ציור מתמונה

29/4/2022

חלופה: למידת מכונה

בית ספר: עירוני ד

עומר חנן 326199080
שם מנחה: שי פרח

תוכן

מבוא

מטרת הפרויקט הזה הוא בניית מודל מייצר בשיטת GAN והפיכת תמונה לציור בסגנון של הצייר מונה. המודל מאפשר להפוך כל תמונה לציור שיכול לשמש כרקע למחשב/טלפון או סתם להתנסות ולראות את היכולות המדהימות של מודלים כאלו.

מהו GAN(Generative Adversarial Network)?

GAN הוא שילוב של שני מודלים או יותר המתחרים אחד בשני על מנת להשיג את המטרה, מודל אחד מייצר(Generator), מטרתו לייצר או לשנות נתונים בהתאם למטרה שלנו, ומודל אחר מבחין (Discriminator) ש"אומר" למודל המייצר אם הצליח או לא ובכך מאמן אותו, המבחין מקבל נתונים אחרים ובכך מתאפשר לא להתאמן(למשל לדעת מה אמיתי ומה לא) ולאמן את המייצר בהתאם למה שלמד, גם הוא באמצעות פונקציית עלות רגילה. שני המודלים מתחילים מאפס, המשמעות היא שהם כלל לא מאומנים. מה יקרה אם ניקח מבחין מאומן ונשתמש בו? המייצר פשוט לא ילמד ולא יגיע לתוצאות, אם כל פעם שהוא מנסה המבחין אומר לו שהוא לא הצליח אז המייצר לא מתקדם. אם שניהם מתחילים לא מאומנים אז המייצר כן ידע לאן להתקדם ולאט לאט הוא יגיע לתוצאה.

והם מתאמנים ביחד בכך שנלחמים אחד בשני, המייצר מנסה "לעבוד" על המבחין והמבחין מנסה להבחין בין מה שהמייצר ייצר לבין הנתונים שמקבל ובכך מאמן את המייצר.

תהליך המחקר

אחרי שהחלטתי לבחור ללמוד את נושא הGAN בהתמחות הייתי צריך לבחור רעיון לפרויקט, אז התחלתי לחפש כל מיני רעיון, נתקלתי בתחרות באתר Kaggle של הפיכת תמונה לציור, הרעיון היה נראה לי מצוין לפרויקט ובנוסף גם מגניב ומעניין.

אחר כך התחיל שלב המחקר, הייתי צריך לבחור במודל המתאים ביותר למשימה, במהלך ההתמחות התנסיתי בבנייה ואימון של מודלים שונים והבנתי של איזה מודל יכול להתאים. התלבטתי בין כמה מודלים, אך במהרה היה לי ברור איזה מודל הכי מתאים, cycleGAN. בהתמחות המודל הצליח להפוך סוס בתמונה לזברה, והיה נראה לי שזה המודל המתאים ביותר, הוא כמו מודלים אחרים שמצליחים לייצר בהתאם למטרה אך היה לו את התכונה החשובה היותר, שמירה על הפרטים והצבעים בתמונה עם פונקציות עלות מיוחדות ומותאמות במיוחד בשביל זה. בהפיכה של תמונה לציור נרצה לשמור על הפרטים והצבעים של התמונה בנוסף ל להפוך אותה לציור. ביצעתי מעט שינויים בשכבות המודל, אך הארכיטקטורה דומה מאוד. (בהמשך יהיה תיאור גרפי והסבר מעמיק של המודל)

אתגרים מרכזיים

במהלך העבודה על הפרויקט, השתמשתי בספריה Pytorch, ספריה חדשה על מנת ליצור מודל, אומנם בהתמחות למדתי את הספרייה ובניתי מודלים אך לא פיתחתי פרויקט בעצמי. זהו אתגר מאוד משמעותי לעבוד עם ספרים חדשה ושונה, עד כה למדו להשתמש ב Keras ספריה יחסית פשוטה שעושה הרבה דברים אוטומטית, אך בשביל מודל כזה צריך להשתמש בPytorch. בנוסף טעינת הנתונים והכנתם לאימון של Pytorch היה מאתגר יחסית, היה די קשה להבין את ההיגיון מאחורי הקוד למרות שהוא לא צריך להיות מסובך, ורוב הדוגמאות באינטרנט הן מאוד מותאמות למודלים בסיסיים מאוד כמו סיווג. על מנת להתאים למשימה שלי הייתי צריך להבין את ההיגיון מאחורי הקוד, משימה שהייתה לא פשוטה כלל.

מבנה / ארכיטקטורה של הפרויקט

שלב איסוף הכנה וניתוח הנתונים

הנתונים נלקחו מתחרות באתר Kaggle.

קישור לאתר Kaggle ממנו נלקחו הנתונים:

https://www.kaggle.com/c/gan-getting-started/data

הdataset מכיל 300 ציורים של הצייר מונה ("monet") ו 7028 תמונות שאנשים צילמו(בעיקר תמונות של טבע ומבנים). שניהם ברזולוציה 256x256 צבעוני(RGB) ובפורמט JPEG.

מאחר והשתמשתי בספריית pytorch על מנת ליצור מודל ולאמן אותו, העברתי את הנתונים לטנסור(Tensor) של pytorch ואז השתמשתי בספרייה לייצור dataset לאימון של pytorch.

ביצירת הdataset, לאימון הגדרתי 295 תמונות וציורים באופן אקראי, ואת שאר התמונות למבחן (משתמשים רק בתמונת)

נרמול – לא נעשה נרמול.

אחרי העברת ערכי הפיקסלים מ0 עד 255 לערכים בין 0 ו 1, העברתי את הערכים לטווח ערכים בין 1- ל 1 טווח זה רחב יותר ומאפשר למודל ללמוד יותר בקלות ובמהירות.

ציור של מונה מה dataset תמונה מה dataset

שלב בנייה ואימון המודל

המודל בפרויקט הוא מודל מייצר, על מנת להשיג מטרה זו לא ניתן להשתמש במודלים שלמדנו עד כה ולכן למדתי על מודלים שמטרתם היא לייצר(GANs)

בפרויקט יישמתי ואימנתי מודל cycleGAN, מודל מורכב המכיל 4 מודלים שונים עם מטרות שונות, שני מודלים מייצרים(Generators) ושני מודלים מבחינים(Discriminators). האימון בוצע על המחשב שלי על ה GPU שלי, והוא נמשך בערך 10 שעות.

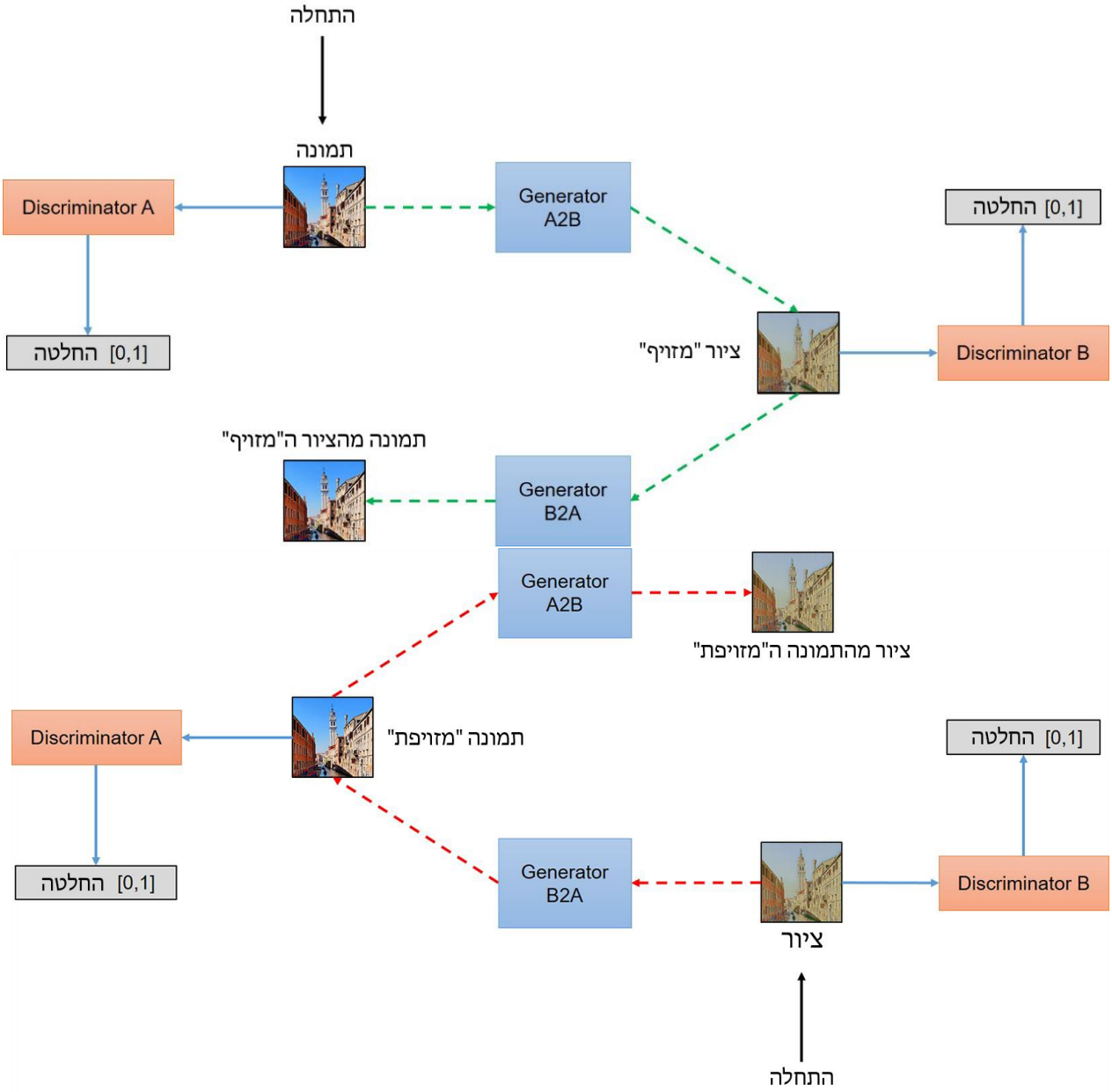
Generator AB – זהו המודל בו נשתמש בסוף והוא המטרה העיקרית, הוא מקבל תמונה ומטרתו להפוך אותה לציור בסגנון של הצייר מונה.

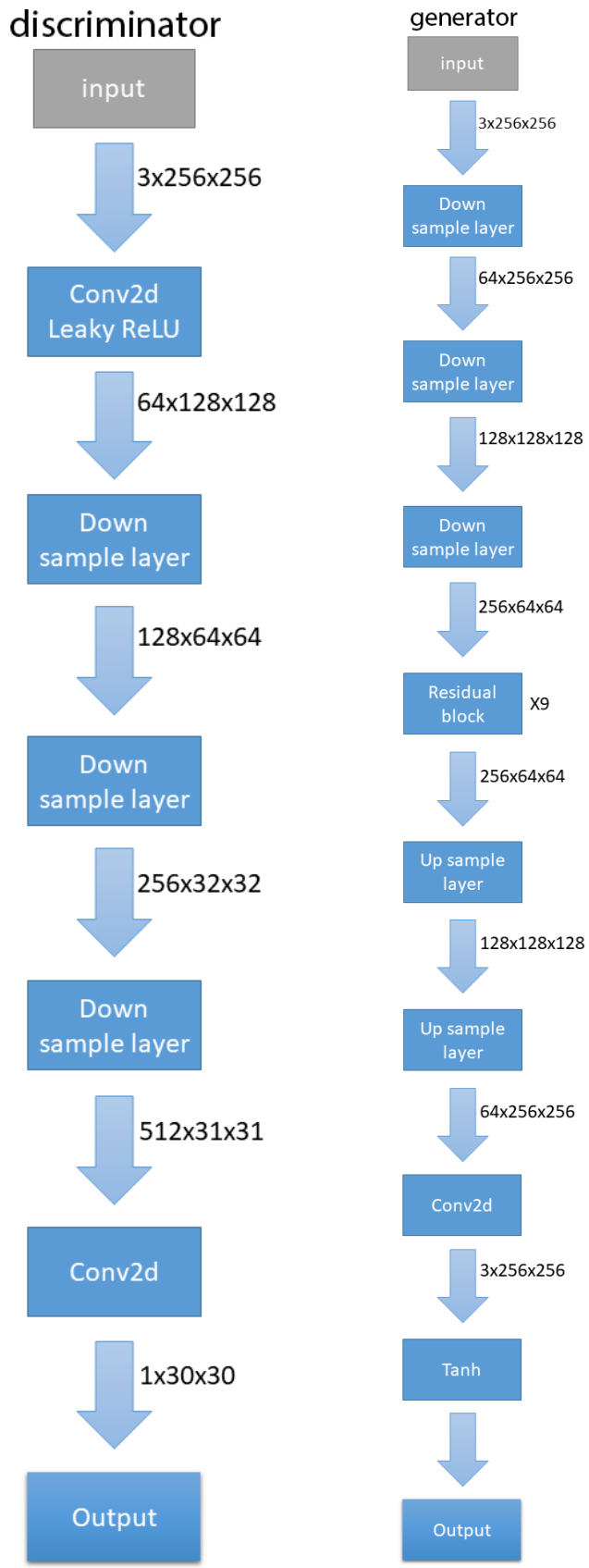
Generator BA – מודל שמטרתו להפוך ציור לתמונה.

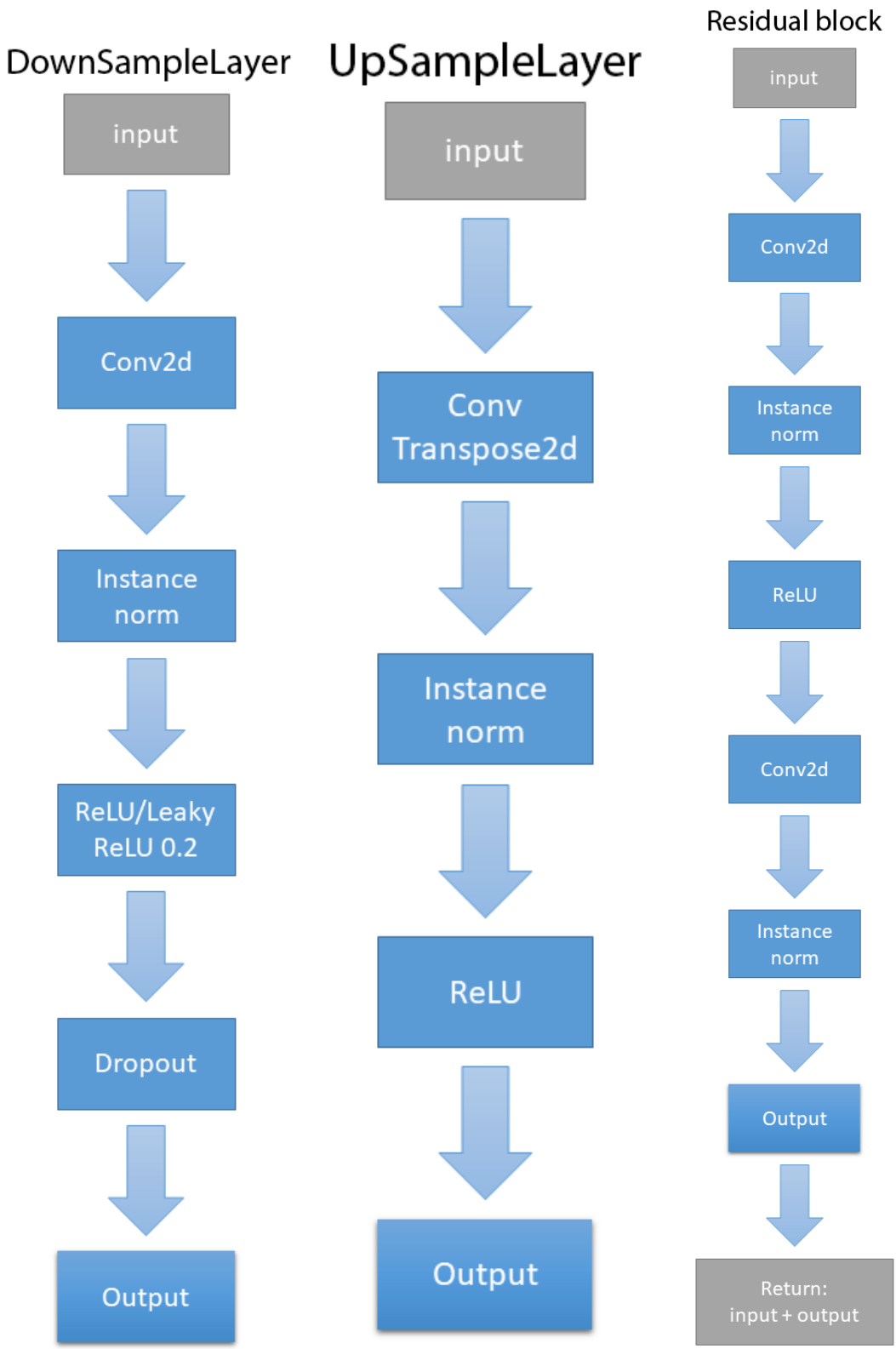
Discriminator A – זהו מודל שמטרתו להבחין אם הקלט שקיבל הוא תמונה אמיתית או לא, הוא מקבל תמונה אמיתי ותמונה מזויפת מGenerator BA. מטרתו לאמן את Generator BA, הוא "אומר" לו אם הצליח להפוך ציור לתמונה.

Discriminator B – זהו מודל שמטרתו להבחין אם הקלט שקיבל הוא ציור אמיתי או לא, הוא מקבל ציור אמיתי וציור מזויף מGenerator AB. מטרתו לאמן את Generator AB, הוא נותן ביקורת למייצר, אם הצליח להפוך תמונה לציור (יותר נכון כמה הצליח).

דיאגרמות של המודל




discriminator
input
3x256x256
Conv2d
Leaky ReLU
64x128x128
Down
sample layer
128x64x64
Down
sample layer
256x32x32
Down
sample layer
512x31x31
Conv2d
1x30x30
Output
generator
input
3x256x256
Down
sample layer
64x256x256
Down
sample layer
128x128x128
Down
sample layer
256x64x64
Residual
block
X9
256x64x64
Up sample
layer
128x128x128
Up sample
layer
64x256x256
Conv2d
3x256x256
Tanh
Output


DownSampleLayer
input
Conv2d
Instance norm
ReLU/Leaky ReLU 0.2
Dropout
Output
UpSampleLayer
input
Conv Transpose2d
Instance norm
ReLU
Output
Residual block
input
Conv2d
Instance norm
ReLU
Conv2d
Instance norm
Output
Return: input + output

הסבר על שכבות המודל

Residual block(Residual networks)

בעבר, חוקרים מצאו שככל שהרשת עמוקה יותר כך היא משיגה תוצאות טובות יותר. ניסו לבנות רשתות עם מאוד שכבות, אך למרבה הצער התוצאות שהשיגו היו פחות טובות מרשתות בעלות מספר נמוך יחסית של שכבות. הבעיה המרכזית נבעה מכך שלאחר מספר שכבות מסוים התקבל ייצוג מספיק טוב, וכל שכבה שהוספה שינתה את התוצאה ואף הרסה אותה, בנוסף, היה לרשת קושי ללמוד בשכבות עמוקות. הפתרון היה פשוט, לשמור את הקלט ולהחזיר אותו ביחד עם אותו הקלט אחרי עיבוד. ובכך התוצאה לא נהרסת, השכבה לא יכולה להרוס אלא רק להוסיף, אם אין צורך בשינוי בקלט השכבה אז לא יהיה.

Down sample layer

כיווץ ודחיסת המידע של התמונה, הגדלת הערוצים(channels) והקטנת גודל התמונה, מאפשר עיבוד יעיל וטוב יותר בהמשך.

Up sample layer

עיבוד המידע והגדלתו לגודל התמונה המקורית(ההפך מDown sample) אחרי עיבוד של שכבות ה residual. מתבצע באמצעות שימוש בקונבולוציה הפוכה.

Leaky ReLU

Leaky ReLU הוא כמו ReLU רק שהשיפוע בחלק השלילי אינו 0(וגם לא 1 כי אז זה יהיה פונקציה ליניארית), ניתן לקבוע אותו(השתמשתי ב0.2).

למה משתמשים בו – ב ReLU יש בעיה, והיא שברגע שמגיעים לחלק השלילי השיפוע הוא 0(וכך גם הנגזרת) מה שגורם לנוירונים מסוימים להפסיק ללמוד. Leaky ReLU פותר את הבעיה הזאת בקלות, הוא פשוט מאפשר לקבוע את השיפוע בחלק השלילי מה שמאפשר לנוירונים להמשיך ללמוד.

Instance normalization

מנרמל את כל הערכים בדוגמא, מונע מערכים חריגים להשפיע בכך שמנרמל אותם.

Dropout

מוריד/מתעלם מערכים מסוימים בצורה אקראית, מקבל hyperparameter שקובע את הסיכוי של כל ערך "ליפול". Dropout הוא פתרון פשוט לoverfitting. אך במודל הזה הdropout היה על 0, משמע לא ביצע כלום(אך היה שם למקרה והייתי רוצה להשתמש בו).

כמות פרמטרים וזיכרון:

מייצר:

11,378,179 פרמטרים סה"כ וניתנים לאימון

כמות זיכרון של פרמטרים: 43.5 MB

זיכרון נדרש בעת אימון על תמונה 3x256x256: 907.00 MB

כמות זיכרון נדרשת למייצר אחד בזמן אימון: 950.5 MB

מבחין:

2,764,737 פרמטרים סה"כ וניתנים לאימון

כמות זיכרון של פרמטרים: 10.55 MB

זיכרון נדרש בעת אימון על תמונה 3x256x256: 64.78 MB

כמות זיכרון למבחין אחד בזמן אימון: 75.33 MB

סה"כ זיכרון נדרש בעת אימון

גודל תמונה: 0.75 MB (תמונה 3x256x256)

2 מייצרים: 1901 MB

2 מבחינים: 150.66 MB

295 תמונה ו295 ציורים: 442.5 MB

סה"כ זיכרון נדרש לאימון: 2,494.16 MB (~2.5 GB)

Hyper Parameters

Learning rate = 0.0002

betas=(0.5, 0.999) של המייעלים

batch size = 1

פונקציות העלות

פונקציית העלות של המבחינים (Discriminators):

השתמשתי בפונקציית עלות בשם MSE(mean squared error), מודד את הממוצע של השגיאה בריבוע, ההפרש בין מה שהמודל ניחש לבין מה שהיה צריך לנחש (0 לציור מזויף של מונה ו1 לציור אמיתי של מונה, וכך גם לתמונה 0 לתמונה מזויפת ו1 תמונה אמיתית).

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

n = מספר הדוגמאות

i = מספר הדוגמא

Ŷ = הערך שהמודל חזה

Y = הערך הנכון (הערך שמודל היה צריך לנחש)

פונקציית העלות של המייצרים (Generators):

פונקציית העלות של המייצרים מורכבת ב3 פונקציות עלות שונות, מחברים את ערכי השגיאה יחד כל אחת עם למבדה אחרת על מנת לתת משקל שונה ומותאם לכל אחת. הסכום הוא ערך השגיאה של המייצרים.

Generator adversarial loss

הציון שקיבלו המייצרים מהמבחינים, האם הצליחו לעבוד עליהם (יותר נכון כמה הצליחו לעבוד עליהם), שימוש בפונקציית העלות MSE, תמונה עוברת במייצר תמונה לציור ומגיעה למבחין, העלות היא לפי הנוסחה למעלה רק ש:

Ŷ = הציון שהמבחין נתן למייצר, כמה קרוב היה לציור אמיתי של מונה.
Y = 1, המטרה של המייצר הוא לקבל ציון 1, משמע הצליח לעבוד על המבחין ומתקרב לייצור ציור של מונה.
אותו הדבר גם למייצר השני רק ש:
Ŷ = הציון שהמבחין נתן למייצר, כמה קרוב היה לתמונה אמיתית.
Y = 1, המטרה של המייצר הוא לקבל ציון 1, משמע הצליח לעבוד על המבחין ולהפוך ציור לתמונה אמיתית.

identity loss

הציון שקיבלו המייצרים כשהעבירו בהם את סוג התמונה שהם צריכים לייצר, מייצר אחד(ציור לתמונה) מקבל תמונה ומנסה להפוך לתמונה, ומייצר שני(תמונה לציור) מקבל ציור ומנסה להפוך לציור.

המטרה של פונקציית העלות הזו היא לעזור למייצרים לשמור על הצבעים בתמונה ולא לתת להם "חופש" ולאפשר להם לעבוד על המבחינים בכל מחיר(אם הם רואים למשל שצבע מסוים מצליח לעבוד על המבחין יותר מצבע אחר).

שימוש בפונקציית העלות L1Loss העלות של הפונקציה הזאת היא המרחק בין ערך הפיקסלים של התמונה המקורית, והתמונה אחרי שהעבירו אותה במייצר הנגדי לה.

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

cycle consistency loss

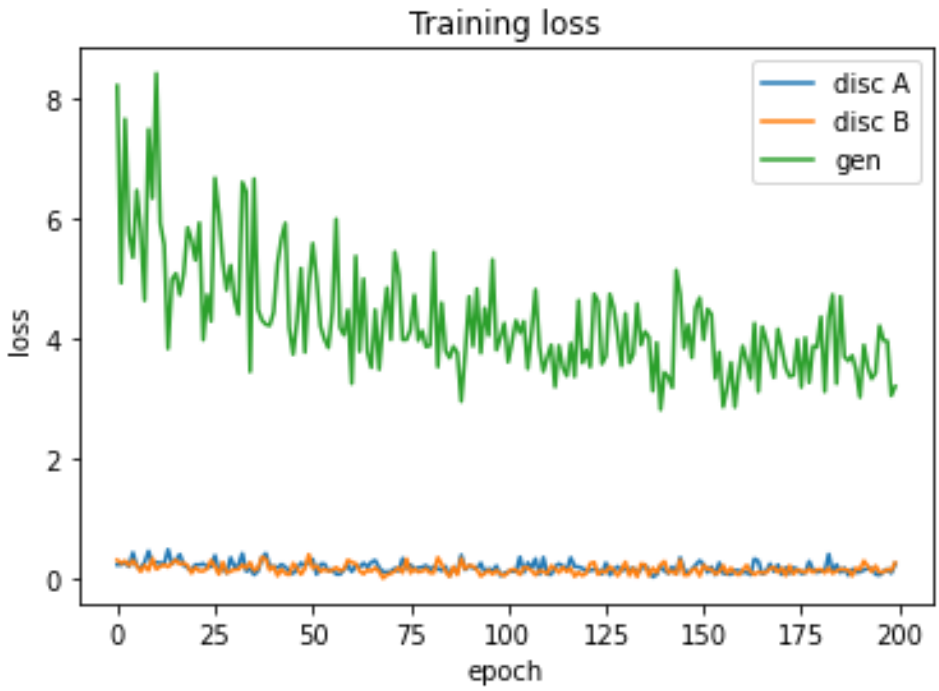
הציון שקיבלו המייצרים אחרי "סיבוב מלא" במודל, תמונה עוברת במייצר תמונה לציור ואז במייצר ציור לתמונה, ואותו הדבר גם עם ציור, העברת ציור במייצר ציור לתמונה ואז במייצר תמונה לציור.

מטרת הפונקציה הזו היא לגרום למייצרים לשמור על הפרטים בתמונה, בכך שיהיה ניתן להחזיר לתמונה למקור. וגם פה לא לתת "חופש" למייצרים ולאפשר להם לעבוד על המבחינים בכל מחיר. אם הצליח להחזיר לתמונה המקורית בדיוק אז הפרטים בתמונה לא נעלמו בדרך ובכך הפרטים נשמרים בציור

החישוב נעשה כמו בפונקציית עלות מס' 2 עם L1Loss(השם של פונקציות העלות שונה בקוד, אבל בפועל כשממשים אותם, הן אותה פונקציית עלות).

תוצאות אימון

גרף פונקציות העלות



הסיבה להבדל בפונקציית העלות בין המייצרים לבין המבחינים היא בגלל שפונקציות העלות שונות וכך גם ערכי השגיאה שלהן. הסיבה לכל העליות והירידות היא שהם תמיד "נלחמים" אחד בשני, הם מאמנים אחד את השני ומתאמנים במקביל, הם תמיד משתפרים.

בנוסף פונקציית העלות של המייצרים תלויה במבחינים(פונקציית העלות היא כמה הצליח "לעבוד" על המבחין), אם גורם למבחין להגיע לדיוק של 50% אז זה אומר שהוא לא יכול להבדיל בין אמיתי למזויף, הוא כאילו מנחש.

בעיה עם ערכי הפלט

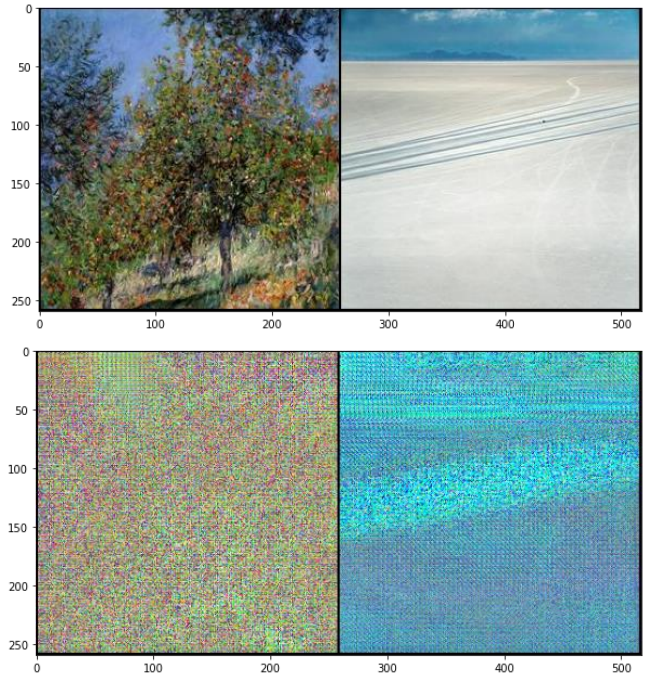
ערכי הפלט של המודל הם בין 1- ל 1, כשרוצים להציג את תמונה נוצרת בעיה, הפונקציות של הצגת התמונה יודעות להתמודד עם ערכים של תמונה רגילה(בין 0 ל255) או בין 0 ל1, כשמנסים להציג תמונה שהערכים שלה הם בין 1- ל1 היא פשוט "מדביקה" את הערכים הקטנים מ0 ל0, מה שיוצר עיוות נוראי בתמונה. הפתרון הוא להוסיף 1 לכל ערכי הפיקסלים ולחלק ב2, זה מעביר את כל הערכים לטווח שהפונקציות יכולות להתמודד(בין 0 ל1) ואז ניתן לראות את התמונה כמו שצריך.

דוגמא:

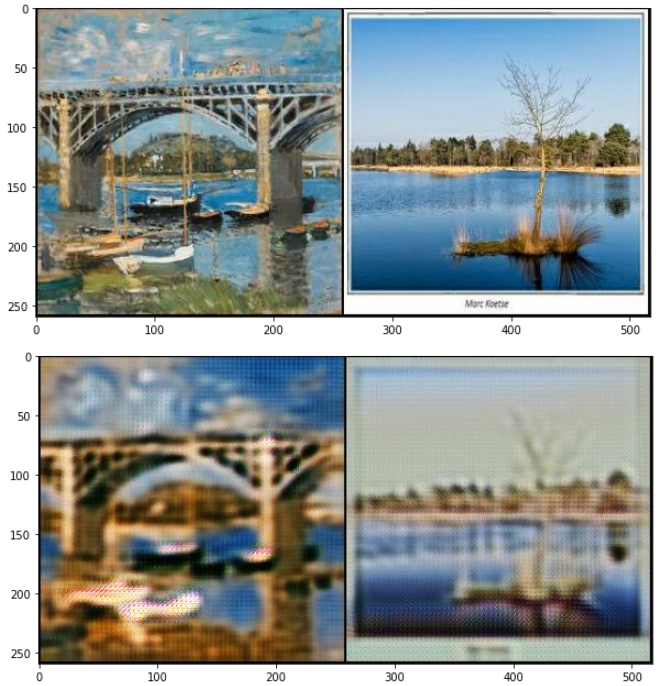
תמונות בזמן האימון

מימין – תמונה לציור, משמאל – ציור לתמונה

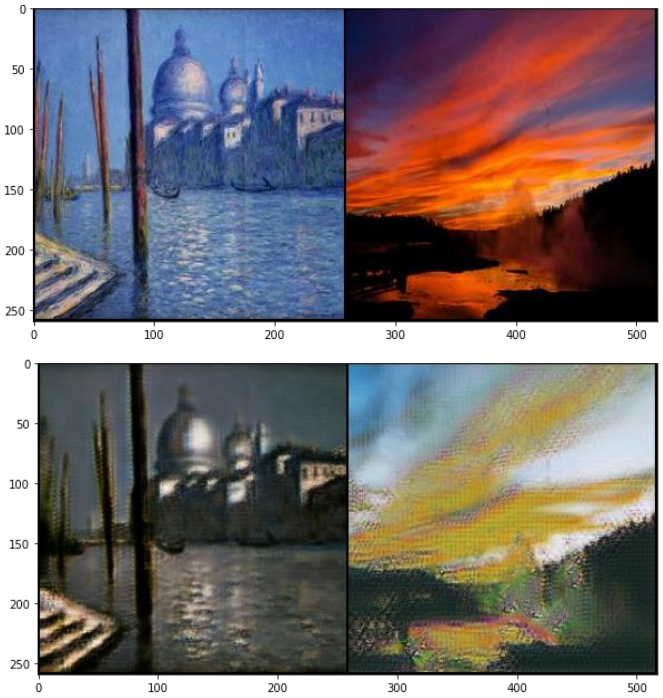
אפוק 0 - הרבה רעש.



אפוק 2 - כבר באפוק השני ניתן לראות שהמודל מתחיל לשמור על הפרטים בתמונה בזכות הCycle loss



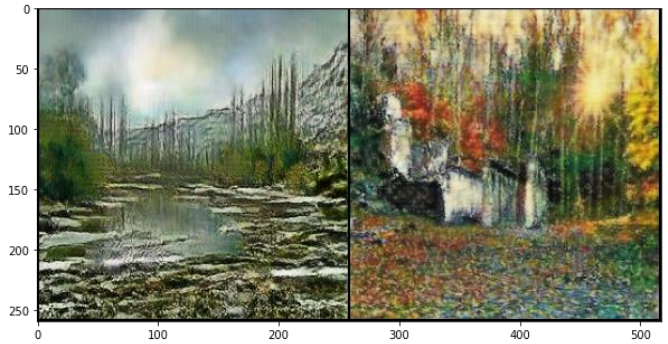
אפוק 40 - ניתן לראות שהצבעים אינם נשמרים היטב, אך זה ישתפר בזכות ה identity loss.



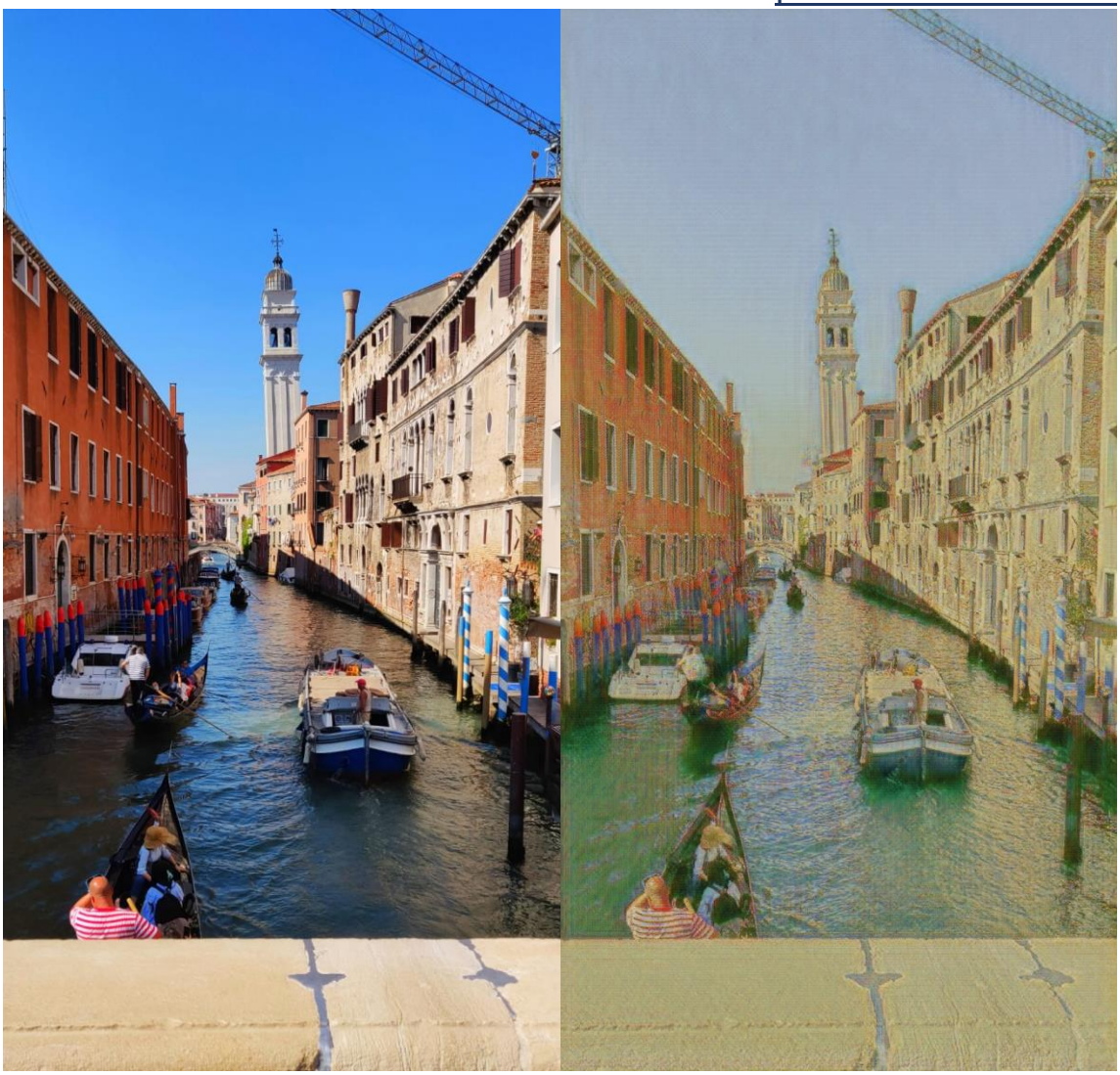
אפוק 101 - הצבעים נשמרים יותר טוב, והמודל הופך את התמונה לציור יותר טוב.

אפוק 147 - המודל מגיע לתוצאות טובות.

אפוק 195 - המודל כבר מגיע לתוצאות מעולות.



דוגמא לתמונה אחרי האימון



מדריך למפתח

קבצים

cycleGAN-_-.pth – קובץ ששומר את כל המשקלים של המודלים

שמירה של היסטורית ערכי השגיאה (קבצי מערכים של numpy):

disc_A_loss_history.npy

disc_B_loss_history.npy

gen_loss_history.npy

Project.ipynb – מחברת הפרויקט בה נעשה כל הפיתוח, קישור למחברת colab

עוד קישור למחברת קלה יותר בלי הפלטים לטעינה מהירה יותר.

תיעוד הקוד

ספריות

```
import torch #הספריה המרכזית ליצירת המודל
from torch import nn #קיצור ליצירת שכבות
from tqdm.auto import tqdm #נתונים בזמן ריצת לולאה
from torchvision import transforms #עיבוד הנתונים
from torchvision.utils import make_grid #מאפשר פלט מסודר של תמונות
from torch.utils.data import DataLoader, Dataset #עם אלו נטען את הנתונים ו
נכין אותם לאימון
import matplotlib.pyplot as plt #ספריה המאפשרת יצירת גרפים ולהראות תמונות
import os #משמש לעבודה עם תיקיות
from PIL import Image #ספריה המאפשרת לעבוד עם תמונות
import numpy as np #ספריה לעבודה עם מערכים
```

פונקציה המציגה תמונות

הפונקציה הזאת נלקחה מאחד התרגילים בהתמחות ה GAN והיא מציגה את התמונות בתוך באץ' (batch) של תמונות, אחד ליד השני. השימוש בה נעשה בזמן אימון.

```
def show_tensor_images(image_tensor, num_images=25, size=(3, 256, 256)):
    '''
    Function for visualizing images: Given a tensor of images, number of i
mages, and
    size per image, plots and prints the images in an uniform grid.
    '''
    image_tensor = (image_tensor + 1) / 2
# מעביר את ערכי בפיקסלים מ -1 עד 1 ל 0 עד 1
    image_shifted = image_tensor
    image_unflat = image_shifted.detach().cpu().view(-1, *size)
    image_grid = make_grid(image_unflat[:num_images], nrow=5)
    plt.imshow(image_grid.permute(1, 2, 0).squeeze())
    plt.show()
```

קוד מודל

בלוק ריזידואל

שומר על ערך הקלט ומוסיף אותו לערך הפלט. השימוש בשכבה מהסוג הזה מאפשר לשים הרבה שכבות מבלי חשש שישבשו את תהליך הלמידה, ובכך מאפשר למודל יותר כוח.
קונבולוציה, נרמול ערכים, אקטיבציה ReLU, קונבולוציה, נרמול ערכים.

```
class ResidualBlock(nn.Module):

    def __init__(self, input_channels): #מקבל את כמות הצ'אנלים של הקלט
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(input_channels, input_channels, kernel_size
=3, padding=1, padding_mode='reflect') #שכבת קונבולוציה
        self.conv2 = nn.Conv2d(input_channels, input_channels, kernel_size
=3, padding=1, padding_mode='reflect') #שכבת קונבולוציה
        self.instancenorm = nn.InstanceNorm2d(input_channels) #שכבת נרמול
        self.activation = nn.ReLU() #אקטיבציה

    def forward(self, x): #הפונקציה שקוראים לה ברגע שמשתמשים בשכבה

        original_x = x.clone() #שכפול ושמירת הקלט
        x = self.conv1(x)
        x = self.instancenorm(x)
        x = self.activation(x)
        x = self.conv2(x)
        x = self.instancenorm(x)
        return original_x + x #מחזיר את הסכום של הקלט והפלט
```

Down sample and Up sample layers

DownsampleLayer:

שכבה המקטינה את "גודל" התמונה ומגדילה את כמות הצ'אנלים (channels), עיבוד של הנתונים.

שכבת קונבולוציה אחת. מאפשרת בחירה של פונקציית האקטיבציה, בין ReLU ל Leaky ReLU.

שכבת נרמול - מנרמל את הערכי בכך שלא יהיה ערכים יוצאי דופן שיוצרים שונות גדולה, ובכך מתאפשר למידה טובה יותר.

שכבת dropout מוריד ערכים באקראיות ובכך מוריד את הסיכוי להתאמת יתר(overfitting)

```
class DownsampleLayer(nn.Module):
    def __init__(self, in_channel, out_channel, kernel_size, stride, paddi
ng, padding_mode, dropout, activation_type="relu"):
        super(DownsampleLayer, self).__init__()
        self.activation = None
        self.instancenorm = nn.InstanceNorm2d(out_channel) #נרמול ערכים
        self.dropout = nn.Dropout(dropout) #שכבת דרופאוט
        self.conv1 = nn.Conv2d(in_channel, out_channel, kernel_size, strid
e, padding, padding_mode=padding_mode) #שכבת קונבולוציה

        if activation_type == "relu":
            self.activation = nn.ReLU(inplace=True) # אקטיבציהReLU
        elif activation_type == "l_relu":
            self.activation = nn.LeakyReLU(0.2,inplace=True)
# אקטיבציהleaky ReLU

    def forward(self, x):   #הפונקציה שקוראים לה כשמעבירים ערכים בשכבה

        x = self.conv1(x)
        x = self.instancenorm(x)
        x = self.activation(x)
        x = self.dropout(x)
        return x
```

UpsampleLayer:

שכבה המגדילה את "גודל" התמונה ומקטינה את כמות הצ'אנלים (channels), אחרי עיבוד של התמונה "מחזיר" אותה לגודל המתבקש.

מתבצע בעזרה קונבולוציה הפוכה. שימוש בשכבת נרמול. ואז פונקציית אקטיבציה ReLU.

```
class UpsampleLayer(nn.Module):
    def __init__(self, in_channel, out_channel, kernel_size, stride, paddi
ng, out_padding, padding_mode):
        super(UpsampleLayer, self).__init__()

        self.main = nn.Sequential(
            nn.ConvTranspose2d(in_channel, out_channel, kernel_size, strid
e, padding, #קונבולוציה הפוכה
                                output_padding=out_padding , padding_mode=
padding_mode),
            nn.InstanceNorm2d(out_channel), #נרמול ערכים
            nn.ReLU() # אקטיבציהReLU
        )

    def forward(self, x): #הפונקציה שקוראים לה כשמעבירים ערכים בשכבה

        x = self.main(x)

        return x
```

מייצר

```
class Generator(nn.Module):

    def __init__(self):
        super(Generator, self).__init__()

        self.DSL1 = DownsampleLayer(3, 64, 7, 1, 'same', "reflect", 0)
        self.DSL2 = DownsampleLayer(64,128, 3, 2, 1, "reflect",0)
        self.DSL3 = DownsampleLayer(128,256,3,2, 1, "reflect", 0)
```

```
        # 9 שכבות ריזידואל
        # 256 צ'אנלים
        self.res1 = ResidualBlock(256)
        self.res2 = ResidualBlock(256)
        self.res3 = ResidualBlock(256)
        self.res4 = ResidualBlock(256)
        self.res5 = ResidualBlock(256)
        self.res6 = ResidualBlock(256)
        self.res7 = ResidualBlock(256)
        self.res8 = ResidualBlock(256)
        self.res9 = ResidualBlock(256)

        self.USL1 = UpsampleLayer(256,128,3,2,1,1, "zeros")
        self.USL2 = UpsampleLayer(128,64,3,2,1,1, "zeros")
        self.conv1 = nn.Conv2d(64, 3, 7, 1, padding='same', padding_mode="
        reflect") #שכבת קונבולוציה
        self.activation = nn.Tanh() # אקטיבציה טנגנס היפרבולי( בין 1- ל1)

    def forward(self, x): #הפונקציה שקוראים לה כשמעבירים ערכים במודל

        x1 = self.DSL1(x)
        x2 = self.DSL2(x1)
        x3 = self.DSL3(x2)

        x4 = self.res1(x3)
        x5 = self.res2(x4)
        x6 = self.res3(x5)
        x7 = self.res4(x6)
        x8 = self.res5(x7)
        x9 = self.res6(x8)
        x10 = self.res7(x9)
        x11 = self.res8(x10)
        x12 = self.res9(x11)

        x13 = self.USL1(x12)
        x14 = self.USL2(x13)
        x15 = self.conv1(x14)
        xn = self.activation(x15)

        return xn
```

מבחין

```
class Discriminator(nn.Module):

    def __init__(self):
        super(Discriminator, self).__init__()

        self.conv1 = nn.Conv2d(3, 64, 4, 2, 1,padding_mode="reflect")
#שכבת קונובלוציה
        self.activation = nn.LeakyReLU(0.2)
# אקטיבציה leaky ReLU  עם שיפוע 0.2

        self.DSL1 = DownsampleLayer(64, 128, 4, 2, 1, "reflect", 0,activat
ion_type="l_relu")
        self.DSL2 = DownsampleLayer(128, 256, 4, 2, 1, "reflect", 0,activa
tion_type="l_relu")
        self.DSL3 = DownsampleLayer(256, 512, 4, 1, 1, "reflect", 0,activa
tion_type="l_relu")

        self.conv2 = nn.Conv2d(512, 1, 4, padding=1, padding_mode="reflect
") #שכבת קונבולוציה

    def forward(self, x): #הפונקציה שקוראים לה כשמעבירים ערכים במודל
        x1 = self.conv1(x)
        x2 = self.activation(x1)
        x3 = self.DSL1(x2)
        x4 = self.DSL2(x3)
        x5 = self.DSL3(x4)
        xn = self.conv2(x5)

        return xn
```

פונקציית עלות של המבחינים

מקבל: תמונה אמיתית X, תמונה מזויפת X, מודל מבחין X, ופונקציית עלות. מחזיר: ערך השגיאה

```
def get_disc_loss(real_X, fake_X, disc_X, adv_criterion):

    disc_fake_X_hat = disc_X(fake_X.detach())
#מנתק את התמונה המזויפת מהמייצר ומעביר במבחין
    disc_fake_X_loss = adv_criterion(disc_fake_X_hat, torch.zeros_like(dis
c_fake_X_hat))
# משיג את ערך השגיאה לפי ערך המבחין על תמונה מזוייפת והשוואה עם 1
    disc_real_X_hat = disc_X(real_X) #מעביר תמונה אמיתית במבחין
    disc_real_X_loss = adv_criterion(disc_real_X_hat, torch.ones_like(disc
_real_X_hat))
# משיג את ערך השגיאה לפי ערך המבחין על תמונה אמיתית והשוואה עם 0
    disc_loss = (disc_fake_X_loss + disc_real_X_loss) / 2
#מחבר את ערכי השגיאה ומחלק ב2

    return disc_loss
```

פונקציות עלות של המייצרים

פונקציית עלות של המייצר לפי המבחין.

מקבל: תמונה אמיתית X, מבחין Y, מייצר XY, פונקציית עלות. מחזיר: ערך השגיאה, תמונה מזויפת Y(על מנת לחסוך חישוב נוסף בזמן אימון)

```
def get_gen_adversarial_loss(real_X, disc_Y, gen_XY, adv_criterion):

    fake_Y = gen_XY(real_X) #מעביר תמונה אמיתית ומשיג תמונה מזוייפת
    disc_fake_Y_hat = disc_Y(fake_Y) #הציון שקיבל מהמבחין
    adversarial_loss = adv_criterion(disc_fake_Y_hat, torch.ones_like(disc
_fake_Y_hat)) # משיג את ערך השגיאה לפי ערך שקיבל מהמבחין והשוואה ל1

    return adversarial_loss, fake_Y
```

מקבל: תמונה אמיתית X, מייצר YX, פונקציית עלות. מחזיר: ערך השגיאה, ותמונה אמיתית X שעברה במייצר YX

```
def get_identity_loss(real_X, gen_YX, identity_criterion):

    identity_X = gen_YX(real_X)
#מעביר תמונה אמיתית ומחזיר תמונה מאותו הסוג
    identity_loss = identity_criterion(identity_X, real_X)
#מחזיר את ערך השגיאה בין הפיקסלים

    return identity_loss, identity_X
```

מקבל: תמונה X אמיתית, תמונה Y מזויפת, מייצר YX, פונקציית עלות.

משלים את ה"סיבוב", מקבל תמונה מזויפת Y ומנסה להחזיר ל Xואז מוצא את ערך השגיאה שהוא המרחק בין הפיקסלים

```
def get_cycle_consistency_loss(real_X, fake_Y, gen_YX, cycle_criterion):

    cycle_X = gen_YX(fake_Y)
# מקבל תמונה מזוייפת ומנסה להחזיר למקור, "סיבוב"משלים
    cycle_loss = cycle_criterion(cycle_X, real_X)
#מחזיר את ערך השגיאה בין הפיקסלים
    return cycle_loss, cycle_X
```

חיבור פונקציות העלות

מקבל: תמונה A, תמונה B, מייצר AB, מייצר BA, מבחין A, מבחין B, שלוש פונקציות עלות לכל פונקציה, למבדה לכל ערך על מנת לתת משקל מתאים לכל ערך עלות.

מחזיר: סך כל השגיאה של המייצרים, תמונה מזויפת A, תמונה מזויפת B

```
def get_gen_loss(real_A, real_B, gen_AB, gen_BA, disc_A, disc_B, adv_crite
rion, identity_criterion, cycle_criterion, lambda_identity=0.1, lambda_cyc
le=10):

    # Adversarial Loss
    adv_loss_BA, fake_A = get_gen_adversarial_loss(real_B, disc_A, gen_BA,
 adv_criterion) #מחזיר ערך שגיאה של מייצר תמונה, ותמונה מזויפת
    adv_loss_AB, fake_B = get_gen_adversarial_loss(real_A, disc_B, gen_AB,
 adv_criterion) #מחזיר ערך שגיאה של מייצר ציור, וציור מזויף
    gen_adversarial_loss = adv_loss_BA + adv_loss_AB #סוכם את ערכי השגיאה

    # Identity Loss
    identity_loss_A, identity_A = get_identity_loss(real_A, gen_BA, identi
ty_criterion) #מחזיר ערך שגיאה נוסף של מייצר תמונה
    identity_loss_B, identity_B = get_identity_loss(real_B, gen_AB, identi
ty_criterion) #מחזיר ערך שגיאה נוסף של מייצר ציור
    gen_identity_loss = identity_loss_A + identity_loss_B
#סוכם את ערכי השגיאה

    # Cycle consistency Loss
    cycle_loss_BA, cycle_A = get_cycle_consistency_loss(real_A, fake_B, ge
n_BA, cycle_criterion) #מחזיר ערך שגיאה נוסף של מייצר תמונה
    cycle_loss_AB, cycle_B = get_cycle_consistency_loss(real_B, fake_A, ge
n_AB, cycle_criterion) #מחזיר ערך שגיאה נוסף של מייצר ציור
    gen_cycle_loss = cycle_loss_BA + cycle_loss_AB #סוכם את ערכי השגיאה

    # Total loss
    gen_loss = lambda_identity * gen_identity_loss + lambda_cycle * gen_cy
cle_loss + gen_adversarial_loss
#סוכם את כל ערכי השגיאה עם המשקל למבדה שלהם

    return gen_loss, fake_A, fake_B
```

הכנת הנתונים לאימון

בנאי מקבל: תיקייה בה שמור הנתונים, מצב אימון/מבחן, פונקציה לעיבוד הנתונים.

פונקציה לטעינת נתונים של Pytorch

```
class MonetPhotoDataset(Dataset):
    def __init__(self, data_dir, mode='train', transforms=None):
        monet_dir = os.path.join(data_dir, 'monet_jpg')
#מחבר את הנתיב לציורים
        photo_dir = os.path.join(data_dir, 'photo_jpg')
#מחבר את הנתיב לתמונות

        if mode == 'train': #אם מיועד לאימון
            self.monet = [os.path.join(monet_dir, name) for name in sorted
(os.listdir(monet_dir))[:295]] # ציורים מהתיקייה 295מייצר רשימה של
            self.photo = [os.path.join(photo_dir, name) for name in sorted
(os.listdir(photo_dir))[:295]] # תמונות מהתיקייה 295מייצר רשימה של
        elif mode == 'test': #אם מיועד למבחן
            self.monet = [os.path.join(monet_dir, name) for name in sorted
(os.listdir(monet_dir))[295:]] #מייצר רשימה של שאר התמונות בתיקייה
            self.photo = [os.path.join(photo_dir, name) for name in sorted
(os.listdir(photo_dir))[295:]] #מייצר רשימה של שאר הציורים בתיקייה

        self.transforms = transforms #פונקציה לעיבוד תמונה

    def __len__(self):
        return len(self.monet) #גודל סט הנתונים לאימון

    def __getitem__(self, index): #פונקציה הנדרשת בשביל לעבוד עם הנתונים
        monet = self.monet[index]
        photo = self.photo[index]

        monet = Image.open(monet)
        photo = Image.open(photo)

        if self.transforms is not None:
            monet = self.transforms(monet)
            photo = self.transforms(photo)

        return monet, photo
```

פונקציה לעיבוד תמונה והפיכה לtensor

```
dset_trans = transforms.Compose([
                transforms.Resize(256), #הופך את כולם לאותו גודל
                transforms.CenterCrop(256), #חותך את כולם לאותו גודל
                transforms.ToTensor(),
                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
# מעביר את ערכי הפיקסלים מ0 עד 1 ל-1 עד 1
            ])
```

יצירת סט הנתונים

```
dataroot = "gan-getting-started" #התיקייה בה נמצאות התמונות
workers = 2 #לפי המלצת האינטרנט
batch_size = 1 #גודל הבאצ'של הנתונים
```

שימוש בפונקציית הכנת הנתונים ממקודם והכנתם לאימון

```
dataset = MonetPhotoDataset(dataroot,"train",dset_trans)
#טעינת סט הנתונים לאימון
dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
#הכנתם לאימון
                                         shuffle=True)

test_dataset = MonetPhotoDataset(dataroot,"test",dset_trans)
#טעינת סט הנתונים למבחן
test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=bat
ch_size, #הכנתם למבחן
                                         shuffle=False)
```

הצגת תמונה וציור אקראיים מהנתונים

```
for real_A, real_B in tqdm(dataloader, 0):
            show_tensor_images(torch.cat([real_A, real_B]))
            break
```

אימון

הגדרת פונקציות עלות

```
adv_criterion = nn.MSELoss() #פונקציית עלות
recon_criterion = nn.L1Loss() #פונקציית עלות

n_epochs = 200 #מספר אפוקים
display_step = 200 #כל כמה צעדים להראות דוגמא
lr = 0.0002 #קצב למידה
device = 'cuda' # הגדרת ההתקן עליו נאמן GPU
```

```
gen_AB = Generator().to(device) # יצירת מייצר והעברה לזיכרון GPU
gen_BA = Generator().to(device) # יצירת מייצר והעברה לזיכרון GPU

gen_opt = torch.optim.Adam(list(gen_AB.parameters()) + list(gen_BA.paramet
ers()), lr=lr, betas=(0.5, 0.999)) #יצירת מייעל למייצרים

disc_A = Discriminator().to(device) # יצירת מבחין תמונות והעברה לזיכרון GPU

disc_A_opt = torch.optim.Adam(disc_A.parameters(), lr=lr, betas=(0.5, 0.99
9)) #יצירת מייעל למבחין תמונות

disc_B = Discriminator().to(device) # יצירת מבחין ציורים והעברה לזיכרון GPU

disc_B_opt = torch.optim.Adam(disc_B.parameters(), lr=lr, betas=(0.5, 0.99
9)) #יצירת מייעל למבחין ציורים

def weights_init(m): #פונקציה לאתחול המשקלים במודלים נלקח מהקורס
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d):
        torch.nn.init.normal_(m.weight, 0.0, 0.02)
    if isinstance(m, nn.BatchNorm2d):
        torch.nn.init.normal_(m.weight, 0.0, 0.02)
        torch.nn.init.constant_(m.bias, 0)

# אתחול משקלים במודלים
gen_AB = gen_AB.apply(weights_init)
gen_BA = gen_BA.apply(weights_init)
disc_A = disc_A.apply(weights_init)
disc_B = disc_B.apply(weights_init)
```

```
plt.rcParams["figure.figsize"] = (10, 10) #גודל הצגת התמונות בזמן אימון

#שמירת ערכי השגיאה במערכים
disc_A_loss_history = []
disc_B_loss_history = []
gen_loss_history = []

#אתחול סופר הצעדים
cur_step = 0

for epoch in range(n_epochs): #לולאה לריצה על מספר אפוקים

    #שמים את המודלים על מצב אימון
    gen_AB.train()
    gen_BA.train()
    disc_A.train()
    disc_B.train()
    # real_B photos
    # real_A monet
    for real_A, real_B in tqdm(dataloader, 0): #רצים על התמונות בסט נתונים

        # העברת התמונות לזיכרוGPU
        real_A = real_A.to(device)
        real_B = real_B.to(device)
        cur_batch_size = real_A.shape[0] #נוכחי 'גודל באץ

        ###  עדכון מבחין תמונות###
        disc_A_opt.zero_grad() # מנקה נגזרות ישנות
        with torch.no_grad(): #מבלי לשמור נגזרות
            fake_A = gen_BA(real_B) #משיג תמונה מזויפת
        disc_A_loss = get_disc_loss(real_A, fake_A, disc_A, adv_criterion)
 #פונקציית עלות
        disc_A_loss.backward(retain_graph=True) # עדכון נגזרות
        disc_A_opt.step() # עדכון משקלים בעזרת מייעל

        ###  עדכון מבחין ציורים###
        disc_B_opt.zero_grad() # מנקה נגזרות ישנות
        with torch.no_grad(): #מבלי לשמור נגזרות
            fake_B = gen_AB(real_A) #משיג ציור מזויף
        disc_B_loss = get_disc_loss(real_B, fake_B, disc_B, adv_criterion)
 #פונקציית עלות
        disc_B_loss.backward(retain_graph=True) # עדכון נגזרות
        disc_B_opt.step() # עדכון משקלים בעזרת מייעל
```

```
        ###  עדכון מייצרים###
        gen_opt.zero_grad() #מנקה נגזרות ישנות
        gen_loss, fake_A, fake_B = get_gen_loss( #פונקציית עלות
            real_A, real_B, gen_AB, gen_BA, disc_A, disc_B, adv_criterion,
 recon_criterion, recon_criterion
        )
        gen_loss.backward() # עדכון נגזרות
        gen_opt.step() # עדכון משקלים בעזרת מייעל

        ###  הצגת דוגמאות###
        if cur_step % display_step == 0:
            print(f"Epoch {epoch}: Step {cur_step}: Generator loss: {gen_l
oss.item()}, Discriminator loss: {disc_A_loss.item()}")
            # מחבר תמונות ומציג אותם בעזרת פונקציה

            show_tensor_images(torch.cat([real_A, real_B]))
            show_tensor_images(torch.cat([fake_B, fake_A]))

        cur_step += 1 #מוסיף צעד כל תמונה

    #שומר את ערכי השגיאה במערכים
    disc_A_loss_history.append(disc_A_loss.item())
    disc_B_loss_history.append(disc_B_loss.item())
    gen_loss_history.append(gen_loss.item())

#אחרי אימון שומר את המודלים והמייעלים
torch.save({
            'gen_AB': gen_AB.state_dict(),
            'gen_BA': gen_BA.state_dict(),
            'gen_opt': gen_opt.state_dict(),
            'disc_A': disc_A.state_dict(),
            'disc_A_opt': disc_A_opt.state_dict(),
            'disc_B': disc_B.state_dict(),
            'disc_B_opt': disc_B_opt.state_dict()
        }, f"cycleGAN-_-.pth")
#שומר את מערכי השגיאה במחשב
np.save('disc_A_loss_history', disc_A_loss_history)
np.save('disc_B_loss_history', disc_B_loss_history)
np.save('gen_loss_history', gen_loss_history)
```

מדריך למשתמש

אין שום צורך בהתקנה, רק להיכנס למחברת colab עם משתמש google ולפעול לפי הוראות פשוטות.

יש להריץ לפי הסדר, אחרי שעושים Setup פעם אחת, לא צריך שוב. ניתן להשתמש גם ברזולוציות גבוהות יותר מהרזולוציה של התמונות באימון, מאחר והשכבות במודל הן קונבולוציה אין הגבלה של גודל הקלט(חוץ מזכרון). היתרון בcolab הוא האפשרות להשתמש בGPU שלהם, מה שמאפשר להשתמש בתמונות ברזולוציה גבוהה, ובזמן ריצה מהיר.

קוד

ספריות

```
import torch
from torch import nn
from torchvision import transforms
import matplotlib.pyplot as plt
from google.colab import files
from PIL import Image
from torchvision.utils import save_image
import requests
import numpy as np
```

הורדת קובץ משקלי המודל

הורדת קובץ משקלי המודל המאומן מהדרופ בוקס שלי. קוד להורדת קובץ נלקח מפורום באינטרנט

```
file_url = "https://www.dropbox.com/s/zk31a8d6e085e98/cool_generator.pth?dl=1"

r = requests.get(file_url, stream = True)

with open("cool_generator.pth", "wb") as file:
  for block in r.iter_content(chunk_size = 1024):
    if block:
      file.write(block)
```

קוד המודל המייצר

קוד המודל

הכנת המודל

יצירת מודל, טעינת המשקלים עליו, לשים על מצב של שימוש ולא אימון, מעביר לזיכרון. פונקציות לעיבוד תמונה והפיכה לטנסור

```
cool_Model = Generator()
cool_Model.load_state_dict(torch.load('/content/cool_generator.pth',map_lo
cation=torch.device('cuda')))
cool_Model.eval()
cool_Model.to('cuda')

Imagetrans = transforms.Compose([transforms.ToTensor()])
Imagetrans2 = transforms.Compose([transforms.Resize(1500), transforms.ToTe
nsor()])
```

העלאת תמונה

```
maybeImage = files.upload()

try:
  for fn in maybeImage.keys():
    path = fn

  im = Image.open(path)
  im = im.convert('RGB')

  Imageimage = Imagetrans(im).unsqueeze(0).to('cuda')
  aNumber = Imageimage.shape[2]*Imageimage.shape[3]
  if aNumber>2250000:
      print('image to large')
      Imageimage = Imagetrans2(im).unsqueeze(0).to('cuda')
except:
  print("make sure to upload IMAGE, and Setup is done")
```

ייצור תמונה

```
try:
  with torch.no_grad():
        outputs = cool_Model.forward(Imageimage)
  torch.cuda.empty_cache()
except:
  print("make sure to upload image(more than 30x30 and less than 4000x4000
), and Setup is done")
```

תצוגה מקדימה של התמונה המיוצרת

```
try:
  print('generated image preview: ')
  plt.imshow((outputs.detach().cpu().squeeze().permute(1, 2, 0) + 1) / 2)
except:
  print("Generate image first")
```

הורדת התמונה המיוצרת

```
try:
  save_image((outputs + 1) / 2, 'generated_image.png')
  files.download('generated_image.png')
except:
  print("Generate image first")
```

סיכום אישי / רפלקציה

העבודה על הפרויקט הייתה חוויה מהנה ומעניינת, ומלאת אתגרים, מהחלק של למידה עצמית של נושא חדש ומרתק ועד לכתיבת הקוד. הלמידה של התמחות הGAN הייתה מאתגרת אך גם מעניינת מאוד באותה המידה. למידת הספריה Pytorch גם הייתה מאתגרת, ספריה חדשה ושונה ממה שידעתי קודם, היא שונה מאוד מKeras, כמו: טעינת נתונים והכנתם לאימון, בניית מודל, ושלב האימון. למידת התמחות הGANs הייתה מרתקת, מדהים לראות מה ניתן לעשות עם מודלים מהסוג הזה, איך הם עובדים ולמה.

מה קיבלתי – אף פעם לא עבדתי על פרויקט בסדר גודל כזה, קיבלתי את היכולת לנהל את הזמן שלי בפרויקט והקצאת זמנים לכל שלב והערכת זמן לכל שלב. למדתי לבד התמחות של למידת מכונה באנגלית, ללא ספק השג אישי.

כלים שאני לוקח איתי להמשך – צברתי ידע נוסף בזכות התמחות הGAN, וצברתי גם הבנה מעמיקה יותר בנושא למידת מכונה. למדתי להשתמש בספריה Pytorch, ספריה שמשתמשים בה באקדמיה, ובמודלים מורכבים. והיכולת להבין מודלים שאנשים אחרים כתבו בספריה תעזור מאוד. עבודה על פרויקט בסדר גודל כזה נתנה יכולות וכלים מאוד חשובים שאקח איתי להמשך.

מה הייתי עושה אחרת לו הייתי מתחיל היום – הייתי מנסה לעבוד על רעיון שהיה לי לפרויקט אך לא ביצעתי אותו מאחר וללכת עליו הוא כמו קפיצה אל הלא נודע, לא היה לי מושג איך להתחיל בכלל, לא הבנתי כלום בנושא, וגם אחרי שלמדתי את ההתמחות לא הייתי בטוח שאני יכול לעשות עם זה משהו. הרעיון היה זיוף קול של אדם אחר, היום עם הכלים שצברתי, המשימה נראת לי שונה לחלוטין, אני מאמין שעם עוד קצת מחקר ועבודה אני אוכל לבצע פרויקט על הרעיון הזה.

ביבליוגרפיה

הקוד לטעינת הנתונים לקחתי והבנתי מקוד של אדם מKaggle -
https://www.kaggle.com/code/davidalf/cycle-gan-pytorch/notebook

עיבוד תמונה/נתונים-

https://pytorch.org/vision/stable/transforms.html

שמירה וטעינה של מודל בפייטורץ' -

https://pytorch.org/tutorials/beginner/saving_loading_models.html

שימוש בשכבת קונבולוציה בפייטורץ' -

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

התמחות הGANs שלמדתי -

https://www.coursera.org/specializations/generative-adversarial-networks-gans?