

התמחות תכנון ותכנות מערכות deep learning

תוכן עניינים

3	מבוא.....
5	ארכיטקטורת ומבנה הפרויקט.....
20	מדריך למפתח.....
42	מדריך למשתמש.....
51	רפלקציה.....
52	ביבליוגרפיה.....

מבוא

מטרת הפרויקט שלי הייתה לזהות את הסוג של שלט תנועה מתוך תמונה חתוכה של השלט.

פרויקט זה בא לתרום למאמץ כלל אנושי המתרחש כבר מספר שנים – הניסיון ליצור כלי רכב אוטונומיים (כאלו שנוהגים לבד). חלק עקרוני במאמץ זה (יש שיגידו שזהו החלק החשוב ביותר), הוא זיהוי מהיר ומדויק של שלטי תנועה. על פי כן, קהל היעד של פרויקט זה היינו אנשים \חברות המעוניינות ליצור תוכנה כלשהי המערבת זיהוי הסוג של שלט תנועה – תוכנת נהיגה אוטונומית תהווה דוגמא של תוכנה שכזו.

לשם הבנה מלאה של מטרת הפרויקט שלי יש לבצע הבחנה בין המושגים detection ו classification. שניהם נחוצים על מנת ליצור תוכנת נהיגה אוטונומית, אך הפרויקט שלי עוסק אך ורק ב classification.

detection – זיהוי הקיום של דבר מסוים, וקבלת מידע מסוים לגביו. במקרה של שלטי תנועה, detection תערב מציאת מיקומם של שלטי התנועה בתמונה גדולה יותר (ולרוב גם cropping שלהם). הפרויקט שלי לא עוסק בחלק זה של ניתוח שלטי תנועה.

classification – זיהוי הסוג של דבר מסוים. במקרה של שלטי תנועה, classification תערב זיהוי של סוג שלט תנועה מסוים, מתמונה מוקטנת\חתוכה שלו. הפרויקט שלי עוסק בחלק זה.

אופן הפעולה של הפרויקט יהיה כזה:

1. קבלת תמונה חתוכה (cropped) של שלט תנועה.
2. קבלה של שמו בחזרה כפלט.

כרגע המצב בשוק הוא כזה הכולל תוכנות רבות לנהיגה חצי אוטונומית (semi-autonomous) הפותחו על ידי חברות גדולות בעלות יכולת לאסוף כמויות עצומות של נתונים. לאומת זאת, ישנו חוסר בשוק לתוכנות קוד פתוח המסוגלות לזהות שלטי תנועה – תחילת הדרך אל תוכנת נהיגה אוטונומית פתוחה, שתאפשר לקהל רכב בהרבה להתנסות בטכנולוגיה פורצת הדרך הזו. לכן, החידוש המרכזי בפרויקט היינו העובדה שהוא מופץ כקוד פתוח ב플טפורמת github, וכל אדם המעוניין בכך יכול להשתמש בו כרצונו.

בפרויקט זה השתמשתי במספר טכנולוגיות שאינן חלק מתוכנית הלימודים:

dill – ספריית פייטון המאפשרת שמירה של אובייקטי פייטון על הכונן והחזרתם אל הזיכרון במידת הצורך. ספרייה זו מרחיבה את היכולות של הספרייה המוכרת pickle.

html, javascript, css – הטכנולוגיות המוכרות בעזרתן בונים דפי אינטרנט, זאת בשל כך שממשק המשתמש שלי בנוי כדף אינטרנט מקומי (ראה "ארכיטקטורת ומבנה הפרויקט: תיאור הטכנולוגיה על פיה מומש הממשק").

eel – ספריית פייטון המאפשרת תקשורת בין פייטון לדף האינטרנט בו מומש הממשק.

בעיה מרכזית איתה אני צופה להתמודד במהלך פיתוח הפרויקט היינה למידה עצמית של מושגים וספריות רבות הקשורות לעולם ה deep learning, בנוסף לקבלת הבנה עמוקה של התחום והמתמטיקה העומדת מאחוריו. זאת משום שאת מרבית החומר של ההתמחות בית הספר לימד אותנו השנה, ועל כן באופן טבעי נוצר מצב בו חלק גדול של החומר (בעיקר זה המערב הבנה עמוקה של התחום) נשאר לתלמידים ללמוד לבד.

בעיה נוספת איתה אני צופה להתמודד במהלך פיתוח הפרויקט היינן תקשורת בין ממשק המשתמש אל החלק הכתוב בפייטון (שכן אחד מהם בנוי בעזרת טכנולוגיות web ורץ בתוך הדפדפן, בעוד השני רץ במעבד).

ארכיטקטורת ומבנה הפרויקט

איסוף הכנה וניתוח הנתונים:

מקור הנתונים היינו: https://benchmark.ini.rub.de/gtsrb_dataset.html

מבנה נתונים זה מכיל מעלה 50,000 תמונות של מעל 40 שלטי תנועה גרמניים שונים, והוא מוכר בתור אחד ממבני הנתונים הגדולים והשימושיים ביותר בתחום הנהיגה האוטונומית שמופצים במלואם באינטרנט. מבנה נתונים זה נאסף על ידי המכון לניוראינפורמטיקה (תחום במדעי המחשב העוסק בניתוח מידע על ידי רשתות נוירונים) שבאוניברסיטת ציריך, והופץ לראשונה כחלק מתחרות בלמידת מכונה שהתקיימה כחלק מהוועידה הבין לאומית בנושא רשתות נוירונים בשנת 2011 (ICNN 2011).

מבנה הנתונים בנוי משתי תיקיות, train ו test, כאשר בתיקיית ה train ישנם תתי תיקיות המסדרות את התמונות לפי המחלקות אותן אנו מזהים.

תהליך העיבוד של כל תמונה היינו:

1. המרת התמונה לרזולוציה סטנדרטית של 30x30:

על מנת להכניס את התמונות אל תוך רשת הניורונים עלינו לוודא שכל התמונות הן באותה "צורה" (shape) – שכן אנו קובעים את מבנה רשת הניורונים מראש ועלינו לקבוע מה יהיה המבנה של שכבת הקלט, שעליו להתאים את התמונות אותן אנו מכניסים.

2. המרת התמונה למערך פיקסלים:

על מנת לנתח תמונה בעזרת רשת נוירונים, עלינו למצוא דרך להכניס את המידע המצוי בתמונה לתוך הרשת. בעוד ישנן דרכים שונות לייצג את המידע הנמצא בתמונה, דרך נפוצה, בעיקר כאשר עובדים עם רשתות נוירונים קונבולוציונליות, היא להפוך את התמונה למערך פיקסלים בעל מספר מספר מימדים, בהתאם למספר הערוצים של התמונה (לדוגמא RGB – 3 ערוצים או RGBA – 4 ערוצים). שיטה זו מאפשרת לרשת הניורונים לנתח את הפיצ'רים של התמונה ישירות (כפי שבדרך כלל נעשה ב CNNs, ראה "הסבר על שכבות הרשת") - זאת לאומת שיטות אחרות לייצג את תמונות שלא בהכרח מדגישות את הפיצ'רים הוויזואליים של התמונה.

3. נרמול ערכי הפיקסלים של המערך:

בשלב זה אנו מנרמלים את ערכי הפיקסלים של מערך התמונה לערכים בין 0 ל 1. בשל העובדה שהערך המקסימלי של פיקסל היינו 255 – תהליך הנרמול הוא חלוקה של כל ערכי הפיקסלים ב 255. מטרת שלב הנרמול היא לספק סקאלה סטנדרטית לנתונים מספריים, שכן סקאלה אבסטרקטית עלולה להאט את תהליך הלמידה ואף לגרום למודל יעיל פחות.

תהליך עיבוד מבני הנתונים (אוספים של תמונות):

1. בניית מבני הנתונים:

- בשלב זה אנו בונים את מבני הנתונים בהם אנו נשתמש ללימוד המודל:
- (1) מבנה נתונים הכולל את כל התמונות מתיקיית ה train, איתו נאמן את המודל.
 - (2) מבנה נתונים הכולל את כל התמונות מתיקיית ה test, איתו נבחן את ביצועי המודל.
 - (3) מבנה נתונים הכולל את מספרי המחלקות של התמונות במיקומים מתאימים למבנה הנתונים של ה train – נשתמש בו על מנת לספק labels לתמונות אלו.
 - (4) מבנה נתונים הכולל את מספרי המחלקות של התמונות במיקומים מתאימים למבנה הנתונים של ה test – נשתמש בו על מנת לספק labels לתמונות אלו.

2. ביצוע one-hot encoding:

בשלב זה אנו מבצעים one-hot encoding על מבני הנתונים המספקים labels לתמונות (מבני נתונים 3 ו 4). One-hot encoding היא פעולת המרה של נתונים ממערך חד מימדי של קטגוריות שונות של נתונים אל מערך דו מימדי (טבלה), כאשר לכל עמודה יש 1 בשורה בעלת הערך המתאים לה ו 0 בכל האחרות (ראה דוגמא למטה). פעמים רבות כחלק מתהליך ההמרה כלולה גם המרה של נתונים לא מספריים (לדוגמא מחרוזות טקסט) אל נתונים מספריים (לכל מחרוזת ממופה מספר מסוים). בפרויקט שלי שלב זה לא בוצע שכן הנתונים הם כבר מספריים. מטרת שלב זה היא להפוך את הנתונים לפורמט "קריא" על ידי רשת הניורונים - שכן הטבלה שתהליך זה יוצר יכולה לשמש כמערך ההסתברויות שהרשת שואפת אליו, המערך ה"נכון" שאליו אנו נשווה את תוצאות הרשת.

Color		Red	Yellow	Green
Red		1	0	0
Red		1	0	0
Yellow		0	1	0
Green		0	0	1
Yellow		0	0	1

3. פיצול הנתונים לנתוני למידה ובדיקה:

בשלב זה אנו מפרידים 20% מהנתונים המשמשים ללמידה, בהם אנו נשתמש כדי להציג את התקדמות למידת הפרויקט אחרי כל "סבב למידה" (epoch).

4. עטיפת הנתונים באובייקט אנונימי:

בשלב זה אנו עוטפים את מבני הנתונים שיצרנו בשלבים הקודמים באובייקט פייטון אנונימי, על מנת ליצור דרך קלה לספק אותם למודל בהמשך.

שלב בנייה ואימון הפרויקט:

תיאור גרפי של המודל:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 15, 15, 32)	2432
max_pooling2d (MaxPooling2D)	(None, 7, 7, 32)	0
dropout (Dropout)	(None, 7, 7, 32)	0
conv2d_1 (Conv2D)	(None, 7, 7, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 3, 3, 64)	0
dropout_1 (Dropout)	(None, 3, 3, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 1024)	590848
dense_1 (Dense)	(None, 512)	524800
dense_2 (Dense)	(None, 256)	131328
dense_3 (Dense)	(None, 43)	11051
Total params: 1,278,955		
Trainable params: 1,278,955		
Non-trainable params: 0		

הסבר על השכבות השונות ברשת:

רשת נוירונים קונבולוציונית (CNN):

רשת נוירונים קונבולוציונית היא רשת נוירונים הטובה במיוחד בניתוח תמונות ומבני נתונים הבנויים בצורת טבלה (מערכים דו ממדיים). פעמים רבות הדרך בה הרשת מזהה היא כך שבכל שכבה של הרשת היא מזהה תכונות שונות של התמונה – לדוגמא בשכבות הראשונות היא תזהה את הפינות של התמונה, ובאחרונות היא תוכל לזהות צורות שלמות. רשת נוירונים כזו מורכבת משכבות שונות שעליהן אני אפרט בהמשך.

המודל שלי בנוי מ 14 שכבות שונות כאשר ישנן:

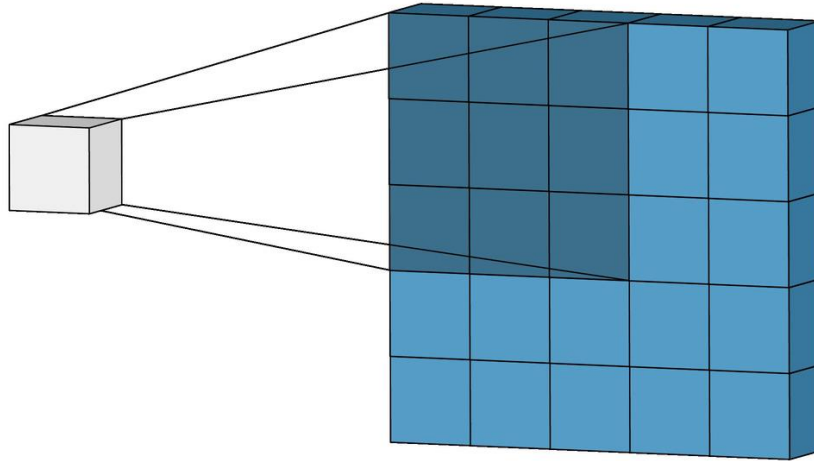
1. 2 שכבות קונבולוציה (כינוס):

שכבת הכינוס (Convolution layer) היא השכבה המרכזית והחשובה ביותר ברשת נוירונים קונבולוציונית. בלב שכבה זו נמצאת פעולת הכינוס (convolution). פעולת הכינוס היא פעולת יישום של של פילטר לקלט. תוצאת יישום הפילטר מספר פעמים היא מפה (מערך דו מימדי) של פיקסלים בעלי אקטיבציה שונה מזו המקורית. לפה זו אנו קוראים מפת תכונות (feature map). במהלך הלמידה, פילטרים אלו משתנים על מנת לקבל פילטרים שמדגישים את התכונות הרלוונטיות למה שאנו מנסים לזהות.

פילטר/קרנל (filter/kernel) – פילטר הוא טבלה דו מימדית (מערך דו מימדי) של ערכים בגודל קטן יותר מגודל הקלט של שכבת הכינוס. ערכי הפילטר מוגדרים כ"משקלים" – ולכן, בין היתר, את ערכים אלו אנו משנים במהלך הלמידה.

פעולת הכינוס (convolution operation) – במהלך פעולת הכינוס הפילטר עובר על מערך הקלט ומבצע מכפלה סקלרית בין הערכים שהפילטר עליהם באותו רגע וערכי הפילטר. לאחר מכן, תוצאת ההכפלה מוספת אל ה feature map. כך הפילטר עובר על כל מערך הקלט (אחד אחד או בקפיצות, כאשר הוא מגיע לסוף השורה הוא יורד לשורה הבאה) ויוצר מערך חדש כאשר ערכים מסויימים של המערך גדולים יותר (באזורים שהתאימו לצורה של הפילטר), בעוד אחרים קטנים יותר (באזורים שלא התאימו לצורת הפילטר).

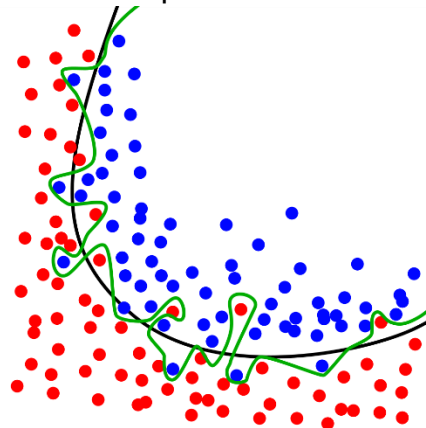
ניתן לראות את פעולת הכינוס ב gif הבא:
הכחול הבהיר – מערך הקלט
הכחול הכהה – הפילטר
הלבן – מפת התכונות שמקבלים



2. שכבות אגרגציה מקסימאלית (max pooling layers):

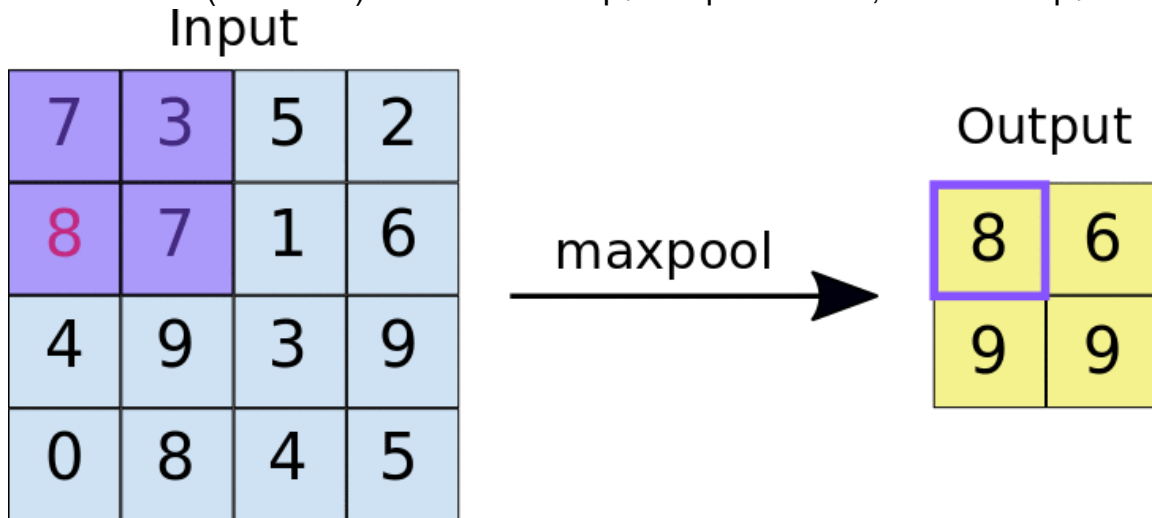
על מנת להבין את מטרתה של שכבה זו יש להבין את משמעות המושג "התאמת יתר" (overfitting).

overfitting הוא מצב המתרחש כאשר המודל מותאם יתר על המידה אל נתוני האימון, ומסתמך על רעש סטטיסטי בנתוני האימון על מנת לבצע חיזויים (זאת לאומת הסתמכות על התכונות הכלליות של הנתונים, אשר נכונות גם לגבי נתונים שהמודל לא ראה). מצב זה יגרום לכך שהמודל טוב מאוד בזיהוי נתוני האימון, אך גרוע מאוד בחיזוי נתונים חדשים. פעמים רבות ניתן לזהות Overfitting במהלך האימון כאשר ה validation accuracy וה validation loss גרועים משמעותית מה accuracy וה loss של האימון (שכן נתוני ה validation חדשים למודל ועל כן הוא מתקשה לזהות אותם במקרה שהוא מסתמך יתר על המידה על רעש הקיים בנתוני האימון).



בתמונה: הקו הירוק מייצג מודל בעל overfitting, בעוד השחור מייצג מודל תקין.

שכבת האגרגציה המקסימאלית היא שכבה אשר מקטינה את גודל המערך תוך כדי שהיא מקטינה את הרעש הסטטיסטי במערך (ולכן משאירה רק את הערכים הגדולים והחשובים בנתונים). בכך היא מקטינה את ההשפעה של שינויים קטנים על תוצאת החיזוי הסופית (ובכך מונעת overfitting). היא עושה זאת בכך שהיא עוברת על חלק ממערך הקלט בכל פעם בעזרת חלון בגודל קבוע (אשר זז (slides) דרך המערך) ובכל פעם בוחר את הערך הגבוה ביותר, אותו הוא לוקח למערך הפלט שהוא יוצר (ראה דוגמה).

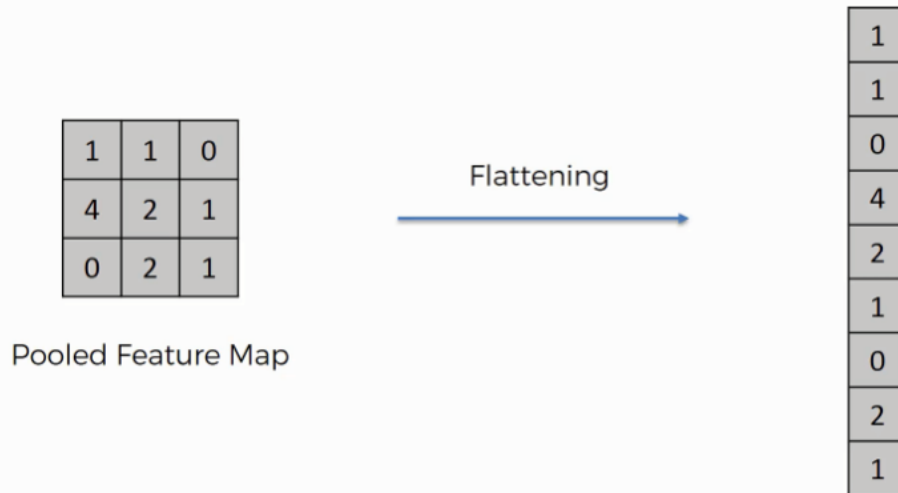


3. שכבות Dropout:

שכבה זו הופכת אחוז מסוים של פיקסלים ("ערכים במערך") לערך 0 בצורה רנדומלית, ובכך "מוחקת אותם" מהתמונה. ערכי שאר הפיקסלים מוגדלים בהתאם לכמות הפיקסלים שנמחקו, ובכך סכום הקלט לא משתנה. מטרת שכבה זו היא למנוע התאמת יתר (overfitting), בכך שהיא מונעת את העובדה שתוצאת המודל תהיה תלויה מאוד בכמה נירונים מסוימים (שינוי בהם יכול לגרום לשינוי מאסיבי בתוצאת המודל) – המודל לא יכול להסתמך על כמות מצומצמת של נירונים בשל כך שבכל פעם אנו בוחרים נירונים אחרים "למחוק".

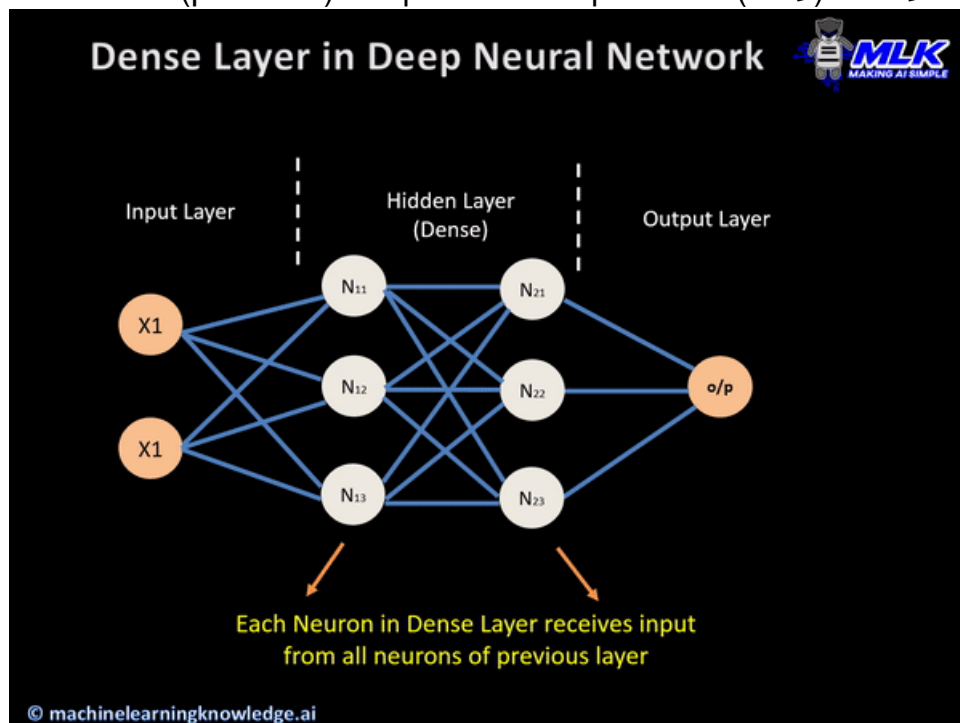
4. שכבת flatten אחת:

שכבה זו הופכת את המערך הדו מימדי שבו השתמשנו עד עכשיו כדי לייצג את התמונה, למערך חד מימדי על מנת שנוכל להכניס אותו לשכבות ה dense (ראה בהמשך).



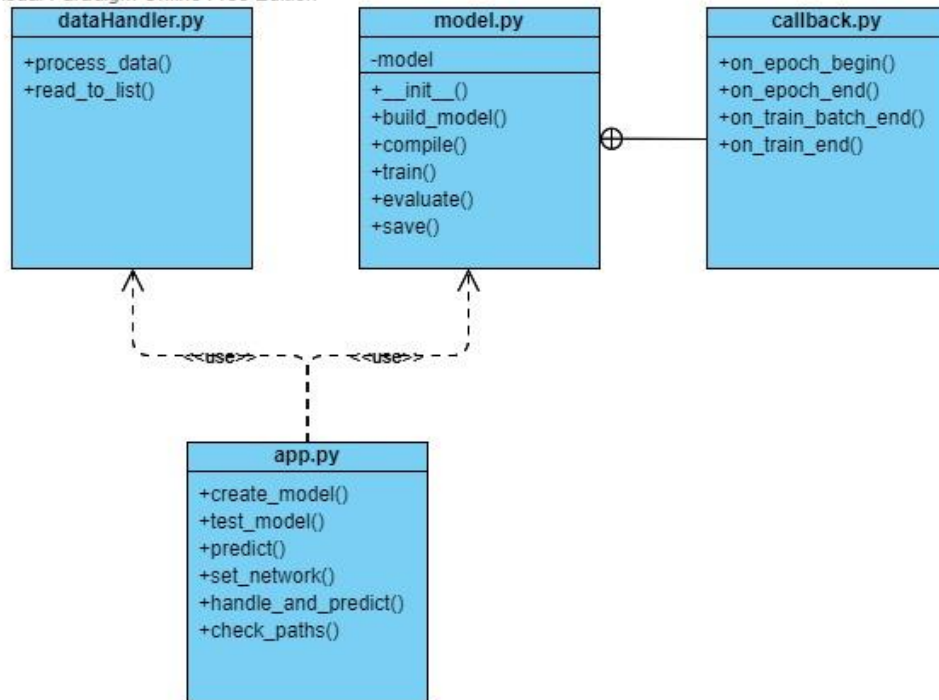
5. 4 שכבות Dense:

שכבה זו היא שכבת למידת המכונה הקלאסית: מקבלת מערך חד מימדי של ניוונים, אשר מחוברים לשכבת ה dense שאחריה, כאשר כל ניוון בשכבה אחת מחובר לכל ניוון בשכבה הבאה בעזרת weights (אשר מנתיבות את ההשפעה/תרומה של ניוון אחד לאחר שאליו הוא מחובר), אותם אנו משנים במהלך האימון. לניוונים בשכבות dense יש פונקציית אקטיבציה שמנתיבה את "הפעלתם" (ערכם) בהתאם לקלט אותו הם מקבלים (ראה סרטון).



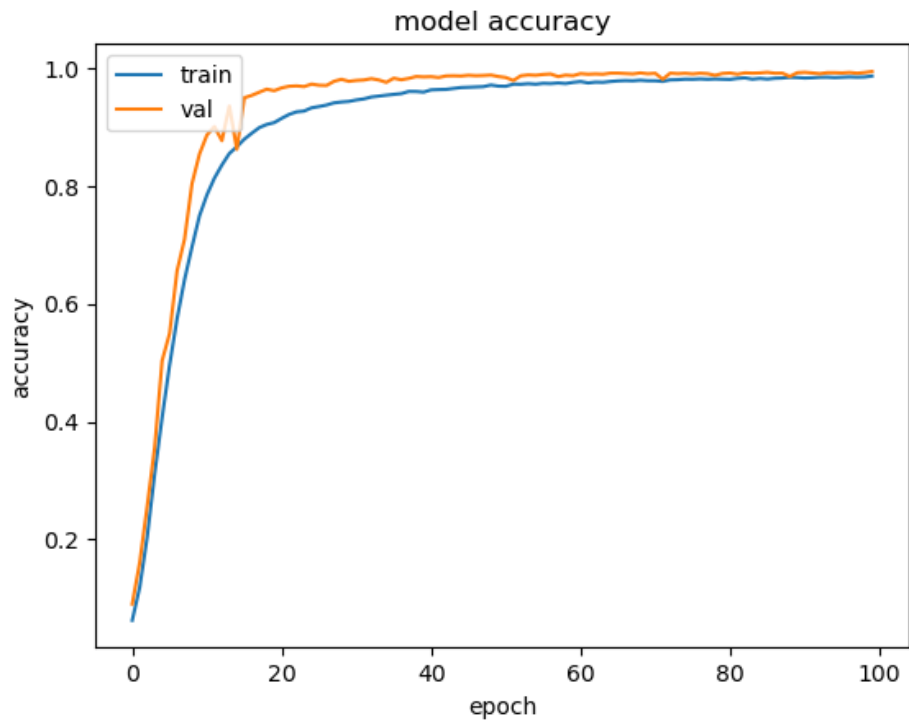
תיאור UML של המחלקות:

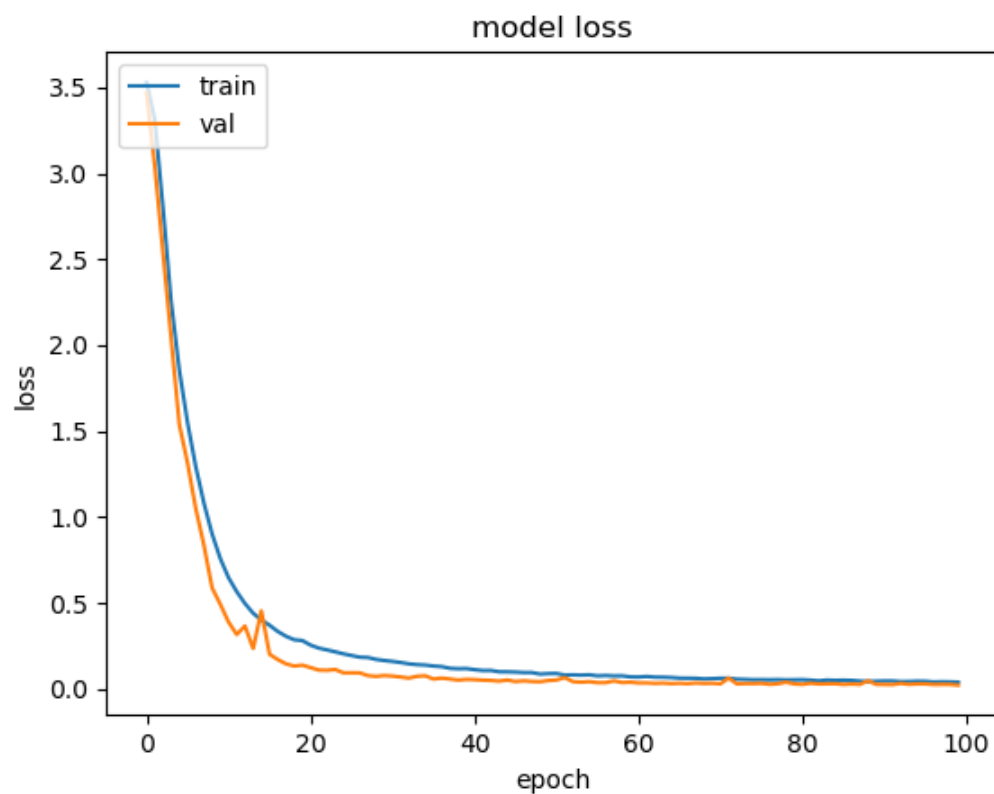
Visual Paradigm Online Free Edition



Visual Paradigm Online Free Edition

תוצאות שלב האימון:





דוח Hyper Parameters:

שם הפרמטר	ערך סופי
epochs	100
Early stopping patience	15
Dropout rate	20%
Max pool window size	2x2
Convolutional kernel size	5x5
Learning rate	0.01

שינויים שבוצעו ב Hyper Parameters :

במהלך כתיבת הפרויקט ביצעתי ניסוי ותהייה רב עם ה Hyper parameters על מנת למצוא את הערכים המספקים את התוצאות הטובות ביותר.

בין היתר, הפרמטרים אותם שיניתי היו ה dropout rate (עלה מ 0.1 אל 0.4, שם ראיתי ירידה בביצועים ולכן הורדתי בחזרה אל 0.2). דוגמא נוספת תהיה שינוי ה Convolutional kernel size,

אותו שינתי מ 3x3 אל 5x5, דבר ששיפר את ביצועי המודל. כמובן שבנוסף לפרמטרים השונים אותם שינתי, ביצעתי גם ניסוי ותהייה רבה עם מבנה המודל עצמו והפרמטרים השונים של השכבות שבו.

הסבר על פונקציית השגיאה:

פונקציית השגיאה היא פונקציה המייצגת את המרחק/הפרש בין הפלט לו ציפינו שהמודל יוציא (הפלט ה"נכון", בהתאם לlabels של הנתונים) לבין הפלט שהמודל הוציא בפועל. בעזרת פונקציית השגיאה אנו יכולים לדעת כמה רחוק המודל היה מלבצע את הפעולה שאליה הוא נבנה – מסיבה זו פונקציית השגיאה היינה בליבה של למידת המכונה.

ישנן דרכים שונות להגדיר את פונקציית השגיאה, כאשר דרכים שונות טובות למטרות שונות.

פונקציית השגיאה שבה אני בחרתי להשתמש היא categorical crossentropy. הסיבה לבחירה זו היא שפונקציית שגיאה זו פועלת בצורה טובה מאוד כאשר הפלט הוא התפלגות הסתברויות, כפי שהוא המצב אצלי – הפלט מהמודל מהווה את התפלגות ההסתברויות של הקלט להיות כל אחד מהשלטים אותם אני מזהה.

הנוסחה ל categorical crossentropy נובעת מתוך ביטוי בשם Kullback Leibler divergence:

$$D_{KL}(P \parallel Q) = \sum_{x \in X} P(x) * \log \frac{P(x)}{Q(x)}$$

נוסחה זו מבטאת את המרחק בין שני התפלגויות הסתברות.

פונקציית השגיאה crossentropy loss מהווה את המרחק בין התפלגות ההסתברויות הרצויה (הסתברות 0 למחלקות השגויות והסתברות 1 למחלקה הנכונה), לבין התפלגות הפלט. לכן, נוכל להשתמש ב KL divergence על מנת למצוא את מרחק זה, ולאחר חישוב נקבל:

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

כאשר \hat{y}_i היא הסתברות פלט של המחלקה מסדר i , בעוד y_i היא הסתברות הרצויה של המחלקה מסדר i .

הסבר על ייעול ההתכנסות (אופטימיזציה):

אלגוריתם האופטימיזציה הוא האלגוריתם איתו אנו משנים את ה weights של המודל במהלך הלמידה על מנת למזער את פונקציית השגיאה – ובכך ללמד את המודל, ולכן הוא מהווה את ליבו של תהליך למידת המכונה.

האלגוריתם בו אני משתמש הוא Stochastic Gradient Descent (sgd) – גרסה שונה במקצת מהאלגוריתם הקלאסי והבסיסי ביותר בלמידת מכונה – Gradient Descent.

Gradient Descent:

Gradient Descent היא שיטת אופטימיזציה למציאת מינימום של פונקציה רבת משתנים. במקרה של למידת מכונה, השיטה משומשת למציאת המינימום של פונקציית השגיאה כתלות במשקלים של המודל (ובכך "ללמד" את המודל ולשפר אותו). בשל העובדה שלא ישום מתמטית למצוא את המינימום המוחלט של פונקציה מורכבת בעלת נקודות מינימום רבות (פונקציה לא כמורה), שיטה זו, כמו שיטות רבות אחרות, מנסה למצוא את הנקודה הנמוכה ביותר באזור הכללי שלה.

על מנת להגיע לנקודת מינימום זו, ננוע בכל פעם כנגד לגרדיאנט של הפונקציה. גרדיאנט היינו מושג המקביל למושג הנגזרת בפונקציה רבת משתנים – הוא מייצג את השינוי בפלט של פונקציה בשינוי קטן בצורה אינסופית של הקלט. גודל התנועה כנגד הגרדיאנט תלוי בקצב הלמידה, שמהווה hyper parameter של המודל.

מציאת המינימום עובדת על פי איטרציה על הנוסחה הבאה:

$$\theta_{j+1} = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

כאשר:

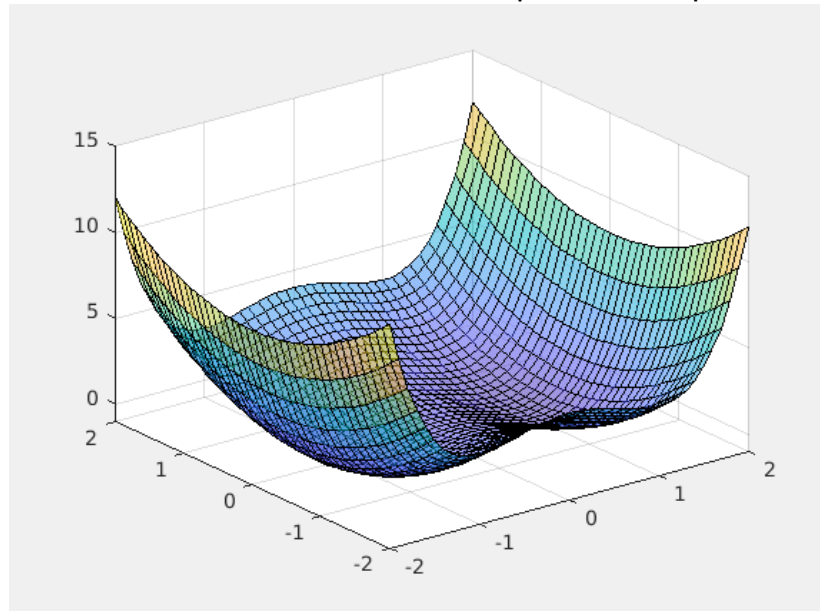
α – קצב הלמידה

θ_j – קלט הפונקציה (במקרה של למידת מכונה, ה weights), לאחר j איטרציות.

$J(\theta)$ – פונקציית השגיאה

לסיכום, ניתן לראות כי אנו משנים את המשקלים בהתאם לנגזרת של פונקציית השגיאה ביחס למשקלים – ובגלל שאנו מחסרים את הנגזרת, אנו זזים כנגד כיוון העלייה, ולכן לכיוון המינימום

האזורי ומקטינים את פונקציית השגיאה.



Stochastic Gradient Descent:

Sgd היינה גרסה של gradient descent אשר במקום לחשב את הגראדיאנט בעזרת כל הנתונים בכל פעם, היא מחשבת את הגראדיאנט בעזרת מקרה אחד רנדומלי בכל פעם. היתרון של שיטה זו הוא שהדבר חוסך מיליוני חישובים בכל epoch, ובכך לימוד המודל מהיר בהרבה. היחסרון היינו שבגלל שאנו לא מחשבים את הגראדיאנט של הנתונים כולם, התנהגות האופטימיזר עלולה להיות פחות צפויה ומתאימה לנותנים כולם, שכן אנו מתייחסים רק לחלקם.

התמודדות עם הטיה ושונות:

הטיה (bias) – מונח המתאר עד כמה המודל מפשט את ניתוח הנתונים ומניח הנחות שונות (לא בהכרח נכונות). הוא מוגדר כהבדל בין הפלט הממוצע של המודל אל הפלט הרצוי.

שונות (variance) – מונח המתאר עד כמה איכות חיזוי המודל משתנה כאשר משתמשים בנתונים שאינם נתוני האימון. הוא מוגדר כעקביות המודל בחיזוי סטים שונים של נתונים – הוא לא מודד את דיוק המודל ככלל.

כאשר ישנה הטיה גבוהה, נוצר מצב של underfitting, בעוד בשונות גבוהה נוצר מצב של overfitting.

בשל העובדה שהטיה ושונות מייצגים קונספטים הפוכים, יש צורך לאזן אותם בצורה שלא תגרום ל Underfitting או overfitting. הדרך המרכזית בה אני ניסיתי "לשחק" עם איזון זה הייתה בעזרת שינויים של מורכבות המודל עצמו (לדוגמא הוספה והורדה של שכבות קונבולוציה ושכבות dense). זאת בשל העובדה שמודל מורכב יותר יגדיל את ה variance ויקטין את bias, בעוד מודל פשוט יעשה את ההפך.

שלב היישום:

שימוש היישום במודל:

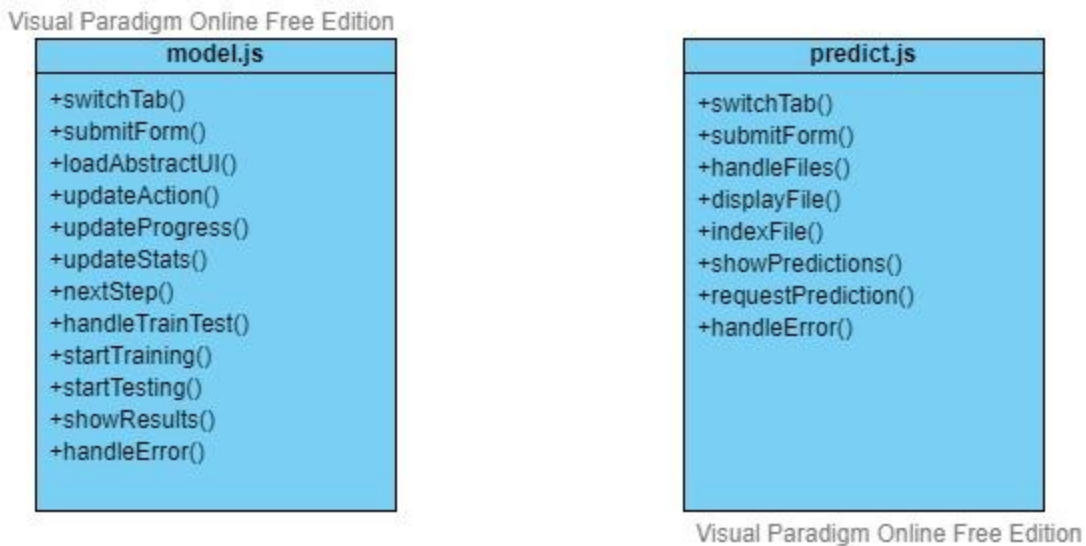
כפי שהמצב הוא פעמיים רבות בדפי אינטרנט, אני מחלק את הקוד לשני חלקים:

Frontend – החלק האחראי על הצגה והתנהגות ממשק המשתמש ועובד בתוך הדפדפן. זהו החלק הכתוב בשפות Html, Css, JavaScript ונמצא תחת תיקיית ה frontend.

Backend – החלק האחראי על ניהול המודל וניתוח הנתונים, ומהודר בפייטון. חלק זה נמצא תחת תיקיית ה backend.

שני החלקים של הקוד מתקשרים ביניהם בצורות שונות ושולחים מידע אחד לשני במידת הצורך. לכן, היישום קורא לפעולות במודל בכל פעם שהוא רוצה לבצע פעולה המערבת את המודל, בעוד backend קורא ליישום בכל פעם שהוא רוצה לעדכן ממשק המשתמש.

תרשים UML של מחלקות הממשק:



תיאור הטכנולוגיה שעל פיה מומש הממשק:

ממש המשתמש אותו יצרתי משתמש בספריית eel על מנת ליצור אפליקציה לוקאלית בתוך דפדפן האינטרנט, אשר מתקשרת עם פייטון. ספרייה זו מבוססת על ה electron framework, אשר מטרתה ליצור אפליקציות שולחן עבודה בעזרת טכנולוגיות מבוססות רשת.

בעזרת ספרייה זו, יכולתי להשתמש בטכנולוגיות רשת כגון html, css, JavaScript על מנת ליצור את ממשק המשתמש שלי. בעזרת טכנולוגיות אלו, ניתן ליצור ממשקי משתמש מורכבים ויפים בקלות.

הסבר קצר על טכנולוגיות אלו:

Html – היינה שפת סימון (markup language), אשר משומשת על מנת ליצור את מבנה דף האינטרנט.

Css – היינם פורמט לעיצוב דפי אינטרנט. הם מהווים את הגורם המשפיע ביותר על המראה הויזואלי של הדף.

JavaScript – שפת התכנות בה משתמשים על מנת להוסיף logic לדפי אינטרנט. בעזרתה תקשרתי מול פייטון והמודל.

תיאור קליטת קוד ה Data של החיזוי והתאמתו:

כאשר אנו שמים תמונה בדף החיזוי, אני מצרף אותה למערך של תמונות המכונות לחיזוי. זאת משום שהאפליקציה שלי תומכת בחיזוי מספר תמונות במקביל ועל כן אני לא מתחיל בניתוח התמונות עד שהמשתמש לוחץ על כפתור החיזוי. לאחר לחיצה על כפתור זה, כל אחת מהתמונות מומרת לבסיס 64, כך שהיא בפורמט של מחרוזת, ומועברת ל backend של האפליקציה, בפייטון. בשלב זה אני ממיר את המחרוזת בחזרה לאובייקט של תמונה ושומר אותה לוקאלית בקובץ זמני. בנוסף, אני בודק בשלב זה אם הפורמט של התמונה אינו מתאים לפורמט אותו אני מזהה, וממיר אותה בהתאם (לדוגמא jpg ל png). לאחר מכן, אני משנה את התמונה לגודל הקבוע של התמונות אותו רשת הניורונים מקבלת, הופך את התמונה למערך פיקסלים ומבצע נורמליזציה. לאחר מכן אני מכניס את התמונה לרשת הניורונים ומחזיר את התוצאה, אשר עוברת בחזרה אל ה frontend.

מדריך למפתח

רשימת הקבצים ותפקידם:

בפרויקט שלי ישנם קבצים רבים, כאשר חלקם אחראים על ניהול ממשק המשתמש (frontend), בעוד אחרים אחראיים על ניהול המודל וניתוח המידע (backend). חלוקה זו תרמה לסדר ולארגון במהלך כתיבת הפרויקט, ובכך תרמה לקלות בה יכולתי לפתור באגים ולהרחיב את סקאלת הפרויקט במידת הצורך.

שם קובץ	מיקום	תפקיד
app.py	\backend\app.py	הקובץ המרכזי של הפרויקט, הרצה שלו מפעילה את ממשק המשתמש. רוב תוכנו אחראי על התנהלות מול ממשק המשתמש וסיפוק פונקציות שונות בהם הממשק יכול להשתמש. קובץ זה משתמש בכל הפונקציות והמחלקות האחרות יחדיו על מנת ליצור חוויה אחידה למשתמש.
dataHandler.py	\backend\dataHandler.py	קובץ האחראי על קריאה וניתוח של הנתונים. קובץ זה מספק פונקציה אשר קוראת את הנתונים מהקובץ וממירה אותם במספר שלבים (ראה "איסוף והכנת הנתונים") לפורמט הסופי אותו המודל מקבל.
model.py	\backend\model.py	קובץ הכולל בתוכו מחלקה אשר מספקת פונקציות לבניית המודל, לאימון המודל ולבחינת המודל.
callback.py	\backend\callback.py	קובץ הכולל בתוכו מחלקת callback אשר מעדכנת את מסך הטעינה של ממשק המשתמש בזמן אימון המודל, בהתאם להתקדמות האימון.
Index.html	\frontend\html\index.html	קובץ html של המסך הראשי של ממשק המשתמש. זהו הקובץ המופעל כאשר מריצים את ממשק המשתמש לראשונה. קובץ זה מפעיל את שאר קבצי ממשק המשתמש בהתאם לפעולות המשתמש.
model.html	\frontend\html\model.html	קובץ html של דף האימון והבדיקה (test) של ממשק המשתמש. הקובץ כולל html של מספר חלוניות שונות (tabs) אשר מופעלים בהתאם לפעולה אשר המשתמש מבצע. קובץ זה מופעל דרך index.html במידת הצורך.
Predict.html	\frontend\html\predict.html	קובץ html של דף ה predict, בו המשתמש יכול לבצע predictions בעזרת מודלים. קובץ זה מופעל דרך index.html במידת הצורך.

model.js	\frontend\js\model.js	קובץ האחראי על כל ה logic המערב את עמודי ה train וה test בממשק המשתמש. כולל, בין היתר, את ניהול מערכת ה tabs, התנהלות מול ה backend במהלך אימון המודל או בדיקתו וניהול שגיאות במידה וישנן בעיות במהלך האימון או הבדיקה.
predict.js	\frontend\js\predict.js	קובץ האחראי על כל ה logic המערב את עמוד ה predict בממשק המשתמש. כולל, בין היתר, את הקליטה של התמונות, המרתן לבסיס 64, שליחתן אל ה backend וקבלת התוצאות.
index.css	\frontend\css\index.css	קובץ האחראי על עיצוב העמוד הראשי של ממשק המשתמש.
model.css	\frontend\css\model.css	קובץ האחראי על עיצוב עמודי ה test וה train בממשק המשתמש.
predict.css	\frontend\css\predict.css	קובץ האחראי על עיצוב עמוד ה predict בממשק המשתמש.
.gitignore	.gitignore	רשימת קבצים שמערכת ניהול הגרסאות git, בה השתמשת, תתעלם מהם.

הסברי פעולות ותוכן:

פעולות בקובץ app.py:

1. create_model(data_path, model_path, pickle_data)

תפקיד:

פעולה אשר לה ממשק המשתמש קורא על מנת לנתח את הנתונים, לאמן את המודל, ולאחר מכן לבחון אותו. פעולה זו קוראת לכל הפעולות הרלוונטיות על מנת לבצע את שלבים אלו.

מקבלת:

data_path : string – מיקום הנתונים בעזרתם מאמנים את המודל.
model_path : string – המיקום אליו שומרים את המודל לאחר האימון.
pickle_data : bool – האם על הפעולה לחפש נתונים אשר כבר עברו עיבוד (הפעולה תחפש קובץ pickle בתוך מיקום הנתונים אשר היא קיבלה).

מחזירה:

כלום

אופן פעולה:

בתחילה הפעולה בודקת אם עליה לחפש נתונים בתור קובץ pickle, אם כן היא בודקת אם הוא קיים ומוציאה את הנתונים מהקובץ אל הזיכרון. במידה והקובץ לא קיים או שארגומנט pickle_data היה false, הפעולה קוראת לפעולה process_data במודול dataHandler.py, שתנתח את הנתונים מאפס. לאחר מכן הפעולה יוצרת מופע של מחלקת Model הנמצאת במודול model.py, וקוראת לפעולות train ו build_model מתוך מחלקה זו, אשר בונות ומאמנות את המודל בעזרת הנתונים אותם הפעולה ניתחה / אספה מתוך קובץ pickle. לאחר מכן היא קוראת לפעולת evaluate מתוך מחלקת המודל על מנת לבחון אותו, שומרת את המודל שיצרנו ושולחת את התוצאות אל ממשק המשתמש בעזרת פעולה eel.showResults.

2. test_model(data_path, model_path, pickle_data)

תפקיד:

פעולה אשר לה ממשק המשתמש קורא על מנת לנתח את הנתונים, ולבחון את המודל. פעולה זו קוראת לכל הפעולות הרלוונטיות על מנת לבצע את שלבים אלו.

מקבלת:

data_path : string – מיקום הנתונים בעזרתם בוחנים את המודל.
model_path : string – מיקום המודל אותו נבחן.
pickle_data : bool – האם על הפעולה לחפש נתונים אשר כבר עברו עיבוד (הפעולה תחפש קובץ pickle בתוך מיקום הנתונים אשר היא קיבלה).

מחזירה:

כלום

אופן פעולה:

בתחילה טוענת את המודל מתוך model_path אל הזיכרון. לאחר מכן, הפעולה בודקת אם עליה לחפש נתונים בתור קובץ pickle, אם כן היא בודקת אם הוא קיים ומוציאה את הנתונים מהקובץ אל הזיכרון. במידה והקובץ לא קיים או שארגומנט pickle_data היה false, הפעולה קוראת לפעולה process_data במודול dataHandler.py, שתנתח את הנתונים מאפס. לאחר מכן היא קוראת לפעולת evaluate מתוך המודל על מנת לבחון אותו, ושולחת את התוצאות אל ממשק המשתמש בעזרת פעולה eel.showResults. במהלך הפעולה ישנן קריאות לפעולה eel.updateAction, אשר מעדכנת את מסך הטעינה בממשק המשתמש עם הנתונים המעודכנים של השלב בו אנו נמצאים בפעולה.

3. predict(img)

תפקיד:

פעולה פנימית אשר מקבלת אובייקט של תמונה, מבצעת את תהליך ה preprocessing הסטנדרטי על מנת להיכנס למודל ומחזירה את ניחוש (prediction) המודל לגבי התמונה.

מקבלת:

Img : PIL.Image – אובייקט התמונה לגביו רוצים לקבל ניחוש.

מחזירה:

String – ניחוש המודל לגבי התמונה אותה הפעולה קיבלה.

אופן פעולה:

פעולה זו מקבלת אובייקט של תמונה, ממירה אותו לפורמט RGB, משנה את הרזולוציה שלו לגודל המתאים למודל של 30x30, הופכת אותו למערך פיקסלים, מנרמלת אותו ולאחר מכן קוראת ל model.predict, שזוהי פעולה פנימית של המודל אשר מבצעת חיזוי על נתונים ללא label. לאחר מכן הפעולה מחזירה את התוצאה.

4. set_network(path)

תפקיד:

פעולה שלה ממשק המשתמש קורא על מנת להגדיר את המשתנה הגלובאלי network, שמגדיר את המודל בו הפעולה predict (ראה למעלה) תשמש על מנת לבצע ניחושים.

מקבלת:

path : string – מיקום המודל אותו הפעולה תשים במשתנה הגלובאלי network.

מחזירה:

כלום

אופן פעולה:

מקבלת את מיקום המודל, קוראת ל keras.models.load_model, שזוהי פעולה של הספרייה לטעינת המודל, ומכניסה את התוצאה אל תוך המשתנה הגלובאלי network. במידה וישנה שגיאה הפעולה קוראת ל eel.handleError, פעולה של ממשק המשתמש לניהול שגיאות.

5. handle_and_predict(img_data_uri)

תפקיד:

פעולה שלה ממשק המשתמש קורא על מנת לקבל את ניחוש המודל (prediction) לגבי תמונה מסוימת.

מקבלת:

img_data_uri : string – נתוני התמונה לגביה רוצים לקבל ניחוש, בפורמט של בסיס 64.

מחזירה:

string – את ניחוש המודל לגבי התמונה.

אופן פעולה:

פעולה אשר מקבלת תמונה בפורמט של בסיס 64, ממירה אותה לקובץ תמונה השמור על גבי הכונן בעזרת הפעולה request.urlopen של הספרייה urllib, לאחר מכן היא בודקת את פורמט התמונה ובמידה והוא לא png, היא קוראת לפעולה handle_other_filetypes, על מנת להמיר אותה ל png (שכן זהו הפורמט שהמודל מקבל). לאחר מכן היא קוראת לפעולה predict על מנת לקבל חיזוי לתוצאה ומחזירה את התוצאה. במידה וישנה שגיאה הפעולה מחזירה מחרוזת בפורמט של "error;" ולאחריו תיאור השגיאה (היא לא קוראת לפונקציית ניהול השגיאות של ממשק המשתמש מפני שפונקציה זו עוצרת את כל תהליך החיזוי ומעבירה למסך שגיאות – זוהי לא ההתנהגות הרצויה במקרה זה).

6. check_paths(data_path, model_path)

תפקיד:

פעולה שלה ממשק המשתמש קורא על מנת לבדוק האם ישנן בעיות במיקום הנתונים והמודל.

מקבלת:

data_path : string – ארגומנט לא חובה. מיקום הנתונים.

model_path : string – ארגומנט לא חובה. מיקום המודל.

מחזירה:

string[] – רשימת שגיאות המערבות את מיקום הנתונים ומיקום המודל.

אופן פעולה:

פעולה זו מורכבת ממספר בדיקות כדי לאתר בעיות במיקומי המודל והנתונים לפני שהם יגרמו לשגיאה בשלב האימון/הבדיקה/החיזוי (הבדיקות מתבצעות רק על הארגומנטים שהתקבלו). בשלב ראשון הפעולה בודקת אם המיקומים שהתקבלו קיימים. לאחר מכן היא בודקת אם מיקום הנתונים כולל את התיקייה/קבצים שצריכים להיות שם. לאחר מכן היא מחזירה את רשימת השגיאות שנמצאו.

7. handle_other_filetypes(path)

תפקיד:

פעולה פנימית הממירה קובץ תמונה מפורמט לא מתאים אל png.

מקבלת:

path : string – מיקום הקובץ אותו הפעולה ממירה.

מחזירה:

כלום

אופן פעולה:

פעולה זו לוקחת מיקום של קובץ בפורמט כלשהו, שומרת אותו לזכרון, ממירה אותו ל RGB, שומרת אותו מחדש בפורמט png ולאחר מכן מוחקת את הקובץ המקורי.

פעולות בקובץ dataHandler.py:

1. process_data(data_path)

תפקיד:

פעולה אשר קוראת את הנתונים, מעבירה אותם לזיכרון ומתאימה אותם לפורמט אותו המודל מקבל (ראה "איסוף והכנת הנתונים"). נקראת על ידי app.py בכל פעם שיש צורך בניתוח נתונים גולמיים.

מקבלת:

data_path : string – מיקום הנתונים אשר הפעולה תנתח.

מחזירה:

anonymous object – מחזירה אובייקט אנונימי המחזיק את המידע הנאסף בפעולה (ראה "אופן פעולה").

אופן פעולה:

זוהי פעולה המאגדת את כל תהליך הכנת הנתונים הגולמיים אל נתונים שהמודל יכול לקבל. בפעולה זו, בכל פעם שהיא מסיימת לבצע שלב מסוים בתהליך הכנת הנתונים נקראת הפעולה eel.updateAction על מנת לעדכן את ממשק המשתמש בהתקדמות. בשלב ראשון הפעולה מכניסה את הנתונים של קובץ ה csv הכולל את מיקומי תמונות המשומשות לאימון וה labels שלהם אל תוך dataframe. לאחר מכן נקראת הפעולה read_to_list אשר קוראת את תמונות אלו מתוך הכונן, ומכניסה אותן אל הזיכרון תוך ביצוע חלק מתהליך הכנת הנתונים על כל תמונה. לאחר מכן אותו התהליך מתבצע על

התמונות המשומשות לבדיקת המודל (הכנסת קובץ ה csv ל dataframe ולאחר מכן קריאה לפעולת read_to_list). לאחר שלב זה ישנם ארבע מערכים : train_x | test_x, train_y | test_y, אשר כוללים את ה labels של נתונים אלו, בהתאמה. בשלב הבא, הפעולה מבצעת על מערכי הפקיסלים נורמליזציה (מחלקת את כל הערכים ב 255, שכן הערך המקסימאלי של פיקסל היינו 255). בשלב שלאחר מכן, הפעולה מבצעת one hot encoding למערכי ה labels (ראה "תהליך עיבוד הנתונים שלב 2: ביצוע One hot encoding"), בעזרת הפעולה של keras, to_categorical. לאחר מכן הפעולה מבצעת train test split לנתוני האימון בעזרת פעולה של הספרייה sklearn, זאת על מנת לקבל נתוני validation למעקב אחרי מהלך האימון. בשלב האחרון הפעולה שמה את כל מערכי הנתונים השונים שיצרנו אל תוך אובייקט אנונימי, על מנת להעביר אותם בין פעולות ולשמור אותם בקלות רבה יותר אחר כך. מערכי הנתונים בסוף פעולה זו (אותם אנו שמים באובייקט האנונימי), הינם:

- xtrain – מערך הפיקסלים של תמונות האימון
- ytrain – מערך ה labels של תמונות האימון.
- xval – מערך הפיקסלים של התמונות שישמשו ל validation במהלך האימון.
- yval – מערך ה labels של התמונות שישמשו ל validation במהלך האימון.
- xtest – מערך הפיקסלים של תמונות הבדיקה (evaluation).
- ytest – מערך ה labels של תמונות הבדיקה (evaluation).

2. read_to_list(path, df, bar, bar_label)

תפקיד:

פעולה פנימית אשר קוראת את הנתונים מתוך הכון ומכניסה אותם אל הזיכרון תוך ביצוע חלק מתהליך המרת הנתונים אל הפורמט אותו המודל מקבל. משומשת על ידי process_data.

מקבלת:

path : string - מיקום הנתונים בכון.
df : pandas.DataFrame – פורמט נתונים הכולל את מיקומי הקבצים בתוך מבנה הנתונים.
bar : ChargingBar – לא ארגומנט חובה. מאפשר בחירה של קו טעינה (בטרמינל) במראה מותאם אישית.
bar_label : string – לא ארגומנט חובה. מאפשר בחירה של תווית מותאמת אישית לקו הטעינה (בטרמינל).

מחזירה:

כלום

אופן פעולה:

פעולה זו עוברת על כל מיקומי הקבצים שנמצאים ב dataframe אותו היא קיבלה ולכל אחד מהם היא משנה את הרזולוציה ל 30x30 (רזולוציית התמונה אותה המודל מקבל), הופכת את התמונה למערך פיקסלים ומוסיפה את המערך אל רשימת תמונות. בכל פעם שהיא מסיימת תמונה היא מעדכנת את סרגלי הטעינה שבממשק המשתמש ובטרמינל בעזרת הפעולות `bar.next` ו `eel.updateProgress`.

פעולות בקובץ `model.py`:

כל הפעולות בקובץ זה נמצאות תחת המחלקה `Model`.

1. `init (model)`

תפקיד:

פעולה בונה למחלקת `Model`.

מקבלת:

`model : keras.model` – ארגומנט לא חובה. מאפשר להגדיר את המופע עם מודל מוכן מראש.

מחזירה:

כלום

אופן פעולה:

מאפשרת להגדיר את פרמטר המודל של המחלקה עם מודל מוכן מראש.

2. `build_model()`

תפקיד:

פעולה שבונה ולאחר מכן מקמפלט (`compiles`) את המודל. נקראת על ידי `app.py` במידת הצורך.

מקבלת:

כלום

מחזירה:

כלום

אופן פעולה:

פעולה זו בונה את המודל בכך שהיא מוסיפה אליו את כל השכבות הרלוונטיות, מגדירה את פרמטר המודל של המחלקה כמודל שהיא הכינה ולאחר מכן קוראת לפעולה compile על מנת לקמפל את המודל.

compile() .3

תפקיד:

פעולה פנימית שמקמפלת (compiles) את המודל. נקראת על ידי build_model.

מקבלת:

כלום

מחזירה:

כלום

אופן פעולה:

קוראת לפעולה הפנימית של המודל model.compile על מנת לקמפל את המודל עם הגדרות האופטימזיר והloss הרלוונטים.

evaluate(data) .4

תפקיד:

פעולה שבוחנת (evaluates) את המודל.

מקבלת:

data : anonymous object – הנתונים שאיתם הפעולה תבחן את המודל.

מחזירה:

Float, Float – מחזירה את ה accuracy או loss של הבחינה.

אופן פעולה:

קוראת לפעולה של המודל model.evaluate ומחזירה את התוצאות, תוך עדכון מסך הטעינה בממשק המשתמש.

train(data) .5

תפקיד:

פעולה שמתחילה את אימון המודל. נקראת על ידי app.py במידת הצורך.

מקבלת:

data : anonymous object – הנתונים שאיתם הפעולה תאמן את המודל.

מחזירה:

כלום

אופן פעולה:

מגדירה early stopping callback ומתחילה את האימון בעזרת הפעולה fit של המודל.

6. save(path)

תפקיד:

פעולה ששומרת את המודל על הדיסק.

מקבלת:

path : string – מיקום שבו הפעולה תשמור את המודל

מחזירה:

כלום

אופן פעולה:

פונקציה עוטפת(wrapper method) לפונקציה של המודל model.save.

פעולות בקובץ callback.py:

1. on_epoch_begin(epoch, logs)

תפקיד:

פעולה שרצה בכל תחילת epoch ומעדכנת את מסך הטעינה בממשק המשתמש.

מקבלת:

Epoch : int – מספר ה epoch הנוכחי.

Logs : dictionary – ארגומנט לא חובה. נתונים על תהליך הלמידה עד כה

מחזירה:

כלום

אופן פעולה:

קוראת לפעולה של ממשק המשתמש updateAction על מנת לעדכן את מספר ה epoch במסך הטעינה.

2. on_epoch_end(epoch, logs)

תפקיד:

פעולה שרצה בכל סוף epoch ומעדכנת את מסך הטעינה בממשק המשתמש.

מקבלת:

Epoch : int – מספר ה epoch הנוכחי.
Logs : dictionary – נתונים על תהליך הלמידה עד כה

מחזירה:

כלום

אופן פעולה:

קוראת לפעולה updateStats של ממשק המשתמש על מנת לעדכן את נתוני loss ו accuracy המופיעים במסך הטעינה.

3. on_train_batch_end(batch, logs)

תפקיד:

פעולה שרצה בכל סוף batch ומעדכנת את מסך הטעינה בממשק המשתמש.

מקבלת:

batch : int – מספר ה batch הנוכחי.
Logs : dictionary – נתונים על תהליך הלמידה עד כה

מחזירה:

כלום

אופן פעולה:

קוראת לפעולה updateProgress של ממשק המשתמש על מנת לקדם את סרגל הטעינה בהתאם להתקדמות ה epoch.

4. on_train_end(logs)

תפקיד:

פעולה שרצה בסוף האימון ומעדכנת את מסך הטעינה בממשק המשתמש.

מקבלת:

Logs : dictionary – נתונים על תהליך הלמידה עד כה

מחזירה:

כלום

אופן פעולה:

קוראת לפעולה nextStep של ממשק המשתמש על מנת לקדם את סרגל השלבים שבמסך הטעינה לשלב הבא (לסמן ששלב ה train הסתיים).

פעולות בקובץ model.js:

switchTab(index) .1

תפקיד:

פעולה המחליפה בין tabs בתוך model.html. tab היינו div בhtml (דיב הוא container הכולל חלק מדף אינטרנט) המייצג חלון אחד בממשק המשתמש.

מקבלת:

Index : int – מספר ה tab אליו הפעולה תעביר את הדף.

מחזירה:

כלום

אופן פעולה:

פעולה זו מקבלת את רשימת כל החלוניות מתוך ה DOM, לאחר מכן בודקת אם קיים כרגע חלון "מופעל" על פי המשתנה הגלובאלי currentTab, אם כן היא מכבה אותו. לאחר מכן הפעולה מדליקה את החלונית בעל ה index אותו היא קיבלה.

submitForm() .2

תפקיד:

פעולה המנהלת את המעבר בין מסך הקלט של מיקומי הנתונים והמודל, אל מסכי האימון או test. פעולה זו בודקת את תקינות הקלט ומפעילה את מסך השגיאות במידת הצורך.

מקבלת:

כלום

מחזירה:

כלום

אופן פעולה:

בשלב הראשון הפונקציה בודקת אם הקלט של מיקום הנתונים ומיקום המודל תקין. אם לא, הפונקציה יוצאת מיידית. אם הקלט תקין, היא בודקת אם קיימות שגיאות במסך השגיאות כבר (בשל כך שהמשתמש כבר "עבר" בו). אם כן, שגיאות אלו נמחקות על מנת שלא יופיעו כפילויות או שגיאות מיותרות במידה והמשתמש יגיע למסך זה שוב. לאחר מכן הפעולה קוראת לפעולה `check_train_paths` שנמצאת ב `backend` על מנת לבדוק אם קיימות בעיות אחרות עם הקלט. הפונקציה מספקת `callback` שירוצץ כאשר `check_train_paths` תסיים לרוץ. אופן פעולת ה `callback`: תחילה ה `callback` בודק אם ישנן איזשהן בעיות עם הקלט ש `check_train_paths` מצא. אם לא, הוא קורא ל `handleTrainTest` על מנת לעבור למסך טעינת האימון/הבדיקה. אם ישנן שגיאות בקלט, ה `callback` מעביר למסך השגיאות, עובר על רשימת השגיאות שנמצאו ומציג כל אחת מהן על המסך.

3. `handleTrainTest()`

תפקיד:

רצה במעבר בין מסך הקלט למסך ה `train\test`. בהתאם למשתנה גלובלי של האפליקציה (שנקבע בהתאם לכפתור עליו המשתמש לוחץ במסך הראשי), פעולה זו משנה את הטקסט והערכים בדף לערכים המתאימים לדף הרלוונטי (`train\test`). בנוסף, היא מפנה אל הפעולה הרלוונטית המתחילה את התהליך הרלוונטי (`train\test`).

מקבלת:

כמות בלתי מוגבלת של ארגומנטים אשר אותם היא מעבירה אל הפעולה המתאימה: `startTraining` במקרה שהמסך הרצוי הוא `train` או `startTesting` במקרה שהמסך הרצוי הוא `test`.

מחזירה:

כלום

אופן פעולה:

תחילה, הפעולה בודקת אם הדף בו נמצא המשתמש עכשיו נמצא במצב `test`. אם לא, הפעולה מיידית קוראת ל `startTraining` עם כל הארגומנטים שהיא קיבלה, שכן זהו מצב ברירת המחדל. אם כן, הפעולה משנה את כותרת החלונית שלאחר מכן ליותרת הרלוונטית למצב הבדיקה ומשנה עוד מספר אלמנטים וויזואלים על מנת להתאים את הדף למצב הבדיקה, ולאחר מכן קוראת ל `startTesting` עם כל הארגומנטים שהיא קיבלה.

startTraining(dataPath, modelPath, pickle) 4.

תפקיד:

פעולה אשר מתחילה את תהליך האימון בbackend ומעבירה את ממשק המשתמש ל tab של האימון.

מקבלת:

dataPath : string – מיקום הנתונים איתם יתבצע האימון.
modelPath : string - המיקום בו ישמר המודל לאחר האימון.
pickle : bool – האם יש לחפש אחר נתונים שנאספו בפורמט pickle.

מחזירה:

כלום

אופן פעולה:

מעבירה לחלונית בעלת ה index השני (חלונית טעינת האימון\בדיקה), ולאחר מכן קוראת לפעולה create_model אשר נמצאת ב backend על מנת להתחיל את תהליך הכנת הנתונים, בניית המודל ואימונו.

startTesting(dataPath, modelPath, pickle) 5.

תפקיד:

פעולה אשר מתחילה את תהליך הtesting בbackend ומעבירה את ממשק המשתמש ל tab של ה testing.

מקבלת:

dataPath : string – מיקום הנתונים איתם יתבצע הבדיקה.
modelPath : string - המיקום בו נמצא המודל אותו אנו בודקים.
pickle : bool – האם יש לחפש אחר נתונים שנאספו בפורמט pickle.

מחזירה:

כלום

אופן פעולה:

מעבירה לחלונית בעלת ה index השני (חלונית טעינת האימון\בדיקה), ולאחר מכן קוראת לפעולה test_model אשר נמצאת ב backend על מנת להתחיל את תהליך הכנת הנתונים, ובדיקת המודל.

6. updateAction(txt, abstract)

תפקיד:

פעולה אשר ה Backend קורא לה במהלך מסך הטעינה של ה train\test כאשר הוא מעוניין לעדכן את התהליך המתבצע באותו רגע. פעולה זו מעדכנת בהתאם את הטקסט המתאר את התהליך ואת סרגל הטעינה/סמלון הטעינה.

מקבלת:

string : Txt – הטקסט המתאר את התהליך המתרחש באותו הרגע
bool : Abstract – ארגומנט לא חובה. במידה והוא true, יופיע סמלון טעינה במקום סרגל טעינה עד שפעולה זו תיקרא שוב. אופציה זו משומשת כאשר התהליך המתבצע היינו תהליך שלא ניתן לעקוב אחרי התקדמותו בצורה מתמשכת.

מחזירה:

כלום

אופן פעולה:

בשלב הראשון הפעולה בודקת אם הארגומנט abstract הוא true, אם כן, הפעולה קוראת ל loadAbstractUI על מנת להוריד את סרגל הטעינה ולהחליפו באייקון הטעינה המופשט. אם לא, הפעולה מאתחלת את התקדמות סרגל הטעינה (שכן תהליך חדש החל בעל טעינה משלו). לאחר מכן (ללא קשר לערכו של הארגומנט abstract), הפעולה מעדכנת את הטקסט המתאר את התהליך המתרחש בזמן הטעינה באותו רגע לזה שהיא קיבלה (בארגומנט txt).

7. loadAbstractUI()

תפקיד:

פעולה פנימית אשר מחליפה את סרגל הטעינה אל אייקון טעינה אשר לא מציג התקדמות. נקראת על ידי updateAction על פי ארגומנט ה abstract (ראה למעלה).

מקבלת:

כלום

מחזירה:

כלום

אופן פעולה:

הפעולה מסתירה את סרגל הטעינה ובמקומו מציגה את אייקון הטעינה המופשט. בנוסף, היא מסתירה את נתוני ה accuracy וה loss, במידה והם הוצגו קודם לכן.

8. updateProgress(count, total)

תפקיד:

פעולה אשר ה Backend קורא לה במהלך מסך הטעינה של ה train\test כאשר הוא מעוניין להזיז את סרגל הטעינה. בשימוש בעיקר בתהליכים המשכיים, שבהם בכל "סבב" פעולה זו נקראת. לדוגמא, פעולה זו נקראת בכל batch במהלך האימון על מנת לקדם את סרגל הטעינה.

מקבלת:

Count : int – ההתקדמות שהתבצעה עד עכשיו.
Total : int – ההתקדמות הכוללת. התקדמות סרגל הטעינה מוגדרת כ count/total .

מחזירה:

כלום

אופן פעולה:

הפעולה מעדכנת את התקדמות סרגל הטעינה לערך חדש של count/total , בהתאם לארגומנטים count ו total אותם היא קיבלה.

9. updateStats(acc, loss)

תפקיד:

פעולה אשר נקראת על ידי ה backend ומעדכנת את נתוני ה accuracy ו loss המוצגים בעמוד האימון בממשק המשתמש. פעולה זו נקראת דרך callback בסוף כל epoch.

מקבלת:

Acc : float – נתון ה accuracy המעודכן.
Loss: float – נתון ה loss המעודכן.

מחזירה:

כלום

אופן פעולה:

הפעולה מציגה את נתוני ה accuracy ו loss, אם הם לא הוצגו עד עכשיו, ומעדכנת את ערכם בהתאם לארגומנטים acc ו loss אותם היא קיבלה.

nextStep().10

תפקיד:

פעולה אשר נקראת על ידי ה backend ומעדכנת (מקדמת לשלב הבא) את תצוגת השלבים הנמצאת בראש עמודי ה train וה test.

מקבלת:

כלום

מחזירה:

כלום

אופן פעולה:

הפעולה עוברת על כל אלמנטי ה html מסוג li ומחפשת את הראשון שאינו מסומן כ active, שזהו השלב הבא (שכן כל השלבים שלפניו סומנו כהושלמו). את שלב זה הפעולה מסמנת כהושלם.

showResults(acc, loss).11

תפקיד:

פעולה אשר נקראת על ידי ה backend ומעבירה את ממשק המשתמש אל ה tab אשר מראה את תוצאות ה train או ה test.

מקבלת:

float : Acc – הדיוק הסופי של המודל לאחר בחינתו (גם במקרה של train וגם ב test).
float : Loss – ה loss הסופי של המודל לאחר בחינתו (גם במקרה של train וגם ב test).

מחזירה:

כלום

אופן פעולה:

פעולה זו מעבירה את החלונית לחלונית בעלת Index 3 (חלונית הצגת תוצאות האימון או הבדיקה) ומעדכנת את התוצאות המוצגות בהתאם לארגומנטים שהיא קיבלה.

handleError(txt).12

תפקיד:

פעולה אשר נקראת על ידי ה backend במידה ונתפסה שגיאה בשלב כלשהו. פעולה זו תעלה את לשונית השגיאות ותציג את השגיאה שהתקבלה.

מקבלת:

txt : string – מחרוזת המתארת את השגיאה שהתקבלה.

מחזירה:

כלום

אופן פעולה:

פעולה זו מעבירה אל מסך השגיאות, ומציגה את השגיאה שהתקבלה בו.

פעולות בקובץ Predict.js:

switchTab(index).1

תפקיד:

פעולה המחליפה בין tabs בתוך predict.html. tab היינו div בhtml (דיב הוא container הכולל חלק מדף אינטרנט) המייצג חלון אחד בממשק המשתמש.

מקבלת:

Index : int – מספר ה tab אליו הפעולה תעביר את הדף.

מחזירה:

כלום

אופן פעולה:

פעולה זו מקבלת את רשימת כל החלוניות מתוך ה DOM, לאחר מכן בודקת אם קיים כרגע חלון "מופעל" על פי המשתנה הגלובאלי currentTab, אם כן היא מכבה אותו. לאחר מכן הפעולה מדליקה את החלונית בעל ה index אותו היא קיבלה.

submitForm().2

תפקיד:

פעולה המנהלת את המעבר בין מסך הקלט של מיקום המודל, אל מסך החיזיון(prediction). פעולה זו בודקת את תקינות הקלט ומפעילה את מסך השגיאות במידת הצורך.

מקבלת:

כלום

מחזירה:

כלום

אופן פעולה:

בשלב הראשון הפונקציה בודקת אם הקלט של מיקום המודל תקין. אם לא, הפונקציה יוצאת מיידית. אם הקלט תקין, היא בודקת אם קיימות שגיאות במסך השגיאות כבר (בשל כך שהמשתמש כבר "עבר" בו). אם כן, שגיאות אלו נמחקות על מנת שלא יופיעו כפילויות או שגיאות מיותרות במידה והמשתמש יגיע למסך זה שוב. לאחר מכן הפעולה קוראת לפעולה `check_train_paths` שנמצאת ב `backend` על מנת לבדוק אם קיימות בעיות אחרות עם הקלט. הפונקציה מספקת `callback` שירוצץ כאשר `check_train_paths` תסיים לרוץ. אופן פעולת ה `callback`: תחילה ה `callback` בודק אם ישנן איזשהן בעיות עם הקלט ש `check_train_paths` מצא. אם לא, הוא קורא לפעולה `set_network` של ה `backend` על מנת להגדיר את המודל בו הוא ישתמש לחיזויים כמודל שהוכנס כקלט. בנוסף, ה `callback` מעביר את ממשק המשתמש אל החלונית השנייה (חלונית החיזוי). אם ישנן שגיאות בקלט, ה `callback` מעביר למסך השגיאות, עובר על רשימת השגיאות שנמצאו ומציג כל אחת מהן על המסך.

3. `preventDefaults(e)`

תפקיד:

פעולה פנימית אשר מבטלת את ההתנהגות ברירת המחדל של אתר האינטרנט בניהול האירוע אותו הפעולה מקבלת כארגומנט.

מקבלת:

`Event` : `e` – האירוע שבו אנו רוצים למנוע את התנהגות ברירת המחדל של אתר האינטרנט.

מחזירה:

כלום

אופן פעולה:

קוראת לפעולת המובנות של האירוע, `stopPropagation` ו `preventDefault`.

4. `handleFiles(e)`

תפקיד:

פעולה פנימית אשר מופעלת כאשר המשתמש גורר או בוחר קבצים לבצע עליהם חיזוי. פונקציה זו קוראת לפונקציות `indexFile` ו `displayFile` עם כל אחד מהקבצים.

מקבלת:

`e : Event` – אירוע גרירת/בחירת הקבצים.

מחזירה:

כלום

אופן פעולה:

הפעולה מקבלת את רשימת הקבצים שנגררו מתוך אירוע הגרירה, במידה וזהו אירוע הגרירה הראשון מאז שהמשתמש ביצע חיזוי (על פי המשתנה הגלובאלי `predicted`), הפעולה מוחקת את כל התמונות מאיזור הצגת התמונות (לאחר חיזוי גרירה תמחק את הצגת כל התמונות הקיימות כבר, שכן המודל כבר חזה אותן). ללא קשר לערכו של `predicted`, הפעולה תקרא לפעולות `indexFile` ו `displayFile` עם כל אחד מהתמונות שנגררו.

5. `displayFile(file)`

תפקיד:

פעולה פנימית אשר נקראת על ידי `handleFiles`, מוסיפה את התמונה שהיא מקבלת אל אזור הצגת התמונות שנבחרו.

מקבלת:

`file : File` – הקובץ אותו הפעולה תוסיף לאזור בו המשתמש יכול לראות את הקבצים (במקרה זה, תמונות) שנבחרו.

מחזירה:

כלום

אופן פעולה:

פעולה זו יוצרת אלמנט מסוג `figure` ומוסיפה אותו אל מערך `containers` אשר יותר מאוחר ישמש על מנת להוסיף לכל התמונות את טקסט החיזוי המתאים להן. לאחר מכן, הפעולה מוסיפה אל אלמנט ה `figure` את התמונה, ויוצרת אלמנט מסוג `figcaption` אשר לאחר חיזוי יהיה הטקסט מתחת לתמונה המציג את חיזוי המודל.

6. indexFile(file)

תפקיד:

פעולה פנימית אשר נקראת על ידי `handleFiles`, ממירה את הקובץ שהיא מקבלת לתמונה ומוסיפה אותה לרשימת תמונות שלאחר מכן יישלחו אל הbackend על מנת לקבל עבורן חיזוי.

מקבלת:

File : file – הקובץ שיומר לתמונה ויתווסף לרשימה.

מחזירה:

כלום

אופן פעולה:

מוסיפה את התמונה אל מערך התמונות אשר ישמש לשליחת כולן אל ה backend לצורך קבלת חיזוי.

7. requestPredictions()

תפקיד:

פעולה אשר קוראת ל backend עם מערך התמונות שנבחרו על ידי המשתמש, מקבלת בחזרה את החיזויים של המודל לתמונות וקוראת ל `showPredictions` כדי להציג אותם.

מקבלת:

כלום

מחזירה:

כלום

אופן פעולה:

שולחת את כל התמונות הנמצאות במערך התמונות אל הפעולה `handle_and_predict` הנמצאת ב backend ומספקת callback אשר ירוץ לאחר סיום הפעולה. אופן פעולת ה callback:

קודם כל ה callback בודק אם ישנן שגיאות בחיזוי התמונה. אם כן, הוא מחליף את טקסט החיזוי ל Failed ומדפיס לקונסולה את השגיאה שהתרחשה. לאחר מכן (ללא קשר לאם התרחשה שגיאה), הוא מוסיף את הטקסט אל מערך הפלט של הפונקציה ובודק אם זוהי הייתה התמונה האחרונה במערך התמונות. אם כן, הוא קורא לפונקציה `showPredictions`.

8. showPredictions(arr)

תפקיד:

פעולה פנימית אשר נקראת על ידי requestPredictions לאחר שלכל התמונות יש חיזוי. פעולה זו מוסיפה את תגית החיזוי מתחת לכל תמונה בתוך ממשק המשתמש.

מקבלת:

arr : string[] – מערך החיזויים של המודל. כל חיזוי מתאים לתמונה ב index שלו.

מחזירה:

כלום

אופן פעולה:

פונקציה זו עוברת על המערך שהיא קיבלה ומוסיפה את הטקסט במערך כטקסט של ה figcaption המתאים במערך ה containers הגלובאלי (ראה אופן פעולת displayFile), ובכך מציג את תוצאות החיזוי מתחת לתמונות המתאימות בממשק המשתמש.

9. handleError(txt)

תפקיד:

פעולה אשר נקראת על ידי ה backend במידה ונתפסה שגיאה בשלב כלשהו. פעולה זו תעלה את לשונית השגיאות ותציג את השגיאה שהתקבלה.

מקבלת:

txt : string – מחרוזת המתארת את השגיאה שהתקבלה.

מחזירה:

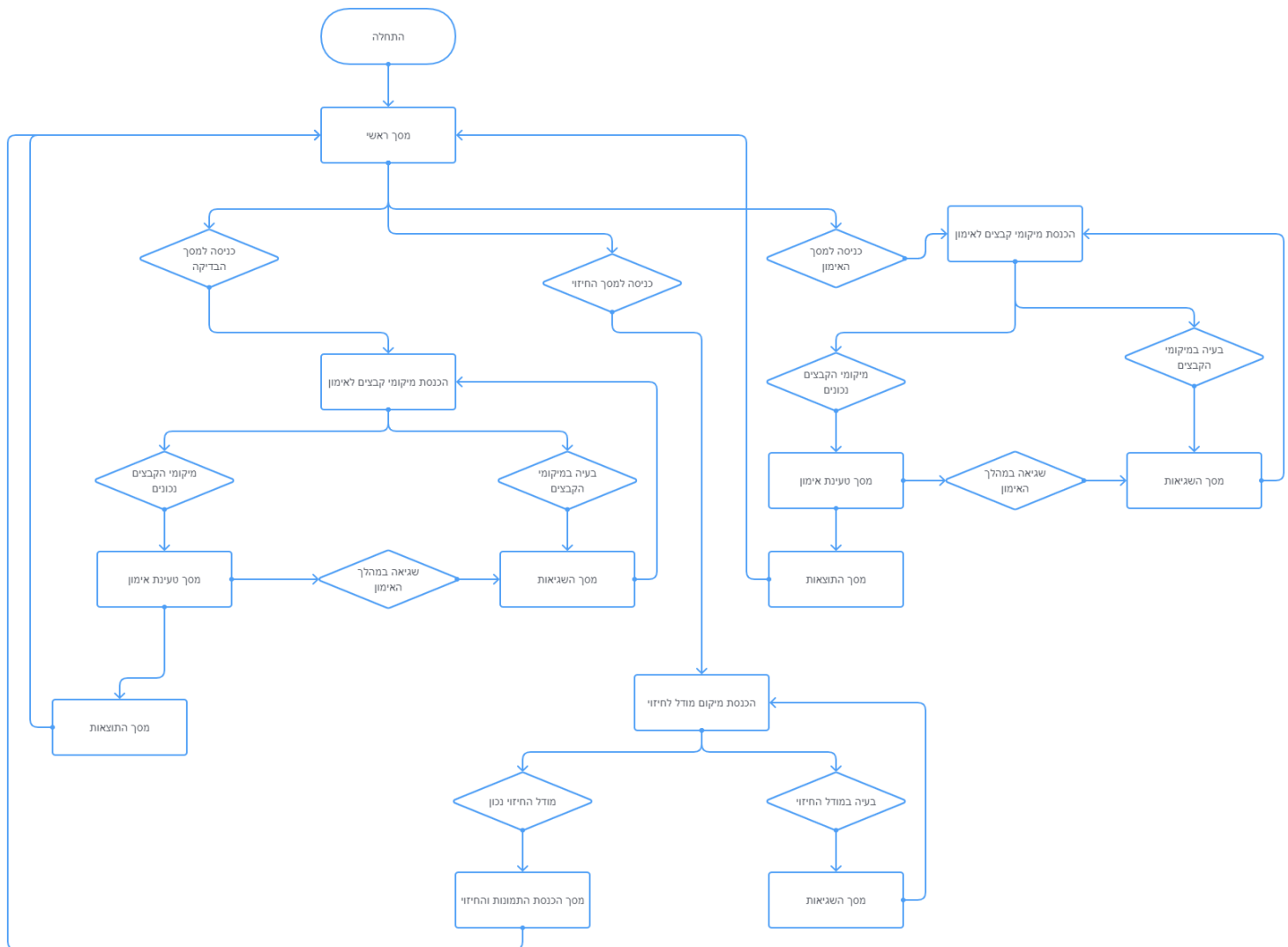
כלום

אופן פעולה:

פעולה זו מעבירה אל מסך השגיאות, ומציגה את השגיאה שהתקבלה בו.

מדריך למשתמש

תרשים מסכים:



רשימת מסכים ותפקידים:

המסך הראשי:



תפקיד:

תפקידו של מסך זה היינו להוות נקודת מעבר בין הפונקציות השונות של ממשק המשתמש, כגון חיזוי, אימון או בדיקה.

אלמנטי תצוגה:

1. כפתור train – כפתור שלחיצה עליו מובילה לממשק המשתמש המאפשר לאמן מודל.
2. כפתור test – כפתור שלחיצה עליו מובילה לממשק המשתמש המאפשר לבחון מודל.
3. כפתור predict – כפתור שלחיצה עליו מובילה לממשק המשתמש המאפשר לבצע חיזויים בעזרת מודל.

מסך קלט נתוני train test:



SETUP

Dataset Path

Model Save Path

☐ Search for pre-made data

Exit Next

תפקיד:

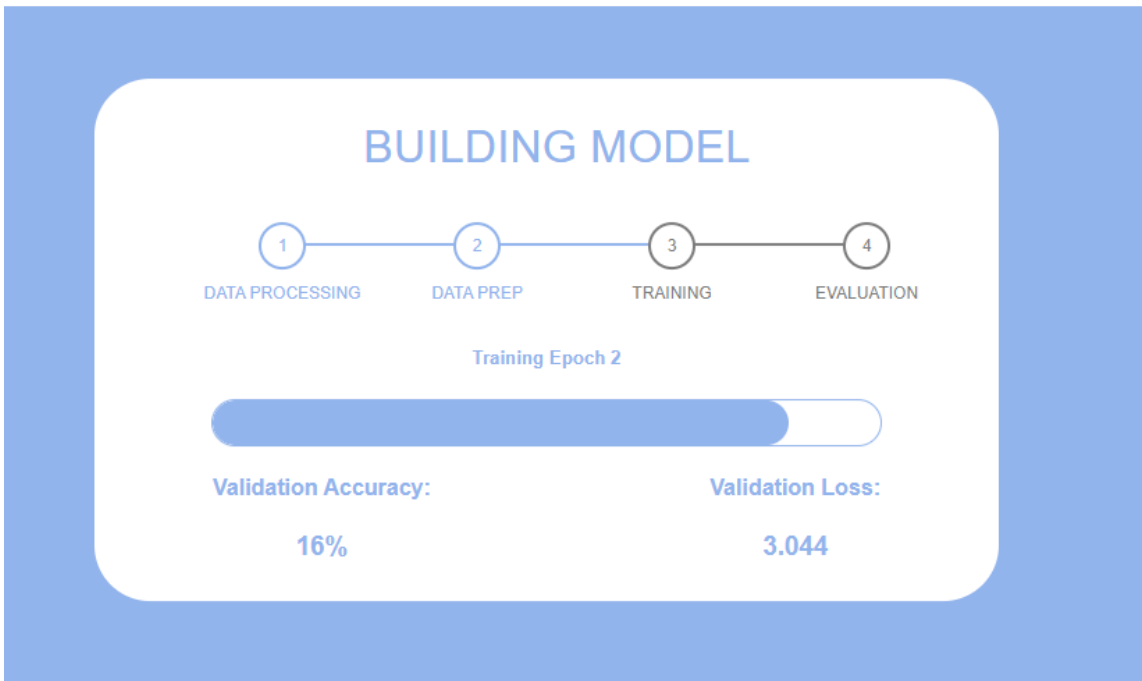
מסך זה משמש כקלט המידע הדרוש על מנת לבצע train\test (בהתאם לכפתור עליו המשתמש לחץ).

אלמנטי תצוגה:

1. dataset path input - שדה קלט המקבל את מיקום בסיס הנתונים, שאיתו יתבצע אימון\בחינת המודל.
2. model path input – שדה קלט המקבל את המיקום בו ישמר המודל (במקרה של אימון) או את המיקום של מודל שמור (במקרה של בחינה).
3. Search for pre-made data toggle – אם המשתמש מסמן את תיבה זו, התוכנה תחפש אם קיים קובץ מוכן מראש של נתונים בתוך ה dataset path (קובץ כזה נוצר בכל פעם שמתבצע ניתוח מלא של הנתונים).

4. כפתור next – מתקדם לשלב הבא, בו מופיע מסך הטעינה של האימון\בחינה. במידה וישנם שגיאות במיקומי המודל והנתונים
5. כפתור exit – מחזיר את המשתמש אל המסך הראשי

מסך טעינת train test:



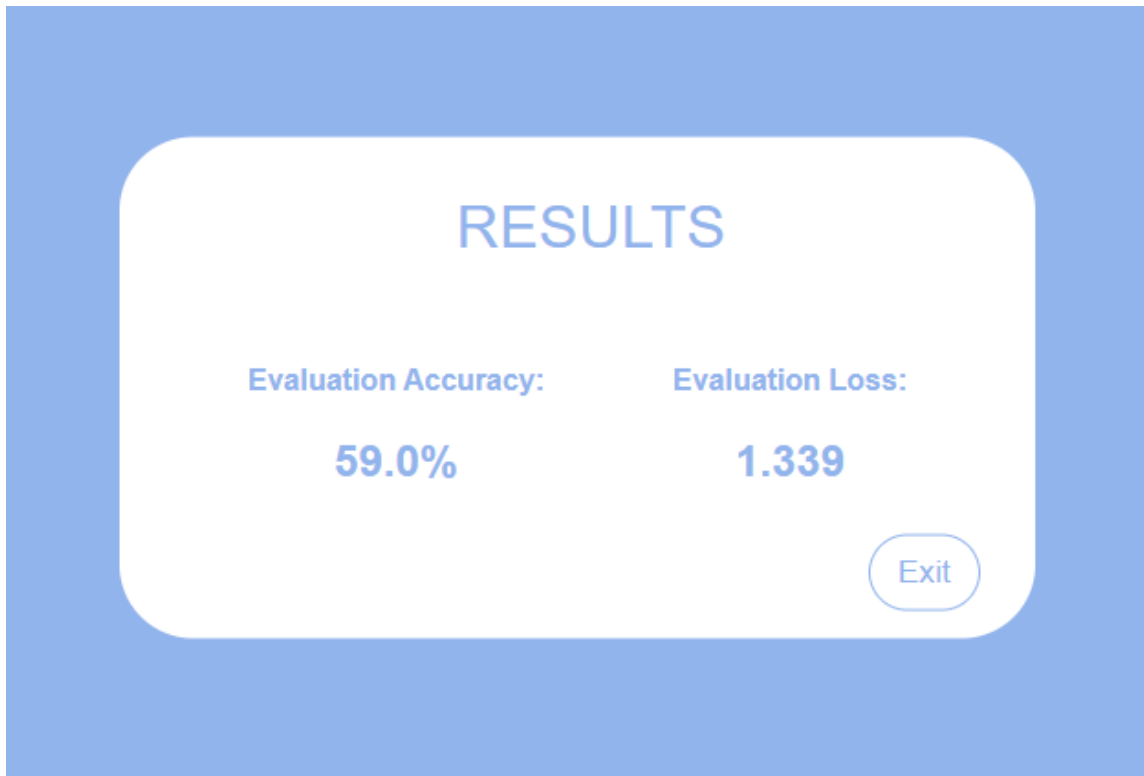
תפקיד:

מסך המציג את התקדמות הטעינה של אימון\בחינת המודל.

אלמנטי תצוגה:

1. תצוגת שלבים – מתחת לכותרת מופיע תצוגת שלבים המציגה את התהליך הכולל שנשאר.
2. סרגל טעינה\אייקון טעינה – מתחת לתצוגת השלבים מופיע סרגל טעינה או אייקון טעינה (בהתאם לפעולה שמתבצעת באותו הרגע). מעל הסרגל\אייקון מופיע טקסט המתאר את הפעולה המתבצעת באותו הרגע.
3. נתוני אימון – במידה ומתבצע אימון מודל, יופיעו נתוני האימון העדכניים מתחת לסרגל הטעינה.

מסך תוצאות:



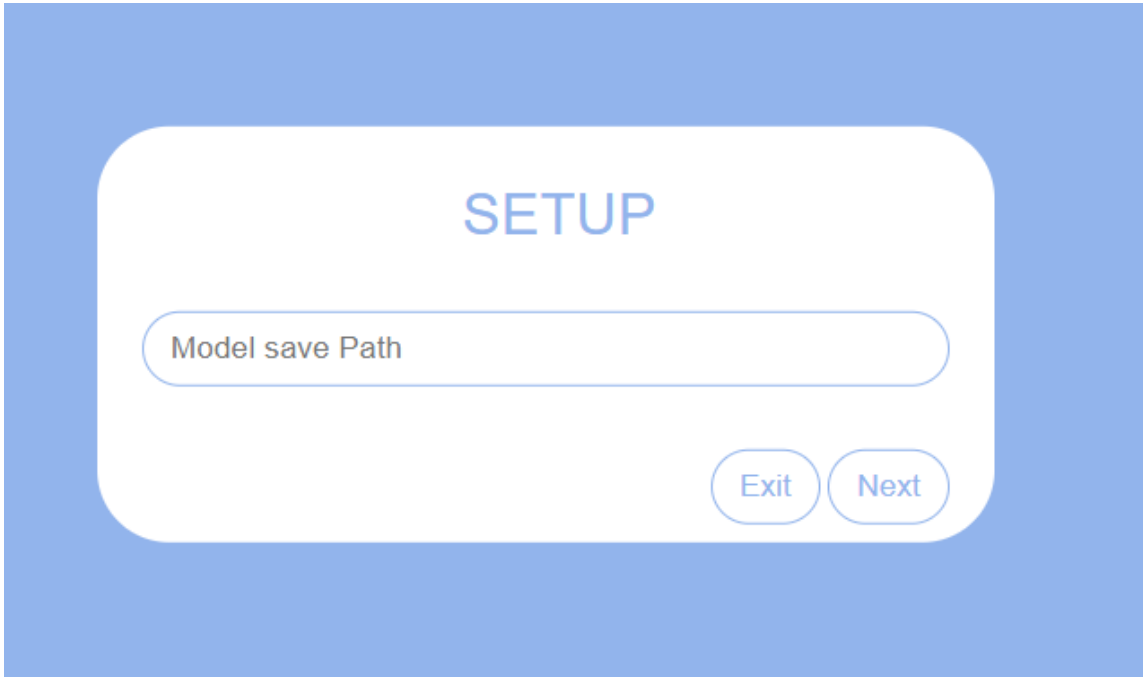
תפקיד:

מסך המציג את תוצאות הבחינה\ האימון (בסוף אימון התוכנה מבצעת בחינה על מנת לבדוק את תפקודו).

אלמנטי תצוגה:

1. תוצאות הבחינה – במרכז המסך, מופיעים נתוני ה loss וה accuracy המהווים את תוצאות הבחינה.
2. כפתור exit – כפתור המחזיר את המשתמש למסך הראשי.

מסך קלט נתונים predict:



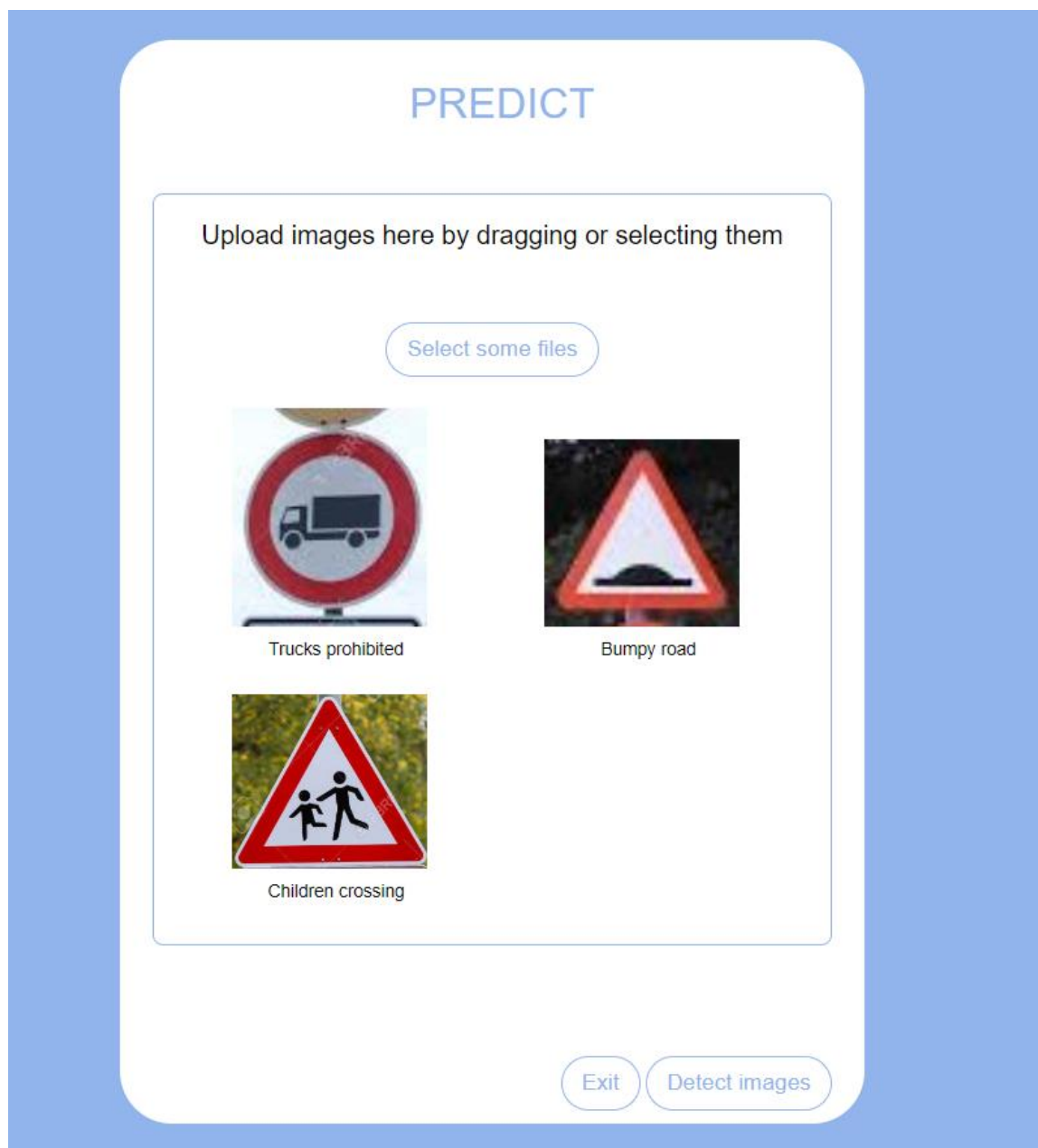
תפקיד:

מסך זה משמש כקלט המידע הדרוש על מנת לבצע חיזוי.

אלמנטי תצוגה:

1. model path input – שדה קלט המקבל את המיקום בו נמצא המודל שבעזרתו המשתמש מעוניין לבצע חיזוי.
2. כפתור next – מתקדם לשלב הבא, בו מופיע מסך המאפשר למשתמש לבצע חיזוי (במידה וישנן בעיות עם הקלט, כפתור זה יוביל את המשתמש אל מסך שגיאות).
3. כפתור exit – מחזיר את המשתמש אל המסך הראשי.

מסך החיזוי:



תפקיד:

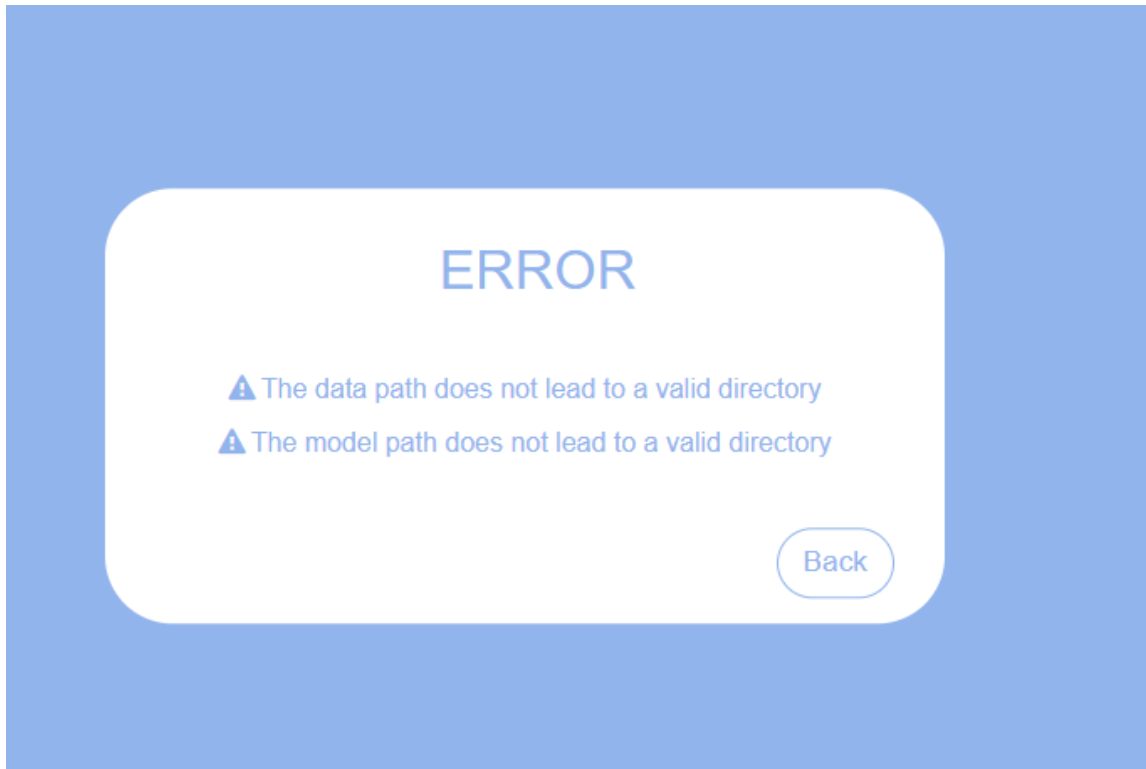
מסך זה מאפשר למשתמש להעלות תמונות, אותם המודל יחזה. תוצאות החיזוי מופיעות מתחת לתמונות.

אלמנטי תצוגה:

1. קופסת הגרירה – כל תמונה שתיגרר אל תוך קופסת הגרירה תופיע בתוכה ויבצע עליה חיזוי כאשר הכפתור Detect images יילחץ.

2. כפתור Select some files – מציג חלון המאפשר למשתמש לבחור קבצים ידנית (ללא גרירה). גם תמונות אלו יופיעו בתוך קופסת הגרירה ויתבצע עליהן חיזוי כאשר Detect images יילחץ.
3. כפתור Detect images – כאשר המשתמש לוחץ על כפתור זה, המודל מבצע חיזוי על כל התמונות הנמצאות בקופסת הגרירה. לאחר מכן, תוצאות החיזוי יופיעו מתחת לכל תמונה.
4. כפתור exit – מחזיר את המשתמש אל המסך הראשי.

מסך שגיאות:



תפקיד:

מציג למשתמש את השגיאות שהתרחשו ובכך מאפשר חווית משתמש המשכית גם במקרה של התנהגות בלתי צפיה.

אלמנטי תצוגה:

1. רשימת שגיאות – במרכז המסך מופיע רשימה של השגיאות שעלו במהלך הריצה.
2. כפתור back – מחזיר את המשתמש אל מסך הכנסת הקלט, שכן במרבית הפעמים השגיאה נובעת מכך שהקלט של המשתמש לא היה נכון.

דרישות והוראות התקנה:

1. יש להוריד את קוד הפרוייקט מ [github](https://github.com/AlonX2/Traffic-Sign-detector) :<https://github.com/AlonX2/Traffic-Sign-detector>
יש לשים לב להוריד את ענף ה master בלבד.
2. במידה ויש צורך לאמן\לבחון מודל, יש להוריד את הנתונים מכאן:
https://benchmark.ini.rub.de/gtsrb_dataset.html
3. יש לוודא כי מותקן על המחשב python3.
4. יש להריץ את הפקודה `pip install -r requirements.txt`
5. במידה ומותקנת anaconda על המחשב, יש להשתמש בה על מנת להתקין ספריות שלא באות איתה (שכן שימוש מקביל ב pip ואנקונדה עלול ליצור בעיות).
יש להריץ את הקובץ `app.py` על מנת להפעיל את ממשק המשתמש.

רפלקציה

העבודה על פרויקט זה הייתה מאתגרת אך מהנה. במהלך העבודה רכשתי ידע וכלים רבים חדשים, כולל ידע רב בעולם למידת מכונה והמתמטיקה העומדת מאחוריו. רכשתי יכולות פיתוח רבות נוספות, כגון ניהול פרויקט גדול והחשיבות של תיעוד הקוד. אחד מהכלים המרכזיים שרכשתי מפרוייקט זה היה הבנה על איך לנהל פרויקטים שמיועדים לרוץ על מערכות שונות ומחשבים שונים מזה שלי – דבר זה גרם לי ללמוד על הכלים השונים בתחום, בין docker לבין `python virtual environments` | `anaconda`. אספקט זה של הפרוייקט היה גם אחד האתגרים המרכזיים בו – שכן התחלתי את הפרוייקט בשימוש ב `anaconda` ומאוחר יותר עברתי לשימוש מלא ב `pip`. מעבר זה היווה אתגר מרכזי בפרוייקט, שכן פעמים רבות כלים שונים לא עבדו ודרשו כמויות מאסיביות של דיבוג ושינויים. בנוסף, במשך כל תקופה זו, לא הייתה לי אפשרות להריץ את הקוד שכן כל הספריות לא עבדו ללא שיטה כלשהי לניהולם (לדוגמא התקנה ב `pip` מהווה שיטה מסוימת לניהול ספריות). קושי מרכזי נוסף היה ניהול העברת המידע בין `python` אל `javascript`. בשל העובדה שהן עובדות בצורה שונה מאוד (`python` מופרשת במעבד בעוד `javascript` בסביבה ווירטואלית בדפדפן), חוויתי הרבה בעיות בתקשורת ביניהם. בעיות רבות בתקשורת נבעו מבעיות שחוויתי עם שפת התכנות `javascript`.

המסקנה המרכזית שלי מהפרוייקט היא שאין סיבה לפחד מלהיכנס לתחומים שונים מאלו שאני כבר רגיל אליהם, וניתן ללמוד כל דבר, מורכב ככל שיהיה, מכלים חינוכיים ברחבי האינטרנט.

אם הייתי מתחיל את הפרוייקט היום הייתי כותב אותו בצורה יותר מסודרת ומאורגנת מההתחלה, תוך שימוש מאורגן ב `pip` ואולי אף ב `docker` על מנת לנהל את ה `dependencies` ושימוש מרחיב יותר ב `git` על מנת לנהל את הקוד עצמו.

אומנם בעיות אלו האריכו את זמן העבודה של הפרוייקט והיו מתסכלות מאוד בזמן שניסיתי לפתור אותן, במבט לאחור אני מבין כי בעיות אלו לימדו אותי רבות לא רק על הדרך הנכונה לנהל אספקטים מסוימים של פרויקטים גדולים, אלא גם הבנה עמוקה יותר של חיפוש מידע ופתרונות באינטרנט, אספקט בלתי נפרד מעולם התכנות ככלל. בנוסף, אני מבין כי בעיות כאלו

ואחרות הינן חלק בלתי נפרד מתכנות – ולכן ניסיון בפתירת בעיות שצצות לאורך הדרך היינו כלי חשוב ביותר בידי של מתכנת טוב.

כסיכום, אני מרגיש שהדבר המרכזי שפרויקט זה נתן לי בסופו של דבר הוא כלים. אם אלו כלים לניהול פרויקטים גדולים, ידע בתחום מרתק של תכנות ומתמטיקה או ניסיון מורכב יותר בפתרון שגיאות ובעיות שונות. אני סמוך ובטוח שכלים אלו יבואו לידי ביטוי בפרויקטים אחרים שאני אעשה בעתיד, ואף בתחומים אחרים בחיי ככלל.

ביבליוגרפיה

מקורות מידע:

<https://www.youtube.com/watch?v=Ilg3gGewQ5U>, 2017, גרנט סאנדרסון.

https://en.wikipedia.org/wiki/Traffic-sign_recognition

Xue ,2019 ,<https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022/pdf>
Ying

Laith ,2021 ,<https://link.springer.com/content/pdf/10.1186/s40537-021-00444-8.pdf>
Alzubaidi1, , Jinglan Zhang , Amjad J. Humaidi , Ayad Al-Dujaili , Ye Duan , Omran Al-Shamma , J. Santamaría , Mohammed A. Fadhel , Muthana Al-Amidie and Laith Farhan

מקורות קוד:

<https://stackoverflow.com>

https://www.tensorflow.org/api_docs

<https://www.geeksforgeeks.org/create-html-user-interface-using-eel-in-python>

<https://www.analyticsvidhya.com/blog/2021/08/beginners-guide-to-convolutional-neural-network-with-implementation-in-python>

<https://developer.mozilla.org/en-US>

https://keras.io/guides/writing_your_own_callbacks

