מכללה: סמינר בנות אלישבע.

שם סטודנט: רבקה קבריטי

ת"ז סטודנט: 328905559

שם המנחה: גב' שולמית ברלין.

תאריך הגשה: 26/06/2022

תוכן העניינים:

1. הצעת פרויקט:

סמל מוסד: 189084

שם מכללה: בנות אלישבע

שם הסטודנט: רבקה קבריטי

ת.ז הסטודנט: 328905559

שם הפרויקט: בדיקת מזוזות.

תיאור הפרויקט:

אפליקציה לבדיקת כשרות מזוזות.

התוכנה תקבל סריקה של מזוזה ותבדוק אם קיימות אותיות יתירות או חסרות, וכן תזהה אותיות מחוברות.

התוכנה תחזיר קובץ בו מפורטים מיקומי השגיאות ומהי השגיאה.

הגדרת הבעיה האלגוריתמית:

- חיתוך האותיות.
- זיהוי אותיות.
- בדיקת נכונות מזוזה (כל האותיות קיימות בסדר הנכון).

רקע תיאורטי בתחום הפרויקט:

מְזוּזָה היא תשמיש קדושה הנקבע בצד הכניסה לבית מגורים ובחדרי הבית היהודי. המזוזה הוא קלף עליו נכתבות בכתב סת"ם (אשורית) ובדיו, פרשיות 'שמע ישראל' ו־'והיה אם שמוע' מחומש דברים. על פי רוב, יריעת הקלף נתונה בתוך בית המזוזה, התקן קשיח שנועד לכיסוי הקלף ולשמירה עליו.

כתיבת המזוזה צריכה להיות "כסדרן" כלומר כתיבה על פי הסדר מתחילת פרשת 'שמע ישראל 'ועד סוף פרשת 'והיה אם שמוע'. אף שאין הגדרה מחייבת לגבי מספר שורות המזוזה ואורכם, מנהג הסופרים הוא לכתוב מזוזה ב־22 שורות, כאשר המילים הפותחות את השורות הן מילים קבועות.

לפני קביעת המזוזה בפתח הבית, על בעל הבית למסור אותה למגיה מומחה, שיבדוק שהיא נכתבה בצורה כשרה. לאחר שנבדקה ונמצאה כשרה, היא מוחזקת ככשרה כל זמן שלא אירע לה דבר העלול לפגוע בכשרותה.

המגיה בודק האם כל המילים קיימות כסדרן, שהאותיות תהיינה מופרדות, שצורת האותיות תהיה לפי כל ההלכות והכללים הנדרשים, שיהיו התגים השייכים לכל אות.

האפליקציה שאכתוב תקבל תמונה סרוקה של מזוזה ותבדוק שהאותיות נכונות , לפי הסדר, אינן מחוברות ועוד. לבסוף תחזיר דף ובו מפורטים מיקומי השגיאות ומה המגיה צריך לתקן.

תהליכים עיקריים בפרויקט :

- קליטת נתונים- סריקה של מזוזה.
- זיהוי רווחים.
- זיהוי וסיווג האותיות.
- השוואת סדר האותיות לסדר האותיות של מזוזה כשרה הקיימת במאגר.
- בניית דף בו מפורטים מיקום השגיאה והסבר השגיאה.

תיאור הטכנולוגיה:

צד שרת: שפת תכנות בצד שרת: Python

צד לקוח: שפת תכנות בשפת לקוח: React

מסד נתונים:

תיקיות לאחסון תמונות של כל אות אשורית.

פרוטוקולי תקשורת:

Http

לוחות זמנים:

נובמבר- תכנון.

דצמבר - לימוד של החומר לצורך אלגוריתם.

ינואר- פבואר - כתיבת אלגוריתם.

מרץ- UI קשר בין השרת ללקוח.

אפריל- ספר פרויקט.

מאי- ניסוי ובדיקות.

יוני- הצגת הפרויקט.

חתימת הסטודנט: ר. קבריטי .

2. מבוא:

2.1 רקע:

בשנים האחרונות התפתחה שיטת הבינה המלאכותית בעולם התכנות, ונכנסתי במסגרת הלימודים לקורס למידת מכונה, המקנה לי ידע והבנה עמוקה של הנושא.

כשחשבתי על רעיון לפרויקט היה ברור לי שאני רוצה להתמחות בנושא זה, וכן שהפרויקט יכלול בתוכו אלגוריתם מורכב וגם טכנולוגיה חדשה ומרתקת.

רציתי להתמחות בעיבוד תמונות במסגרת הבינה מלאכותית, ואבי שהוא מגיה מוסמך של ספרי תורה, תפילין ומזוזות, נתן לי את הרעיון לפתח אפליקציה שתשמש ככלי עזר לסופר סת"ם (כך מכונה הלבלר הכותב בין היתר מזוזות) ולמגיה, על מנת שיוכלו להגיה את המזוזות בצורה ממוחשבת.

המשתמש יסרוק או יצלם את המזוזה, ולאחר עיבוד התמונה ושליחה לרשת נוירונים שמפענחת את תוכן המזוזה, יוחזר ללקוח טופס ובו מפורטים תוצאות ההגהה (מיקומי השגיאות וכדומה), לאחר מכן הוא יוכל לשמור את הטופס כקובץ ולהוריד למחשב.

כל מה שנשאר לו זה לתקן ע"פ הממצאים מיקומים אלו. (כאשר זה אפשרי כמובן)

2.2 תהליך המחקר:

בפרויקט שני חלקים מרכזיים:

1. זיהוי תוכן התמונה וחלוקה לתווים ואותיות:

החלטתי לממש זאת באופן כזה: בתחילה לעבור על הפיקסלים שבתמונה ולחתוך ממנה כל שורה ושורה, ע"י זיהוי רווחים בין השורות.
לאחר מכן לחתוך כל שורה לאותיות באופן שכל אות תשמר כתמונה נפרדת.
ואז שליחת כל אות לרשת נוירונים (שאימנתי בעצמי) המזהה אותה.

2. זיהוי האות:

ניתן היה לפתור את הבעיה ברובה, ע"י שימוש בספריות של עיבוד תמונה הקיימות בפיתון שבאופן אוטומטי מעבדות תמונה ומוציאות מתוכה את הכתוב – אפשרות זו נפסלה משום שרציתי לדעת לעומק את עניין הבינה המלאכותית וכיצד עובד אימון רשת נוירונים ולא להשתמש בספריות מוכנות שמסתירות את הקוד העומד מאחוריו.

כמו כן , היה ניתן לכתוב את הקוד במגוון שפות כמו java או c# אך אני בחרתי את פיתון מכיוון שהיא השפה הכי מפותחת בנושא וגם כי היא נכנסה מאד לשוק העבודה ורציתי להתמקצע בה ולהכיר אותה יותר לעומק.

2.3 סקירת ספרות:

- ✓ https://github.com/
- ✓ לימוד Python https://reshetech.co.il/python-tutorials/all-the-tutorials
- ✓ לימוד למידת מכונה https://reshetech.co.il/machine-learning-tutorials/all-the-tutorials
- ✓ ספריית OpenCV https://opencv.org/
- ✓ Learn Deep Learning & OpenCV https://pyimagesearch.com/
- ✓ ספריית PIL https://pillow.readthedocs.io/en/stable/#
- ✓ ספריית Keras https://keras.io/
- ✓ ספריית TensorFlow https://www.tensorflow.org/
- ✓ ספריית NumPy https://numpy.org/doc/stable/index.html
- ✓ ויקיפדיה
- ✓ Geeks For Geeks https://www.geeksforgeeks.org/
- ✓ StackOverFlow

3. מטרות:

- הגהת המזוזה בדרך חכמה, מדויקת ומהירה .
- לחסוך זמן לסופר סת"ם/ מגיה.
- לשפר את חווית המשתמש בהגהת המזוזה.
- התנסות בפיתון והכרת ספריות משמעותיות בצורה יסודית יותר.
- למידת הנושא artificial intelligence (בינה מלאכותית) לעומק ויצירת מודל של רשת נוירונים.
- לשפר ולייעל את היכולת של הלימוד העצמי בתחומים שונים.

יעדים:

- תכנון המערכת בצורה הטובה ביותר וכתיבת אלגוריתם יעיל ומסודר .
- איסוף תמונות רבות של אותיות על מנת לאמן רשת נוירונים אמינה ביותר.
- אימון מספר רשתות ובחירת הרשת הנותנת את התוצאות המדויקות ביותר .
- בניית אתר נעים לעין, שנותן למשתמש תוצאות מהירות.
- הדגשת מיקומי השגיאות כדי שיהיה ברור למשתמש מה עליו לתקן באופן עצמאי.

4. אתגרים:

בפיתוח המערכת ניצבו בפני אתגרים רבים:

- בינה מלאכותית ואימון רשת: ראשית כל היה עלי ללמוד ולהבין את כל הנושא של למידת מכונה, לדעת לאמן את המחשב בקליטת מושגים שמוח אנושי מבין אך אינו יודע לתת לכך כללים ברורים .

  האתגר הגדול הוא אימון רשת הנוירונים ולהצליח להגיע לרשת נוירונים טובה שמביאה את התוצאה הנכונה ביותר. לשם כך היה עלי לבנות את שכבות הרשת בצורה יעילה.
- איסוף תמונות : על מנת לאמן את המודל, היה עלי לאסוף כמות גדולה של תמונות של אותיות, על מנת לגרום לרשת להיות מדויקת כמה שיותר. לצורך כך צילמתי הרבה מזוזות וחתכתי מתוכם את האותיות ולאחר מכן הפעלתי על מאגר התמונות פונקציה נוספת המעבה את התמונות , עד שלבסוף היו בידי יותר מ400 תמונות מכל אות!
- חילוץ האותיות: מאחר ואותיות אלו (כמו אותיות עבריות בכתב יד) עולות אחת על השנייה, יורדות לכיוון השורה הבאה, ועולות כלפי מעלה, באופן שאין כללים ברורים המספקים את מאפייני מיקום התחלת האות וסיומה, היה לי אתגר גדול להגיע לנוסחה כללית למציאת את מיקום האותיות על מנת לחתוך אותם ולשלוח לרשת לזיהוי. לדוגמא:

ל' שעולה על האותיות הבאות אחריה וכלפי השורה שמעליה, ן' וק' שיורדות לכיוון השורה הבאה. ועוד.

- איכות התמונה: היה אתגר גדול להגיע לאיכות תמונה טובה אך שלא תשנה מן המציאות. מהסיבה שאיכות הצילום משפיעה מאוד על התוצאות. במיוחד במקרה שלנו – לדוגמה אם יש חיבור כלשהו בין שני תווים סמוכים זו בעיה חמורה הפוסלת את המזוזה, ואם הצילום לא חד מספיק נוצרים חיבורים בין תווים סמוכים, או מקרים הפוכים בו מזהים הפרדה בין שני חלקי תו שמחוברים בחבור דק שלפעמים מיטשטש בתמונה שאינה חדה.

צילום מזוזה בצורה טובה ומקצועית אינה פשוטה, בשל הצורך לשמור אותה ללא קיפולים וחלקה (יש לצורך זה סורקים מיוחדים). כמובן ניתן להשתמש גם במצלמות רגילות רק שהאיכות לא תהיה גבוהה.

- כתיבת קוד: כתיבת הקוד בפיתון דרש הרבה לימוד עצמי, מכיוון ששפת האם שלי היא C#, אותה אני מכירה לעומק. וכמובן שהיה לי יותר נוח לכתוב בה את הפרויקט, אך דווקא מסיבה זו העדפתי להתמקצע בצורה יסודית בפיתון כדי להתמקצע בשפה נוספת שפופולארית היום בשוק.

5. מדדי הצלחה למערכת:

המודל שאימנתי מזהה ב- 99.4%

אם המערכת תצליח לזהות לפחות 93% מהשגיאות, זה יהווה הצלחה של המערכת. זיהוי של פחות מכך יורה על חוסר הצלחה.

6. תיאור מצב קיים:

קיימות מספר תוכנות מסחריות שמספקות פתרון של הגהה ממוחשבת, אך כולן נסחרות והקוד איננו פתוח, ורצוני הוא לבנות תוכנה חינמית שכל אדם יוכל להיעזר בה בקלות.

7. רקע תיאורטי :

זיהוי האותיות :

ניתן לזהות אותיות בתמונה באמצעות אלגוריתם של עיבוד תמונה. באופן זה הופכים קודם את התמונה לשחור -לבן, ממירים אותה למטריצת פיקסלים ואז ניתן לבדוק אם קיימות צורות של אותיות במטריצה ע"פ הפיקסלים הצבועים בשחור.

הבעיה בשיטה הזאת היא החוסר דיוק , אם האות קצת מטושטשת או חתוכה בקצוות העיבוד לא יהיה אמין ולכן לא נוכל להסתמך על זה, וכן בכתב יד יכול להיות שינויים בסגנון

הכתב, אע"פ שיש כללים לכתב המזוזה. לכן בחרתי בבינה מלאכותית שמניבה תוצאות נכונות גם במקרים אלו .

בחירת רשת הנוירונים המתאימה:

ניתן להשתמש בסוגים שונים של רשתות נוירונים , חלקם אפילו מובנים בשפה (למשל), אך אני בחרתי ברשת שאימנתי בעצמי מהסיבה שהיא מביאה לדיוק מירבי וכן כי רציתי ללמוד לפתח רשת נוירונים.

8. ניתוח חלופות מערכתי:

ישנן אפשרויות אחרות להגהת מזוזה.

הראשונה, הגהה ידנית של המגיה (כמו שנעשה עד היום) שעובר אות אות ובודק האם כל המילים קיימות כסדרן, שהאותיות תהיינה מופרדות, שצורת האותיות תהיה לפי כל ההלכות והכללים הנדרשים.

רעיון נוסף שהמגיה יקליד ידנית ,לפי הסדר, את האותיות שקיימות במזוזה הנבדקת. האפליקציה תעשה את ההגהה ותחזיר את התוצאות. אמנם רעיון זה יותר נח למשתמש מהשוואה ידנית, אך ההקלדה לוקחת זמן, ויכולות להיות טעויות אדם. וכן לא נפתרת בצורה זו הבעיה של אותיות מחוברות.

9. תיאור החלופה הנבחרת:

הדרך הנבחרת היא כאמור, שהמשתמש יעלה לאפליקציה תמונה סרוקה של מזוזה והאפליקציה תבדוק שהאותיות נכונות , לפי הסדר, אינן מחוברות ועוד. לבסוף תחזיר דף ובו מפורטים מיקומי השגיאות ומה המגיה צריך לתקן.

בחרתי בדרך הזאת משום שהיא אתגרה אותי ולימדה אותי נושאים חדשים בתחום הלמידה החישובית ובלמידת פיתון באופן יסודי, וכן כי היא בסופו של דבר היא הכי נוחה למשתמש.

10. אפיון המערכת:

10.1 דרישות המערכת:

- ✓ כתיבה בסטנדרטים גבוהים ומקצועיים, באופן מסודר תוך הקפדה על תיעוד בכל שלב.
- ✓ מעל 99% הצלחה של זיהוי האותיות.
- ✓ בניית ממשק נעים למשתמש.
- ✓ כתיבת אלגוריתם יעיל ככל האפשר.

10.2 מודול המערכת -תהליכים מרכזיים :

- חלק ראשון- הכנת הנתונים (התמונות) לרשת הנוירונים:
  - איסוף סריקות של מזוזות.
  - חיתוך לאותיות.
  - הפיכתם לצבעי שחור- לבן.
  - שינוי גודל התמונה על מנת שיוכלו לעבד אותה ולזהות את הכתוב בה בצורה הטובה ביותר.
  - עיבוי מאגר התמונות ע"י פונקציה של עיבוי תמונה.
  - חלוקת התמונות לשלוש תיקיות validate, train, test: באופן רנדומלי .כאשר בכל תיקיה ישנם 28 תיקיות שכל אחת מהם מכילה תמונות של אות אחת.
- חלק שני – אימון רשת נוירונים:
  - בניית הרשת ואימונה על התמונות שנאספו ועובדו.
  - בדיקת אמינות הרשת.
- חלק שלישי – תהליך הפרויקט ופעולותיו בצד השרת:
  - קבלת התמונה מה-client.
  - הפיכת התמונה לשחור לבן.
  - הסרת מסגרת לבנה (אם יש)
  - חיתוך התמונות ל22 שורות.
  - חיתוך המילים לאותיות.
  - שליחת כל אות לזיהוי ברשת נוירונים.
  - הכנסת הנתונים שזוהו לאובייקט.
  - ההגהה (השוואה אם זו האות הרצויה)
  - בניית רשימת מיקומי השגיאות.

- שליחת מסמך המכיל את כל הממצאים ופרטי השגיאות ללקוח.

- <u>חלק רביעי – תהליך הפרויקט ופעולותיו בצד הלקוח:</u>
  - כניסת המשתמש לאתר והעלאת סריקה של המזוזה המיועדת לבדיקה, מהמחשב.
  - שליחת הסריקה לשרת.
  - קבלת מסמך המכיל את הפרטים שזוהו מתוך ההגהה.
  - אפשרות של הורדת המסמך למחשב .

10.3 אפיון פונקציונלי:

חלק א': עיבוד מקדים לתמונות:

image_quality - במחלקה זו מספר פונקציות שאחראיות על איכות התמונה.

- change_color_to_black_and_white- פונקציה המשנה את התמונה לשחור- לבן.
- resolution_and_resize_with_white_background- פונקציה המשנה את הרזולוציה וגודל התמונה ל28X28. (ושמה ברקע מסגרת לבנה לפי הצורך)

הכנת תשתית ללמידת מכונה:

Preprocessing- במחלקה זו שתי פונקציות שתפקידם להכין את מסד הנתונים:

- image_augmantion – פונקציה המעבה את מאגר התמונות פי 7
- TrainTestValidateSplit - מחלקת את מאגר התמונות לשלושה תיקיות כאשר כל אחת מהם מחולקת ל-28 קטגוריות.

חלק ב': בניית רשת נוירונים:

Letters_Recognition במחלקה זו שלוש פונקציות עיקריות:

- load_model - טעינת מאגר התמונות
- baseline_model - בניית וקימפול המודל.
- fit_model – אימון המודל.

חלק ג': תהליך הפרויקט ופעולותיו בצד הServer :

Main מחלקה ראשית שמנהלת את כל תהליך ההגהה.

- uploadFile קבלת המזוזות מצד הלקוח.
- MainFunc פונקציה הקוראת לכל הפונקציות של עיבוד התמונה הכולל חיתוך השורות ואז האותיות מתוך המזוזה ושליחתם לרשת נוירונים. לאחר מכן, שליחתם להגהה ולבסוף החזרת אובייקט ללקוח המכילה את כל השגיאות שנמצאו.

חלק ד': תהליך הפרויקט ופעולותיו העיקריות בצד הClient:

- upload_file– העלאת סריקת מזוזה להגהה ע"י המשתמש, ושליחתה לשרת.
- DownloadFile – הורדת טופס התוצאות בפורמט PDF.

10.4 ביצועים עיקריים:

התוכנה מקבלת סריקה של מזוזה חותכת ממנה לאותיות, שולחת אותן לזיהוי, ועושה את ההגהה. לאחר מכן מחזירה טופס ובו מפורטים הממצאים.

10.5 אילוצים:

- זיהוי מדויק של האותיות והגהת המזוזה בזמן תגובה סביר ככל הניתן.
- זיהוי אותיות אשוריות בלבד.

11. תיאור הארכיטקטורה :

תכנון הפרויקט נעשה מן הכלל אל הפרט, בתחילה תכנון הפרויקט נעשה באופן כללי ובכל שלב ירדתי פנימה, יותר לפרטים ולעומק עד לביצוע מלא ומקיף של הפרויקט.

11.1 תיאור הרכיבים:

11.2 פרוטוקולי תקשורת: http

11.3 תיאור טכנולוגית השרת והלקוח:

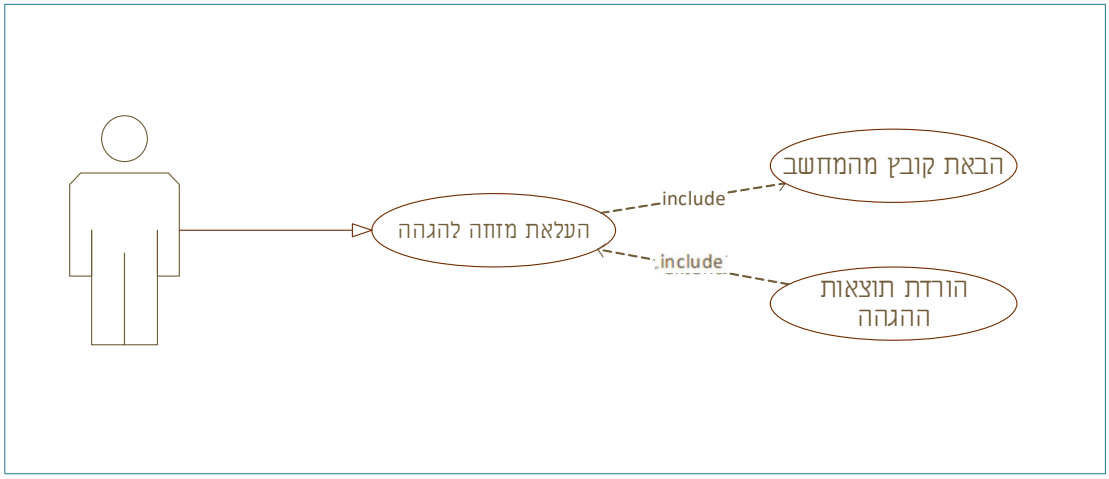
הפרויקט פותח בצד שרת - Server :

בשפת Python עם תוכנת PyCharm גרסה 3.7 , עם טכנולוגיית - Flask ספרייה ב Python – המשמשת סביבה לפיתוח Web וממשק ה API- בשפת Python .

בצד לקוח – Client :

פותח בשפות Html, CSS, JS בטכנולוגיית React, עם תוכנת Visual Studio Code .

12. ניתוח ותרשים UML של המערכת המוצעת :



12.1 רשימת Use cases

- ❖ UC1: העלאת מזוזה להגהה

12.2 תיאור ה-UC העיקריים של המערכת:

:UC1

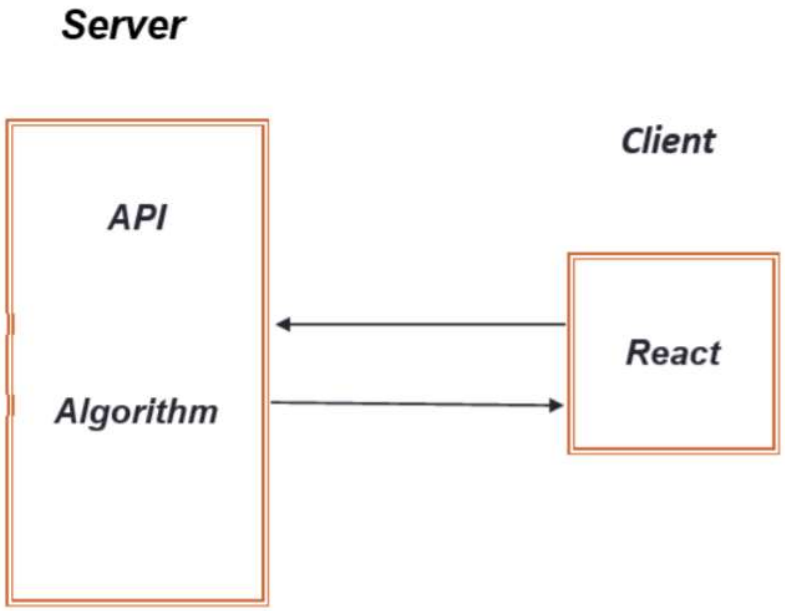
- **Name-** Uploading a Mezuza for proofreading.
- **Identifier -** UC Uploads a Mezuza for proofreading.
- **Description –** המשתמש יעלה סריקה של מזוזה שברצונו לבדוק (מהמחשב). ולאחר מכן באמצעות לחיצה על כפתור שליחה, המזוזה תשלח לשרת על מנת להגיה אותה. כאן ימומש האלגוריתם הראשי של המערכת. לאחר ההגהה המשתמש יוכל להוריד את תוצאות ההגהה.
- **Actors –** סופר סת"ם או אדם המעוניין לבדוק את כשרות המזוזה שברשותו.
- **Frequency –** בדרך כלל בסיום כתיבת המזוזה ולפני רכישת מזוזה. אך יכול להיות בכל עת שירצה.
- **Pre-condition** – הסריקה ברורה, בלי לכלוכים, ופרוסה במדויק.
- **Post-condition** – המזוזה תשלח להגהה, ויוחזר טופס ובו מפורטות תוצאות ההגהה.
- **Included use case** - *הבאת קובץ מהמחשב

  *הורדת קובץ התוצאות בפורמט PDF.
- **Assumptions –** המזוזה כתובה בכתב אשורי כהלכה.

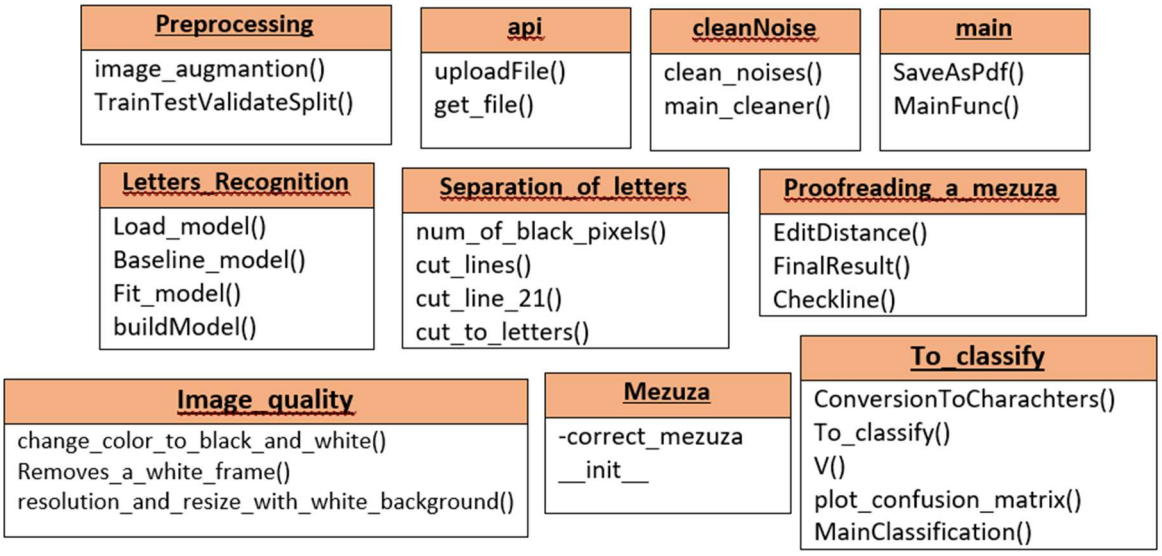
12.3 מבני נתונים המשמשים את הפרויקט:

1. מטריצות – אחסון תמונות שונות במטריצה, על מנת שאוכל להפעיל עליהם כל מיני פעולות.
2. רשימות – נעזרתי בחיתוך שורות ואותיות, על מנת לשמור שורות החשודות כרווחים.
3. מילונים – על מנת לשמור את אותיות המזוזה הכשרה. כאשר המפתח הוא מספר השורה, והערך הוא רשימה של האותיות בשורה זו.
וכן לשמירת האותיות שנמצאו במזוזה הנבדקת, באותו אופן (מפתח=מס' שורה, ערך=רשימת האותיות)

12.4 עץ מודולים:



12.5 תרשים מחלקות – class diagram:



12.6 תיאור המחלקות:

מחלקות בצד server :

מחלקת Mezuza :

תפקיד המחלקה: המחלקה שומרת את התכונות של המזוזה הכשרה שאותה נשווה עם הנבדקות.

תכונות: correct_mezuza – זהו מילון כאשר המפתח הוא מספר השורה, והערך הוא רשימה של האותיות בשורה זו.

פונקציות המחלקה:

- Init – פעולה הבונה מופע מסוג המחלקה ומאתחלת בנתונים את המילון correct_mezuza.

מחלקת Image_quality :

תפקיד המחלקה: המחלקה אחראית להכין את הסריקה של המזוזה למצב שבו יהיו מתאימות לשלוח אותם לחיתוך אותיות. וכן להכנת מאגר התמונות למודל.

קלט: נתיב לתמונה.

פלט: נתיב לתמונה לאחר העיבוד.

פונקציות המחלקה:

- change_color_to_black_and_white- פונקציה המשנה את התמונה לשחור- לבן
- Removes_a_white_frame – הסרת השוליים הלבנות שמסביב המזוזה עצמה.
- resolution_and_resize_with_white_background - פונקציה המשנה את הרזולוציה וגודל התמונה ל28x28. (ושמה ברקע מסגרת לבנה לפי הצורך)

מחלקת Separation_of_letters:

תפקיד המחלקה: המחלקה אחראית לחתוך את המזוזה לאותיות.

קלט: נתיב לתמונה.

פלט: נתיב לתיקיה שבו מאוחסנות כל האותיות שנחתכו.

פונקציות המחלקה:

- num_of_black_pixels - פונקציה המחזירה את מספר הפיקסלים השחורים והלבנים בתמונה.
- cut_lines – פונקציה שמקבלת תמונה של מזוזה וחותכת אותה לשורות.
- cut_line_21 – פונקציה שחותכת את שורה 21 לשתי שורות.
- cut_to_letters – פונקציה שחותכת את השורות לאותיות.

מחלקת cleanNoise:

תפקיד המחלקה: המחלקה מנקה כמה שיותר את השורות שנחתכו מרעשים.

פונקציות המחלקה:

- clean_noises - פונקציה המנקה תמונה שמקבלת מרעשים.
- main_cleaner – הפונקציה שולחת לפונקציה הנ"ל את כל השורות שנחתכו.

מחלקת Preprocessing:

תפקיד המחלקה: המחלקה מכינה את התמונות שנאספו למסד הנתונים, לרשת נוירונים.

פונקציות המחלקה:

- image_augmantion – פונקציה המעבה את מאגר התמונות פי 7
- TrainTestValidateSplit - מחלקת את מאגר התמונות לשלושה תיקיות כאשר כל אחת מהם מחולקת ל-28 קטגוריות.

מחלקת Letters_Recognition:

תפקיד המחלקה: המחלקה העיקרית בפרויקט, ותפקידה פיתוח ואימון המודל, עד לקבלת רשת נוירונים חזקה ומדויקת.

פונקציות המחלקה:

- Load_model - הפונקציה טוענת את התמונות מתוך שלושת התיקיות: train, test וvalidate.
- Baseline_model - שלב פיתוח וקימפול המודל , הפונקציה מחזירה מודל המורכב משכבות שהתווספו בשלב זה. (יפורט בהרחבה באלגוריתמים מרכזיים.)
- Fit_model – הפונקציה מאמנת את המודל על מאגר התמונות שהוטענו. הפונקציה מחזירה רשת נוירונים בעלת משקולות שנוצרו במהלך האימון.
- buildModel - -הפונקציה מפעילה את שלושת הפונקציות הנ"ל.

מחלקת To_classify:

תפקיד המחלקה: להציג את נכונות המודל. ולשלוח את התמונות של האותיות לסיווג במודל.

פונקציות המחלקה:

- ConversionToCharachters –הפונקציה מקבלת מספר שפוענח והתקבל מרשת הנוירונים וממירה אותו לאות המתאימה.
- To_classify - הפונקציה שולחת למודל את התמונה שרוצים לסווג ומחזירה את האות שהתקבלה מהמודל.
- V – הפונקציה מחזירה מטריצת בלבול פשוטה עבור תיקיית הvalidate ואת המדדים השונים על נכונות המודל.
- plot_confusion_matrix – הפונקציה מציגה מטריצת בלבול מעוצבת וברורה יותר.
- MainClassification – הפונקציה שולחת כל תמונה של אות שנחתכה מהמזוזה ל-To_classify ומחזירה מילון שהמפתח הוא מספר השורה, והערך הוא רשימה של אותיות שזוהו משורה זו.

מחלקת Proofreading_a_mezuza:

תפקיד המחלקה: הגהת המזוזה.

קלט: מילון של האותיות שזוהו.

פלט: קובץ txt שבו מפורטים מיקומי השגיאות שנמצאו.

פונקציות המחלקה:

- EditDistance – פונקציה המקבלת רשימה של האותיות הנבדקות, ורשימה של האותיות הנכונות. הבונה מטריצה ע"פ אלגוריתם של תכנות דינאמי ומחזירה את מספר הטעויות המינימלי ואת המטריצה שעל פיה הגיעה לתוצאה זו.
- FinalResult – פונקציה המחשבת ע"פ המטריצה הנ"ל את המיקומים של השגיאות ושמה '-' במיקומים אלו ברשימה של השורה הנבדקת או של השורה הנכונה, ומחזירה שורות אלו.
- Checkline – פונקציה שיוצרת קובץ txt ובו היא כותבת על פי הנ"ל את מיקומי השגיאות, והסבר השגיאות, שיש בשורה מסוימת שקיבלה.

<u>מחלקת api:</u>

תפקיד המחלקה: מממשת את התקשורת בין הclient לserver. מקבלת תמונה של מזוזה מהclient ומחזירה קובץ תוצאות מהserver.

פונקציות המחלקה:

- uploadFile – פונקציה המקבלת תמונה של מזוזה מה-client ושולחת אותה לעיבוד והגהה.
- get_file – פונקציה המחזירה את קובץ התוצאות ל-client.

<u>מחלקת main:</u>

תפקיד המחלקה: מחלקה ראשית שמנהלת את כל תהליך ההגהה.

פונקציות המחלקה:

- SaveAsPdf – פונקציה הממירה קובץ txt לפורמט PDF.
- MainFunc - פונקציה ראשית המקבלת מזוזה וקוראת לכל הפונקציות של עיבוד התמונה הכולל חיתוך השורות ואז האותיות מתוך המזוזה, ושליחתם לרשת נוירונים. לאחר מכן, שליחתם להגהה ולבסוף החזרת אובייקט ללקוח המכילה את כל השגיאות שנמצאו.

קומפוננטות בצד Client:

- **App:**

הקומפוננטה הראשית שמציגה את כותרת האתר ומציגה את הקומפוננטה UploadComponent.

- **UploadComponent :**

כפתור להעלאת תמונה.

פונקציות:

- onDrop – מציגה את התמונה שנבחרה על המסך.
- upload_file – הפונקציה שולחת את התמונה לשרת.

- **Download:**

קומפוננטה להורדת קובץ השגיאות.

פונקציות:

- downloadFile – מורידה את הקובץ שהוחזר מהשרת.

- **CircularIndeterminate:**

קומפוננטה להצגת המתנה לעיבוד המזוזה וקבלת טופס התשובות.

13. תיאור התוכנה:

13.1 סביבת פיתוח:

חומרה: מעבד i5, 8GB RAM.

עמדת פיתוח: מחשב intel.

מערכת הפעלה: windows 10.

הפרויקט פותח בגרסת Python 3.7

13.2 שפות תכנות:

צד שרת: פותח בשפת Python עם טכנולוגיית Flask.

באמצעות תוכנת PyCharm, גרסה 3.7.

צד לקוח: פותח בשפת React.

באמצעות תוכנת VisualStudioCode.

חיבור לאינטרנט: נדרש.

דפדפנים:Explorer, chrome.

14. אלגוריתמים מרכזיים:

האלגוריתם המרכזי מורכב מכמה שלבים עיקריים:

1. **הכנת התמונות למודל:**

על מנת לאמן את הרשת בצורה הטובה והמדויקת ביותר היה צורך במאגר גדול של תמונות עליהם הרשת תתאמן.

ראשית, היה עלי לאסוף תמונות רבות של כל אות, ע"י חיתוך מזוזות רבות לאותיות.

לאחר מכן, הגדלתי את מסד הנתונים באמצעות פונקציה לעיבוי תמונות .

ספריית Keras מציעה את הפונקציה ImageDataGenerator שיכולה להפיק כמה תמונות שונות על בסיס תמונה מקורית שמסופקת לה.

הספריות הנדרשות הן:

```
from keras.preprocessing.image import
ImageDataGenerator,array_to_img, img_to_array, load_img
import os
```

אופן הפעולה:

הפונקציה מקבלת תמונה ויוצרת על בסיס תמונה זו שש תמונות חדשות שונות.

השוני בין התמונות בא לידי ביטוי בצורה ויזואלית לדוגמא: הזווית בה כתובה האות, עובי האות, בהירות וחדות התמונה וכדומה.

```
def image_augmantion(ori_path,k,dest_path):
    datagen = ImageDataGenerator(
        rotation_range=20,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=False,
        brightness_range=(0.5, 1.5))

    # Loading a sample image
    img = load_img(ori_path)
    # Converting the input sample image to an array
    x = img_to_array(img)
    # Reshaping the input image
```

```
    x = x.reshape((1,) + x.shape)

    # Generating and saving 6 augmented samples
    # using the above defined parameters.
    i = 55
    for batch in datagen.flow(x,
batch_size=1,save_to_dir=dest_path,save_prefix='0'+str(k)+str(i),
save_format='png'):
        i += 1
        if i > 60:
            break
```

הפונקציה שמממשת את הפונקציה הנ"ל על מאגר התמונות המבוקש :

הלולאה עוברת על כל תיקיה ובתוך כל תיקיה על כל תמונה ומעבה אותה ויוצרת ממנה עוד שש תמונות.

```
def changesInImages(path_to_db):
    for letter in os.listdir(path_to_db):
        i = 0
        for img in os.listdir(fr'{path_to_db}/{letter}'):

image_augmantion(fr'{path_to_db}/{letter}/{img}',i,fr'{path_to
_db}/{letter}')
            i += 1
```

לאחר עיבוי המאגר יש לשנות את הגודל של כל התמונות לגודל שווה וקטן יותר.

הפונקציה שמשנה גודל של תמונה לגודל של r_sizeXr_size (ברירת מחדל: 28X28 פיקסלים:

```
def resolution_and_resize_with_white_background(path_ori,
path_dest, r_size=28):
    size = r_size
    img = Image.open(path_ori)
    # resize and keep the aspect ratio
    img.thumbnail((size, size), Image.ANTIALIAS)
    # add the white background
    img_w, img_h = img.size
```

```
    background = Image.new('RGB', (size, size), (255, 255, 255))
    bg_w, bg_h = background.size
    offset = ((bg_w - img_w) // 2, (bg_h - img_h) // 2)
    background.paste(img, offset)
    background.save(path_dest)
```

הפונקציה שאחראית לזמן את הפונקציה הנ"ל עבור התמונות שבנתיב המבוקש. במקרה זה נשלח נתיב למאגר התמונות:

```
def main_change_size(path):
    for l in os.listdir(path):
        for im in os.listdir(fr'{path}/{l}'):

resolution_and_resize_with_white_background(fr'{path}/{l}/{im}',

fr'{path}/{l}/{im}')
```

לאחר שמאגר התמונות מוכן לשימוש עבור אימון המודל לזיהוי אותיות, עלינו לחלק את התמונות לשלוש תיקיות train, test ו- validate.

<u>החלוקה מתבצעת באופן הבא:</u>

20% מכלל התמונות במאגר יוכנסו באופן רנדומלי (בעזרת ספרית Scikit–Learn) לתיקיית ה-test, והשאר (80%) יוכנסו לתיקיית ה-train.

לאחר מכן, מתוך התמונות שבתיקיית train ניקח 10% לתיקיית ה-validate.

<u>תפקידה של כל תיקיה:</u>

train- עם התמונות שבתיקייה זו, יתאמן המודל וילמד איך נראית כל אות.

test- עם תמונות אלו יבחן המודל את נכונותו (כחלק משלב האימון והלמידה).

validate- תמונות אלו יוצגו לרשת בסיום האימון, כלומר אלו תמונות שהרשת לא תכיר כלל במהלך האימון ואיתם בחנתי את נכונות המודל.

הקוד שמבצע את הנ"ל:

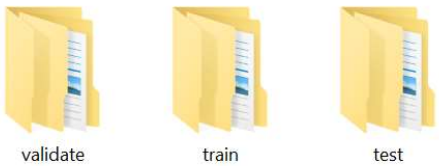
הספריות הנדרשות:

```
from sklearn.model_selection import train_test_split
import shutil
```

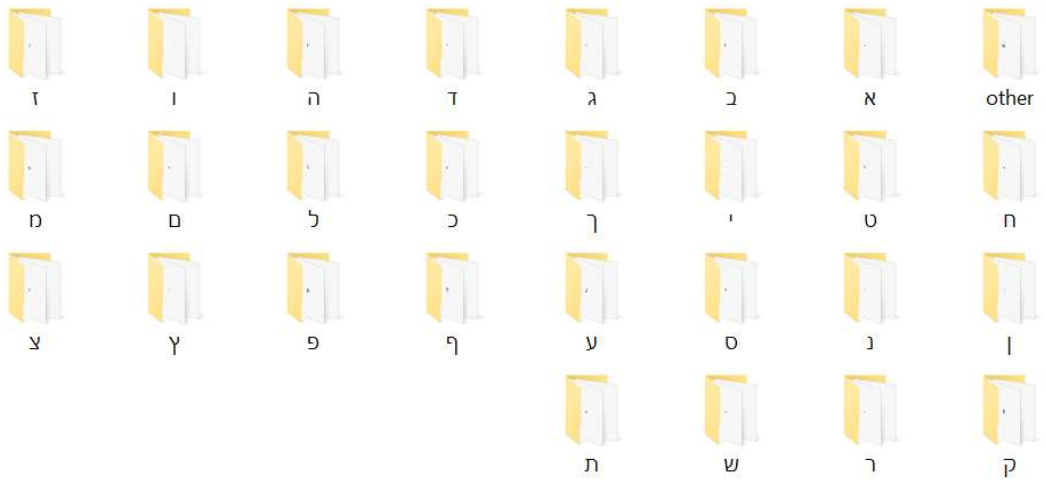
הפונקציה:

```
def TrainTestValidateSplit(path):
    X = []
    for file in os.listdir(path):
        current_path = os.path.join(path, file)
        for img in os.listdir(current_path):
            X.append(img)
        train_valid = path + '_train_valid'
        train = path + '_train'
        test = path + '_test'
        valid = path + '_valid'
        train_valid, test_name = train_test_split(X,
test_size=0.2, random_state=42)
        train, valid = train_test_split(train_valid,
test_size=0.1, random_state=42)
        for item in train:
            ori_path = current_path + '/' + item
            dest = fr'train_test_validate/train/{file}'
            shutil.copy(ori_path, dest)
        for item1 in test_name:
            ori_path = current_path + '/' + item1
            dest = fr'train_test_validate/test/{file}'
            shutil.copy(ori_path, dest)
        for item2 in valid:
            ori_path = current_path + '/' + item2
            dest = fr'train_test_validate/validate/{file}'
            shutil.copy(ori_path, dest)
        train_valid.clear()
        train.clear()
        test_name.clear()
        valid.clear()
        X.clear()
```

התוצאה:



כל אחת מן התיקיות מכילה 28 תיקיות של אותיות (בother ישנן אותיות צמודות):



2. פיתוח המודל:

ראשית ייבאתי את הספריות הבאות:

```
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,
Dropout,Flatten,Conv2D, MaxPooling2D
```

```
class ModelTraining:
```

טעינת התמונות עליהם תתאמן הרשת:

```
@staticmethod
def load_model():
    # create generator
    datagen = ImageDataGenerator()
    # load and iterate training dataset
    train_it = datagen.flow_from_directory(
    r'db_letters\train_test_validate\train',
    class_mode='categorical', color_mode="grayscale",
    batch_size=63693, target_size=(28, 28))

    # load and iterate test dataset
    test_it = datagen.flow_from_directory(
    r'db_letters\train_test_validate\test',
    class_mode='categorical',
```

```
    batch_size=4206,
    target_size=(28, 28), color_mode="grayscale")

    # confirm the iterator works
    X_train, y_train = train_it.next()
    X_test, y_test = test_it.next()
    # normalize inputs from 0-255 to 0-1
    X_train = X_train / 255
    X_test = X_test / 255

    num_classes = y_test.shape[1]
    return X_train, y_train, X_test, y_test, num_classes
```

בניית המודל:

```
@staticmethod
def baseline_model(num_classes):
    # Create a layer type model
    model = Sequential()
    # convolution layer: 32 filters, 3*3, use with relu for
activation function
    model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1),
activation='relu'))
    # [add this layer to decrease the loss]
    model.add(Conv2D(52, (3, 3), input_shape=(28, 28, 1),
activation='relu'))
    model.add(Conv2D(64, (3, 3), input_shape=(28, 28, 1),
activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(90, (3, 3), input_shape=(28, 28, 1),
activation='relu'))
    model.add(Conv2D(95, (3, 3), input_shape=(28, 28, 1),
activation='relu'))
    model.add(Conv2D(98, (3, 3), input_shape=(28, 28, 1),
activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # ignore from 20% from the noironim
    model.add(Dropout(0.2))
```

```
    # we have 3 chanels. flat them to one long vector
    model.add(Flatten())
    # another noirinim layer (with activation function)
    model.add(Dense(128, activation='relu'))
    # output
    model.add(Dense(28, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
    model.summary()
    return model
```

הסבר מורחב על המודל:

יצירת מודל מסוג שכבות:

```
model = Sequential()
```

השכבה החבויה היא שכבת הקונבולוציה (convolution):

```
model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1),
activation='relu'))
```

שכבה זאת מריצה 32 פילטרים בגודל 3*3 פיקסלים על כל תמונה. הפילטרים הם הנוירונים של השכבה הכוללים פונקציית אקטיבציה. כל פילטר עובר על כל התמונה ע"י שמזיזים אותו בכל פעם בפיקסל בודד.

התוצאה היא סידרה של ייצוגים שונים של התמונה – כל פילטר יצר ייצוג שונה ולפיכך במקרה שלנו יצרנו 32 ייצוגים שונים של תמונה.

בתהליך האימון, הרשת מחשבת פונקציית loss אשר נותנת לה משוב עד כמה הפרדיקציה שלה טובה וכמה היא צריכה לשנות את המשקלים על מנת לשפר את הביצועים.

ככל שהערך של פונקציית ההפסד (loss) נמוכה יותר, כך המודל "צודק" יותר.

הוספתי עוד שכבות על מנת להוריד את ערך הloss.

שכבה נוספת של המודל היא שכבת הדגימה:

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

לאחר כל 3 קונבולוציות הפעלתי את שכבת הmax-pooling.

תפקידה של שכבה זאת הוא לצמצם את המטריצה (ריבוע של הפיקסלים) שממדיו 2*2 פיקסלים שנמסר לה מהשכבה הקודמת לרבע, כלומר המטריצה תצומצם לפיקסל בודד.

זה נעשה על ידי דגימת הערך הגבוה מבין ארבעת הפיקסלים שמרכיבים את המטריצה שהתקבלה ואותו היא מעבירה לשכבה הבאה.

השכבה הבאה היא שכבת הdropout:

```
model.add(Dropout(0.2))
```

השכבה זאת היא לא שכבה קריטית במודל עצמו אלא היא מונעת מהמודל ללמוד יותר מדי טוב את הדוגמאות שנתנו לה לאימון, מה שיגרום לירידת היכולת לזהות אותיות שלא נכללו בסט הדוגמאות ששימשו לאימון.

הסיווג בפעול של התמונות נעשה ע"י רשת נוירונית מסוג Dense.

על מנת להעביר את המידע מבלוק הקונבולוציה (שכולל את השכבות המפורטות לעיל) לרשת נוירונית, משתמשים בפונקציית flatten שתפקידה לשטח את המידע לווקטור (מטריצה ששיש בה שורה אחת ואורכה כמספר הסיווגים) משום ששכבת ה- Dense חייבת לעבוד על ווקטור ולא על מטריצה.

```
model.add(Flatten())
```

שכבת ה-Dense:

```
model.add(Dense(128,activation='relu'))
```

מכילה 128 פיקסלים בלבד, ובה כל נוירון מחובר לכל אחד מנוירוני הקלט.

שכבת ה-dense האחרונה מסווגת בפועל לאחת מ-28 קטגוריות (האותיות) באמצעות פונקציית האקטיבציה softmax.

```
model.add(Dense(28, activation='softmax'))
```

קימפול המודל:

```
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

בקומפילציה של המודל השתמשתי בפונקציית Adam **שמגדילה** את השינויים שעושים לפרמטרים של המודל – **כשרחוקים** מהמינימום שהוא פתרון המתמטי של המודל, ו**מקטינה** את השינויים ככל **שמתקרבים** למינימום. גודל השינויים הוא חשוב מפני ששינויים גדולים מדי יכולים לגרום לפספוס המינימום בעוד שינויים קטנים מדי יגרמו לאימון לרוץ יותר מדי זמן.

אימון המודל:

```
@staticmethod
def fit_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=25, batch_size=64, verbose=2)
    model.save("model_try3.h5")
```

לאחר אימון המודל אחוזי הדיוק בסיווג היו 99.4%!!!!

```
Epoch 20/25
122/122 - 22s - loss: 0.0142 - accuracy: 0.9959 - val_loss: 0.0398 - val_accuracy: 0.9884 - 22s/epoch - 181ms/step
Epoch 21/25
122/122 - 21s - loss: 0.0216 - accuracy: 0.9922 - val_loss: 0.0238 - val_accuracy: 0.9926 - 21s/epoch - 172ms/step
Epoch 22/25
122/122 - 21s - loss: 0.0124 - accuracy: 0.9964 - val_loss: 0.0137 - val_accuracy: 0.9958 - 21s/epoch - 172ms/step
Epoch 23/25
122/122 - 21s - loss: 0.0218 - accuracy: 0.9924 - val_loss: 0.0361 - val_accuracy: 0.9912 - 21s/epoch - 174ms/step
Epoch 24/25
122/122 - 21s - loss: 0.0097 - accuracy: 0.9968 - val_loss: 0.0117 - val_accuracy: 0.9968 - 21s/epoch - 172ms/step
Epoch 25/25
122/122 - 21s - loss: 0.0059 - accuracy: 0.9982 - val_loss: 0.0189 - val_accuracy: 0.9940 - 21s/epoch - 172ms/step
```

כדי לבדוק את אמינות המודל ואחוזי הדיוק בעבור כל קטגוריה הוספתי פונקציות שיחשבו את המדדים לכל קטגוריה ויבנו מטריצת בלבול.

הפונקציה הבאה עוברת על אותיות תיקיית הvalidation (שהרשת לא אומנה עליה) ושולחת אותן לסיווג במודל שהורץ. ולפי התוצאות מחשבת מדדי f1 score, accuracy ו- precision לכל קטגוריה.

```
# בדיקת נכונות המודל, על פי תיקיית הולידטיון
def V():
    list_validation = []
    list_true = []
    # בגלל שבולידטיון יש 32 תמונות מכל אות נבנה רשימה של 32 פעמים מכל האותיות הנכונות
    for directory in os.listdir('db_letters/train_test_validate/validate'):
        for i in range(32):
            list_true.append(directory)
    # שליחת כל תמונה מהולידטיון לסיווג ושמירת תוצאת הסיווג שלה
    for directory in os.listdir('db_letters/train_test_validate/validate'):
        for image in os.listdir(fr'db_letters/train_test_validate/validate/{directory}'):
            list_validation.append(To_classify(fr'db_letters/train_test_validate/validate/{directory}/{image}'))

    y_true = list_true
    y_pred = list_validation
    cm = confusion_matrix(y_true, y_pred)  # שמירת מטריצת בלבול פשוטה בלי עיצובים

    class_names = ['other', 'א', 'ב', 'ג', 'ד', 'ה', 'ו', 'ז', 'ח', 'ט', 'י', 'ך', 'כ', 'ל', 'ם', 'מ', 'ן', 'נ', 'ס',
                   'ע', 'ף', 'פ',
                   'ץ', 'צ', 'ק', 'ר', 'ש', 'ת']
    # הצגת כל מדדי הדיוק
    print(classification_report(y_true, y_pred,
                                target_names=class_names))
```

.וזו התוצאה שהתקבלה:

	precision	recall	f1-score	support
other	0.97	1.00	0.98	32
32	1.00	1.00	1.00	א
32	1.00	1.00	1.00	ב
32	0.97	0.97	0.97	ג
32	0.93	1.00	0.86	ד
32	1.00	1.00	1.00	ה
32	1.00	1.00	1.00	ו
32	0.92	0.84	1.00	ז
32	1.00	1.00	1.00	ח
32	1.00	1.00	1.00	ט
32	1.00	1.00	1.00	י
32	0.98	1.00	0.97	ך
32	0.98	0.97	1.00	כ
32	0.98	1.00	0.97	ל
32	0.98	0.97	1.00	ם
32	1.00	1.00	1.00	מ
32	0.97	1.00	0.94	ן
32	0.97	0.97	0.97	נ
32	0.97	0.94	1.00	ס
32	0.98	1.00	0.97	ע
32	0.98	0.97	1.00	ף
32	0.97	1.00	0.94	פ
32	0.94	1.00	0.89	ץ
32	1.00	1.00	1.00	צ
32	0.98	0.97	1.00	ק
32	0.93	0.88	1.00	ר
32	1.00	1.00	1.00	ש
32	0.97	0.94	1.00	ת
accuracy			0.98	896
macro avg	0.98	0.98	0.98	896
weighted avg	0.98	0.98	0.98	896

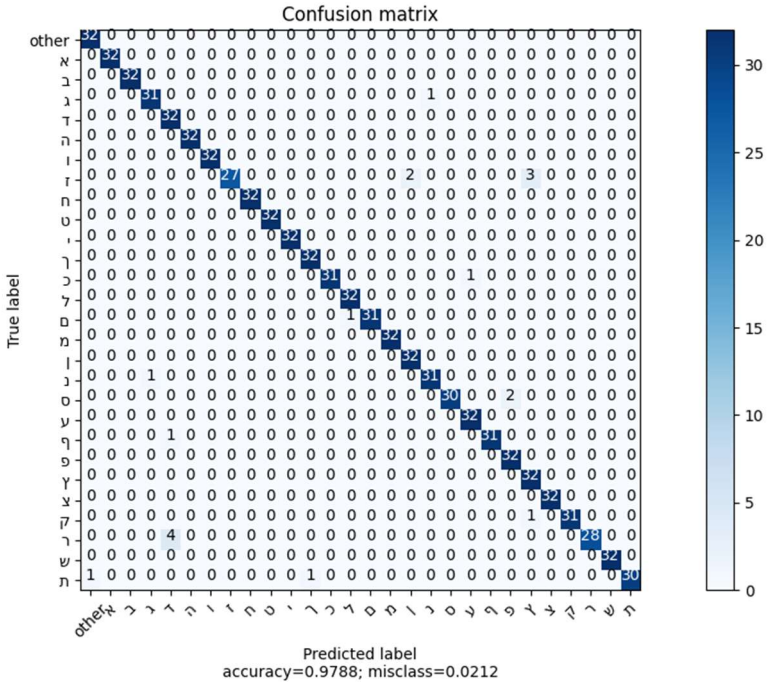
פונקציה שתציג את מטריצת הבלבול בצורה גרפית ויפה (הסבר על הקלט בפנים):

```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=False):
        Arguments
    ---------
    cm:           confusion matrix created by
sklearn.metrics.confusion_matrix
```

```
    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium',
'low']

    title:        the text to display at the top of the matrix

    cmap:         the gradient of the values displayed from
matplotlib.pyplot.cm

    normalize:    If False, plot the raw numbers
                  If True, plot the proportions


    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        if normalize:
```

```
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f};
misclass={:0.4f}'.format(accuracy, misclass))
    plt.show()
```

לאחר ששלחתי כקלט מטריצת בלבול פשוטה של תיקיית הvalidation, ורשימה של שמות האותיות, הוצגה התמונה הבאה:



ניתן לראות שבאופן כללי, המודל סיווג טוב את 32 האותיות שהיו בכל תיקיה שבתיקיית הvalidation. ואילו את האותיות המעטות שלא סיווג נכון הוא סיווג לאותיות דומות שהגיוני שאפילו אדם היה מסווג כך. כמו אות ז' כ-ן', ר' כ-ד'...

(ייתכן שבמציאות ישנה בעיה בצורת האות...;)

3. הכנת המזוזה לשליחתה להגהה:

בתחילה עלינו לחתוך את המזוזה ל22 שורות על מנת שנוכל לחתוך לאותיות ואח"כ לשלוח כל אות לזיהוי ע"י המודל.

שלב זה מתחלק לכמה פונקציות:

- שינוי התמונה לצבעי שחור לבן.
- חתיכת השוליים המקיפים את המזוזה.
- חיתוך התמונה לשורות.
- ניקוי רעשים מיותרים לכלוכים ונקודות מיותרות.
- חיתוך כל שורה לאותיות.
- שליחת האותיות לזיהוי ברשת הנוירונים.

כעת נפרט כל שלב:

הפונקציה הבאה משנה את התמונה לצבעי **שחור לבן**:

```
def change_color_to_black_and_white(path):
    original_image = cv2.imread(path, 1)
    # Extracting the height and width of an image
    h, w = original_image.shape[:2]
    # Convert the image to black and white by going over each
pixel
    # Each pixel has three features: red, green, blue,
    # and against each color there is a number that indicates a
quantity of it.
    # (255 of all colors is white)
    for i in range(h):
        for j in range(w):
            # If the amount of quantities is greater than 160 it
is closer to white,
            # else it is a letter = black
            if ((int(original_image[i, j, 0]) +
int(original_image[i, j, 1]) + int(
                    original_image[i, j, 2])) / 3) >= 160:
                original_image[i, j] = [255, 255, 255]
            else:
                original_image[i, j] = [0, 0, 0]
    # Save the black and white image
```

```
    a=os.path.split(path)[1]
    cv2.imwrite(fr'images/white-black-mezuzas/{a}',
original_image)
    return fr'images/white-black-mezuzas/{a}'
```

בדרך כלל המזוזה עצמה תהיה במקום כלשהוא בדף, וחלק נרחב מהתמונה יהיה מיותר ולכן **נחתוך את השוליים** המיותרים מסביב המזוזה.

```
def Removes_a_white_frame(path):
    im = Image.open(path)
    bg = Image.new(im.mode, im.size, im.getpixel((0, 0)))  #
create a new image white colored
    diff = ImageChops.difference(im, bg)  # computes the
'absolute value of the pixel-by-pixel difference
    # between the two images', which results in a difference
image that is returned
    diff = ImageChops.add(diff, diff, 2.0, -100)
    bbox = diff.getbbox()
    if bbox:
        iii = im.crop(bbox)
        iii.save(path)
    return path
```

חיתוך המזוזה לשורות:

מספר השורות במזוזה הוא קבוע, 22.

מאחר שחלק מאותיות המזוזה (כמו אותיות בכתב יד) יורדות לכיוון השורה הבאה, ועולות כלפי מעלה. כמו לדוגמא: האות ל' עולה כלפי מעלה, האותיות ן', ך', ק' יורדות. לא ניתן לקבוע את מיקום התחלת וסיום השורות ע"פ שורות הפיקסלים הרצופות הלבנות.

לכן נדרשתי למצוא את כמות הפיקסלים השחורים ביחס ללבנים, שיש בשורות הפיקסלים של הרווחים שבין השורות בממוצע. ולפי זה לקבוע במדויק את התחלת וסיום השורה.

לצורך כך כתבתי פונקציה בשם num_of_black_pixels המחזירה את מספר הפיקסלים השחורים והלבנים בתמונה.

עברתי על התמונה מלמעלה כלפי מטה וחיפשתי שורות פיקסלים שמספר הפיקסלים השחורים בשורה גדול מ0.001*מספר הפיקסלים השחורים בתוך כל התמונה. ושם קבעתי את נקודת החיתוך.

הפונקציה שומרת את השורות בתיקיית images/lines תחת התיקיה בעלת שם התמונה.

הקוד שמבצע את זה:

```
def cut_lines(path_to_mezuza):
    blacks, whites = num_of_black_pixels(path_to_mezuza)
    image = Image.open(path_to_mezuza)
    arr = np.array(image)
    rows_suspected_list = []  # A list in which the rows that are
suspected as spaces are inserted
    for i in range(image.height):
        counter = 0  # Counts the number of black pixels in a row
        for j in range(image.width):
            if arr[i][j].all() == 0:  # If the pixel is black
                counter += 1
        if counter >= (0.001 * blacks):
            rows_suspected_list.append(i)
    lines_list = [rows_suspected_list[1]]  # List of pixel rows of
beginning and end of a line in a mezuzah
    for i in range(1, len(rows_suspected_list)):
        if (rows_suspected_list[i] - rows_suspected_list[i - 1]) > 3:
            lines_list.append(rows_suspected_list[i])

    dire = os.path.split(path_to_mezuza)[1]
    dire = os.path.splitext(dire)[0]
    os.mkdir(fr'images/lines/{dire}')
    for i in range(len(lines_list) - 1):
        # Crop the line from the image
        image1 = image.crop((0, lines_list[i] - int(image.height *
0.02), image.width, lines_list[i + 1]))
        image1.save(fr'images/lines/{dire}/line{i + 1}.png')
    # Save the last row
    image1 = image.crop((0, lines_list[len(lines_list) - 1] -
int(image.height * 0.02), image.width, image.height))
    image1.save(fr'images/lines/{dire}/line{len(lines_list)}.png')
    return fr'images/lines/{dire}'
```

הפונקציה הראשית שחותכת לשורות:

```
def main_separation_to_lines(path):
    path_to_my_lines = cut_lines(path)
    if len(os.listdir(path_to_my_lines)) == 21:
```

```
        dir = os.path.split(path_to_my_lines)[1]
        cut_line_21(dir, 'line21.png')
    return path_to_my_lines
```

מכיוון שבין שורות 21-22 אין כמעט פיקסלים שחורים, והשטח הלבן גדול.

האחוזים משתנים ולכן שלחתי לפונקציה cut_line_21 את שורה 21 כדי שתחתוך שורה זו ל2 עם אחוזים שונים ומתאימים. הפונקציה עובדת באותו אופן של הפונקציה cut_lines.

ניקוי השורות:

כאמור לעיל, יש אותיות שעולות ויורדות ולכן לאחר חיתוך השורות מתקבלת תוצאה כמו שרואים בתמונה:

ובכדי שנוכל לחתוך לאותיות כמו שצריך, כתבתי פונקציה המנקה כמה שיותר את חלקי האותיות שלא קשורים לשורה.

בפונקציה זימנתי את הפונקציה connectedComponentsWithStats המסופקת ע"י ספריית CV2. זהו יישום אלגוריתמי של תורת הגרפים המשמש למציאת רכיבים קשירים דמויי "blob" בתמונה בינארית.

בעזרת פונקציה זו נוכל לפלח ולנתח בקלות רבה יותר את הרכיבים הללו. מאחר והיא מחזירה 4-tuple של:

1. מספר הרכיבים הקשירים שזוהו.
2. מסכה בשם labels, שיש לה את אותם ממדים מרחביים כמו תמונת הקלט. עבור כל מיקום בlabels, יש לנו ערך מזהה integer שמתאים לרכיב המחובר אליו שייך הפיקסל.
3. stats: סטטיסטיקה על כל רכיב מחובר, כולל הקואורדינטות והשטח של התיבה התוחמת (בפיקסלים).
4. centroids: המרכז (x ,y) של כל רכיב מחובר.

להלן הפונקציה:

```
def clean_noises(directory, image_line):
    img = cv2.imread(fr'{directory}/{image_line}', 0)
    hight, width = img.shape
    _, blackAndWhite = cv2.threshold(img, 127, 255,
cv2.THRESH_BINARY_INV)
    # apply connected component analysis to the blackAndWhite image
    nlabels, labels, stats, centroids =
cv2.connectedComponentsWithStats(blackAndWhite, None, None, None, 8,
cv2.CV_32S)
    sizes = stats[1:, -1]  # get CC_STAT_AREA component
    img2 = np.zeros((labels.shape), np.uint8)

    for i in range(0, nlabels - 1):
        # extract the connected component statistics and centroid for
        # the current label
        x = stats[i, cv2.CC_STAT_LEFT]  # the starting x coordinate
of the component
        y = stats[i, cv2.CC_STAT_TOP]  # the starting x coordinate of
the component
        w = stats[i, cv2.CC_STAT_WIDTH]  # the width (w) of the
component
        h = stats[i, cv2.CC_STAT_HEIGHT]  # the height (h) of the
component
        area = stats[i, cv2.CC_STAT_AREA]  # The centroid (x, y)-
coordinates of the component
        (cX, cY) = centroids[i]

        small_dotted = sizes[i] >= 20
        down = y < hight / 2 + 7
        up = y > 0 and (y + h) < (hight / 2)

        if (small_dotted and down and up):
                img2[labels == i + 1] = 255
    # change again to white-background and black-letters
    res = cv2.bitwise_not(img2)
    cv2.imwrite(fr'{directory}/{image_line}', res)
```

התוצאה המתקבלת על הדוגמא לעיל:

(המודל מצליח לזהות את האות עם הלכלוכים המעטים הללו שנשארו)

חיתוך לאותיות:

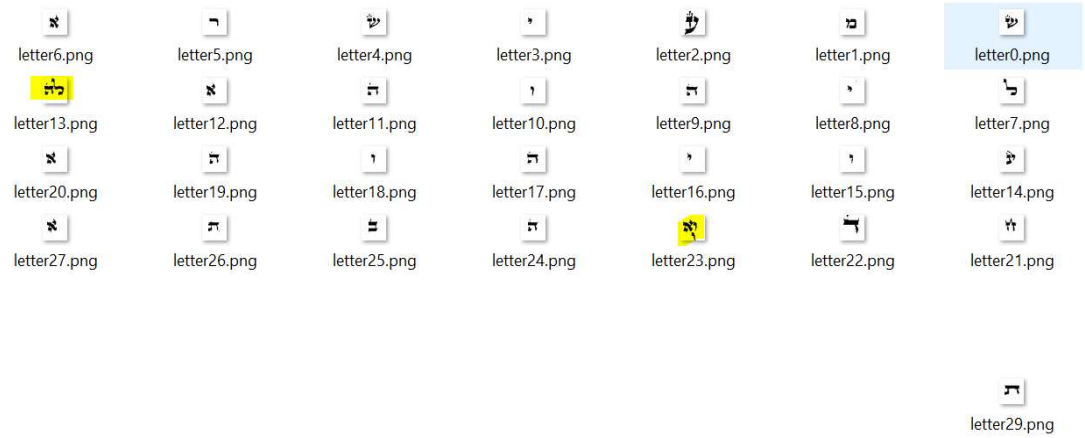
הפונקציה עוברת על התמונה לרוחבה (מימין לשמאל) ומחפשת את הרווחים שיש בין האותיות (כאשר אחוזי הפיקסלים השחורים בעמודת הפיקסלים נמוכה) ושם היא קובעת את נקודת החיתוך. הפונקציה עובדת באותו האופן של חיתוך השורות.

הפונקציה שומרת את האותיות בתיקיית images/letters תחת תיקיה בעלת שם התמונה, שבתוכה תיקייה לכל שורה בנפרד, שבתוכה האותיות שנחתכו משורה זו.

להלן הקוד:

```
def cut_to_letters(dir, line):
    image = Image.open(fr'images/lines/{dir}/{line}')
    arr = np.array(image)
    list1 = [image.width - 1]  # A list in which the columns that are
suspected as spaces are inserted
    for i in range(image.width - 1, 0, -1):  # From right to left because of
the order in Hebrew letters
        counter = 0  # Counts the number of black pixels in a column
        for j in range(image.height):
            if arr[j][i].any() == 0:  # If the pixel is black
                counter += 1
        if (counter / image.height) > 0.042:
            list1.append(i)
    list2 = [list1[0]]  # List of pixel columns of beginning and end of a
letter in a line
    for i in range(1, len(list1) - 1):
        if (list1[i + 1] - list1[i]) < -1:
            list2.append(list1[i])
    num_line = os.path.splitext(line)[0]
    os.makedirs(fr'images/letters/{dir}/{num_line}')
    for i in range(len(list2) - 1):  # Crop the letters from the line
        image1 = image.crop((list2[i + 1], 0, list2[i], image.height))
        image1.save(fr'images/letters/{dir}/{num_line}/letter{i}.png')
    image1 = image.crop((0, 0, list2[len(list2) - 1], image.height))
    image1.save(fr'images/letters/{dir}/{num_line}/letter{len(list2)}.png')
```

להלן דוגמא לתוצאות:



כפי שניתן לראות, האותיות נחתכות יחסית בסדר אבל לפעמים 2 אותיות נחתכות לתמונה אחת למרות שאינן מחוברות, וזו בעיה.

בתחילה חשבתי לפתור זאת באמצעות הפונקציה למציאת רכיבים קשירים, שזימנו בניקוי האותיות... אך ישנה בעייתיות בתווים 'ה', 'ק' שם יש חלקי תו שאינם מקושרים.

מצד שני, אי אפשר גם לקבוע תיבה מלבנית מסביב לכל האותיות כי יש מקרים (כשרים הלכתית לגמרי) בהם יש חדירה של תווים לתוך השטח של התווים הסמוכים כמו 'ב' שנכנסת מתחת ל'ר' שלפניה או 'ע' שגולשת מתחת ל י' שלאחריה, ל' שעולה על אות שאחריה וכיוצא בזה. לדוגמא: 

בנוסף, נעזרתי בטכנולוגיית Tesseract-OCR המזהה מיקומים של מילים ואותיות, למרות שאומרים שהיא מתקשה עם כתב יד. וקיוויתי אולי היא תצליח לזהות בצורה טובה את מיקומם המדויק של האותיות, אך התוצאות שהניבה בסך הכללי היו פחות מוצלחות מהפונקציה הנ"ל שכתבתי. (פעמים רבות חתכה אותיות ל2 בלי הגיון...)

סיווג האותיות:

בשלב הבא יש לשלוח את התמונות של האותיות לרשת הנוירונים על מנת שתפענח את האות.

מכיוון שהמודל מחזיר מספר קטגוריה בין 0 ל27 יש צורך בפונקציה שממירה את הערך לאות בעצמה.

```
def ConversionToCharachters(num: int):
    classes = ['other', 'א', 'ב', 'ג', 'ד', 'ה', 'ו', 'ז', 'ח',
'ט', 'י', 'ך', 'כ', 'ל', 'ם', 'מ', 'ן', 'נ', 'ס', 'ע',
               'ף', 'פ',
               'ץ', 'צ', 'ק', 'ר', 'ש', 'ת']
    return classes[num]
```

להלן הפונקציה ששולחת תמונה לזיהוי במודל:

```
def To_classify(path):
    img = Image.open(str(path))
    # Small changes to the image in order to fit it to the model
    img = np.array(img).astype('float32')
    img = transform.resize(img, (28, 28, 1))
    img = np.expand_dims(img, axis=0)
    my_model = load_model("model_try3.h5")
    output = my_model.predict(img)
    i = np.argmax(output, axis=1)
    return ConversionToCharachters(int(i))
```

להלן הפונקציה הראשית המקבלת כקלט נתיב לתיקיית האותיות, ומחזירה מילון שבו המפתח: מספר השורה, והערך: רשימת האותיות שזוהו בשורה זו.

```
def MainClassification(path):
    iq.main_change_size(path)
    dict={}
    for line in os.listdir(path):
        list = []
        for letter in os.listdir(fr'{path}/{line}'):
            list.append(To_classify(fr'{path}/{line}/{letter}'))
        dict.update({line:list})
    return dict
```

4. <u>הגהת המזוזה:</u>

בחלק זה נמצא אם ישנן:

- אותיות מיותרות.
- אותיות חסרות.
- אותיות שגויות.
- אותיות החשודות כצמודות.

השוואה פשוטה בין התבנית הנכונה ($P = p_1p_2\ldots p_m$), לטקסט התווים הנבדק ($T = t_1t_2\ldots t_m$) לא תמיד תביא תוצאה נכונה. לדוגמא: כאשר הסופר יחסיר אות או יכתוב אות מיותרת, ההשוואה תהיה חסרת משמעות, כי היא תיתן שיש הרבה שגיאות למרות שהשגיאה היא אחת, ואם נוסיף אות זאת או נמחוק אותה, הכל יהיה תקין.

לכן עלינו לבדוק איזה השוואה תביא למספר טעויות מינימלי.

לצורך כך נגדיר 3 סוגי שגיאות:

- <u>Mismatch:</u> כאשר יש התאמה מלאה (כולל אינדקסים), למעט אות אחת, נאמר שיש שגיאה אחת מסוג mismatch. לדוגמא:

  T = ו**ע**בדתם

  P = ו**א**בדתם

- <u>Insertion:</u> כאשר הייתה יכולה להיות התאמה, אך הוכנסה אות נוספת <u>לטקסט</u>. נאמר שיש שגיאה אחת מסוג insertion. לדוגמא:

  T = מז**ו**זות

  P = מזזות

  כאן אפשר לומר שיש 3 שגיאות mismatch או שגיאת insertion אחת.

- <u>Deletion:</u> כאשר הייתה יכולה להיות התאמה, אך הוסרה אות אחת <u>מהטקסט</u>. נאמר שיש שגיאה אחת מסוג deletion. לדוגמא:

  T = לטטפת

  P = לט**ו**טפת

  שוב, כאן ניתן לומר שיש 3 שגיאות mismatch או שגיאת deletion אחת.

לפתרון בעיה זו בצורה יעילה, מימשתי את אלגוריתם **edit distance** המבוסס על תכנות דינאמי. הסבר האלגוריתם:

כרגיל בתכנות דינאמי נבנה טבלה $m \times n$ (בגודל אורכי המחרוזות) של פתרונות לתתי-בעיות בעזרת אינדוקציה, עד שנגיע לפתרון הבעיה השלמה.

האיבר ה- $a_{i,j}$ יהיה שווה למס' המינימלי של פעולות edit כך ש- $p_1p_2\ldots p_i$ תתאים לטקסט המסתיים במקום ה-j:

<u>בסיס:</u> אם i=0 מדובר בעצם בתבנית ריקה ולכן $a_{0,j} = 0$ (אין צורך בשום פעולות עריכה כדי להתאים את התבנית).

אם j = 0 , אז הטקסט הוא ריק, כלומר ניתן לחשוב שבוצעו i פעולות מחיקה על הטקסט ולכן $a_{i,0} = i$.

<u>מקרה 1</u>: אם התווים זהים $p_i = t_j$ אזי לא נדרשת שום פעולת עריכה ולכן $a_{i,j} = a_{i-1,j-1}$.

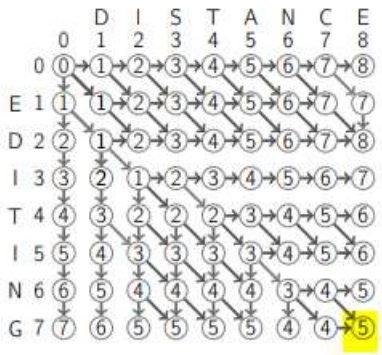
<u>מקרה 2</u>: אם התווים שונים $p_i \neq t_j$ אזי ניקח את המינימום מפעולות העריכה: או mismatch ואז $a_{i,j} = a_{i-1,j-1} + 1$, או insertion ואז $a_{i,j} = a_{i,j-1} + 1$, או deletion ואז $a_{i,j} = a_{i-1,j} + 1$.

בסה"כ:

$$a_{i,j} = \min(a_{i-1,j-1}, a_{i-1,j}, a_{i,j-1}) + 1.$$

לאחר שסיימנו למלא את הטבלה, השורה האחרונה היא למעשה כל ההתאמות של התבנית המלאה לטקסט שמסתיים בכל מקום j. ועל פי טבלה זו קל לשחזר את המסלול ואת פעולות הedit שבוצעו. (פרוט השחזור בהמשך בפונקציה FinalResult)

בתמונה הבאה ניתן לראות איך תתמלא המטריצה עבור 2 המילים DISTANCE, EDITIMG.



התא האחרון (המודגש) מסמל את מספר התיקונים המינימלי על מנת להגיע למחרוזות שוות.

להלן הקוד שמממש את האלגוריתם:

```
def EditDistance(check, correct):
    s1 = check
    s2 = correct
    # הגדרת גודל המטריצה + 1 כדי שיוכל להתחיל ממספר 1 והמיקום 0 ישאר מאופס
    a = [[0 for x in range(len(check) + 1)] for y in range(len(correct) + 1)]

    # יצירת המטריצה והכנסת קירוב השינויים
    # אתחול עמודה ושורה ראשונים
    for i in range(len(s2) + 1):
        a[i][0] = i
    for j in range(len(s1) + 1):
        a[0][j] = j
    # ריצה על כל המטריצה והכנסת מרחק השינויים לפי החישוב של תכנון דינאמי
    for i in range(1, len(s2) + 1):
        for j in range(1, len(s1) + 1):
            insertion = a[i][j - 1] + 1
            deletion = a[i - 1][j] + 1
            match = a[i - 1][j - 1]
            mismatch = a[i - 1][j - 1] + 1
            if s1[j - 1] == s2[i - 1]:
                a[i][j] = min(insertion, min(deletion, match))
            else:
                a[i][j] = min(insertion, min(deletion, mismatch))
    return a, a[len(a) - 1][len(a[0]) - 1]
```

פונקציה שמחשבת דרך חזור במטריצה. מקבלת את אורך 2 המחרוזות (i ,j), המחרוזות והמטריצה.

```
def FinalResult(i, j, check, correct, a):
    s_correct.clear()
    s_check.clear()
    if i == 0 and j == 0:
        return
    if i > 0 and a[i][j] == (a[i - 1][j] + 1):
        FinalResult(i - 1, j, check, correct, a)
        s_correct.append(correct[i - 1])
        s_check.append("-")
    else:
        if j > 0 and a[i][j] == (a[i][j - 1] + 1):
            FinalResult(i, j - 1, check, correct, a)
```

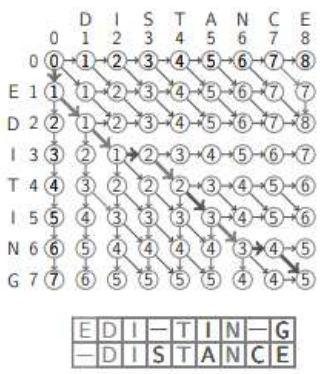
```
            s_correct.append("-")
            s_check.append(check[j - 1])
        else:
            FinalResult(i - 1, j - 1, check, correct, a)
            s_correct.append(correct[i - 1])
            s_check.append(check[j - 1])
    return s_check, s_correct
```

הסבר:

נעמיד את 2 האותיות האחרונות של המילים זו תחת זו, ונתחיל מהמשבצת האחרונה במטריצה ומשם נלך אחורה לכוון המספר המינימלי מבין השכנים של המשבצת. אם האותיות לא מתאימות נשאיר אותם כרגיל חוץ משתי מקרים: אם האות במשבצת שלפני, כן שווה לאות הזאת (ה-i וה-j שווים) נשים קו במקום הזה במילה הנכונה כיון שהאות הנוכחית כנראה מיותרת. ואם קורה אותו דבר באותה עמודה נשים קו במילה הלא נכונה כיון שזה אומר שישנה אות חסרה.

אחרי שהמילים מסודרות בצורה כזאת יותר פשוט להשוות בין המילים ולהגיע לתוצאה של מינימום טעויות.

דוגמא:




```
OutputAlignment(i, j)
if i = 0 and j = 0:
  return
if backtrack(i, j) = ↓:
  OutputAlignment(i − 1, j)
  print A[i] / –
else if backtrack(i, j) = →:
  OutputAlignment(i, j − 1)
  print – / B[j]
else:
  OutputAlignment(i − 1, j − 1)
  print A[i] / B[j]
```

לאחר זימון הפונקציות הנ"ל, נקבל שתי רשימות מורחבות ובמקומות בהם חסרים אותיות ישנו מקף במילה הנבדקת ובמקומות בהם יש אותיות מיותרות ישנו מקף במילה הנכונה.

להלן הפונקציה שמשתמשת באלגוריתם הנ"ל ועושה את הבדיקה בפועל, תוך כתיבת התוצאות בקובץ שיוחזר למשתמש.

```
def Checkline(num_of_line, letters_for_check, correct_letters, path_to_result_file):
    result_file = open(path_to_result_file, 'a', encoding='utf-8')
    mat, min_of_mistakes = EditDistance(letters_for_check, correct_letters)
    s_check, s_correct = FinalResult(len(correct_letters), len(letters_for_check), letters_for_check, correct_letters,
                                     mat)
    if min_of_mistakes == 0:
        return
    result_file.write(f"{num_of_line}: \n")
    for i in range(len(s_check)):
        if s_check[i] == s_correct[i]:
            continue
        if s_check[i] != s_correct[i] and s_check[i] != '-' and s_correct[i] != '-':
            if s_check[i] == 'other':
                result_file.write(
                    f"   letter {i + 1}: the letters are probably stuck together instead of {s_correct[i]} {s_correct[i + 1]} \n")
                i = i + 2
                continue
            else:
                result_file.write(
                    f"   letter {i + 1}: you wrote {s_check[i]} instead of {s_correct[i]} \n")  # את האות הלא נכונה ללא סיבה נראית לעין
        if (s_check[i] == '-' and s_check[i - 1] != 'other'):
            result_file.write(f"   letter {i + 1}: you missed out the letter {s_correct[i]} \n")
        if (s_correct[i] == '-'):
            result_file.write(f"   letter {i + 1}: you wrote the letter {s_check[i]} its spare letter \n")
    result_file.close()
```

הפונקציה שמזמנת את הפונקציה הנ"ל עבור כל המזוזה.

```
def MainChecker(mezuza_name, dic):
    import Mezuza
    m = Mezuza()
    for line, list in dic.items():
        Checkline(line, list, m.correct_mezuza[line],
fr"images/results_of_{mezuza_name}.txt")
```

ב`m.correct_mezuza` יש את המילון של המזוזה המקורית הכשרה.

לאחר מכן התוצאה המתקבלת בקובץ txt על דוגמא מסוימת:

```
line1:
   letter 9: the letters are probably stuck together instead of ה י
   letter 14: the letters are probably stuck together instead of ה ל
   letter 19: you wrote ץ instead of י
   letter 20: you missed out the letter ה
   letter 29: the letters are probably stuck together instead of ת ב
line2:
   letter 4: you wrote ת instead of ה
   letter 6: the letters are probably stuck together instead of ה ל
   letter 10: the letters are probably stuck together instead of כ ב
   letter 11: the letters are probably stuck together instead of ל כ
   letter 13: you missed out the letter ל
   letter 15: you missed out the letter ב
```

15. קוד התוכנית

כעת אפרט על מחלקות נוספות בצד השרת:

מחלקת Mezuza :

המחלקה שומרת את התכונות של המזוזה הכשרה שאותה נשווה עם הנבדקת.

- Init – פעולה הבונה מופע מסוג המחלקה ומאתחלת בנתונים את המילון correct_mezuza.

```
class Mezuza:
    def __init__(self):
        self.correct_mezuza = dict({"line1": ["ש", "מ", "ע", "י", "ש", "ר", "א", "ל", "י", "ה", "ו", "ה", "א", "ל",
                                              "ב", "י", "נ", "ו", "י", "ה", "ו", "ה", "א", "ח", "ד", "ו", "א", "ה",
                                              "ב", "ת", "א", "ת"],
                                 "line2": ["י", "ה", "ו", "ה", "א", "ל", "ה", "י", "ד", "ב", "כ", "ל", "ל", "ב",
                                           "ב", "ד", "ו", "ב", "כ", "ל", "נ", "פ", "ש", "ד", "ו", "ב", "כ", "ל",
                                           "מ", "א", "ר", "ד", "ו", "ה", "י", "ו"],
                                 "line3": ["ה", "ד", "ב", "ר", "י", "ם", "ה", "א", "ל", "ה", "א", "ש", "ר", "א",
                                           "נ", "כ", "י", "מ", "צ", "ו", "ד", "ה", "י", "ו", "ם", "ע", "ל",
                                           "ל", "ב", "ב", "ד", "ו", "ש", "נ", "נ", "ת", "ם"],
                                 "line4": ["ל", "ב", "נ", "י", "ד", "ו", "ד", "ב", "ר", "ת", "ב", "ם", "ב", "ש",
                                           "ב", "ת", "ד", "ב", "ב", "י", "ת", "ד", "ו", "ב", "ל", "כ", "ת", "ד",
                                           "ב", "ד", "ר", "ד"],
                                 "line5": ["ו", "ב", "ש", "כ", "ב", "ד", "ו", "ב", "ק", "ו", "מ", "ד", "ו", "ק", "ש",
                                           "ר", "ת", "ם", "ל", "א", "ו", "ת", "ע", "ל", "י", "ד", "ד", "ו",
                                           "ה", "י", "ו", "ל", "ט", "ט", "פ", "ת"],
                                 "line6": ["ב", "י", "ו", "ע", "י", "נ", "י", "ד", "ו", "כ", "ת", "ב", "ת", "ם",
                                           "ע", "ל", "מ", "ד", "ד", "ו", "ת", "ב", "י", "ת", "ד", "ו", "ב", "ש",
                                           "ע", "ר", "י", "ד"],
                                 "line7": ["ו", "ה", "ר", "ה", "א", "ם", "ש", "מ", "ע", "ת", "ש", "מ", "ע", "ו",
                                           "א", "ל", "מ", "צ", "ו", "ת", "י", "א", "ש", "ר", "א", "נ", "כ",
```

מחלקת main:

מחלקה ראשית שמנהלת את כל תהליך ההגהה.

```
from PIL import Image
import os
import image_quality as iqc
import cleanNoise as c
import Separation_of_letters as s
import To_classify
import Proofreading_a_mezuza as p
from fpdf import FPDF
import codecs
```

פונקציות המחלקה:

- SaveAsPdf – פונקציה הממירה קובץ txt לפורמט PDF.
מקבלת נתיב לקובץ txt ושומרת קובץ pdf מתאים בשם זהה.

```
def SaveAsPdf(path):
    # save FPDF() class into a variable pdf
    pdf = FPDF()
    # Add a page
    pdf.add_page()
    # set style and size of font that you want in the pdf
pdf.add_font("DeJaVu",'',r'C:\Users\1\PycharmProjects\MyFinalPr
oject\venv\Lib\site-packages\ttf\DejaVuSans.ttf',uni=True)
    pdf.set_font("DeJaVu", size=11)
    # open the text file in read mode

    f = codecs.open(f"{path}", "r", encoding="UTF-8")
    # insert the texts in pdf
    for x in f:
        pdf.cell(200, 10, txt=x, ln=1, align='l')
    # save the pdf with name .pdf
    pdf.output(f"{path}.pdf")
```

- MainFunc - פונקציה ראשית המקבלת נתיב למזוזה וקוראת לכל הפונקציות של עיבוד התמונה הכולל חיתוך השורות ואז האותיות מתוך המזוזה, ושליחתם לרשת נוירונים. לאחר מכן, שליחתם להגהה ולבסוף החזרת אובייקט ללקוח המכילה את כל השגיאות שנמצאו.

```
def MainFunc(path):
    image_mezuza = Image.open(path)
    path = iqc.main_image_quality(path)  # הכנת המזוזה לחיתוך שורות ואותיות
    print('חותך לשורות')
    path_to_my_lines = s.main_separation_to_lines(path)  # חיתוך ל22 שורות
    print('מנקה את השורות')
    c.main_cleaner(path_to_my_lines)  # על מנת לנקות את השורות
    print('חותך לאותיות')
    path_to_my_letters = s.main_separation_to_letters(path_to_my_lines)  # שליחת השורות הנקיות לחיתוך אותיות
    print('מסווג')
    dict = To_classify.MainClassification(path_to_my_letters)  # שליחת האותיות לזיהוי
    mezuza_name = os.path.splitext(path)[0]
    print('מגיה')
    p.MainChecker(mezuza_name,dict)  # שליחת האותיות להגהה
    print('גמר')
```

<u>מחלקת api:</u>

תפקיד המחלקה: מקשרת בין הclient לserver. ומעבירה נתונים בינהם.

הספריות הנדרשות:

```
from flask import Flask, request, send_file
from flask_cors import CORS
import asyncio

loop = asyncio.get_event_loop()
app = Flask(__name__)
CORS(app)
import main
```

פונקציות המחלקה:

- uploadFile – פונקציה המקבלת תמונה של מזוזה מה-client ושולחת אותה לעיבוד והגהה.

```
@app.route('/upload-file', methods=['POST'])
def uploadFile():
    my_file = request.files['image']
    print(type(my_file))
    print(my_file.filename)
    my_file.save(f'./{my_file.filename}')
    main.MainFunc(f'./{my_file.filename}')
    return 'true'
```

- get_file – פונקציה המחזירה את קובץ התוצאות ל-client.

```
@app.route("/download")
async def get_file():
    global mezuza_name
    mezuza_name = os.path.splitext(mezuza_name)[0]
    try:
        return
send_file(fr'images/results/{mezuza_name}.txt',
attachment_filename='results.txt',as_attachment=True
)

    except Exception as e:
        return str(e)
```

פונקציות מרכזיות בצד לקוח

Upload.jsx- קומפוננטה להעלאת קובץ לדף האתר:

כדי שהמשתמש יוכל להעלות סריקה של מזוזה, יצרתי קומפוננטה שתהיה אחראית לכך.

את המזוזה תבדוק התוכנה ותחזיר עליה משוב.

לקומפוננטה יש כמה מאפייני עיצוב כדי להביא ללקוח חווית שימוש טובה והבסיס שלה נראה כך:

```
return (
    <div>
      {
         showUpload ?
         <div className="upload">
           <ImageUploader className="fileUploader c"
             withIcon={true}
             withLabel={false}
             withPreview={true}
             buttonText={"Add a Mezuzah"}
             fileSizeError={"File size is too big!"}
             fileTypeError={"This extension is not supported!"}
             onChange={onDrop}
             imgExtension={props.imgExtension}
             maxFileSize={props.maxFileSize}
             buttonClassName='fileContainer chooseFileButton'
           />
           {showConfirm?
            <div className="confirm">
             <button type="button" class="btn btn-outline-dark"
onClick={upload_file}>Confirm upload</button>
           </div>:''
}
           </div> :
               <div>
                 <Download className='download' />
               </div>

      }

    </div>
  )
```

כמו שרואים ישנה פונקציה שמעלה את המזוזה לשרת. כלומר, שולחת אותה מה-React לPython. פונקציה זאת מופעלת על ידי לחיצה על הכפתור שכתוב עליו Confirm upload.

כפתור זה יוצג רק לאחר שהמשתמש ייבא קובץ שמעוניין לשלוח לבדיקה.

הפעולה onDrop מתבצעת באופן אוטומטי בזמן העלאת הקבצים.

כשהמשתמש מעלה קובץ, הקובץ נשמר בState שנקרא File.

```
  const onDrop = (pictureFiles, pictureDataURLs) => {
    setShowConfirm(true);for display the button  confirm upload
    setFile(file.concat(pictureFiles))
    const newImagesUploaded = pictureDataURLs.slice(
      props.defaultImages.length
    );
    console.warn("pictureDataURLs =>", newImagesUploaded);
    props.handleChange(newImagesUploaded);
  };
```

בעת לחיצה על הכפתור Confirm upload תתבצע הפונקציה upload_file:

```
onClick={upload_file}
```

```
const upload_file = () => {
    setShowUpload(false);
    const formData = new FormData();

    formData.append('image', file[0]);
    console.log(formData)
    axios.post('http://127.0.0.1:5000/upload-file', formData, {

      onUploadProgress: (progressEvent) => {
        //how much time need finish
        if (progressEvent.lengthComputable) {
          const copy = { ...uploadProgress };
          copy[file[0].name] = {
            state: "pending",
            percentage: (progressEvent.loaded / progressEvent.total) *
100
          };
          setUploadProgress(copy);
        }
      },
      headers: {
        'Content-Type': 'multipart/form-data'
      }
    })
      .then(response => {
        const copy = { ...uploadProgress };
        copy[file[0].name] = { state: "done", percentage: 100 };
        setUploadProgress(copy);
```

```
        setSuccessfullUploaded(true)
        setResult(response.data)
        console.log(response.data)
        return true;
      })
      .catch((error) => {
        if (error.response) {
          console.log(error.response)
          console.log(error.response.status)
          console.log(error.response.headers)
          const copy = { ...uploadProgress };
          copy[file[0].name] = { state: "error", percentage: 0 };
          setUpload(true)
          Progress(copy)
          return false;
        }
      })
  }
```

הפונקציה מוספיה לformdata את הקובץ שנשמר לפני כן עם העלאתו ושולחת אותו לכתובת בה נמצא השרת:

```
axios.post('http://127.0.0.1:5000/upload-file', formData,
```

אם הועלתה שגיאה השגיאה תפורט בconsole בגלל שהשגיאות נתפסו בשורות האחרונות של הפונקציה.

Download.jsx- קומפוננטה להורדת הקובץ:

```
export default function Download() {
    const downloadFile = () => {
       FileService.get_file()
    }

    return (
        <div>
          <button className="btn btn-outline-dark"
onClick={downloadFile}>Download Results</button>
        </div>
    )
```

כפתור זה יוצג לאחר קבלת קובץ השגיאות מהserver.

כמו שרואים בלחיצה על כפתור זה תופעל הפונקציה downloadFile שמזמנת את get_file:

```
   get_file() {
        return axios({
            url: 'http://127.0.0.1:5000//download',
            method: 'GET',
            responseType: 'blob',
        }).then((response) => {
            const url = window.URL.createObjectURL(new
Blob([response.data]));
            const link = document.createElement('a');
            link.href = url;
            link.setAttribute('download', 'results.txt');
            document.body.appendChild(link);
            console.log(link)
            link.click();
        });
    }
```

16. תיאור מסד הנתונים:

מסד הנתונים מורכב מתמונות רבות של אותיות אותם חתכתי מתוך הרבה סריקות שונות של מזוזות.

בנוסף, על מנת לעבות את מאגר התמונות כתבתי קוד עם יכולת פונקציונלית של עיבוד תמונה שיוצרת מכל תמונה – מספר תמונות שונות .

עד שהגעתי לכמות שכוללת של כמעט מעל 11,200 תמונות !

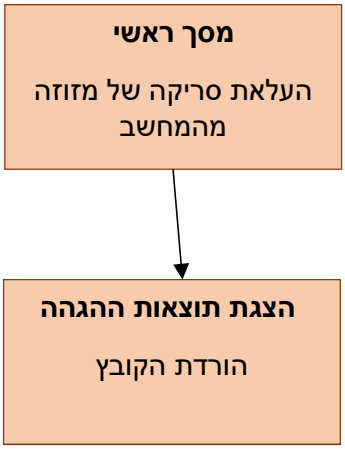
בנוסף, נתינת גודל אחיד לכלל התמונות במאגר וכן שינוי הצבעים לשחור -לבן עבור אימון תקין ונכון של הרשתות השונות.

לאחר מכן, חלוקת מאגר הנתונים ל – 3 תתי מאגרים עליהם יתאמנו הרשתות.

בתמונה הבאה ניתן לראות את החלוקה המפורטת לתיקיות, כל תיקייה מכילה 28 תתי תיקיות המכילות את כל האותיות מא'-ת' וכן תיקיית other של אותיות צמודות.

17. תיאור מסכים:

17.1 תרשים מסכים - Screen flow diagram:



17.2 תפקידי המסכים:

1. מסך פתיחה בו המשתמש מעלה סריקה של מזוזה.

2. מסך המציג את תוצאות ההגהה. ואפשרות להורדה.

17.3 תיאור מסך הפתיחה:

במסך הפתיחה על המשתמש לייבא סריקה של מזוזה מתוך המחשב. רצוי שהתמונה תהיה חדה, ברורה ונקיה.

לאחר בחירת המזוזה, היא תוצג למשתמש וביכולתו להתחרט על הבחירה ולבחור מחדש.

לאחר העלאת המזוזה יוצג למשתמש כפתור שעל ידו יועלו הקבצים לשרת.

אם התמונה תקינה, המסך יתחלף למסך השני המציג את התוצאות.

17.4 מסך האפליקציה בליווי הסברים:

מסך פתיחה:

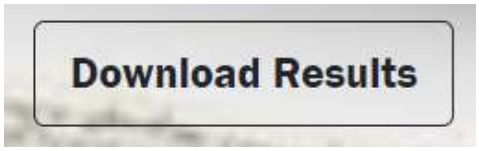
במסך זה המשתמש יעלה את המזוזה אותה הוא רוצה לשלוח לבדיקה.

ע"י לחיצה על כפתור זה תיפתח האפשרות לבחירת קובץ מהמחשב.

לאחר העלאת המזוזה היא תוצג למשתמש:

אחרי העלאת המזוזה שרצה, על המשתמש ללחוץ על הכפתור הזה:

מסך שני – הצגת תוצאות ההגהה:



בלחיצה על כפתור זה תוצאות ההגהה יורדו, בפורמט PDF.

18. מדריך למשתמש:

בתחילה יעלה המשתמש את המזוזה שברצונו לבדוק, ולאחר מכן ילחץ על הכפתור שמעלה את המזוזה לשרת.
לאחר הגהת המזוזה תוצגנה למשתמש תוצאות ההגהה, והוא יוכל להוריד אותם בפורמט PDF.

19. בדיקות והערכה:

רשת הנוירונים:

על מנת לבדוק האם הרשת עובדת כראוי ומזהה את האותיות באופן מדויק היה צריך לעשות בדיקות תקינות רבות.

בהתחלה הרצתי את הפונקציות שמאמנות את המודל והם הביאו תוצאה שלא הייתה מדויקת מאד, לכן הוספתי עוד שכבות במודל והרצתי שוב ועדיין התוצאה לא הייתה מספקת. לאחר מכן הגדלתי את מספר ה-epoch , שזה מספר פעמים שהרשת מתאמנת על התמונות, ולאחר כמה ניסויים אחוז ההצלחה של המודל היה גבוה (99.4).

לאחר מכן, כדי לבדוק האם הרשת באמת מדויקת הרצתי את המודל על הרבה תמונות שהוא לא הכיר קודם ולא התאמן עליהם. ואכן הוא דייק וידע לסווג את התמונות במדויק. (פרוט בפרק של אלגוריתמים מרכזיים).

הגהת המזוזה:

כדי לראות האם המערכת מוצאת את השגיאות בצורה נכונה, לקחתי כמה מזוזות ומחקתי מהם אותיות מסוימות, שיניתי אותיות לאחרות, הוספתי אותיות. ואכן, התוצאות שהאלגוריתם מצא היו מדויקות להפליא.

20. ניתוח יעילות:

סיבוכיות אלגוריתם פיתוח המודל הינו סיבוכיות גבוהה ולכן הרצתי מספר מוגדר של epoch עד לקבלת התוצאה הרצויה.
מספר סבבי האימון שהמודל מריץ כל תמונה -epoch - אם כך הסיבוכיות הינה כמספר התמונות שנמצאות בתיקיה train.
עבור הרשת הנבחרת ביצעתי 25 סבבים.
המודל ייעצר מלהתאמן מאחת משתי הסיבות הנ"ל:

- כשהמודל יגיע ל25 הסבבים שהוגדרו.

- ייעצר עקב אימון רב מדי כך שאם ימשיך להתאמן יותר יגיע לתוצאה פחות טובה (OverFitting).

לאחר מכן כאשר הרשת מוכנה המערכת רצה במספר שכבות הרשת.

21. מסקנות:

בתחילת הדרך כאשר היה עלי לתכנן את הפרויקט הבנתי שהמשימה שעומדת לפני לא תהיה פשוטה כלל.
הייתי צריכה להשקיע שעות רבות ללמוד את החומר ולהבין אותו לעומק, כל זה עוד לפני שלב הכתיבה עצמה של הקוד, כשהגעתי לשלב הכתיבה שהוא השלב העיקרי, נדרש ממני הרבה ידע והתנסות על מנת להגיע לתוצאות יעילות וטובות.
כתיבת הקוד הייתה מורכבת אך גם מאוד מאתגרת, למדתי המון נושאים חדשים שלולא הפרויקט לא הייתי יודעת אותם בתור סטודנטית, בעיקר בנושא של עיבוד תמונה.
ובכלל זכיתי לדעת פיתון ברמה גבוהה ולהכיר ספריות חשובות ומהותיות שקימות בה, לעומק.
ניתן לומר שהפרויקט קידם אותי המון בתור מהנדסת, בפעם הראשונה בניתי אפליקציה שימושית שכל אחד יכול להנות ממנה וזה גרם לסיפוק רב ומוטיבציה לעשות ולהמשיך.

22. פיתוחים עתידיים:

לכל תוכנה יש מקום לגדול ולהתפתח. למשל להרחיב את התוכנה לבדיקה של ספרי תורה, תפילין, מגילות.
וכן לבדיקת כשרות צורת האות.

23. ביבליוגרפיה:

- ✓ לימוד Python https://reshetech.co.il/python-tutorials/all-the-tutorials
- ✓ לימוד למידת מכונה https://reshetech.co.il/machine-learning-tutorials/all-the-tutorials
- ✓ ספריית OpenCV https://opencv.org/
- ✓ Learn Deep Learning & OpenCV https://pyimagesearch.com/
- ✓ ספריית PIL https://pillow.readthedocs.io/en/stable/#
- ✓ ספריית Keras https://keras.io/
- ✓ ספריית TensorFlow https://www.tensorflow.org/

- ✓ ספריית NumPy https://numpy.org/doc/stable/index.html
- ✓ ויקיפדיה
- ✓ Geeks For Geeks https://www.geeksforgeeks.org/
- ✓ StackOverFlow