reconstruction real photo image.

### 3. Identity loss

אם למשל יש לנו generator שממפה תמונות לתחום  $Y$  ונתנו לו כ-input תמונה מתחום זה אז היינו רוצים שהוא "יצליח להבין" שהוא קיבל תמונה מאותו תחום שאליו הוא ממפה ושיוציא את התמונה שהוא קיבל ב-input בתור ה-output שלו. זה "מכריח" את ה-generator ללמוד איזה תכונות מאפיינות את התחום שאליו הוא ממפה.

$$(9) \mathcal{L}_{id}(G_X) = \mathbb{E}_{x \sim p_X(x)} [\|G_X(x) - x\|_1]$$

$$(10) \mathcal{L}_{id}(G_Y) = \mathbb{E}_{y \sim p_Y(y)} [\|G_Y(y) - y\|_1]$$

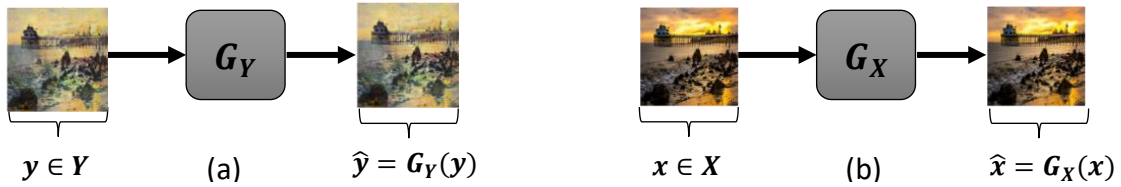


Fig 8: (a) Identity forward of Monet image and (b) Identity forward of real photo image.

### Total loss

$$(11) \mathcal{L}_{TOT}(G_X, G_Y, D_X, D_Y) = \mathcal{L}_{Adv}(G_Y, D_Y) + \mathcal{L}_{Adv}(G_X, D_X) + \lambda_{cyc} \mathcal{L}_{cyc}(G_X, G_Y) + \lambda_{id} (\mathcal{L}_{id}(G_X) + \mathcal{L}_{id}(G_Y))$$

המטרה: למצוא פונקציות מיפוי  $G_X^*, G_Y^*$  המקיימות:

$$(12) G_X^*, G_Y^* = \underset{G_X, G_Y}{\operatorname{argmin}} \max_{D_X, D_Y} \mathcal{L}_{TOT}(G_X, G_Y, D_X, D_Y)$$



## בעיות שיכולות להיווצר ב-CycleGAN:

### 1. דאטאסט "קטן מידי":

במקרה שבו יש לנו כמות קטנה יחסית של תמונות מאחד התחומים יכול להיווצר מצב שבו ה-discriminator נכנס ל-overfitting. בבעיה שלנו יש 300 ציורים של Monet ו-7038 תמונות שצולמו ע"י מצלמה. כלומר, יש לנו כמות קטנה של ציורי Monet ביחס לכמות התמונות מהעולם האמיתי. אם ה-discriminator של הציורי Monet יכנס ל-overfitting זה יבוא לידי ביטוי בכך שהוא ישנן את ה-300 ציורים שנתונים באימון ואם ניתן לו ציור Monet שלא מופיע באימון הוא יחשוב שזה לא ציור אמיתי.

\* בכל step בתוך אפוק במהלך האימון, אנו משתמשים גם בתמונה שצולמה ע"י מצלמה וגם ציור Monet כדי שבכל step יאומנו שני ה-GANים (האחד שיוצר ציורי Monet והשני שיוצר תמונות מהעולם האמיתי). כלומר, בכל אפוק צריכה להיות אותה כמות של ציורי Monet ואתה כמות של תמונות שצולמו ע"י מצלמה ומכיוון שאין את אותה כמות של תמונות מכל סוג אז או שיהיה מצב של under-sampling או של over-sampling.

#### דרכים לנסות לפתור את בעיית הדאטא הלא מאוזן:

- **Under-sampling**: צמצום התמונות שצולמו ע"י מצלמה. בכל אפוק אנו מתמשים ב-300 תמונות מכל סוג (ציורים או תמונות שצולמו ע"י מצלמה). החיסרון הוא שהמודל לא רואה את כל התמונות שצולמו ע"י מצלמה שיש לנו בדאטא במהלך כל אפוק בשלב האימון.
- **Over-sampling**: איזון הדאטא ע"י העלאת כמות התמונות מהתחום עם הכמות הקטנה יותר (ציורי Monet במקרה זה). ההעלאה של הכמות נעשית ע"י שכפול של התמונות הקיימות, כלומר, בכל אפוק אותה תמונה יכולה להופיע מספר פעמים. החיסרון פה הוא שהמודל ישנן את התמונות מהתחום הקטן ותר (הציורי Monet).

#### דרכים לנסות לפתור את בעיית ה-overfitting:

- **Augmentation**: במקרה של over-sampling על מנת למנוע את המצב שבו כל ציור Monet יכול להופיע מספר פעמים, ניסינו להשתמש באוגמנטציות אקראיות שפועלות על התמונות בשלב ה-preprocessing כלומר לפי ההכנסה לרשת. החיסרון שבשימוש באוגמנטציות הוא, שהן יכולות להשפיע על ההתפלגות ממנה באו התמונות שה-GAN "רואה" ובכך לפגוע בביצועים. האוגמנטציות שבהן השתמשנו הן: Flip ו-Rotation ב-Fig 9 ניתן לראות דוגמאות לאוגמנטציות.

\* על מנת שבכל אפוק יהיו אוגמנטציות אחרות ביצענו את זה באמצעות layers שמבצעות אוגמנטציות אקראיות שהן חלק מהמודל (כפי שלמדנו בעבודה 1), כאשר, שכבות אלה מופעלות רק בתהליך האימון ואין להן פרמטרים ללמידה. כלומר, layers אלה משמשות כ-preprocessing.

\* במקרה זה אנו רוצים לייצר ציורים (של Monet) ולכן נשתמש באוגמנטציות שלא פוגעות בגווני צבעים כדי שישפיעו כמה שפחות על ההתפלגות ממנה הגיעו הציורי Monet.

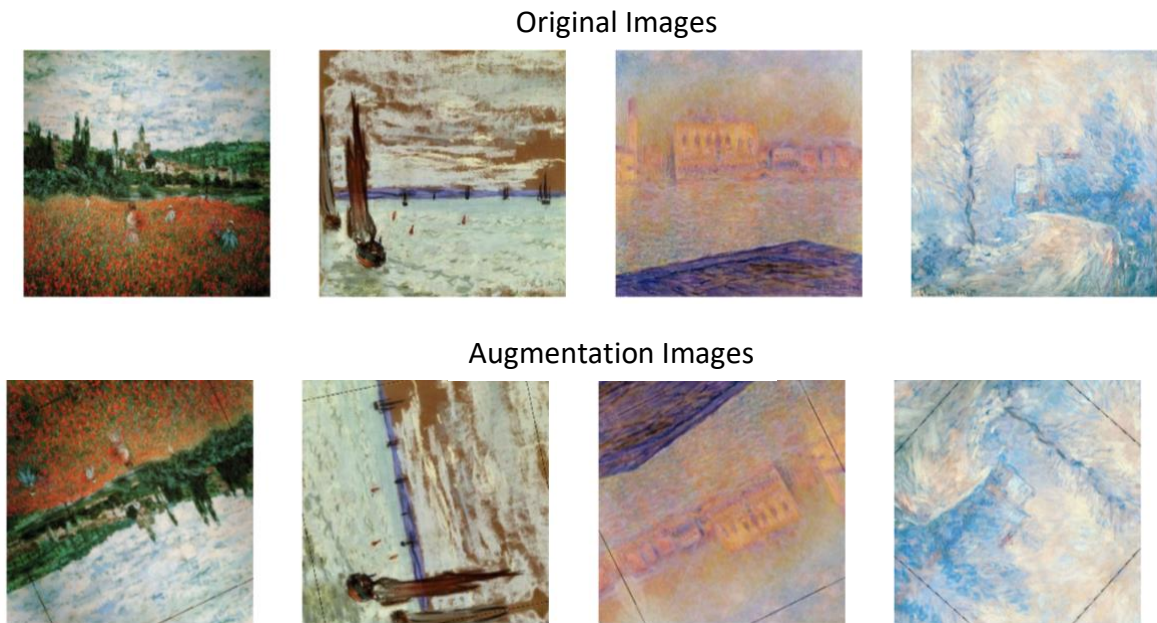


Fig 9: Original images and images with augmentations (Flip and rotation).

- **Diff-Augmentation [3]:** שיטה שבה משתמשים באוגמנטציות שניתן לייצג אותן באמצעות פונקציות גזירות כחלק מתהליך האימון כפי שמתאור ב-Fig 10. בהשוואה לאוגמנטציות "רגילות" (שבשלב ה-preprocessing) ב-Diff-augmentation האימון "מתחשב" באוגמנטציות אלה בתהליך ה-back propagation ולכן הדרישה היא שהן יהיו גזירות. החיסרון הוא שהאוגמנטציות הגזירות הן אוגמנטציות שמשנות את הגוונים בתמונה (brightness, contrast) או שהן מאפסות חלקים בתמונה (cutout) וזה עלול להטעות את ה-GAN. ב-Fig 11 ניתן דוגמאות לאוגמנטציות: brightness, contrast and cutout.

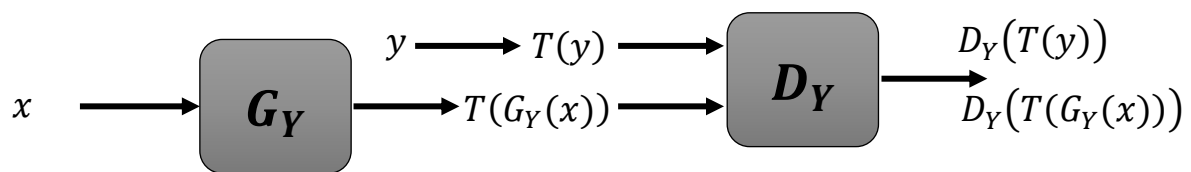


Fig 10: diff augmentation in block diagram.

The function  $T(\cdot)$  it's the differentiable augmentation. The discriminator gets augmentation of the real image  $y$  and augmentation of the generated image  $G_Y(x)$ .

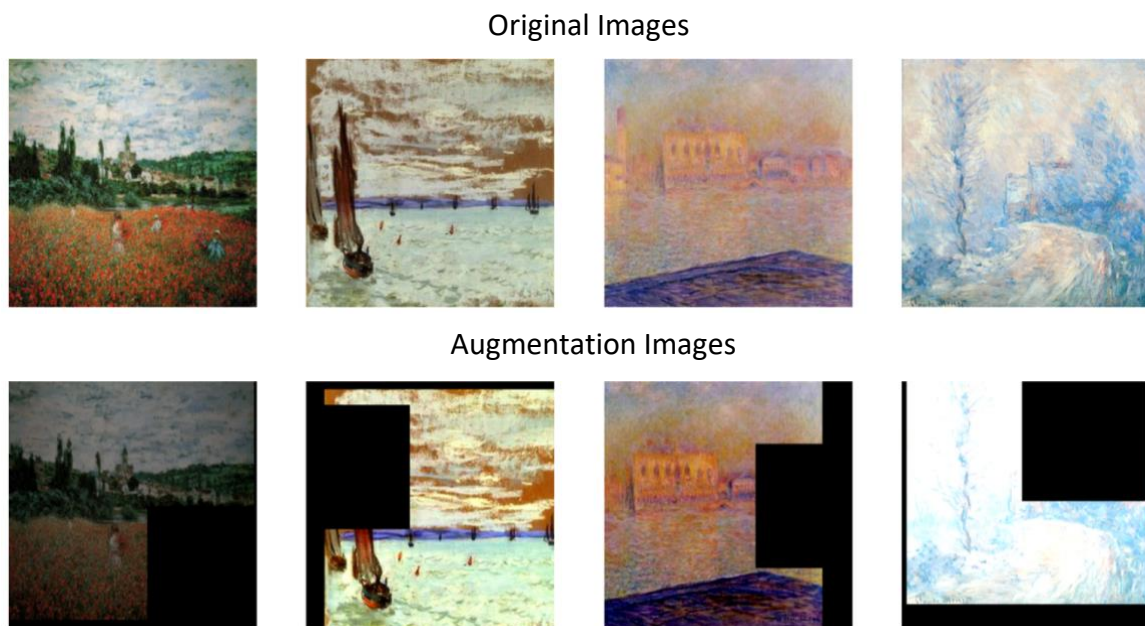


Fig 11: Original images and images with diff augmentation (brightness, contrast, and cutout).

## 2. Self-adversarial attack [4]:

CycleGAN, באמצעות ה-cycle-consistency loss או "מכריחים" את ה-generator לשמור תכונות של ה-input במוצא כך שיהיה ניתן לשחזר את ה-input. הבעיה שיכולה להיווצר היא שהמודל "ישקיע" יותר בשיחזור מאשר בשימור על תכונות חשובות מה-input שנראות לעין. כלומר, הגבלת השיחזור יכולה לגרום ל-generator "להחביא" את התכונות החשובות לשיחזור בתוך רעש במוצא שלא ניתן לזהויו ע"י העין האנושית או ע"י ה-discriminator.

\* בפועל בעיה זו באה לידי ביטוי בכך שהשחזור לא חסין להוספת רעש, כלומר, אם נוסיף רעש למוצא ה-generator ואז ננסה לשחזר את ה-input באמצעות ה-generator השני נקבל תמונה שלא נראית בכלל כמו ה-input כלומר, לא נצליח לשחזר (זה ינבע מכך שהתכונות החשובות נשמרות בתוך רעש בתמונה).  
ב-Fig 12 ניתן לראות דוגמא לבעיה זו שלקוחה מהמאמר [4].

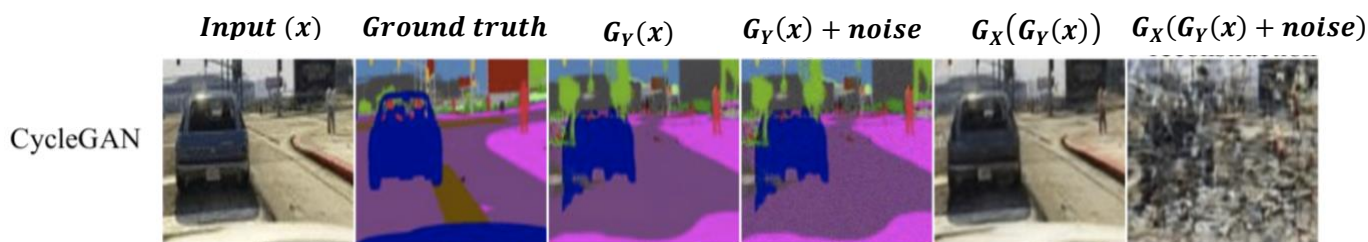


Fig 12: example of reconstruction of CycleGAN that trained to create segmentation. The noise is gaussian noise  $N(0, 0.08)$ .

ב-Fig 13 ניתן לראות איך תוספת רעש משפיעה על השיחזור של ציור Monet של ה-CycleGAN בבעיה שלנו. המודל שמוצג הוא CycleGAN מאומן ללא אוגמנטציות עם over-sampling אך מקבלים את אותה המסקנה גם עבור שאר המודלים שאומנו. מה-Fig 13 נובע שהשיחזור חסין להוספת רעש, כלומר, בעיה זו לא קיימת במקרה זה.

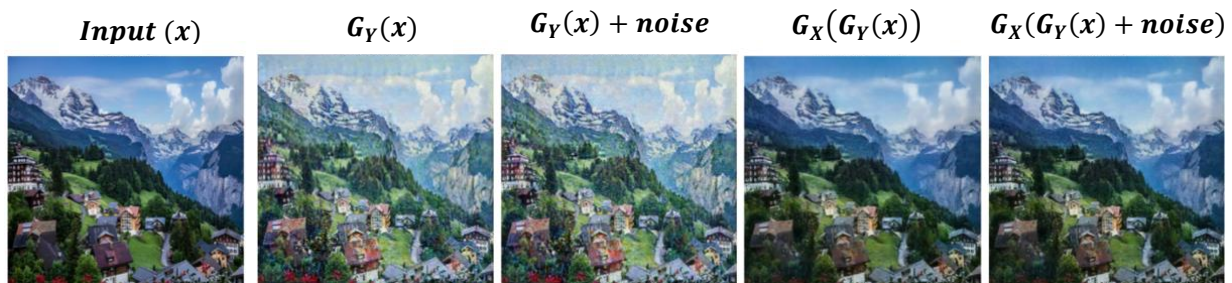


Fig 13: example of reconstruction of CycleGAN that trained to create segmentation. The noise is gaussian noise  $N(0, 0.08)$ .

\*דרך להתמודד עם הבעיה הזו היא להכניס רעש ל-cycle consistency loss במתואר במשוואה (13), כדי להפוך את תהליך השיחזור להיות חסין לרעש ולמנוע מה-generator "להחביא" את התוכנות של ה-input תוך רעש.

$$(13) \mathcal{L}_{cyc}(G_X, G_Y) = \mathbb{E}_{x \sim p_X(x)} [\|G_X(G_Y(x) + noise) - x\|_1] + \mathbb{E}_{y \sim p_Y(y)} [\|G_Y(G_X(y) + noise) - y\|_1]$$

\*אך במקרה זה כפי שראינו מקודם, תהליך השיחזור יחסית חסין לרעש.

## תוצאות:

לצורך מציאת המודל שנותן את ה-MiFID score הנמוך ביותר (הביצועים הטובים ביותר) אימנו את המודלים הבאים:

1. CycleGAN+Os (Over-sampling)
2. CycleGAN+Us (Under-sampling)
3. CycleGAN+Os+Aug (Augmentation)
4. CycleGAN+Us+Aug
5. CycleGAN+Os+DiffAug (Diff-Augmentation)
6. CycleGAN+Us+DiffAug

\*נזכיר שאת מדד MiFID המחושב ב-Kaggle מחשבים עם ציורי Monet שלא נתונים לנו ולכן, לא ניתן להעריך במהלך האימון את ה-score האמיתי שאנו צריכים לקבל.

\*המודל שקיבל את ה-MiFID score הנמוך ביותר (הטוב ביותר) הוא CycleGAN+Os ולכן נציג עבורו את כל הגרפים של האימון ובסוף מופיעה טבלה שמכילה את ה-MiFID score שקיבל כל מודל.

\*אנו מציגים את ה-learning curves רק עבור המודל שנתן את הביצועים הטובים ביותר כדי לקבל דוח יותר קריא ומסודר שמכיל את הפרטים החשובים.

## :Learning curves

לפי משוואה (12) אנו רוצים במהלך האימון להביא למקסימום את ה-Loss לפי ה-discriminator ולמינימום את ה-Loss לפי ה-generator. היינו מעדיפים שנצטרך להביא למינימום את כל ה-Loss-ים על מנת שנוכל להשתמש ב-gradient descent בכל המקרים.

לצורך כך, כאשר אנו מאמנים את ה-discriminator נפתור את בעיית המינימיזציה הבאה:

$$(14) \min_{D_Y} \left\{ - \left( \mathbb{E}_{y \sim p_Y(y)} [\log(D_Y(y))] \right) + \mathbb{E}_{x \sim p_X(x)} [\log(1 - D_Y(G_Y(x)))] \right\}$$

Discriminator loss  
(part of the adversarial loss for the discriminator training)

כלומר, מוסיפים מינוס. וה-generator loss, שזה החלק מה-adversarial loss שמביאים למינימום באימון ה-generator, מוגדר ע"י:

$$(15) \operatorname{argmin}_{G_Y} \left\{ \mathbb{E}_{x \sim p_X(x)} [\log(1 - D_Y(G_Y(x)))] \right\}$$

generator loss  
(part of the adversarial loss for the generator training)

### הערות/מסקנות:

ב-Fig 15 ניתן לראות את ה-lossים שהתקבלו עבור המודל CycleGAN+Os.

- בגרף (e) ניתן לראות את ה-cycle loss שהולך וקטן עם האימון, כלומר ה-generator ימים מצליחים ללמוד בצורה יותר טובה תכונות מה-input שמאפשרות שיחזור.
- בגרף (a) ניתן לראות את ה-generator loss עבור תמונות photo (תמונות שנוצרו ע"י מצלמה) וב-(b) את ה-photo discriminator loss. נשים לב, שבהתחלה שניהם יורדים ואז ה-discriminator loss מתחיל קצת לעלות (כלומר, ה-generator loss "מצליח לעבוד" על ה-discriminator). מסקנות דומות ניתן להסיק גם עבור ה-lossים של Monet.

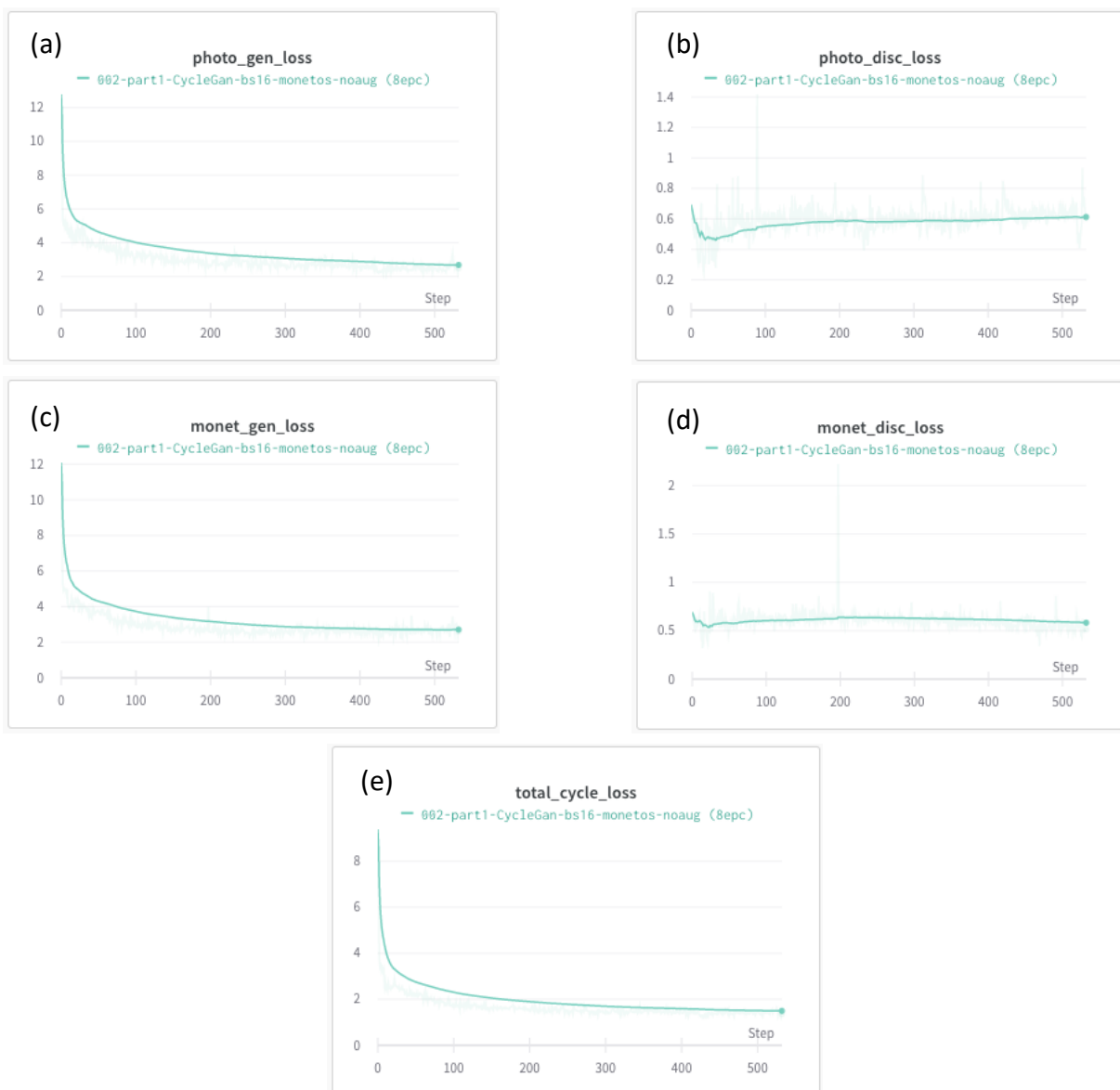


Fig 14: (a) photo (camera image) generator loss (b) photo discriminator loss (c) Monet generator loss (d) Monet discriminator loss (e) Cycle Consistency Loss (photo +Monet).



## MiFID Score:

הטבלה שמופיעה מטה מציגה את ה-MiFID Score שהתקבל עבור כל אחד מהמודלים שאומן.

Model name	CycleGAN +Os	CycleGAN +Us	CycleGAN +Os+Aug	CycleGAN +Us+Aug	CycleGAN +Os+DiffAug	CycleGAN +Us+DiffAug
MiFID Score	38.553	38.674	47.369	48.475	54.991	59.493

## הערות/מסקנות:

- המודל שקיבל את ה-MiFID הטוב ביותר (הנמוך ביותר) הוא CycleGAN+Os.
- ניתן לראות שהאוגמנטציות (בשני הסוגים) לא עזרו לשפר את הביצועים. ככל הנראה הסיבה לכך היא שאוגמנטציות משנות את ההתפלגות ממנה הציורי Monet הגיעו וכתוצאה מכך הביצועים של ה-GAN נפגעים.
- ניתן לראות שבאופן כללי עם שימוש ב-Under sampling מקבלים ביצועים פחות טובים מאשר Over-sampling. אומנם ב-Over-sampling יש סכנה שמודל ישנן את ה-300 ציורי Monet אך היתרון הוא שהמודל מתאמן על יותר תמונות שצולמו ע"י מצלמה ויתרון זה מאפשר ככל הנראה למודל לת ביצועים טובים.

מקום 17 בתחרות הכוללת נכון לתאריך 17/07/2022

17

Vincent GAN Gogh



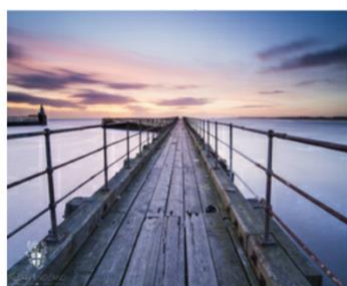
38.55316

## דוגמאות ליצירת ציורי Monet עם המודל CycleGAN+Os:

Photo (camera image)



Generated Monet image  $G_Y(x)$



**Reference:**

- [1] Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." Proceedings of the IEEE international conference on computer vision. 2017
- [2] Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017
- [3] Zhao, Shengyu, et al. "Differentiable augmentation for data-efficient gan training." Advances in Neural Information Processing Systems 33 (2020)
- [4] Bashkirova, Dina, Ben Usman, and Kate Saenko. "Adversarial self-defense for cycle-consistent GANs." Advances in Neural Information Processing Systems 32 (2019)