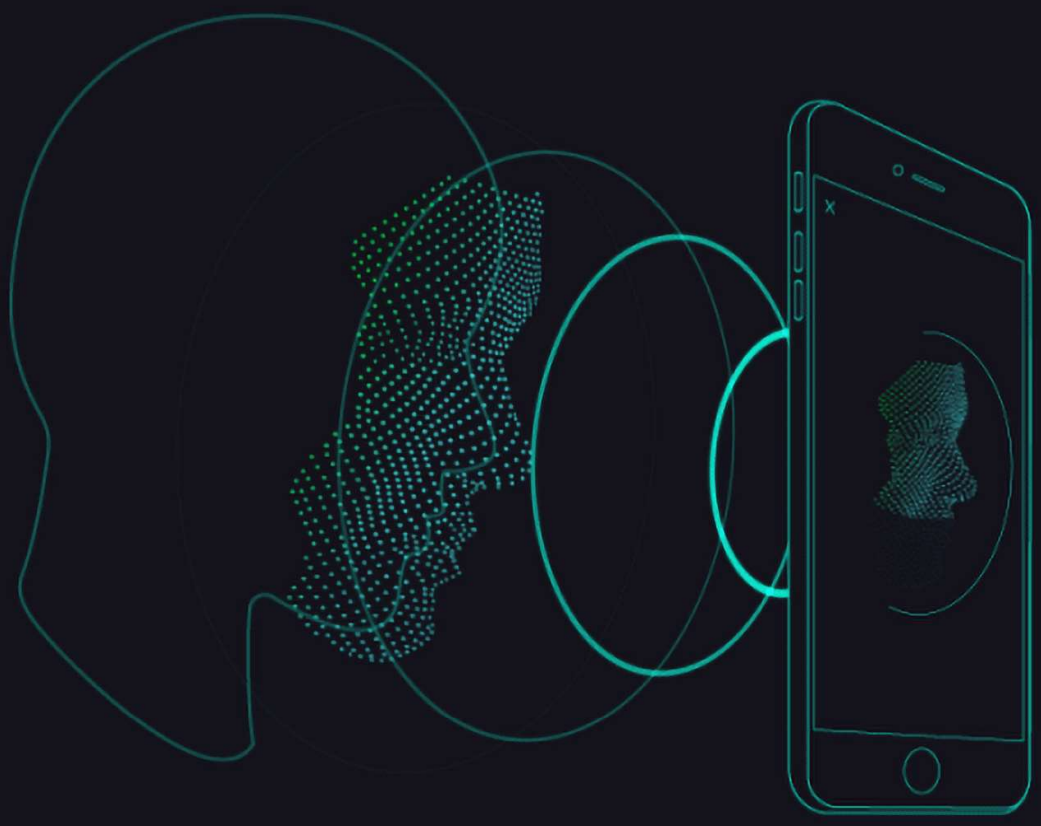
Face Detection and Recognition

פרויקט 5 יחידות במדעי המחשב

מנחה: יהודה אור



מגיש: מיכאל ברזובסקי 322382441

כיתה: יב'9 הכפר הירוק

תאריך: 30.5.2019

תוכן עניינים:

מבוא

על הפרויקט:

החלטתי בפרויקט שלי להתמקד בנושא מאוד "חם" ומעניין שתופס את עיניי רבים בתקופה האחרונה. הכוונה כמובן ל - Machine learning, למידת מכונה. ללמידת מכונה שימושים רבים, אך החלטתי להתמקד בתחום זיהוי הפנים, שכן בימינו טכנולוגיית זיהוי הפנים מתפתחת בקצב מסחרר והשימוש בה גדל בהתאם – בטלפונים, שדות תעופה ואפילו בכניסות למשרדים. מכיוון שבניית מודל זיהוי פנים מאפס היא משימה מאוד גדולה שאפילו חברות גדולות התקשו לבצע, החלטתי לעשות שימוש בספריית הקוד הפתוח TensorFlow שפותחה על ידי גוגל ומיועדת לפיתוח ובנייה של מודלים של למידת מכונה. בנוסף השתמשתי בארכיטקטורת FaceNet המבוססת על ארכיטקטורת GoogLeNet שגוגל פיתחו אשר מאפשרת זיהוי מהיר ומדויק של פנים אנושיות.

בעצם מה שהחלטתי לעשות בפרויקט זה מערכת לזיהוי פנים בזמן אמת מווידאו, ומתן תגובה כלשהי כאשר זוהה אדם מוכר - במקרה שלי החלטתי לפתוח דלת בתמונה. בעיקרון המטרה שלי הייתה ליצור מערכת אשר מזהה פנים בזמן אמת ונותנת את העובדה שהאדם זוהה ו/או שם המזוהה כפלט, כך שניתן לעשות עם התוצאה דברים רבים, כגון: פתיחת דלתות, פתיחת מכשירים נעולים, הודעות אישיות ועוד.

רפלקציה:

בתחילת השנה המנחה, יהודה אור, אמר כי כל אחד מאיתנו יכול ללמוד איתו בניית משחק ב #C או שיכול לעשות פרויקט עצמאי. החלטתי כי אני רוצה להתנסות ולעשות פרויקט על ידי למידה עצמאית ובנושא אחר. ברגע שבחרתי לעשות את הפרויקט על למידת מכונה הבנתי שעליי ללמוד הרבה תיאוריה לא פשוטה, אבל החוויה הייתה מאתגרת במידה סבירה ומעניינת מאוד ועל כן אני שמח שבחרתי בנושא הזה. אבל עוד בתחילת הפרויקט נתקלתי בקשיים רבים: התחלת עבודה ב TensorFlow ובייחוד עם כרטיס מסך (שיותר מהיר ממעבד רגיל אז חשבתי שאוכל לחסוך כך זמן), כפופה בהתקנות רבות ומהן נבעו תקלות רבות. או שהייתה חסרה איזו ספרייה או איזה תוסף שצריך בשביל שהמערכת תעבוד, או שהקבצים היו במקום הלא נכון והתכנה לא מצאה אותן, מה שבטוח זה שלקחו לי ימים שלמים בשביל לפתור את כל הבעיות בשביל רק להתחיל עם עבודת זיהוי הפנים. לקח לי כמה שעות לארגן את כל התמונות של הפנים שבחרתי ולסמן בהן איפה הפנים נמצאות, מה שהיה קצת מתסכל, מכיוון שחשבתי שאוכל לחסוך זמן זה על ידי מציאת מאגרי תמונות מסומנות מראש, אך לא הצלחתי למצוא כאלה. אחרי שלמדתי מסרטונים ומאמרים איך ללמד את המודל של זיהוי האובייקטים של גוגל לזהות משהו חדש, ניגשתי ללמד את המודל לזהות פנים בתמונה באמצעות התמונות שהכנתי. בכדי שהמודל יוכל להשתמש בתמונות

ובסימונים נאלצתי לכתוב קוד עזר אשר יגדיר בתוך הקבצים המתארים את מיקום הפנים בתמונה, את המיקום של הקובץ ושמו שיתאים לתמונה ולמיקום שלה, מכיוון שהתוכנה שבה השתמשתי לסימון התמונות בלבלה את הסדר והמיקום של הקבצים. מזה למדתי מה זה XML (מכיוון שזה סוג הקבצים שהתכנה שמרה) ולמדתי לכתוב סקריפטים ב Python שיכולים לבצע שינויים בקבצים במהירות ויעילות (במקום שאצטרך לעבור על יותר מ-500 קבצים ידנית ולא להתבלבל בתהליך).

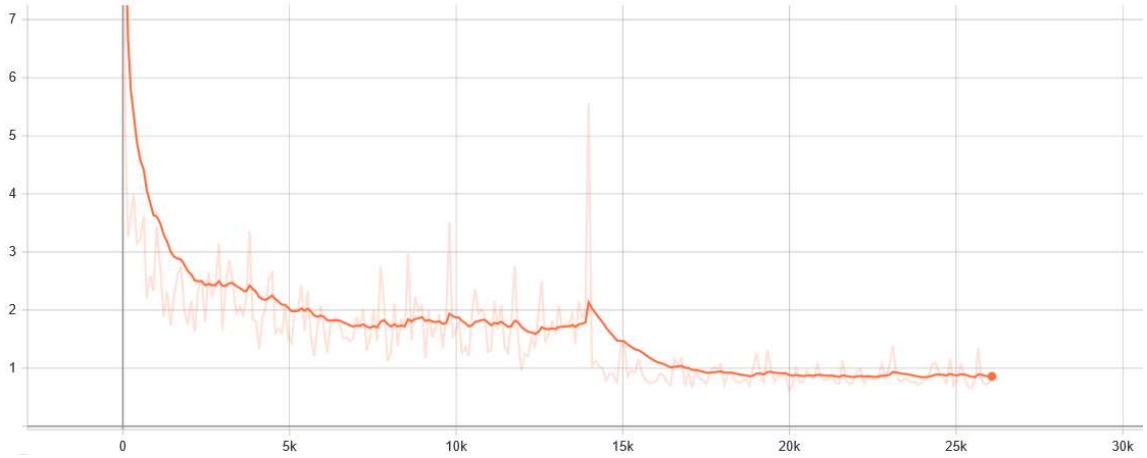
לצערי הדבר שהכי עוצר את הפרויקט זו העובדה שכל פעם שאני מפעיל את התוכנית מחדש אז ישנה טעינה ארוכה (2-3 דקות) של המשקלים של המודל לזיהוי הפנים, ועוד לא הצלחתי לגרום לכך שאוכל להוסיף אדם בלי צורך בכיבוי התוכנית הכללית.

בסופו של דבר אני גאה מהתוצר שיצרתי מכיוון שהוא עונה על כל הקריטריונים שהצבתי: הוא מזהה פנים, לאחר מכן מזהה אם הפנים מוכרות למערכת ואם כן אז למי הן שייכות, ובאותו זמן ממשיך לקרוא מידע מהמצלמה.

למדתי על עצמי שאם משהו מעניין ומסקרן אותי, אז אני יכול לשבת שעות על חומר מסובך ולא מובן וגם על בעיות ותקלות רק בשביל כדי להצליח בנושא. אני מאמין שאמשיך לעבוד על הפרויקט אחרי שאגיש אותו, וכבר יש לי כמה רעיונות שאנסה בקרוב, שכן יש עוד הרבה שיפורים ותיקונים אשר דרושים בשביל להפוך את הפרויקט למוצר מוגמר.

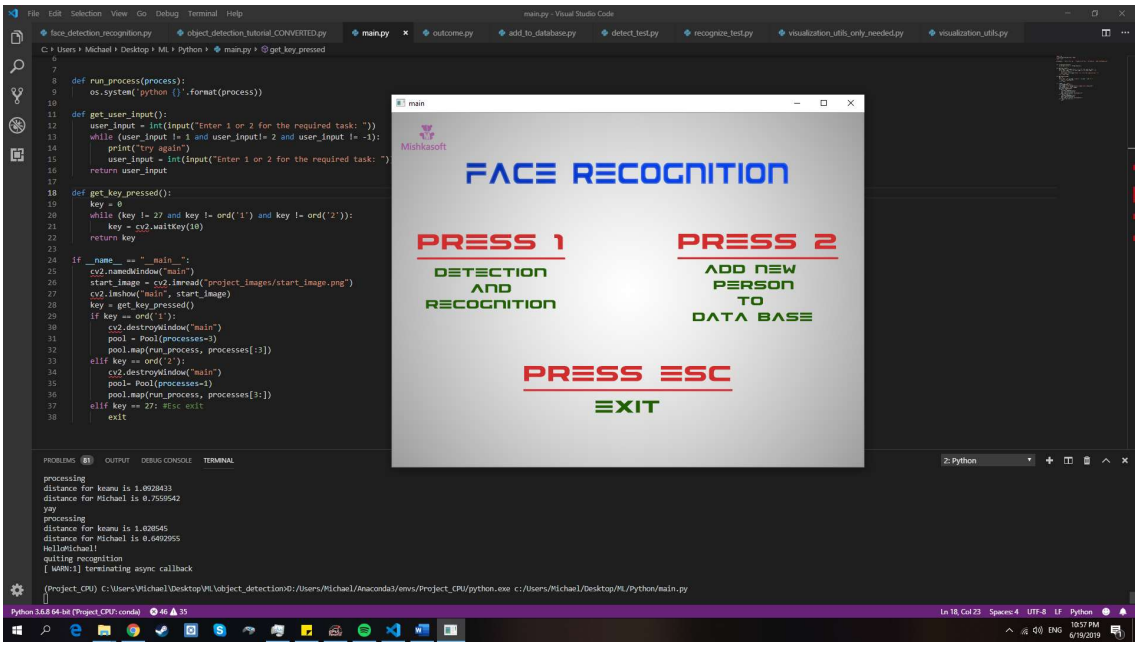
תמונות:

גרף פונקציית המחיר – Loss

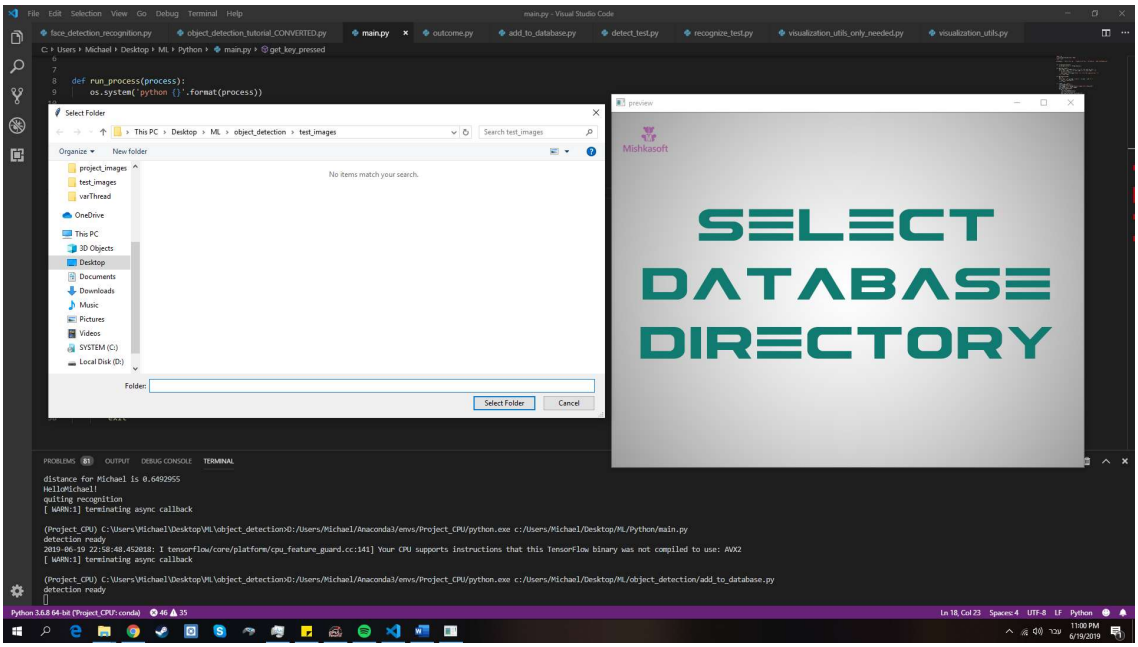


חלק מהתמונות שהלכו לאימון המודל:

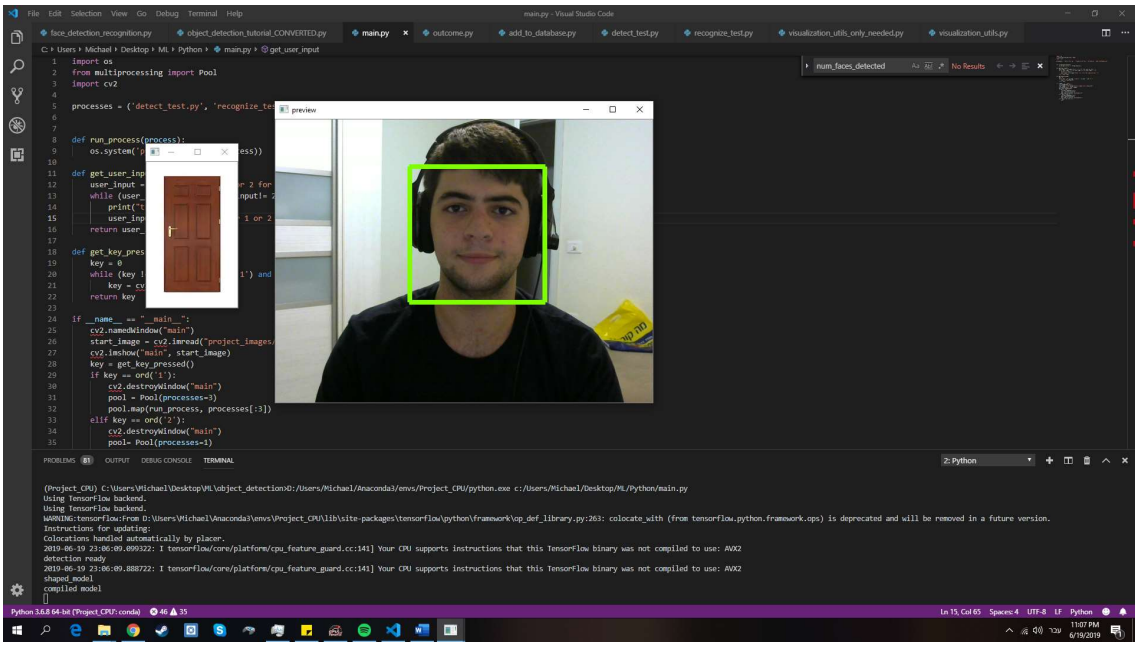
תמונה כאשר מריצים את התוכנית:



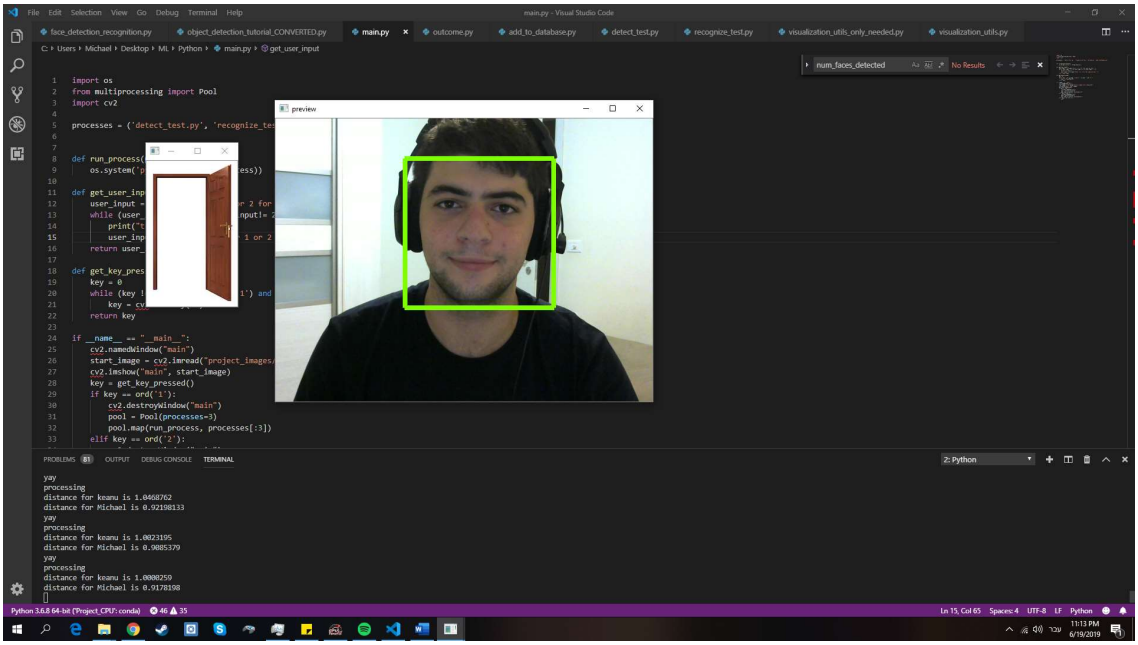
תמונה של מסך בחירת תיקיית מסד הנתונים:



תמונה כאשר אותרו פנים אך עוד לא זוהו (או שהם של אדם זר):



תמונה כאשר אותרו וזוהו פנים והדלת נפתחת:



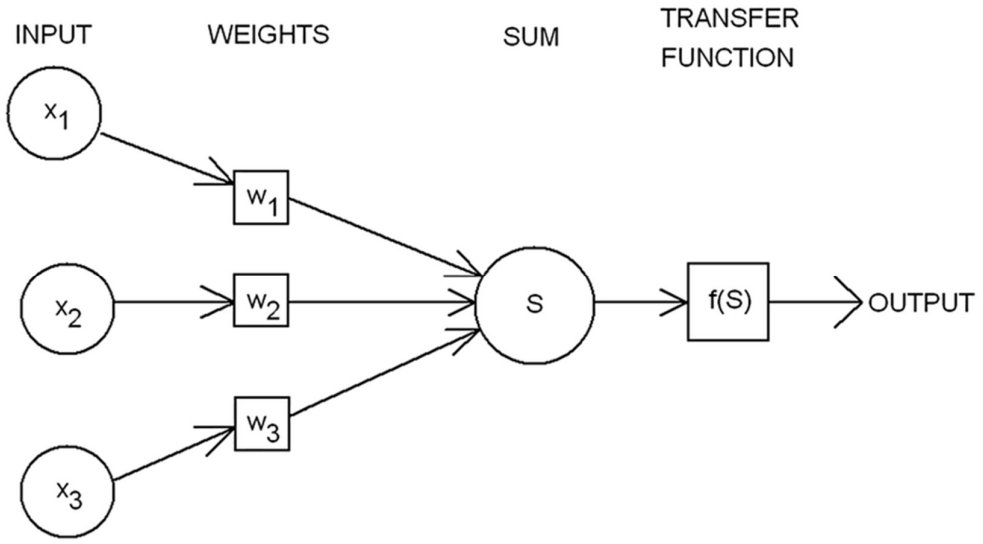
חלק תאורטי

למידת מכונה – Machine Learning

התחום עוסק בפיתוח אלגוריתמים המיועדים לאפשר למחשב ללמוד מתוך דוגמאות באופן בלתי תלוי בבן אדם, ופועל במגוון משימות חישוביות בהן התכנות הרגיל אינו מאפשר פיתרון .לדוגמה ניתן לקחת זיהוי אובייקט בתמונה – לבן אדם קל לזהות אם הוא רואה חתול או כלב בתמונה, אבל איך מחשב יכול לבצע את הזיהוי הזה? בתכנות רגיל היינו נותנים למחשב רשימת כללים שלפיהם היה עובר על התמונה ומנסה לזהות אובייקט. שיטה זו אינה טובה ואינה יעילה מכיוון שלתמונות שונות הכללים עלולים שלא להתאים. למידת מכונה לעומת זאת "מראה" למחשב תמונה ואת הזיהוי של האובייקט בתמונה, ומכך המחשב לומד חוקים וכללים בעצמו, אשר הוכחו כיעילים ומדויקים יותר מחוקים הנכתבו על ידי בני אדם.

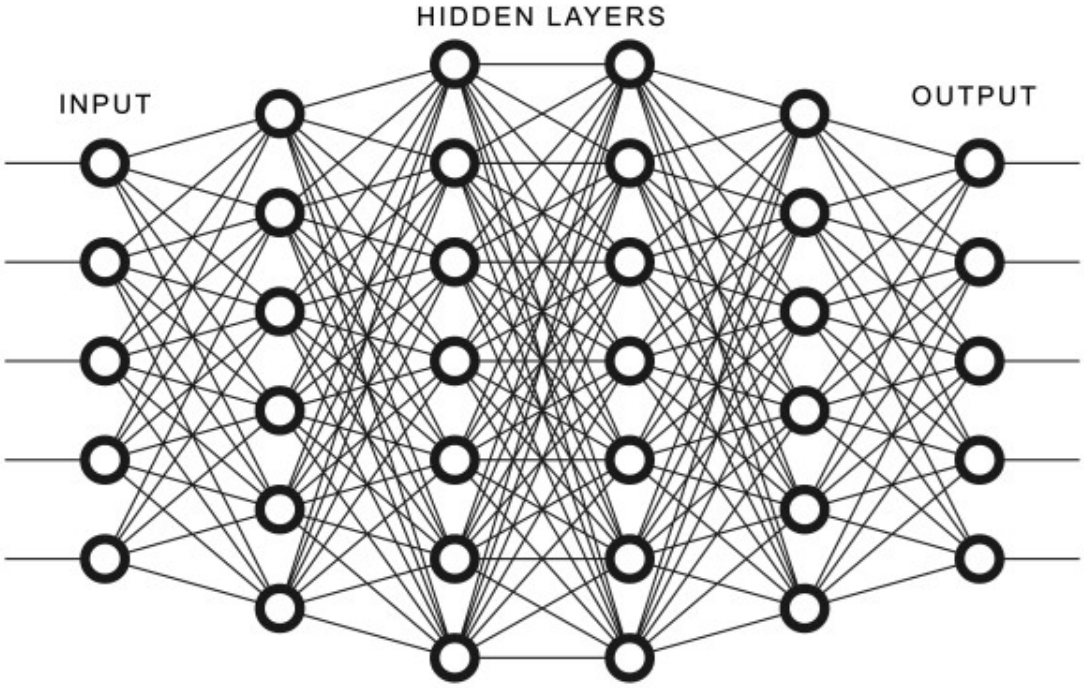
רשת עצבית מלאכותית - Artificial Neural Network

רשת עצבית מלאכותית או Artificial neural network היא מודל מתמטי חישובי שפותח בהשראת המוח האנושי. המוח האנושי מעבד מידע ומריץ תהליכים קוגניטיביים באמצעות רשתות עצביות טבעיות – דבר שמודל מתמטי זה מנסה לחקות במסגרת למידת מכונה.



כפי שניתן לראות בתרשים, נוירון ברשת מורכב מקלטים(INPUT) כאשר את הקלטים מכפילים ב"משקל(WEIGHTS)" – השפעה כלשהי לקלטים, ולאחר מכן הקלטים משולבים/ "נסכמים(SUM)" באמצעות פונקציה הנקבעת לפעולה הזו, ובסופו של דבר הסכום מועבר לפונקציית מעבר או הפעלה "TRANSFER/ACTIVATON FUNCTION" בכדי ליצור אי-לינאריות(המשמעות תוסבר בנושא Stochastic Gradient Descent).

רשת נוירונים מורכבת משכבות של נוירונים, כאשר השכבה הראשונה היא השכבה המקבלת את הקלט ומכניסה אותו לרשת, השכבה האחרונה היא המוציאה כפלט את התוצאה שהרשת יצרה מעיבוד הקלט, והשכבות שבינן נקראות "נסתרות" שכן לא פשוט לעקוב אחרי כל השפעה של נוירון בה על התוצאה הסופית. רשת נוירונים נראית כך:

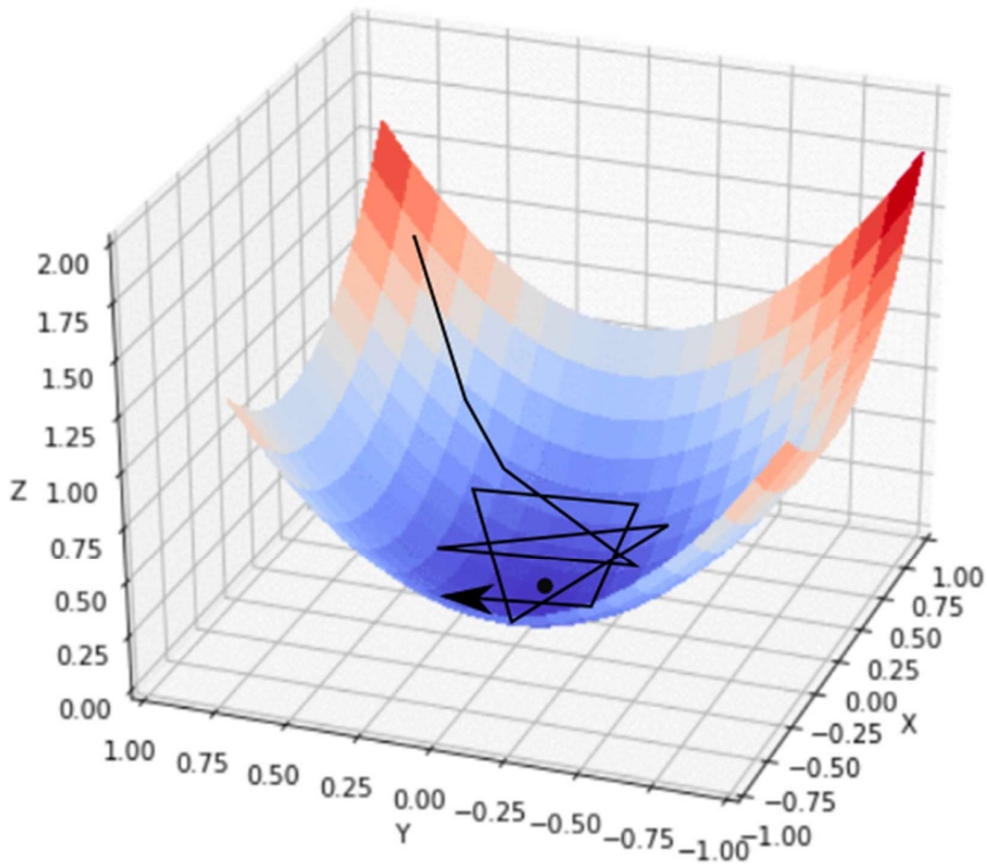


רשת הנוירונים היא בסופו של דבר פונקציה של המשקלים (שכן אין תלות שלהם בעולם החיצוני). בעזרת רשת הנוירונים/פונקציה נוכל לבצע משימות שבתכנות רגיל המחשב יתקשה או אפילו לא יהיה מסוגל לבצע, לדוגמה - זיהוי אובייקט בתמונה, זיהוי קול וניתוח טקסטים. ניקח לדוגמה זיהוי פנים: אנחנו עושים זאת על ידי אסיפת דוגמאות רבות (תמונות של פנים) שאפשר ליצור בעזרתן הכללה של מה הן פנים. בנוסף אנחנו ממציאים פונקציה מתמטית אחרת, אשר בודקת עד כמה רשת הנוירונים טעתה בזיהוי. לפונקציה זו קוראים פונקציית מחיר - LOSS. פונקציית המחיר הינה פונקציה גזירה, שכאשר מוצאים את המינימום שלה, ההפרש בין הפלט הצפוי לפלט האמיתי יהיה הקטן ביותר שאפשר, כלומר זיהוי נכון של הפנים בתמונה. מכיוון שלא פשוט לפתור נגזרת של פונקציה עם הרבה קלטים(רב ממדית) ושינויים נשתמש ב Stochastic Gradient Descent ובעזרתו נשנה כל פעם את המשקלים של הפונקציה

במידה מסוימת(לרוב קטנה), בכדי שנתקרב ליצירת פונקציה הממפה באופן הטוב ביותר את הקלט לפלט המתאים בנוירון ספציפי, כך שהפלט הסופי של הרשת יהיה הכי מדויק שאפשר.

Stochastic Gradient Descent

Stochastic Gradient Descent היא דרך למציאת מינימום (או קירוב שלו) של פונקציה, כאשר אי אפשר (או לא יעיל) פשוט למצוא אותו בערך פתירת משוואה, ונגזרתה ידועה לנו. עיקרה של הדרך היא חישוב ערך הנגזרת, ו"הליכה במורד" השיפוע של הפונקציה בכדי להתקרב על המינימום. מרחק ההליכה שאנחנו עושים תלוי בגודל השיפוע ו"בערך הלמידה – Learning rate" הנקבע מראש. למעשה, העניין הוא מאוד פשוט: מכפילים את ערך הנגזרת בערך הלמידה וב - 1-, והשינוי שווה למכפלה. הנה תמונה המבטאת את התהליך בפונקציה תלת ממדים (עם 3 קלטים):



ניתן לראות כי אנחנו מתחילים בנקודה גבוהה (אדומה) ויורדים בצעדים לאזור הנמוך (כחול). כאשר אנחנו מתקרבים למינימום (במרכז) אנחנו מסתובבים סביבו שכן הצעדים עדיין גדולים ואנחנו מפספסים את המינימום, אך מתקרבים אליו.

TensorFLow

ספריית קוד פתוח שמיועדת ללמידת מכונה ובעיקר לבנייה ואימון של רשתות עצביות. היא מפותחת על ידי חברת גוגל. לספרייה קיים API (ערכה של ספריות קוד, פקודות, ופונקציות מוכנות, בהן יכולים המתכנתים לעשות שימוש פשוט) לשפות תכנות רבות כגון: Python, C ו-JavaScript לצד רבות אחרות.

TensorFlow Object Detection API

אוסף של קטעי קוד, פונקציות, מודלים של למידת מכונה ומשקלים מתאימים לזיהוי אובייקטים, אשר פותחו על ידי גוגל. המודלים של זיהוי התמונות בAPI מאומנים על מיליוני תמונות של קלאסים (אובייקטים) שונים ברמות דיוק ומהירות זיהוי שונות(ככל שהמודל מהיר יותר כך הזיהוי פחות מדויק). בחרתי להשתמש בAPI זה מכיוון שהמודלים מאומנים על מיליוני תמונות וכך רמת הדיוק והמהירות שלהם טובה מאוד. אם הייתי עושה את המודל מאפס, אז לא הייתי מגיע לרמות דיוק ומהירות מספיק טובות בכדי לנתח פנים מווידאו בזמן אמת. מה שמאוד שימושי במודל שהחלטתי להשתמש בו הוא שניתן לבצע בו Transfer Learning.

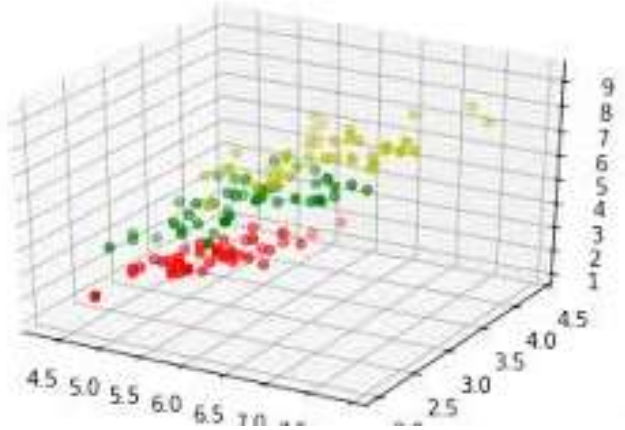
Transfer Learning

עיקר ה Transfer learning הוא לימוד מודל קיים לזהות דבר חדש על בסיס מה שהוא כבר למד בעבר. לדוגמה ניתן לקחת מסנן ספאם בדואר אלקטרוני: גוגל אימנו את המודל שלהם לזהות ספאם אך המשתמש יכול להגדיר מכתב מסוים כספאם ואז המודל ילמד לזהות מכתבים דומים כספאם. בפרויקט שלי ביצעתי שימוש ב Transfer learning בלימוד המודל של זיהוי אובייקטים, לזהות פנים אנושיות באמצעות 512 תמונות שבכל תמונה סימנתי במלבן היכן נמצאים הפנים.

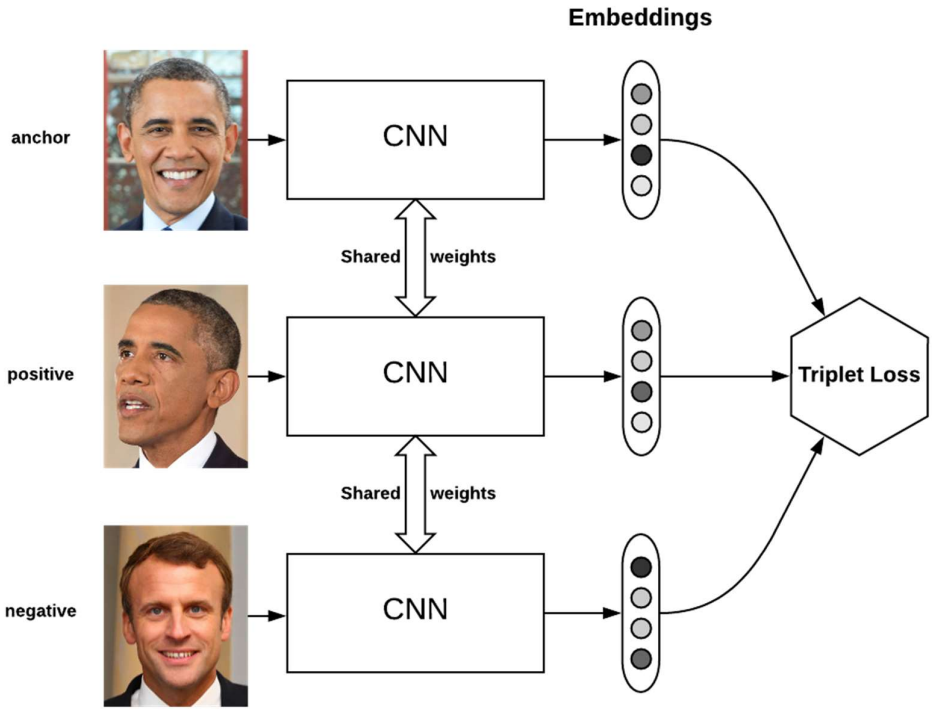
FaceNet and Triplet Loss

FaceNet היא ארכיטקטורה ומודל אשר הרעיון בה הוא שאפשר לתאר תמונה באופן של וקטור (נקרא Embedding) במרחב 128 ממדי, ובעזרת תיאור זה ניתן למצוא עד כמה שונות תמונות זו מזו, על ידי בחינת המרחק בין הווקטורים המתארים את התמונות. בכדי ליצור את התיאור הווקטורי לתמונה משתמשים ב CNN – Convolutional Neural Network שהיא בעצם רשת נוירונים מורכבת אשר מתאימה בייחוד לעבודה עם תמונות שכן לנוירונים בה תפקידים ספציפיים לגבי התמונה, לדוגמה: קבוצת נוירונים כלשהי יכולה להיות אחראית על לבדוק איפה יש קווים ישרים בתמונה וקבוצת נוירונים אחרת אחראית על בדיקת צבע מסוימת באזור מסוים בתמונה.

דוגמה ל3 אובייקטים שונים (במקרה הזה פנים של 3 אנשים שונים) במרחק תלת ממדי (אין אפשרות להציג 128 ממדים באופן ברור):



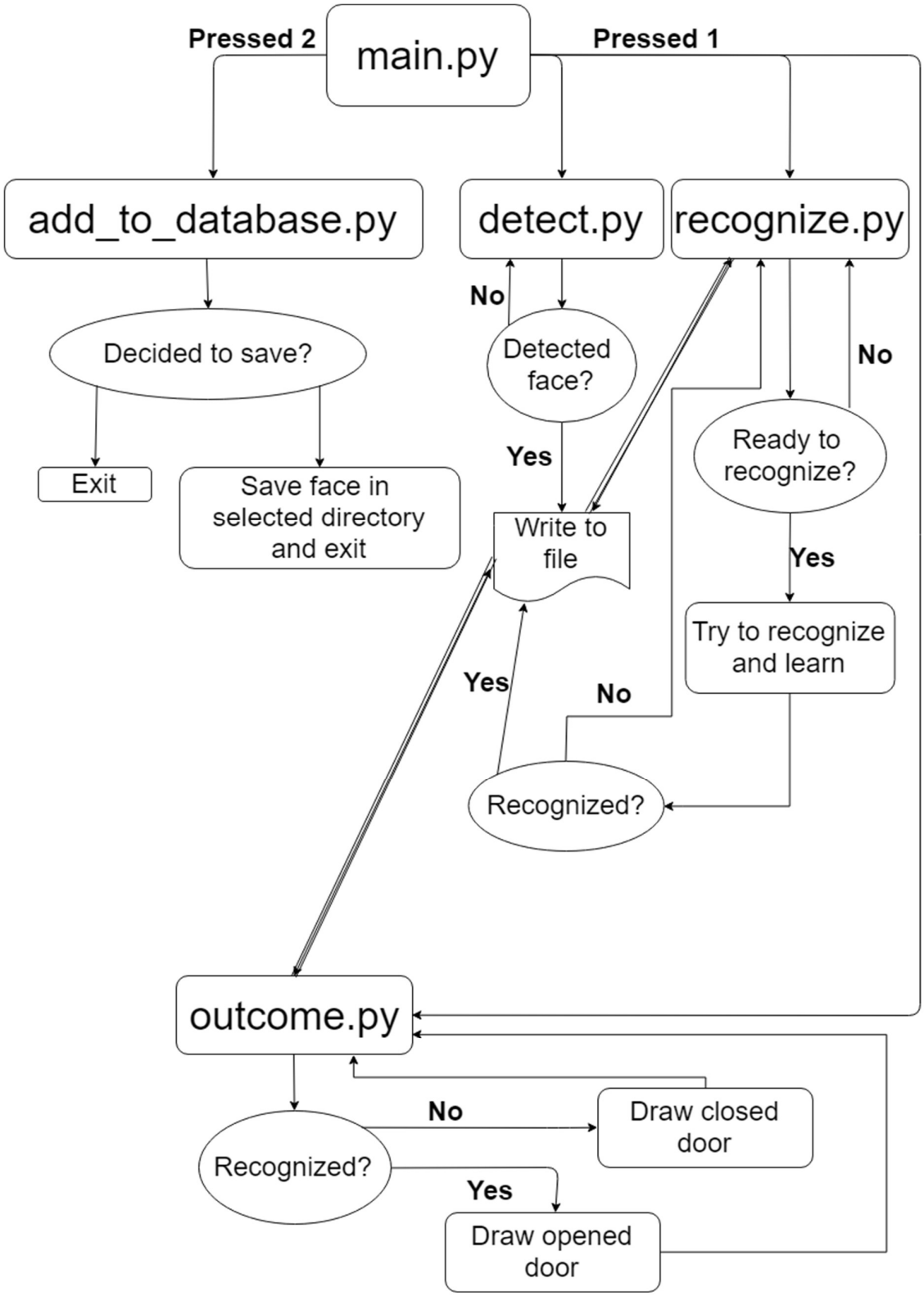
Triplet loss זה סוג של פונקציית מחיר שבה מנסים לקרב בין דוגמאות זהות ולהרחיק מהן את הדוגמה השונה במרחב כלשהו. במקרה של FaceNet הקירוב והריחוק של הדוגמאות מתרחב במרחב 128 ממדי שכן לכל Embedding 128 ממדים. הדבר מתרחש באמצעות מציאת שלשה של דוגמאות בהן אחת משמשת כעוגן – כלומר לא נעה ממקומה במרחב, בעוד שדוגמה אחרת משמשת כדוגמה זהה – Positive ותמונה אחרת משמשת כלא זהה – Negative. אילוסטרציה ל Triplet loss:



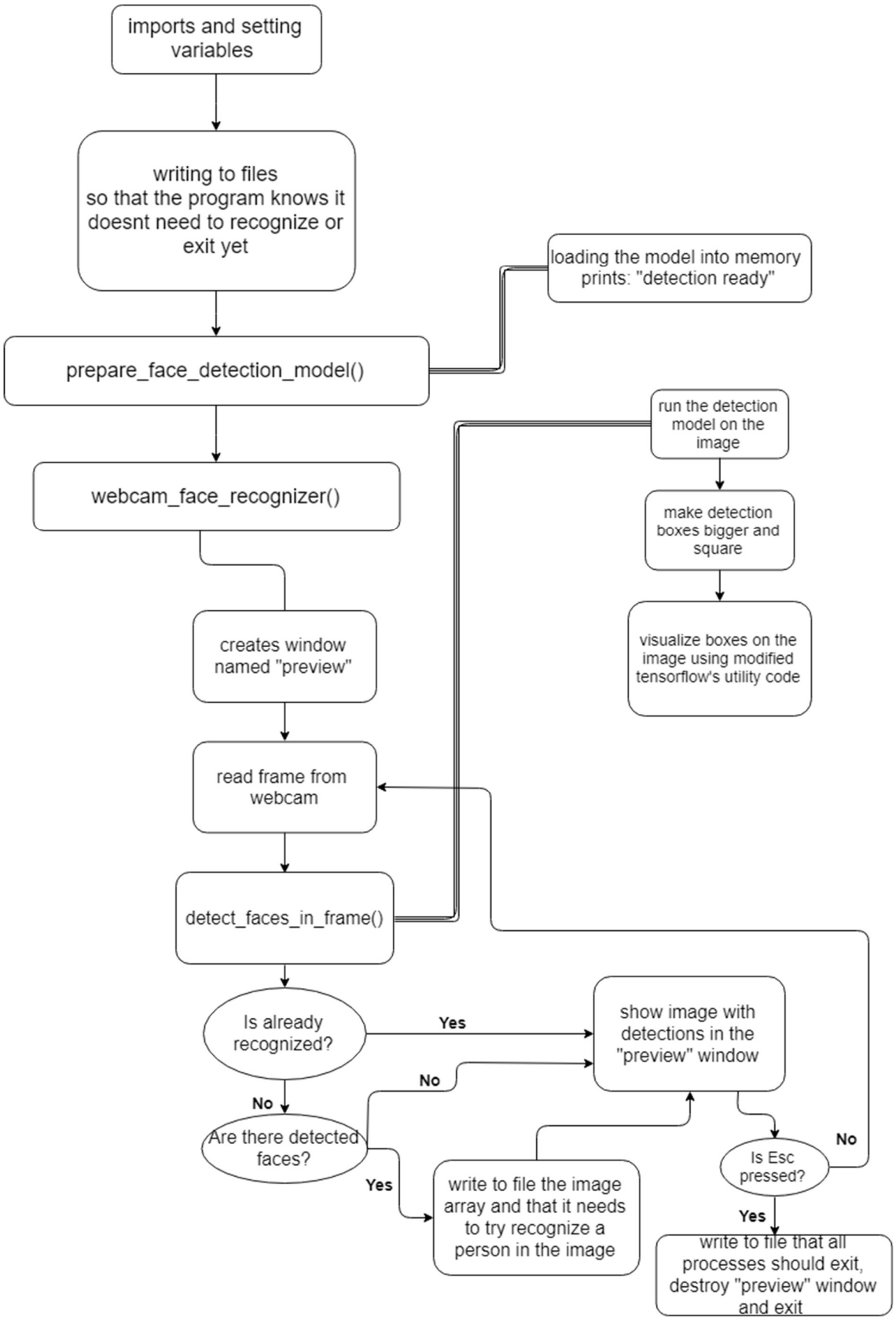
כאן אנחנו מנסים למקם את הEmbedding המתאר את הפנים של אובמה במקום המתאים ביותר, בעוד שמנסים להרחיק את הEmbedding המתאר את הפנים של מאקרון. שימוש ב CNN עם משקלים משותפים כמתואר באיור (הנקרא גם רשת סיאמית - Siamese network מכיוון המשקלים של הרשתות משותפים), מאפשר לנו ליצור Embeddings שהם כלליים למטרה (במקרה שלנו – פנים) וכך יהיה סטנדרט מסוים ונוכל בעזרתו להשוות בין ה-Embeddings השונים.

חלק מעשי

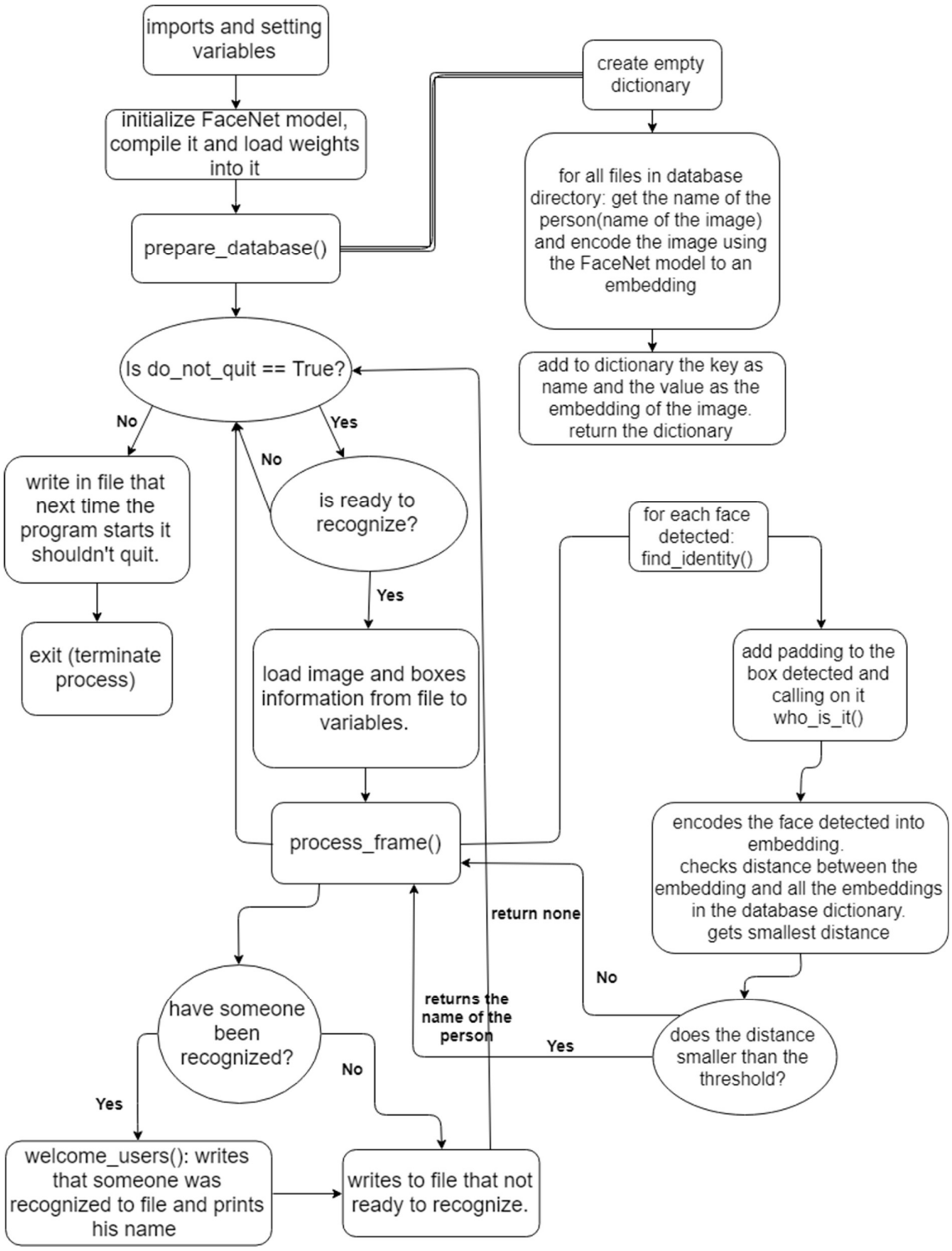
תרשים כללי main.py:



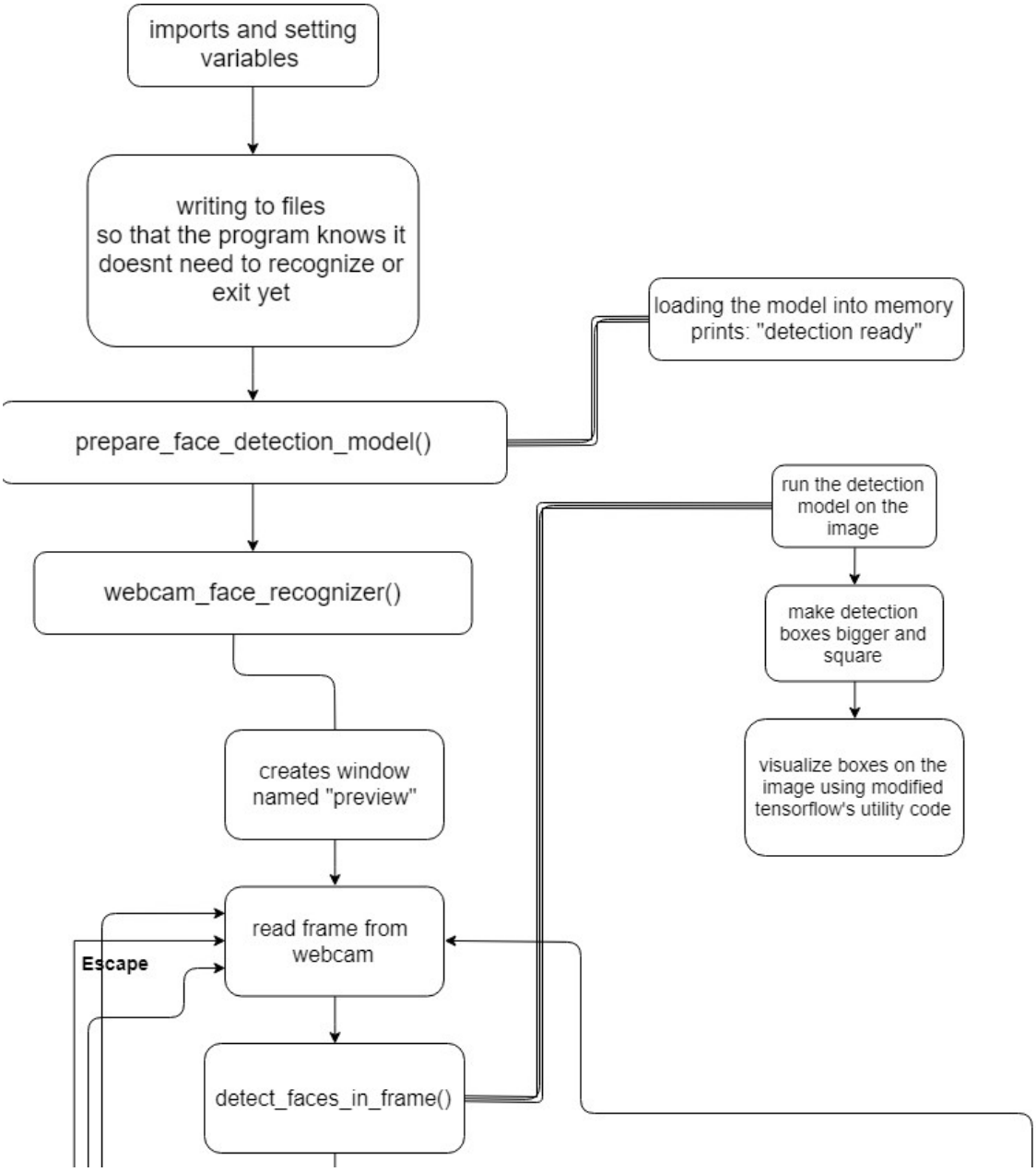
תרשים detect.py:

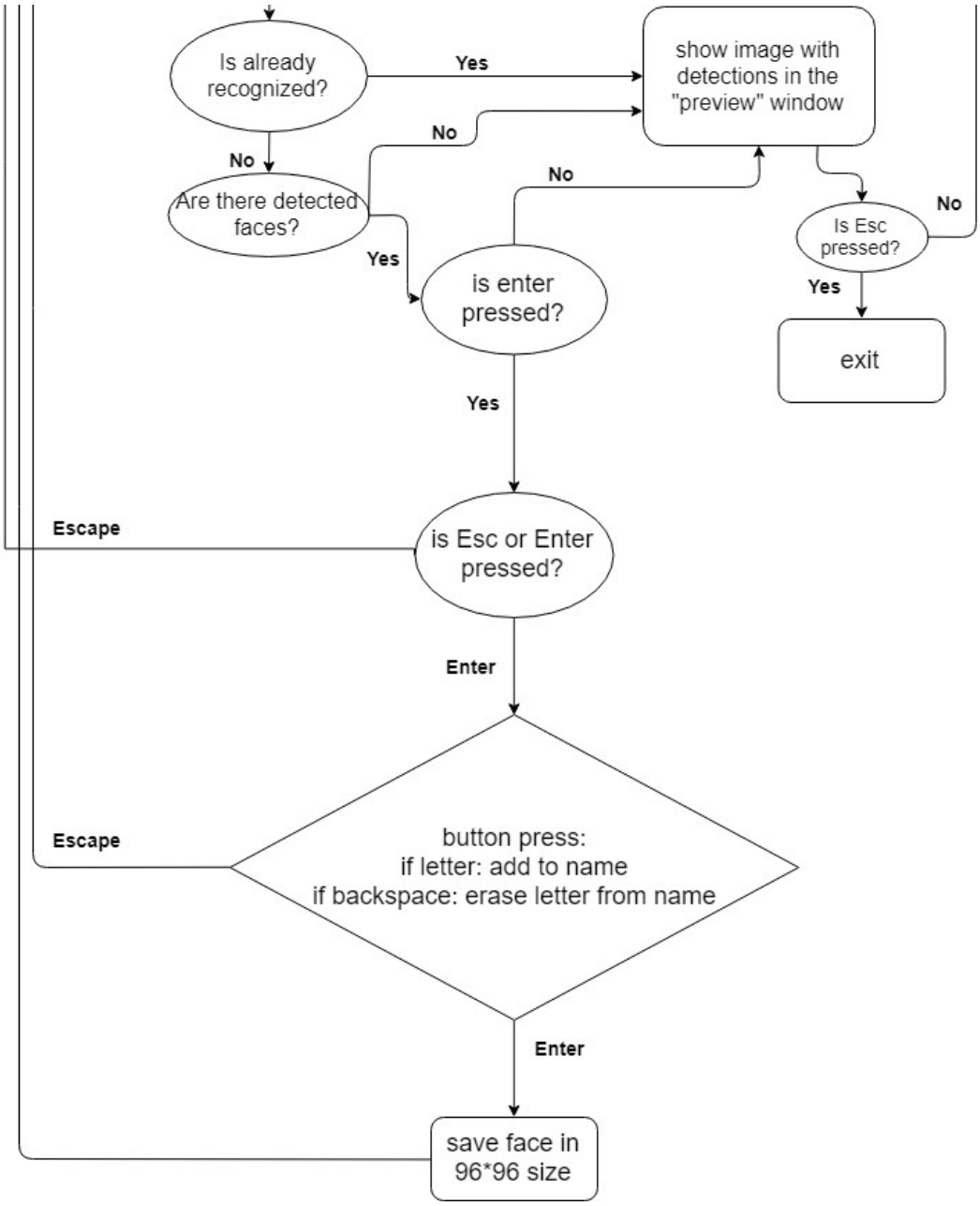


תרשים recognize.py:



תרשים add_to_database.py:




Is already recognized?
Yes
show image with detections in the "preview" window
No
No
Are there detected faces?
No
Is Esc pressed?
No
Yes
is enter pressed?
Yes
exit
Yes
Escape
is Esc or Enter pressed?
Enter
Escape
button press:
if letter: add to name
if backspace: erase letter from name
Enter
save face in 96*96 size

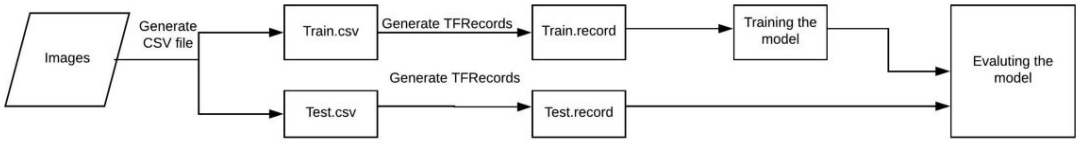
אופן פעולת התוכנה:

הפרויקט שלי מורכב מכמה תהליכים (Process). התהליך הראשון הוא תהליך המריץ את הקובץ main.py שבו אני נותן למשתמש לבחור את ה"מצב" של התוכנה – איתור וזיהוי הפנים או הוספת פנים חדשות למסד הנתונים. כאשר המשתמש בוחר ב"מצב" כלשהו אז אני מתחיל תהליכים חדשים המריצים את קבצי הפייתון המתאימים.

החלטתי להריץ את התוכנה בתהליכים נפרדים על מנת שאוכל לבצע איתור פנים בתמונה בזמן שתהליך אחר מבצע זיהוי על התמונה שנמצאה, בכדי שהמעקב אחרי הפנים בתמונה יהיה חלק.

איתור הפנים:

השתמשתי במודל זיהוי אובייקטים בתמונה של גוגל, ואימנתי אותו לאתר פנים בתמונות באופן המתואר בתרשים הבא:

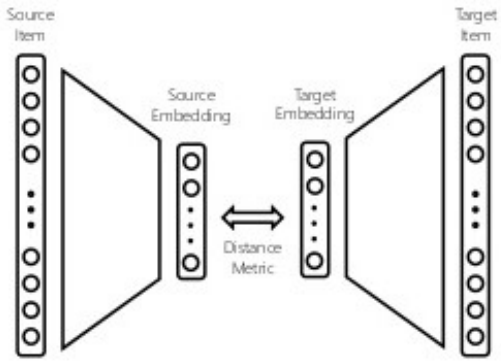


בחרתי 512 תמונות ממאגר תמונות גדול מאוד של אנשים שונים וסימנתי בהם את המיקום של הפנים עם מלבן, התכנה בה עשיתי זאת שמרה את המיקומים של המלבנים בקובץ XML. לאחר מכן באמצעות סקריפט ב Python תרגמתי את קבצי הXML לCSV. בנוסף הפרדתי כ450 תמונות לאימון Train ואת השאר לבחינה Test בכדי שהמודל יוכל להתאמן ולבחון את עצמו. לאחר מכן השתמשתי בסקריפט שניתן על ידי הAPI של Object Detection של TensorFlow לתרגום של קבצי ה CSV לקבצי tfrecord שהמודל של TensorFlow יודע לקרוא ולהבין. לאחר מכן הרצתי את קובץ האימון שגם נתון ב API ונתתי למודל כ-4 שעות להתאמן. לאחר מכן עצרתי אותו ושמרתי את המשקלים שאליו הוא הגיע בקובץ מסוג Prediction Graph – PB. כך אני יכול להשתמש בגרף/מודל הזה לזיהוי הפנים.

אני מבצע את איתור הפנים בתהליך נפרד המריץ את הקובץ detect.py. כאשר המודל מאתר פנים, הוא בודק אם התוכנה מוכנה לזהות אותם, ואם כן כותב את הפנים ונתונים נוספים לקובץ.

זיהוי הפנים:

בכדי לזהות בן אדם ספציפי אליו הפנים שייכות, אני החלטתי להשתמש במודל שמשתמש ב FaceNet על מנת לתאר פנים בתור וקטור ככה שאוכל למצוא מרחק אוקלידי בין ווקטורי פנים שונים.



החלטתי שאקח פריים מהווידאו של המצלמה, ואעשה עליו את חיפוש הפנים. במקרה שנמצאו פנים אז התמונה שנמצאת במלבן שנמצא)הפנים(מועברים לניתוח על ידי המודל של FaceNet בעוד שהמצלמה ממשיכה להראות את הסרטון רץ. אחרי שהתמונה מועברת לתיאור כווקטור, אז אני מחפש את המרחק הקצר ביותר בין הווקטור הזה לווקטורים של האנשים במסד הנתונים. המרחק הקצר ביותר כנראה יהיה האדם שאליו הפנים מתאימות, ולכן אחזיר את שמו.

כאשר התוכנה מוכנה לזהות פנים היא קוראת מתוך קבצים את התמונה ונתונים נוספים אשר נחוצים לזיהוי הפנים, כמו מיקום הפנים בתמונה, ולאחר הזיהוי מכניסה את המידע על הזיהוי לקבצים.

את זיהוי הפנים אני עושה בתהליך נפרד המריץ את הקובץ recognize.py.

המחשת תוצאת הזיהוי:

לתוצאות הזיהוי יכולים להיות שימושים רבים כמו שהצגתי במבוא הפרויקט, אך אני החלטתי להציג את התוצאה בכך שכאשר מזוהה אדם מוכר תמונה של דלת פתוחה תוצג, וכאשר לא אז הדלת תישאר סגורה.

את המחשת תוצאות הזיהוי אני עושה בתהליך נפרד הרץ במקביל לאיתור והזיהוי המריץ את קובץ outcome.py.

הוספת פנים (אדם) חדשים למערכת:

בכדי להקל על הוספת אדם חדש למסד הנתונים ויצירת סטנדרט מסוים בשבילו, בניתי מ detect.py סקריפט אשר נותן את האפשרות לשמור תמונה שאני חושב שהפנים בא אותרו היטב על ידי המערכת. לסקריפט זה קראתי add_to_database.py והוא רץ בתהליך נפרד כאשר המשתמש בוחר להריצו. כאשר יצרתי את הסקריפט הזה למדתי להוסיף טקסט על גבי התמונה והוספתי פונקציה לפונקציות העזר של מפתחי TensorFlow.

הקוד

:main.py

```
import os
from multiprocessing import Pool
import cv2

#tuple of the names of the files to run in the processes.
processes = ('detect.py', 'recognize.py', 'outcome.py', 'add_to_database.py')

def run_process(process):
    """Runs the python file"""
    os.system('python {}'.format(process))

def get_key_pressed():
    """Return the key pressed by the user if it's 1,2 or Esc"""
    key = 0
    while (key != 27 and key != ord('1') and key != ord('2')):
        key = cv2.waitKey(10)
    return key

if __name__ == "__main__":
    cv2.namedWindow("main") #create window "main"
    start_image = cv2.imread("project_images/start_image.png") #load image
    cv2.imshow("main", start_image) #show image
    key = get_key_pressed()
    if key == ord('1'):
        cv2.destroyWindow("main") #close the window

        #run the first 3 files in *processes* tuple
        #in different processes so they run at the same time.
        pool = Pool(processes=3)
        pool.map(run_process, processes[:3])
    elif key == ord('2'):
        cv2.destroyWindow("main") #close the window
        #run the add_to_database.py file in different process.
        pool= Pool(processes=1)
        pool.map(run_process, processes[3:])
    elif key == 27: #Esc exit
        exit
```

:detect.py

```
import cv2
import os
import numpy as np
import tensorflow as tf
from PIL import Image

# ## Object detection imports
# Here are the imports from the object detection module.
from utils import label_map_util
from utils import visualization_utils_only_needed as vis_util

#variables for the detection
min_score_threshold = 0.6
MAX_FACES_TO_DETECT = 1
ready_to_detect = True

def num_faces_detected(faces_scores):
    """returns the amount of faces detected based on the amount of scores in the
scores list that are bigget than the threshold"""
    num_faces = 0
    for x in range(0,MAX_FACES_TO_DETECT):
        if faces_scores[x] >= min_score_threshold:
          num_faces += 1
    return num_faces

def prepare_face_detection_model():
    """loads the detection model I trained"""
    # What model is going to be used.
    MODEL_NAME = 'face_graph'

    # Path to frozen detection graph. This is the actual model that is used for
the object(face) detection.
    PATH_TO_MODEL = MODEL_NAME + '/frozen_inference_graph.pb'

    # List of the labels that the model can identify (notains 1 label - face).
    PATH_TO_LABELS = os.path.join('training', 'face-detection.pbtxt')

    #amount of classes the model can identify (1 which is face).
    NUM_CLASSES = 1

    #loading the frozen model (weights saved) into memory
    detection_graph = tf.Graph()
    with detection_graph.as_default():
        od_graph_def = tf.GraphDef()
        with tf.gfile.GFile(PATH_TO_MODEL, 'rb') as fid:
            serialized_graph = fid.read()
            od_graph_def.ParseFromString(serialized_graph)
```

```
            tf.import_graph_def(od_graph_def, name='')

    #map of all the labels (1 - face)
    label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
    #create list of dictionaries with 'id' and 'name' keys, for example: {'0' :
'face'}
    categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
    #create dictionary with 'id' and category to this id
    category_index = label_map_util.create_category_index(categories)

    print("detection ready")
    return detection_graph, category_index

def detect_faces_in_frame(img, detection_graph, category_index, sess):
    """detects faces in the frame and returns the boxes in the face's location"""
    # Expand dimensions since the model expects images to have shape: [1, None,
None, 3]
    img_np_expanded = np.expand_dims(img, axis=0)
    image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
    # Each box represents a part of the image where a particular object was
detected.
    boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
    # Each score represent how level of confidence for each of the objects.
    # Score is shown on the result image, together with the class label.
    scores = detection_graph.get_tensor_by_name('detection_scores:0')
    classes = detection_graph.get_tensor_by_name('detection_classes:0')
    num_detections = detection_graph.get_tensor_by_name('num_detections:0')
    # Actual detection.
    (boxes, scores, classes, num_detections) = sess.run([boxes, scores, classes,
num_detections],
    feed_dict={image_tensor: img_np_expanded})

    #make the detection bounding box square because facenet requires,
    # and adding padding for extra safety(so all face will be in the box)
    image_pil = Image.fromarray(np.uint8(img)).convert('RGB')
    im_width, im_height = image_pil.size
    for box in boxes:
        box[0][0] = box[0][0]*im_height
        box[0][1] = box[0][1]*im_width
        box[0][2] = box[0][2]*im_height
        box[0][3] = box[0][3]*im_width
        deltaY = abs(box[0][0] - box[0][2])
        deltaX = abs(box[0][1] - box[0][3])
        diff = deltaY - deltaX
        divided_diff = diff/2
        box[0][1] -= divided_diff
        box[0][3] += divided_diff
```

```
    #visualize the bounding boxes on the frame
    return vis_util.visualize_boxes_and_labels_on_image_array(
    img,
    np.squeeze(boxes),
    np.squeeze(classes).astype(np.int32),
    np.squeeze(scores),
    category_index,
    use_normalized_coordinates=False,
    skip_labels=True,
    skip_scores=True,
    line_thickness=8,
    min_score_thresh=min_score_threshold), boxes[0], scores[0]

def webcam_face_recognizer():
    """
    Runs a loop that extracts images from the computer's webcam and checks if it
contains a face.
    If it contains a face, it will update the flags in the files so that the
recognition could start
    """
    global ready_to_detect
    #create camera window and set input
    cv2.namedWindow("preview")
    cam_input = cv2.VideoCapture(0)

    #loop for going frame after frame
    with detection_graph.as_default():
        with tf.Session(graph=detection_graph) as sess:
            while cam_input.isOpened():

                _, frame = cam_input.read()
                img = np.copy(frame)

                if ready_to_detect == True:
                    img, face_boxes, face_scores = detect_faces_in_frame(img,
detection_graph, category_index, sess)  #detect faces in the frame
                    ready_to_recognize_file =
open("varThread\\ready_recognize.txt", "r") #checks if we even need to check
whether faces detected,

#as if it's recognizing we do not need to send the information
                    if int(ready_to_recognize_file.read(1)) == 0: #not recognizing
                        ready_to_recognize_file.close()
                        num_detected_faces = num_faces_detected(face_scores) #how
many faces detected
                        if num_detected_faces > 0:
```

```
                            np.savez('varThread\\vars_to_pass.npz', img = frame,
face_boxes = face_boxes[:num_detected_faces]) #save the image array in file
                            ready_to_recognize_file =
open("varThread\\ready_recognize.txt", "r+") #make the ready to recognize 'flag'
true - 1
                            ready_to_recognize_file.seek(0, 0)
                            ready_to_recognize_file.write("1")
                            ready_to_recognize_file.close()
                    ready_to_recognize_file.close()
                key = cv2.waitKey(10) #variable to check if exit key is pressed
                cv2.imshow("preview", img) #show the video input with the
detections on screen

                if key == 27: # exit on ESC
                    is_quit_file = open("varThread\\is_quit.txt", "w")
                    is_quit_file.write('1')
                    is_quit_file.close()
                    break

            cv2.destroyWindow("preview")


#code starts
#initalize the flags in the files to be: don't recognize and don't exit
ready_to_recognize_file = open("varThread\\ready_recognize.txt", "w")
ready_to_recognize_file.write("0")
ready_to_recognize_file.close()

is_quit_file = open("varThread\\is_quit.txt", "w")
is_quit_file.write('0')
is_quit_file.close()

detection_graph, category_index = prepare_face_detection_model() #initialize model
webcam_face_recognizer() #start detection
```

:recognize.py

```
from keras import backend as K
K.set_image_data_format('channels_first')
import time
import cv2
import os
import glob
import numpy as np
import tensorflow as tf
from fr_utils import *
from inception_blocks_v2 import *

#variables for the model
PADDING = 50
MAX_FACES_TO_DETECT = 10
MIN_DIST_THRESHOLD = 0.7

def triplet_loss(y_true, y_pred, alpha = 0.3):
    """
    Implementation of the triplet loss as defined by formula (3)

    Arguments:
    y_pred -- python list containing three objects:
            anchor -- the encodings for the anchor images, of shape (None, 128)
            positive -- the encodings for the positive images, of shape (None,
128)
            negative -- the encodings for the negative images, of shape (None,
128)

    Returns:
    loss -- real number, value of the loss
    """

    anchor, positive, negative = y_pred[0], y_pred[1], y_pred[2]

    # Step 1: Compute the (encoding) distance between the anchor and the positive,
you will need to sum over axis=-1
    pos_dist = tf.reduce_sum(tf.square(tf.subtract(anchor, positive)), axis=-1)
    # Step 2: Compute the (encoding) distance between the anchor and the negative,
you will need to sum over axis=-1
    neg_dist = tf.reduce_sum(tf.square(tf.subtract(anchor, negative)), axis=-1)
    # Step 3: subtract the two previous distances and add alpha(learning
variable).
    basic_loss = tf.add(tf.subtract(pos_dist, neg_dist), alpha)
    # Step 4: Take the maximum of basic_loss and 0.0. Sum over the training
examples.
    loss = tf.reduce_sum(tf.maximum(basic_loss, 0.0))
```

```
    return loss

def prepare_database():
    database = {}

    # load all the images of individuals to recognize into the database
    for file in glob.glob("test_images/*"):
        identity = os.path.splitext(os.path.basename(file))[0] #
        database[identity] = img_path_to_encoding(file, FRmodel) #encode to
embedding
    print("finished data")
    return database


def process_frame(img, frame, boxes):
    """
    Determine whether the current frame contains the faces of people from our
database
    """
    faces = boxes
    # Loop through all the faces detected and determine whether or not they are in
the database
    identities = []

    for (y, x, h, w) in faces: #add padding
        x1 = int(x-PADDING)
        y1 = int(y-PADDING)
        x2 = int(w+PADDING)
        y2 = int(h+PADDING)

        identity = find_identity(frame, x1, y1, x2, y2)

        if identity is not None:
            identities.append(identity)

    if identities != []:
        welcome_users(identities)

    #set flag to not recognize until face detected again
    ready_to_recognize_file = open("varThread\\ready_recognize.txt", "r+")
    ready_to_recognize_file.seek(0,0)
    ready_to_recognize_file.write('0')
    ready_to_recognize_file.close()

def find_identity(frame, x1, y1, x2, y2):
    """
    Determine whether the face contained within the bounding box exists in our
database
```

```
    x1,y1_____________
    |                 |
    |                 |
    |_________________x2,y2

    """
    height, width, channels = frame.shape
    # The padding is necessary since the OpenCV face detector creates the bounding
box around the face and not the head
    part_image = frame[max(0, y1):min(height, y2), max(0, x1):min(width, x2)]

    return who_is_it(part_image, database, FRmodel)

def who_is_it(image, database, model):
    """
    Arguments:
    image -- array that represents the image
    database -- database containing image encodings along with the name of the
person on the image
    model -- FaceNet model

    Returns:
    min_dist -- the minimum distance between image_path encoding and the encodings
from the database
    identity -- string, the name prediction for the person on image_path
    """
    encoding = img_to_encoding(image, model) #encodes the current face image

    min_dist = 100
    identity = None

    # Loop over the database dictionary's names and encodings.
    for (name, db_enc) in database.items():

        # Compute L2 distance between the target "encoding" and the current "emb"
from the database.
        dist = np.linalg.norm(db_enc - encoding)

        print('distance for %s is %s' %(name, dist))

        # If this distance is less than the min_dist, then set min_dist to dist,
and identity to name
        if dist < min_dist:
            min_dist = dist
            identity = name

    if min_dist > MIN_DIST_THRESHOLD: #check if distance is bigger than the
threshold
```

```
        return None
    else:
        return str(identity)

def welcome_users(identities):
    """ Create file that is used for checking if person is recognized.
        Prints the name of the recognized person.
    """
    welcome_message = 'Hello'
    is_recognized_file = open('varThread\\recognized.check', 'w+')
    is_recognized_file.write('1')
    is_recognized_file.close()
    if len(identities) == 1:
        welcome_message += '%s!' % identities[0]
        print(welcome_message)
    else:
        for identity_id in range(len(identities)-1):
            welcome_message += '%s, ' % identities[identity_id]
        welcome_message += 'and %s, ' % identities[-1]
        welcome_message += '!'
        print(welcome_message)

def is_ready_to_recognize():
    """Checks file and return True if flag 1 and False when flas is 0"""
    time.sleep(0.01)
    ready_to_recognize_file = open("varThread\\ready_recognize.txt", "r")
    if int(ready_to_recognize_file.read(1)) == 1:
        ready_to_recognize_file.close()
        return True
    else:
        return False


#code starts here
#initalizing model and database
FRmodel = faceRecoModel(input_shape=(3, 96, 96))
print("shaped_model")
FRmodel.compile(optimizer = 'adam', loss = triplet_loss, metrics = ['accuracy'])
print("compiled model")
load_weights_from_FaceNet(FRmodel)
print("loaded FaceNet model")
database = prepare_database()

do_not_quit = True

while(do_not_quit == True):
    if is_ready_to_recognize(): #check if ready to recognize
        #load all the variables from the files
        recognize_variables = np.load("varThread\\vars_to_pass.npz")
```

```
    img = recognize_variables['img']
    frame = np.copy(img)
    boxes = recognize_variables['face_boxes']
    process_frame(img, frame, boxes)
is_quit_file = open("varThread\\is_quit.txt", "r+")
if int(is_quit_file.read(1)) == 1: #check if ready to quit
    is_quit_file.seek(0,0)
    is_quit_file.write('0')
    is_quit_file.close()
    do_not_quit = False
    print("quiting recognition")
```

:add_to_database.py

```
import cv2
import os
import numpy as np
import tensorflow as tf
from PIL import Image
import tkinter as tk
from tkinter import filedialog

# ## Object detection imports
# Here are the imports from the object detection module.
from utils import label_map_util
from utils import visualization_utils_only_needed as vis_util

#variables for the detection
min_score_threshold = 0.6
MAX_FACES_TO_DETECT = 1
MARGIN = 20
ready_to_detect = True

def prepare_face_detection_model():
    """loads the detection model I trained"""
    # What model is going to be used.
    MODEL_NAME = 'face_graph'

    # Path to frozen detection graph. This is the actual model that is used for
the object(face) detection.
    PATH_TO_MODEL = MODEL_NAME + '/frozen_inference_graph.pb'

    # List of the labels that the model can identify (notains 1 label - face).
    PATH_TO_LABELS = os.path.join('training', 'face-detection.pbtxt')

    #amount of classes the model can identify (1 which is face).
    NUM_CLASSES = 1

    #loading the frozen model (weights saved) into memory
    detection_graph = tf.Graph()
    with detection_graph.as_default():
        od_graph_def = tf.GraphDef()
        with tf.gfile.GFile(PATH_TO_MODEL, 'rb') as fid:
            serialized_graph = fid.read()
            od_graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(od_graph_def, name='')

    #map of all the labels (1 - face)
    label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
    #create list of dictionaries with 'id' and 'name' keys, for example: {'0' :
'face'}
```

```
    categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
    #create dictionary with 'id' and category to this id
    category_index = label_map_util.create_category_index(categories)

    print("detection ready")
    return detection_graph, category_index

def detect_faces_in_frame(img, detection_graph, category_index, sess):
    """detects faces in the frame and returns the boxes in the face's location"""
    # Expand dimensions since the model expects images to have shape: [1, None,
None, 3]
    img_np_expanded = np.expand_dims(img, axis=0)
    image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
    # Each box represents a part of the image where a particular object was
detected.
    boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
    # Each score represent how level of confidence for each of the objects.
    # Score is shown on the result image, together with the class label.
    scores = detection_graph.get_tensor_by_name('detection_scores:0')
    classes = detection_graph.get_tensor_by_name('detection_classes:0')
    num_detections = detection_graph.get_tensor_by_name('num_detections:0')
    # Actual detection.
    (boxes, scores, classes, num_detections) = sess.run([boxes, scores, classes,
num_detections],
    feed_dict={image_tensor: img_np_expanded})

    #make the detection bounding box square because facenet requires,
    # and adding padding for extra safety(so all face will be in the box)
    image_pil = Image.fromarray(np.uint8(img)).convert('RGB')
    im_width, im_height = image_pil.size
    for box in boxes:
        box[0][0] = box[0][0]*im_height
        box[0][1] = box[0][1]*im_width
        box[0][2] = box[0][2]*im_height
        box[0][3] = box[0][3]*im_width
        deltaY = abs(box[0][0] - box[0][2])
        deltaX = abs(box[0][1] - box[0][3])
        diff = deltaY - deltaX
        divided_diff = diff/2
        box[0][1] -= divided_diff
        box[0][3] += divided_diff

    #visualize the bounding boxes on the frame
    return vis_util.visualize_boxes_and_labels_on_image_array(
    img,
    np.squeeze(boxes),
    np.squeeze(classes).astype(np.int32),
```

```
    np.squeeze(scores),
    category_index,
    use_normalized_coordinates=False,
    skip_labels=True,
    skip_scores=True,
    line_thickness=8,
    min_score_thresh=min_score_threshold), boxes[0], scores[0]

def num_faces_detected(faces_scores):
    """returns the amount of faces detected based on the amount of scores in the
scores list that are bigget than the threshold"""
    num_faces = 0
    for x in range(0,MAX_FACES_TO_DETECT):
        if faces_scores[x] >= min_score_threshold:
          num_faces += 1
    return num_faces

def webcam_face_recognizer():
    """
    Runs a loop that extracts images from the computer's webcam and determines
whether or not
    it contains faces.
    If it contains a face, it will give the option to save the image.
    """
    #create camera window and set input
    cv2.namedWindow("preview")
    choose_directory_image =  cv2.imread("project_images/choose_directory.png")
    cv2.imshow("preview", choose_directory_image)
    #open file explorer to choose database directory for saving the image
    root = tk.Tk()  # open tkinter widget
    root.withdraw() # Close the root window
    path = filedialog.askdirectory()

    cam_input = cv2.VideoCapture(0)

    #loop for going frame after frame
    with detection_graph.as_default():
        with tf.Session(graph=detection_graph) as sess:
            while cam_input.isOpened():

                _, frame = cam_input.read()
                img = np.copy(frame)

                key = cv2.waitKey(25)

                if key == 27: # exit on ESC
                    break
```

```
                if ready_to_detect == True:
                    img, face_boxes, face_scores = detect_faces_in_frame(img,
detection_graph, category_index, sess)  #detect faces in the frame
                    num_detected_faces = num_faces_detected(face_scores) #checks
if we had detected any face
                    if num_detected_faces > 0:
                        key = cv2.waitKey(15) #variable to check if enter key is
pressed
                        if key == 13: # stop on enter key
                            key = 0
                            vis_util.add_text_on_image(img, "If you like the image
press enter to save. \nEsc if otherwise.")
                            while (key != 27 and key != 13):
                                cv2.imshow("preview", img) #show the video input
with the detections on screen
                                key = cv2.waitKey(15) #variable to check if enter
key is pressed
                                if key == 13:
                                    img = np.copy(frame)
                                    name_done = False
                                    name = ""
                                    while(name_done == False): #writing the name
of the person
                                        img_with_name = np.copy(img)
                                        img_with_name =
vis_util.add_text_on_image(img_with_name, "Enter a name:" + name)
                                        cv2.imshow("preview", img_with_name)
                                        key = cv2.waitKey(15)
                                        if(key == 27):
                                            break
                                        elif(key == 13):
                                            name_done = True
                                        elif(key == 8):
                                            if(name != ""):
                                                name = name[:-1]
                                        elif(key == -1):
                                            pass
                                        else:
                                            name += chr(key)
                                    if name_done == True:
                                        #crop the face from the image
                                        cropped_img = frame[int(face_boxes[0][0])
- MARGIN:int(face_boxes[0][2])+MARGIN,
                                        int(face_boxes[0][1])-
MARGIN:int(face_boxes[0][3])+MARGIN]
                                        cropped_img = cv2.resize(cropped_img, (96,
96)) #resize to 96*96 so it will match FaceNet's input and be small
```

```
                                        cv2.imwrite( str(path) + name  +".jpg",
cropped_img ) #save image
                                    else:
                                        break

                cv2.imshow("preview", img) #show the video input with the
detections on screen

            cv2.destroyWindow("preview")


#code start
detection_graph, category_index = prepare_face_detection_model() #initialize
webcam_face_recognizer() #start detecting
```

:outcome.py

```
import os
import cv2
import time

def show_opened_door():
    """Shows the door opened"""
    opened_door = cv2.imread("project_images/opened_door.png")
    cv2.imshow("outcome", opened_door)
    timer = 0
    start_time = time.time()
    #after starting, the door is opened for 3 seconds
    while(timer <= 3):
        key = cv2.waitKey (25)
        if key == 27:
            exit()
        timer = time.time() - start_time


def show_door():
    """Draws the door in another window"""
    cv2.namedWindow("outcome") #create window
    closed_door = cv2.imread("project_images/closed_door.png") #load image
    while True:
        cv2.imshow("outcome", closed_door) #show the door closed
        #check if person recognized
        is_recognized_file = open('varThread\\recognized.check', 'r+')
        is_recognized = is_recognized_file.read(1)
        if is_recognized == '1':
            #set that person has been recognized to not and show the open door
            is_recognized_file.seek(0,0)
            is_recognized_file.write('0')
            is_recognized_file.close()
            show_opened_door()
        else:
            is_recognized_file.close()
            #check if need to quit
            is_quit_file = open("varThread\\is_quit.txt", "r+")
            if int(is_quit_file.read(1)) == 1:
                is_quit_file.seek(0,0)
                is_quit_file.write('0')
                is_quit_file.close()
                exit()
        key = cv2.waitKey (25)
        if key == 27: #exit on hitting Esc
            exit()
```

```
#code starts
#making sure that the recognized flag in 0 in the beginning so the door is closed.
is_recognized_file = open('varThread\\recognized.check', 'w')
is_recognized_file.write('0')
is_recognized_file.close()
show_door() #start showing the door
```

:xml_to_csv.py

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET


def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax',
'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df


def main():
    for directory in ['train', 'test']:
        image_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv('data/{}_labels.csv'.format(directory), index=None)
        print('Successfully converted xml to csv.')


main()
```