



מסמך פרויקט

Attribute-Based Facial

Generation – תשפ"ג

צוות: רפאל רייכרודל ואורי רצון

כיתה: גליל מערבי 7

ראש צוות: עדן צחקו

מנטור: דרור קיפניס

הסבר כללי על המסמך

מסמך שילווה אתכם לשבועות הקרובים וגם בהמשך השנה.
אין ליצור עותקים של המסמך, ויש להשאירו בפורמט של גוגל-דוק! כל דרך אחרת עלולה ליצור בעיות לראש הצוות או למנטור שלכם אשר עוקב אחר ההתקדמות בעבודה במסמך ובכלל. בזכות העבודה דרך גוגל-דוקס, יש גיבוי אוטומטי, סנכרון תוכן המסמך בין כל המשתתפים בו, מתן תגובות ומעקב גרסאות.

פרק 1: יזום

הסבר על המסמך:

מסמך שמטרתו סקירה כללית של הפרויקט. המסמך עוזר לפרט מהו הרעיון שלכם, וכיצד אתם מתכוונים להתחיל לעבוד עליו. הוא כולל בתוכו את הרעיון, המוטיבציה לרעיון, הצבת מטרות, מיפוי אתגרים ומשאבים שיסייעו לצוות. שימו לב שבחלק זה יש לכם דוגמאות לניסוחים משלושה פרויקטים שונים אשר ממוספרים בספרות 1-3. הדוגמאות הן יחסית בסיסיות והנכם נדרשים לפרט כמה שניתן בחלק זה. זמן הכנת מסמך יזום טוב לוקחת מספר ימים ולא ניתן לסיים את המסמך בכמה שעות בודדות.

תיאור כללי

תוכנה שתקבל מהמשתמש תמונת פרצוף אנושי, ותזהה מגוון תכונות (40) הקשורות לתווי הפנים שלו. כפיצ'ר עיקרי נוסף, תתאפשר גם האפשרות ליישם את התהליך ההפוך – המרה מתכונות של פרצוף אנושי לתמונת פנים המשקפת אותו.

מטרת הפרויקט

תחום ה-machine learning הוא נרחב ומרובה, ובנוסף מאוד חם בעשור האחרון בתעשייה עם התפתחותה של הטכנולוגיה שהביאה איתה אתגרים שאינם אפשריים לפתור באמצעות תכנות "רגיל" בלבד. לאור כך, חשוב לנו להרחיב את הידע בנושא ולהתנסות בו, וכך נוכל גם ליישם אותו בעתיד בצבא, בראיונות עבודה וכו'.

ליבה טכנולוגית

הליבה הטכנולוגית של הפרויקט היא היכולת לזהות ולאפיין תווי הפנים קונקרטיים (צבע שיער, צבע עיניים, מין וכד') ואבסטרקטיים (פנים צעירות או זקנות, שמנות או רזות וכד') מתמונת פרצוף אנושי הניתנת על ידי המשתמש.

טכנולוגיות עיקריות ושיקולים עיקריים

על מנת לממש את הליבה אנו נעמיק ברשתות נוירונים ומימושם בפייתון, ובנוסף נחקור על מודלים מבוססים רשת נוירונים שמטרתם להפיק תמונה (GAN).

נעבוד עם פייתון וניצור תחילה מודל מבוסס רשת נוירונים שמטרתו לקבל תמונה של פנים ולהפיק ממנה את התכונות הויזואליות, חלק זה יעזור לנו להבין טוב יותר את התחום ולהיות מנוסים יותר בשילוב התחום בפייתון.

מכיוון שבחרנו לעבוד עם פייתון, PyTorch היא אופציה טובה יותר מספריות אחרות לטעמנו מכיוון שהיא פותחה באופן מקורי ב-Python, קלה יותר ללמידה ולניפוי באגים, חלקים שיש להתחשב בהם עצם העובדה שאנו חדשים בתחום ולא בקיאים בו.

בנוסף, ניעזר ב-pandas בשביל סידור הדאטהסט שלנו לדטהפריים מדומה טבלה מה שיאפשר לנו גישה נוחה וקלה יותר למגוון הנתונים שבדאטהסט, ב-matplotlib על מנת להציג את התמונות המקוריות, את התמונות שיווצרו על ידי המודל או סטטיסטיקות כאלה אחרות המכריעות את פיתוח והתקדמות המודל או את מידת הטובות שלו, sklearn על מנת לחשב סטטיסטיקות כאלה ואחרות בהקשר ישיר, כנראה וכראוי, עבור matplotlib (כגון accuracy, r2score וכו'), ו-keras על מנת לעבד את התמונות (כמובן שהספרייה לא נועדה נטו לעיבוד תמונה, אך היא כוללת מגוון modules שתומכים בכך). ייתכן גם שימוש עם opencv במידה ונצטרך עיבוד תמונה נרחב יותר מהאחד שמוצע ב-keras, בעיקר בגלל שתכליתה של opencv כספרייה היא כמובן עיבוד תמונה.

לבסוף, נשתמש ב-tkinter על מנת לפתח GUI נוח למשתמש שיאפשר לו להתעסק בכלל הפיצ'רים המוצעים בפרויקט.

אתגרים טכנולוגיים ומקורות

בין האתגרים הטכנולוגיים הצפויים לנו בתהליך העבודה על הפרויקט, ניתן להניח כי ניתקל בסינון מידע לא נכון בחקר שלנו.

יכול להיות שניתקל בבעיית חומרה, מכיוון שתהליך למידת המכונה הוא computationally expensive, ולכן נצטרך להיעזר ב-Google Colab. בנוסף, ייתכן שיהיו בעיות דיזיין שיוגרמו למודל שלנו לעבוד לא כמצופה/כראוי (כדוגמת overfitting/underfitting), וכמובן שגם יהיו בעיות תכנותיות שבדרך כלל ינבעו מבעיות דיזיין (אך לא בהכרח); אותן יהיה כנראה יחסית קל לתקן בזכות מקורות המידע הנרחבים הזמינים לנו באינטרנט, בין אם זה מדריכים ביוטיוב, StackOverflow, או בלוגים ומאמרים, שיעזרו לנו לטפל בכך.

לבסוף, הקושי הצפוי ביותר יהיה למידת החומר עצמו גם לאחר סינון נכון, בעיקר עקב העובדה שקריאה והבנה מלאה של scientific papers, והנושא עצמו בו נתעסק, הוא לא דבר מובן מאליו.

מקורות מידע שהשתמשנו בהם עד כה ונוכל להמשיך להשתמש בהם במידת הצורך:

- [תרשים ענק שנותן מקורות מידע שנקודות מבט רעיוניים למידת מכונה](#)
- [האתר הרשמי של פייטורץ', בו מוסבר בהיי-לבל השימוש בספרייה](#)
- [הסבר בניית רשת נוירונים באמצעות פייטורץ'](#)
- [סדרת סרטונים שמלמדת בהרחבה על כיצד רשתות נוירונים עובדות](#)
- [חלק מקורס מטעם גוגל על למידת מכונה, המסביר על מולטיקלאסים ומימוש הרעיוני](#)
- [הסבר בוויקיפדיה על ארכיטקטורה של רשתות נוירונים שמשמשת בדרך כלל לראייה ממוחשבת](#)
- [עוד ארכיטקטורה של רשתות נוירונים שמשמשת גם היא לראייה ממוחשבת](#)
- [מאמר מדעי על השימוש במגוון סוגי GANים על מנת לסנטז תמונה](#)

ועוד...

סוגי משתמשים / קהל היעד

התוכנית יכולה להיות בשימוש גם על ידי הציבור הרחב למטרות בידור וכד', וגם על ידי גורמים אוביי

חוק על מנת ליצור תמונת פרצוף שתעזור להם לאתר פושע על פי תכונות שניתנו מעדי ראייה.

דרישות חומרה

לא נשלב רכיבי חומרה.

פתרונות קיימים

הפרויקט שלנו מהותו *face generation* שנקבע על פי קלט נוסף לתמונה (תכונות), והמוצר באתר הנ"ל מיישם את עיקרון זה: <https://generated.photos/face-generator/new> המוצר באתר מתבסס על דטהסט הרבה יותר מורכב, איכותי ואף עולה כסף, לכן סביר להניח כי לא נוכל להגיע לאותה רמת איכות היפרריאליזם כמותה; אין ממש "בעיות" ניכרות במודל תכונות-תמונה זה שנצליח להתעלות מעליהם במודל שלנו. אמנם, גם באמצעות הדאטהסט שבה אנחנו נשתמש, נצליח ארכיטקטורות רשתות נוירונים קומפלקסיות כאלה ואחרות ליצור תמונות מדויקות ואסתטיות. היתרון העיקרי שלנו על המוצר באתר הוא בעיקר העובדה שבפרויקט שלנו יהיו כמה וכמה מודלים נוספים (תלוי בתקציב הזמן שלנו) שיהיו אחראים על מגוון פיצ'רים כאלה ואחרים, כגון: פענוח תמונה של פרצוף אנושי לתכונות שלה, קליטת תמונה של פרצוף אנושי מתוך מצלמה ולפענח ממנה את התכונות שלה וכד'.

פרק 2: אפיון

הסבר על המסמך:

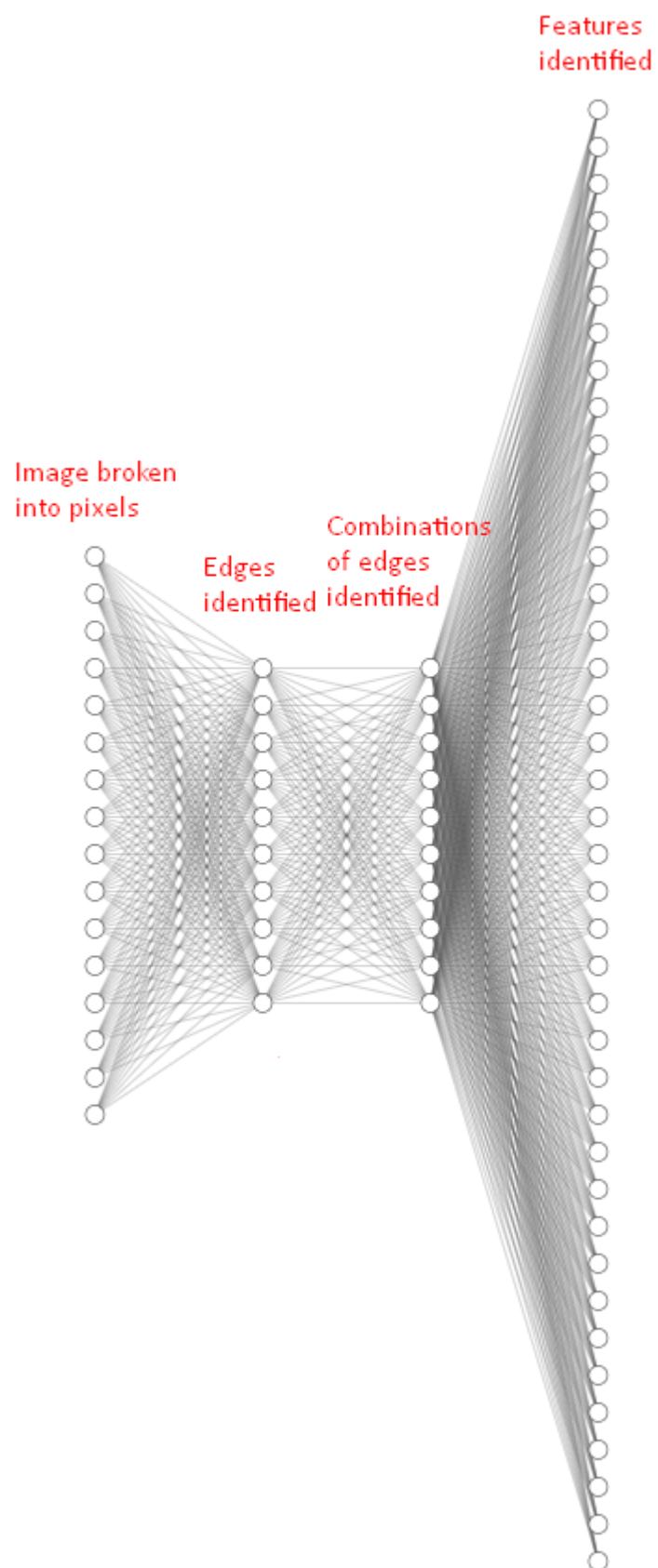
מטרת המסמך היא לפרט יותר לעומק על הפרויקט. דמיינו שמסמך זה יינתן למפתח שיתכנן ויפתח במקומכם את הפרויקט. עליכם לפרט באופן מלא ככל האפשר את מגוון הפיצ'רים שברצונכם שהמערכת תציע. כמו כן, יש מקום לפרט על טכנולוגיות, מבנה בסיס הנתונים ותרשימי זרימה.

פיצ'רים ותהליכים עיקריים

- 1. ניתוח תכונות קונקרטיות ואבסטרקטיות של פרצוף אנושי מתמונה
 - קלט: קובץ תמונה מהמשתמש (jpg, png, jfif), לא משנה ממש מה הפורמט - רק שיהיה פורמט של תמונה כמובן, סינון של דבר כזה הוא מאוד פשוט), עם פרצוף אנושי בתוכה. הרזולוציה המועדפת היא 178x218, בגלל שהתמונות מהדאטהסט שלנו הן באותה הרזולוציה הזאת (אולי תיתכן בדיקת קלט באמצעות מודל face recognition שאכן יכריע אם בתמונה יש פרצוף אנושי, ובנוסף תעשה את ה-adjustments ההכרחיים כדי לקבל תמונה שתראה כמה שיותר כמו תמונה מהדאטהסט על מנת לקבל את התוצאות הטובות ביותר וכמובן להקל על המודל בכך שאנחנו מאכילים לו רק את המידע הנחוץ).
 - פלט: רשימה של 40 תכונות פרצוף אנושי המאופיינות בתמונה שניתנה כקלט. הרשימה תתקבל בצד התוכנה לאחר פירוק ההסתברויות שיתקבלו מרשת הנוירונים שתנתח את התכונות, ובצד הלקוח תפורמט על מנת להציג את הנתונים בצורה אלגנטית וברורה; לדוגמה: אם נוצר מצב של 12% לעיניים כחולות, 85% לעיניים ירוקות ו-1% לעיניים חומות, יוצג למשתמש כי יש בתמונה שהזין עיניים ירוקות. אן שיוצגו ההסתברויות לכל אחד מצבעי העיניים (האפשרות השנייה יכולה להיות כלי דיבאג'ינג או סתם משהו סטנדרטי, בעיקר בגלל הנראות ה"פורמאלית" יותר).
 - מה הפיצ'ר עושה: מעבד תמונת פרצוף האנושי שניתנת כקלט ופולט רשימת 40 תכונות המאופיינות בה.
 - איך הוא עושה את זה: באמצעות מודל face to attributes שנבין במו ידינו. המודל יהיה על בסיס רשת נוירונים שתקבל כקלט מערך פיקסלים בתמונה, תבצע pattern recognition (ראה פיצ'ר 6, "איך הוא עושה את זה" להסבר מפורט יותר על שילוב המושג הנ"ל בפיצ'ר 1 עצמו, וגם בתרשים המוצג כדקלמן) ב-2 ה-hidden layers (שכבות נסתרות הן ה"רוטב הסודי" ברשת נוירונים. ישנם שני סוגים עיקריים של hidden layers, והם fully connected ו-convolutional, כאשר בפרויקט שלנו נשלב בין השניים. לרשת עצמה נשתמש ב-fully connected, ובהמשך נוסיף "מאחוריה" convolutional. שכבות נסתרות. מאפשרות לדגמן נתונים מורכבים באמצעות ה-nodes/נוירונים שלהן (node/נוירון) (שתי המילים בעלי אותה משמעות) = יחידה חישובית שיש לה חיבור קלט משוקלל אחד או יותר, פונקציית שפעול (activation function) המשלבת את קלטי הכניסות בצורה כלשהי וחיבור פלט), והן "מוסתרות" מכיוון שהערכים האמיתיים של ה-nodes/נוירונים שלהן אינם ידועים במערך האימון) וכפלט תוציא את "רמת הביטחון" שלה - הסתברויות התרחשות - לקיימותן של כל אחת מ-40 התכונות. פלט ההסתברויות הנ"ל של רשת הנוירונים יועבר באמצעות מבנה נתונים כזה או אחר (כנראה מערך

- הכי נוח, מהיר והגיוני כמובן) לצד התוכנה ומשם ללקוח בצורה מסודרת ומובנת.

תרשים להמחשה (תקף גם לפיצ'ר 1 וגם לפיצ'ר 6):



• 2. ייצור תמונה מתכונות קונקרטיות ואבסטרקטיות של פרצוף אנושי

- קלט: תכונות קונקרטיות ואבסטרקטיות של פרצוף אנושי שניתנות מהמשתמש כמערך של מינוס אחדים ואחדים; (מינוס אחד = התכונה לא תתקיים בתמונה שתיווצר, אחד = התכונה תתקיים בתמונה שתיווצר) הקלט יסודר ויסונן כך שלא ייווצרו התנגשויות שלא קיימות בכלל בדאטהסט שלנו ובכך ליצור עומס מיותר ולא הגיוני על הרשת שלא אומנה לשם דברים כאלה, לדוגמה: קלט שבו גם עיניים כחולות וגם עיניים ירוקות מסומנות כאחד לא יתאפשר מלכתחילה (אף על פי שבכל זאת ננסה לאפשר ולבדוק דבר כזה בשלב הדיבאגינג כדי לראות אם תיווצר אנומליה כזאת או אחרת שתואמת לציפייה אנושית, לדוגמה עין שמאלית כחולה ועין ימנית ירוקה, בניגוד לחוסר בה[הטרוכרומיה](#) בדאטהסט, כפי שכבר הוזכר לעיל).
- פלט: תמונת פרצוף אנושי שמותאמת לתיאורים הקונקרטיים והאבסטרקטיים שהמשתמש נתן בקלט.
- מה הפיצ'ר עושה: ליצור, על סמך אימון מותאם בין התמונות לבין התכונות שלהן, ועל סמך התכונות שהמשתמש יתן לנו כקלט, תמונה שמבטאת בצורה הטובה ביותר את חזון הפרצוף שהמשתמש תיאר.
- איך הוא עושה את זה: המחקר בתחום *attributes to face* הוא יחסית מצומצם ומסובך, לכן קשה לפרט כאן בצורה קולחת את דרך הפיתוח והארכיטקטורה של רשת הנוירונים. הרשת תיבנה באמצעות [הכלים שמוצעים לנו ב-PyTorch](#), כאשר הארכיטקטורה שלה תותאם לרשת בתיאור התיאורי שנמצא ב[דף המדעי שעוסק בתחום זה](#), על סמך התוצאות האמינות [שמודל המבוסס על דף זה](#) נותן, במידה ונחליט לפתח את רשת הנוירונים מאחורי הפיצ'ר בכוחות עצמנו ולא באמצעות המודל המוכן שמובא לעיל.

● 3. ניתוח תכונות קונקרטיות ואבסטרקטיות של פרצוף אנושי ממצלמה

- קלט: תמונה שנלכדת ממצלמת המשתמש (יכולה להיות סתם תמונה, או במקרה הכי טוב - תמונה של הפרצוף שלו). מפורט בהרחבה יותר בפיצ'ר 10.
- פלט: רשימת של 40 תכונות המאופיינות בתמונה שנלכדה ממצלמת המשתמש
- מה הפיצ'ר עושה: לוכד תמונת פרצוף ממצלמת המשתמש ופולט רשימת 40 תכונות המאופיינות בה
- איך הוא עושה את זה: באמצעות [מודל face recognition מפותח ו-pretrained](#), יהיה ניתן ללכוד בצורה מדויקת פרצוף אנושי, גם בפורטרייט וגם בפרופיל. לאחר הלכידה, התמונה תועבר למודל ה-*face to attributes* (פיצ'ר 1), ומשם תיפלט רשימת התכונות.

● 4. יצירת דאטהפריים

- קלט: קובץ (Comma Separated Values) csv, דמוי טבלה כאשר ננסה לפתוח אותו ב-Microsoft Excel. במקרה שלנו, זה יהיה הקובץ שמכיל את התכונות הבא מה-CelebA Dataset
- פלט: מבנה נתונים דמוי טבלה, שמאפשר גישה ושליפה של נתונים בצורה נוחה ומהירה
- מה הפיצ'ר עושה: יוצר דאטהפריים דמוי טבלה באמצעות קלט של קובץ csv.
- איך הוא עושה את זה: באמצעות הפונקציה *read_csv* של הספרייה *pandas* (אין כאן מה להרחיב ממש, הפונקציה עושה הכל). הדאטהפריים מאפשר גישה ושליפה של נתונים בצורה נוחה ומהירה, במקום בצורה סיזיפית ובזבזנית עם *hard code* מתוך קובץ ה-csv. בנוסף, *pandas* הוכיחה את עצמה בתור ה-*go-to library* לכל מה שקשור ל-*data analysis*, לכן אנחנו לא חוששים להשתמש בה ולהעדיף אותה

על ספריות כאלה ואחרות, הן על סמך פופולריותה והן על סמך התוצאות המרשימות שהשימוש בה מבטיחות.

• 5. פירוק תמונה למערך פיקסלים

- קלט: קובץ תמונה מהמשתמש (jpg, png, jfif), לא משנה ממש מה הפורמט – רק שיהיה פורמט של תמונה כמובן, סינון של דבר כזה הוא מאוד פשוט).
- פלט: מערך NumPy תלת מימדי שמכיל בכל אחד מהתאים שלו ייצוג של פיקסל אחד באמצעות הפורמט (height, width, channels) (לא נתעסק בפרטים ופשוט נצא מנקודת הנחה שזה "pixel data" שנחזף לנו).
- מה הפיצ'ר עושה: יוצר מערך פיקסלים באמצעות תמונת קלט
- איך הוא עושה את זה: באמצעות הפונקציה `img_to_array` של התת-ספרייה `keras.preprocessing.image`. קובץ התמונה מתקבל באמצעות הפונקציה `load_img` של אותה תת ספרייה, כך שהוא נשמר כאובייקט ממבנה הנתונים PIL (Python Imaging Library), אובייקט שמשמש לשמירת מידע גרפי, בעיקר תמונות. בהינתן העברת האובייקט הנ"ל ל-`img_to_array`, פלט הפונקציה יהיה מערך ה-NumPy התלת מימדי שהוזכר לעיל, שבו ניתן יהיה להשתמש לצורכי הקלט של רשת הנורונים שלנו, שהם קבלת התמונה כמערך פיקסלים.

• 6. רשת נורונים – תמונה לתכונות

- קלט: מערך NumPy תלת מימדי (ראה פיצ'ר 5).
- פלט: רשימת "רמות ביטחון", או הסתברויות, שקשורות למידת הניכרות של כל אחד מתכונות הפרצוף האנושיות שבתמונה. לדוגמה: 83% "ביטחון" לעיניים ירוקות, אן פשוט "עיניים ירוקות" (לא החלטנו על הקטע ה"ויזואלי" בצורה חד משמעית עדיין; מה שחשוב זה שנקבל הסתברויות כפלט מהרשת)
- מה הפיצ'ר עושה: לקלוט מערך NumPy תלת מימדי שמייצג את הפיקסלים שבתמונה שניתנה מהמשתמש, ולפלוט את ניכרות התכונות הוויזואליות – קונקרטיות ואבסטרקטיות – של הפרצוף
- איך הוא עושה את זה: הרשת תיבנה באמצעות הכלים שמוצעים לנו ב-PyTorch. הרשת תקבל את מערך ה-NumPy התלת מימדי ואז באמצעות שתי `hidden layers` תבצע `pattern recognition` לכל אחד מהפיצ'רים ההכרחיים – שכבה אחת שתזהה את קצוות ה"אובייקטים" שנצטרך, לדוגמה: מסביב לאחת מהעיניים שבפרצוף, יהיו קווים קטנים שיסודרו בתהליך ארוך; מאין משיקים לעין עצמה, ושכבה אחרת שתחבר ביחד את הקווים הקטנים האלו למצולע, במקרה שלנו לאליפסה, שהיא העין עצמה כמובן, ומתוכה נוכל להסיק שבאמצע האליפסה נמצאים האישונים, שמורכבים משני עיגולים, והאחד הגדול יותר מכיל עליו את צבע העין עצמה. בתכלית הדברים, אנחנו מצמצמים את חיפוש הפיצ'רים לצורות גיאומטריות. לבסוף, הרשת פולטת את "רמת הביטחון" שלה לגבי כל אחד מ-40 הפיצ'רים, מה שמועבר לאחר מכן חזרה לצד התוכנה כמערך. תיתכן אפשרות כי במקום לנסות לפתח את שתי ה-`hidden layers` בעצמנו, ניקח מודל של רשת אחרת, המיישמת את אותו קונספט כמו שלנו, ונחבר אליה את שכבות הקלט והפלט – תהליך זה מכונה `transfer learning`. דבר זה יכול להבטיח תוצאות טובות יותר לטווח הארוך וגם לחסוך לנו זמן ותעסוקה יקרים, אבל משום שהכל נתון עדיין לשיקולי דעת ובגדר דבר אבסטרקטי ועתידי, יש מצב שנוותר על הרעיון ברגע האמת.

• 7. רשת נורונים – תכונות לתמונה

- קלט: תכונות קונקרטיות ואבסטרקטיות של פרצוף אנושי שניתנות מהמשתמש כמערך של מינוס אחדים ואחדים; (מינוס אחד = התכונה לא תתקיים בתמונה שתיווצר, אחד = התכונה תתקיים בתמונה שתיווצר) הקלט יסודר ויסונן כך שלא יוצרו התנגשויות שלא קיימות בכלל בדאטהסט שלנו ובכך ליצור עומס מיותר ולא הגיוני על הרשת שלא אומנה לשם דברים כאלה, לדוגמה: קלט שבו גם עיניים כחולות וגם עיניים ירוקות מסומנות כאחד לא יתאפשר מלכתחילה (אף על פי שבכל זאת ננסה לאפשר ולבדוק דבר כזה בשלב הדיבאגינג כדי לראות אם תיווצר אנומליה כזאת או אחרת שתואמת לציפייה אנושית, לדוגמה עין שמאלית כחולה ועין ימנית ירוקה, בניגוד לחוסר בה[הטרוכרומיה](#) בדאטהסט, כפי שכבר הוזכר לעיל).
- פלט: תמונת פרצוף אנושי שמותאמת לתיאורים הקונקרטיים והאבסטרקטיים שהמשתמש נתן בקלט.
- מה הפיצ'ר עושה: ליצור, על סמך אימון מותאם בין התמונות לבין התכונות שלהן, ועל סמך התכונות שהמשתמש יתן לנו כקלט, תמונה שמבטאת בצורה הטובה ביותר את חזון הפרצוף שהמשתמש תיאר.
- איך הוא עושה את זה: המחקר בתחום *attributes to face* הוא יחסית מצומצם ומסובך, לכן קשה לפרט כאן בצורה קולחת את דרך הפיתוח והארכיטקטורה של רשת הנוירונים. הרשת תיבנה באמצעות [הכלים שמוצעים לנו ב-PyTorch](#), כאשר הארכיטקטורה שלה תותאם לרשת בתיאור התיאורי שנמצא ב[דף המדעי שעוסק בתחום זה](#), על סמך התוצאות האמינות [שמודל המבוסס על דף זה](#) נותן, במידה ונחליט לפתח את רשת הנוירונים מאחורי הפיצ'ר בכוחות עצמנו ולא באמצעות המודל המוכן שמובא לעיל.

● 8. רשת נוירונים - אימון (תמונה לתכונות)

- קלט: חלק ה-*train* של ה-CelebA דאטהסט.
- פלט: מודל שמצליח לקטלג תכונות פרצוף אנושי בתמונת פרצוף קלט שניתנת מהמשתמש.
- מה הפיצ'ר עושה: מקבל כקלט את חלק ה-*train* ב-CelebA דאטהסט, ומצליח באמצעותו לפתח באמצעות רשת הנוירונים אלגוריתם שמאפשר לו לסווג תמונה לתכונות שלה.
- איך הוא עושה את זה: משום שבספריית PyTorch הדאטהסט הנ"ל מובנה כבר דרך `torchvision.datasets.CelebA`, ננצל את זה על מנת לקצר תהליכים, ונניבא כך את ה-*train-validation-test* באמצעות *ratios* שיצרו כבר בשבילנו. ניקח כל תמונה מה-*training segment*, נאכיל אותה לרשת הנוירונים לאחר עיבוד (ראו פיצ'ר 6) ולבסוף נבדוק את הביצועים שלה - ניתן למכונה את ה-*test segment* ונראה אם התוצאות שהיא מניבה הגיוניות. לבסוף, באמצעות ה-*validation segment* ושיקולים אנושיים על סמך תוצאות הפלט, נדע פחות או יותר למה לעשות *tweak* על מנת לקבל את המודל הטוב ביותר.

● 9. רשת נוירונים - *benchmarking* / ניתוח גרפי וסטטיסטי של איכות הרשת

- קלט: רשת נוירונים שמאומנת לנתח תכונות פרצוף אנושיות, קונקרטיות ואבסטרקטיות מתמונת פרצוף אנושי.
- פלט: גרפים וסטטיסטיקות כאלה ואחרות שעוזרות לנו לקבוע את איכותו של המודל שיצרנו.
- מה הפיצ'ר עושה: מקבל רשת נוירונים וקובע את איכותה באמצעות מגוון פרמטרים.
- איך הוא עושה את זה: ניתן יהיה להשתמש ב-*sklearn.metrics* וב-*matplotlib* כדי למדוד את איכות המודל בצורה סדייפית אך באותו הזמן ברגנית (אנו יודעים

ש-*sklearn* זוהי ספרייה שבאמצעותה ניתן לאמן מודלים וכד', אך בתוך *metrics*. ישנם מגוון פונקציות שיעזרו בתהליך הבנצ'מרקינג מה שמשתייך גם לאפיון ושרטוט הגרפים), או באמצעות [שירות אונליין](#) כדה או אחר.

להלן דוגמה לבניית ואימון רשת נוירונים, אך עם ה-MNIST דאטהסט:

בניית המודל

```
# Imports
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
import torchvision.datasets as datasets
import torchvision.transforms as transforms

# Create Fully Connected Network
class NN(nn.Module):
    def __init__(self, input_size, num_classes):
        super(NN, self).__init__()
        self.fc1 = nn.Linear(input_size, 50)
        self.fc2 = nn.Linear(50, num_classes)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

הכנת הדאטהסט לאימון

```
# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Hyperparameters
input_size = 784
num_classes = 10
learning_rate = 0.001
batch_size = 64
num_epochs = 1

# Load Data
train_dataset = datasets.MNIST(root='dataset/', train=True, transform=transforms.ToTensor(), download=True)
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_dataset = datasets.MNIST(root='dataset/', train=False, transform=transforms.ToTensor(), download=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=True)

# Initialize network
model = NN(input_size=input_size, num_classes=num_classes).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

אימון רשת הנוירונים + פונקציה שמאמנת את המודל ולבסוף מבצעת *benchmarking* קטן

```
# Train Network
for epoch in range(num_epochs):
    for batch_idx, (data, targets) in enumerate(train_loader):
        # Get data to cuda if possible
        data = data.to(device=device)
        targets = targets.to(device=device)

        # Get to correct shape
        data = data.reshape(data.shape[0], -1)

        # forward
        scores = model(data)
        loss = criterion(scores, targets)

        # backward
        optimizer.zero_grad()
        loss.backward()

        # gradient descent or adam step
        optimizer.step()

# Check accuracy on training & test to see how good our model
def check_accuracy(loader, model):
    num_correct = 0
    num_samples = 0
    model.eval()

    with torch.no_grad():
        for x, y in loader:
            x = x.to(device=device)
            y = y.to(device=device)
            x = x.reshape(x.shape[0], -1)

            scores = model(x)
            _, predictions = scores.max(1)
            num_correct += (predictions == y).sum()
            num_samples += predictions.size(0)

    print(f'Got {num_correct} / {num_samples} with accuracy {float(num_correct)/float(num_samples)*100:.2f}')

    model.train()
    return acc
```

קריאה לפונקציה שמאמנת את המודל, ראשית על חלק ה-*train* ואז על החלק ה-*test*

```
check_accuracy(train_loader, model)
check_accuracy(test_loader, model)
```

בניית ואימון רשת נוירונים ל-MNIST מסובכת בהרבה פחות מרשתות הנוירונים שניצור לפיצ'רים 6 ו-7 (בעיקר 7, כי ה-MNIST דאטהסט לרוב, בדומה לרעיון שלנו, שקשור ל-CelebA דאטהסט, משמש לזיהוי מספר (תכונה קונקרטית אחת ויחידה מתוך 10 אפשרויות - המספרים 0-9) מתמונה); התמונות משומשות להמחשה בלעדית לכיצד תהליך הבנייה והאימון הבסיסיים של רשת נוירונים תראה בלבד, וסוטה בקיצוניות מהקוד שיהיה מהתוצר הסופי שלנו.

• 10. לכידת תמונה ממצלמת המשתמש והתאמתה (ניכר בפיצ'ר 3 ו-11)

- קלט: מצלמה עובדת המכוונת, בעדיפות ראשונית, לפרצוף המשתמש.
- פלט: תמונה שבתוכה רק הפרצוף האנושי מתוך תמונה שנלכדת מהמצלמה.
- מה הפיצ'ר עושה: לוכדת ספציפית פרצוף אנושי מתמונה שנלכדת ממצלמה.
- איך הוא עושה את זה: תחילה, נלכוד תמונה מהמצלמה עצמה באמצעות *opencv*. לאחר מכן, באמצעות מודל face recognition מפותח ו-*pretrained*, יהיה ניתן ללכוד בצורה מדויקת פרצוף אנושי, גם בפורטרייט וגם בפרופיל (כמובן שלכידת

הפרצוף עצמו קונקרטי ניתן לביצוע עם opencv, אך הוא לא עיקר הפרויקט
(כמובן)

• 11. ניקוי רעשים מתמונה

- קלט: קובץ תמונה מהמשתמש (jpg, png, jpeg), לא משנה ממש מה הפורמט – רק שיהיה פורמט של תמונה כמובן, סינון של דבר כזה הוא מאוד פשוט) או תמונה שנלכדת במצלמת המשתמש (ראה פיצ'ר 10).
- פלט: תמונה שהרעש שלה מופחת במידה ניכרת.
- מה הפיצ'ר עושה: מקבל תמונה ומפחית את הרעש שבה.
- איך הוא עושה את זה: נשתמש ב**מודל pretrained שמטרתו לנקות כמה שיותר רעש מתמונה**, במקום לנסות בדרך סיזיפית לעשות את זה ידנית עם opencv (על אף שהדבר אפשרי, הוא פשוט יקח יותר זמן, אך שוב, דבר זה מיותר משום שזהו לא עיקר הפרויקט, אלא פיצ'ר אחד יחסית קטן מבחינת צורך השימוש – כלי עזר, אם לדייק).

• 12. עריכת תמונה

- קלט: קובץ תמונה מהמשתמש (jpg, png, jpeg), לא משנה ממש מה הפורמט – רק שיהיה פורמט של תמונה כמובן, סינון של דבר כזה הוא מאוד פשוט), עם פרצוף אנושי בתוכה (אולי תיתכן בדיקת קלט באמצעות מודל face recognition שאכן יכריע אם בתמונה יש פרצוף אנושי, ובנוסף תעשה את ה-adjustments ההכרחיים כדי לקבל תמונה שתראה כמה שיותר כמו תמונה מהדאטהסט על מנת לקבל את התוצאות הטובות ביותר וכמובן להקל על המודל בכך שאנחנו מאכילים לו רק את המידע הנחוץ; זה לא משהו שאנחנו בונים עליו בינתיים, אבל נקודה טובה למחשבה). בנוסף, יסופקו פרמטרים חדשים
- פלט: תמונת פרצוף אנושי חדשה, המכילה את השינויים שניתנו לתכונות מהקלט.
- מה הפיצ'ר עושה: נותן למשתמש את האפשרות לשנות תכונות קונקרטיות ואבסטרקטיות בפרצוף האנושי שצירף בתמונה.
- איך הוא עושה את זה: נשתמש במודל ה-face to attributes שלנו על מנת לנתח את התמונה לתכונות שלה. לאחר מכן, נאפשר למשתמש לעשות את העריכות ההכרחיות לטעמו, ולבסוף נשלח את התכונות החדשות למודל ה-attributes to face, כך שנחולל תמונה עם התכונות החדשות יחד עם אלה שלא שונו. כמובן שלא נוכל לקבל את אותה התמונה המקורית עם התכונות בלבד, לכן נאתחל איזה שהוא noise התחלתי, בהשפעת התמונה המקורית, שיעזור למודל בעת סינתוז התמונה לכוון לפרצוף שדומה לפרצוף שהיה בתמונה המקורית. יתכן כי במקום לסבך את כל העניין, ניעזר ב-DiscoGAN, קונספט ואף מודל ממומש של רשת נוירונים המיישמת את ארכיטקטורת ה-GAN (Generative Adversarial Network), שמאפשר לנו לעשות את בדיוק מה שאנחנו רוצים – לקחת תמונה, לשנות בה תכונה אחת או יותר, ואז להנפיק תמונה חדשה עם העריכות שהזנו. שימוש ב-DiscoGAN עשוי להבטיח תוצאות הרבה יותר טובות ממימוש עצמי וכך לחסוך זמן יקר, שכן הוא כבר מצליח לפתור את בעיית התמונה –> קבלת תכונות התמונה –> הזנת תכונות חדשות –> פליטת תמונה שדומה לתמונה המקורית אך עם התכונות החדשות.

תוצאות DiscoGAN עם הדאטהסט שלנו, CelebA:



Figure 7. (a,b) Translation of gender in Facescrub dataset and CelebA dataset. (c) Blond to black and black to blond hair color conversion in CelebA dataset. (d) Wearing eyeglasses conversion in CelebA dataset (e) Results of applying a sequence of conversion of gender and hair color (left to right) (f) Results of repeatedly applying the same conversions (upper: hair color, lower: gender)

• GUI.13

- קלט: תלוי במשתמש - הוא יכול להשתמש כרצונו בכל אחד מהפיצ'רים שהוצגו בתרשים בפיצ'ר 1.
- פלט: הפלט תלוי בפיצ'ר שהמשתמש בחר להשתמש בו מהפיצ'רים שהוצגו בתרשים בפיצ'ר 1.
- מה הפיצ'ר עושה: נותן למשתמש גישה נוחה, ויזואלית וגמישה לכלל הדברים שהפרויקט מציע.
- איך הוא עושה את זה: נעשה זאת באמצעות [tkinter](#), ספריית פייתון שבאמצעותה אפשר לפתח ממשקי GUI (Graphical User Interface). החלק הזה של הפרויקט הוא הכי פחות חשוב, לכן נקדיש לו זמן רק לאחר שנסיים לפתח את כל הפיצ'רים העיקריים, משום שזה עדיין אפשרי לספק קלט ולקבל פלט מהפיצ'רים העיקריים בפרויקט גם בהיעדר GUI.

• 14. חלוקה ל-utilities: א. שליפת נקודתית של תמונה אחת או יותר לפי תכונות ניתנות (בניית המודל בשלבים)

- קלט: תכונה או מקבץ תכונות אשר המשתמש בוחר לדלות ממאגר הנתונים
- פלט: הפלט תלוי בקלט; הוא יהיה תמונה או תמונות (נדיר שנקבל תמונה אחת בלבד מתוך הסיווג) שיוחזרו כמערך של אובייקטים מסוג PIL (אובייקט שמשמש לשמירת מידע גרפי, בעיקר תמונות).

- מה הפיצ'ר עושה: מאפשר לנו לספק למודל גישה נוחה לתמונות בעלות תכונה או תכונות ספציפיים לפיהם הוא יתאמן לזהות אותם (לדוגמה: אימון אינדיבידואלי רק על אנשים מרכיבי משקפיים, על מנת זיהוי ממוקד וחד יותר של אנשים מרכיבי משקפיים).
- איך הוא עושה את זה: ניצור פונקציית עזר אשר תקבל מידע מסוים לפיו תסנן את התמונות המכילות מידע זה מתוך הדאטהפריים (ראה פיצ'ר 5), נעבור על הדאטהפריים ועבור כל תמונה נבדוק האם התכונה לפיה בחר המשתמש לסנן מתקיימת בה. ניקח כל שם של תמונה שמצאנו התאמה בשבילה (לדוגמה: 0000001, 0000004, 0000005 וכו' הם אטרקטיביים), נשים אותה בפונקציה `PIL.Image.open()` כדי לקבל "אובייקט תמונה" לכל אחת מהן וכך נשלוף את התמונות הנחוצות מהדאטהסט.

● **15. חלוקה ל-utilities: ב. פירוק המודל הגדול שבפיצ'ר 6 למודלים קטנים שמזהים תכונה אחת או יותר, ואימונם (בניית המודל בשלבים)**

- קלט: הפלט של פיצ'ר 14; תמונה או תמונות שעונות על התכונות שנספק לפונקציית העזר, לפי המסווג של המודל הקטן אותו נאמן.
- פלט: מודלים, או יותר נכון רשתות נוירונים, שמסוגלים לזהות בתמונה תכונה אחת או יותר מתוך אלה שסופקו לה.
- מה הפיצ'ר עושה: מאפשר פיתוח יותר נקודתי של פיצ'ר 6 בכך שהוא מאפשר למודל להתעסק קונקרטי בתכונה אחת או יותר, ולא ישר בכל ה-40.
- איך הוא עושה את זה: לאחר קבלת התמונות מתוך פונקציית העזר בפיצ'ר 14, נשלח כל תמונה לעיבוד (ראה פיצ'ר 5) ולאחר מכן לרשת נוירונים הדומה בארכיטקטורה שלה לזאת שבפיצ'ר 6, ונאמן אותה לזיהוי אותו מידע מסוים בתמונה (כדלקמן למובא לעיל, סינון מרכיבי משקפיים. חשוב לציין כי אימון המודלים הקטנים יעשה בצורה כמעט זהה לחלוטין לזו שמוזכרת ב"איך הוא עושה את זה" שבפיצ'ר 1 + יש דוגמת קוד מפורטת ב"להלן דוגמה לבניית ואימון רשת נוירונים, אך עם ה-MNIST דאטהסט"). כך, נוכל לא רק לעשות מוניטורינג ספציפי יותר על האימון של כל אחת מהתכונות, אלא נאפשר פיתוח חד ואיכותי יותר לכל אחת מהתכונות, בניגוד לאם היינו מנסים ישר להכריח את הרשת לסווג את כל 40 התכונות בבת אחת. בסופו של דבר, יהיו לנו מספר רשתות נוירונים קטנות אותן "נחבר" יחד לרשת הנוירונים שבפיצ'ר 6 (ראה פיצ'ר 16). בעצם, נפתח גרסאות דמו של רשת הנוירונים הנ"ל.

● **16. חלוקה ל-utilities: ג. חיבור המודלים הקטנים למודל אחד גדול (בניית המודל בשלבים)**

- קלט: מודלים "קטנים" (כאשר כל אחד יודע לסווג תמונות לפי מספר מצומצם של תכונות) אשר ירכיבו את המודל הגדול.
- פלט: רשת הנוירונים שבפיצ'ר 6.
- מה הפיצ'ר עושה: מבצע מעין `transfer learning` (לקיחת מודל קיים המאמן כבר לבצע סיווג תמונה, הורדת ה-layer האחרון שלו, "הקפאת" ה-layers הקודמים לו אשר מאומנים כבר והוספת layers משלנו המתאימים לסיווג תכונה כלשהי ולאחר מכן אימון קונקרטי של רק ה-layers אותם הוספנו לקבלת המודל הרצוי, אן פיתוח השכבות לפני ה-layer האחרון בעצמנו, מה שפחות מומלץ בהינתן העובדה כי מודלים שמטרתם לסיווג תמונה `pretrained` כבר נותנים תוצאות מעולות, לכן אין

היגיון בלנסות להמציא את הגלגל מחדש, אלא אם יש דרישה שחורה על גבי לבנה לכך).

○ איך הוא עושה את זה: מבחינת קונספט, זה כנראה ייכרך ב"הוספת" (הקושי העיקרי בהבנת הדבר ויישום הפיצ'ר) ה-nodes של ה-hidden layer/s מתוך רשתות הנוירונים שפיתחנו בפיצ'ר 15, שכן ייתכן כי לכל רשת נוירונים כזאת לא יהיו הרבה nodes ב-hidden layer/s עקב דיהוי רק תכונה אחת או מספר קטן של תכונות. בפועל, מעט מאוד אנשים מאמנים רשת [Convolutional Neural Network](#) שלמה מאפס (עם אתחול אקראי), כי נדיר יחסית שיש מערך נתונים בגודל מספיק. במקום זאת, מקובל לאמן מראש ConvNet על מערך נתונים גדול מאוד (למשל ImageNet, המכיל 1.2 מיליון תמונות עם 1000 קטגוריות), ולאחר מכן להשתמש ב-ConvNet כאתחול או כ-fixed feature extractor (מחלף תכונות קבוע, בינתיים נתייחס אליו במלואו כקופסה שחורה כי מה שהוא עצמו עושה, מא' עד ת', לא ממש משנה - במבט על, פחות או יותר מעבד את התמונה) עבור המשימה העניינית.

שני תרחישי transfer learning עיקריים אלה נראים כך:

1. כוונת עדין של ה-ConvNet: במקום אתחול אקראי, אנו מאתחלים את הרשת עם רשת מאומנת מראש. שאר האימון נראה כרגיל.

2. ConvNet כ-fixed feature extractor: כאן, נקפיד את המשקולות עבור כל הרשת מלבד זו של ה-layer הסופית המחוברת במלואה. ה-layer האחרונה המחוברת במלואה מוחלפת בשכבה חדשה עם משקלים אקראיים ורק ה-layer הזו מאומנת.

מבחינת הקוד, ניעזר ב**דוגמה הזאת על ה-CIFAR10 דאטהסט** שמשתמש בתמונות צבעוניות בניגוד לשחור-לבן (בדומה לדאטהסט שלנו) כדי לנסות ליישם CNN משלנו / להיעזר במודל CNN שכבר מאומן לדיהוי תמונה (האופציה המועדפת) שאיתו נשלב את ה-transfer learning, וב**מדריך הזה** על מנת לבצע את ה-transfer learning הממשי עצמו.

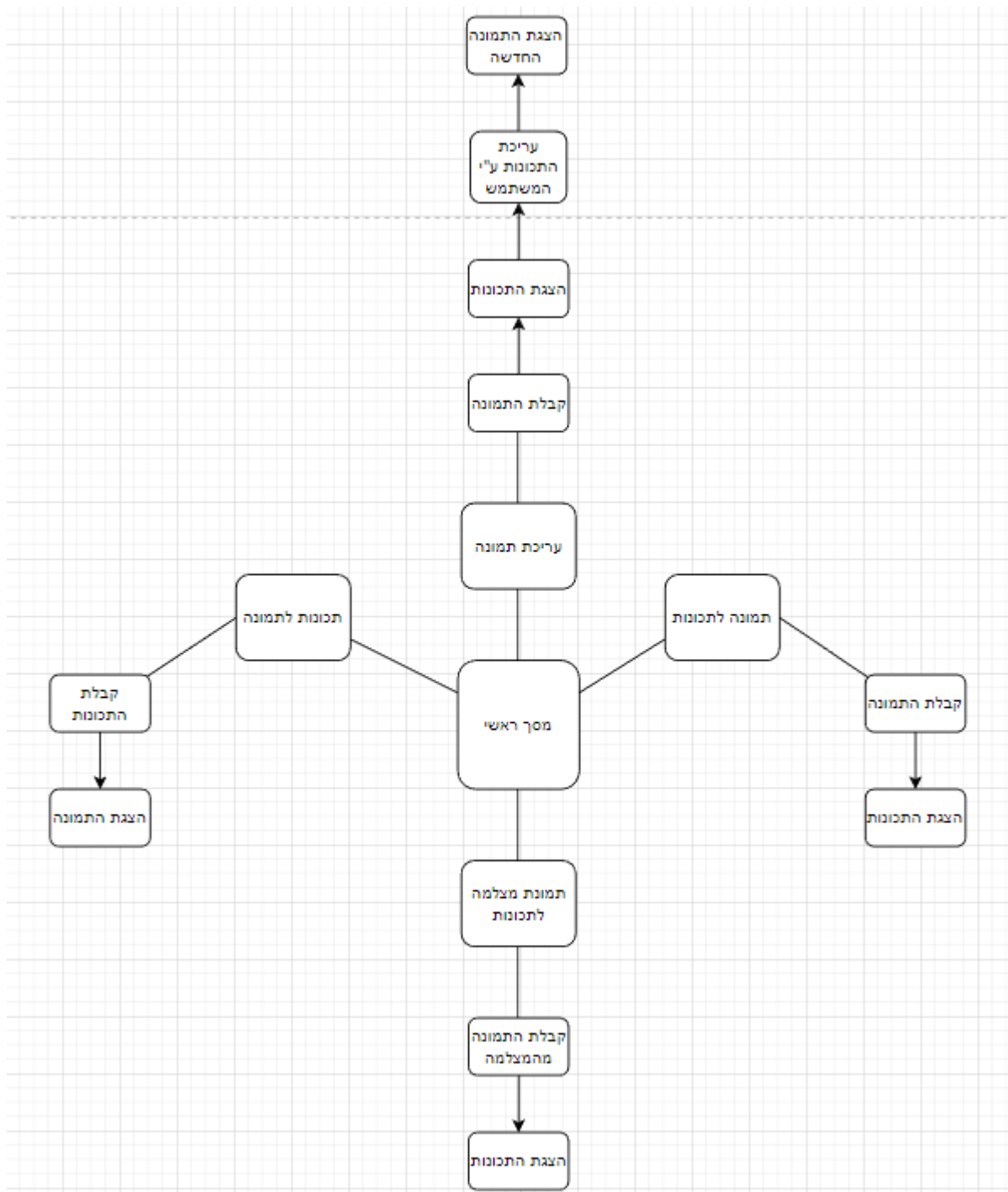
טכנולוגיות

<u>פיצ'ר/תהליך</u>	<u>טכנולוגיות ושפות תכנות</u>	<u>משאבים נדרשים ושירותים חיצוניים</u>
ניתוח תכונות קונקרטיות ואבסטרקטיות של פרצוף אנושי מתמונה	בניית מודל מבוסס רשת נוירונים בעזרת ספריות <i>PyTorch</i> ו- <i>NumPy</i> . נשתמש בשפת פייתון.	כרטיס מסך טוב על מנת לקבל תוצאות טובות מהמודל מפאת הדרישה ה-computationally-expensive של למידת מכונה. משום שכרטיס מסך טוב לא זמין לנו, נשתמש ב-Google Colab בשביל ביסוס אימון המודל.
ייצור תמונה מתכונות קונקרטיות ואבסטרקטיות של פרצוף אנושי	תהליך בניית המודל ואימונו תתבצע באופן דומה לאימון המודל המוצג לעיל. נשתמש בשפת פייתון.	כרטיס מסך טוב על מנת לקבל תוצאות טובות מהמודל מפאת הדרישה ה-computationally-expensive של למידת מכונה. משום שכרטיס מסך טוב לא זמין לנו, נשתמש ב-Google Colab בשביל ביסוס אימון המודל.
ניתוח תכונות קונקרטיות ואבסטרקטיות של פרצוף אנושי ממצלמה	נשתמש במודל <i>face recognition</i> -pretrained ללכידת התמונה, ולאחר מכן התמונה תועבר למודל ה-face-to attributes שיוציא את התכונות. נשתמש בשפת פייתון.	מצלמה מהמשתמש + הדרישות של פיצ'ר 1 בגלל ההסתמכות שלו עליו.
יצירת דאטהפריים	נשתמש בשפת פייתון, בה באמצעות הספרייה <i>pandas</i> והפונקציה <i>read_csv</i> המסופקת ממנה, נוכל להפיק טבלת נתונים קלה לגישה ושליפה של נתונים באמצעות קלט קובץ <i>csv</i> .	לא נדרשים כרגע.
פירוק תמונה למערך פיקסלים	נשתמש בשפת פייתון, בה באמצעות תת הספרייה <i>keras.preprocessing.image</i> נשתמש בפונקציה <i>img_to_array</i> וניצור כך מערך פיקסלים מתמונה רלוונטית.	לא נדרשים כרגע.
רשת נוירונים - תמונה לתכונות	הרשת תיבנה באמצעות הכלים שמוצעים לנו ב- <i>PyTorch</i> , בשפת פייתון. הרשת תקבל את	לא נדרשים כרגע.

	<p>מעריך ה-NumPy התלת מימדי ואז באמצעות שתי <i>hidden layers</i> מבצע <i>pattern recognition</i> לכל אחד מהפיצ'רים ההכרחיים.</p>	
לא נדרשים כרגע.	<p>הרשת תיבנה באמצעות הכלים שמוצעים לנו ב-PyTorch, בשפת פייתון, כאשר הארכיטקטורה שלה תותאם לרשת בתיאור התיאורטי שנמצא בדף המדעי שעוסק בתחום זה, על סמך התוצאות האמינות שמודל המבוסס על דף זה נותן, במידה ונחליט לפתח את רשת הנורונים מאחורי הפיצ'ר בכוחות עצמנו ולא באמצעות המודל המוכן שמובא לעיל.</p>	רשת נורונים - תכונות לתמונה
<p>כרטיס מסך טוב על מנת לקבל תוצאות טובות מהמודל מפאת הדרישה ה-computationally expensive של למידת מכונה. משום שכרטיס מסך טוב לא זמין לנו, נשתמש ב-Google Colab בשביל ביסוס אימון המודל.</p>	<p>בספריית PyTorch שבשפת פייתון הדאטהסט הנ"ל מובנה כבר דרך <code>torchvision.datasets.CelebA</code>, לכן ננצל את זה על מנת לקצר תהליכים ולזרז את תהליך האימון.</p>	רשת נורונים - אימון
לא נדרשים כרגע.	<p>נשתמש בשפת פייתון. ניתן יהיה להשתמש ב-<code>sklearn.metrics</code> וב-<code>matplotlib</code> כדי למדוד את איכות המודל בצורה סזיפית אך באותו הזמן בררנית, או באמצעות שירות אונליין כזה או אחר.</p>	רשת נורונים - benchmarking
מצלמה מהמשתמש.	<p>נשתמש בשפת פייתון. תחילה, נלכוד תמונה מהמצלמה עצמה באמצעות <code>opencv</code>. לאחר מכן, באמצעות מודל face recognition מפותח ו-pretrained, יהיה ניתן ללכוד בצורה מדויקת פרצוף אנושי, גם בפורטרייט וגם בפרופיל.</p>	<p>לכידת תמונה ממצלמת המשתמש והתאמתה</p>

ניקוי רעשים מתמונה	נשתמש בשפת פייתון, בה ניעזר במודל pretrained שמטרתו לנקות כמה שיותר רעש מתמונה , במקום לנסות בדרך סזיפית לעשות את זה ידנית עם opencv .	לא נדרשים כרגע.
עריכת תמונה	נשתמש בשפת פייתון, בה ניעזר במודל ה- <i>face to attributes</i> וההופכי לו, מודל ה- <i>attributes to face</i> שלנו. ייתכן כי נקל על התהליך הסזיפי הנ"ל עם מודל DiscoGAN ממומש .	לא נדרשים כרגע.
GUI	באמצעות tkinter , ספריית פייתון שבאמצעותה אפשר לפתח ממשקי GUI (Graphical User Interface).	לא נדרשים כרגע.
חלוקה ל- <i>utilities</i> : א. שליפת נקודתית של תמונה אחת או יותר לפי תכונות ניתנות (בניית המודל בשלבים)	שפת פייתון, ומגוון פונקציות ו- <i>array slicing</i> מתוך ספריית <i>pandas</i> (אופן המימוש עצמו גמיש, ועדיין לא נחרט באבן)	לא נדרשים כרגע.
חלוקה ל- <i>utilities</i> : ב. פירוק המודל הגדול למודלים קטנים שמזהים תכונה אחת או יותר, ואימונם (בניית המודל בשלבים)	הרשת תיבנה באמצעות הכלים שמוצעים לנו ב-PyTorch, בשפת פייתון. הרשת תקבל את מערך ה-NumPy התלת מימדי ואז באמצעות שתי <i>hidden layers</i> תבצע <i>pattern recognition</i> לפיצ'ר נקודתי / מספר פיצ'רים.	כרטיס מסך טוב על מנת לקבל תוצאות טובות מהמודל מפאת הדרישה ה- <i>computationally expensive</i> של למידת מכונה. משום שכרטיס מסך טוב לא זמין לנו, נשתמש ב-Google Colab בשביל ביסוס אימון המודל.
חלוקה ל- <i>utilities</i> : ג. חיבור המודלים הקטנים למודל אחד גדול (בניית המודל בשלבים)	ניעזר במדריך הזה על מנת לבצע את ה- <i>transfer learning</i> . נשתמש בשפת פייתון.	לא נדרשים כרגע.

תרשים זרימה



מבנה בסיס נתונים

א. מאגר תמונות:

- הדאטהסט שלנו מכיל מאגר של 202599 תמונות מסוג `.jpg`, כל אחת מהן ברזולוציה של `178x218` פיקסלים.

ב. טבלת תכונות:

קובץ `csv`. ששמו `list_attr_celeba.csv`. מימדיו הם 202600 שורות על 41 טורים.

- שדה `image_id` - מסוג `String`, מכיל את שמות הקבצים של כל אחת מהתמונות במאגר.
- שדות `2-40` - כל מה שבא אחרי `image_id`: מסוג `Integer`, מכילים בתוכם תיוג בינארי (1 או -1, קיים או לא קיים) לכל אחת מתכונות הפרצוף האנושי המתאימות לתמונה מאותה שורה בטבלה. לדוגמה, התמונה הראשונה, `000001`, היא בעלת שיער (התכונה `Bald` מסומנת ב-1), היא נקבה, היא צעירה וכו'.

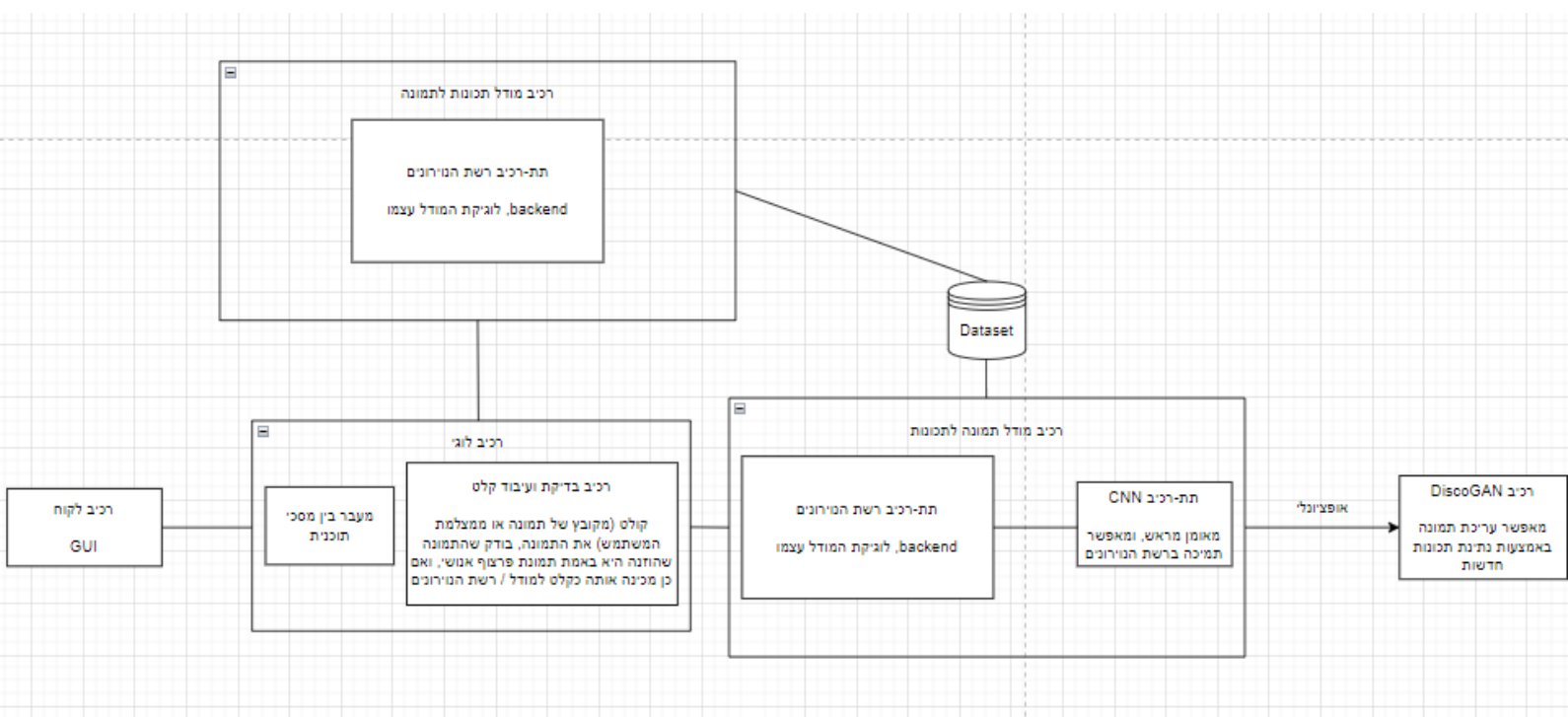
פרק 3: ארכיטקטורה

הסבר על המסמך

ברכותינו! סיימתם לתכנן את עיקר "הסיפור העסקי" של הפרויקט, עוד רגע יגיע הזמן להתחיל לממש... המסמכים עליהם עבדתם עד כה עסקו בתוכן של הפרויקט שלכם - מה הוא עושה. כעת הגיע הזמן להבין ולתכנן **כיצד** המימוש יתבצע.

מטרת מסמך הארכיטקטורה היא להבנות את השלד העיקרי למימוש הפרויקט שלכם - האופן שבו הוא יבנה, הרכיבים השונים, חלוקת האחריות והקשרים ביניהם, ועוד. תכנון נכון של הפרויקט, באמצעות מסמך זה, יסייע לכם להמנע מטעויות בזמן הפיתוח, וגם יאפשר לחלק את העבודה על הפרויקט בין חברי הצוות וכן לחלקה למספר ספרינטים/איטרציות - כאשר בכל איטרציה תתמקדו במספר מצומצם של רכיבים או חלקים מתוך כלל הפרויקט. במסמך זה נתעסק בתמונה הגדולה, ולאחר מכן מדי 3 שבועות תפתחו ב"ספרינט" חלק ספציפי מתוך הפרויקט.

מבט על



הסבר:

- **רכיב לקוח (GUI)**
- רכיב שנותן למשתמש גישה נוחה, ויזואלית וגמישה לכלל הדברים שהפרויקט מציע.
- נעשה באמצעות [tkinter](#), ספריית פייתון שבאמצעותה אפשר לפתח ממשקי GUI

● רכיב לוגי

- רכיב העוזר למשתמש להתנהל עם התוכנה ומבצע את ההכנות עבור הפיצור אותו בחר.
- תת רכיב בדיקת ועיבוד קלט
 - רכיב האחראי על קבלת קלט תמונה דרך קובץ ממחשב המשתמש או לכידת תמונה מהמשתמש שלו.
 - בנוסף, הרכיב יבצע *face recognition, noise reduction* ו-*cropping* לתמונת הקלט.
 - תמונת הקלט תועבר לרכיב המודל תכונות לתמונה.
 - קליטת התמונה תתבצע דרך ה-GUI עם *image upload* או באמצעות *opencv*, והעיבוד שלה באמצעות [מודל face recognition](#) - בפייתון.
- תת רכיב ניווט בין מסכים
 - נכלל בתוך ה-GUI; רכיב שאחראי על ניווטו של המשתמש למסכי הקלט/פלט המתאימים לכל פיצ'ר.

● Dataset

- קובץ *csv*. שבאמצעותו מאומנים שתי רשתות הנורונים.
- שליפת הנתונים מהקובץ מתבצעת באמצעות ספריית *pandas* שבפייתון.

● רכיב מודל תמונה לתכונות

- רכיב האחראי על קבלת תמונה ופליטת תכונות הפרצוף האנושיות שלה.
- המודל יהיה מבוסס רשת נורונים, ויוכן בפייתון עם הכלים שלספריית *PyTorch* יש להציע.
- הקלט יהיה דרך רכיב הבדיקת ועיבוד קלט.
- תת רכיב *CNN*
 - תת רכיב האחראי על "תמיכה" ברשת הנורונים.
 - יהיה מבוסס על ארכיטקטורת [CNN \(Convolutional Neural Network\)](#).
- תת רכיב רשת הנורונים
 - תת רכיב האחראי על לוגיקת רשת הנורונים, מקבלת מערך פיקסלים ועד לזיהוי תכונה.
 - יבנה באמצעות *transfer learning* של מספר מודלים קטנים, שכל אחד מהם מזהה תכונה אחרת.

● רכיב מודל תכונות לתמונה

- רכיב האחראי על קבלת תכונות פרצוף אנושיות ופליטת תמונה המתבססת על התכונות הנ"ל, בתוספת *noise* רנדומלי בשביל תוצאות שונות כל פעם.
- המודל יהיה מבוסס רשת נורונים וארכיטקטורת *GAN*, ויוכן בפייתון עם הכלים שלספריית *PyTorch* יש להציע.
- הקלט יהיה דרך *dropdown menu* שמציג 40 תכונות קלט אפשריות.
- רכיב רשת הנורונים
 - רכיב האחראי על לוגיקת רשת הנורונים, מקבלת תכונות ועד סינתוז תמונה.
 - הרשת תיבנה באמצעות [הכלים שמוצעים לנו ב-PyTorch](#), כאשר הארכיטקטורה שלה תותאם לרשת בתיאור התיאורטי שנמצא ב[דף המדעי שעוסק בתחום זה](#), על סמך התוצאות האמינות [שמודל המבוסס על דף זה](#)

נותן, במידה ונחליט לפתח את רשת הנוירונים מאחורי הפיצ'ר בכוחות עצמנו ולא באמצעות המודל המוכן שמובא לעיל.

• רכיב DiscoGAN

- הרכיב הינו קונספט ואף [מודל ממומש](#) של רשת נוירונים המיישמת את ארכיטקטורת ה-[GAN \(Generative Adversarial Network\)](#), המאפשר לנו לקחת תמונה, לשנות בה תכונה אחת או יותר, ואז להנפיק תמונה חדשה עם העריכות שהזנו.

עיצוב הנתונים ויישויות מידע

המידע המועבר במערכת הוא בעיקר ליסטים בפייתון (מערבים) שמאחסנים מגוון סוגי מידע, מ"מספרים אמיתיים" ועד יצוגים מספריים ו-*metadata* של תמונות. המידע הנשמר במערכת הוא אובייקטים של תמונות המשתמש מעלה או מצלם, לטווח זמן זמני

מידע העובר במערכת:

- מערך בינארי (1 או 0-) של תכונות המשתמש המבוקשות בחלק התכונות -> תמונה
- הסתברויות הביטחון (עד כמה תכונת פרצוף אנושי מסויימת סבירה להיות בתוך התמונה, נע בין 0 ל-1) לגבי כל אחת מהתכונות בתמונה בחלק התמונה -> תכונות
- מערך ה-NumPy שאנו נעזרים בו כדי להפוך תמונה למערך פיקסלים
- תוצאות קריאת פונקציית העזר בפיצ'ר 14, שהוא מערך אובייקטי PIL
- עוד משהו שכנראה שכחתי

מידע הנשמר במערכת:

- תמונות שהמשתמש מעלה או מצלם, לצורך עיבוד. הן נשמרות זמנית כאובייקט PIL, שכן אין לנו צורך או סיבה לשמור אותן לטווח ארוך בהינתן הפיצ'רים שפירטנו

טכנולוגיות עיקריות

- רכיב הלקוח (GUI) יבנה ב-tkinter של שפת פייתון, וירופץ על מחשב המשתמש, כלומר לוקאלי.
- הרכיב הלוגי/בדיקת ועיבוד הקלט יבנה בשפת פייתון ויכלול גם שימוש ב-opencv, בעיקר עקב נוחות השפה והתיעוד האדיר שיושב מאחוריה לכל דבר קטן שנצטרך, הן עבור מגבלות השפה בשפת תכנות כשרה והן עבור נושא הפרוייקט עצמו.
- רכיב רשת הנוירונים (תמונה לתכונות + תכונות לתמונה) - יבנה בשפת פייתון ובאמצעות מגוון ספריות למידת המכונה NumPy, PyTorch ו-Pandas, מדעי המידע, ושלשפה ולקהילה שלה יש להציע. בחרנו בפייתון כי ברמת התעשייה ובכלל בהיסטוריית קיומה, פייתון הוכיחה את עצמה כ-go-to language עבור הנושאים שהוצגו לעיל. אנו אוהבים פייתון ומרגישים שיש לנו את הידע למימוש הפרוייקט אך יש ספריות שנצטרך ללמוד בצורה מעט יותר עמוקה ממה שלמדנו עד כה.
- רכיב DiscoGAN יהווה אפשרות עבור המשתמש לערוך תכונה בתמונה ולהפיק תמונה חדשה, נשתמש בפייתון.

התאמה לאפיון

פצ'ר	רכיבים רלוונטים
ניתוח תכונות קונקרטיות ואבסטרקטיות של פרצוף אנושי מתמונה	לקוח (GUI) < רכיב לוגי < רכיב בדיקת ועיבוד קלט < רכיב מודל תמונה לתכונות < רכיב לוגי < לקוח (GUI)
ייצור תמונה מתכונות קונקרטיות ואבסטרקטיות של פרצוף אנושי	לקוח (GUI) < רכיב לוגי < רכיב בדיקת ועיבוד קלט < רכיב מודל תכונות לתמונה < רכיב לוגי < לקוח (GUI)
ניתוח תכונות קונקרטיות ואבסטרקטיות של פרצוף אנושי ממצלמה	לקוח (GUI) < רכיב לוגי < רכיב בדיקת ועיבוד קלט < רכיב מודל תמונה לתכונות < רכיב לוגי < לקוח (GUI)
יצירת דאטהפריים	<code>pandas</code> של <code>read_csv</code> Dataset
פירוק תמונה למערך פיקסלים	תמונה < מערך NumPy תלת מימדי
רשת נוירונים - תמונה לתכונות	מערך NumPy תלת מימדי < רשת הנוירונים < <code>pattern recognition</code> < תוצאות הסתברותיות
רשת נוירונים - תכונות לתמונה	תכונות כרשימה עם ערכים בינאריים < רשת הנוירונים < סינתוז תמונה באמצעות ארכיטקטורת GAN < תמונה
רשת נוירונים - אימון	האכלת תמונות מ- <code>training segment</code> לרשת נוירונים <code>fully connected</code> < ביצוע <code>hyperparameter tweaking</code> עם ה- <code>validation segment</code> < ביצוע בדיקת איכות עם ה- <code>test segment</code> ושיקולים אנושיים
רשת נוירונים - <code>benchmarking</code>	מודל רשת נוירונים מוכן < שירות אונליין או מקבץ פונקציות בפייתון < תוצאות (חשופות רק לנו, המפתחים)
לכידת תמונה ממצלמת המשתמש והתאמתה	לקוח (GUI) < רכיב לוגי < לכידת תמונה באמצעות <code>opencv</code> < שימוש במודל <code>face recognition</code> על מנת לחתוך את התמונה
ניקוי רעשים מתמונה	לקוח (GUI) < רכיב לוגי < שימוש במודל <code>noise reduction</code>
עריכת תמונה	לקוח (GUI) < רכיב לוגי < רכיב בדיקת ועיבוד קלט < רכיב מודל תמונה לתכונות < רכיב <code>DiscoGAN</code> < רכיב לוגי < לקוח (GUI)

GUI	ספריית tkinter
חלוקה ל- <i>utilities</i> : א. שליפת נקודתית של תמונה אחת או יותר לפי תכונות ניתנות (בניית המודל בשלבים)	בקשה פנימית (בתוך הקוד) מהפונקציה < כניסה לדאטהפריים < חיפוש שמות כל התמונות שלהן מותאמת תכונה אחת או יותר, והפיכתן לאובייקטי PIL ששומרים מידע פיזי של תמונה < יצירת רשימת תמונות
חלוקה ל- <i>utilities</i> : ב. פירוק המודל הגדול למודלים קטנים שמזהים תכונה אחת או יותר, ואימונם (בניית המודל בשלבים)	רשימת התמונות שמשווייכות לתכונה אחת או יותר < האכלת התמונות לרשת נוירונים הזהה בארכיטקטורתה לזאת שאנחנו משתמשים בה ב"רשת נוירונים - תמונה לתכונות" < חזרה על התהליך הנ"ל לכלל התכונות
חלוקה ל- <i>utilities</i> : ג. חיבור המודלים הקטנים למודל אחד גדול (בניית המודל בשלבים)	CNN מודל + <i>transfer learning</i>

פרק 4: תוכנית עבודה

הסבר על המסמך:

לאחר שמסמך הארכיטקטורה שלכם הושלם ואושר, יש לפרוט אותו לשלבים שונים על גבי ציר הזמן. מצפים לכם כמה חודשים של פיתוח, ולכן יש לתכנן מראש את סדר שלבי הפיתוח בצורה חכמה והולמת. לא כל הדרישות נולדו שוות, וגם לא תוכלו לפתח את כולן בבת-אחת. לכן יש להחליט מהי העדיפות של כל דרישה, ובהתאם לכך מה יהיה סדר הפיתוח שלהן.

בעת מתן עדיפות לדרישות, כלומר איזו דרישה תפותח קודם, השיקולים צריכים להיות בראש ובראשונה של תוכן - ולא של אילוצים טכניים. כלומר יש להתחיל מהרכיבים והדרישות שמהווים את עיקרי המהות של הפרויקט (הליבה הטכנולוגית), ולאחר מכן לבנות סביבם את יתר המערכת. זוהי גישה של **מוצר מינימלי מתפקד** - כך שבסיומה של כל איטרציה ניתן יהיה להדגים פרויקט רץ ומתפקד (גם אם באופן מאוד חלקי ולא הכי יציב או מתוחכם). כמובן שגם כל איטרציה אמורה להציג פרויקט רץ אשר מתפקד יותר טוב או עם יותר פיצ'רים מאשר זה שהוצג באיטרציה(ספרינט) הקודמת.

מתן עדיפות לפיצ'רים, חלוקה לשלבים מסודרים

תיעוד:

- 1. יצירת דאטהפריים
- 2. חלוקה ל-utilities - שליפה מהדאטהסט לפי תכונה/ות ספציפית/ות
- 3. פירוק תמונה למערך פיקסלים
- 4. חלוקה ל-utilities - בניית מודלים לפי מסווגים יחידים
- 5. אימון כל מודל מינימליסטי "קטן" באמצעות התמונות שנשלפו מהדאטהסט
- 6. חלוקה ל-utilities - חיבור המודלים למודל ראשי "גדול"
- 7. לכידת תמונה ממצלמת המשתמש
- 8. ניקוי רעשים, זיהוי פנים, cropping
- 9. תכונות לתמונה
- 10. אפשור עריכת תמונה
- 11. GUI

מס' פיצ'ר	משימה	תלויות תשתית (האם המשימה תלויה במשימה אחרת)	נימוק והערות	רכיב רלוונטי
1	יצירת דאטהפריים	הפיצ'ר אינו תלוי במשימות אחרות	ניצור באמצעות read_csv של ספריית pandas	Dataset
2	שליפה לפי תכונה/ות (יצירת utilities)	הפיצ'ר תלוי במשימה הקודמת	ניצור פונקציית סינון אשר תדע לגשת לדאטהסט ולשלוף את התמונות הנחוצות	רכיב מודל תמונה לתכונות
3	פירוק תמונה למערך פיקסלים	הפיצ'ר אינו תלוי במשימות אחרות	לאחר שמירת התמונות נפרק כל תמונה למערך פיקסלים באמצעות ספריית NumPy כדי שתהיה מוכנה כקלט לרשת	רכיב לוגי

	הנוירונים			
4	יצירת מודלים לפי מסווגים יחידים	הפיצ'ר אינו תלוי במשימות אחרות	ניצור מודלים נפרדים אשר "יתרכזו" בזיהוי מידע לפי מסווג יחיד	רכיב מודל תמונה לתכונות
5	חלוקה ל-batches	הפיצ'ר תלוי במשימה "שליפה לפי תכונה/ות"	כשאנחנו מייבאים את הדאטהסט שלנו מתוך PyTorch אנו מקבלים גם על הדרך חלוקה ל-batches. נעביר כל batch לפונקציית הסיווג שלנו, כדי לקבל רק תמונות שמכילות תכונה/ות ספציפית/ות כרצוננו.	רכיב מודל תמונה לתכונות
5	אימון כל מודל נפרד	הפיצ'ר תלוי במשימה הקודמת	כל מודל "קטן" יאומן באופן נפרד על batches של תמונות רלוונטיות מהדאטהסט	רכיב מודל תמונה לתכונות
5	חישוב ה-loss הממוצע עבור כל batch במודל קטן ספציפי	הפיצ'ר תלוי במשימה הקודמת	באמצעות הפלט על כל תמונה נחשב loss. לאחר סיום batch מסוים נוכל לחשב את ה-loss הממוצע באמצעות פונקציית MSE (Mean Square Error) ולפיו לעדכן את המודל	רכיב מודל תמונה לתכונות
5	אימון מחדש של המודל הקטן	הפיצ'ר תלוי במשימה הקודמת	לאחר קבלת ה-loss הממוצע, יתבצע backward propagation שיאפשר שיפור סיווג התכונה/ות הספציפי/ות בתמונה באמצעות שינוי ה-weights. נשלוח batch חדש ונראה את השיפור שהתקבל לנו	רכיב מודל תמונה לתכונות
6	חיבור הקטנים למודל כלל מסווגים	הפיצ'ר תלוי בכך שהמודלים הקטנים מוכנים ומאומנים היטב	את כלל המודלים המבצעים קלסיפיקציה לפי מסווג יחיד נאחד למודל אחד אשר בצע קלסיפיקציה לפי כלל המסווגים	רכיב מודל תמונה לתכונות
7	לכידת ממצלמה	הפיצ'ר תלוי בכך שהמשתמש בחר באופציה זו	באמצעות ספריית opencv נלכוד תמונה מהמצלמה של המשתמש	רכיב לוגי
8	ניקוי רעשים	הפיצ'ר תלוי בכך שהמשתמש בחר ללכוד תמונה ממצלמה	לתמונה נבצע ניקוי רעשים באמצעות מודל noise reduction מאומן מראש וכך נוכל להפוך את התמונה לטובה יותר, מבחינת הנראות ומבחינת העיבוד העתיד לבוא לו	רכיב לוגי
8	זיהוי פנים ו-cropping	הפיצ'ר תלוי בכך שהמשתמש בחר ללכוד תמונה ממצלמה	באמצעות מודל face recognition מאומן מראש ו-opencv נוכל לחתוך את התמונה בצורה שתתאים לרשת על ידי זיהוי הפנים	רכיב לוגי
9	אימון generator	הפיצ'ר אינו תלוי במשימות אחרות	יצור תמונות "מזויפות" שינסו להתקיל את ה-discriminator. כל פעם יקבל שדרוג עד שיצור תמונה מספיק אמיתית שתתעלה	רכיב מודל תמונות לתמונה

	על יכולותיו של ה-discriminator - זהו בעצם מודל התכונות לתמונה שלנו, רק ללא תכונות ספציפיות שנבחרו מהמשתמש			
9	אימון discriminator	הפיצ'ר אינו תלוי במשימות אחרות	"ילחם" ב-generator, בכך שהוא ינסה לסווג את התמונות המזויפות שהוא מייצר מהאמיתיות שבדאטהסט	רכיב מודל תכונות לתמונה
9	תכונות לתמונה	הפיצ'ר תלוי בכך שהמשתמש בחר באופציה של הפקת תמונה מתכונות, ובטיבתם של מודלי ה-generator discriminator שנוצרו מקודם	יהווה מעין מקשר בין ה-generator וה-discriminator. תכליתו היא הקניית התכונות הנבחרות ל-generator המאומן שלנו, כך שיצור תמונות לא רק לפי noise רנדומלי כפי שהיה באימון (מה שאפשר ויאפשר יצירת תמונות שונות מראה כל פעם) אלא גם באמצעות תכונות שהמשתמש יבחר	רכיב מודל תכונות לתמונה
10	אפשר עריכת תמונה	הפיצ'ר תלוי בכך שהמשתמש בחר לערוך תמונה כלשהי	נאפשר למשתמש לבצע שינויים בתכונות התמונה אשר נלכדה מהמצלמה	רכיב DiscoGAN
10	שימוש במודל DiscoGAN	הפיצ'ר תלוי בכך שהמשתמש בחר לערוך תמונה כלשהי	שימוש במודל DiscoGAN, שכבר מוכן מראש, אשר יודע לקחת את התמונה המקורית ולהפיק תמונה חדשה דומה בה ההבדלים המשמעותיים הם התכונות אותן בחר המשתמש לשנות	רכיב DiscoGAN
11	GUI	הפיצ'ר תלוי ב-accessibility של כל הפיצ'רים העיקריים (תמונה לתכונות, תמונה לתכונות לפי תמונה מהמצלמה, תכונות לתמונה, עריכת תמונה), מה שכבר אמור להיות מובטח מראש אם עשינו הכל כהלכה	יאפשר גישה נוחה לכלל הפיצ'רים שלפרוייקט יהיה להציע	רכיב לקוח

חלוקה לאיטרציות (ספרינטים)

איטרציה 1:

- יצירת דאטהפריים (פיצ'ר 1)
- שליפה לפי תכונה/ות (פיצ'ר 2)
- פירוק תמונה למערך פיקסלים (פיצ'ר 3)
- יצירת מודלים לפי מסווגים (פיצ'ר 4)
- חלוקה ל-batches (פיצ'ר 5)

- אימון כל מודל נפרד (פיצ'ר 5)
- חישוב ה-loss הממוצע עבור כל batch במודל קטן ספציפי (פיצ'ר 5)
- אימון מחדש של המודל הקטן (פיצ'ר 5)

איטרציה 2:

- חיבור המודלים הקטנים למודל כלל מסווגים (פיצ'ר 6)
- לכידת תמונה ממצלמה (פיצ'ר 7)
- ניקוי רעשים (פיצ'ר 8)
- זיהוי פנים ו-cropping (פיצ'ר 8)

איטרציה 3:

- אימון generator (פיצ'ר 9)
- אימון discriminator (פיצ'ר 9)
- תכונות לתמונה (פיצ'ר 9)

איטרציה 4:

- אפשרור עריכת תמונה
- שימוש במודל DiscoGAN

איטרציה 5:

- GUI

פרק 5: עיצוב - מדי איטרציה

זהו השלב האחרון במסמך הפרויקט שלכם - אך תרחיבו אותו מדי איטרציה, בתחילת כל איטרציה. לאחר שכבר החלטתם אילו פיצ'רים מפותחים בכל איטרציה, מטרת המסמך היא לתכנן מראש את הקוד שתפתחו עבור האיטרציה הקרובה. התוכן של פרק זה עשוי להיות שונה עבור כל איטרציה, בהתאם לדגשים ולנושאים המובילים בה. יש לכלול תכנון מפורט עבור כל איטרציה, בצירוף שרטוטים והסברים.

איטרציה 1 - מתמקדת בהכנת תשתית להמשך הפרויקט

החזון שלנו לסיום הספרינט:

בסיום הספרינט הנ"ל, יהיו לנו מספר מודלים מאומנים עם אחוזי הצלחה גבוהים כאשר כל אחד ידע לסווג תכונה אחת או יותר מתוך תמונה.

חלוקת עבודה ולוח זמנים:

בספרינט זה נעבוד לפי שלבים:

שלב 1 - שליפת התמונות והכנתן כקלט לרשתות הנוירונים - את שלב זה נחלק בינינו לפי המשימות שציננו.

שלב 2 - יצירת המודלים "הקטנים" ואימונם - נחלק את זה כך שכל חבר צוות יעבוד על מספר זהה של מודלים ויבצע עבור כל אחד את המשימות הנדרשות.
*** ייתכן כי נעבוד תחילה על מודל אחד או שניים יחד כדי לקבל התאמה ובסיס זהה.**

במהלך כל שבוע נעדכן אחד את השני בהתקדמות.

תקציר, הסבר ותוצר:

שליפת התמונות תבצע ע"י יצירת דאטהפריים שיאפשר לנו גישה נוחה לתיווגי התמונות, לפי הדאטהפריים נוכל לדעת אילו מהתמונות רלוונטיות ואת מה לשלוח לצורך מודל מסוים.
ניצור פונקציית סינון אשר תדע לשלוח מהדאטהסט את התמונות, ולבסוף נבין את התמונות כקלט לרשת הנוירונים על ידי הפיכתן למערך תלת-מימדי של פיקסלים.

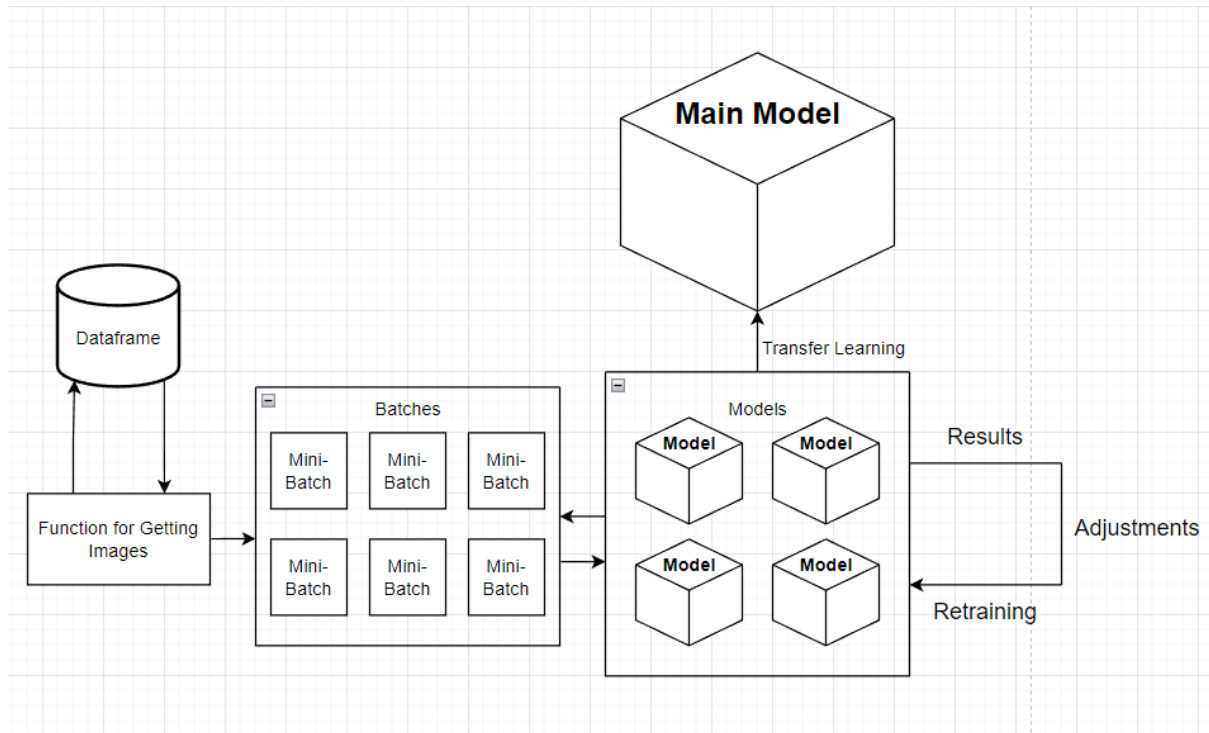
ניצור מספר מודלים (לפי מספר התיווגים של התמונות), כל אחד יידע "להתפקס" על איזור/תכונה ספציפית בתמונה ויתאמן לסווג לפי אותה תכונה.

נחלק את סט האימון ל-*mini-batches*, כאשר כל אחד כזה יכיל מספר קבוע של תמונות שכל אחת תינתן למודל לצורך האימון.

עבור כל מודל נבצע *transfer learning* כך שאימון המודל ייקח פחות זמן וגם ביצועיו יהיו טובים יותר.

כל מודל יחשב $loss$ עבור כל תמונה (באמצעות פונקציית MSE), ובסוף כל $batch$ ייקח את ה- $loss$ הממוצע ולפיו יעדכן את המודל ע"י תהליך $backward-propagation$.


המודל ישנה את ה- $weights$ בהתאם, וכך נוכל לאפשר אימון מחדש.




תכולות טכנולוגיות:

את יצירת הדאטהפריים נעשה באמצעות הפונקציה `read_csv` של ספריית `pandas`.
פירוק התמונה למערך פיקסלים יתבצע באמצעות ספריית `NumPy`.
קבלת ושמירת תמונות באמצעות שמות הקובץ שלהן תתבצע עם ספריית `PIL` ועם `list` בפייתון.
את בניית ואימון המודלים נבצע באמצעות הספרייה `PyTorch`.

ניצור מחלקה אחת שתהיה אחראית על מימוש רשת הנוירונים, ומחלקה אחרת שתיעזר בה ואז תשמור את המודל המוגמר לתוך משתנה.

 Net
- CNN Layers
- FCNN Layers
- Flattening
- ReLU

 Models & Helper Functions
- Model No. 1
...
- Model No. N
+ GetImages()
+ Train()
+ GetModels()

איטרציה 2 – מתמקדת בסיום מודל התמונה לתכונות והתחלת פיתוח פיצ'ר לכידת התמונה מהמצלמה לתכונות

החזון שלנו לסיום הספרינט:

בסיום הספרינט הנ"ל, יהיה לנו מודל מאומן עם אחוזי הצלחה גבוהים שידע לסווג את 40 התכונות המאופיינות בתמונה, שיתמוך בקבלת תכונות מתמונה הנלכדת ממצלמת המשתמש.

חלוקת עבודה ולוח זמנים:

חוץ מהחלק הראשון של הספרינט, אנו צפויים לסיים אותו מהר.

שלב 1 – אימון המודלים הקטנים.

שלב 2 – חיבור המודלים למודל אחד רב-מסווג.

שלב 3 – לכידת תמונה ממצלמה והתאמתה כקלט למודל הרב-מסווג ע"י ביצוע *cropping* וניקוי רעשים.

במהלך כל שבוע נעדכן אחד את השני בהתקדמות.

תקציר, הסבר ותוצר:

נאמן את שאר המודלים הקטנים ונחבר אותם למודל אחד שיידע לסווג תמונה ל-40 תכונות. ניישם את החיבור הזה ונסיים כך את מודל התמונה לתכונות, שיאפשר למשתמש לקחת כל תמונה ולקבל את תכונות הפרצוף האנושי המאופיינות באותה תמונה.

לאחר מכן, נתחיל לממש את פיצ'ר התמונה הנלכדת ממצלמה לתכונות. תחילה נתנסה עם לכידת תמונה *raw*, מה שאמור להיות יחסית קל וקצר.

אחר כך, ניקח את התמונה ה-*raw* ונעביר אותה תחילה דרך מנגנון שיקח את התמונה, יזהה שהיא אכן מכילה פרצוף אנושי ואז יעשה *crop* למימדי התמונה כך שתתפקס על הפרצוף. לאחר מכן נעביר את התמונה ה-*cropped* למנגנון אחר שיעשה לה *noise reduction*.

את התמונה הסופית הזו נעביר למודל התמונה לתכונות, וכך נסיים את הפיצ'ר העיקרי השני.

תכולות טכנולוגיות:

ניעזר בכלל הטכנולוגיות שהשתמשנו בהם עד כה בשביל גמירת מודל התמונה לתכונות. בשביל חלק לכידת התמונה מהמצלמה, ניעזר קודם ב-*opencv* כדי ללכוד את התמונה ה-*raw* מהמצלמה עצמה של המשתמש, ולאחר מכן באמצעות מודלי זיהוי פנים ו-*cropping* וסינון רעשים - *pretrained* - נקבל תמונה שניתן יהיה לעבוד איתה.

איטרציה 3 - מחקר ופיתוח של מודל GAN אשר מקבל מידע על פנים ומפיק ממנו תמונה

החזון שלנו לסיום הספרינט:

בסיום הספרינט יהיה לנו מודל GAN אשר מותאם לצורכי הפרויקט. המודל יהיה מאומן חלקית וניתן לשיפור, בעלי ביצועים סבירים.

חלוקת עבודה ולוח זמנים:

ספרינט זה עוסק בחלק גדול מאוד בפרויקט אשר דורש מחקר מעמיק והבנה, לכן נחקור במקביל על נושאים רלוונטיים. בנוסף נחלק כל שלב בבניית ואימון המודל כאשר לכל אחד תהיה עבודה על כל שלב כך שלא יוצרו פערים. נעדכן אחד את השני במחקר ובהתקדמות.

תקציר, הסבר ותוצר:

ראשית, נחקור את הנושא ונבין איך לממש מודל GAN הדומה לשלנו בקוד.

לאחר הבנה נוכל להתחיל לעבוד על יצירת ארכיטקטורת מודל GAN נכונה המתאימה לצרכינו אשר תחולק לשני חלקים:

מודל Generator: נבנה מחלקה עבור מודל ה-Generator אשר יידע לקבל מידע על פנים אנושיות ולהפיק ממנו מערך תלת מימדי אשר ייצג את התמונה הממחישה את התכונות.

מודל Discriminator: יקבל ממודל ה-Generator את התמונה שהופקה וישווה אותה עם תמונה אמיתית זאת לצורך אימונו כדי שיוכל להבחין בין תמונה אמיתית לבין תמונה אשר הופקה ממודל ממוחשב (Generator).

מכאן נקבל מודל GAN הבנוי משני מודלים אשר כרוכים אחד בשני לצורך אימון ושיפור, היודע לקבל מידע על פנים אנושיות ולהפיק תמונה ממנו.

תכולות טכנולוגיות:

עבור העבודה על המודלים נשתמש בספריית Pytorch אשר אנו כבר מנוסים בה.. לצורך הצגת התמונה אשר מתקבלת כמערך תלת מימדי ממודל ה-Generator ניעזר בספריית PIL ו-Numpy.

איטרציה 4 - שימוש במודל DiscoGAN

החזון שלנו לסיום הספרינט:

בסיום הספרינט תהיה לנו אינטגרציה למודל DiscoGAN שיאפשר למשתמש לשנות תווי פנים מהתוצאה שקיבל ממודל התמונה לתכונות.

חלוקת עבודה ולוח זמנים:

ספרינט זה משתמש במודל pretrained, לכן מחקר פחות נחוץ לנו אך יוכל לעזור לנו במידה ונתקע.

נחלק את האינטגרציה לשלבים וננסה לעבוד על המודל יחדיו הפעם מפאת חוסר הצורך לפירוק העבודה בין שני חברי הצוות.
נעדכן אחד את השני במחקר ובהתקדמות.

תקציר, הסבר ותוצר:

המודל כבר בא pretrained, לכן כל מה שנשאר לנו לעשות זה:

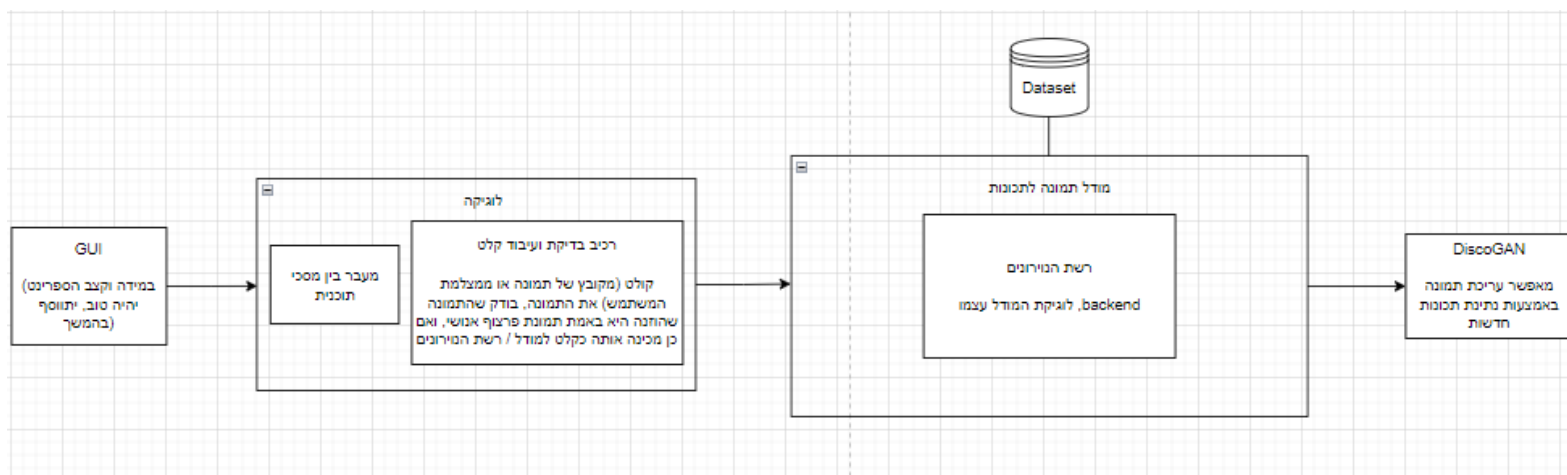
- לראות איך הוא עובד במצב העכשווי שלו (בדיקה)
- לנסות לשלב את התכונות הקיימות במודל התמונה לתכונות, ואת שאר התכונות שהדאטהסט שלנו מכיל עם היכולות של המודל

בעצם, למשתמש יהיה interface שלאחר שהוא קיבל את התכונות שלו ממודל התמונה לתכונות, תינתן לו האפשרות לשנות את התכונות שקיבל ואז לראות שינוי ממשי בתמונה המקורית.

לדוגמה: המשתמש הכניס תמונה של זכר, צעיר, עם שיער שחור. המודל יציג לו את התוצאות הנ"ל ולאחר מכן יתן לו את האפשרות לשנות את המין (זכר או נקבה), מידת הזקנה (צעיר או זקן) וצבע השיער (בלונדיני, חום, אפור או משהו אחר), ולאחר מכן יקבל את אותה התמונה אך עם השינוי שרצה שיתחולל.

תכולות טכנולוגיות:

עבור העבודה על המודל נשתמש בכלל הטכנולוגיות שהשתמשנו בהם כבר, בין אם זה בשפת פייתון או בשלל ספריות ה-machine learning/data science שלה שהניבו לנו שימוש קודם לכן.



איטרציה 5 - יצירת GUI

החזון שלנו לסיום הספרינט:

בסיום הספרינט יהיה לנו GUI שיאפשר לנו גישה לכלל ממשקי הפרויקט.

חלוקת עבודה ולוח זמנים:

תחילה ננסה לתכנן כיצד אמור להיראות ה-GUI מלכתחילה, ובנוסף איך לעבוד עם *tkinter* כדי להתחיל לפתח אחד. אחרי שנסיים עם החלק הזה, נוכל לעבוד על ה-GUI ביחד. לאחר פיתוח ה-*interface* עצמו נצטרך להבין גם כיצד לקשר בין ה-*frontend* שהוא ה-GUI לבין ה-*backend* שהם המודלים עצמם. נעדכן אחד את השני במחקר ובהתקדמות.

תקציר, הסבר ותוצר:

ניצור GUI עם מסך פתיחה שנותן גישה לכלל ממשקי הפרויקט. מבחינה אינטואיטיבית זה אמור להיות החלק הכי קצר של הפרויקט כי הוא כבר הוכרע כהכי פחות משמעותי. הקישור בין ה-GUI למודלים מעט מעומעם לנו בינתיים לכן זה כנראה החלק שיקח לנו הכי הרבה זמן בספרינט.

תכולות טכנולוגיות:

עבור העבודה על ה-GUI נשתמש בעיקר ב-*tkinter* כפי שהוזכר לעיל.