זיהוי מצב מסכה בתמונות סלפי – פרויקט למידת מכונה

Mask Wearing Status at Selfie Pictures Classification Program

שם המגיש :גלעד בר אילן

תעודת זהות: 327575734

כיתה :יב4

בית ספר :מקיף י"א ראשונים, ראשון לציון.

שם המנחה :דינה קראוס

שם החלופה: למידת מכונה

תאריך ההגשה: 16.6.2022

תוכן עניינים

מבוא

דיפ לרנינג הוא תת נושא בלמידת מכונה אשר מנסה לדמות את שכבות הנוירונים שיש במוח בכדי ללמוד. באמצעות שיטות שונות כגון פונקציית שגיאה, optimizer, כמויות שונות של נוירונים, activation functions וכו' הוא יכול לדמות בצורה מלאכותית חשיבה של אדם ובכך להגיע ליכולות כגון סיווג שמאפשרות לו לתת תגיות על תמונות שהוא לא מכיר (למשל לזהות סוגי פרחים).

מטרת הפרויקט הינה לסווג מקרים שונים של חבישת מסיכה בתמונות סלפי. קהל היעד של הפרויקט יכול להיות מקומות בילוי, עיסוק ,בתי ספר ,גנים ובכלל כל מקום שכולל מתחם סגור אשר רוצים לסווג מי מבין האנשים רשאי להיכנס אל המתחם בכך שיראו שהוא חובש את המסכה, או לחילופין במצב שבו אותו אדם אינו חובש את המסכה בצורה נכונה לבקש ממנו לחבוש את המסכה בצורה נכונה.

הרחבה עתידית אפשרית של הפרויקט היא אפשרות להכניס אותו למערכות שיוכלו לתת בזמן אמת סטטוס על מי חובש מסכה, מי לא חובש מסכה, ומי חובש את המסכה שלו בצורה שהיא בלתי תקינה. באמצעות תוספות אפשריות לפרויקט למשל זיהוי בני אדם, חברות יוכלו לסווג אילו מן העובדים שלהם אינם עומדים בכללים ובכך לשלוח להם הודעה / להתקשר אליהם ולהודיע להם על מה הם עושים לא בסדר.

הסיבה שבחרתי בפרויקט הזה היא מכיוון שאני קודם כל רואה את החשיבות העילאית של לשמור על הבריאות של כולנו, באמצעות אי חבישת מסיכה אותם אנשים מסכנים את עצמם ואת הסביבה.

מקור המידע שבו השתמשתי לפרויקט הינו kaggle משם לקחתי את הdataset של כל התמונות. לתמונות עצמן היה label בשם הקובץ שתיאר את סטטוס המסכה בתמונה. הדבר הראשון שעשיתי לכל התמונות היה לשנות את גודל התמונה לגודל קבוע של x150150 פיקסלים מכיוון שגודלי התמונה הקודמים היו גדולים מאוד ובעיה עיקרית מאוד הייתה שגודל התמונות על הדיסק היה מאות ג'יגה בייט. דבר נוסף ששיניתי באיך שהמידע היה מאורגן הוא שלקחתי את התמונות לאחר ההקטנה וחילקתי אותן לשלוש תיקיות שונות של train, test, validation ובתוכם 4 תיקיות שמייצגות את המצבים השונים של המסיכות masked, unmasked, half mask, bottom mask. הסיבה העיקרית שממנה בחרתי לחלק את התמונות לתיקיות היה בכדי לחסוך בזמן ריצה, אם כל התמונות היו בתיקייה אחת הייתי צריך למיין את כל התמונות למשתנים הנכונים בכל פעם שהייתי רוצה להתחיל לאמן את המודל. באמצעות החלוקה לתיקיות יכולתי פשוט לקלוט בעבור כל משתנה מתוך תיקייה קבועה במערכת שיצרתי.

הבעיה הראשונה שצפויה לצוץ היא העובדה שרוב החומר הוא לא בעברית. כלומר יש לחפש מידע ולחקור מאמרים בשפה זרה ולהבין אותם לעומק.

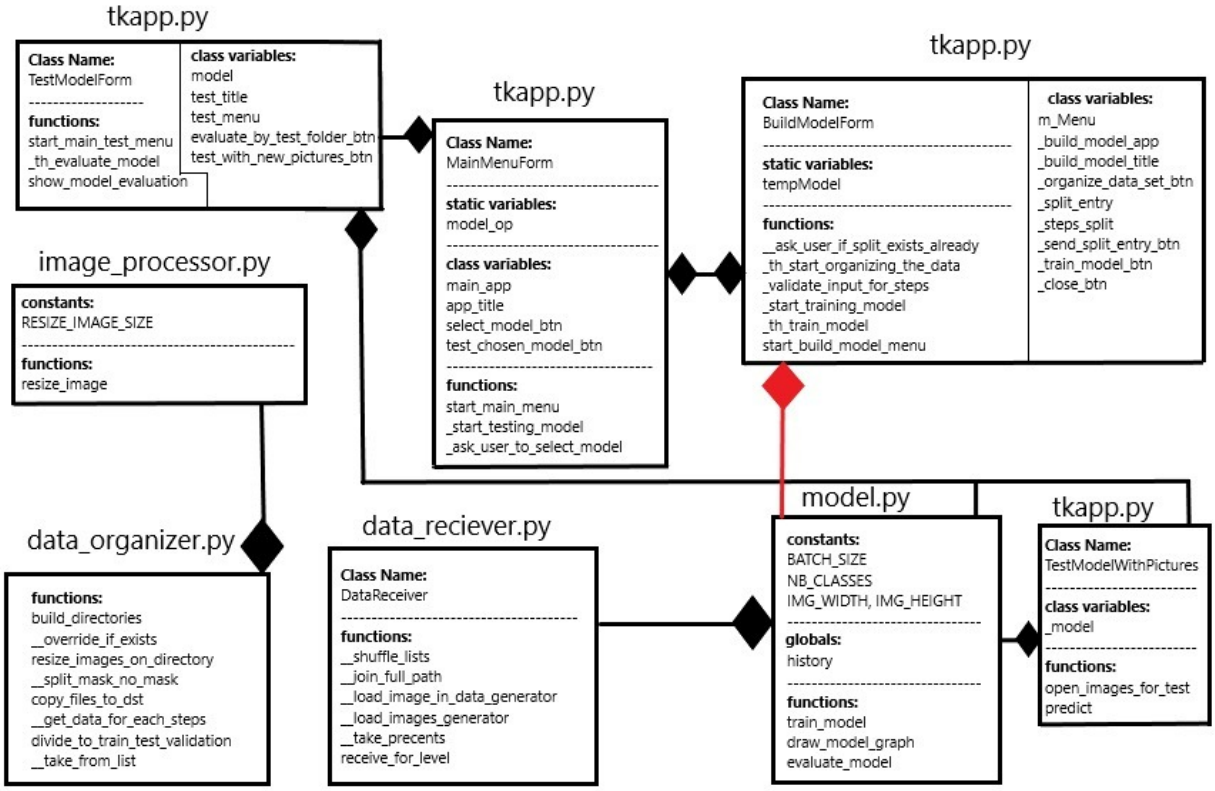
בעיה נוספת שיכולה להיות היא שנושא הדיפ לרנינג הוא תחום שלא עסקתי בו לפני וכן ואין לי בו ניסיון על כן יש הרבה חומר שצריך ללמוד בכדי להצליח להגיע להצלחה גבוהה.

בעיה נוספת היא שמודלים צריכים מחשב טוב להתאמן עליהם ומכיוון שהמחשב שלי לא הכי טוב זה יכול לגרום שלל בעיות כמו: ריצה איטית לתכנית, אי הפעלה של תוכנות אחרות על המחשב במקביל להרצה של התכנית. במהלך הפרויקט (בעיקר בתחילתו) יצא לי להשתמש במקביל למחשב שלי ב google colab שאפשר לי להריץ דרך האינטרנט חלק מהקוד וכך יכלתי לבדוק אותו במהירות רבה יותר מבלי להעמיס על המחשב שלי.

הפרויקט הזה בא לתת פתרון לאי שמירה על ההנחיות במקומות סגורים וציבוריים ומטרתו לשמור על כולנו מוגנים ובריאים מפני המגפה.

מבנה / ארכיטקטורה של הפרויקט

תרשים **UML** של כל המחלקות והמודולים בתכנית



*הסימון באדום כדי שלא נתבלבל עם הקו המצטלב.

שימוש במודולים סימנתי גם כ"הכלה".

שלב איסוף וניתוח הנתונים

תיאור מבנה הנתונים:

https://www.kaggle.com/datasets/tapakah68/medical-masks-p4 - (מדובר על דאטה סט אחד מתוך 7 דאטה סטים שונים של נתונים מהם לקחתי שלושה דאטה סט).

מקור המידע שבו השתמשתי לפרויקט הינו Kaggle.

כל אחד מהדאטה סטים היה שייך לאותו מקור ומסודר באותה צורה.

כל אחד מן ממאגרי המידע היה למעשה תיקייה אחת עם המון תמונות שעברו validation על ידי TrainingData.ru ונאספו על ידי Toloka.ai את מאגרי המידע האלה Kucev Roman לאתר Kaggle.

כל אחת מן התמונות מכיל label בשם הקובץ **photo type, person's age, gender, user ID**.

כאשר כל התייחסות לlabel ספציפי מתוכם מופרד באמצעות קו תחתון.

התייחסות לסוגי הphoto type

1 – מסכה על האף.

2 – מסכה על הפה.

3 – מסכה על הסנטר.

4 – ללא מסכה.

לדוגמה בעבור שם הקובץ:

MALE_23_000002_1_000002

1 – photo type – סוג המסכה – במקרה זה על הפה.

23 – age – גילו של חובש המסכה.

Male – Gender – מינו של חובש המסכה.

User ID - 000002.

תיאור ותהליך הכנת הDataset לאימון כולל הסבר אודות שיטת נרמול הנתונים:

תהליך בניית הDataset בפרויקט מתחלק לשני חלקים, החלק הראשון הוא הדרך בה חילקתי את המידע על הדיסק והחלק השני הוא הדרך בה קראתי את המידע לאחר שהוא חולק את הדיסק.

ארגון המידע על הדיסק:

הקושי הראשוני שנתקלתי בו במהלך בניית הפרויקט הוא שגודל התמונות שקיבלתי היה לא קבוע, ושגודל התמונות היה גדול מאוד. גודל גדול מאוד לתמונה יכול לגרור כמה בעיות, דבר ראשון הוא יכול לגרום למודל שלנו לעבוד בצורה מאוד איטית ולהכביד מאוד על המחשב עליו אנחנו מריצים את הפרויקט וכמו כן גודלו על הדיסק היה גם מאוד גדול ולא היה באפשרותי לאחסן את כל התמונות הללו בבת אחת מכיוון שגודלן על הדיסק היה כמה מאות של ג'יגה בייטים.

מסיבה זו הדבר הראשון שבחרתי לעשות הוא להקטין את גודל התמונות, בחרתי גודל שמצד אחד לא יהיה גדול מדי ומצד שני לא יפגע באיכות של התמונות והלכתי על גודל של x150150.

הדבר הראשון שעשיתי היה לשמור את התמונות המוקטנות בתיקייה בשם ResizedImages.

חשוב לציין לפני שממשיכים

תיקייה נוספת שתהיה בפרויקט היא תיקיית Images, הקטנת התמונות התבצעה בקוד בשלבים המוקדמים שלו ומכיוון שאין ביכולתי לאחסן את כל התמונות שהיו לפני ההקטנה (מדובר על כמה מאות של ג'יגה בייט) אני אראה PoC (Proof of Concept) על תמונות בודדות.

את התמונות המוקטנות אנחנו ניצור בתיקייה בשם ResizedImagesPOC (אין צורך ליצור את התיקייה באופן ידני) מכיוון שאני לא רוצה לדרוס את אלו שבResizedImages.

```
import data_organizer as dorg
dorg.resize_images_on_directory()
```

בכדי להריץ את הPOC

חשוב לציין שוב כי מדובר בPOC ותהליך ההקטנה המלא כבר קרה.

לאחר שהקטנתי את מאגר התמונות התחלתי למיין את התמונות לתיקיות train, test, validation ובכל תיקייה 4 תתי תיקיות שהן mask, unmasked, half_mask, bottom_mask.

השתמשתי בlabel של photo type מתוך המאגר של התמונות המוקטנות (לא שיניתי את שמותיהם של הקבצים לאחר ההקטנה) ויצרתי 4 רשימות בעבור כל אחד מהlabels.

לאחר שקיבצתי את כל התמונות לכל הרשימות עשיתי ערבול לכל אחת מן הרשימות בכדי שבהרצות שונות נוכל לקבל ערבולים שונים לtrain, test, validation.

לאחר מכן נתתי אפשרות באמצעות האפליקציה להעביר את האחוזים בהם רוצים לחלק את הdataset לtrain, test, validation כאשר הדרך הdefault היא 0.7 לtrain, 0.2 לtest, ו0.1 לvalidation.

חשוב לציין כי בעבור כמות אחוזים שעוברת את ה100 אחוז התכנית תזרוק שגיאה.

כעת יש לנו את כל הנתונים שצריך בשביל לחלק את המידע לתיקיות.

הדרך בה חילקתי את המידע היא באמצעות פונקציה שמקבלת את ארבעת הרשימות ואת האחוזים ששייך לאותו חלק שאנחנו עושים עליו את הפעולה (למשל בtrain היא תקבל את ארבעת הרשימות ו0.7).

הפעולה תיקח מכל אחד מן הרשימות את כמות הערכים שמתאימה לאחוזים שלה (למשל בעבור 100 תמונות עם מסכה, train שיש לו 70% תפוסה יקח 70 תמונות). בכדי למנוע מצב בו כמות התמונות לא מתחלקת בצורה שלמה למשל 70% מתוך 101 תמונות אנחנו ניקח math.floor של הכמות בכדי למנוע שגיאה.

כל ערך ברשימות של ארבעת הclasses הוא למעשה רשימה בגודל 2 שבמקום הראשון מכיל את שם התמונה ובמקום השני מכיל flag שאומר האם התמונה נלקחה כבר או לא.

בכל פעם שניקח תמונה נחליף את הflag במקום השני לTrue ובכך נדע בstep (הכוונה ל train, test, validation) איזה תמונה אפשר לקחת (מכיוון שהפעולות בעבור כל step נקראות אחד אחרי השנייה לא רוצים שיצא מצב שילקחו אותן תמונות). לאחר שכל step יקח את כמות התמונות שמתאימה לאחוזים שלו מכל רשימה הוא למעשה יקרא לפעולה אחרת שתיקח את רשימת שמות כל הקבצים שהstep לקח ותעתיק אותם מן התיקייה ResizedImages אל התיקייה המתאימה לclass המתאים בstep.

למשל רשימת הmasked בstep שהוא train תעתיק את כל התמונות ברשימה אל תיקייה בשם train/masked.

למה לחלק בכלל לתיקיות?

באופן עקרוני אפשר היה להסתדר גם בלי לחלק את זה לתיקיות בדיסק אבל חשוב להבין את היתרון הטמון בכן לחלק את המידע. התהליך המתואר למעלה הינו תהליך ארוך שיכול לקחת זמן ריצה יקר בכל

הרצה של תכנית. באמצעות החלוקה של המידע לתיקיות אנחנו למעשה חוסכים מעצמנו את ביצוע החלוקה הלוגית בכל הרצה מכיוון שכל מה שעלינו לעשות הוא לקרוא את הקבצים מתוך תיקיות קבועות במערכת.

קבלת המידע מן הדיסק:

לאחר שדיברנו על איך מחלקים את המידע על הדיסק, חשוב לדבר על איך אנחנו קוראים אותו מהדיסק. מכיוון שלא ביצענו חלוקה לוגית מתוך תיקייה אחת בכל הרצה מדובר על שני תהליכים שונים לגמרי וחסרי תלות מלבד העובדה שיש צורך שהתיקיות עם התמונות יהיו קיימות במערכת בשמות של הsteps ובשמות של הclasses.

תהליך קבלת המידע הינו זהה ונקרא שלוש פעמים (בעבור כל אחד מהsteps).

אנחנו מקבלים את שם הstep עליו אנחנו רוצים לקבל את המידע ואת הbatch_size.

הדבר הראשון שאנחנו עושים הוא לקרוא לתוך 4 רשימות (כל class מקבל רשימה משלו) את המידע הנמצא בתוך התיקייה.

לאחר מכן מה שאנחנו עושים זה לבדוק באיזה רשימה יש את הכמות הכי קטנה של תמונות, כלומר נניח ובחלוקת הקבצים יצא שלmasked יש 20 תמונות, לunmasked יש 19 תמונות, לhalf mask יש 21 תמונות ולbottom mask יש 30 תמונות. במקרה הזה נוצרת בעיה כי כל אחד מהמצבים יאומן בכמות שונה של תמונות לכן אנחנו לוקחים לפי הכמות הקטנה ביותר במקרה הזה ניקח מכל אחד מהם את ה19 תמונות הראשונות.

לאחר שסידרנו מצב בו יש לנו כמות זהה של תמונות מה שאנחנו נעשה זה לחבר את כל הרשימות לתוך רשימה אחת. מכיוון שאנחנו יודעים את הסדר שבו חיברנו את הרשימות ואת האורך של כל אחת מהן אנחנו ניצור רשימה חדשה שבה יהיה את הlabel בעבור כל אחת משמות הקבצים שקיבלנו.

לאחר שעשינו את זה נעשה ערבול לשתי הרשימות באותה צורה (כך שעדיין אינדקסים מקבילים יהיו קשורים אחד לשני) ולאחר מכן ניצור generator שיכיל תמונה טעונה וlabel.

```
@staticmethod
def __load_images_generator(li, batch_size):
    print(len(li))
    li = li[:len(li) - len(li) % batch_size]

    while True:
        for batch_offset in range(0, len(li), batch_size):
            x_list = []
            y_list = []

            for idx in range(batch_size):
                aligned_index = batch_offset + idx
                x_list.append(DataReceiver.__load_image_in_data_generator(li, aligned_index))
                y_list.append(li[aligned_index][1])

            x_list = np.array(x_list)
            y_list = np.array(to_categorical(y_list, dtype='uint8'))

            yield x_list, y_list
```

לאחר שביצענו את כל החלוקה הזו אנחנו ניצור generator שיעביר את התמונות לפי הbatch_size.

הסיבה ללמה אנחנו משתמשים בכלל בgenerator היא מכיוון שלטעון את כל התמונות בבת אחת מבעוד מועד ולהעביר אותם כפרמטרים לfit יכול להכביד מאוד על הRAM ובכך לגרום לקריסה של התכנית.

הgenerator מקבל רשימה שמכילה tuples שבמקום אחד יש את שם הקובץ ובמקום השני יש את הlabel שלו ואת הbatch_size שלפיו הוא עושה yield.

אנחנו מורידים חלק מהרשימה שלא מתחלקת בדיוק בכמות הbatch.

הסיבה ללולאה אינסופית היא מכיוון שאחרי שהgenerator יגמר במהלך האימון אנחנו רוצים שהוא יתחיל מההתחלה ולא יזרוק לנו שגיאה שאין לו יותר מה להעביר.

בכל לולאת for אנחנו יוצרים שני רשימות בגודל batch_size שאחת מהן מכילה את התמונה שטענו והשנייה מכילה את הlabel של התמונה באינדקסים מקבילים. בסוף כל לולאת for כלומר לאחר כל חישוב של batch אנחנו שולחים את הרשימות שיצרנו.

בתהליך יצירת הgenerator נטען כל תמונה באמצעות openCV ונעשה נרמול לתמונה עם בכך שנחלק אותה ב255:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

במקרה שלנו הx(min) הוא 0 והx(max) הוא 255

כלומר בעבור כל x יצא כאילו אנחנו מחלקים אותו פשוט ב255.

מדובר באחת משיטות הנרמול המשומשות והמוכרת ביותר.

מכיוון שהמקסימום שיכול להיות לצבע בכל אחד מהcolor channels הוא 255 אז בהכרח כאשר נחלק המספרים יהיו בתחום הערכים שבין 0-1.

בחזרה מהפונקציה נחזיר שני ערכים, הערך הראשון יהיה הgenerator ואילו הערך השני יהיה ה(validation_steps/steps_per_epoch/test_steps) בכדי שנדע כמה פעמים הוא צריך לקבל batch לפי epoch. החישוב נתון על ידי הנוסחה len(list) / batch_size.

שלב בנייה ואימון המודל

תיאור גרפי של המודל:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 16)      448

 max_pooling2d (MaxPooling2D  (None, 74, 74, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 dropout (Dropout)           (None, 7, 7, 64)          0

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 4)                 12548

=================================================================
Total params: 45,380
Trainable params: 45,380
Non-trainable params: 0
```

מילון מושגים:

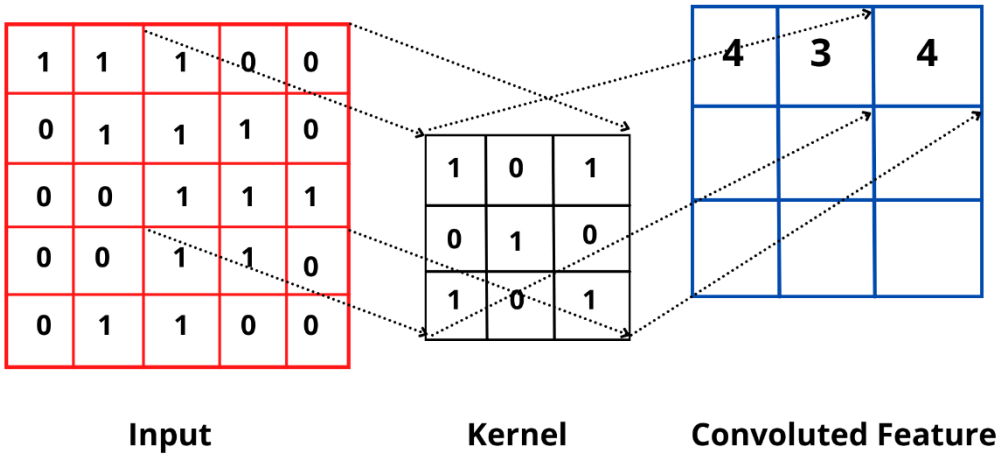
Conv2D

קונבולוציה הוא נושא שמוכר גם בתחום הimage processing בנושאים כמו טשטוש תמונה למשל.

הConv2D הוא שכבת קונבולוציה שבעצם נותן לנו את האופציה לבצע את הקונבולוציה על מערכים דו מימדיים או במקרה שלנו תמונות.

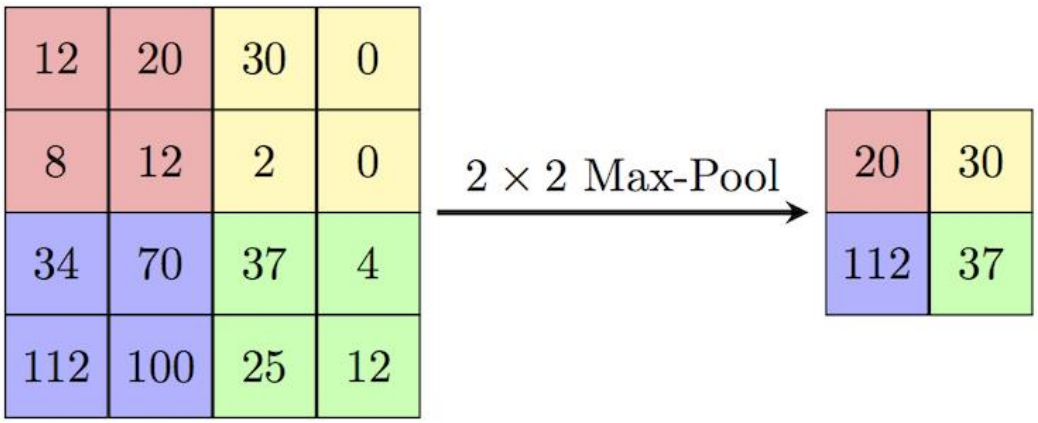
מה שקורה בעצם זה שאנחנו קובעים את כמות הפילטרים בארגומנט הראשון ואת גודל הפילטר במודל השני, המודל מנסה להריץ פילטרים בכדי ללמוד מהתמונה דברים למשל למצוא קצוות.

מה שקורה בפועל זה שאנחנו עוברים עם מטריצה של פילטר על פיקסלים בתמונה ואנחנו מכפילים בין המטריצות ומחברים.



MaxPooling2D

הMaxPooling עובר על התמונה עם הpool size שהוא קיבל ומתוך כל מטרציה (במקרה שלנו) של x22 הוא לוקח את הערך הכי גדול

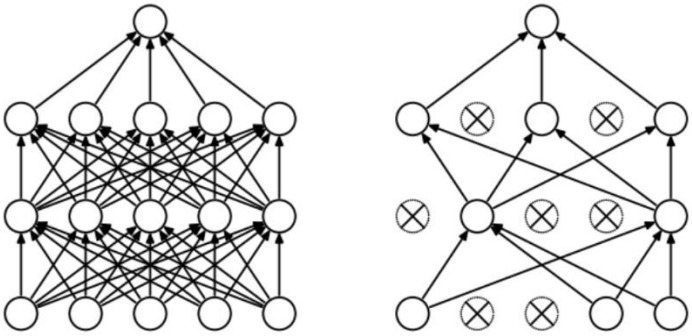


כמו שאפשר לראות בתמונה למעלה, הMaxPooling עובר על 12,20,8,12 הוא רואה ש20 הכי גדול והוא לוקח רק אותו (הוא עושה אותו דבר בעבור 30,2,0,0 וגם 34,70,112,100 וגם 37,4,25,12).

Dropout

הDropout בוחר בצורה רנדומלית נוירונים ומבטל אותם (מתעלם מהם) ומגדיל את הנוירונים שלא בוטלו בכדי שהסכום ישאר זהה.

מטרתו העיקרית של הDropout הוא לבטל **overfitting** שזה מקרה בו מאמנים על כמות קטנה של מידע והמודל יודע לזהות את הדוגמאות שהוא התאמן עליהם אבל לא מצליח לזהות בהצלחה דוגמאות שונות.

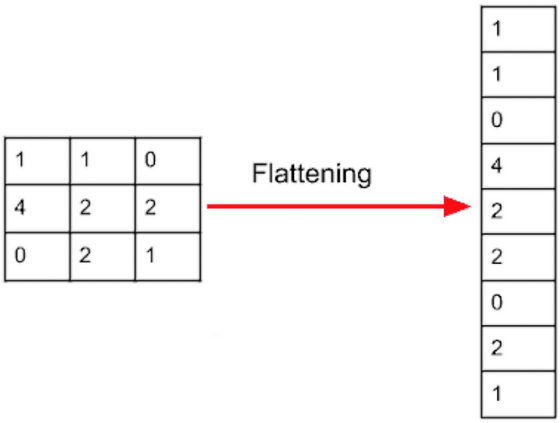


(a) Standard Neural Net (b) After applying dropout.

Flatten

פעולת Flatten למעשה הופכת את המערך לחד-מימדי.

לצורך הקלסיפיקציה אנחנו לא יכולים לתת output של ערך דו מימדי ולכן אנחנו עושים flatten לפני השכבה האחרונה (שכבת הoutput),



הסבר על סוגי השכבות השונים ברשת:

שכבה ראשונה

Input Shape	(3 ,150 ,150)
Neurons	16
Filter	3,3
Activation Function	Relu

שכבה שנייה

Neurons	32
Filter	3,3
Activation Function	Relu

שכבה שלישית

Neurons	32
Filter	3,3
Activation Function	Relu

שכבה רביעית

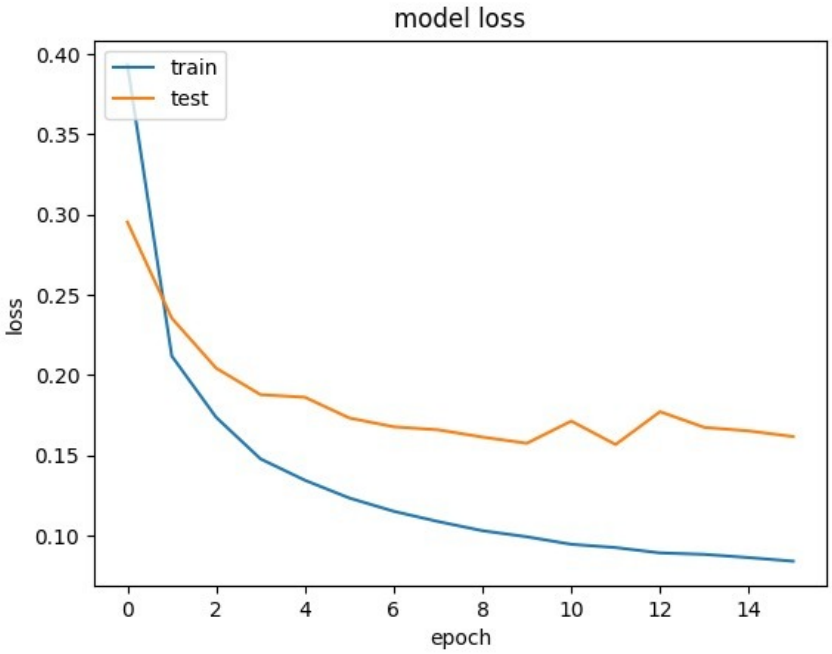
Neurons	64
Filter	3,3
Activation Function	Relu

שכבה חמישית

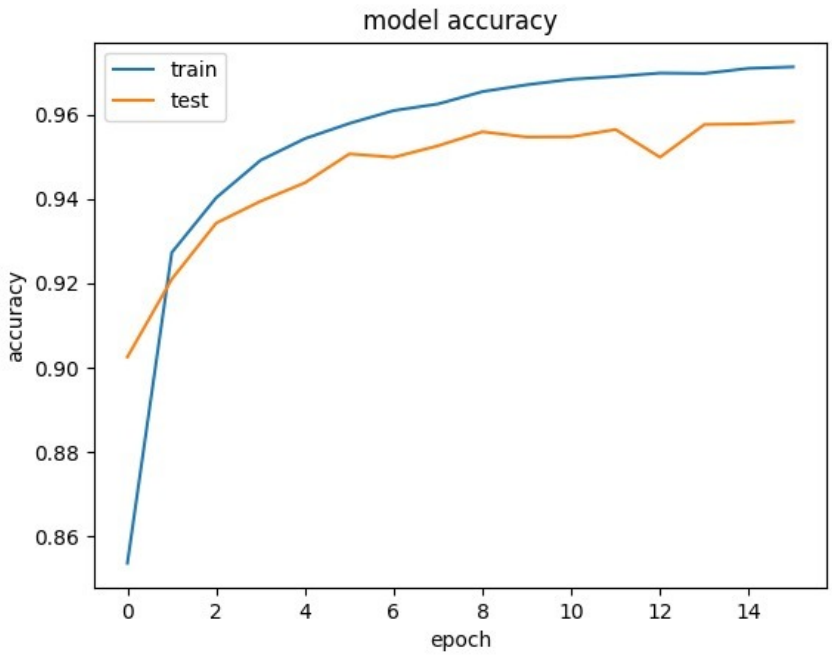
Dense	4
Activation Function	Softmax

דוחות וגרפים המתארים את תוצאות שלב האימון

גרף המראה את גודל השגיאה בvalidation ובtrain לפי כמות הepoch



גרף המראה את גודל הדיוק בvalidation ובtrain לפי כמות הepoch



```
105620
Epoch 1/16
3300/3300 [==============================] - ETA: 0s - loss: 0.3932 - accuracy: 0.853715092

Epoch 1: saving model to saved-model-01-0.30.hdf5

Epoch 1: val_loss improved from inf to 0.29539, saving model to best-saved-model-01-0.30.hdf5
3300/3300 [==============================] - 25233s 8s/step - loss: 0.3932 - accuracy: 0.8537 - val_loss: 0.2954 - val_accuracy: 0.9025
Epoch 2/16
3300/3300 [==============================] - ETA: 0s - loss: 0.2119 - accuracy: 0.9273
Epoch 2: saving model to saved-model-02-0.24.hdf5

Epoch 2: val_loss improved from 0.29539 to 0.23548, saving model to best-saved-model-02-0.24.hdf5
3300/3300 [==============================] - 826s 250ms/step - loss: 0.2119 - accuracy: 0.9273 - val_loss: 0.2355 - val_accuracy: 0.9210
Epoch 3/16
3300/3300 [==============================] - ETA: 0s - loss: 0.1737 - accuracy: 0.9402
Epoch 3: saving model to saved-model-03-0.20.hdf5

Epoch 3: val_loss improved from 0.23548 to 0.20435, saving model to best-saved-model-03-0.20.hdf5
3300/3300 [==============================] - 29352s 9s/step - loss: 0.1737 - accuracy: 0.9402 - val_loss: 0.2043 - val_accuracy: 0.9342
Epoch 4/16
3300/3300 [==============================] - ETA: 0s - loss: 0.1478 - accuracy: 0.9491
Epoch 4: saving model to saved-model-04-0.19.hdf5

Epoch 4: val_loss improved from 0.20435 to 0.18785, saving model to best-saved-model-04-0.19.hdf5
3300/3300 [==============================] - 17289s 5s/step - loss: 0.1478 - accuracy: 0.9491 - val_loss: 0.1878 - val_accuracy: 0.9394
Epoch 5/16
3300/3300 [==============================] - ETA: 0s - loss: 0.1345 - accuracy: 0.9543
Epoch 5: saving model to saved-model-05-0.19.hdf5

Epoch 5: val_loss improved from 0.18785 to 0.18623, saving model to best-saved-model-05-0.19.hdf5
3300/3300 [==============================] - 70303s 21s/step - loss: 0.1345 - accuracy: 0.9543 - val_loss: 0.1862 - val_accuracy: 0.9438
Epoch 6/16
3300/3300 [==============================] - ETA: 0s - loss: 0.1235 - accuracy: 0.9579
Epoch 6: saving model to saved-model-06-0.17.hdf5

Epoch 6: val_loss improved from 0.18623 to 0.17331, saving model to best-saved-model-06-0.17.hdf5
3300/3300 [==============================] - 20619s 6s/step - loss: 0.1235 - accuracy: 0.9579 - val_loss: 0.1733 - val_accuracy: 0.9506
Epoch 7/16
3300/3300 [==============================] - ETA: 0s - loss: 0.1152 - accuracy: 0.9609
```

```
3300/3300 [==============================] - ETA: 0s - loss: 0.1152 - accuracy: 0.9609
Epoch 7: saving model to saved-model-07-0.17.hdf5

Epoch 7: val_loss improved from 0.17331 to 0.16783, saving model to best-saved-model-07-0.17.hdf5
3300/3300 [==============================] - 2027s 614ms/step - loss: 0.1152 - accuracy: 0.9609 - val_loss: 0.1678 - val_accuracy: 0.9498
Epoch 8/16
3300/3300 [==============================] - ETA: 0s - loss: 0.1088 - accuracy: 0.9625
Epoch 8: saving model to saved-model-08-0.17.hdf5

Epoch 8: val_loss improved from 0.16783 to 0.16596, saving model to best-saved-model-08-0.17.hdf5
3300/3300 [==============================] - 942s 286ms/step - loss: 0.1088 - accuracy: 0.9625 - val_loss: 0.1660 - val_accuracy: 0.9526
Epoch 9/16
3300/3300 [==============================] - ETA: 0s - loss: 0.1031 - accuracy: 0.9654
Epoch 9: saving model to saved-model-09-0.16.hdf5

Epoch 9: val_loss improved from 0.16596 to 0.16142, saving model to best-saved-model-09-0.16.hdf5
3300/3300 [==============================] - 1094s 332ms/step - loss: 0.1031 - accuracy: 0.9654 - val_loss: 0.1614 - val_accuracy: 0.9559
Epoch 10/16
3300/3300 [==============================] - ETA: 0s - loss: 0.0993 - accuracy: 0.9670
Epoch 10: saving model to saved-model-10-0.16.hdf5

Epoch 10: val_loss improved from 0.16142 to 0.15758, saving model to best-saved-model-10-0.16.hdf5
3300/3300 [==============================] - 971s 294ms/step - loss: 0.0993 - accuracy: 0.9670 - val_loss: 0.1576 - val_accuracy: 0.9546
Epoch 11/16
3300/3300 [==============================] - ETA: 0s - loss: 0.0947 - accuracy: 0.9683
Epoch 11: saving model to saved-model-11-0.17.hdf5

Epoch 11: val_loss did not improve from 0.15758
3300/3300 [==============================] - 69886s 21s/step - loss: 0.0947 - accuracy: 0.9683 - val_loss: 0.1714 - val_accuracy: 0.9547
Epoch 12/16
3300/3300 [==============================] - ETA: 0s - loss: 0.0926 - accuracy: 0.9690
Epoch 12: saving model to saved-model-12-0.16.hdf5

Epoch 12: val_loss improved from 0.15758 to 0.15681, saving model to best-saved-model-12-0.16.hdf5
3300/3300 [==============================] - 1917s 581ms/step - loss: 0.0926 - accuracy: 0.9690 - val_loss: 0.1568 - val_accuracy: 0.9564
Epoch 13/16
3300/3300 [==============================] - ETA: 0s - loss: 0.0893 - accuracy: 0.9698
Epoch 13: saving model to saved-model-13-0.18.hdf5
```

```
Epoch 13: val_loss did not improve from 0.15681
3300/3300 [==============================] - 1497s 454ms/step - loss: 0.0893 - accuracy: 0.9698 - val_loss: 0.1772 - val_accuracy: 0.9498
Epoch 14/16
3300/3300 [==============================] - ETA: 0s - loss: 0.0884 - accuracy: 0.9697
Epoch 14: saving model to saved-model-14-0.17.hdf5

Epoch 14: val_loss did not improve from 0.15681
3300/3300 [==============================] - 1060s 321ms/step - loss: 0.0884 - accuracy: 0.9697 - val_loss: 0.1674 - val_accuracy: 0.9576
Epoch 15/16
3300/3300 [==============================] - ETA: 0s - loss: 0.0864 - accuracy: 0.9709
Epoch 15: saving model to saved-model-15-0.17.hdf5

Epoch 15: val_loss did not improve from 0.15681
3300/3300 [==============================] - 1016s 308ms/step - loss: 0.0864 - accuracy: 0.9709 - val_loss: 0.1653 - val_accuracy: 0.9577
Epoch 16/16
3300/3300 [==============================] - ETA: 0s - loss: 0.0841 - accuracy: 0.9712
Epoch 16: saving model to saved-model-16-0.16.hdf5

Epoch 16: val_loss did not improve from 0.15681
3300/3300 [==============================] - 951s 288ms/step - loss: 0.0841 - accuracy: 0.9712 - val_loss: 0.1617 - val_accuracy: 0.9583
```

תוצאות הtest של המודל:

```
30176
943/943 [==============================] - 246s 261ms/step - loss: 0.1226 - accuracy: 0.9632
```

ריכוז הHyperparameters:

Hyperparameter Name	Value
Number Of Hidden Layers	3
Dropout	0.4
Activation Functions	Relu, Relu, Relu, Relu, Softmax
Weights initialization	glorot_uniform
Learning Rate	0.001
Epoch, iterations and batch size	Epoch – 16, batch_size – 32, iterations – 3300
Optimizer Algorithm	Adam
Loss function	CategorialCrossentropy

תיעוד והסבר של פונקציית השגיאה:

```
loss=tf.keras.losses.CategoricalCrossentropy(),
```

מטרתה של פונקציית שגיאה היא לחשב את המרחק בין התשובה של המודל לתשובה האמיתית וככל שערך השגיאה שהתקבל קטן יותר אזי שהמרחק בין התוצאות קטן יותר. כלומר, ככל שהמרחק קטן יותר אנחנו קרובים יותר לתשובה האמיתית ולכן המודל שלנו טוב יותר.

פונקציית שגיאה שאנו משתמשים בה בפרויקט מיועדת לשימוש בmulticlass classification כלומר בסיווג בו יש כמה מחלקות שבינהן אנו מסווגים (למשל כמו במקרה שלנו ארבעה מחלקות שונות עם מסכה, בלי מסכה, חצי מסכה, מסכה על הסנטר).

from_logits=True כאשר from_logits=True זה אומר לפונקציית שגיאה שהערכים אינם מנורמלים, כלומר כאשר אנחנו משתמשים בfrom_logits=True אנחנו לא משתמשים בפונקציית softmax במודל עצמו (אנחנו משתמשים בfrom_logits=False).

והlayer האחרון יהיה:

Keras.layers.add(Dense(NB_Classes))

בפרויקט אנחנו משתמשים בפונקציית השגיאה CategoricalCrossentropy.

בכדי להשתמש בפונקציית השגיאה הזו אנחנו חייבים להמיר את הlabels שלנו להיות one-hot encoded באמצעות to_categorial.

כלומר במקום שהlabels שלנו יהיו מספרים שלמים הם יוצגו באמצעות מטריצות.

למשל בעבור ה0,1,2,3 labels

יהיה הצגה של:

0 -> [1,0,0,0]

1 -> [0,1,0,0]

2 -> [0,0,1,0]

3 -> [0,0,0,1]

פונקציית השגיאה של CategoricalCrossentropy מחושבת על ידי הנוסחה הבאה:

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

y(i) – הערך שציפינו שיתקבל

y^(i) – הערך שהתקבל

Output size – כמות הclasses.

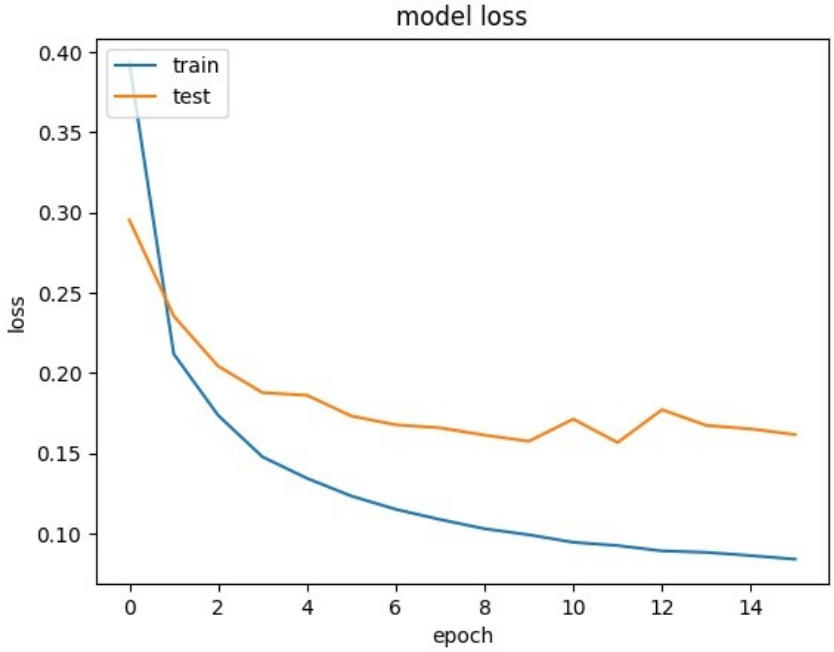
ניתו דוגמה לחישוב הloss:

בעבור תוצאת מכונה [0.05,0.05,0.9]

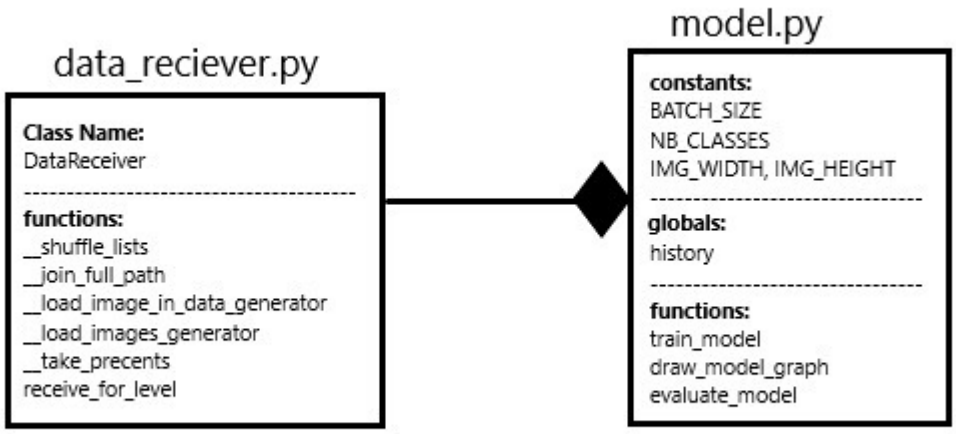
בעבור תוצאה אמיתית [0,0,1]

log(0.05) + 0 * log(0.05) + 1 * log(0.9) = -Loss * 0

Loss = 0.04



תרשים UML של המחלקות הממשות את המודל

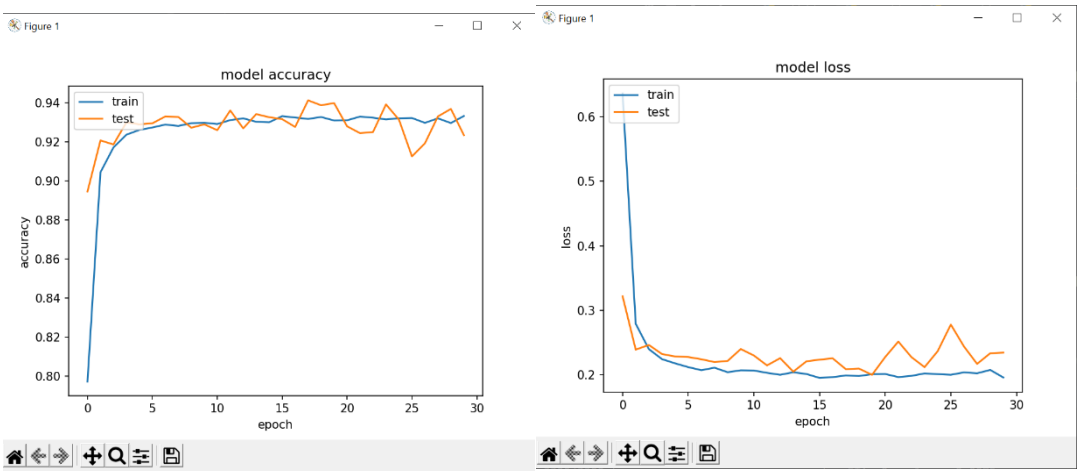


*בדוגמה זו התייחסנו לשימוש בmodule כ"הכלה"

תיעוד כל השינוים שנעשו בhyperparameters לשיפור תוצאת האימון

```
train, validation = DataReceiver.receive_for_level('train', BATCH_SIZE), DataReceiver.receive_for_level('validation', BATCH_SIZE)
model = models.Sequential()
model.add(layers.Conv2D(16, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)))
model.add(layers.MaxPooling2D((2, 2)))
# model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.1))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.1))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))
model.add(layers.Dense(NB_CLASSES, activation='softmax'))
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

המודל הראשון שהרצנו עליו



תוצאות הגרפים על המודל שהרצנו

בעת הרצת המודל על הטסט התקבל loss גבוה של 0.22 ולכן החלטתי שלא להשתמש במודל הזה.

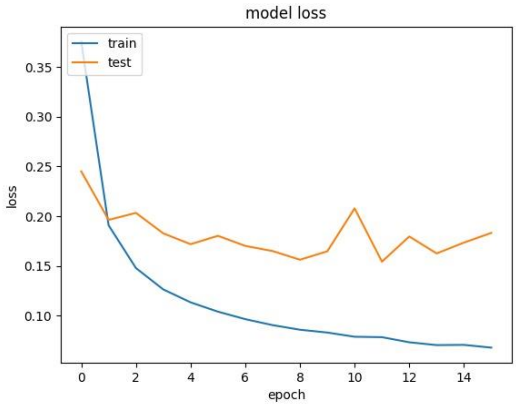
- חשוב לציין שהtest בגרפים הוא למעשה הtest בזמן אימון המודל כלומר הכוונה היא לvalidation!

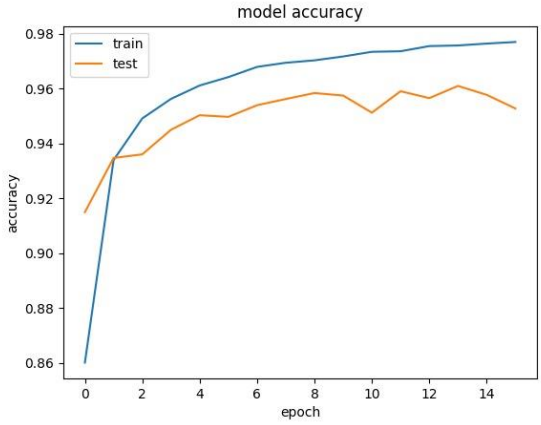
Hyperparameters Table

Hyperparameter Name	Value
Number Of Hidden Layers	3
Dropout	0.2 ,0.3 ,0.1 ,0.1
Activation Functions	Relu, Relu, Relu, Relu, Softmax
Weights initialization	glorot_uniform
Learning Rate	0.001
Epoch, iterations and batch size	Epoch – 30, batch_size – 32, iterations – 3300
Optimizer Algorithm	Adam
Loss function	SparseCategorialCrossentropy

```
train, validation = DataReceiver.receive_for_level('train', BATCH_SIZE), DataReceiver.receive_for_level('validation', BATCH_SIZE)
model = models.Sequential()
model.add(layers.Conv2D(16, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)))
model.add(layers.MaxPooling2D((2, 2)))
# model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
#model.add(layers.Dropout(0.1))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
#model.add(layers.Dropout(0.1))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
#model.add(layers.Conv2D(128, (3, 3), activation='relu'))
#model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))
model.add(layers.Flatten())
#model.add(layers.Dropout(0.2))
model.add(layers.Dense(NB_CLASSES, activation='softmax'))
model.compile(optimizer='adam',
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=['accuracy'])
```

המודל השני שהרצנו עליו





תוצאות הגרפים על המודל שהרצנו

תוצאות הטסט על המודל היו 0.14 שגיאה ו95 אחוז דיוק. החלטתי לא להשתמש במודל בגלל גרף השגיאה.

- חשוב לציין שהtest בגרפים הוא למעשה הtest בזמן אימון המודל כלומר הכוונה היא לvalidation!

Hyperparameters Table

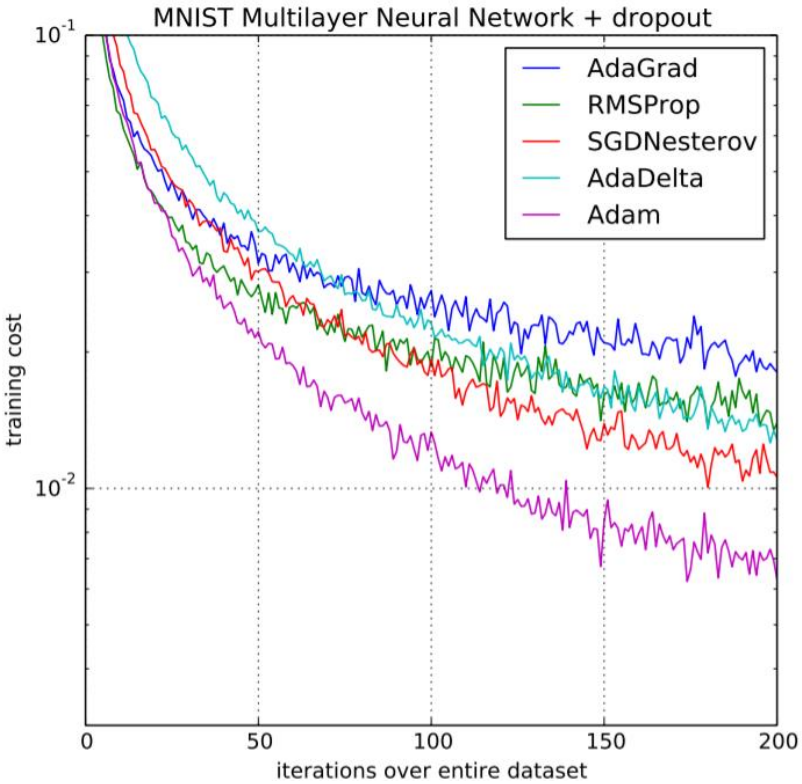
Hyperparameter Name	Value
Number Of Hidden Layers	3
Dropout	0.3
Activation Functions	Relu, Relu, Relu, Relu, Softmax
Weights initialization	glorot_uniform
Learning Rate	0.001
Epoch, iterations and batch size	Epoch – 16, batch_size – 32, iterations – 3300
Optimizer Algorithm	Adam
Loss function	CategorialCrossentropy

הסבר שיטת הייעול Adam Optimizer

מטרתם של אופטימייזרים בכללי הוא להוריד את הloss של המודל על ידי שינוי תכונות, למשל שינוי המשקלים, או שינוי קצב הלמידה. הדרך בה התכונות משתנות תלוי באופטימייזר בו אנחנו משתמשים.

Adam נחשב לאחד הoptimizers הפופלאריים ביותר בגלל היותו חסכני, יעיל ומדויק ובגלל ביצועי הטובים על מאגרי מידע גדולים.

Adam הוא למעשה שילוב של שני optimizers שונות AdaGrad וRMSProp.



אפשר לראות בגרף זה בו קיימת ההשוואה בין האופטימייזרים השונים שלAdam יש את הloss הכי נמוך כלומר נותן את הביצועים הטובים ביותר.

שלב היישום

תיאור והסבר כיצד היישום משתמש במודל:

החלק הראשון ביישום הוא בחירת המודל. היישום שואל את המשתמש באיזה מודל הוא יהיה מעוניין להשתמש.

כל משתמש מקבל מודל ברירת מחדל שעליו הוא יכול לבצע את כל הבדיקות. אך עם זאת, כל משתמש מקבל גם אופציה לאמן את המודל מחדש ולארגן את כל המידע.

לאחר שהמשתמש מקבל את המודל שבו הוא מעוניין על המשתמש לבצע בדיקות על המודל.

למשתמש ישנם כמה דרכים לבצע בדיקות:

Model Evaluation

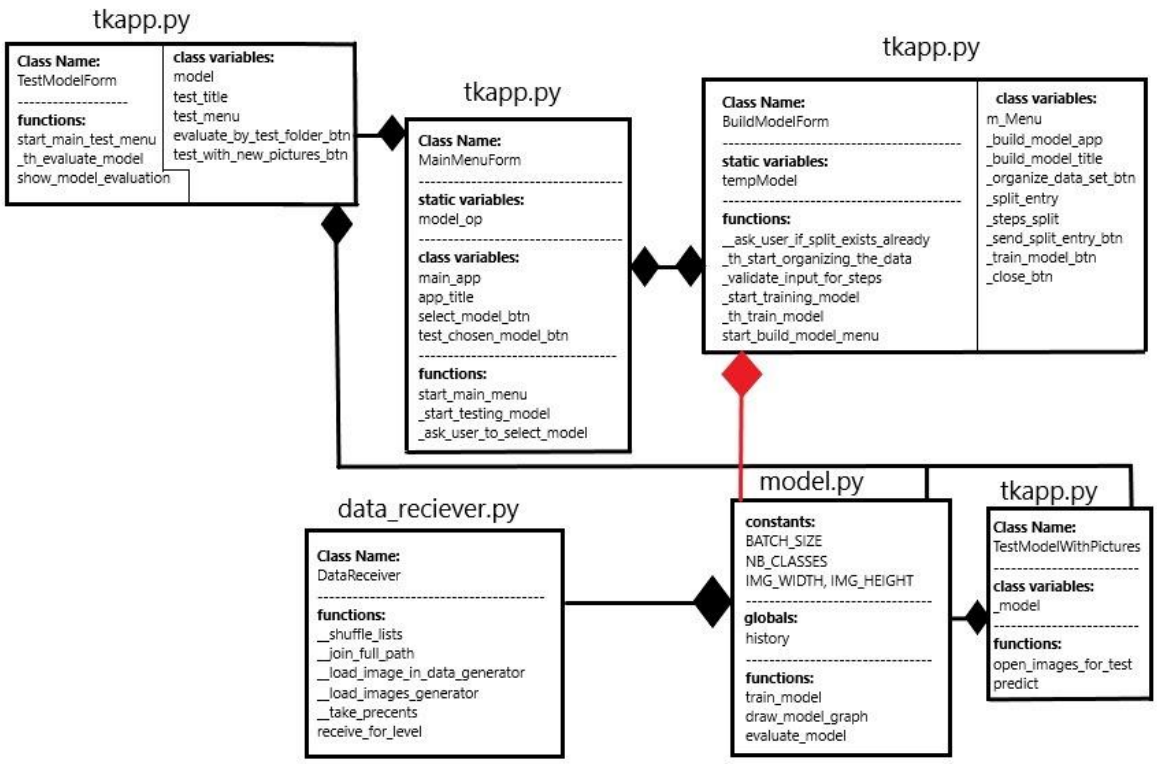
חלק זה מאפשר לנו לקבל מידע בסיסי על המודל עוד לפני האימון (חשוב לציין כי אין חובה להשתמש בחלק זה). חלק זה מחזיר לנו מידע אודות המודל. המידע שמוחזר מכיל את רמת הדיוק של המודל וגודל השגיאה שלו. את הבדיקה הוא מבצע על תיקיית הtest בדיסק שנוצרה בתהליך מיון המידע שמפורט בחלק של **"שלב איסוף הכנה וניתוח הנתונים".**

New images test

דרך נוספת לבדוק את המודל שלנו היא באמצעות תמונות חדשות. המשתמש יכול להעביר קבצי png, jpeg, jpg בכדי לבדוק את איכות המודל עליהם. כל שם קובץ שמתקבל נקרא באמצעות openCV ומשתנה לגודל קבוע של x150150 פיקסלים (שעליהם המודל אומן).

לאחר מכן יוצג למשתמש מעין Linear Linked List של תמונות כלומר, כל תמונה שהוא סוגר מקדמת אותו תמונה אחת קדימה. התג שזוהה בעבור כל תמונה מתקבל כחלק מהתמונה כלומר את הlabel שהמכונה זיהתה אנחנו רושמים על התמונה ומציגים את התמונה ביחד עם הlabel שרשום עליה.

תרשים הUML:



*חשוב לציין כי התייחסנו לשימוש במודולים כ"הכלה".

המחלקות שנמצאות תחת הסקריפט tkapp.py הן המחלקות שיוצרות את ממשק המשתמש.

המודולים האחרים משומשים בממשק המשתמש גם כן כדי לבצע את הפונקציונליות מאחורי הקלעים ובכדי להראות את הלוגים במסך ההרצה.

תיאור הטכנולוגיה שעל פיה מומש ממשק המשתמש:

הטכנולוגיה שבה השתמשתי בכדי לממש את ממשק המשתמש הינה tkinter.

tkinter הינה ספרייה סטנדרטית במערכות ההפעלה המובילות למחשבים אישיים Linux, Windows, Mac. והיא משמשת בכדי ליצור אפליקציות GUI בפייתון.

מכיוון שמדובר בספרייה סטנדרטית היא אמורה לבוא כבר עם ההתקנה של פייתון ולכן אין צורך להתקין אותה באופן ידני אך במידה ומשום מה היא איננה מופיעה כאחת מן הספריות אפשר להתקין אותה בקלות אמצעות pip install tk.

תיאור קוד הקולט את ה-DATA שעליו יבוצע החיזוי והתאמתו למבנה נתונים המתאים לחיזוי:

```python
# called first to select the images the user wants to pick.
def open_images_for_test(self):
    file_names = askopenfilenames()  # the file name we are willing to use.

    # we first validate our inputs.
    for _file_name in file_names:
        f_ext = os.path.splitext(_file_name)[1]  # the file extension
        if f_ext.lower() not in ['.jpg', '.png', '.jpeg']:
            mbox.showerror(title='Invalid Input',
                           message=f'.jpg was expected but {f_ext} was the input.')
            return

    for _name in file_names:
        cv2_img = cv2.imread(_name)
        cv2_img = cv2.resize(cv2_img, (150, 150))
        cv2_img = cv2_img / 255
        predicted_label = ModelSupport.predict(self._model, cv2_img)
        cv2.putText(img=cv2_img, text=predicted_label, org=(0, 30), fontFace=cv2.FONT_HERSHEY_TRIPLEX,
                    fontScale=0.8, color=(0, 255, 0), thickness=1)

        cv2.imshow(_name, cv2_img)
        cv2.waitKey(0)
```

החלק שקשור בקליטת מידע מן המשתמש הוא חלק משלב היישום אשר מאפשר לקבל תמונות חדשות מהמשתמש ולבדוק את המודל עליהן.

בניגוד לשאר חלקי האפליקציה החלק שקשור בקליטת המידע אינו כולל **חלון אפליקציה** מיוחד.

בחלק הראשון אנחנו משתמשים בפונקציה מתוך tkinter.filedialog אשר מאפשרת לנו לפתוח **חלון קבצים** שממנו אפשר לבחור את הקבצים עליהם אנחנו רוצים לבדוק את המודל. הדבר הראשון שאנחנו עושים בחלק זה הוא בדיקה של סיומות הקבצים ובכך אנחנו מוודאים כי הקובץ שהוכנס הוא אכן קובץ תמונה. במידה ואחד מן הקבצים הוא קובץ לא תקין אנחנו מחזירים שגיאה וחוזרים מן הפונקציה.

במידה וכל הקבצים הם קבצים תקינים אנחנו עוברים על כל הקבצים ובעבור כל קובץ:

- פותחים אותו עם openCV.

- משנים את הגודל שלו לגודל קבוע של x150150 שזה הגודל שהמודל אומן עליו.
- מנרמלים את התמונה.
- שולחים את הקובץ למודל ומקבלים בחזרה את הlabel שהמודל חושב ששייך לתמונה.
- רושמים את הlabel שהמודל נתן על הקובץ בצבע ירוק
- מראים את התמונה למשתמש ומחכים שהמשתמש יחליט לקבל את התמונה הבאה שתעבור את אותו התהליך בכך שיסגור את חלון התמונה שפת

מדריך למפתח

data_receiver.py

מדובר על הscript שאחראי על תהליך קריאת המידע מן הדיסק.

המודול מכיל מחלקה בשם DataReceiver בה יש פעולות סטטיות שמטרתן לעזור לטעון את המידע.

רוב הפעולות במחלקה הן פעולות פרטיות (כלומר לא **אמורים** להשתמש בהן בשימוש מחוץ למחלקה).

שם הפונקציה	שמות הפרמטרים שמקבלת ותפקידם	מטרתה של הפונקציה
receive_for_level	**Function arguments:** **level_name –** שם השלב שעליו אנחנו רוצים לקבל את המידע, יכול לקבל train, test, validation. **batch_size –** גודל הbatch שלפיו נחלק את הgenerator. **Function variables:** **lvl** – רשימה המכילה את כל ארבעת הרשימות לאחר שחלקן הורידו חלק מן התמונות. **mask_files, unmasked_files, half_mask_files, no_mask_files** – מכילות את שמות הקבצים בעבור כל אחד מהמצבים לאותו level שקיבלנו כארגומנט.	מטרתה של הפונקציה היא להכין את המידע ההכרחי ליצירת הgenerator ולהחזיר אותו למשתמש ביחד עם כמות הsteps_per_level. הפונקציה טוענת את שמות הקבצים לארבעת הרשימות שמייצגות את המחלקות ובודקת איזה רשימה היא הכי קצרה (אנחנו רוצים שיהיה כמות זהה בכל אחד מהמצבים) לאחר מכן אנחנו לוקחים מכל רשימה את אותו המספר בדיוק. לאחר מכן אנחנו מחברים את כל הרשימות אל רשימה אחת lvl ויוצרים רשימת תגיות מקבילה בכך שאנחנו לוקחים את האורך של lvl מחלקים אותו ב4 (ארבעה מצבים) ואז יוצרים רשימות בגודל האורך לפי המספרים שמייצגים את התגיות. לאחר מכן שיש לנו את רשימת שמות הקבצים ורשימת התגיות כאשר כל האינדקסים מקבילים (כלומר התגית במקום הראשון ברשימת התגיות מתאים לשם הקובץ במקום הראשון ברשימת שמות הקבצים). לאחר מכן הפונקציה מאחדת בין שני הרשימות אל רשימה אחת, מערבלת אותה ושולחת אותה לפונקציית הgenerator ביחד עם הbatch_size.
__load_images_generator	**Function arguments:**	הסבר מעמיק על פונקציית הgenerator אפשר למצוא בחלק הארכיטקטורה בספר.

	li – הרשימה המתקבלת היא רשימה שמכילה tuples כאשר במקום האפס יש את שם הקובץ ובמקום הראשון יש את התגית שלו. **batch_size** – גודל הbatch שלפיו אנחנו מחלקים את המידע.	
load_image_in_data_generator_	**Function arguments:** **li** – הרשימה המתקבלת היא רשימה שמכילה tuples כאשר במקום האפס יש את שם הקובץ ובמקום הראשון יש את התגית שלו. **i –** האינדקס ברשימה שאותו אנו רוצים לטעון בעבור הgenerator.	הפונקציה מקבלת אינדקס של ערך מהרשימה, טוענת את התמונה שבשם הקובץ מנרמלת את התמונה ומחזירה אותה.
join_full_path__	**Function arguments:** **li –** רשימת שמות קבצים **path –** נתיב שרוצים לשרשר	מטרתה של הפעולה היא לשרשר את path לכל הstrings ברשימה li.

data_organizer.py

סט הפעולות תחת הסקריפט data_organizer.py הוא הסקריפט העיקרי שאחראי על ארגון המידע על הדיסק.

שם הפונקציה	משתנים	מטרת הפעולה
build_directories	אין	הפעולה בונה את כל התיקיות שצריך (כתיקיות ריקות). כלומר בונה תיקיות train, test, validation שבכל אחד יש תיקיות mask, unmasked, half_mask, bottom_mask. הפונקציה משתמשת לצורך הבנייה בפונקציה override_if_exists__ שדורסת את התיקייה אם קיימת.
override_if_exists_	**Function arguments:** **directory** – שמה של התיקייה	הפעולה מקבלת שם של תיקייה:

		אם התיקייה קיימת היא דורסת אותה ויוצרת תיקייה ריקה במקומה. אם התקייה לא קיימת היא יוצרת אותה.
resize_images_on_directory	אין	הפעולה לוקחת את כל התמונות מתוך תיקייה בשם Images, משנה את גודלן של התמונות ל150x150 באמצעות הפעולה **resize_image** מתוך המודול **image_processor.py** אל תוך התיקייה ResizedImages.
divide_to_train_test_validation	**Function arguments:** **train** – כמה אחוזים לשלב הtrain (ברירת מחדל 0.7). **test** – כמה אחוזים לשלב הtest (ברירת מחדל 0.2). **validation** – כמה אחוזים לשלב הvalidation (ברירת מחדל 0.1). **Function variables:** **mask** – רשימה המכילה את כל שמות הקבצים של האנשים עם מסכה והFlag שמציין אם התמונה נלקחה או לא (יפורט בהמשך). **half_mask**– רשימה המכילה את כל שמות הקבצים של האנשים עם מסכה על הפה והFlag שמציין אם התמונה נלקחה או לא (יפורט בהמשך). **bottom_mask** – רשימה המכילה את כל שמות הקבצים של האנשים עם מסכה על הסנטר והFlag שמציין אם התמונה נלקחה או לא (יפורט בהמשך). **no_mask** – רשימה המכילה את כל שמות הקבצים של האנשים ללא מסכה והFlag שמציין אם התמונה נלקחה או לא (יפורט בהמשך).	הפונקציה מהווה מעטפת לפונקציות אחרות ומטרתה הכוללת היא לחלק את המידע לתיקיות בחלוקה לפי האחוזים שהתקבלו בפעולה. הפעולה קוראת בהתחלה ל__split_mask_no_mask (ניתן לצפות בהסבר בטבלה) בכדי לקבל את הרשימות לפי המחלקות ולאחר מכן קוראת שלוש פעמים לפעולה __get_data_for_each_steps (כל פעם בעבור כל step) ומטרתה של פעולה זו היא לחלק את המידע לתיקיות.
__split_mask_no_mask	אין	הפונקציה קוראת את כל הקבצים מתוך התיקייה ResizedImages ומחלקת אותם לארבעה רשימות לפי mask, unmasked,

		bottom_mask, half_mask לפי שמות הקבצים. הערך שמוסף לרשימה אינו רק שם הקובץ אלא שם הקובץ וflag שערכו False שמטרתו בהמשך ליידע אם התמונה נלקחה כבר על ידי step אחר (הסבר מפורט יותר לגבי הflag יהיה בפעולה take_from_list__).
get_data_for_each_step__s	**:Function arguments** **mask** – רשימת שמות הקבצים שבהם האדם בתמונה עם מסכה. **no_mask** – רשימת שמות הקבצים שבהם האדם בתמונה בלי מסכה. **half_mask** – רשימת שמות הקבצים שבהם האדם בתמונה עם מסכה על הפה. **bottom_mask** – רשימת שמות הקבצים שבהם האדם בתמונה עם מסכה על הסנטר. **precent** – האחוזים בעבור אותו שלב. **step_name** – שם השלב (train, test, validation). **:Function variables** **_mask** – רשימת שמות הקבצים לאותו שלב שבהם האדם בתמונה עם מסכה. **_no_mask** – רשימת שמות הקבצים לאותו שלב שבהם האדם בתמונה בלי מסכה. **_half_mask** – רשימת שמות הקבצים לאותו שלב שבהם האדם בתמונה עם מסכה על הפה. **_bottom_mask** – רשימת שמות הקבצים לאותו שלב שבהם האדם בתמונה עם מסכה על הסנטר.	מטרתה של הפונקציה היא לקחת כמות ערכים לפי האחוזים שניתנו מכל אחת מן הרשימות. הפעולה שאחראית על כך היא הפעולה **take_from_list__** לאחר שנלקחו כמות הקבצים הנדרשת לפי האחוזים מכל אחד מן הרשימות אנחנו מעתיקים את הקבצים מתוך התיקייה ResizedImages אל תוך התיקייה שקיבלנו בstep_name. ההעתקה נעשת באמצעות הפעולה **copy_files_to_dst.**
take_from_list__	**:Function arguments** **li** – רשימה (רשימות ה mask, unmasked, half_mask, bottom_mask). **precent** – כמות האחוזים לקחת.	הפעולה אחראית על לקחת אחוזים מהרשימה. הפעולה הזו אמורה להיקרא שלוש פעמים בעבור כל אחד מה steps – train, test, validation.

	:Function variables **taken_list** – רשימת הקבצים שנקלחה. **amount_of_elements_to_take** – מכיל את כמות הקבצים שאמור להיות בtaken_list לפי האחוזים שקיבלנו ואורך הרשימה.	בכניסה לכל פעולה נחשב לפי האחוזים וגודל הרשימה כמה אלמנטים יש לקחת. בכדי שלא יצא מצב שבשלב מאוחר ילקחו תמונות שכבר נלקחו בשלב מוקדם יותר אנחנו נשנה את הflag לTrue בעבור כל תמונה שנלקחה. ככה שבאיטרציה נדלג על כל תמונה שכבר נקלחה ונמשיך לאסוף תמונות עד שנגיע לכמות שחישבנו בתחילת הפונקציה.
copy_files_to_dst	**li –** רשימת שמות קבצים. **dst –** התיקייה אליה אנחנו מעתיקים את הקבצים.	הפעולה מקבלת שמות קבצים ללא נתיב מלא ומניחה שהם נלקחו מתוך התיקייה ResizedImages (כי משם נקרא כל הרשימות). הפעולה מעתיקה את הקבצים בהוספת הנתיב של ResizedImages אל הנתיב dst שקיבלנו בפעולה.

image_processor.py

הקובץ אחראי על תהליכי עיבוד תמונה.

שם הפונקציה	משתנים	מטרת הפונקציה
resize_image	**filename** – שם הקובץ לו אנחנו רוצים לשנות את הגודל. **newfilename** – שם הקובץ החדש שינתן לקובץ שאנחנו משנים לו את הגודל.	הפונקציה מקבלת שם של קובץ אותו רוצים לשנות לו את הגודל. היא משנה לו את הגודל x150150 ורושמת את הקובץ החדש לשם הקובץ החדש שקיבלנו.

model.py

הקובץ אחראי על אימון ובחינת המודל.

שם הפונקציה	משתנים	מטרת הפעולה
train_model	**:Function arguments** **model –** המודל שאנחנו יוצרים. **train –** מדובר על tuple שמכיל במקום הראשון generator של train ובמקום השני את כמות האיטרציות. **validation–** מדובר על tuple שמכיל במקום הראשון generator של	הפעולה אחראית על בניית המודל, אימון המודל והחזרה של המודל. הפונקציה מקבלת את הgenerator של train ושל הvalidation + כמות האיטרציות מהפעולה **receive_for_level** מתוך המודול **data_reciever.py** באמצעות קריאה מן התיקיות train, validation. הפונקציה אחראית על לשמור את ההיסטוריה של המודל במשתנה הגלובלי history.

	validationה ובמקום השני את כמות האיטרציות. **history** – שומר את היסטוריית המודל כדי שנוכל לסרטט גרפים מאוחר יותר.	
evaluate_model	Function arguments: model – מודל עליו אנחנו רוצים לעשות test.	לעשות test על המודל שקיבלנו בפעולה. הפעולה מקבלת את הgenerator + כמות האיטרציות באמצעות הפעולה **receive_for_level** במודול **data_receiver.py** באמצעות קריאה מן התיקיות train, validation.
draw_model_graph	Function variables: **history** – היסטוריית המודל שמשמש אותנו בכדי לבצע את הגרפים.	הפונקציה משתמשת במשתנה גלובלי history בכדי לשרטט שני גרפים ולשמור אותם. גרף שמראה את השינוי בmodel_loss בtrain ובvalidation. גרף שמראה את השינוי בaccuracy בtrain ובvalidation.

tkapp.py

MainMenuForm

הscript אחראי על כל מה שקשור לתקשורת עם המשתמש.

הסקריפט מחולק למחלקות כאשר כל מסך באפליקציה מיוצג באמצעות מחלקה בקוד.

לכן אנחנו נבצע חלוקה לפי מחלקות בהסבר.

רוב המשתנים שייצויינו כאן מאותחלים כבר ב__init__ לNone אך הם מקבלים את ערכם ואת משמעותם בפונקציות המחלקה.

מדובר בחלון הראשי של האפליקציה ומייצג מעין נקודת ניווט.

רק דרכו אפשר להגיע לחלון של בחינת המודל ורק דרכו אפשר להגיע לחלון של בניית המודל.

משתני המחלקה:

– model_op

משתנה סטטי שבמידה והמשתמש יבחר במודל ברירת המחדל יטען אליו המודל ברירת המחדל.

אחרת ערכו יהיה None.

start_main_menu

בפונקציה זו אנחנו מתחילים את האפליקציה ומאתחלים את המשתנים שלה.

- main_app

המשתנה הזה מהווה את הroot של האפליקציה.

– select_model_btn

הכפתור שאחראי לתת למשתמש את האופציה לבחור מודל (במידה וקיים מודל אחרת יוביל ישר ליצירה) או ליצור מודל.

בעת לחיצה על הכפתור תיקרא הפעולה ask_user_to_select_model_.

– test_chosen_model_btn

הכפתור שאחראי להתחיל את המסך של חלק הtest לאחר שהמודל נבחר.

הכפתור לא פעיל עד שלא נבחר מודל.

בעת לחיצה על הכפתור יופעל הפעולה start_testing_model_.

ask_user_to_select_model_

הפונקציה אינה מקבלת פרמטרים.

בתחילה הפונקציה תבדוק אם קיים תיקייה model_data (בתיקייה זו אמור להיות המודל ברירת מחדל).

אם קיימת תיקייה כזו המודל ישאל את המשתמש אם הוא מעדיף להשתמש במודל הקיים או ליצור מודל חדש.

אם המשתמש יאמר שהוא מעדיף את המודל ברירת מחדל, אזי המודל ברירת מחדל הוא זה שיטען לאחר מכן הכפתור select model כבר לא יהיה פעיל ובמקומו יהיה פעיל הכפתור האחראי על בחינת המודל.

אם אינה קיימת תיקייה כזו / המשתמש העדיף ליצור מודל אז הפעולה תפתח חלון חדש דרך המחלקה BuildModelForm שאחראי על לבנות את המודל.

start_testing_model_

הפונקציה אינה מקבלת פרמטרים.

הפונקציה פותחת את החלון של TestModelForm שאחראי על בדיקת המודל.

BuildModelForm

רוב המשתנים שייצויינו כאן מאותחלים כבר ב__init__ לNone אך הם מקבלים את ערכם ואת משמעותם בפונקציות המחלקה.

החלון הזה מיועד בכדי לבנות את המודל.

משתני המחלקה:

- tempModel

מדובר על משתנה סטטי במחלקה (אינו תלוי בinstance מסוים) שאמור להכיל את המודל שאנחנו יוצרים בחלון.

– m_Menu

משתנה שאמור להחזיק instance של הform הראשי.

המשתנה מאותחל ב __init__.

– is_form_closed

המשתנה מציין אם החלון של הinstance של המחלקה הנוכחי נסגר או לא.

start_build_model_menu

הפונקציה אחראית על להתחיל את החלון בניית המודל באפליקציה.

– build_model_app_

root של חלון בניית המודל.

– build_model_title_

מכיל את הכותרת שמופיעה בחלון "Build Your Model:"

– organize_data_set_button_

הכפתור האחראי על תחילת ארגון המידע באפליקציה.

בעת לחיצה על הכפתור תתבצע בדיקה, אם קיימות תיקיות train, test, validation האפליקציה תשאל את המשתמש האם הוא מעדיף לסדר מחדש ולדרוס את התיקיות הקיימות.

אם המשתמש אמר שהוא מעדיף לדרוס / אין למשתמש תיקיות כאלה עדיין האפליקציה תפתח למשתמש הודעה בה היא תסביר לו על העובדה שכעת עליו לבחור את הדרך בה המידע יחולק (כמה אחוזים לכל אחד מהשלבים) ותסביר לו את הדרך בה הוא צריך לרשום את הinput (באיזה פורמט).

חשוב לציין כי הפונקציה אחראית גם על זריקת שגיאות במידה והפורמט שבו המידע הוכנס הוא אינו תקין (אם הוכנס כמות אחוזים ששונה מ100, אם יש אחוזים שליליים, אם לא הוכנס מספיק ערכים וכו').

את תהליך מיון המידע ניתן לראות במסך ההרצה של התכנית.

הפעולה של הכפתור תתבצע בפעולה **ask_user_if_split_exists_already_.**

– train_model_btn_

הכפתור האחראי על תחילת האימון של המודל.

בתחילת פתיחת החלון הכפתור יהיה לא פעיל והסיבה לכך היא שאי אפשר להתחיל לאמן את המודל לפני שהמידע מסודר.

לאחר שהמשתמש יסדר את המידע (בכך שיקח את המידע כמו שהוא / ימיין מחדש) הכפתור יעבור למצב נורמלי ויהיה לחיץ.

בעת לחיצה על הכפתור יתחיל תהליך האימון.

את תהליך אימון המידע ניתן לראות במסך ההרצה של התכנית.

בעת לחיצה על הכפתור תיקרא הפעולה **start_training_model_.**

– close_btn_

הכפתור אחראי על סגירת החלון הנוכחי.

באמצעות הפעולה onclose המפורטת בהמשך.

ask_user_if_split_exists_already_

הבדיקה הראשונה שהפונקציה מבצעת זה האם קיימות תיקיות train, test, validation.

אם התיקיות קיימות היא שואלת את המשתמש אם הוא מעוניין לדרוס אותם, אם הן לא קיימות היא לא מציגה לו את השאלה ופשוט דורסת.

אם הוא בחר שלהישאר עם הנתונים הקיימים היא הופכת את הכפתור של ארגון המידע ללא פעיל ואת הכפתור של תחילת האימון לפעיל.

אם לא היו קיימות לו תיקיות / הוא בחר לדרוס ולארגן מחדש יופיע בפניו הודעת הסבר.

הודעת ההסבר תסביר לו על הפורמט שבו הוא יצטרך להכניס את המידע של באיזה אחוזים הוא מעוניין לארגן.

בנוסף לכך יתווספו שני אובייקטים נוספים למסך:

split_entry_ –

האובייקט הזה הוא למעשה שדה טקסט שבו המשתמש אמור להכניס את האחוזים שלפיהם הוא רוצה לחלק את המידע.

אם הוא לא מכניס כלום אזי שהוא מעוניין לחלק את המידע בדרך ברירת המחדל שהיא 70,20,10.

send_split_entry_btn_ –

בעת לחיצה על הכפתור תתבצע ולידציה על הנתונים שהוכנסו בשדה הטקסט של split_entry_ באמצעות הפונקציה **validate_input_for_steps_**.

validate_input_for_steps_

הפונקציה אחראית לבצע בדיקות על הקלט ובמידה והכל תקין לקרוא ל**th_start_organizing_the_data_** שאחראית על ארגון תהליך מיון המידע.

הבדיקה הראשונה שהיא מבצעת היא האם מדובר בשדה ריק. אם מדובר בשדה ריק אזי המשתמש מעוניין לחלק בדרך ברירת המחדל שהיא 70,20,10.

הפונקציה מתחילה thread לפונקציה בשם **th_start_organizing_the_data_**.

שמטרתו להתחיל למיין את המידע.

במידה ולא מדובר בשדה ריק הפונקציה תבצע כמה בדיקות נוספות.

במהלך הפעולה נשתמש במשתנה בשם temp_split_format מדובר על משתנה זמני כמי ששמו מרמז שמשמש אותנו לצורך הבדיקות במידה ויעבור את כל הבדיקות הערכים בו יהיו ואלידים ואכן יהיה בו רשימה של ערכי החלוקה לפי אחוזים ולכן נשים את ערכיו בערכים של משתנה המחלקה steps_split_.

הבדיקה הראשונה שהיא תבצע היא האם הוכנסו שלושה ערכים. הדרך שבה התבקשו להיכנס הערכים על פי הודעת ההסבר היא x,y,z כלומר 70,20,10 או 60,30,10 כאשר בדוגמה א 70 אחוז הולך לחלק ה train 20 אחוז הולך לחלק הtest והה10 אחוז האחרונים הולכים לvalidation.

אם הוכנסו שלושה ערכים ממשיכים לבדיקות נוספות אם לא יוצאים מהפונקציה ומחזירים הודעת שגיאה שלא הוכנסו כמות הערכים כנדרש.

הבדיקה הבאה שאנחנו מבצעים זה האם כל הערכים שהוכנסו הם מספרים שלמים.

כלומר בעבור כל ערך שהוא לא מספר שלם (מספר ממשי, מילה) ייזרק הודעת שגיאה ונצא מהפונקציה.

אם הבדיקה הזאת עברה נבצע בדיקה נוספת שתבדוק האם כל המספרים הם גדולים מאפס. שכן אי אפשר לבצע לפי אחוזים שליליים.

אם הבדיקה עוברת נמשיך לבדיקה הבאה אם לא נזרוק הודעת שגיאה שנמצאו ערכים שליליים.

הבדיקה האחרונה שנבצע היא שכמות האחוזים מסתכמת ל100 אחוז בדיוק. אם מסתכם נתחיל thread עם הפונקציה **th_start_organizing_the_data_** שיתחיל למיין את המידע לפי הנתונים שהוכנסו, אם לא נצא מהפונקציה.

למה בכלל להתחיל thread?

הסיבה ללמה להתחיל thread חקוקה בעובדה שאם לא נתחיל thread ונריץ את הפונקציות כמו שהן האפליקציה תקרוס.

אנחנו נקבל הודעה שלאפליקציה לוקח יותר מדי זמן להגיב וזה קורה מכיוון שאנחנו קוראים לפונקציות שלוקח להם הרבה זמן להסתיים בעת לחיצה על הכפתור ועד שהפונקציה מסתיימת האפליקציה קורסת כבר.

בכדי להתמודד עם הבעיה שהכפתור לחוץ יותר מדי זמן אפשר בקלות רק להתחיל את הthread בפונקציה שהcommand מריץ ישירות.

מכיוון שכל שאר האופציות חסומות אין למשתמש דרך למשל להתחיל את תהליך האימון כי הכפתור שלו הוא לא פעיל.

בשורה האחרונה של הפונקציה שאותה אנחנו מריצים עם הthread אנחנו הופכים את הכפתור לפעיל וכך בעצם חיכינו לthread שיסתיים מבלי באמת לחכות לו.

th_start_organizing_the_data_

הפונקציה אחראית על ארגון תהליך מיון המידע.

הדבר הראשון שהפונקציה עושה זה לפתוח הודעת הסבר למשתמש כי הוא יכול לראות את הלוגים במסך ההרצה ושבסיום יופיע הודעת Success.

לאחר מכן הפונקציה הופכת את הכפתור של send_split_entry_btn_ ללא פעיל.

לאחר מכן מתחילה את תהליך ארגון המידע באמצעות הנתונים של החלוקה שהמשתמש בחר.

ברגע שחלק המיון נגמר היא הופכת את הכפתור של train לפעיל ואת הכפתור של organize_data_set_btn_ ללא פעיל.

ומראה למשתמש הודעת Success.

start_training_model_

הכפתור אחראי על תחילת תהליך האימון.

בעת לחיצה על הכפתור תופיע למשתמש הודעת הסבר שמסבירה כי הוא יכול לראות את תהליך האימון במסך ההרצה ובעת סיום התהליך תופיע בנוסף גם הודעת Success.

לאחר שנסגרת ההודעה יתחיל thread שיהיה אחראי להתחיל את תהליך האימון.

th_train_model_

בעת תחילת הפונקציה היא מתחילה את תהליך האימון ומיד לאחריו מציגה את הגרפים.

(גרף של loss לפי epoch בtrain + validation וגרף של accuracy לפי epoch בtrain + validation).

בתחילת ההרצה היא הופכת את הכפתור של תחילת האימון ללא פעיל, כמו כן הכפתור של בחירת המודל במסך הראשי נשאר לא פעיל בסוף ההרצה (כי נבנה כבר מודל).

והיא הופכת את הכפתור של הtest model במסך הראשי לפעיל (בסוף ההרצה).

חשוב לציין כי המודל יבדק רק אם החלון עוד לא נסגר! כלומר אם החלון נסגר לפני שהמודל סיים את ההרצה שלו המודל שיווצר בסוף לא יהיה זה שיבדק בתוכנה. את חלק זה אנחנו בודקים בעזרת הFlag של is_form_closed.

– onclose

הפונקציה נקראת מתי שהחלון נסגר.

הפונקציה שמה flag של **is_form_closed** כTrue.

הפונקציה גם שמה את הכפתור של select model כEnabled במידה ואין מודל קיים עדיין.

הפונקציה גם סוגרת את החלון.

TestModelForm

החלון האחראי על חלק הtest.

רוב המשתנים שייצויינו כאן מאותחלים כבר ב__init__ לNone אך הם מקבלים את ערכם ואת משמעותם בפונקציות המחלקה.

משתני מחלקה:

– model

משתנה מחלקה שמאותחל ב__init__.

בכדי להגיע לחלון בדיקת המודל חייב שאו שיהיה מודל ברירת מחדל טעון או שהמשתמש יצור מודל משלו. (חייב שיהיה מודל אחרת הכפתור לפתיחת החלון היה לא לחיץ).

אנחנו בודקים אם המודל בחלון של יצירת המודל הוא None אם כן כלומר לא יצרנו שם מודל ולכן המודל שנטען הוא מודל ברירת המחדל ולכן ניקח אותו מן המחלקה של המסך הראשי (גם הוא משתנה סטטי במחלקה).

– start_main_test_menu

הפעולה אחראית על לפתוח את החלון הראשי לבחינת המודל.

– test_menu

הroot של החלון הראשי לtest.

– test_title

מכיל את הכותרת בחלון של הtest "Test Your Model: ".

– evaluate_by_test_folder_btn

בעת לחיצה על הכפתור תיקרא הפעולה **show_model_evaluation** שאמורה להציג נתונים יבשים על המודל לפי תיקיית הטסט. כלומר כמה loss וכמה accuracy היה לו בבדיקה על תיקיית הטסט.

– test_with_new_pictures_btn

הכפתור אחראי על לפתוח את הפעולה של **open_images_for_test** תחת המחלקה **TestWithPicturesForm** שאמורה לתת לנו את האופציה לבחור קבצים משלנו לעשות עליהם טסט.

– show_model_evaluation

הפונקציה אחראית על לבדוק את המודל שנוצר / שנטען כברירת מחדל על חלק הטסט בתיקיות בפרויקט.

בהתחלה הפעולה תציג הודעה למשתמש כי התחיל התהליך של לבדוק את המודל ותאמר לו שיופיע הודעת success ברגע שיהיה תוצאות כמו כן תאמר לו שהוא יכול לצפות בתהליך קורא בלייב במסך ההרצה בתכנית.

מכיוון שתהליך זה לוקח זמן הפעולה תתחיל thread על הפעולה **_th_evaluate_model.**

– th_evaluate_model_

הפונקציה אחראית על ארגון חלק הtest לפי תיקיית הtest באפליקציה.

הפעולה מחשבת את גודל השגיאה של המודל ואת גודל הדיוק שלו ולאחר שהיא מקבלת את הגודל הסופי היא מציגה אותו למשתמש

TestWithPicturesForm

מחלקה זו אחראית על לבדוק את המודל על תמונות שהמשתמש בוחר.

– open_images_for_test

בתחילת הפונקציה היא פותחת חלון בחירת קבצים ואנחנו נותנים למשתמש לבחור כמה קבצים שהמשתמש רוצה.

לאחר שהוא בחר את הקבצים אנחנו מבצעים עליהם ולידציה ומכריחים שהסיומת של הקובץ שנבחר תהיה או jpg או png או jpeg אחרת אנחנו זורקים שגיאה ויוצאים מהפונקציה.

אם שמות הקבצים עברו את הבדיקה אנחנו עוברים קובץ קובץ משנים את הגודל שלו לגודל של x150150 מנרמלים אותו ואז עושים predict מכיוון שאנחנו רוצים לקבל label ולא מספר אנחנו קוראים לפונקציה predict_label.

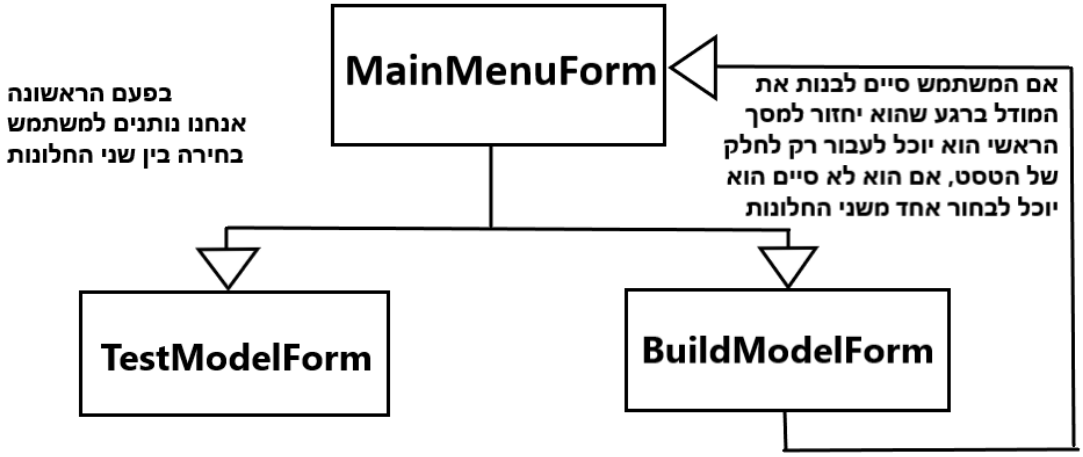
את התוצאה שקיבלנו מהpredict אנחנו רושמים בצבע ירוק על התמונה עצמה ומציגים אותה למשתמש בעת כל סגירה של תמונה יוצג תמונה חדשה מאלה שהמשתמש בחר.

predict_label

הפונקציה מוצאת איזה מספר תגית חושב שהתמונה שניתנה וממיר את המספר לlabel שם ומחזיר את הlabel שם.

מדריך למשתמש

Screen Flow Diagram



הסבר על דיאגרמת המסכים:

במסך הראשון יש למשתמש בחירה בין לטעון מודל ברירת-מחדל לבין לבנות מודל חדש.

1. במידה ובחר לטעון את המודל הקיים האופציה היחידה הקיימת מבחינתו היא בדיקת המודל (חלק הtest).
2. במידה והוא בחר לבנות מודל חדש הוא יועבר אל חלון בניית המודל.
    1. אם הוא סיים לבנות את המודל הוא יוכל לחזור למסך הראשי ותיפתח בפניו רק האופציה לבדוק את המודל.
    2. אם הוא יצא מן המסך לפני שסיים לבנות את המודל יהיה פתוחות בפניו כל האופציות האפשריות.

קבצים ותיקיות שנצטרך בכדי להריץ את האפליקציה

נפתח תיקייה בשם PythonProject (כל שם שנבחר תקין לצורך הדוגמה בחרתי את השם הזה).

בתיקייה הזו יש לשים את כל הסקריפטים של התכנית:

data_receiver.py

data_organizer.py

model.py

image_processor.py

tkapp.py

את הסקריפטים יש להוריד מתוך הGitHub לתיקייה **ולשים אותם ישירות תחת התיקייה PythonProject ולא תחת תיקייה פנימית!**

את התיקיות שמכילות מידע הכנסנו לקבצי zip והעלנו לדרייב כלומר בכדי להוריד את תיקיות ה train, test, validation, ResizedImages, Images, model_data יש להיכנס ולהוריד מהדרייב בקישור

https://drive.google.com/drive/folders/1iQQRS1M6UXxEWdrIdlrwEivEIzvYcz9g.

הסבר קצר על מטרתן של התיקיות:

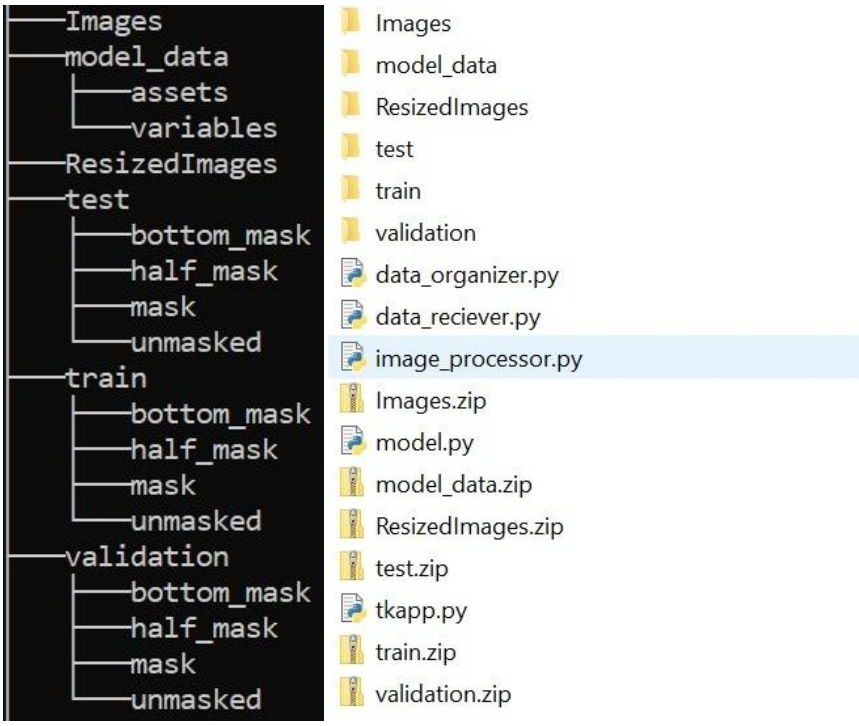
model_data.zip – קובץ הzip מכיל תיקייה המכילה את המודל ברירת מחדל.

Images.zip – קובץ zip זה מכיל תיקייה המכילה כמה תמונות להוכחת ההיתכנות של שינוי גודל התמונות לגודל של 150x150.

ResizedImages.zip – מכיל תיקייה אשר מכילה את מאגר כל התמונות בהם אנחנו משתמשים לאחר ההקטנה.

train.zip, test.zip, validation.zip – כל אחד מן הקבצים מכיל את התיקייה השייכת לשלב אותו הוא מייצג (train, test, validation). התיקיות מכילות חלוקת ברירת מחדל של 70,20,10 שעליו המודל ברירת מחדל אומן.

את קבצי הzip יש לפתוח בPythonProject ישירות כך שיראו כמו בתמונות בעמוד הבא.

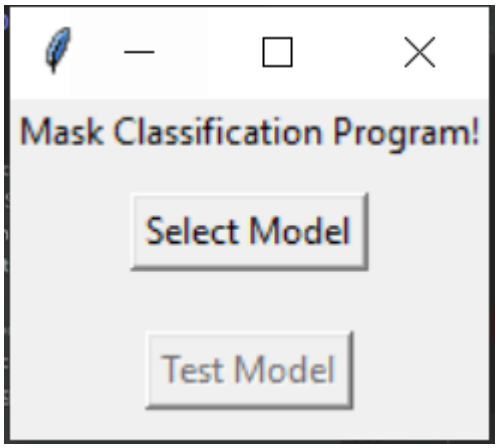


היררכיית התיקיות המצופה בתוך Python Project.

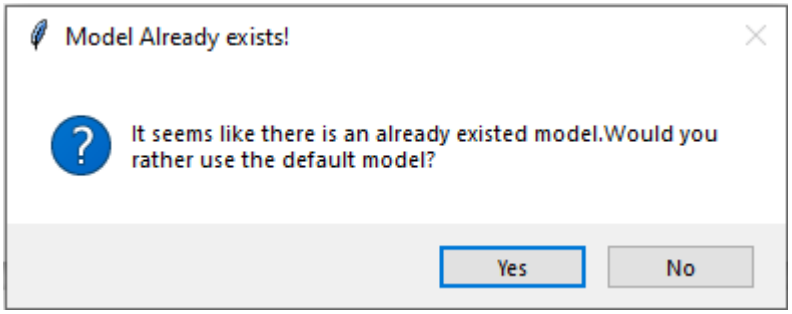
איך להוריד את הספרייה	שם הספרייה
pip install tensorflow	tensorflow(2.9.0)
pip install numpy	numpy(1.22.4)
pip install tk	tkinter
pip install opencv-python	cv2
pip install matplotlib	matplotlib
pip install keras	keras(2.9.0)

הסברים על החלונות

Main Menu Form



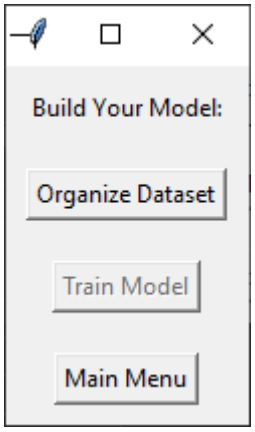
משמש מסך ניווט, בתחילת האפליקציה אנחנו נותנים למשתמש לבחור את המודל. בלחיצה על הכפתור Select Model תופיע ההודעה הבאה על המסך:



1. אופציה א היא להשתמש במודל ברירת מחדל שכבר אומן על המידע.
   1. אם הוא בחר להשתמש במודל הקיים, הוא יוכל ישר להתחיל לבחון את המודל באמצעות החלון של test model.
2. אופציה ב היא לבנות את המודל בחלון חדש ולראות את ההרצות בזמן אמת.

חשוב לציין כי בעת לחיצה על הכפתור לבניית המודל הכפתור Select Model יהיה לא פעיל. הכפתור יחזור להיות פעיל אך ורק אם יסגר החלון שנפתח לפני שנגמר תהליך סיום המודל. כלומר אם כבר נוצר המודל בתהליך הכפתור ימשיך להיות לא פעיל.

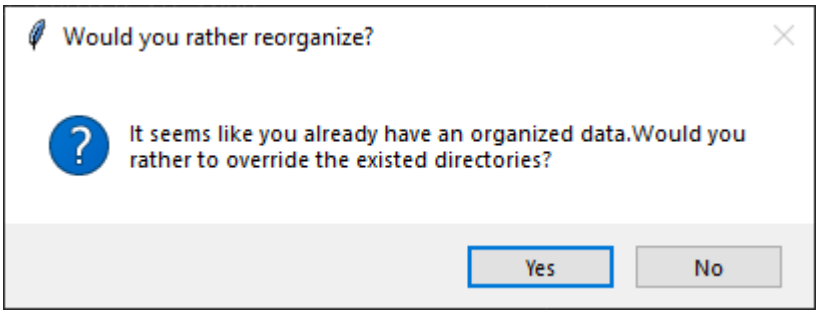
Build Model Form



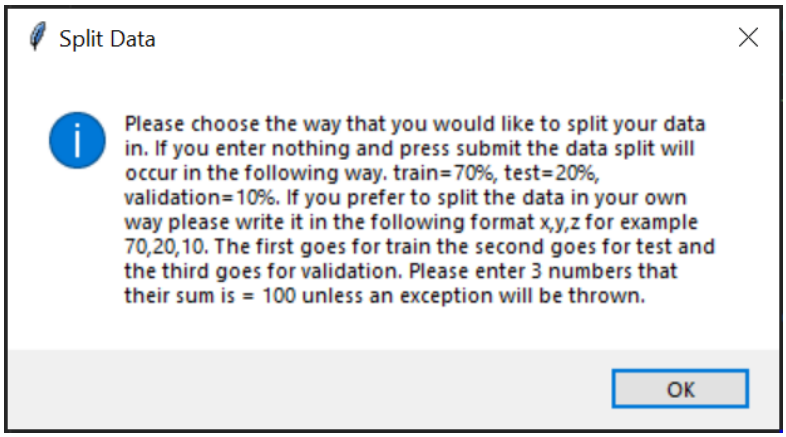
חלון זה משמש את המשתמש בכדי לבנות את המודל בשלבים. החלק הראשון בבנייה של מודל היא ארגון המידע, הכפתור Organize Dataset אחראי לחלוקת המידע לtrain, test, validation מתוך תיקיית המידע בשיטות שהוצגו בחלק של הארכיטקטורה.

בעת לחיצה על הכפתור למשתמש יהיו שני אופציות לבחור:

1. במידה וקיימות כבר תיקיות train, test, validation בדיסק תוצג שאלה למשתמש:



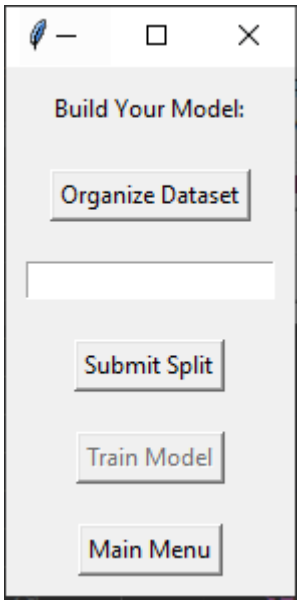
1. אם המשתמש יהיה מעוניין לעשות את כל ארגון המידע מחדש יפתחו בפניו כמה דברים חדשים.
   1. ראשית כל תיפתח בפניו הודעת הסבר:



הודעה זו מסבירה כמה דברים בנוגע לשלבים הבאים וחשוב לקרוא אותה לפני שממשיכים.

1. דבר ראשון, ההודעה מסבירה כי אם יוכנס בתיבת הטקסט שתיפתח טקסט ריק (כלומר כלום לא ירשם בפנים) החלוקה לtrain, test, validation תתבצע באופן ברירת-מחדל כלומר ביחס של 70% לחלק הtrain, 20% לחלק הtest, ו10% לחלק הvalidation.
2. הדבר השני שהיא מציגה היא שאם ויש רצון להכניס חלוקה שונה מהחלוקה הרגילה יש להכניס את זה בפורמט של x,y,z כלומר מספר פסיק מספר פסיק מספר ולהשתמש במספרים שלמים.
3. מותר לסכום להיות קטן מ100 כלומר, אין מניעה מלא להשתמש בכמות מידע מלאה אך יש מניעה מלשים אחוזים שיוצרים יותר מ100 אחוז.

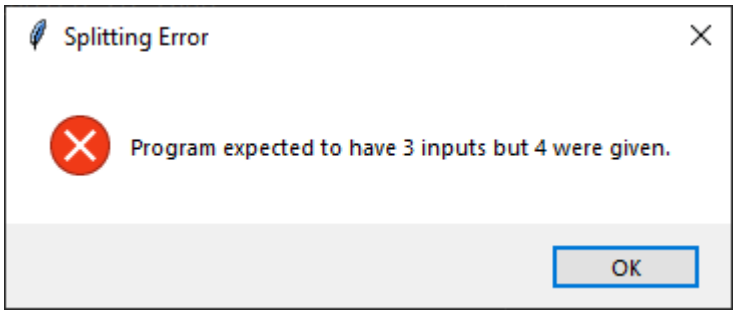
.2



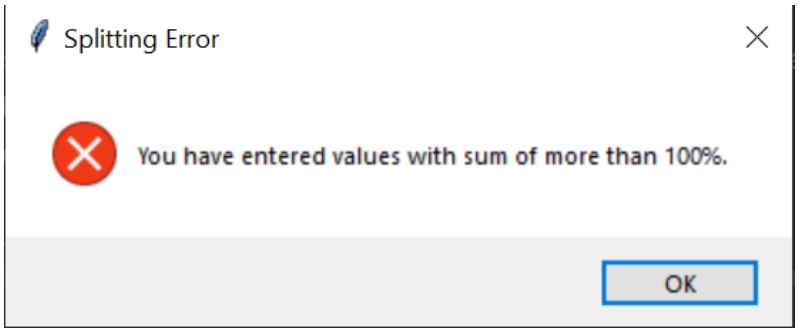
לאחר ההסכמה להודעת ההסבר נראותו של החלון תשתנה מאט ויתווספו שני אובייקטים נוספים לחלון.

האובייקט הראשון הוא הטקסט שבו אנו רושמים את הדרך בה אנחנו רוצים לחלק את ה train, test, validation. כפי שהוסבר קודם לכן בהודעת ההסבר יש לעקוב אחר הכללים ובמידה ואחד מן הכללים יופר תישלח הודעת שגיאה למשתמש.

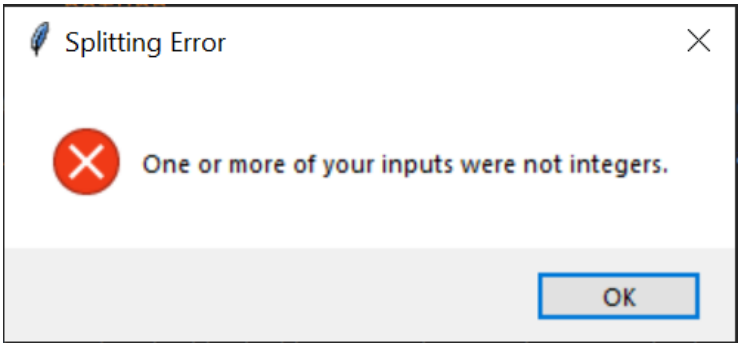
חשוב לציין כי בהרצות שונות של Organize Dataset יתקבלו תוצאות ארגון שונות מכיוון שהחלטנו בארגון שלא להשתמש בseed. הסיבה לכך היא שאנחנו רוצים לתת למשתמש את היכולת לשנות את הסדר, במידה והוא רוצה לשמור על הסדר כמו שהוא באותו הרגע הוא יכול פשוט לדלג על חלק מיון המידע מחדש ולסמן שהוא מעדיף את הסדר הנוכחי. ככה אנחנו מעניקים למשתמש את שני האופציות.



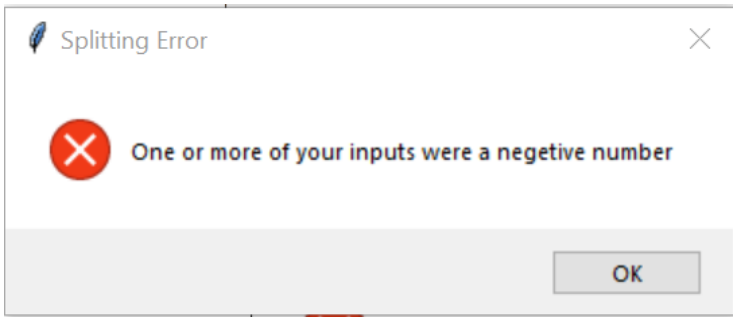
מדובר בהודעת דוגמא שבה הכנסנו 1,1,1,1 בEntry



הודעת שגיאה בעבור הכנסת ערך של 200,10,10



הודעת שגיאה בעבור הכנסת ערך של aa,aa,aa.



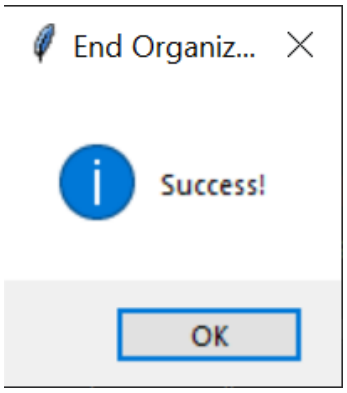
הודעת שגיאה בעבור הכנסה של 1-,1-,1-.

אם המשתמש הכניס את החלוקה בפורמט הנכון יופיע במסך ההרצה ממנו התחיל להריץ את האפליקציה לוגים על סטטוס מיון המידע והכפתור של ארגון המידע יהפוך ללא פעיל.

```
Grouping into subjects...
Start organizing step: train
Start mask in train
Start bottom_mask in train
Start half_mask in train
Start unmasked in train
Finished organizing step: train
Start organizing step: test
Start mask in test
Start bottom_mask in test
Start half_mask in test
Start unmasked in test
Finished organizing step: test
Start organizing step: validation
Start mask in validation
Start bottom_mask in validation
Start half_mask in validation
Start unmasked in validation
Finished organizing step: validation
```

דוגמה ללוגים

ברגע שיסתיים חלק הארגון מידע המשתמש יקבל הודעת Success ויפתח בפניו הכפתור הבא Train Model.



דוגמה להודעת הSuccess.

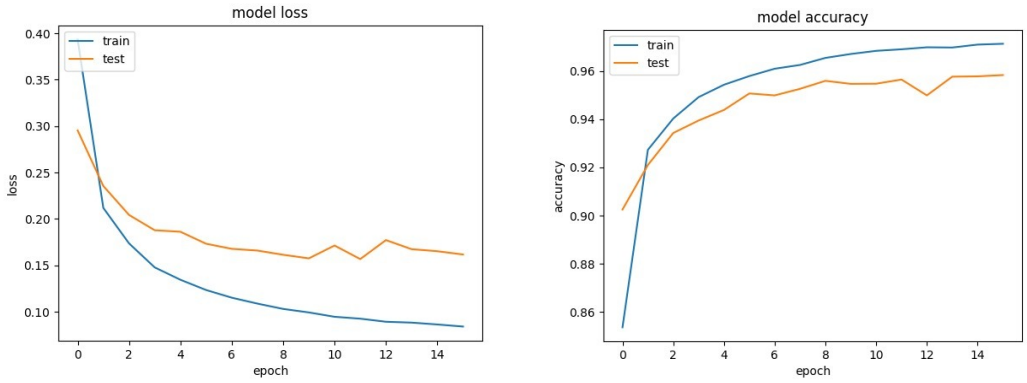
2. בעת לחיצה על הכפתור Train Model יחל אימון המודל. בעת לחיצה על הכפתור יופיע במסך ההרצה ממנו התחיל להריץ את האפליקציה לוגים על סטטוס אימון המודל.

```
Epoch 1/16
  42/3300 [..............................] - ETA: 20:30 - loss: 1.3399 - accuracy: 0.3408
```

דוגמה ללוגים

ברגע שיסתיים אימון המודל תופיע הודעת Success. חשוב לציין כי בעת סיום אימון המודל כפתור הTest Model בMain Form יהפוך מDisabled לEnabled.

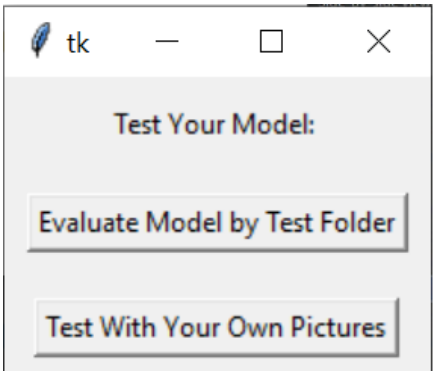
בסיום ההרצה יופיעו שני גרפים:



דוגמה לגרפים שיוצגו

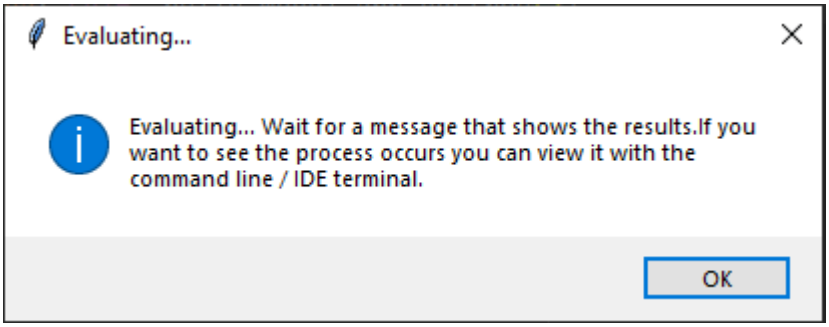
3. אופציית החזרה למסך הראשי – בעת לחיצה על הכפתור המשתמש יוכל לחזור למסך הראשי של האפליקציה.

Test Model Form

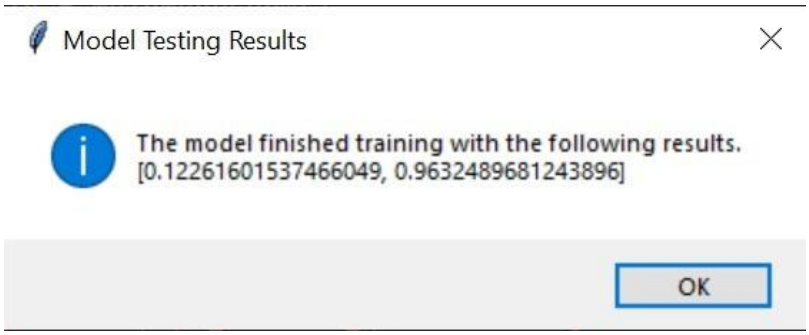


בחלון בדיקת המודל ישנם שני אפשרויות:

1. **Evaluate Model By Test Folder** – בעת לחיצה על הכפתור תופיע הודעת הסבר למשתמש:



הודעה זו מסבירה למשתמש כי ברגע שיסתיים התהליך תופיע הודעה ובה יוצגו תוצאותיו הסופיות של evaluation.



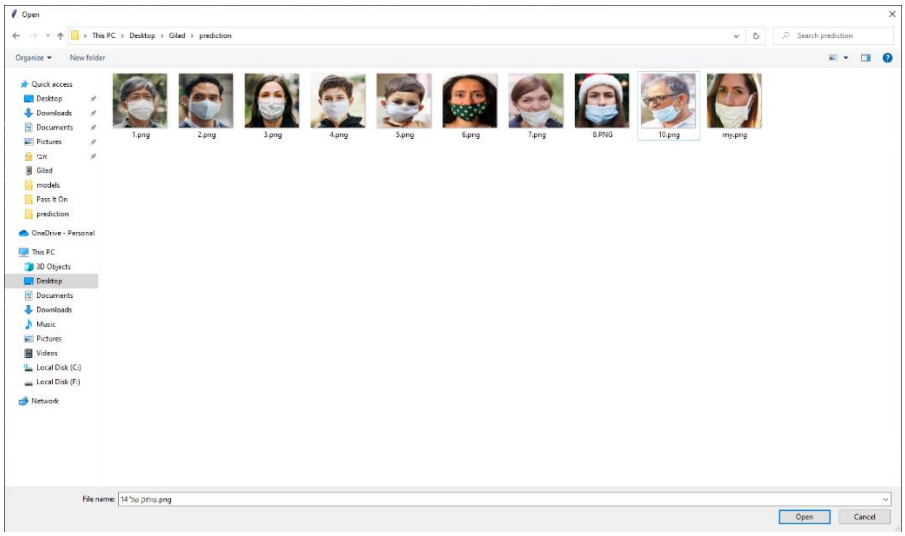
דוגמה להודעת evaluation.

ובמידה והוא רוצה לצפות בתהליך קורא בזמן אמת הוא יכול לראות את הלוגים דרך מסך ההרצה ממנו הריץ את האפליקציה.

```
30176
 99/943 [==>.............................] - ETA: 1:29 - loss: 0.1185 - accuracy: 0.9605
```

דוגמה ללוגים.

2. **Test model with pictures –** בעת לחיצה על הכפתור יפתח חלון בחירת קבצים שעליהם נבצע את המודל.



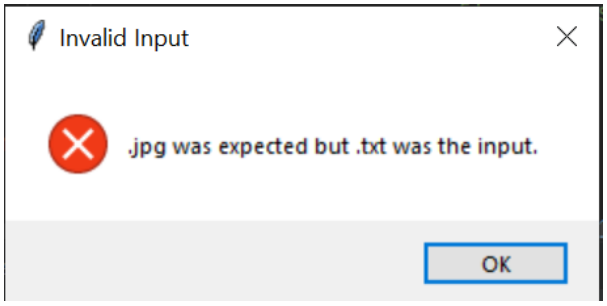
דוגמה לחלון שנפתח

המשתמש יכול לבחור כמה תמונות שירצה והתמונות יוצגו בדרך הבאה:

- על התמונה יהיה רשום את שם הזיהוי בכתב ירוק (Full, Half, No, Bottom) .
- כל תמונה שתיסגר תפתח את התמונה הבאה שהוא בחר. כלומר, ניתן לראות רק תמונה אחת כל פעם.

דוגמה לתמונה שנבחרה.

אם הקובץ שנבחר אינו .jpeg. png. jpg תיזרק הודעת שגיאה ולא ייבחנו גם שאר התמונות!



דוגמה להודעת שגיאה בעבור קובץ טקסט

רפלקציה

העבודה על הפרויקט הייתה מהנה מאוד, במהלך העבודה רכשתי כלים רבים להמשך גם טכניים וגם כלי התמודדות. בחלקים הטכניים רכשתי ידע בנושא של דיפ לרנינג ורכשתי ידע וניסיון בטכנולוגיות רבות כמו keras, tensorflow, opencv, tkinter וכו'. כמו כן רכשתי גם ידע בשפה נוספת, במהלך הפרויקט למדתי והתנסתי בשפת התכנות פייתון ולמדתי את מרכיביה השונים. הבנתי איך הדברים בפייתון עובדים מאחורי הקלעים כמו למשל איך תכנית פייתון רצה, שימוש בספריות שונות בפייתון, יצירת מודולים ועוד.

יצא לי כמו כן לרכוש כישורי התמודדות עם פרויקטים ודברים שאני לא מכיר כתוצאה מעבודה על הפרויקט. הקשיים שהיו בהתחלת העבודה על הפרויקט הפכו לכישורים שהשגתי בסופו כמו איך לחקור? איך לחפש דברים באינטרנט בצורה טובה? איך להתמודד עם מאמרים בשפה זרה?.

המסקנות שלי מן הפרויקט הן שהתרגול הוא מה שעושה את ההבדל ככל שעבדתי יותר על הפרויקט יותר הבנתי את הדברים שקורים מאחורי הקלעים וקצת ההתקדמות שלי הלך וגדל. למדתי שעם כל הקשיים שציינתי למעלה אפשר להתמודד אם מסיקים מסקנות מטעויות ומתנסים כמה פעמים שרק אפשר.

אם הייתי מתחיל לעבוד על הפרויקט היום הייתי עושה עיצוב נכון יותר לקוד מלכתחילה. בעיה עיקרית שהייתה לי בפרויקט הייתה בחלק של האפליקציה, העיצוב שם היה לא טוב וברגע שהתקדמתי עם הקוד או שחזרתי אליו אחרי הרבה זמן התקשתי להבין אותו. הקוד לא היה עם הסברים ולא היה מחולק למחלקות לאחר מכן הייתי צריך לשכתב את כל האפליקציה. אני שמח שהדבר הזה קרה כי זה גרם לי להבין עד כמה חשוב העיצוב של הקוד. עד כמה חשוב שהוא יהיה קריא ומובן ואני שמח שגיליתי את זה ולמדתי מהטעות שלי ויכלתי לעצב אותו מחדש בצורה שאני אהיה מרוצה ממנה ואבין אותה ברגע.

אני חושב שאם היה לי מחשב יותר חזק העבודה הייתה יותר יעילה עבורי. מכיוון שלמחשב שלי לוקח זמן לעבד יצא לי לחכות זמן רב להרצות עד שהן יסתיימו. במחשב טוב יותר הייתי יכול לעבוד באופן יעיל יותר ולהגיע לתוצאות מהר יותר.

כמו שציינתי מעלה אני חושב שאם הייתי כותב עיצוב נכון לכל הקוד שלי מלכתחילה ההתקדמות שלי הייתה מהירה יותר, כאשר הקוד שלי לא היה מעוצב נכון היה לי קשה להתקדם ולפתח אותו מעבר. לאחר ששיניתי את הדרך שבה הקוד מעוצב לעיצוב נכון וקריא היה לי הרבה יותר פשוט להמשיך ולפתח אותו הלאה.

ביבליוגרפיה

D.Amos, (2022), Real Python, “Python GUI Programming With Tkinter”,

https://realpython.com/python-gui-tkinter/

J.Brownlee, (2016), Machine Learning Mastery, “Display Deep Learning Model Training History in Keras”

https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

Nilesh, (2018), Towards Data Science, “Custom Keras Generators”,

https://towardsdatascience.com/writing-custom-keras-generators-fe815d992c5a

A.Ponraj, (2021), Medium, “A Tip A Day — Python Tip #8: Why should we Normalize image pixel values or divide by 255? | Dev Skrol”,

https://medium.com/analytics-vidhya/a-tip-a-day-python-tip-8-why-should-we-normalize-image-pixel-values-or-divide-by-255-4608ac5cd26a

Sanya, (2021), GeeksForGeeks, “How to create Models in Keras?”,

https://www.geeksforgeeks.org/how-to-create-models-in-keras/