



תיק פרויקט הגנת סייבר



מגיש: יובל מנדל (216496349) תיכון הרצוג כפר סבא יב'3

שם פרויקט: DigitNet

שם מנחה: אופיר שביט

שם חלופה: הגנת סייבר ומערכות הפעלה

תאריך הגשה: 24/05/2025

תוכן עניינים

4.....	מבוא
4.....	ייזום
4.....	תיאור כללי של המערכת
4.....	הגדרת הלקוח
5.....	מטרות ויעדים
5.....	בעיות תועלות וחסכונות
6.....	סקירת פתרונות קיימים
7.....	טכנולוגיה
7.....	תיחום
9.....	פירוט תיאור המערכת
9.....	תיאור מפורט והיכולות שיש לכל רכיב
10.....	פירוט הבדיקות (קופסא שחורה)
12.....	לוח זמנים
13.....	ניהול הסיכונים ודרכי התמודדות
14.....	תיאור תחום הידע
14.....	פירוט מעמיק של היכולות
14.....	יכולות צד שרת
16.....	יכולות צד לקוח
19.....	מבנה הפרויקט
19.....	תיאור הארכיטקטורה
19.....	תיאור הטכנולוגיה הרלוונטית
20.....	תיאור זרימת המידע במערכת
24.....	תיאור האלגוריתמים המרכזיים בפרויקט
24.....	פתרונות אפשריים
24.....	הפתרון שלי – CNN
25.....	תיאור סביבת הפיתוח
27.....	תיאור הפרוטוקול
31.....	תיאור מסכי המערכת
31.....	מסכי המערכת
36.....	תרשים מסכים
37.....	מבני נתונים
37.....	מסד נתונים
37.....	תור משימות בלקוח

38.....	הורדת תמונות צד לקוח
38.....	סקירת חולשות ואיומים
41.....	מימוש הפרויקט
41.....	סקירת המודולים
41.....	מודולים חיצוניים
42.....	מודולים ומחלקות פנימיות
45.....	ImagesORM (img_db_orm.py)
58.....	הסבר אלגוריתמים
58.....	Backward של שכבת קונבולוציה
60.....	רשתות נוירונים Gradient Descent
63.....	מסמך בדיקות
63.....	הבדיקות שתכננתי בשלב האפיון
64.....	הבדיקות נוספות שביצעתי
65.....	מדריך למשתמש
65.....	כלל קבצי המערכת
68.....	התקנת המערכת
68.....	פירוט הסביבה הנדרשת
68.....	מיקומי הקבצים
68.....	משתמשי המערכת
68.....	שימוש בלקוח
71.....	שימוש בשרת
72.....	רפלקציה
73.....	בבליוגרפיה
74.....	נספחים

מבוא

ייזום

תיאור כללי של המערכת

בפרויקט זה הינו מערכת שיכולה לזהות ספרות (0–9) מתוך תמונות, תוך שימוש בטכניקות של למידת מכונה, ובעיקר רשת נוירונים קונבולוציונית (CNN).

רשת הנוירונים הקונבולוציונית תמומש מבלי ספריות כמו tensorflow | pytorch אלא רק באמצעות numpy.

המערכת תכלול ממשק גרפי פשוט ונוח, שבו משתמשים יוכלו להעלות תמונות שכוללות ספרות, ולקבל את תוצאת הזיהוי בצורה של טקסט. בנוסף, תהיה אפשרות לצפות בתמונות שהועלו על ידי משתמשים רשומים, לראות איזו ספרה זוהתה בכל תמונה, להוריד את התמונות ששמורות בשרת ולהירשם/להתחבר (login/sign up).

התמונות והמידע הנלווה (כמו תוצאת הזיהוי) שנשלחו על ידי משתמשים שעברו התחברות (login) יישמרו במסד נתונים, כך שיהיה אפשר לשמור היסטוריה של הפעולות שהתבצעו במערכת. המערכת תשלב בין צד לקוח (GUI) שבו המשתמש פועל, לבין צד שרת שמטפל באימון המודל, בביצוע הזיהוי בפועל, ובניהול מסד הנתונים.

בחרתי בפרויקט הזה כי תחום הבינה המלאכותית מעניין אותי מאוד, במיוחד נושא של רשתות נוירונים ולמידה עמוקה. אני רוצה להבין איך רשת נוירונים מצליחה ללמוד לזהות תבניות מתוך מידע חזותי, ואיך אפשר ליישם את זה בפועל על בעיה אמיתית כמו זיהוי ספרות מתמונה. בנוסף, אני רואה בפרויקט הזה הזדמנות לשלב בין תיאוריה ומימוש מעשי, וללמוד על כל התהליך – החל מאיסוף הנתונים והכנתם, דרך אימון המודל, ועד לשילובו במערכת שלמה שפועלת מול משתמשים.

האתגרים שאני צופה לי בפרויקט הם אימון מדויק של מודל הבינה המלאכותית כך שהזיהוי יהיה אמין והבנה ומימוש של רשתות נוירונים קונבולוציונית.

הגדרת הלקוח

המערכת מיועדת לכל אדם שמעוניין לזהות ספרות מתוך תמונות בצורה מהירה, פשוטה ונגישה. היא מתאימה לשימושים חינוכיים, ניסיוניים או יישומיים – למשל לצורך זיהוי של ספרות שנכתבו ביד. קהל היעד כולל תלמידים, מורים, חוקרים ומפתחים המעוניינים לבדוק מודלים של למידת מכונה או להשתמש בתוצרי הזיהוי לצורכי ניסוי ולמידה.

המערכת מאפשרת למשתמשים להעלות תמונות, לקבל את תוצאת הזיהוי באופן מיידי, ולצפות בהיסטוריית הזיהויים שבוצעו בעבר. הממשק תוכנן להיות נוח לשימוש גם עבור מי שאין לו רקע טכני, והמערכת שמה דגש על חוויית משתמש פשוטה וברורה.

יתרון מרכזי של המערכת הוא באפשרות לאגור את התמונות שזוהו יחד עם תוצאות הזיהוי. מאגר זה משמש לא רק לתיעוד, אלא גם ככלי עזר ללמידת מכונה – ניתן לעשות בו שימוש חוזר לצורך אימון ושיפור של מודלים אחרים, מה שהופך את המערכת לכלי עבור תהליכים מתקדמים של ניתוח ולמידה.

מטרות ויעדים

המטרות שלי הן לפתח מערכת שמסוגלת לזהות ספרות מתוך תמונות באופן אוטומטי בעזרת למידת מכונה. לממש ולהבין רשת נירונים קונבולוציונית. לצבור ידע וניסיון מעשי בתחום של בינה מלאכותית וזיהוי תמונה.

בעיות תועלות וחסכוניות

הפרויקט עוסק בזיהוי אוטומטי של ספרות מתוך תמונות בעזרת בינה מלאכותית. מטרת המערכת היא לבצע את הזיהוי בצורה מהירה, מדויקת ויעילה, גם בסביבות עתירות נתונים ובתנאי קלט לא אחידים – כגון תמונות באיכות משתנה, עם רעש חזותי או כתב יד לא אחיד.

בנוסף לפונקציית הזיהוי הבסיסית, המערכת כוללת רכיב חשוב של אגירת מידע: כל תמונה שעוברת זיהוי נשמרת באופן שיטתי במאגר פנימי יחד עם תוצאת הסיווג ופרטים נוספים. המאגר הזה אינו רק אמצעי שמירה, אלא משמש תשתית ללמידה מתקדמת – ניתן לאמן על בסיסו מודלים חדשים או לשפר את המודל הקיים באמצעות הנתונים שהמערכת עצמה צברה.

באופן זה, המערכת לא רק מזהה ספרות מתמונה, אלא גם בונה בסיס נתונים שיכול לעזור באימון של מודלים אחרים בעתיד.

סקירת פתרונות קיימים

הפרויקט שלי מתמקד בלפתור 2 בעיות, זיהוי ספרה ובניית מאגר תמונות יש להן שיוך למספר המתאים.


יש הרבה מוצרים שמסוגלים לזהות ספרות ואפילו מספרים מילים ומשפטים.

לדוגמה:

[/https://www.newocr.com](https://www.newocr.com)

מערכת ocr שמסוגלת לזהות טקסט בתמונה (כולל ספרות)

האתר הזה נותן יותר יכולת מהפרויקט שלי (מסוגל טקסט בכללי).

תמונה	תוצאה של הפתרון הקיים	תוצאה שלי
	5	3
	SL	7
	(0	0

האתר אינטרנט נתן תוצאות לא נכונות.

הבדל נוסף בין הפתרון הזה לבין הפתרון שלי הוא שהפתרון הזה הוא עמוד web ובעוד שלי הוא שרת ולקוח עד פרוטוקול יחודי. בנוסף לכך האתר הזה אינו מאפשר להסתכל על תוצאות של אחרים.

הבעיה השנייה שאוצתה אני פותר היא מאגר מידע, המאגר מידע של ספרות הכי פופולרי הוא MNIST שבו אני משתמש כדי לאמן את המודל שלי. בעוד שמאגר MNIST כולל תמונות סינתטיות

בפורמט אחיד – ספרות ממורכזות, ללא רעש, בעובי וקונטרסט קבועים – המאגר שאני בונה מורכב מתמונות קלט אמיתיות כפי שהועלו על ידי המשתמשים. לכן, הוא מייצג בצורה נאמנה יותר אתגרים מהעולם האמיתי ויכול לשמש ככלי עזר לmnist באימון של מודלים אחרים. (יתרון של mnist הוא שהוא מקוטלג על ידי בני אדם ולכן הסיכוי שיהיה טעות בזיהוי הוא אפסי לעומת המאגר שלי)

טכנולוגיה

הטכנולוגיה שעליה מבוסס הפרויקט אינה חדשה, היא עושה שימוש ברשתות קונבולוציה- סוג של אלגוריתם בלמידת מכונה שמיועד במיוחד להבנה של תמונות. הרשת "לומדת" לזהות דפוסים וצורות חוזרות בתמונה, כמו קווים, עיקולים או מבנה של ספרות, בדיוק כפי שעין אנושית שמה לב לפרטים. בזכות היכולת הזו, רשתות קונבולוציה מתאימות במיוחד למשימות של זיהוי חזותי.

מכיוון שאני מממש את רשת ה־ CNN מהיסוד וללא שימוש בספריות מתקדמות או מודלים מוכנים מראש, תהליך האימון פועל גבי מעבד (CPU) רגיל ולא על כרטיס גרפי (GPU) מה שמגביל את היכולת לבצע אימונים כבדים או להשתמש במודלים גדולים. בנוסף, מאחר שאין ברשותי חומרה חזקה במיוחד, נדרש איזון בין איכות המודל לזמן הריצה והיעילות. שני הגורמים האלו מגבילים את היכולת שלי לאמן מודלים גדולים וחזקים ופוגעים באיכות הסופית של המוצר. כדי להתגבר על המכשולים האלו ניסיתי לממש את הרשת בצורה יעילה.

תיחום

הפרויקט עוסק במגוון של תחומים טכנולוגיים:

- למידת מכונה: הפרויקט מתמקד בזיהוי ספרות מתוך תמונות באמצעות רשתות נוירונים קונבולוציוניות תוך מימוש עצמי של תהליך האימון והחישוב, ללא שימוש במודלים מוכנים מראש.
- רשתות תקשורת: המערכת כוללת תקשורת בין לקוח לשרת באמצעות פרוטוקול TCP (בשביל אמינות בהעברת המידע) וממשת פרוטוקול ייחודי להעברת בקשות וקבצים בצורה אמינה ויעילה. התקשורת תומכת בהעברת מידע בינארי (תמונות) ובניהול תהליך שליחה וקבלה בשכבת האפליקציה.
- הצפנה ואבטחת מידע: כל התקשורת בין הלקוח לשרת מוצפנת באמצעות AES (הצפנה סימטרית), כאשר מפתח ההצפנה מועבר בצורה מאובטחת באמצעות אלגוריתם RSA (הצפנה אסימטרית). שילוב זה מאפשר הגנה מלאה על המידע לאורך כל שלבי ההעברה.
- מערכות הפעלה: הפרויקט כולל שימוש ב- Threads לניהול תהליכים מקבילים בשרת ובלקוח, וכן עבודה מול מערכת הקבצים (קריאה, כתיבה ושמירה של תמונות ונתונים), כחלק מהאינטגרציה עם מערכת ההפעלה.

הפרויקט אינו עוסק ביצירת מערכת הרשאות או ממשק משתמשים עם שם משתמש וסיסמה. אין בו מנגנון התחברות, שמירת סיסמאות או ניהול משתמשים, והוא לא כולל אימות זהות של הלקוח.

פירוט תיאור המערכת

תיאור מפורט והיכולות שיש לכל רכיב

המערכת נועדה לזהות ספרות (0–9) מתוך תמונות המועלות על ידי המשתמשים, ולהציג את תוצאת הזיהוי בצורה נגישה, מהירה וברורה. היא מבוססת על רשת נוירונים קונבולוציונית (CNN) שזו שיטה מתקדמת בלמידת מכונה המתמחה בניתוח מידע חזותי. רשת זו מסוגלת לזהות תבניות חזותיות מורכבות מתוך תמונה, גם כאשר הקלט אינו אחיד – כלומר, גם כאשר הספרה אינה ממורכזת, הקווים דקים או עבים מהרגיל, קיימים רעשים בתמונה, או שהתמונה צולמה או נסרקה באיכות משתנה.

אחד היעדים המרכזיים של המערכת, מעבר לזיהוי המדויק עצמו, הוא בניית מאגר תמונות ותיגוים שנוצרו מתוך קלטים אמיתיים של משתמשים. מאגר זה מהווה תשתית חשובה לאימון עתידי של מודלים נוספים, כך שהמערכת לא רק מזהה אלא גם **לומדת ומשתפרת** לאורך זמן. בכך, נוצרת פלטפורמה שמבוססת על נתוני אמת ולא רק על מאגר סטטי כמו MNIST, ומאפשרת גמישות והתאמה לשימושים רחבים יותר.

המערכת מחולקת לשני חלקים עיקריים:

1. צד לקוח:

חלק זה של המערכת הוא הממשק הגרפי שמולו פועל המשתמש. הוא נועד להיות פשוט, ברור ונגיש גם למי שאין לו רקע טכנולוגי קודם. הממשק מאפשר למשתמש להעלות קבצי תמונה מהמחשב האישי או מכל מקור אחר. לאחר ההעלאה, התמונה נשלחת לשרת, ותוצאת הזיהוי (הספרה שזוהתה) מוצגת מיד בצורה ברורה.

בנוסף לכך, הממשק כולל אפשרות לעיין בתמונות קודמות שעברו זיהוי – כולל תמונות שהועלו על ידי משתמשים אחרים. כל תמונה מוצגת יחד עם הספרה שזוהתה בה, וניתן גם להוריד את התמונות למחשב המקומי. תכונה זו מאפשרת למשתמשים לצפות בנתונים שכבר עברו דרך המערכת, לבחון את איכות הזיהוי ולהשתמש בתמונות לצרכים לימודיים או ניסיוניים.

יכולות: בקשה מהשרת לזהות תמונה, העלאת תמונה, התחברות והרשמה למשתמש (log in / sign in) הצגת תוצאה של השרת, הצגת/הורדת תמונות ותוצאות שנאגרו בשרת.

2. צד שרת:

חלק זה אחראי על עיבוד הנתונים. השרת מקבל את התמונות שנשלחות מהלקוח, מפענח את המידע ומכין אותו לקראת ניתוח. לאחר מכן, התמונה מועברת למודל ה-CNN שמבצע את תהליך החיזוי כלומר, קובע איזו ספרה מופיעה בתמונה שהתקבלה.

התוצאה מוחזרת ללקוח לצורך הצגה בממשק, אך גם נשמרת בצד השרת לצורך תיעוד וניתוח עתידי. כל תמונה מזוהה, יחד עם תוצאת הזיהוי שלה ופרטי עזר נוספים, נשמרים במסד נתונים פנימי. בכך המערכת בונה לעצמה באופן שוטף מאגר של דוגמאות מתויגות – בסיס נתונים חשוב שיכול לשמש לאימון נוסף, סטטיסטיקות, או מחקר עתידי בתחום הזיהוי החזותי.

יכולות: זיהוי ספרה בתמונה, שמירת תוצאות ותמונות, תקשורת, שמירה של לקוחות ומתן שירות ללקוח

פירוט הבדיקות (קופסא שחורה)

1. בדיקת העלאת תמונה תקינה

- מטרה: לבדוק האם המערכת מזהה תמונה שמכילה ספרה בודדת באופן תקין.
- אופן ביצוע: המשתמש יעלה תמונה איכותית, ברורה, של ספרה אחת (למשל, ספרה 3 כתובה בכתב יד ברור).
- תוצאה צפויה: המערכת תזהה את הספרה בצורה נכונה, והתוצאה תוצג בממשק ותישמר במסד הנתונים.

2. בדיקת העלאת תמונה באיכות נמוכה / עם רעש

- מטרה: לבדוק את עמידות המערכת לתמונות פחות איכותיות – לדוגמה, עם רעש, טשטוש קל או כתב יד לא ברור.
- אופן ביצוע: העלאת תמונה שצולמה מטלפון נייד בתאורה חלשה או עם רעש מלאכותי שנוסף לקובץ.
- תוצאה צפויה: המערכת תצליח לזהות את הספרה למרות הירידה באיכות.

3. בדיקת העלאת קובץ שאינו תמונה

- מטרה: לבדוק שהמערכת יודעת להתמודד עם קלט שגוי.
- אופן ביצוע: ניסיון להעלות קובץ מסוג PDF, קובץ טקסט או קובץ אקראי שאינו תמונה.
- תוצאה צפויה: המערכת תסרב לקבל את הקובץ ותציג הודעת שגיאה מתאימה (למשל: "סוג קובץ לא נתמך").

4. בדיקת תצוגת ההיסטוריה

- מטרה: לוודא שניתן לצפות בתמונות שזוהו בעבר יחד עם תוצאת הזיהוי שלהן.
- אופן ביצוע: לאחר מספר העלאות, המשתמש ילחץ על כפתור צפה בתוצאות קודמות.

- תוצאה צפויה: תוצג רשימה של תמונות קודמות עם תוצאת הזיהוי שלהן, כולל אפשרות להוריד כל אחת מהן.
- 5. בדיקת תגובתיות הממשק
- מטרה: לבדוק שהממשק הגרפי (GUI) מגיב בצורה חלקה לכל פעולה – העלאה, הצגה, מעבר בין כפתורים.
- אופן ביצוע: ביצוע רצף של פעולות משתמש – העלאה, מעבר להיסטוריה, חזרה למסך ראשי וכו'.
- תוצאה צפויה: כל פעולה תתבצע באופן מיידי ללא תקיעות או קריסות של הממשק.
- 6. בדיקת תקשורת עם השרת
- מטרה: לבדוק האם התקשורת בין הלקוח לשרת תקינה, והאם מתבצע תהליך שליחה וקבלה תקני.
- אופן ביצוע: ניתוק יזום של החיבור לשרת, או הרצת הלקוח כשהשרת כבוי.
- תוצאה צפויה: המערכת תציג הודעה על כשל בחיבור ולא תקרוס.
- 7. בדיקת אחסון במסד הנתונים
- מטרה: לבדוק שהמידע – תמונה, תוצאה וכו' – אכן נשמרים במאגר הנתונים לאחר כל זיהוי.
- אופן ביצוע: ביצוע זיהוי ולאחר מכן בדיקה (לוגית או תצוגתית) של הופעת הנתון החדש ברשימת ההיסטוריה.
- תוצאה צפויה: הנתון החדש יופיע עם כל הפרטים במיקום הנכון.

לוח זמנים

משימה	תכנון	בפועל
חקר והבנה של תהליך למידת מכונה. להבין את השלבים בלמידה ובעיבוד של רשתות נוירונים Fully connected	23.11.24-1.1.25	12.12.24-20.1.25
מימוש של רשת Fully connected	2.1.25-15.2.25	21.1.25 – 24.1.25
להכין את השרת/לקוח(ממשק גרפי) + חיבור למסד נתונים	20.2.25 – 3.4.25	20.2.25 – 3.4.25
הבנה של רשת קונבולוציונית	3.4.25 – 15.4.25	3.4.25 – 20.4.25
מימוש של רשת קונבולוציונית	15.4.25 – 30.4.25	20.4.25 – 2.5.25
הוספה של פיצ'רים נוספים/שיפור של המערכת	1.5.25-10.5.25	2.5.25 – 3.5.25

ניהול הסיכונים ודרכי התמודדות

בעיה	דרך מוצעת להתמודדות	מה שנעשה בפועל
עיבוד תמונות ברזולוציות שונות	Resize לתמונה טרם העיבוד	Resize לתמונה טרם העיבוד
תפיסה של מקום גדול באחסון של השרת בגלל השמירה של תוצאות עבר	לשמור רק מספר קבוע של תמונות אחרונות	לשמור רק מספר קבוע של תמונות אחרונות
אי זיהוי תווים כתוצאה מודל לא מאומן מספיק	הרחבה של סט הנתונים	שינוי המבנה של המודל, אימון ארוך יותר והוספת מוטציות לתמונות על מנת לשפר generalization
קבצים גדולים מאוד עולים לקחת הרבה מקום באחסון של השרת	לשמור את התמונות אחרי resize	לשמור את התמונות אחרי resize
משתמש מעלה אותה תמונה מספר פעמים	מניעת כפילויות במסד הנתונים על ידי hash של התוכן	מניעת כפילויות במסד הנתונים על ידי hash של התוכן
קריסת הלקוח כשהשרת קורס	לסגור את הלקוח במקרה שהשרת קורס	להפריד בין guin לתקשורת ובמקרה של התנתקות מהשרת לנסות להתחבר בחזרה כל כמה שניות

תיאור תחום הידע

פירוט מעמיק של היכולות

יכולות צד שרת

1. קבלת תמונות מהלקוח זיהוי התמונה ושליחת תוצאות בצורה מאובטחת-
לאפשר ללקוח לשלוח תמונה לשרת ולקבל תוצאת זיהוי – בצורה אמינה ומוצפנת.
פעולות/יכולות שנדרשות:
 - הלקוח יתחבר לשרת באמצעות פרוטוקול TCP.
 - שרת יקבל את התמונה (בבייטים).
 - השרת יזהה את הספרה שבתמונה.
 - לאחר זיהוי – שליחת התוצאה (ספרה + רמת ביטחון) חזרה ללקוח.
 - כל זה יתבצע תוך הצפנה מאובטחת של המידע.
 אובייקטים/מודולים נחוצים:
 - TCP Socket
 - הצפנה סימטרית (AES)
 - החלפת מפתחות (RSA)
 - פרוטוקול תקשורת אפליקטיבי
 - encryptor/decryptor
 - מודל לזיהוי ספרות מאומן
2. עיבוד תמונה לזיהוי ספרה באמצעות מודל CNN-
עיבוד של תמונה שהתקבלה מהלקוח וביצוע זיהוי של הספרה שבתוכה באמצעות רשת נוירונים קונבולוציונית מאומנת.
פעולות/יכולות שנדרשות:
 - המרת התמונה לפורמט אחיד (שחור-לבן, גודל קבוע, נירמול).
 - המרת התמונה למערך מספרי שניתן להזין למודל.
 - הפעלת המודל המאומן לקבלת תוצאה – הספרה שזוהתה.
 - שמירה על מהירות תגובה סבירה לביצוע בזמן אמת.
 אובייקטים/מודולים נחוצים:
 - מודל CNN יעיל מאומן לזיהוי ספרות (לממש, maxpooling, dense layer, conv layer ועוד)
 - פונקציות preprocessing שמממירות את התמונה לפורמט מתאים ועושות שינויים כדי לשפר את אחוזי ההצלחה בזיהוי

3. שמירת תוצאות במסד נתונים-

שמירה של כל תוצאה של זיהוי במסד נתונים, כולל מידע כמו הספרה שזוהתה, רמת הביטחון, והקובץ עצמו, על מנת לאפשר שליפה עתידית ובקרה.

פעולות/יכולות שנדרשות:

- יצירת רשומה חדשה במסד הנתונים עבור כל תמונה.
- שמירה של מזהה ייחודי (UUID), תוצאה מזוהה, אחוז ביטחון, ונתיב לקובץ
- שמירה של קובץ התמונה עצמו במערכת הקבצים.
- הגבלת כמות התמונות השמורות (למשל ל-100 אחרונות בלבד).
- מניעת כפילויות באמצעות hash של תוכן התמונה.
- מחיקת קבצים ישנים שאינם קיימים יותר במסד הנתונים
- שליפה של תמונות לפי דרישות – לדוגמה לפי ספרה מסוימת.

אובייקטים/מודולים נחוצים:

- מסד נתונים (SQLite)
- מחלקה לניהול מסד הנתונים
- פונקציות לניהול קבצים (read, open, write, delete)
- מחולל hash (SHA-256) למניעת כפילויות.
- שאילתות SQL לשליפה, עדכון ומחיקה.

4. אימון מודל CNN לזיהוי ספרות-

לבנות ולאמן רשת נוירונים קונבולוציונית (CNN) מאפס, כך שתלמד לזהות ספרות מתוך תמונות באופן מדויק. האימון מתבצע על גבי מאגר נתונים הכולל תמונות מתווגות.

פעולות/יכולות שנדרשות:

- בניית ארכיטקטורה של רשת
- בניית כל החלקים שמרכיבים את הרשת (dense conv2d flatten maxpool)
- הגדרת פונקציית עלות (loss) מתאימה.
- ביצוע forward pass לקבלת חיזוי.
- חישוב backward pass (גרדיאנטים לפי השרשרת) לעדכון המשקלים.
- שימוש באלגוריתם אופטימיזציה (לדוגמה SGD)
- חלוקה ל- epochs ומעקב אחר שיפור באחוזי הדיוק.
- אפשרות לבצע augmentations (עיוותים אקראיים) בתמונות לשיפור היכולת להכליל.
- שמירה של המודל המאומן לשימוש עתידי.

אובייקטים/מודולים נחוצים:

- מחלקת CNN הכוללת את כל השכבות והפונקציות הדרושות.
- מחלקת optimizer
- נתונים לאימון (mnist)
- פונקציות serialization כדי לשמור את האימון שנעשה
- פונקציות שעושות עיוותים למידע המתקבל

5. ניהול משתמשים – הרשמה והתחברות
הוספת מנגנון הרשמה והתחברות של משתמשים למערכת, לצורך התאמה אישית, הפרדה בין משתמשים, שמירה על פרטיות, ושיוך תמונות למשתמשים רשומים בלבד.

פעולות/יכולות שנדרשות:

- אימות זהות המשתמש עם התחברות מבוססת שם משתמש וסיסמה.
- אפשרות יצירת חשבון חדש במערכת (רישום).
- שמירת מידע משתמשים (כולל סיסמה מוצפנת).
- בדיקת תקינות סיסמה בעת התחברות.
- קישור תוצאות זיהוי למשתמש שנכנס למערכת.
- חסימת פעולות מסוימות (כמו צפייה בהיסטוריה) למשתמשים לא מחוברים.
- שמירה על session פעיל לאחר התחברות.

אובייקטים/מודולים נחוצים:

- מסד נתונים וטבלת users ב SQLite
- פונקציית hash לסיסמאות (SHA-256) או (bcrypt)
- מערכת session לשיוך לקוח מחובר
- שכבת הרשאות בסיסית לפעולות מוגנות

יכולות צד לקוח

1. יצירת חיבור מאובטח לשרת-

התחברות לשרת בצורה מאובטחת לצורך שליחה וקבלה של מידע רגיש (תמונות ותוצאות זיהוי).

פעולות/יכולות שנדרשות:

- פתיחת חיבור TCP לשרת.
- קבלת מפתח ציבורי מהשרת לצורך הצפנה אסימטרית. (RSA)
- יצירת מפתח AES אקראי.
- הצפנת המפתח הסימטרי באמצעות RSA ושליחתו לשרת.
- מעבר לתקשורת מוצפנת ב AES-לכל ההודעות הבאות.

אובייקטים/מודולים נחוצים:

- TCP Socket
- RSA Encryptor
- AES Encryptor/Decryptor
- ניהול session להצפנה
- ממשק תקשורת מול השרת

2. העלאת תמונה לשרת לצורך זיהוי-
לאפשר למשתמש לבחור תמונה מהמחשב ולשלוח אותה לשרת בצורה פשוטה ומוצפנת.

פעולות/יכולות שנדרשות:

- הצגת כפתור לבחירת קובץ מהמחשב.
- קריאת קובץ התמונה וטעינתו לזיכרון.
- בנייה של ההודעה לפי הפרוטוקול
- שליחת ההודעה לשרת במבנה פרוטוקול מוגדר מראש.
- המתנה לקבלת תשובה מהשרת.

אובייקטים/מודולים נחוצים:

- File chooser
- מנגנון קריאת קבצים - read()
- AES Encryptor

3. קבלת תוצאות מהשרת והצגתן למשתמש-
להציג למשתמש את הספרה שזוהתה בתמונה ואת אחוז הדיוק של המודל, באופן ברור ונוח.

פעולות/יכולות שנדרשות:

- קבלת תגובת שרת בפורמט מוגדר (מוצפן).
- פענוח
- פירוק תוכן ההודעה – ספרה + אחוז ביטחון.
- הצגת התוצאה על גבי הממשק.

אובייקטים/מודולים נחוצים:

- AES Decryptor
- Protocol parser
- רכיבי GUI להצגת טקסט
- לוגיקת ניתוח הודעות

4. הצגת היסטוריית תמונות ותוצאות-

לאפשר למשתמש לצפות בתמונות קודמות שזוהו בעבר (את כולן וסינון לפי ספרות)
כולל תוצאה מזוהה ואפשרות להורדה.

פעולות/יכולות שנדרשות:

- שליחת בקשה לשרת לקבלת היסטוריית תמונות.
- קבלת רשימת תמונות + תוצאות.

- הצגת התמונות בתצוגת גלריה.
 - הצגת הספרה שזוהתה בכל תמונה.
 - אפשרות להוריד את התמונות
- אובייקטים/מודולים נחוצים:
- התמונות שהתקבלו מהבקשה
 - מציד תמונות תמונות (image viewer)
5. הורדת תמונות מהשרת-
לאפשר למשתמש להוריד למחשב האישי שלו את התמונות שנשלחו וזוהו בעבר.

פעולות/יכולות שנדרשות:

- הצגת כפתור "הורד" ליד כל תמונה בהיסטוריה.
 - שליחת בקשה לשרת לקבל הקבצים וקבלתם.
 - שמירה של הקובץ במחשב המקומי בפורמט מתאים.
- אובייקטים/מודולים נחוצים:
- File dialog לבחירת תיקייה
 - יוצר קבצי zip

6. התחברות והרשמה למשתמשים-

לאפשר למשתמשים להירשם למערכת ולהיכנס עם שם משתמש וסיסמה לפני ביצוע פעולות.

פעולות/יכולות שנדרשות:

- מסך כניסה ומסך רישום עם שדות שם משתמש וסיסמה.
- שליחת בקשה לשרת לצורך התחברות או רישום.
- הצגת הודעות שגיאה (למשל משתמש כבר קיים, סיסמה שגויה וכו').
- מעקב אחר סטטוס התחברות והצגתו בממשק הראשי.
- הגבלת פעולות מסוימות (למשל שליחה/צפייה בהיסטוריה) למשתמשים מחוברים בלבד.

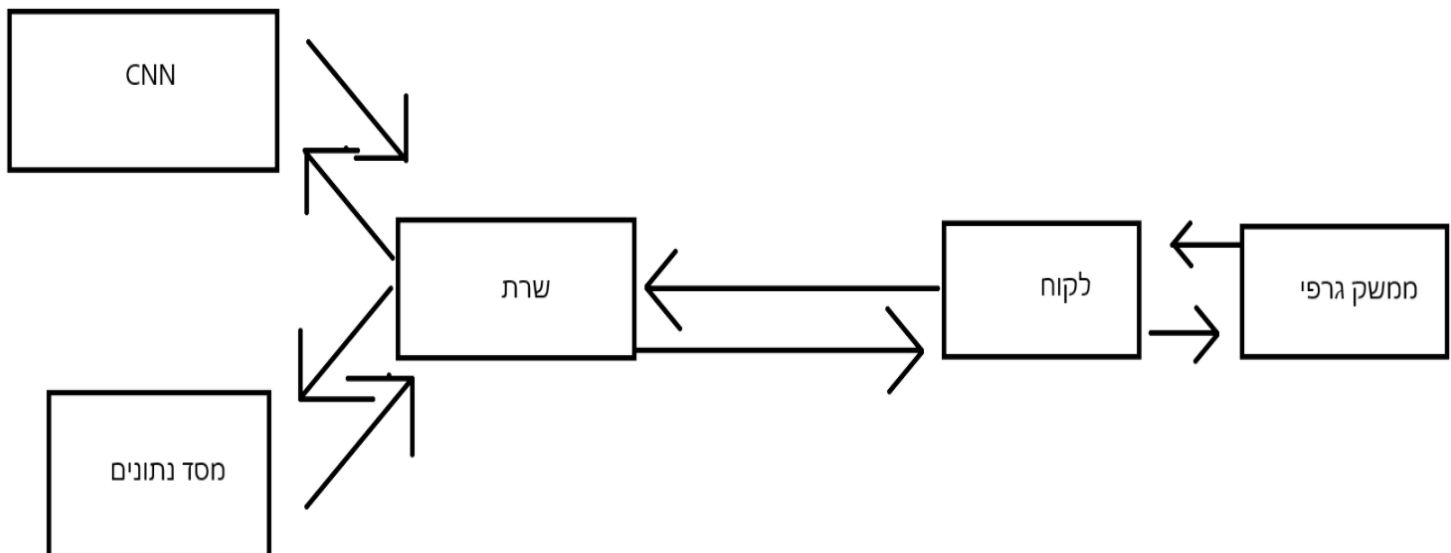
אובייקטים/מודולים נחוצים:

- רכיבי GUI לשדות קלט וכפתורים
- משתנה סטטוס התחברות (logged_in_user)
- הודעות סטטוס ועדכונים בממשק

מבנה הפרויקט

תיאור הארכיטקטורה

לפרויקט 5 רכיבים עיקריים: ממשק גרפי, לקוח, שרת, רשת נוירונים קונבולוציונית ומסד נתונים



תיאור הטכנולוגיה הרלוונטית

כל הקוד בפרויקט נכתב בשפת python (בפרט כדי לכתוב את ה- CNN השתמשתי בספריית Numpy).

בחרתי לכתוב את הפרויקט בשפת פייתון מכיוון שהיא שפה פשוטה, קריאה וגמישה, שמתאימה במיוחד לפרויקטים שמשלבים תקשורת. לפייתון יש קהילה רחבה וכלים מוכנים לעבודה עם תמונות ומערכים רב מימדיים בצורה יעילה (Numpy, PIL). היא מתאימה מאוד לפרויקטים לימודיים שמטרתם הבנה של הלוגיקה מאחורי המימוש – ולא רק שימוש בספריות מוכנות מראש.



השרת והלקוח רצים במערכת הפעלה windows 11.

התקשורת במערכת מנוהלת בפרוטוקול TCP בשביל אמינות של המידע. בנוסף התקשורת פועלת לפי פרוטוקול יעודי שהגדרתי למען הפרויקט.

הפרויקט עוסק בשילוב של תחומים טכנולוגיים מגוונים: למידת מכונה וזיהוי תבניות חזותיות בעזרת רשתות נוירונים קונבולוציוניות (CNN), הצפנה ואבטחת מידע (RSA, AES) ופיתוח מערכות מבוססות לקוח-שרת עם תקשורת אמינה בפרוטוקול TCP.

בנוסף, הוא כולל ניהול נתונים באמצעות מסד נתונים מקומי, עבודה עם קבצים ותמיכה בממשק גרפי אינטראקטיבי. תחומי עניין אלו מבטאים חיבור בין מדעי המחשב העיוניים לבין יישומים מעשיים של תוכנה ובינה מלאכותית.

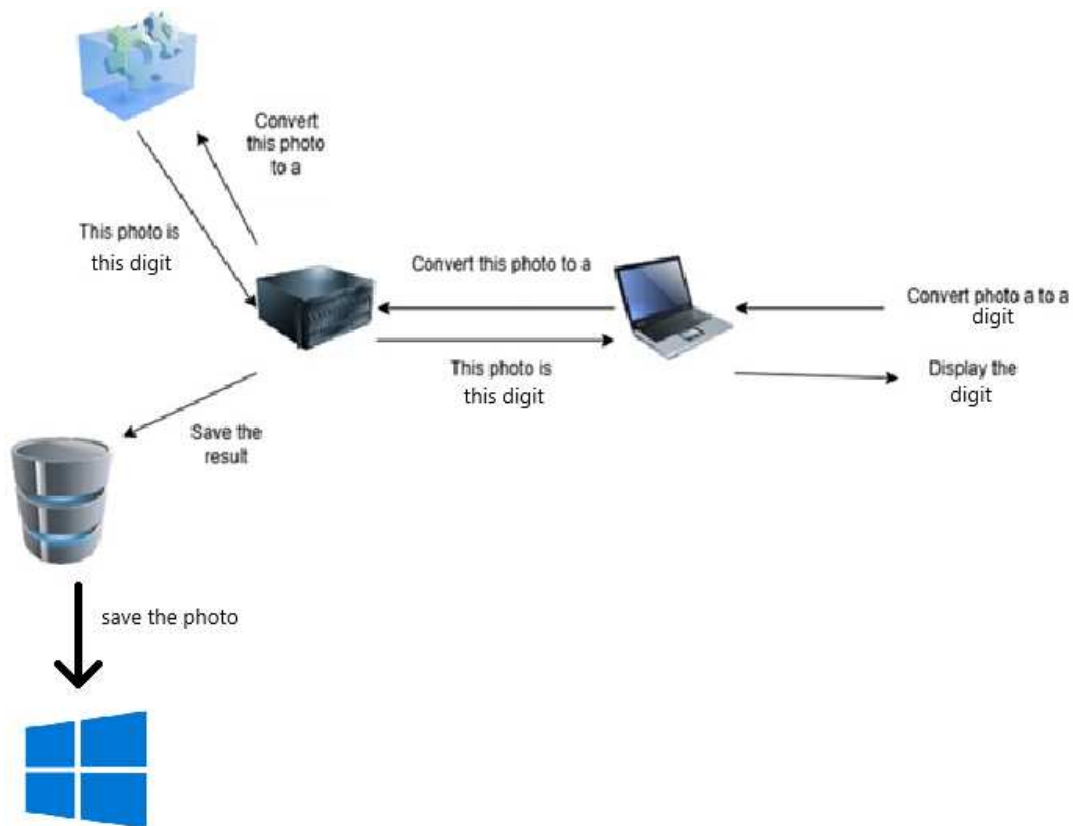
תיאור זרימת המידע במערכת

*חיצים בין הלקוח לשרת הם בקשות/תגובות ברשת בעוד שחצים אחרים הם מעבר של מידע בתוך process של השרת/לקוח

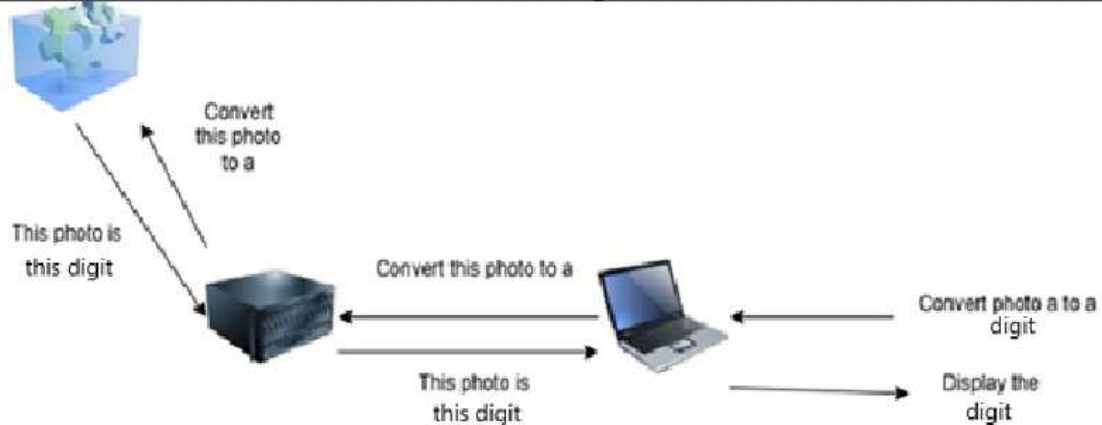
בקשה של הלקוח לפענח ספרה מסויימת:

המשתמש יזין בממשק הגרפי תמונה שהוא רוצה להמיר, הממשק הגרפי יודיע ללקוח לשלוח הודעה לשרת, הלקוח ישלח את התמונה לשרת לפיענוח, השרת ישתמש ב-CNN כדי לפענח את התמונה, אם המשתמש רשום ישמור במסד הנתונים את התוצאה (שישמור את התמונה מול מערכת ההפעלה) וישלח את התוצאה ללקוח שיציג את התוצאה בממשק הגרפי של המשתמש.

משתמש רשום:

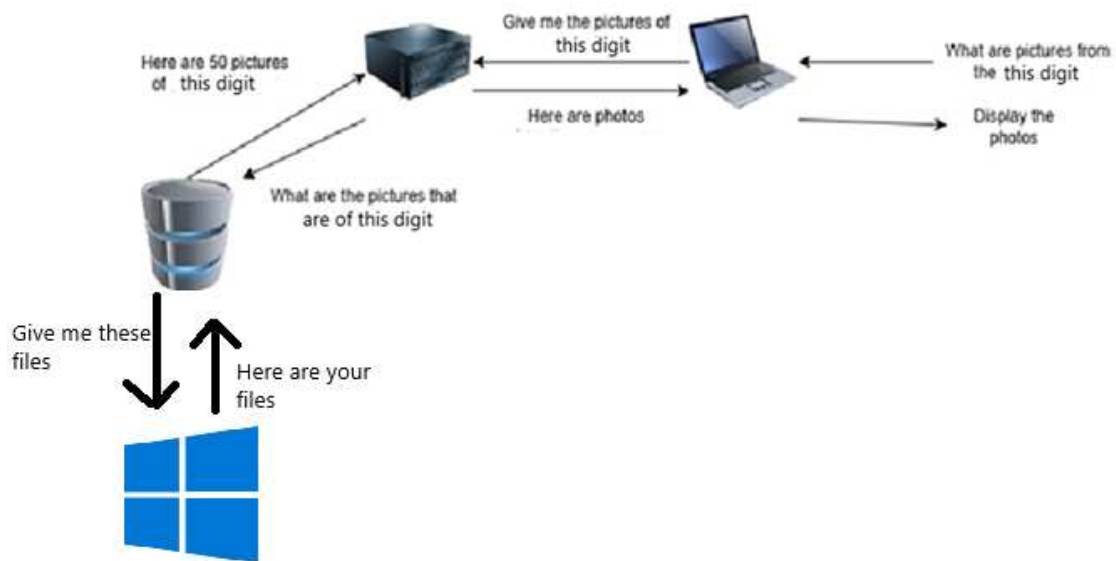


משתמש לא רשום:



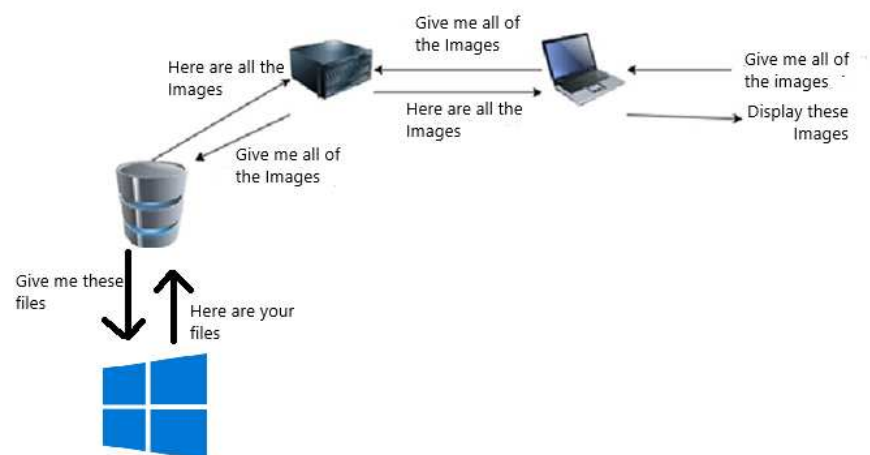
בקשה של הלקוח של תמונות של ספרה מסויימת:

המשתמש יבקש תוצאות קודמות של ספרה מסויימת דרך הממשק הגרפי, הממשק הגרפי יודיע ללקוח לשלוח הודעה לשרת, הלקוח יבקש מהשרת תשובות קודמות, השרת ישלח ממשד הנתונים את התוצאות הקודמות, השרת ישלח את התוצאות ללקוח שיציג את התוצאות בממשק הגרפי.

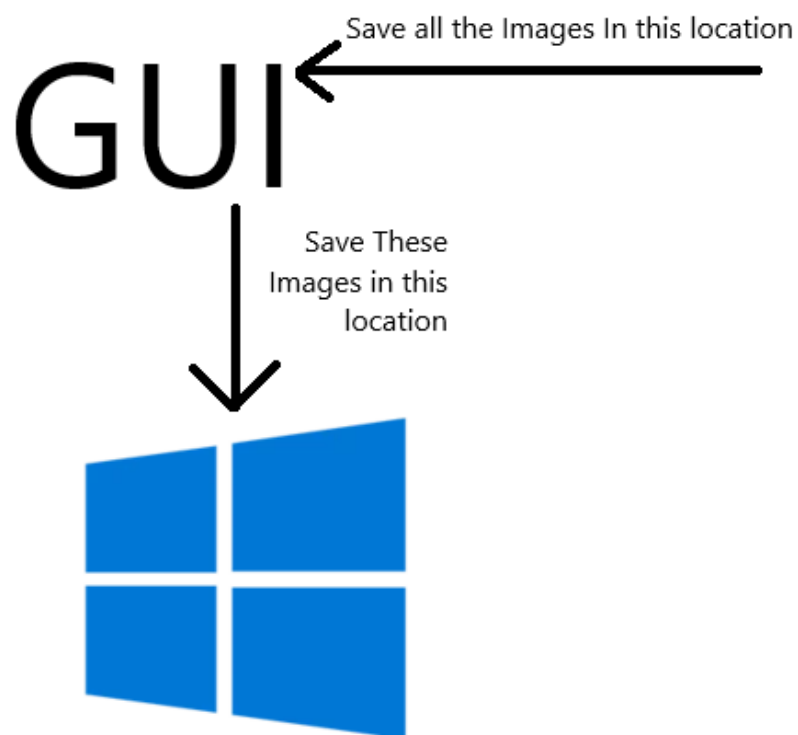


בקשה של הלקוח של כל התמונות שיש לשרת:

המשתמש יבקש את כל התמונות שיש לשרת דרך הממשק הגרפי, הממשק הגרפי יודיע ללקוח לשלוח הודעה לשרת, הלקוח יבקש מהשרת תשובות קודמות, השרת ישלוח ממסד הנתונים את התוצאות הקודמות, השרת ישלח את התוצאות ללקוח שיציג את התוצאות בממשק הגרפי.

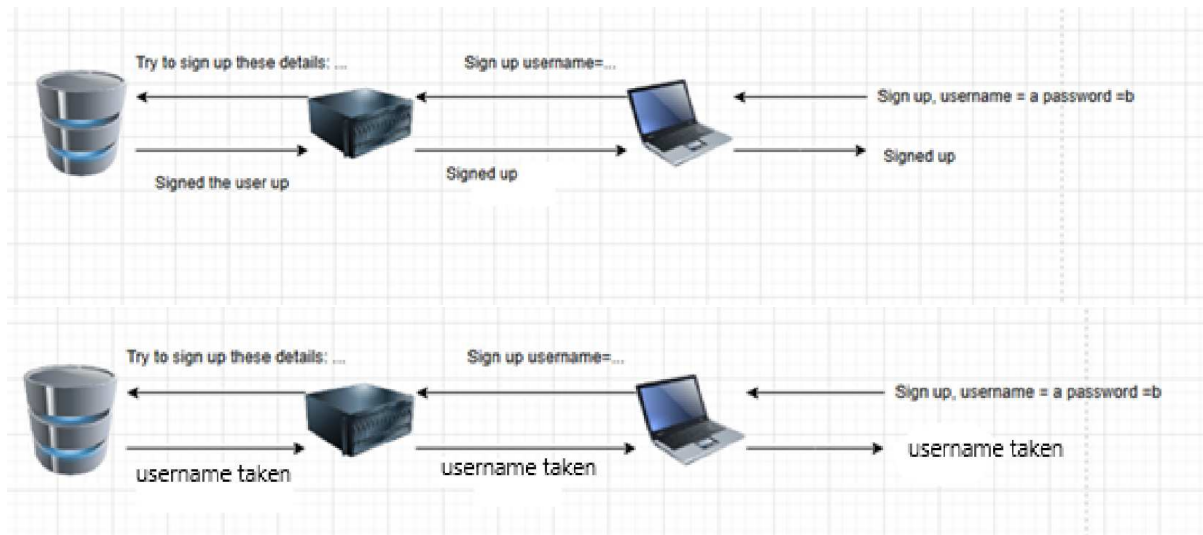


המשתמש רוצה להוריד את כל הקבצים שנשלחו לו (יכול לקרות רק אחרי שקיבל תמונות (בשני הדרכים שצויינו לעיל)



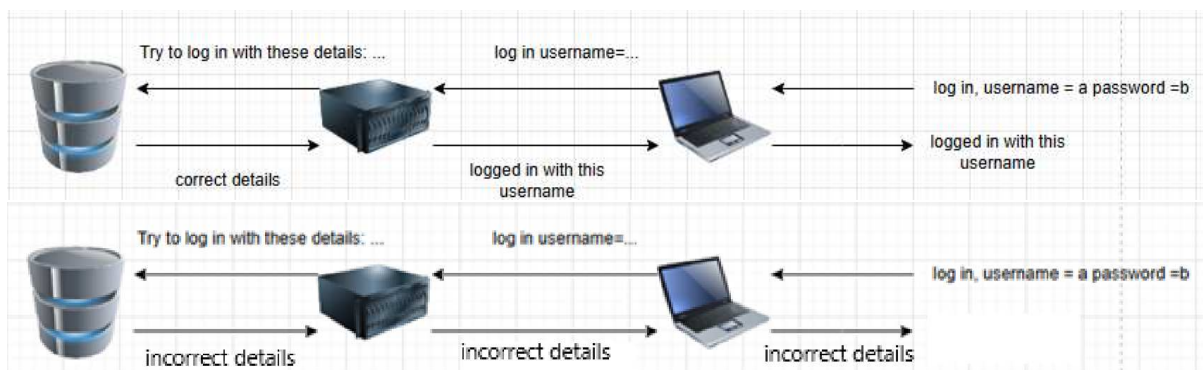
רישום של משתמש sign up-

המשתמש מכניס פרטים של משתמש, שולח לשרת בקשה, אם השם משתמש פנוי השרת שומר את המשתמש (אם לא אז הוא שולח ללקוח שההרשמה נכשלה) אם ההרשמה הושלמה אז השרת שולח שההרשמה נעשתה בהצלחה ומחזיר את השם משתמש.



התחברות של משתמש sign in-

המשתמש מכניס פרטים שולח בקשה להתחבר. אם הנתונים נכונים אז השרת מאשר שהלקוח נכנס אם לא שולח הודעה שהנתונים לא נכונים.



תיאור האלגוריתמים המרכזיים בפרויקט

זיהוי תמונות (Image Recognition) הוא תחום בלמידת מכונה ובראייה ממוחשבת, שמטרתו להבין את תוכן התמונה – לזהות עצמים, תבניות או מאפיינים בתמונה, ולהמיר מידע חזותי לייצוג סמנטי או מספרי. בפרויקט זה, המשימה ממוקדת בזיהוי **ספרות** (0–9) מתוך תמונות, לרוב בכתב יד, שהן בעלות גיוון גדול במראה: מיקום שונה בתמונה, סגנון כתיבה אישי, רעש רקע, עיוותים, שינויי תאורה ועוד.

לבני אדם, לזהות ספרה בתמונה זה דבר קל שבא בטבעיות – גם אם הספרה כתובה בעט, בגיר, בעובי שונה, בזווית, או אפילו עם טיפה רעש ברקע. מערכת הראייה האנושית יודעת להכליל, להשלים פרטים חסרים ולהשתמש בהקשר כדי להבין במהירות מה מופיע בתמונה. לעומת זאת, עבור מחשב, משימה זו מורכבת בהרבה: הוא מקבל מערך של פיקסלים מספריים חסרי משמעות סמנטית, וצריך ללמוד כיצד תכונות מקומיות בתמונה מרמזות על ספרה שלמה.

פתרונות אפשריים

לבעיה הזו כמה פתרונות אפשריים.

-Random Forest

Random Forest הוא אלגוריתם של למידת מכונה שמבוסס על ריבוי של עצי החלטה (Decision Trees) כל עץ מקבל החלטה עצמאית לגבי הסיווג, ו"היער" (forest) מצביע על התשובה הסופית לפי הרוב. העיקרון מאחורי השיטה הוא שכאשר משקללים את ההחלטות של הרבה עצים שונים, מקבלים תוצאה מדויקת ועמידה יותר לשגיאות.

מכיוון שתמונה היא מערך של פיקסלים לא כדאי להזין את הפיקסלים עצמם ל-Random Forest. זה היה סובל מרעש ומורכבות גבוהה מדי. לכן, קודם כל מחולצים מהתמונה מאפיינים, שמספקים ייצוג קומפקטי של צורת הספרה: קווים, כיוונים, קימומים וכו'.

לאחר מכן, הפיצ'רים הללו מוזנים Random Forest שמסווג את הקלט לאחת מ-10 הספרות האפשריות.

בעיה אחת בפתרון הזה היא שצריך לבחור את הפיצ'רים בעצמינו ואם הפיצ'רים לא טובים זה עלול לפגוע מאוד באיכות של המודל.

- k-Nearest Neighbors

שיטה פשוטה שמבוססת על השוואת תמונה חדשה לדוגמאות שכבר סומנו. כל תמונה נמדדת לפי "מרחק" לתמונות אחרות, והספרה הכי שכיחה מבין השכנים הקרובים היא הבחירה.

הבעיה בפתרון הזה היא שהוא איטי מאוד בזיהוי (חייב לעבור על כל הדוגמאות), רגיש לרעש, לא יעיל בזיכרון.

הפתרון שלי – CNN

מה זה CNN?

CNN (Convolutional neural network) היא רשת נוירונים מיוחדת שנועדה לעבודה עם תמונות. הרשת מקבלת את תמונת הפיקסלים עצמה, ולומדת מתוכה לבד איך לזהות תבניות. היא עושה זאת באמצעות שכבות קונבולוציה – מסננים (filters) שמזהים קווים, עיקולים, גבולות ועוד, ושלבם של pooling להקטנת המידע תוך שימור המבנה החשוב.

איך CNN מזהה ספרות?

- התמונה עוברת עיבוד מקדים (נרמול שינוי גודל וכו')
- היא מוזנת לרשת CNN, שבה שכבות לומדות תכונות – החל מקווים פשוטים בשכבות הראשונות ועד למבנה הספרה השלמה בשכבות העמוקות.
- בסוף, שכבת Fully Connected מוציאה חיזוי: מהי הספרה בתמונה ומה מידת הביטחון בזיהוי.

למה זה מתאים?

- הרשת לומדת לבד את התכונות החשובות מתוך התמונות – בלי צורך בהנדסת פיצ'רים.
- היא יודעת לזהות גם תמונות עם רעש, סיבוב, שינוי גודל או כתב לא אחיד.
- היא מסוגלת להגיע לרמות דיוק גבוהות מאוד.

יתרונות:

- דיוק גבוה מאוד.
- עמידות לשינויים בתמונה.
- מתאים במיוחד לזיהוי תבניות חזותיות.

חסרונות:

- דורש הרבה יותר כוח חישוב (במיוחד בזמן אימון).
- קשה יותר להבנה ולהסבר פנימי של תהליך ההחלטה.
- דורש מאגר נתונים איכותי ומגוון כדי להכליל היטב.

מקורות שעזרו לי ללמוד ולחקור על רשתות נוירונים:

<http://neuralnetworksanddeeplearning.com/chap1.html>

https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

תיאור סביבת הפיתוח

כלי הפיתוח הנדרשים

הפרויקט נבנה כולו בשפת Python, תוך שימוש ב IDE (PyCharm)

לצורך פיתוח המודל והמערכת נדרשו הכלים הבאים:

- שפת פיתוח: Python 3.9
- ספריות לעיבוד תמונה: PIL (Pillow), io, os
- ספריות חישוביות: NumPy
- ספריות להצפנה: Crypto.Cipher.PublicKey.RSA, Crypto.Cipher.AES
- עבודה עם מסד נתונים: sqllite3

- GUI: tkinter

- תקשורת: threading

כלים וסביבה הדרושים לבדיקה:

- מערכת הפעלה: Windows 11

- סביבת הרצה: Interpreter של Python 3.9

- הבדיקות כללו בדיקות תקשורת מקצה לקצה, (Client ↔ Server) בדיקות תקינות הצפנה, בדיקת תגובות לזיהוי שגוי ובדיקות GUI.

תיאור הפרוטוקול

מטרת הפרוטוקול היא לאפשר שירות זיהוי תמונות. הפרוטוקול המידע שעובר צריך להיות מוצפן והפרוטוקול צריך לאפשר מעבר של קבצים למען בקשות של הלקוח ולמען תגובות של השרת.

מאפייני הפרוטוקול:

- טקסטואלי – (Text-Based Protocol) הנתונים עצמם מקודדים ב־ Base64 ומופרדים באמצעות תו מפריד “~”.
- מבוסס – TCP תקשורת אמינה וסדר הודעות שמור.
- סינכרוני – הלקוח שולח בקשה וממתין לתשובה.
- מוצפן לחלוטין לאחר handshake
- שדה אורך בתחילת כל הודעה (7 תווים לא מוצפנים), כדי לאפשר קריאה רציפה של ההודעה, גם עבור קבצים גדולים.
- כדי להימנע מפתיחה של מספר קבצים בו זמנית על ידי אותו threadn בשרת כאשר השרת שולח תמונות ללקוח הוא ישלח הודעה מראש שאומרת שהוא הולך לשלוח הודעות של תמונות וכמה מהן הוא מתכנן לשלוח, אז ישלח מספר הודעות עם התמונות וכשיסיים ישלח הודעה שהוא סיים. (נשלח הודעת סיום וגם כמות ההודעות שישלחו מראש כדי להימנע ממצב שבו השרת לא מוצא תמונה ואז הלקוח נתקע בrecv)
- כל ההודעות נשלחות בצ'אנקים של עד 4K (הודעה יכולה להיות יותר גדולה מ־4K אך פשוט תפורק למספר צאנקים המקבל ידע מתי להפסיק לעשות recv לפי שדה האורך)
- *סכנה תאורטית אפשרית היא אם מישהו רוצה לשנות את 4K למספר קטן יותר מאורך שדה האורך, ואז לא כל שדה האורך נשלח בצ'אנק הראשון מה שיגרום לבעיה.

תהליך ה-handshake:

1. השרת שולח ללקוח את המפתח הציבורי שלו (RSA – PEM)
2. הלקוח יוצר מפתח AES ו־ IV אקראיים, מצפין את המפתח AES עם המפתח הציבורי של השרת ושולח אותם לשרת (2 הודעות נפרדות)
3. השרת מפענח, שומר את המפתח ואת ה־IV, ושולח הודעת (GKSC) ACK מוצפנת באמצעות AES.
4. מכאן והלאה, כל ההודעות מוצפנות ב־ AES במצב CBC.

מבנה הודעה כללי:

- שדה אורך: 7 תווים (לא מוצפן)
- גוף ההודעה: שדות BASE64 מופרדות באמצעות ~
- השדה הראשון יהיה קוד הודעה
- הפענוח מתבצע לאחר קבלת כל ההודעה המוצפנת.

סוגי בקשות: Server → Client –

RIPP- בקשה לזיהוי ספרה מתוך תמונה

פרמטרים:

1. שם הקובץ (מחרוזת)

2. תוכן התמונה (בייטים)

הגבלות:

- גודל מקסימלי של קובץ 3,000,000: בתיים (3MB)
- פורמט תמונה חייב להיות חוקי ונתמך (בדיקה עם PIL.Image.verify())
- התמונה יכולה להיות צבעונית או בשחור-לבן – המרה תתבצע בצד שרת.

RIHP- בקשה לקבל את כל התמונות

פרמטרים: אין

RIHD- בקשת שליפת תמונות לפי ספרה מסוימת

פרמטרים:

1. ספרה אחת (0–9) במחרוזת

CRSU- בקשה להירשם sign up

פרמטרים:

1. שם משתמש

2. סיסמה

CRSI- בקשה להיכנס login

פרמטרים:

1. שם משתמש

2. סיסמה

תגובות: Client → Server –

RIPR- תגובת זיהוי לתמונה

פרמטרים:

1. ספרה מזוהה (0–9)
2. אחוז ביטחון – (Confidence) מספר עשרוני בין 0 ל-1 (טקסט)

RIHL- תגובה לבקשה של תמונות, מתחיל את תהליך העברת הקבצים. לאחר ההודעה הזו השרת שולח הודעות מסוג RILF עד שמסיים את התמונות ואז שולח RIHE

פרמטרים:

- מספר התמונות המשוער (אם אחת התמונות לא נמצאה אז פשוט ישלחו פחות הודעות RIHE בכל מקרה הלקוח מפסיק כאשר מקבל RIHE)
- RIHL- הודעה ששולחת הודעה אחת ללקוח (הודעה זו נשלחת בשרשרת ולאחר השרשרת תמיד תופיע RIHE)

פרמטרים:

1. מזוהה תמונה (UUID)
2. תוכן התמונה (בייטים בפורמט PNG)
3. ספרה מזוהה
4. אחוז ביטחון

RIHE- הודעה שמסמנת את סוף שליחת התמונות מהשרת

פרמטרים- אין

CRSA- תגובה חיובית לבקשה להירשם (sign up)

פרמטרים: אין

CRSD – תגובה שלילית לבקשה להירשם (sign up)

פרמטרים: אין

CRLA – תגובה חיובית לבקשה להתחבר (login)

פרמטרים:

- שם המתמש שאיתו נרשמו

CRFD- תגובה שלילית לבקשה להתחבר (login)

פרמטרים: אין

ERRR- שגיאה

פרמטרים:

1. קוד שגיאה (מחרוזת):

- 1- קובץ אינו תמונה חוקית או בפורמט לא נתמך
- 2- קובץ גדול מדי
- 3- בקשה לא תקינה
- 4- שגיאה כללית

GKSC- אישור של השרת שהוא מוכן לתקשורת (אחרי handshake), אין פרמטרים

דוגמה לתקשורת מלאה:

1. הלקוח פותח חיבור TCP ומקבל את מפתח ה־RSA של השרת.
2. הלקוח שולח מפתח AES מוצפן ואת ה־IV.
3. השרת משיב GKSC.
4. הלקוח שולח RIPP עם שם ותוכן תמונה.
5. השרת שולח RIPP עם ספרה ורמת ביטחון.
6. הלקוח שולח RIHD עם הספרה 3.
7. השרת משיב ב־RIHL.
8. השרת שולח הודעות RILF כדי לשלוח את התמונות (הודעה עבור כל תמונה).
9. השרת שולח הודעת RIHE כדי לסמן שכל התמונות נשלחו.

פרטים נוספים:

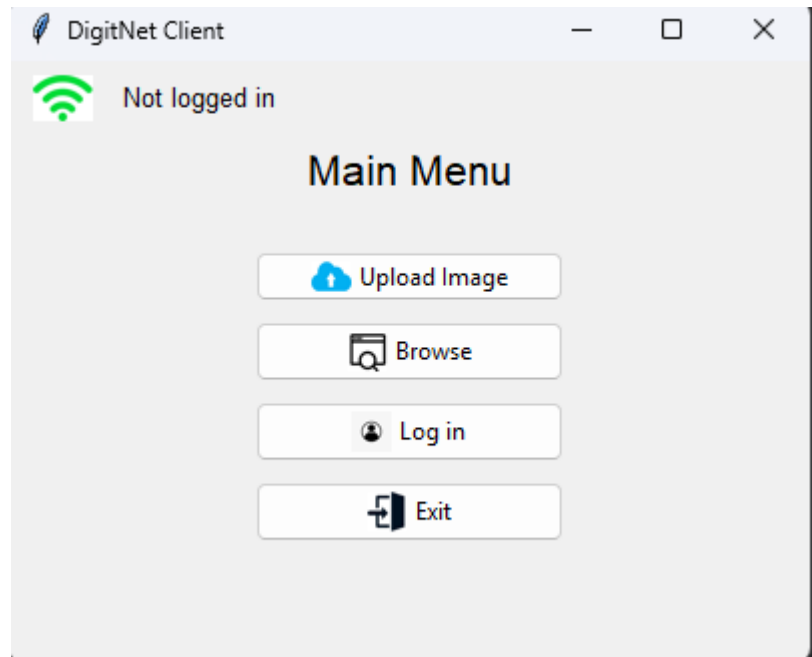
- כל ההודעות ב־AES מוצפנות במצב CBC עם padding.
- לכל לקוח Session הצפנה ייחודי משלו.
- המודל בצד השרת שומר את התמונה, מזהה ספרה, ומעדכן מסד נתונים (עד 100 תמונות).
- קבצים נשמרים בפורמט PNG בגודל מרבי של 256*256 (עוברים resize)

תיאור מסכי המערכת

מסכי המערכת

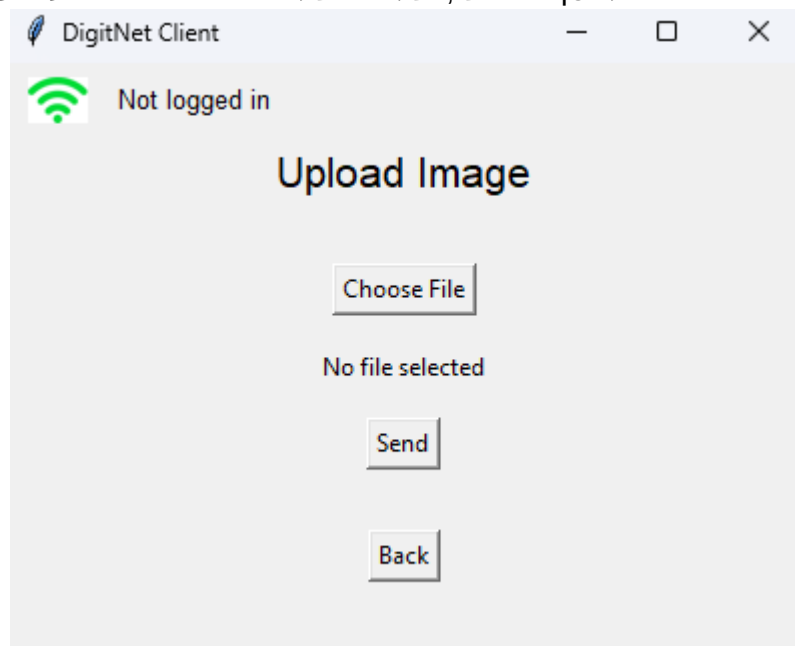
מסך ראשי-

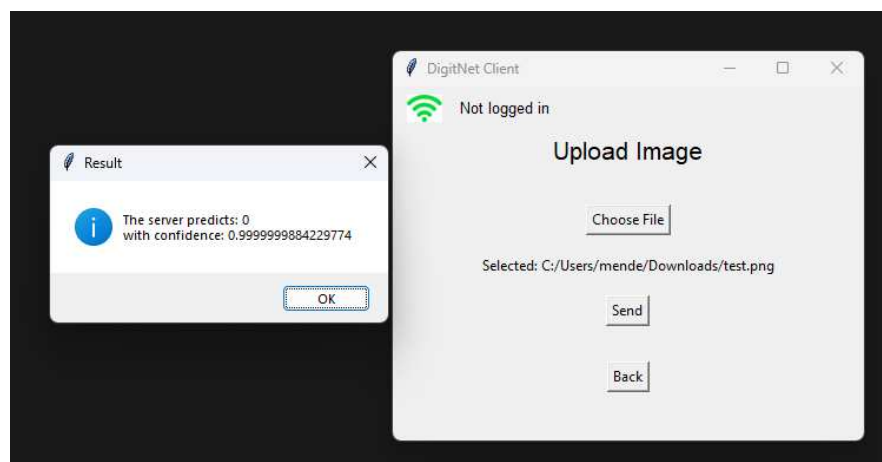
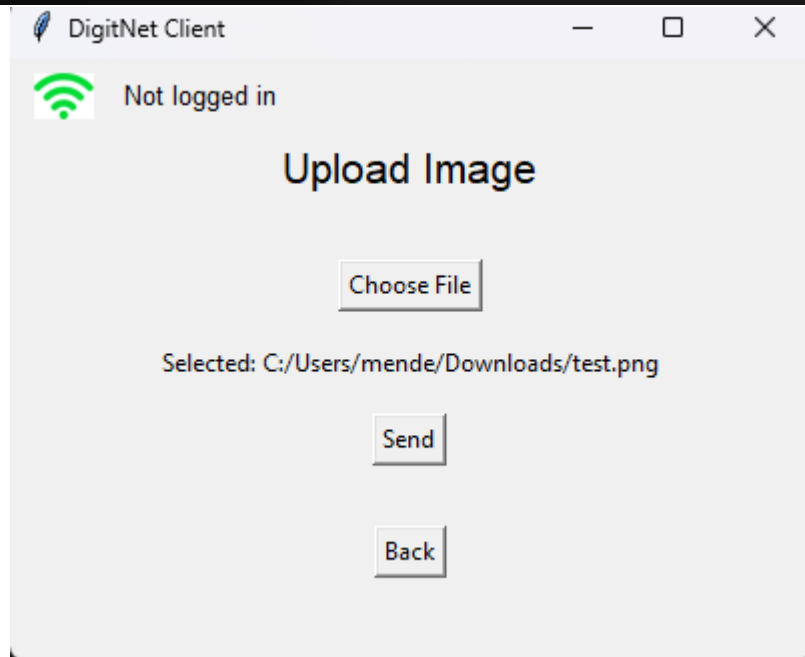
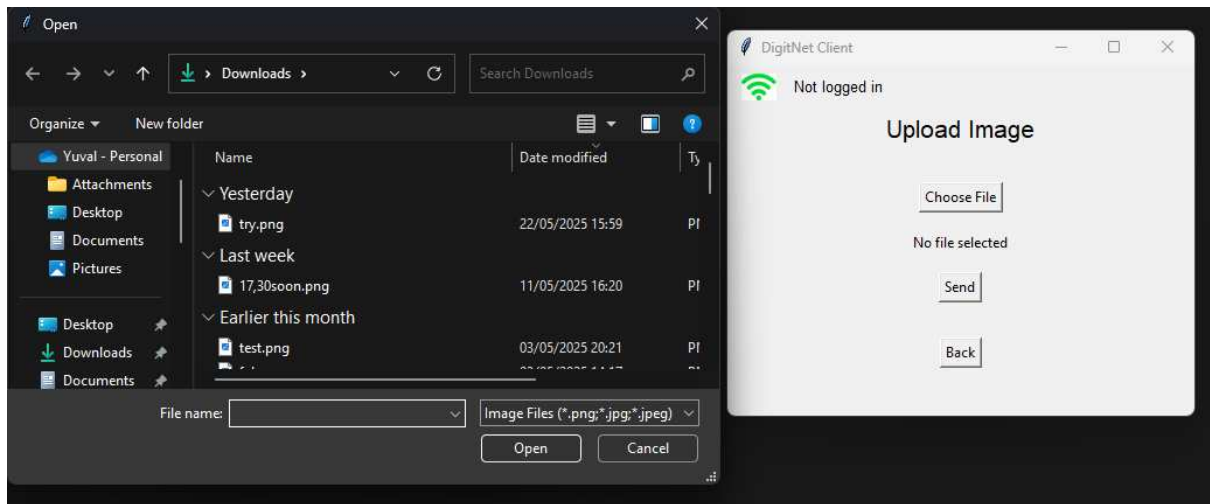
נותן לעבור למסכים האחרים (או לסגור את התוכנה), מוצג בפינה השמאלית למעלה את מצב החיבור (מחובר או מנותק מהשרת, מחובר למשתמש או לא)

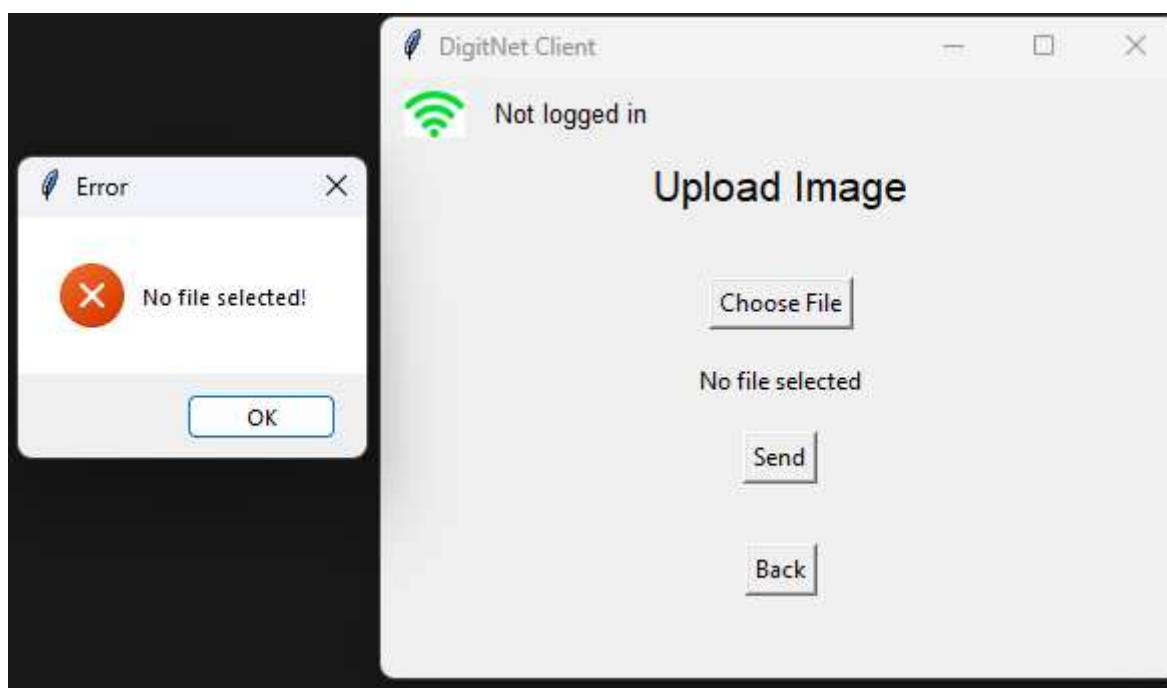


מסך העלאת תמונות-

נותן את האפשרות להעלות תמונות ולשלוח אותן (מראה את השם של התמונה שכרגע בחרו אותה) וכפתור חזרה למסך הראשי, שליחה של תמונה תביא הודעה עם שגיאה או תגובה של השרת.



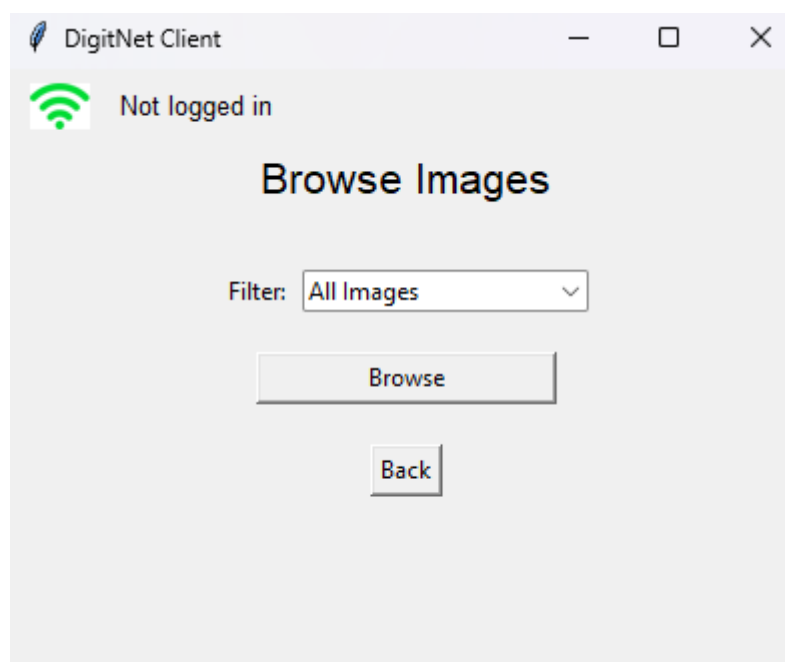


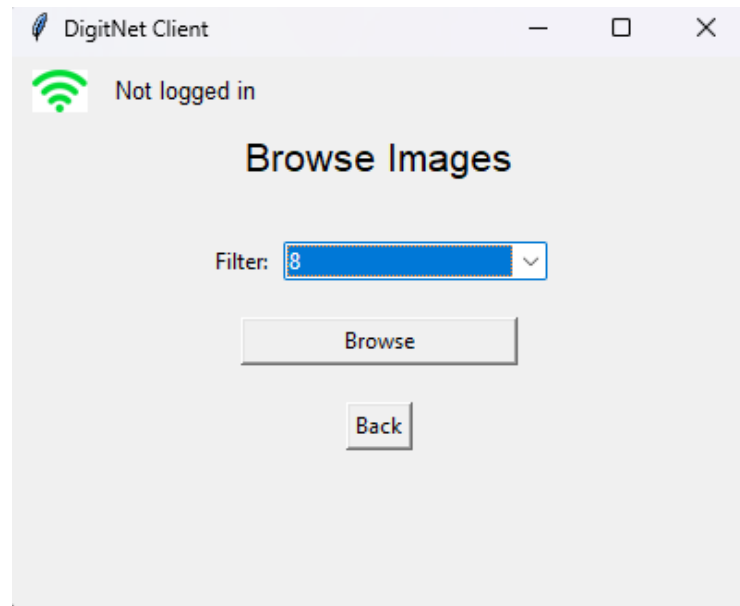


מסך עיון –

מאפשר לבקש לצפות בתמונות (browse) ולבחור תמונות של איזה ספרה לבקש (או לקבל את כל התמונות)

(נפתח תפריט שאפשר לבחור ממנו ספרה אך לא מאפשר לצלם אותו)



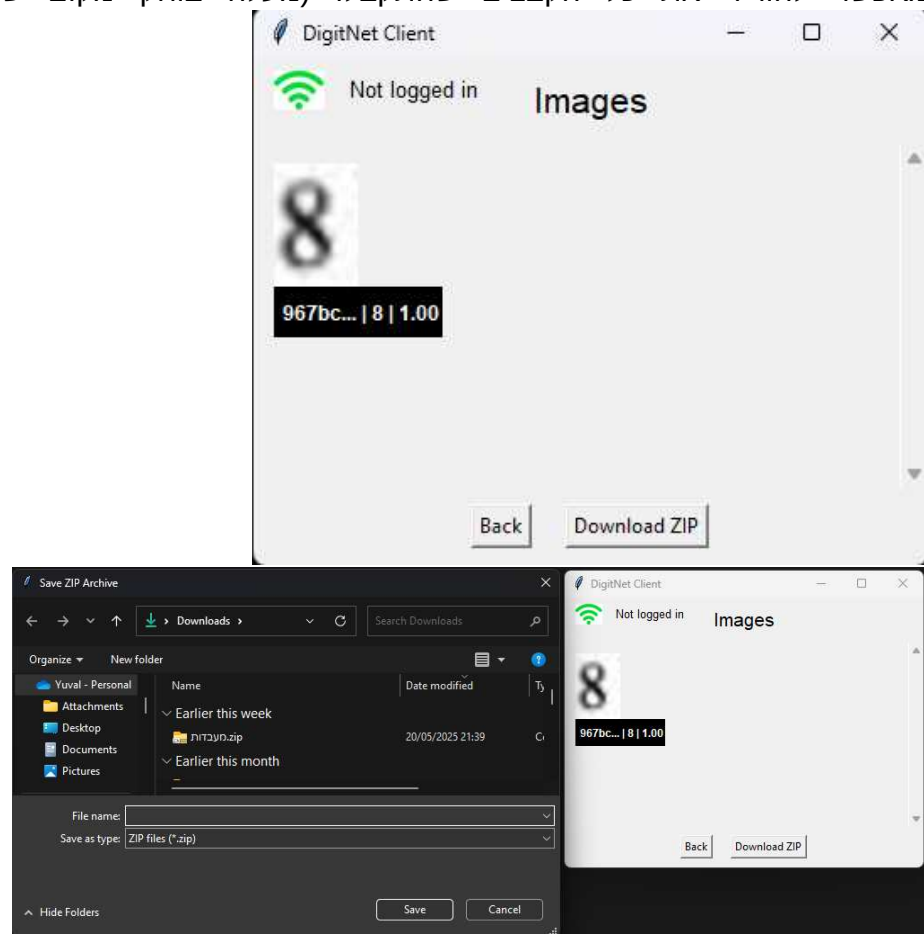


מסך צפייה בתמונות-

מאפשר לראות את התמונות, מתחת לכל תמונה רשום (משמאל לימין) את חמשת האותיות הראשונות של הid (השם), את הספרה שהמודל עשה לו predict, ו confidence (0 עד 1)

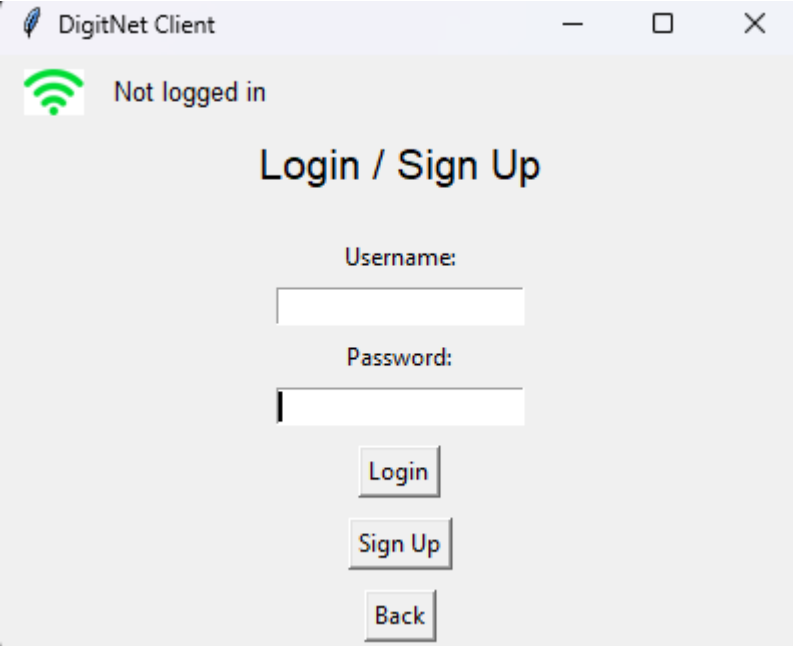
מאפשר לחזור למסך הראשי

מאפשר להוריד את כל הקבצים שהתקבלו (מעלה בוחק מקום של מערכת ההפעלה)

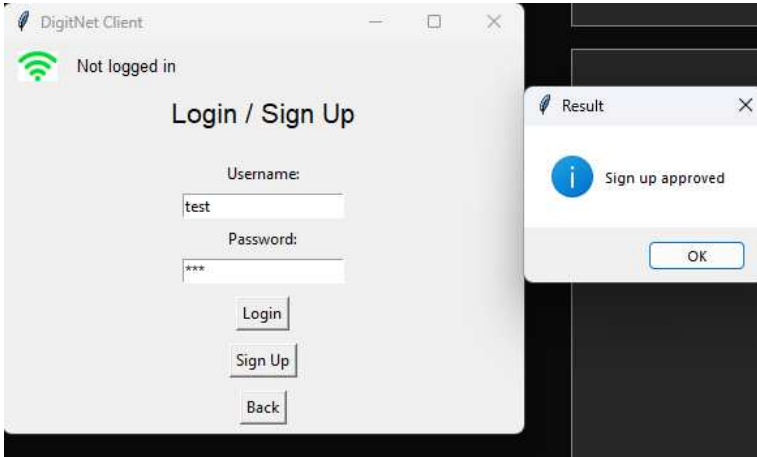


מסך הרשמה-

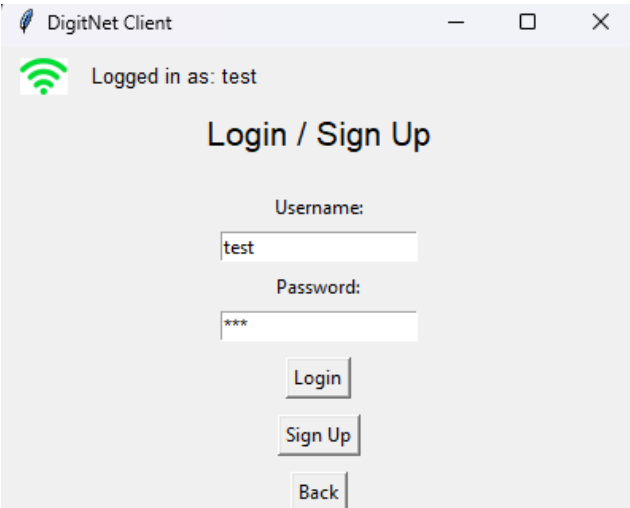
מאפשר להירשם/להתחבר למשתמש על מנת שהתמונות שנשלחות מהלקוח ישמרו במסד הנתונים.
מאפשר הכנה של הנתונים (שם משתמש וסיסמא) והרשמה/התחברות



The screenshot shows the DigitNet Client application window. At the top, it says "Not logged in" next to a green Wi-Fi icon. The main title is "Login / Sign Up". Below this, there are two input fields: "Username:" and "Password:". Under the password field, there are three buttons: "Login", "Sign Up", and "Back".

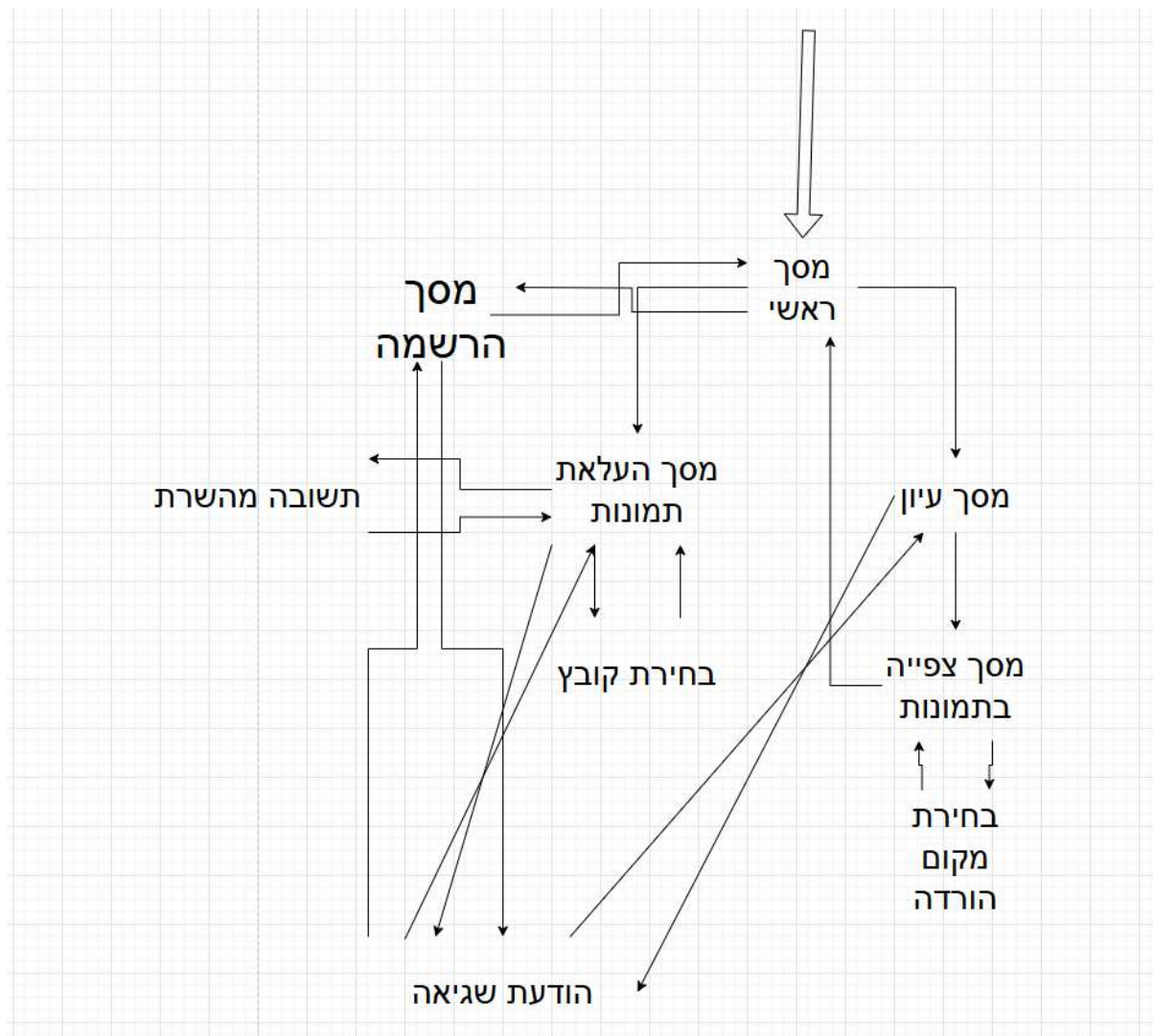


This screenshot shows the same DigitNet Client window, but with the "Username:" field filled with "test" and the "Password:" field filled with "****". A small "Result" dialog box is open on the right, displaying a blue information icon and the text "Sign up approved", with an "OK" button at the bottom.



This screenshot shows the DigitNet Client window after a successful login. The status at the top now says "Logged in as: test" next to the green Wi-Fi icon. The "Login / Sign Up" title and the input fields (Username: "test", Password: "****") remain. The "Login", "Sign Up", and "Back" buttons are still present.

תרשים מסכים

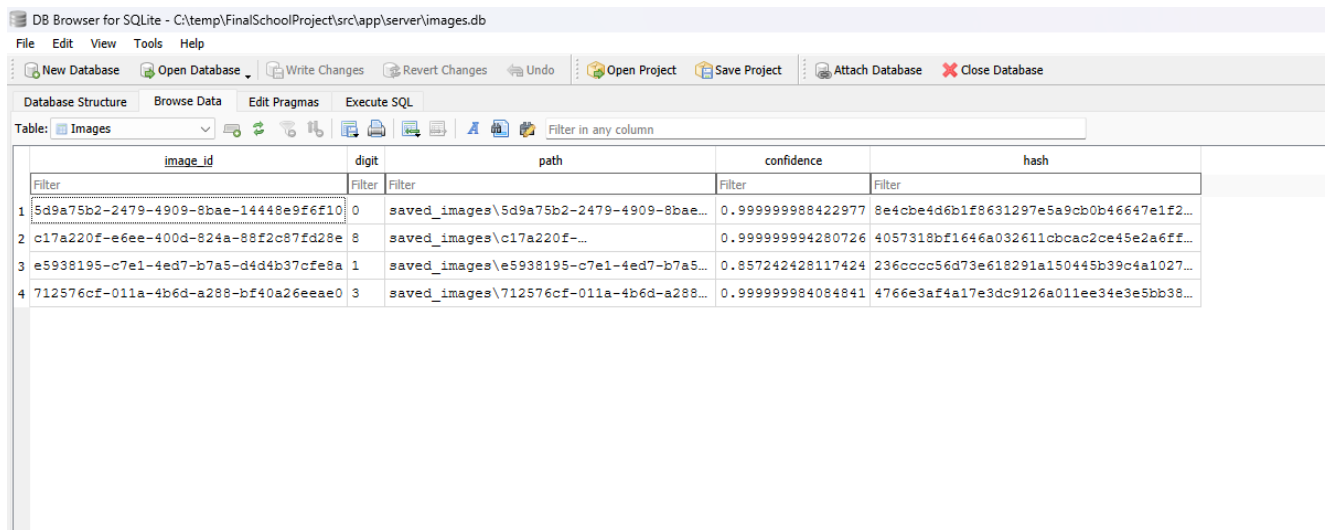


מבני נתונים

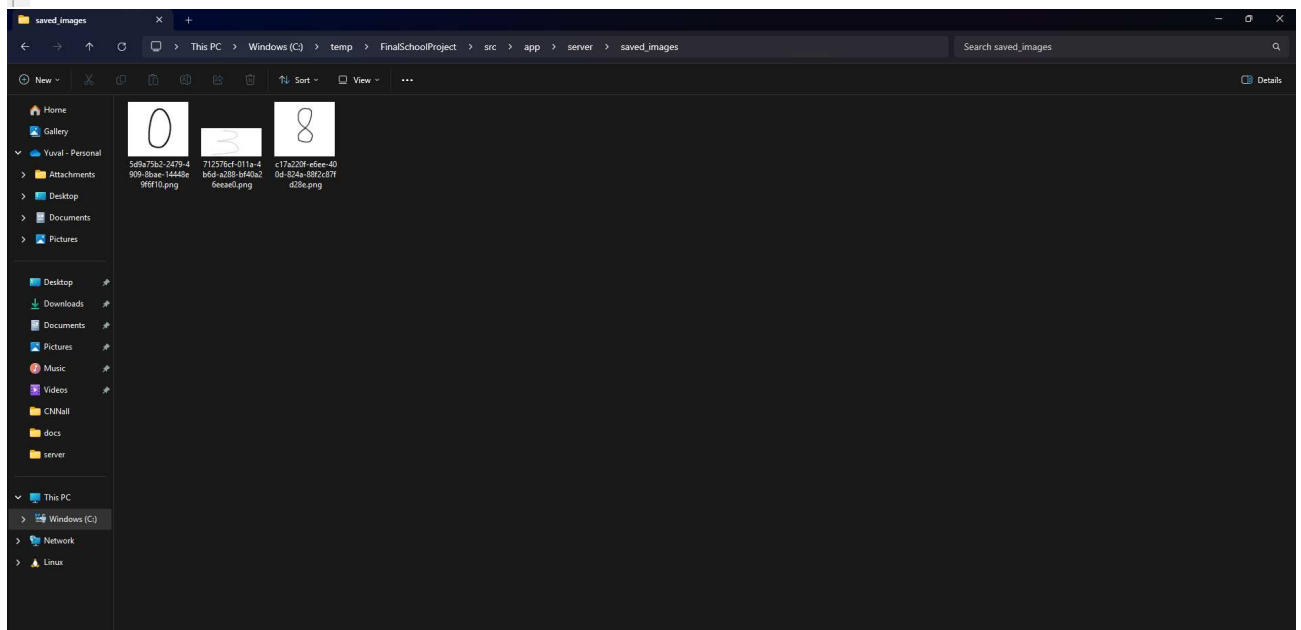
מסד נתונים

מסד הנתונים הוא SQLite עם טבלה אחת, הכוללת את השדות הבאים: מזהה תמונה ייחודי (image_id TEXT PRIMARY_KEY), הספרה שסווגה (digit TEXT), הנתיב לקובץ התמונה המקומי (path TEXT), רמת הביטחון של המודל (confidence REAL) וגיבוב מסוג SHA-256 של התמונה (hash TEXT UNIQUE), המאפשר לזהות תמונות כפולות. השדה hash מוגדר כייחודי כדי למנוע שמירה של אותה תמונה פעמיים, גם אם היא התקבלה ממקורות שונים.

התמונות עצמן שמורות בתת התיקיה saved_images עם image_id בתור השם, בפורמט png, מוקטן לעד 256*256



	image_id	digit	path	confidence	hash
1	5d9a75b2-2479-4909-8bae-14448e9f6f10	0	saved_images\5d9a75b2-2479-4909-8bae...	0.999999988422977	8e4cbe4d6b1f8631297e5a9cb0b46647e1f2...
2	c17a220f-e6ee-400d-824a-88f2c87fd28e	8	saved_images\c17a220f-...	0.999999994280726	4057318bf1646a032611cbcac2ce45e2a6ff...
3	e5938195-c7e1-4ed7-b7a5-d4d4b37cfe8a	1	saved_images\e5938195-c7e1-4ed7-b7a5...	0.857242428117424	236cccc56d73e618291a150445b39c4a1027...
4	712576cf-011a-4b6d-a288-bf40a26eeae0	3	saved_images\712576cf-011a-4b6d-a288...	0.999999984084841	4766e3af4a17e3dc9126a011ee34e3e5bb38...



תור משימות בלקוח

כדי לנהל את המשימות מהמשתמש threadn של הלקוח (מנהל את התקשורת) משתמש בתור, ללקוח יש פעולות שהgui משתמש בהם ששמות משימות בתור. אז threadn של הלקוח מקבל את המשימה בqueue, המשימה מורכבת משני דברים "קוד למשימה" (קוד שהוסכם מראש שיהיה הקוד

של המשימה- לרוב יהיה פשוט הקוד הרלוונטי בפרוטוקול) והפרמטרים (arguments), אם המשימה היא None זה סימן של הgui שהאפליקציה נסגרת.

הורדת תמונות צד לקוח

הלקוח מוריד zip של התמונות, בקובץ zip יש קובץ csv שמכיל את נתונים על הקבצים. בנוסף לכך יש תקייה בשם /images שמכילה את הקבצים עצמם.

קובץ הcsv:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	filename	label	confidence													
2	712576cf-011a-4b6d-a288-bf40a26...	3	1													
3	c17a220f-e6ee-400d-824a-88f2c87f...	8	1													
4	5d9a75b2-2479-4909-8bae-14448e9...	0	1													
5																
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																
16																
17																
18																
19																
20																
21																
22																

תקייה images:

Name	Type	Compressed size	Password ...	Size	Ratio	Date modified
5d9a75b2-2479-4909-8bae-14448e9...	PNG File	8 KB	No	8 KB	0%	05/05/2025 17:57
712576cf-011a-4b6d-a288-bf40a26...	PNG File	4 KB	No	4 KB	0%	05/05/2025 17:57
c17a220f-e6ee-400d-824a-88f2c87f...	PNG File	7 KB	No	7 KB	0%	05/05/2025 17:57

סקירת חולשות ואיומים

-SQL injection

אחת מהפגיעויות הנפוצות והמסוכנות ביותר באפליקציות מבוססות מסד נתונים היא SQL injection מדובר בטכניקת תקיפה שבה תוקף מזין קלט זדוני (למשל במקום ערך של שם משתמש או מזהה) שמכיל קוד SQL. כאשר הקלט לא מסונן כראוי ומוזן ישירות לבקשת SQL, תוקף יכול לשנות את משמעות השאילתה – למשל כדי לחשוף מידע רגיש, למחוק טבלאות, או לעקוף מנגנוני אימות.

בפרויקט שלי, אני עושה שימוש מודע ובטוח ב־ SQLite באמצעות מודול sqlite3 של פייתון. כל השאילתות שמקבלות קלט מהמשתמש נכתבות באמצעות prepared statements (הצהרות מוכנות) – כלומר, אני משתמש ב־ ? בתור מציין מקום לערכים, ומספק את הנתונים כטופל נפרד.

שימוש בשיטה זו מבטיח שהקלט של המשתמש לא יפורש לעולם כחלק מהתחביר של שאילתות SQL אלא רק כערך, גם אם הוא מכיל תווים מסוכנים כמו גרשיים, נקודה־פסיק, או מילות מפתח של SQL.

הסיבה שצריך להגן על מסד הנתונים למרות שאין בו סיסמאות או מידע מאוד רגיש היא שלא רוצים שהמשתמש יכול לקבל את הpath של התמונות שעלול לרמוז על המבנה של המחשב של השרת.

הצפנות-

כדי שלא יוכלו להקשיב לתקשורת בין השרת והלקוח אני מצפין את התקשורת. המידע שעובר הוא לא בהכרח מידע רגיש (כל לקוח שרוצה יכול להתחבר ולקבל את כל התמונות) אך ההצפנות מסייעות בכך שמסתירות את איזו תמונה נשלחה על ידי איזה משתמש.

כדי להצפין את התקשורת בצורה בטוחה אני משתמש בהצפנה אסימטרית (RSA) כדי להחליף את המפתחות להצפנה הסימטרית (AES) שבה אני משתמש כדי להצפין את המידע שעובר בתקשורת.



העלאת קבצים-

בתקשורת שלי אני מעביר קבצים ושומר קבצים שלקוחות שולחים. יש הרבה סכנות בזה, מה אם לקוח שולח קובץ מאוד גדול וזה תופס הרבה מקום (ב-RAM בהתחלה ואז גם בדיסק). מה אם שולחים הרבה מאוד קבצים, יגמר המקום בדיסק. מה אם שולחים הרבה קבצים ואז אחסון של השרת "נאכל". משתמש זדוני עלול לנסות לנצל את חולשות אלו כדי לפגוע בשרת.

על מנת שהשרת יהיה בטוח מהתקפות מהסוג הזה קבעתי כמה הגבלות.

קודם כל יש כמות מוגבלת של מידע שיכול לעבור בהודעה (יש שדה של גודל הודעה בגודל 7 כלומר עשר מליון זה הכי הרבה בתים שיכולים להיות בהודעה), אז בלתי אפשרי לשלוח תמונות גדולות מידי, מעבר לזה הגבלתי את הגודל של התמונות שהשרת מוכן לקבל ל-3000000. מעבר לכך אני

עושה resize לתמונות לפני שאני שומר אותך לגודל 256x256 כדי שלא יוכלו לקחת לי כמה מקום באחסון שהלקוח ירצה. בנוסף לכך גם הגבלתי את כמות הקבצים שהשרת שלי שומר למספר קבוע (קבוע שאפשר לשנות בקובץ שמנהל את מסד הנתונים) (מסד הנתונים מזהה כפילויות כך שזה באמת יהיו תמונות שונות).

כך השרת שלי מוגן מתקיפות שמשתמש זדוני עלול לבצע באמצעות העמסה של קבצים.

בנוסף לכך יש סכנה של שליחה של הקבצים ללקוחות. נגיד מחוברים 100 לקוחות ויש לי 100 תמונות וכל הלקוחות מבקשים את כל התמונות. לפתוח את כל התמונות במקביל יקח הרבה מאוד מקום ב-RAM לכן פיצלתי את השליחה של התמונות לתמונה אחת עבור כל לקוח (כלומר שולח תמונה ואז פותח את התמונה הבאה כך שמתפנה המקום של התמונה הקודמת ב-RAM)



מימוש הפרויקט

סקירת המודולים

מודולים חיצוניים

- **Socket**: משמש לתקשורת TCP בין הלקוח לשרת. המודול מאפשר פתיחת חיבורים, שליחת מידע וקבלת נתונים.
- **Threading**: מאפשר הפעלת תהליכים (threads) במקביל, מה שמאפשר לשרת לנהל מספר לקוחות בו־זמנית וללקוח להריץ תקשורת ברקע.
- **Queue**: תור בטוח לשרשרורים, משמש בלקוח כדי לנהל משימות שמתקבלות מה־GUI ונשלחות לשרת.
- **Os, sys**: מספקים פונקציונליות לעבודה עם מערכת הקבצים והנתיבים, ולשליטה בתצורת הריצה של הקוד.
- **io**: משמש ליצירת אובייקטים מסוג **buffer** שניתן להעביר בהם תמונות בפורמט בינארי (bytes).
- **Pickle**: מאפשר שמירה וטעינה של מודלים מאומנים לדיסק בצורה נוחה.
- **Base64**: שימושי לקידוד מידע בצורה בטוחה להעברה דרך פרוטוקול טקסטואלי, במיוחד כאשר יש צורך בשליחה של מידע בינארי (כמו תמונות) מעל TCP.
- **Numpy**: ספריית חישוב מטריציונית מרכזית שמשמשת לאורך כל הפרויקט לכל פעולות ה־forward, backward, עדכוני משקולות, עיבוד תמונות ועוד.
- **PIL (pillow)**: משמשת לפתיחה, שמירה, שינוי קנה מידה של תמונות ועוד פעולות עיבוד תמונה.
- **Cv2 (opencv)**: משמש בעיקר במודול ההגדלה (augment) של התמונות, לדוגמה סיבוב, תזוזה, דילול והוספת רעש.
- **Crypto (pycryptodome)**: משמש להצפנת תקשורת הלקוח־שרת. נעשה שימוש ב־AES להצפנה סימטרית וב־RSA להצפנה א־סימטרית של מפתח AES.
- **Tkinter**: משמש לבניית הממשק הגרפי של הלקוח.
- **time**: משמש למדידת זמני הריצה של שלבי אימון המודל והדפסת משך האפוק.
- **Copy**: משמש לשמירה של עותק עמוק של המודל במהלך הלמידה במקרה שעוצרים את הלמידה מוקדם מהמתוכנן.
- **Gzip**: משמש לפתיחת קבצי ה־MNIST הדחוסים בעת טעינת הדאטאסט.
- **Urllib.request**: משמש להורדת קבצי הדאטאסטים (MNIST, CIFAR-10) מהאינטרנט במידת הצורך.
- **Tarfile**: משמש לחילוץ (extract) של קובצי הארכיון של CIFAR-10 לאחר הורדתם.
- **ABC**: משמש להגדרת מחלקות אבסטרקטיות (כמו Layer, Loss) שאינן ניתנות להופעה ישירה אלא משמשות כבסיס למחלקות מורשות.

- Zipfile: משמש ליצירת הקובץ zip שמכיל את התמונות והמידע הנלווה בשביל המשתמש
- csv: משמש לכתיבה של הטבלת הנתונים בתוך קובץ הקובץ zip שהשלוח מוריד

מודולים ומחלקות פנימיות

*אם לא מצוין מה הפונקציה מחזירה היא לא מחזירה ערך בעל משמעות.

Client (client.py) - מחלקה יורש מ threading.Thread

- **תפקיד:** אחראי על התקשורת מול השרת – פתיחת חיבור, שליחת קבצים, קבלת תגובות וביצוע Handshake מוצפן. (RSA+AES)

• **תכונות:**

- dest: כתובת השרת
- socket TCP -Sock
- Request_queue: תור משימות
- Connected: סטטוס חיבור
- Crypto: אובייקט הצפנה מסוג ClientCrypto
- Gui_callback: הממשק הגרפי בשביל תצוגה

• **מתודות:**

- __init__(dest_ip, dest_port=protocol.PORT, gui_callback=None) – פעולה בונה מקבלת את הקו של השרת את הפורט שהשרת מחכה עליו ואת gui של התכונה מאתחלת את כל התכונות של המחלקה.
- connect() – יוזם את החיבור עם השרת ומבצע תהליך Handshake מוצפן. אין פרמטרים. לא מחזיר ערך.
- run() – מפעיל את הלקוח כ- thread ומריץ את לולאת העבודה. אין פרמטרים. לא מחזיר ערך.
- Handshake() – מחליף מפתחות הצפנה עם השרת באמצעות RSA ומאמת את הקשר. אין פרמטרים. לא מחזיר ערך.
- queue_task(task_code, *args) – מוסיף משימה לתור לביצוע עתידי. מקבל קוד פעולה וארגומנטים. לא מחזיר ערך.
- activate() – מנהל את לולאת הביצוע המרכזית שמבצעת משימות ומתקשרת עם השרת. אין פרמטרים. לא מחזיר ערך.
- handle_task(code, args) – מבצע שליחת בקשה לפי קוד פעולה ופרמטרים. מחזיר ערך בוליאני – האם לצפות לתגובה מהשרת.
- send(*msg) – שולח הודעה מוצפנת לשרת. מקבל כל הודעה בפורמט תואם. לא מחזיר ערך.

- `recv()` – מקבל הודעה מוצפנת מהשרת, מפענח ומחזיר אותה כמבנה נתונים מנותח (רשימת שדות).
- `close()` – סוגר את החיבור TCP ומסיים את פעולת הלקוח. לא מחזיר ערך.
- `send_file(path)` – מקבל `path` של קובץ. שם ב `queue` בקשת קובץ. לא מחזיר ערך.
- `request_images(digit=None)` – שם ב `queue` בקשת תמונות מהשרת. אם מצוין ספרה – מקבל רק תמונות מתויות בספרה זו. לא מחזיר ערך.
- `request_sign_up(username, password)` – שם ב `queue` בקשה להרשמה `sign_up` מקבל שם משתמש וסיסמה ולא מחזיר ערך
- `request_log_in(username, password)` – שם ב `work queue` בקשה להרשמה `login`, מקבל שם משתמש וסיסמה ולא מחזיר ערך
- `business_logic(response)` – מקבל תגובה של השרת, מפרש את התגובה שהתקבלה מהשרת ומבצע פעולה מתאימה בממשק. לא מחזיר ערך.
- `convert_image_string_to_tuple(list)` – מקבל רשימה של מחרוזות, ממיר רשימת מחרוזות מהשרת למבנה רשומות של תמונות. מחזיר רשימת טאפלים (`id, image, digit, confidence`)
- `recv_files_process(, amount_of_files)` – עושה את התהליך של קבלת התמונות מהשרת (מקבל `amount_of_files` כמות הקבצים המצופה) מחזיר רשימת טאפלים (`id, image, digit, confidence`)

ClientCrypto (client.py) - מחלקה

- **תפקיד:** מטפל בהצפנה ופענוח של הודעות בין הלקוח לשרת באמצעות AES במצב CBC, כאשר מפתח AES עצמו מוצפן ב־RSA.

• תכונות:

- `aes_key`: מפתח סימטרי באורך 32 בנים
- `aes_iv`: וקטור אתחול באורך 16 בנים

• מתודות:

- `__init__()` – פעולה בונה, מאתחלת את המפתח ואת ה `iv` של AES.
- `encrypted_key_iv(rsa_key)` – מקבל מפתח RSA ציבורי ומחזיר את המפתח הסימטרי AES כשהוא מוצפן וכן את ה־IV. משמש בעת Handshake.
- `encrypt(plaintext)` – מקבל טקסט מצפין טקסט בפורמט `string` בעזרת AES. מחזיר מחרוזת מוצפנת ב־`bytes`.
- `decrypt(encrypted_text)` – מפענח מחרוזת `bytes` ומחזיר את הטקסט המקורי כ־`string`.

ClientGUI (gui.py) - מחלקה

- תפקיד: ממשק משתמש גרפי עבור לקוח מערכת לזיהוי ספרות. מאפשר העלאת תמונות, עיון בגלריה, התחברות למשתמש, והורדת תמונות.
- **תכונות:**
 - Root: חלון ראשי של Tkinter.
 - Client: מופע של מחלקת Client המטפל בתקשורת עם השרת.
 - logged_in_user: שם המשתמש המחובר כעת, או None אם אין חיבור.
 - connection_label: תווית המציגה סטטוס חיבור לרשת (אייקון).
 - status_frame: מסגרת עליונה הכוללת סטטוס חיבור וסטטוס התחברות.
 - login_status_label: תווית המציגה את שם המשתמש המחובר או סטטוס אי-התחברות.
 - file_path: הנתיב לתמונה שנבחרה להעלאה.
 - file_label: תווית המציגה את שם הקובץ הנבחר.
 - current_images: רשימת התמונות שהתקבלו מהשרת בעת עיון בגלריה.
 - image_refs: הפניות לאובייקטים של תמונות (PhotoImage) למניעת מחיקתן מהזיכרון.
 - no_internet_img, connected_img: תמונות המייצגות מצב חיבור.
 - filter_var: ערך תיבת הבחירה לסינון תמונות בגלריה.
 - filter_dropdown: תיבת בחירה להצגת תמונות לפי תווית ספרה.
- **מתודות:**
 - __init__(): מאתחל את כל התכונות של gui, יוצר את החלון המרכזי ופותח את העמוד המרכזי.
 - activate(): מפעיל את לולאת הממשק הגרפי ומתחיל בדיקת סטטוס חיבור תקופתית.
 - display_result(message, message_type): מקבל הודעה, וסוג ההודעה - "error" זה אומר הודעת שגיאה מציג הודעה למשתמש result – כהודעת מידע, error כהודעת שגיאה.
 - create_main_screen(): יוצר את תפריט הבית: העלאה, עיון, התחברות ויציאה.
 - open_login_screen(): יוצר מסך התחברות/הרשמה, מאפשר שליחת בקשות התחברות/הרשמה דרך Client.
 - gui_set_logged_in_user(username): מעדכן את שם המשתמש המחובר להצגה בממשק.
 - update_login_status(username): מקבל שם משתמש משנה את תווית סטטוס ההתחברות בהתאם למשתמש הפעיל.
 - create_status_frame(): יוצר את המסגרת העליונה עם סטטוס חיבור וסטטוס התחברות.
 - update_connection_status(): בודק אם הלקוח מחובר ומעדכן את האייקון בהתאם.
 - start_connection_polling(interval_ms): מקבל אינטרוול במילישניות. מתחיל בדיקה מחזורית (כל כמה אלפיות שניה) של מצב החיבור.

- `open_upload_screen()`
יוצר את מסך העלאת התמונה.
- `upload_image()`
פותח חלון לבחירת קובץ תמונה ושומר את הנתוב.
- `send_image()`
שולח את התמונה שנבחרה לשרת דרך `Client`.
- `open_browse_screen()`
יוצר את ממשק העיון בתמונות ומאפשר לבחור פילטר.
- `browse_images()`
שולח בקשה לשרת להחזרת תמונות, בהתאם לפילטר שנבחר.
- `display_images(images)`
מציג את התמונות בגלריה, כולל תצוגה מוקטנת של כל תמונה עם מזהה, תווית וביטחון.
- `download_zip()`
שומר את התמונות הנוכחיות בקובץ ZIP, כולל `labels.csv` עם מידע על כל תמונה.
- `exit_gui()`
סוגר את הממשק, סוגר את הלקוח ומסיים את התהליך.
- `handle_server_response(response)`
מקבל תגובה של השרת
מציג תגובת שרת שהתקבלה מהלקוח (למשל תוצאה או שגיאה).

ImagesORM (`img_db_orm.py`)

תפקיד:

ניהול מסד נתונים מקומי (SQLite) הכולל שמירת תמונות ומטא־נתונים, הגבלת היסטוריה, ניהול משתמשים, אימות, ושיוך תמונות למשתמשים.

תכונות:

`db_path`
נתיב לקובץ מסד הנתונים (ברירת מחדל: `'images.db'`).

`image_dir`
תיקייה מקומית לשמירת קבצי תמונה.

`conn`
חיבור פעיל למסד הנתונים (אובייקט `sqlite3.Connection`).

`cursor`
אובייקט `cursor` לביצוע שאילתות מול בסיס הנתונים.

פעולה בונה:

`_init_(db_path='images.db', image_dir='saved_images')` – פעולה בונה, מאתחל את כל התכונות ומכין תיקייה לשמור בה את התמונות.

מתודות לניהול בסיס נתונים:

`open_DB()`
פותחת חיבור למסד הנתונים. חובה לפני פעולות כתיבה/קריאה.

`close_DB()`
סוגרת את החיבור למסד הנתונים.

`commit()`
שומרת שינויים למסד הנתונים.

`create_tables()`
יוצרת את הטבלאות `Users` ו-`Images` אם אינן קיימות.

מתודות לניהול משתמשים:

`register_user(username, password)`
מקבל שם משתמש וסיסמה.
רושמת משתמש חדש. מחזירה `user_id` אם הצליח, או `None` אם שם המשתמש תפוס.

`authenticate_user(username, password)`
מקבל שם משתמש וסיסמה.
מאמתת משתמש לפי הסיסמה. מחזירה `user_id` אם הסיסמה נכונה, אחרת `None`.

מתודות לטיפול בתמונות:

`save_image_file(image_bytes, max_size=256)`
מקבל בתים של תמונה וגודל מקסימלי שבו התמונה תישמר.
שומרת תמונה מוקטנת, מחזירה מזהה, נתיב, ו-`hash` של התמונה.

`insert_image(image_id, digit, path, confidence, hash_val, user_id)`
מקבל `id` של התמונה, הספרה שהמודל זיהה, המיקום של התמונה, ביטחון של המודל בנכונות של הזיהוי, ערך `hash` של הבתים של התמונה, `id` של המשתמש ששלח את התמונה.
מוסיפה תמונה לטבלה אם לא קיימת לפי `hash`, מוחקת תמונות ישנות אם יש יותר מ-100.

`delete_old_files()`
מוחקת קבצים מקומיים שאינם נמצאים במסד הנתונים ומנקה רשומות חסרות.

`process_and_store(image_bytes, digit, confidence, user_id)`
מקבל את הבתים של תמונה, הספרה שהמודל זיהה, ביטחון של המודל בנכונות הזיהוי, וה-`id` של המשתמש ששלח את התמונה.
מבצע תהליך מלא: שמירה, הכנסת שורה למסד, וניקוי קבצים מיותרים.

מתודות לשליפת תמונות:

`get_all_images_files()`
מחזירה את כל התמונות שנשמרו עם המידע המלא כקבצים.

`get_image_by_digit_files(digit)`

מקבל ספרה.

מחזירה רק את התמונות שתויגו עם ספרה מסוימת.

Files(img_db_orm.py) – מחלקה

- תפקיד: לשמש את השרת בפתיחת וקראת הקבצים שהוא שולח ללקוח אחד אחד, מממש iterator
- תכונות:

- Rows שורות מהdatabase של קבצים שהשרת הולך לשלוח ללקוח
- Self.index מספר השורה שהאובייקט נמצא בו עכשיו

• מתודות:

- `__init__(rows)` – פעולה בונה, מקבל את השורות מהdatabase
- `__len__()` – מחזיר את מספר התמונות שיש באובייקט הזה
- `__iter__()` – מאתחל את index
- `__next__()` – נותן את התוכן של הקובץ הבא בקובייקט ומקדם את index

CNN (cnn.py) – מחלקה

- תפקיד: מייצג את המודל המורכב של רשת קונבולוציה (CNN) כתצורת שכבות מותאמות אישית, המאפשרת ביצוע מעבר קדימה ואחורה ואימון.

• תכונות:

- layers: רשימת השכבות המרכיבות את המודל.

• מתודות:

- `__init__(layers)` – פעולה בונה, מקבל את השכבות שממנה מורכבת הרשת ומאתחלת את התכונה
- `forward(input)` – מבצע מעבר קדימה דרך כל השכבות. מקבל טנזור קלט ומחזיר את פלט הרשת.
- `backward(grad)` – מחשב את גרדיאנט השגיאה לאחור דרך כל השכבות. מקבל את הגרדיאנט מהשכבה האחרונה.
- `update_parameters(learning_rate)` – מקבל קצב למידה מעדכן את פרמטרי כל שכבה נלמדת לפי גרדיאנט ואחוז הלמידה.

Trainer (trainer.py) – מחלקה

- תפקיד: מבצע את תהליך האימון של המודל עם תמיכה בירידת למידה, אימות ו-early stopping.

• תכונות:

- model: המודל לאימון
- optimizer: אובייקט SGD
- loss_function: פונקציית איבוד
- learning_rate, decay_epochs, decay_rate: פרמטרי שליטה בקצב הלמידה
- validator: אובייקט ממחלקת Tester לצורך הערכה
- early_stopping, patience: בקרים להפסקת אימון מוקדמת

• **מתודות:**

- train(data_loader, num_epochs, val_data_loader=None) – מקבל טוען נתונים, מספר חזרות על הנתונים, וטוען נתונים כדי לבדוק את המערכת. מאמן את המודל על בסיס הנתונים, מבצע אימות תקופתי, שומר את המודל הטוב ביותר ומטפל בהפסקת אימון מוקדמת אם נדרש.

Tester (tester.py) - מחלקה

- **תפקיד:** מודול להערכת ביצועי המודל המאומן באמצעות סט בדיקה או אימות.

• **תכונות:**

- model: המודל להערכה
- loss_function: פונקציית איבוד

• **מתודות:**

- __init__(model, loss_function=None) – פעולה בונה, מקבל את המודל ואת פונקציית האיבוד ומאתחלת את התכונות
- test(data_loader) – מקבל טוען נתונים מבצע מעבר קדימה על כל הדאטאסט ובודק דיוק ואיבוד ממוצע. מחזיר איבוד ודיוק.

Conv2D (conv.py) - מחלקה יורש מ TrainableLayer

- **תפקיד:** שכבת קונבולוציה דו־ממדית. מבצעת סינון של התמונות באמצעות פילטרים נלמדים (kernels).

• **תכונות:**

- weights: משקלי הפילטרים
- biases : ההטיות (bias) לכל פילטר
- weights_gradient, biases_gradient: גרדיאנטים של הפילטרים וההטיות לצורך עדכון באימון

• **מתודות:**

- __init__(in_channels, out_channels, kernel_size, initialization, stride=1, padding=0) פעולה בונה, מקבלת את כמות הערוצים שהפילטרים יפעלו עליהם, כמות

הפילטרים, גודל הפילטרים, פונקציה לאיתחול הפילטרים, כמה לקפוץ בין כל כפעלה של הפילטר בקונבולוציה מאתחר את כל התכונות כפי שדרוש.

- `forward(input)` - מקבל טנסור מבצע קונבולוציה בין הפילטרים לבין הקלט, מחזיר מפה חדשה.
- `backward(output_grad)` - מקבל את גרדיאנט של הפלט מחשב את הגרדיאנט של הקלט והמשקלים לפי גרדיאנט היציאה.
- `update_parameters(learning_rate)` - מקבל קצב למידה מעדכן את הפילטרים וההטיות לפי הגרדיאנטים שנצברו וקצב הלמידה

Dense (dense.py) - מחלקה יורש `TrainableLayer`

- **תפקיד:** שכבה מלאה (Fully Connected) הממפה קטור כניסה לוקטור יציאה בעזרת מטריצת משקולות.

• **תכונות:**

- `weights`: מטריצת המשקולות
- `biases`: ההטיות
- `weights_gradient, biases_gradient`: גרדיאנטים של הפרמטרים

• **מתודות:**

- `__init__(output_size, input_size, initialization)` - פעולה בונה, מקבלת מספר הנירונים בשכבה, גודל הקלט, ופונקציית אתחול למשקולות. מאתחלת את כל התכונות כדרוש
- `forward(input)` - מקבלת טנסור מחשבת מכפלה מטריציונית בין הקלט למשקולות ומוסיפה את ההטיות.
- `backward(output_grad)` - מקבל את הגרדיאנט של הפלט מחשבת את הגרדיאנט לפי חוק השרשרת עבור כל פרמטר.
- `update_parameters(learning_rate)` - מקבל קצב למידה מעדכנת את הפרמטרים לפי הגרדיאנטים ואחוז הלמידה.

Flatten (flatten.py) - מחלקה יורש מ `Layer`

- **תפקיד:** ממירה טנזור מרובה ממדים לווקטור חד-ממדי לצורך מעבר לשכבות צפופות (Dense).

- **תכונות:**

- `input_shape`: שמירה על מבנה הקלט המקורי לצורך חישוב `backward`

- **מתודות:**

- `__init__()` – פעולה בונה, מאתחלת את התכונות כדרוש
 - `forward(input)` – מקבל טנזור, ממירה את הקלט לפורמט שטוח `(batch_size, -1)`.
 - `backward(output_grad)` – מחזירה את הגרדיאנט למימדים המקוריים של הקלט.

MaxPool2D (maxpool.py) - מחלקה יורש Layer

- **תפקיד:** שכבת מקסימום שמבצעת דגימה מרחבית על ידי בחירת הערך המקסימלי בכל חלון (pooling window).

- **תכונות:**

- `pool_size`: גודל החלון
 - `stride`: קפיצת החלון בין אזורים
 - `input, mask`: קלט ו-`mask` לזכירת מיקום המקסימום, בשביל חישוב `backward`

- **מתודות:**

- `__init__(pool_size, stride=None, padding=0)` - פעולה בונה. מקבלת את גודל הפולינג (גודל החלונות שממנו לוקחים את המקסימום, כמה קפיצה לעשות עם החלון, וכמה `padding` לשים לקלט מכל צד. מאתחלת את התכונות כדרוש
 - `forward(input)` – מקבל טנזור מחזירה מפת מאפיינים מדוללת עם הערכים המקסימליים מכל חלון.
 - `backward(output_grad)` – מקבלת גרדיאנט לפלט מפזרת את הגרדיאנט חזרה רק למקומות שהכילו את המקסימום המקורי.

InputLayer (input_layer.py) - מחלקה יורש Layer

- **תפקיד:** שכבת קלט פשוטה שמשמשת להעברת מידע לתוך הרשת בצורה עקבית.

- **מתודות:**

- `__init__(input_shape)` – פעולה בונה, מקבלת את גודל הקלט הצפוי, מאתחלת את התכונות כדרוש

- `forward(input)` – מקבלת טנסור מחזירה את הקלט כפי שהוא.

- `backward(output_grad)` – מקבלת גרדיאנט על הפלט מחזירה את הגרדיאנט כפי שהוא.

ReLU (relu.py) - מחלקה יורש Layer

- **תפקיד:** פונקציית אקטיבציה רקטיפית (Rectified Linear Unit) שמאפסת ערכים שליליים בקלט.

• מתודות:

- `forward(input)` – מקבלת טנסור מחזירה קלט שבו כל ערך שלילי מאופס, וכל ערך חיובי נשמר. משמשת כאקטיבציה לא־ליניארית.
- `backward(output_grad)` – מקבל גרדיאנט של הפלט מחשבת את הגרדיאנט של ReLU שומרת על גרדיאנט עבור ערכים חיוביים ומאפסת את השאר.

Sigmoid (sigmoid.py) - מחלקה יורש Layer

- **תפקיד:** פונקציית אקטיבציה שממפה את הקלט לערכים בין 0 ל־1 באמצעות נוסחת הסיגמואיד.

• מתודות:

- `forward(input)` – מקבל טנסור מחזירה את הערכים שעברו דרך פונקציית הסיגמואיד
- `backward(output_grad)` – מקבלת גרדיאנט של הפלט מחשבת את הגרדיאנט על פי הנגזרת של פונקציית הסיגמואיד ומחזירה את הגרדיאנט המותאם.

Softmax (softmax.py) - מחלקה יורש Layer

- **תפקיד:** פונקציית אקטיבציה לרמות פלט (output layer) שמעבירה את הקלט לחלוקה נורמלית (סכום 1) לצורך חישוב הסתברויות.

• מתודות:

- `forward(input)` – מקבלת טנסור מחשבת את הפלט הרך של הרשת כהסתברויות לכל מחלקה.

- backward(output_grad) –

מקבל גרדיאנט של הפלט.

מחשבת גרדיאנט של פונקציית Softmax בהנחה שתחושב יחד עם פונקציית איבוד CCE.

Loss(/losses/base.py) – מחלקה

- תפקיד- מחלקה אבסטרקטית עבור פונקציות איבוד
- תכונות – prediction הערך שהוציא המודל, targets הערך הרצוי
- מתודות:
 - _compute_loss – מחשב את את הערך של פונקציית האיבוד
 - _compute_grad – מחשב את הגרדיאנט
 - Forward(prediction, target) – מעטפת לחישוב הערך הפונקציה, מקבל את מה שהמודל חזה ואת התוצאה הרצויה
 - Backward() – מעטפת לחישוב הגרדיאנט

CategoricalCrossentropy (cce.py) - יורש מLoss

- תפקיד: פונקציית איבוד למדידה של מרחק בין התפלגות הפלט של המודל לבין תיוג אמיתי.
- תכונות:
 - predictions, targets : מאחסן את הקלטים עבור חישוב קדימה ואחורה.
- מתודות:
 - forward(predictions, targets) – מקבל את מה שהמודל חזה ואת התוצאה הרצויה מחזירה את ערך ה־ loss של CCE על סמך חישוב הלוגריתם של הפלט מול התיוג האמיתי.
 - backward() – מחזירה את גרדיאנט האיבוד: ההפרש בין הפלטים לבין התיוג (מניח שכבר עבר softmax)

MeanSquaredError (mse.py) - מחלקה יורש מLoss

- תפקיד: מחשבת את ממוצע ריבועי ההבדלים בין הפלט לתשובה האמיתית.
- תכונות:
 - predictions, targets : ערכי פלט ותשובות לצורך חישוב ההפרש.
- מתודות:
 - forward(predictions, targets) – מקבל את מה שהמודל חזה ואת התוצאה הרצויה מחשבת את ממוצע ריבועי הסטיות בין הפלט לתשובה.
 - backward() – מחשבת את הגרדיאנט של פונקציית האיבוד ביחס לפלט.

Layer (/layers/base.py) - מחלקה

- תפקיד: מחלקה אבסטרקטית עבור שכבות ברשת נוירונים. משמשת כבסיס לכל שכבה עם תבנית לפונקציות forward backward.

• **מתודות:**

- `forward(input)` – הגדרה אבסטרקטית. כל שכבה מממשת זאת בעצמה.
- `backward(output_grad)` – הגדרה אבסטרקטית. חובה למימוש בשכבות ירושה.

Layer - TrainableLayer (base.py)

- תפקיד: מחלקה אבסטרקטית עבור שכבות שאפשר ללמד אותן (יש להן פילטרים, משקולות כלשהן או `bias`)
 - תכונות: `weights_gradient`, `biases_gradient`, `weight`, `biases` – מקשולות הטייות והגרדיאנטים שלהם (לצורך למידה)
 - מתודות:
 - `__init__()` – פעולה בונה, מאתחלת את התכונות כדרוש.
 - `update_parameters(learning_rate)`
- מקבל קצב למידה, מעדכן את המקבלים וההטיות לפי הגרדיאנטים וקצב הלמידה.

SGD (optimizer.py)

- **תפקיד:** מבצע את עדכון המשקולות לפי אלגוריתם SGD (stochastic gradient descent)
 - **מתודות:**
 - `step(learning_rate)` – מקבל קצב למידה
- עובר על כל שכבה שניתנת לאימון (`TrainableLayer`) ומעדכן את משקליה לפי הגרדיאנט שנצבר וקצב הלמידה.

Augment (augment.py) - מודול

- **תפקיד:** מבצע אוגמנטציה של תמונות עבור הגדלת המגוון של הדאטאסט, כולל רעש, סיבוב, שיקוף ועוד.
- **מתודות:**
 - `__init__(model)` – פעולה בונה, מקבלת מודל מאתחלת את התכונה כדרוש
 - `random_transform(image)` – מקבל תמונה ומחזיר גרסה משופצת שלה לפי פרמטרים אקראיים.
 - `rotate_image(image)` – מקבל תמונה מסובב את התמונה בזווית אקראי ומחזיר אותה.
 - `flip_image(image)` – מקבל תמונה מחזיר את התמונה במצב שיקוף אופקי או אנכי באופן אקראי.
 - `add_noise(image)` – מקבל תמונה מוסיף רעש גאוסיאני לתמונה ומחזיר אותה

Initializer (initializers.py) - מודול

- **תפקיד:** פונקציות אתחול עבור שכבות – מגדירות ערכים התחלתיים עבור משקולות.

- **מתודות:**

- `xavier_initializer(shape)` – מקבל צורה של טנסור מחזיר מטריצה בצורה לפי הקלט מאותחלת לפי, Xavier initialization, כלומר ערכים שנדגמו מאחידות סביב אפס לפי גודל השכבה.

TensorPatches (tensor_patches.py) - מודול

- **תפקיד:** עוזר פנימי למודול – Conv2D מספק פעולות המרה של טנסור ל-`columns` וחזרה לצורך קונבולוציה יעילה.

- **מתודות:**

- `im2col(input_data, filter_h, filter_w, stride=1)` – מקבל את הטנסור, גודל הפילטר וכמה לדלג בכל פעם שמכפילים את את הפילטר ממיר את טנסור התמונות לטבלת טלאים (patches) לצורך פעולת קונבולוציה מהירה ומחזיר את התוצאה.
- `col2im(col, input_shape, filter_h, filter_w, stride=1)` – מקבל טבלת טלאים (patches) גודל הקלט המקורי, גודל הפילטר וכמה לדלג בכל פעם שמכפילים את את הפילטר מבצע את ההמרה ההפוכה: מטבלת טלאים חזרה לטנסור המקורי ומחזיר את התוצאה

DataLoader (dataloader.py) - מחלקה

- **תפקיד:** אחראי על חלוקת הדאטאסט לבאצ'ים, ניהול ערבוב וסדר, ותמיכה בלולאת אימון.

- **תכונות:**

- `images, labels`: נתוני הקלט והתשובות
- `batch_size`: גודל באץ'
- `shuffle`: האם לערבב את הסדר בכל אפוק
- `indices, current_index`: מעקב אחר מקום בלולאה

- **מתודות:**

- `__init__(dataset, batch_size, shuffle=True, transform=None)` – פעולה בונה מקבלת את סט הנתונים, גודל הבאץ', האם לערבב, ופונקציה שעורכת שינויים בתמונות. מאחלת את התכונות כדרוש.
- `__iter__()` – מאתחל את האיטרטור, כולל ערבוב אם נדרש.
- `__next__()` – מחזיר את הבאץ' הבא לפי גודל ומיקום.
- `__len__()` – מחזיר את מספר הדגימות הכולל בדאטאסט.

CIFAR10Loader (cifar10.py) ו־MNISTLoader (mnist.py) מודולים

- **תפקיד:** טוענים את הדאטאסטים MNIST ו־CIFAR-10 כולל הורדה מהאינטרנט, המרה לפורמט NumPy וחלוקה לסטים.

• **מתודות עיקריות (בשניהם):**

- download_dataset() – מוריד את קבצי הדאטאסט מהאינטרנט אם לא קיימים מקומית.
- extract_dataset() – פותח את הקבצים (gzip / tar) ומכין את הדאטא לטעינה.
- load_data() – טוען את הנתונים לקובצי NumPy ומחזיר tuple של (images, labels).

Server (server.py) - מחלקה

- **תפקיד:** שרת ראשי שמאזין לבקשות TCP מהלקוחות, יוצר תהליכונים (threads) עבור כל לקוח, מנהל Handshake ומעבד בקשות תמונה.

• **תכונות:**

- rsa_key : מפתח RSA ל־Handshake.
- online : דגל סטטוס השרת.
- Socket : TCP socket
- clients : רשימת לקוחות פעילים.
- server_lock : מנגנון נעילה למניעת התנגשויות בעת קבלת חיבורים.

• **מתודות:**

- __init__() – פעולה בונה, מאתחלת את התכונות כדרוש ומכינה את הsocket לקבלת לקוחות.
- activate_server() – מאזין לחיבורים חדשים, יוצר ClientHandler ומפעיל כל חיבור ב־Thread נפרד.
- close() – עוצר את השרת ומסיים את כל התקשורת עם הלקוחות.

ClientHandler (server.py) - מחלקה יורשת Treading.Thread

- **תפקיד:** מנהל את התקשורת עם לקוח ספציפי: מבצע Handshake, מקבל בקשות ומחזיר תגובות.

• **תכונות:**

- crypto : הצפנת AES ו־RSA מול הלקוח.
- soc : החיבור מול הלקוח.
- connected : דגל שמציין אם החיבור פעיל.
- ai : המודל הטעון לזיהוי ספרות.
- db_orm : גישה למסד הנתונים של התמונות.

- **מתודות:**

- `__init__(soc, rsa_key)` – פעולה בונה, מקבלת סוקט ומפתח `rsa`
- `run()` – מפעיל Handshake ומאזין ללולאת הבקשות.
- `handshake()` – שולח מפתח ציבורי ומקבל מפתח AES מוצפן מהלקוח.
- `send(*msg)` – שולח הודעה מוצפנת ללקוח.
- `recv()` – מקבל הודעה מהלקוח ומפענח אותה.
- `business_logic()` – מפענח את סוג הבקשה (העלאת תמונה, שליפה לפי ספרה וכו') ומגיב בהתאם.
- `build_return_images_msg(files)` – בונה את הפורמט של ההודעה הכוללת את כל התמונות מהמסד.
- `identify_num(picture_content)` – מבצע עיבוד תמונה, העברת המידע למודל, ושמירה במסד.

ServerCrypto (server.py) - מחלקה

- **תפקיד:** טיפול בהצפנת AES/PKCS1 בצד השרת.

- **תכונות:**

- `rsa_key, aes_key, aes_iv`: מפתחות להצפנה ופענוח.

- **מתודות:**

- `__init__(rsa_key)` – פעולה בונה מקבלת מפתח RSA ומאתחלת את התכונות כדרוש.
- `get_public()` – מחזיר את המפתח הציבורי של השרת. (RSA)
- `decrypt_aes_key(aes_key, aes_iv)` – מפענח את מפתח ה־AES וה־IV שנשלחו מהלקוח.
- `encrypt(plaintext)` – מצפין טקסט בפורמט AES CBC
- `decrypt(encrypted_text)` – מפענח הודעה לפי מפתח AES.

Protocol Functions (protocol.py) - מודול

- **תפקיד:** מנהל את פרוטוקול התקשורת בין השרת ללקוח, כולל קידוד, קידוד בסיסי, שליחה לפי גודל, לוגים.

- **קבועים:**

- `HOST, PORT, SEPARATOR, MAX_FILE_SIZE, OPCODES`

- **פונקציות:**

- `recv_by_size(sock, return_type="bytes")` – מקבל סוקט ואת סוג המידע שאמור לחזור

קורא הודעה לפי גודל ההודעה שמופיע בתחילת ההודעה ומחזיר את מה שקיבל, bytes או string לפי הקלט .

- – `send_by_size(sock, data)`
מקבל סוקט ומידע
שולח נתונים עם prefix של הגודל של ההודעה.
- – `format_message(args)`
מקבל רשימה שדות.
מקודד רשימת שדות למחרוזת אחת בעזרת Base64 (לפי הפרוטוקול), ומחזיר אותה
- – `unformat_message(msg)`
מקבל מחרוזת ארוכה
מפענח את ההודעה לפורמט רשימת שדות. (לפי הפרוטוקול)
- – `__log(prefix, data, max_to_print=100)`
מקבל את ה prefix שאיתו מדפיסים (send או recv), המידע שצריך להדפיס ואת הכמות המקסימלית של תווים שצריך להדפיס.
מדפיס את המידע לפי הקלט.

הסבר אלגוריתמים

Backward של שכבת קונבולוציה

```
def backward(self, output_grad):
    """Computes the gradient of the loss with respect to the input (and with respect to
    the weights and biases).
    - output_grad: The gradient of the loss with respect to the output.
    - Returns the gradient of the loss with respect to the input.
    """

    # reshape the output gradient to match the shape of the patches
    output_grad_flat = output_grad.transpose(0, 2, 3, 1).reshape(-1,
self.out_channels)

    # Calculate the gradient of the loss with respect to weights
    self.weights_gradient = np.dot(output_grad_flat.T, self.patches)
    self.weights_gradient = self.weights_gradient.reshape(self.out_channels,
self.in_channels, self.kh, self.kw)

    # Calculate the gradient of the loss with respect to bias
    # sum of the output gradient (batch-wise)
    self.biases_gradient = np.sum(output_grad_flat, axis=0)

    # Calculate the gradient of the loss with respect to the input (for the next layer)
    # -----
    # Gradient w.r.t. the input ( $\partial L / \partial \text{input}$ ) - Intuition:
    #
    # For each input pixel:
    # - Look at all output pixels that were computed using it.
    # - Each of those output pixels has a gradient telling how much it wants to
    change.
    # - The input pixel affected that output pixel through a specific weight in the filter.
    # - So we multiply the output gradient by that weight (the strength of the
    connection).
    # - Then we sum up all of these contributions.
    #
    # From the output's perspective:
    # - Each output pixel "spreads" its gradient back to the input patch it came from.
    # - It does so proportionally to the filter weights used during the forward pass.
    #
    # The final result tells each input pixel: "Here's how much you need to change to
    reduce the loss."
    # -----
    N, C, H, W = self.input_data.shape
    out_h = (H - self.kh) // self.stride + 1
    out_w = (W - self.kw) // self.stride + 1
    input_grad_patches = np.dot(output_grad_flat,
```

```

self.weights.reshape(self.out_channels, -1))
input_grad_patches = input_grad_patches.reshape(N, out_h, out_w, C, self.kh,
self.kw)
input_grad_patches = input_grad_patches.transpose(0, 3, 4, 5, 1, 2)
# input_grad_patches holds the gradient of the loss with respect to each input
patch.
# Shape: (N, C, kh, kw, out_h, out_w)
#
# For each image in the batch (N), each input channel (C), and each output
location (out_h, out_w),
# this array tells us how much each pixel in the receptive field (defined by the
kernel window kh x kw)
# should change to reduce the loss.
#
# This will be scattered back into the full input gradient image, summing
overlapping contributions.
# -----
input_grad = col2im(input_grad_patches, self.input_data.shape, (self.kh, self.kw),
self.stride, self.padding)
return input_grad

```

כאשר עושים backward מקבלים את הגרדיאנט של loss ביחס לפלט של השורה וצריך לחשב את הגרדיאנט של loss ביחס למשקולות, ביחס להטייה וביחס לקלט (כדי למסור לשכבה הבאה כמו שהשכבה הזו קיבלה), תהליך זה משתמש בכלל השרשרת כדי לחשב איך "להזיז" כל פרמטר כדי שהתוצאה הסופית תהיה כמו שאנחנו רוצים.

קטע הקוד עושה את החישוב הזה עבור שכבת קונבולוציה

1. גרדיאנט ביחס למשקולות: (weights)

עבור כל פילטר בשכבה, נחשב את ההשפעה שהייתה לכל קלט על התוצאה, כפי שהיא באה לידי ביטוי בגרדיאנט שקיבלנו מהשכבה הבאה. לשם כך, אנו ממירים את הקלט (input) לייצוג של "טלאים" (patches) "התואם לגודל הקרנל, כך שכל שורה מייצגת אזור קלט קטן שהופעל עליו פילטר. לאחר מכן, אנו מבצעים מכפלה מטריציונית בין הגרדיאנט של הפלט לבין אותם טלאים. בצורה זו, אנחנו מקבלים את השינוי הרצוי בכל משקל של הפילטרים כדי להפחית את פונקציית העלות.

2. גרדיאנט ביחס להטייה: (bias)

ההטייה משפיעה באופן ישיר על כל פלט של הפילטר. לכן הגרדיאנט ביחס להטייה הוא פשוט סכום של כל גרדיאנטי הפלט – כלומר, כמה כל יציאה תרמה ל- loss, כלל קשר למיקום הספציפי בקלט. זה מתבצע על ידי סכימה על כל הדוגמאות בבאטצ'.

3. גרדיאנט ביחס לקלט: (input)

מטרת חלק זה היא להעביר את הגרדיאנט הלאה לשכבה הקודמת. כדי להבין כמה כל פיקסל קלט תרם לשגיאה, אנו "מפזרים" את גרדיאנט הפלט חזרה לפי מפת ההשפעה שהייתה בקרנלים (המשקולות) על כל אזור קלט. כלומר, עבור כל גרדיאנט פלט, אנו מכפילים אותו במשקולות המתאימות וממקמים את התוצאה באזור הקלט שהשפיע עליו, תוך כדי סכימה של אזורים חופפים. פעולה זו מתבצעת באמצעות פונקציית עזר col2im,

שמפזרת את הגרדיאנטים מהטלאים חזרה למבנה הקלט המקורי, כך שניתן להמשיך ולהעביר את הגרדיאנט לאחור ברשת.

החישוב השלישי מעניין, אפשר להסתכל עליו משני דרכים
לכל פיקסל בקלט:

- הסתכל על כל הפיקסלים ביציאה שחושבו באמצעותו.
- לכל אחד מהפיקסלים ביציאה יש גרדיאנט שמציין עד כמה הוא "רוצה" להשתנות.
- הפיקסל בקלט השפיע על אותו פיקסל ביציאה דרך משקל מסוים במסנן (filter).
- לכן, נכפיל את גרדיאנט היציאה במשקל הזה (שמשקף את עוצמת ההשפעה).
- ואז נסכם את כל התרומות האלו.

מנקודת המבט של היציאה:

- כל פיקסל ביציאה "מפזר" את הגרדיאנט שלו חזרה אל הטלאי בקלט שממנו הוא הגיע.
 - הוא עושה זאת באופן פרופורציונלי למשקולות המסנן ששימשו במהלך המעבר קדימה (forward pass).
- התוצאה של זה תהיה עבור כל פיקסל בקלט, הנה כמה שאתה צריך להשתנות כדי שהloss יעלה.

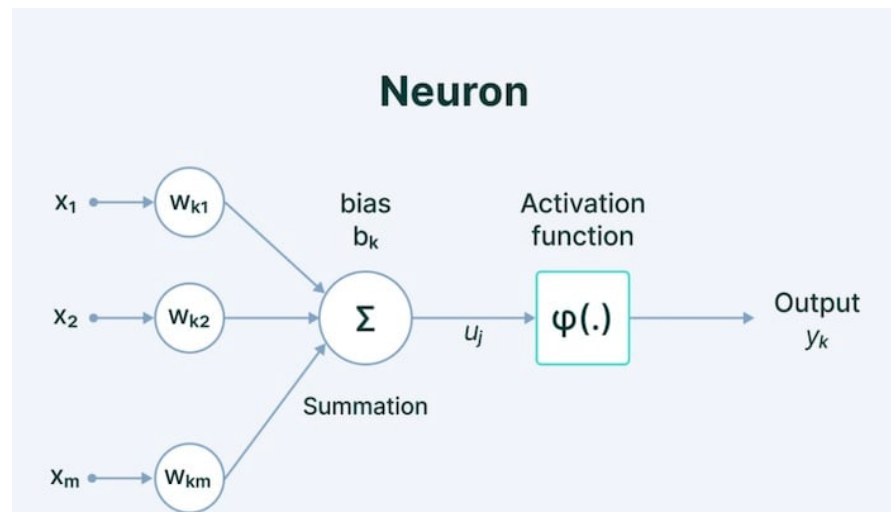
רשתות נוירונים Gradient Descent

נסתכל על רשת נוירונים ואיך מלמדים אותה.

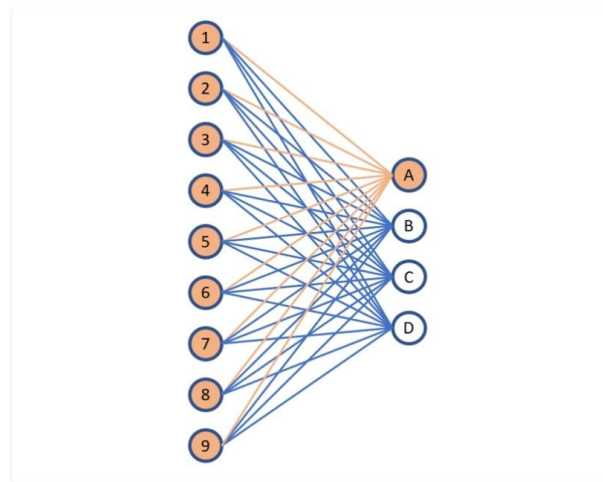
רשת נוירונים היא פונקציה שמורכבת ממספר שכבות. כל אחת מקבלת קלט מהשכבה שלפניה ומוציאה פלט לשכבה שאחריה.

נסתכל על שכבה fully connected (FC) כדי להבין טיפה יותר לעומק.

שכבה FC מורכבת ממספר נוירונים, לכל נוירון יש משקולות (כמות המשקולות כמספר הנוירונים בשכבה הקודמת) והטייה, ואיך שהנוירון מחשב את הפלט שלו זה על ידי סכימת כל הפלטים של הנוירונים של השכבה הקודמת במשקולות של הנוירון בשכבה שלנו (לכ נוירון בשכבה הנוכחית יש משקולות קבועה עבור כל נוירון בשכבה הקודמת) הוספה של ההטייה לסכום הזה, ומעבר לפונקציה אקטיבציה.

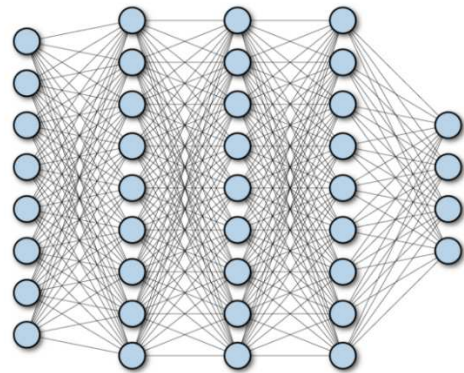


לכן שכבה FC (שמורכבת ממספר ניוונים) תיראה כך:



(המספרים זה הפלט של השכבה הקודמת)

ואם שמים מספר שכבות כאלו אחד אחרי השני זה יראה כך:



כאשר משתמשים ברשתות ניוונים כדי לזהות תמונות הרעיון הכללי הוא כזה: נגיד ויש לי תמונות של ספרות 0-9, ושכבה הראשונה של הראת שלי היא פשוט הערכים של התמונה בעוד שהשכבה האחרונה בגודל 10 מייצגת את הספרות 0-9 (השכבה הראשונה לא מחשבת פלט ואין לה משקולות הפלט שלה הוא פשוט הערכים של יצוג מערכי של התמונה). נגיד ואני שם בשכבה הראשונה שלי (שכבת הקלט) תמונה של 3 אני רוצה שהמשקולות וההטיות שלי יגרמו לחישוב המתמטי ל"הדליק" את התא הרביעי בשכבה האחרונה, ואז אני יכול לזהות ספרות בתמונה.

בשביל שהרשת תצליח לזהות ספרות צריך לשנות את הערכים של המשקולות כך שבסוף כל תמונה של 0 תוציא 0 כל תמונה של 1 תוציא 1 וכן הלאה (מתבסס על זה שיש קשר בין תמונות של אותה הספרה).

כדי לעשות זאת נשתמש באלגוריתם למידה בשם gradient descent. האלגוריתם דורש מאגר של תמונות שאנחנו יודעים מה הספרות בהן. מה שהאלגוריתם הזה עושה בעצם זה מפעיל את רשת הניוונים על תמונה שאני יודע שהיא מספרה מסויימת ואז מסתכלים על הפלט, עכשיו אנחנו יודעים מה הפלט הרצוי, נגדי והתמונה הייתה של 0 אנחנו רוצים שהתא הראשון בשכבה האחרונה יהיה 1 וכל שאר הערכים יהיו 0.

אז ככל שהמרחק בין התוצאה שיצא לי למה שאני רוצה קטן יותר הרשת שלי פועלת יותר טוב! המטרה היא שהמרחק הזה יהיה מינימלי

אפשר להסתכל על המרחק הזה- בין הפלט לפלט הרצוי כפונקציה של כל המשקולות וההטיות ברשת. כלומר אם אני משנה את משקולת מסויימת הפלט שלי ישתנה ולכן גם המרחק הזה ישתנה. נחשוב על זה שניה כפונקציה על מישור, יש את ציר המרחק וציר המשקולת, כרגע למשקולת יש ערך מסויים כלומר נמצאים על נקודה מסוימת בגרף הזה. אם אני אמצא את השיפוע של המשיק לגרף הפונקציה אני אדע כמה אני צריך לרדת ולעלות כדי ללכת לכיוון ערך יותר נמוך בפונקציה (אני מזכיר המטרה היא לגרום למחרק להיות מינימלי) אז אחשב את השיפוע של המשיק של גרף המרחק כפונקציה של כל אחד מהמשקולות/הטיות שיש וזה יתן לי ערכים שאומרים לי כמה אני צריך לשנות כל משקל ומשקל (או הטייה) כדי שהמרחק יהיה יותר קטן. לערכים האלו קוראים גרדיאנט אני אחסיר את ההגרדיאנט מהמשקולות/הטיות וכך שיפרתי את הרשת שלי בזיהוי של ספרה כלשהיא. אחזור על אותו התהליך עם הרבה תמונות שיודעים את הספרה שלהם מראש, ויצא לי רשת מאומנת שאמורה להיות מסוגלת לזהות ספרות.

הסבר הזה היה רק על שכבות FC ולא כלל את החישוב עצמו של הגרדיאנט. הרחבה על החישוב של הגרדיאנט בשכבת CNN (שכבה מסוג אחר שעובדת שונה) אפשר לראות למעלה בהסבר שלי על backward בשכבת קובנלוציה.

מסמך בדיקות

הבדיקות שתכננתי בשלב האפיון

1. העלאת תמונה תקינה
המטרה הייתה לבדוק האם המערכת מזהה תמונה שמכילה ספרה ברורה באופן תקין.
בדקתי את הרשת נוירונים שלי כמו שתיכננתי והרשת הצליחה לזהות את הספרות הברורות.
2. העלאת תמונה באיכות נמוכה/רעש
המטרה הייתה לבדוק האם הרשת עמידה בפני תמונות פחות איכותיות אמנם סיכויי ההצלחה יורדים מעט והרשת לא מצליחה לזהות בכמעט 100 כמו שהרשת מצליחה תמונות ברורות עדיין הרשת הצליחה לזהות את הרוב המוחלט של הספרות גם אם היו בהן רעש או שאיכות התמונה היתה נמוכה יותר
3. העלאת קובץ שאינו תמונה
המטרה היא לבדוק שהמערכת יודעת להתמודד על קלט לא חוקי.
בגלל שטיפלתי באופן ספציפי במקרה הזה השרת שלי זיהה שזה לא קובץ של תמונה ושלה הודעה מתאימה.
4. בדיקת תצוגת ההיסטוריה
המטרה הייתה לוודא שניתן לצפות בתמונות שזוהו בעבר והועלו על ידי משתמשים אחרים. כדי לעשות את זה חיברתי כמה לקוחות והעלתי מהם תמונות, יכולתי לצפות בכל התמונות דרך כל אחד מהלקוחות כפי שהיה דרוש.
5. בדיקת תגובתיות הממשק
המטרה הייתה לבדוק האם הממשק הגרפי עובד בצורה מהירה וחלקה, בדקתי את כל היכולות/כפתורים של הממשק הגרפי וגיליתי בעייה בגלילה עם בעכבר בגלריית תמונות. מסתבר שהקוד שהשתמשתי בו כדי לגלול עם העכבר היה עבור גרסא ישנה יותר של tkinter ולכן החלפתי את הדרך שבה אני גולל עם העכבר ואז הגלילה עבדה כראוי.
6. לנתק את השרת בזמן שהלקוח מחובר אליו ולנסות להפעיל לקוח כששרת לא מחובר והפוך
המטרה הייתה לבדוק האם השרת והלקוח לא קורסים כאשר הצד האחר מתנתק או לא מחובר. ביצעתי את כל הבדיקות וכל הבדיקות עבדו כראוי, הלקוח מתחבר לשרת אם יש שרת שפועל ואם שרת מתנתק הלקוח מחפש שרת חדש. והשרת לא קורס ורושם הערה כאשר לקוח מתנתק ממנו.
7. בדיקת אחסון במסד הנתונים
המטרה הייתה לבדוק שהמידע שאמור להיסמר במסד הנתונים נשמר כראוי. כדי לעשות את זה שלחתי תמונה וראיתי שנוצר עמודה חדשה בטבלה (בעזרת SQLite viewer) וראיתי שנוצרה תמונה חדשה במקום הנכון, לאחר מכן העלתי את אותה התמונה ולא נוצר קובץ חדש/שורה חדשה כפי שרציתי.

הבדיקות נוספות שביצעתי

1. מחקתי קבצים של תמונות שהיו שמורות והיה מעקב אחריהן במסד נתונים המטרה הייתה לבדוק מה קורה במקרה של מחיקה של קבצים שהמערכת חושבת שנמצאים שם גיליתי שנוצר error בשרת ולכן הוספתי בדיקה לפני שאני פונה לקבצים ובנוסף מחקתי שורות במסד נתונים של תמונות שלא נמצאו בתקיה שבה התמונות אמורות להיות שמורות.
2. הזנתי תמונה של תווים דקים ושל תווים שהוזזו המטרה הייתה לבדוק האם המערכת מסוגלת לזהות תווים דקים/שעברו הוזזה המערכת התקשתה לזהות תווים דקים ושעברו הוזזה, כדי להתגבר על זה הגברתי את השינויים שאני עושה למידע שאני מתאמן עליו, הגדלתי את המודל (מגביר יכולת אך מאריך זמן אימון וזמן עיבוד של תצמונות) והוספתי preprocessing שלי חלק שהופך את הכתב להיות טיפה יותר עבה. השינויים האלו שיפרו את הביצועים של המערכת שלי בהרבה.
3. סדקתי את מערכת ההרשמה וההתחברות. בדקתי שהשרת לא שומר תמונות שאני שולח כאשר אני לא מחובר למשתמש, הכנתי משתמש, ניסיתי להכין משתמש עם אותו שם משתמש, נכנסתי למשתמש מלקוח אחר (בדקתי שהוא לא מכניס אותי אם הסיסמה שגויה) ובדקתי שלאחר שאני מחובר למשתמש הוא שומר את התמונות שאני שולח לו. המערכת פעלה כצפוי ונתנה את הפלטים הרצויים.

מדריך למשתמש

כלל קבצי המערכת

כל הקבצים שצריך כדי להריץ את המערכת, בזמן הרצה עלולים להיפתח תקינות וקבצים חדשים (נגיד לשמירה על dataset שהורידו בשביל אימון או בשביל לשמור תמונות בתוך (/server/saved_image

.

```

├── client
│   ├── client.py
│   ├── gui.py
│   ├── main.py
│   └── static
│       ├── browse.png
│       ├── connected.png
│       ├── exit.png
│       ├── no_connection.png
│       ├── login.png
│       └── upload.png
├── protocol.py
├── requirements.txt
└── server
    ├── CNNall
    │   ├── CNN
    │   │   ├── __init__.py
    │   │   ├── data
    │   │   │   ├── __init__.py
    │   │   │   ├── dataloader.py
    │   │   │   └── datasets
    │   │   │       ├── __init__.py
    │   │   │       ├── cifar10.py
    │   │   │       └── mnist.py
    │   │   └── layers
    
```

```

| | | ├── __init__.py
| | | ├── activations
| | | | ├── __init__.py
| | | | ├── relu.py
| | | | ├── sigmoid.py
| | | | └── softmax.py
| | | ├── base.py
| | | ├── conv.py
| | | ├── dense.py
| | | ├── flatten.py
| | | ├── input_layer.py
| | | └── maxpool.py
| | ├── losses
| | | ├── __init__.py
| | | ├── base.py
| | | ├── cce.py
| | | └── mse.py
| | ├── models
| | | ├── __init__.py
| | | └── cnn.py
| | ├── trainers
| | | ├── __init__.py
| | | ├── tester.py
| | | └── trainer.py
| | └── utils
| | | ├── __init__.py
| | | ├── augment.py
| | | ├── initializers.py
| | | ├── optimizer.py
| | └── tensor_patches.py

```

```
|  ├── init.py
|  └── main.py
|── saved_images
|── img_db_orm.py
|── main.py
|── images.db
|── main_model.pkl
└── server.py
```

התקנת המערכת

פירוט הסביבה הנדרשת

כדי להשתמש בפרויקט צריך windows 11/10 וpython3.9.

כל הקבצים נמצאים ב-

<https://github.com/YuvaMendel/FinalSchoolProject/tree/main/src/app>

הספריות שצריך להוריד מצויות ב requirements.txt שנצבא בroot, (כדי להוריד את כל המודולים להריץ בשורת הפקודה: "pip install -r requirements.txt").

כדי להריץ את השרת יש להריץ "python main.py" מתקיית /server

כדי להריץ את הלקוח יש להריץ "python main.py {server_ip}" מתקיית /client (ברירת מחדל של ה ip זה 127.0.0.1) בשורת הפקודה.

מיקומי הקבצים

המבנה הכללי של הפרויקט הוא כזה:

יש תקייה server ו client | קובץ protocol.py אם מריצים רק את השרת אין צורך בתקיית client אבל צריך את protocol.py בתקייה שבה נמצאת התקייה server.

כמו כן לא צריך את תקיית server בשביל להריץ את הלקוח, צריך את protocol.py באותה תקייה שבה התקייה client נמצא בו.

בתוך התקייה server יש קובץ pickle שנקרא main_model.pkl זה המודל לזיהוי תמונה שבו השרת משתמש, כדי לשנות את המודל צריך להחליף את הקובץ הזה.

כדי לאמן מודל חדש צריך להיכנס לCNN בתוך תקיית server ולהריץ שם את main.py (אימון עלול לקחת הרבה זמן, תלוי בכמה גדול המודל וכמה מאמנים אותו, אפשר לשנות את המודל שמאמנים אבל בשביל זה צריך לשנות את main.py)

משתמשי המערכת

יש 2 משתמשים, שרת ולקוח, אחד מנהל מאגר ואחרים פונים אליו לזיהוי תמונות.

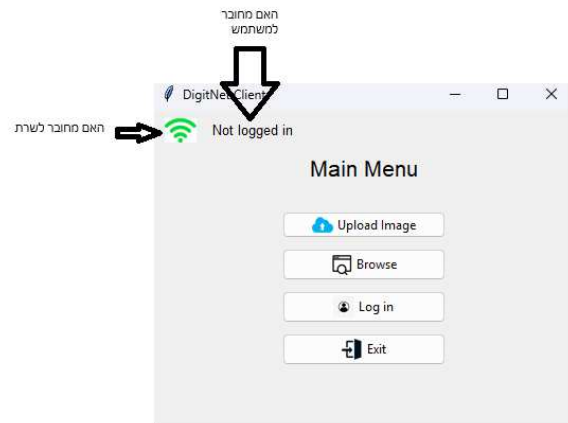
שימוש בלקוח

כדי להריץ את הלקוח יש להריץ "python main.py {server_ip}" מתקיית /client (ברירת מחדל של ה ip זה 127.0.0.1) בשורת הפקודה.

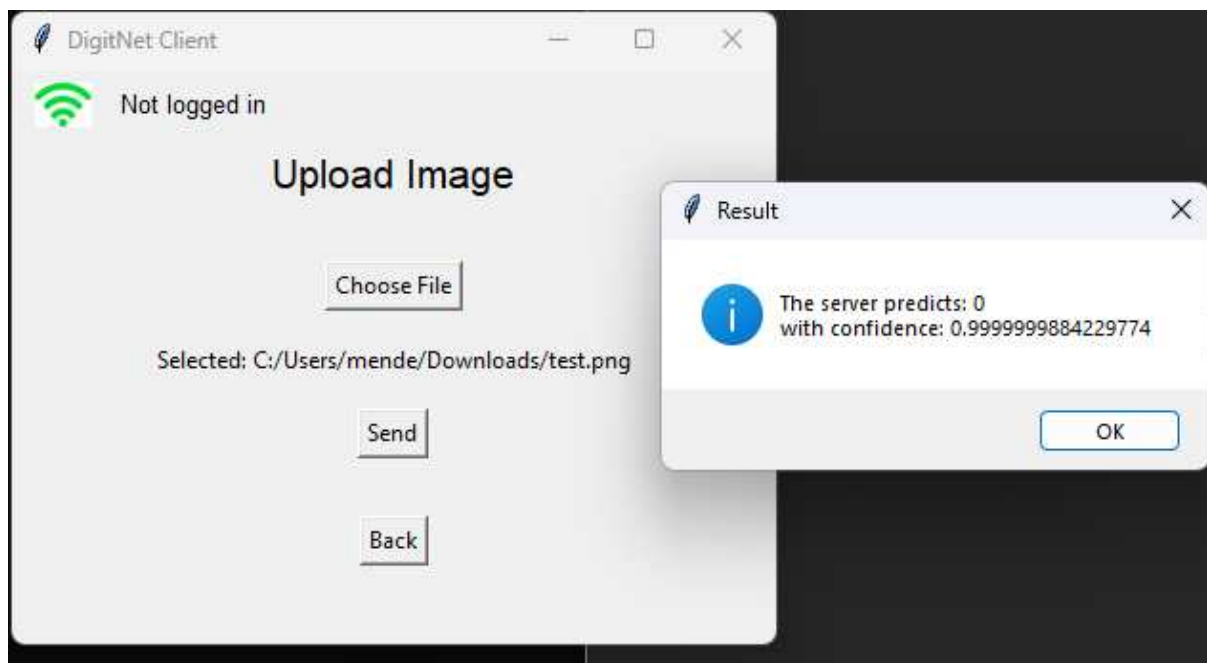
יפתח העמוד הראשי

בצד שמאל למעלה יש מידע על החיבור לשרת (האם המשתמש מחובר לשרת האם הוא התחבר למשתמש)

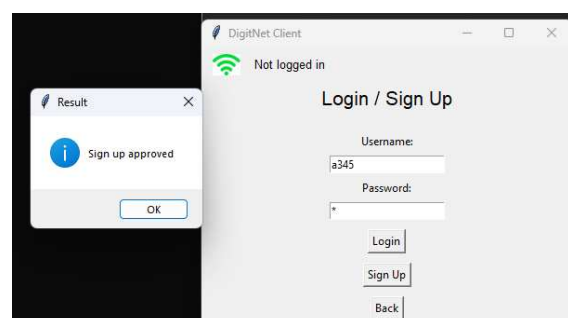
בנוסף לכך יש כפתורים למעבר להעלאת תמונות, צפייה בתמונות שהשרת שמר, והרשמה/התחברות למשתמש.

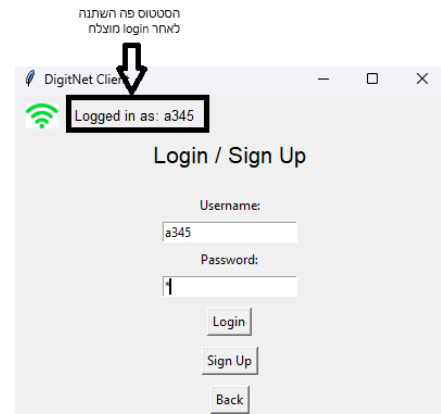


במסך העלאת תמונות אפשר ללחוץ על "Choose File" כדי לפתוח בוחר קבצים ולבחור שם תמונה להעלות. אם נבחרה תמונה אז השם של התמונה יופיע מעל הכפתור send. כדי לשלוח את התמונה לשרת יש ללחוץ send, במקרה הזה תופיע הודעה עם התשובה של השרת. אם המשתמש מחובר התמונה תישמר בשרת. כדי לחזור למסך הראשי יש ללחוץ על כפתור Back

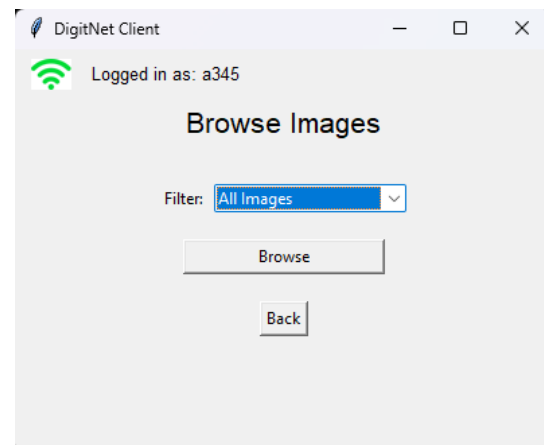


במסך רישום (sign up | log in) אפשר לרשום את השם משתמש והסיסמה ולהירשם/להתחבר למשתמש קיים. אם השם משתמש תפוס/סיסמה לא מתאימה הודעה תעלה בהתאם.



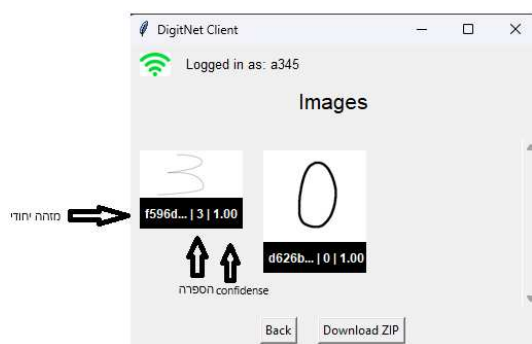


במסך בחירת תמונות אפשר לבחור את הפילטר (או כל תמונה או לבחור ספרה ספציפית). כדי לצפות התמונות יש ללחוץ browse, כדי לחזור למסך הראשי יש ללחוץ Back.



במסך צפייה בתמונות אפשר לצפות בתמונות של החיפוש שנעשה ולחזור למסך הראשי. Download zip הכפתור יאפשר להוריד זיפ של התמונות ביחד עם הזיהוי שהמודל יחס להן.

אפשר לגלול עם הגלגלת של העכבר, מתחת לכל תמונה יופיע 3 פרטים: האותיות הראשונות של המזהה היחודי של התמונה, הספרה שהמודל זיהה בתמונה, ה"ביטחון" (confidence) של המודל בנכונות של התוצאה.



שימוש בשרת

כדי להריץ את השרת יש לכתוב בשורת הפקודה "python main.py" בתקיית /server .
לחיצה על הכפתור "q" יסגור את השרת.

המשתמשים והmetadatan על התמונות ישמרו ב-databases בשם images.db התמונות שנשלחות על ידי לקוחות מחוברים (עשו login) ישמרו בתקיית /server/saved_images .

המודל שבו משתמשים כדי לפענח את התמונות שמור בקובץ /server/main_model.pkl , אם רוצים להחליף את המודל יש להחליף את הקובץ הזה.

כדי לאמן מודל חדש יש להריץ את "python main.py" בתוך תקיית /server/CNNall שיוציא מודל חדש בקובץ main.pkl בתקיית /server/CNNall (כדי להפוך אותו למודל שבו השרת משתמש לשנות את שמו לmain_model.pkl ולהעביר אותו לתקיית /server כל לקוח חדש שיכנס יקבל שירות מהמודל החדש)

אם רוצים לשנות את פרמטרים של האימון אפשר לערוך את main.py ב /server/CNNall

CNNall	03/05/2025 18:51	File folder	
saved_images	23/05/2025 21:12	File folder	
images.db	23/05/2025 21:12	Data Base File	28 KB
img_db_orm.py	15/05/2025 21:42	Python File	7 KB
main.py	15/05/2025 23:45	Python File	1 KB
main_model.pkl	29/04/2025 19:49	PKL File	31,961 KB
server.py	23/05/2025 20:41	Python File	11 KB

רפלקציה

במהלך העבודה על הפרויקט התעסקתי בהרבה תחומים טכנולוגיים, תקשורת בין שרת ללקוח, הצפנה אבטחת מידע, שימוש ברשתות נוירונים לזיהוי תמונות ועוד.

לאורך הדרך נעזרתי במשאבים מגוונים – מדריכים באינטרנט, קורסים קודמים שלמדתי, ולעיתים גם בשאלות שהפנית ל- ChatGPT שסייע לי להבין נושאים לעומק, לנסח קוד בצורה ברורה, ולחדד הסברים שכתובים בתיעוד.

הקושי הכי גדול שהיה לי היה להבין כיצד רשתות נוירונים עובדות בתאוריה וכיצד לממש אותן בפועל. לקח לי הרבה זמן לקרוא ולהבין איך לכתוב כל חלק. לדוגמה, לאחר שהבנתי איך להעביר את הגרדיאנט אחורה בצורה תאורטית ברשת קונבולוציה ניסיתי לממש את זה ונתקלתי בבעיה גדולה, להמיר את הידע התאורטי שרכשתי לקוד היה מטלה מאוד קשה.

אחד הדברים החשובים שלמדתי הוא כיצד לפרק בעיה מורכבת לחלקים קטנים, להבין כל רכיב לעומק, ואז להרכיב אותם יחד לפתרון שלם שעובד.

מעבר לצד הטכנולוגי, למדתי גם רבות על עבודה מסודרת – כתיבת תיעוד, שמירה על קוד קריא ומודולרי, ניהול גרסאות, והצגה של המערכת בצורה שתהיה ברורה גם למי שלא כתב את הקוד.

בראייה לאחור הייתי מיישם בדרך אחרת את השרת והלקוח, אמנם ניסיתי לשמור על מודולריות אך חלק מהקוד של האובייקטים של ההצפנה של השרת והלקוח חופפים ואני חושב שלשלב את 2 המחלקות למחלה משוטפת זה פתרון יותר יפה. בנוסף לכך הייתי משנה את המבנה של הקבצים, את המודול שהכנתי ל-CNN הכנתי בצורה יפה מאוד ואם הייתי מיישם את השרת והלקוח מחדש הייתי מחלק את הקבצים ואת הפונקציות בצורה יותר מסודרת.

במידה והיו ברשותי עוד משאבים הייתי משפר את הפרויקט בכך שהייתי משנה את המודל ללמוד איך לזהות דברים יותר מסובכים מאשר רק ספרות. בגלל החומרה המוגבלת שלי ובגלל שממימשי את הפרויקט ב-numpy (ולא השתמשתי בספרייה כמו pytorch) היכולת שלי לאמן מודל גדול מאוד מוגבלת. אם היו לי יותר משאבים הייתי בונה מערכת שמזהה דברים יותר מורכבים כמו משפטים או ממש ממירות תמונות של מסמכים לטקסט (OCR).

לסיכום, הפרויקט הזה היווה עבורי הזדמנות אמיתית לצמיחה אישית ומקצועית. הוא חיבר בין ידע תאורטי לבין יישום מעשי, ואתגרים טכנולוגיים הפך להזדמנויות למידה. אני מרגיש שהצלחתי לפתח לא רק מערכת שעובדת, אלא גם דרך חשיבה ויכולת פתרון בעיות שישמשו אותי בהמשך הדרך, בכל פרויקט עתידי שאקח בו חלק.

בבליוגרפיה

3Blue1Brown. (2017, October 5). *But what is a neural network? | Deep learning chapter 1* [Video]. YouTube. <https://www.youtube.com/watch?v=aircAruvnKk>

3Blue1Brown. (2017, October 16). *Gradient descent, how neural networks learn | Deep learning chapter 2* [Video]. YouTube. <https://www.youtube.com/watch?v=IHZwWFHWa-w>

3Blue1Brown. (2017, November 3). *What is backpropagation really doing? | Deep learning chapter 3* [Video]. YouTube. <https://www.youtube.com/watch?v=Ilg3gGewQ5U>

3Blue1Brown. (2017, November 3). *Backpropagation calculus | Deep learning chapter 4* [Video]. YouTube. <https://www.youtube.com/watch?v=tIeHLnjs5U8>

Nielsen, M. A. (2015). *Chapter 1: Using neural nets to recognize handwritten digits. In Neural networks and deep learning.* <http://neuralnetworksanddeeplearning.com/chap1.html>

IBM Technology. (2021, October 6). *What are Convolutional Neural Networks (CNNs)?* [Video]. YouTube. <https://www.youtube.com/watch?v=QzY57FaENXg>

נספחים

■ protocol.py

```
import base64
# General Protocol Constants
HOST = "127.0.0.1" # Server address
PORT = 6627 # Communication port
SIZE_OF_SIZE = 7 # Size of the size field in the beginning of the message
MAX_FILE_SIZE = 3000000 # Maximum file size (1MB)
CLIENT_TIMEOUT = 2 # Timeout for client operations in seconds
SEPERATOR = '~' # Seperator for the fields of the message
# Message Codes (OPCODES)
ACK_START = 'GKSC' # server got key from client and is ready to start communication
REQUEST_IMAGE = 'RIPP' # client request image recognition
IMAGE_IDENTIFIED = 'RIPR' # server identified image
ERROR = 'ERRR' # server error
REQUEST_IMAGES = 'RIHP' # client requests images that have been recognized (from the database)
REQUEST_IMAGES_BY_DIGIT = 'RIHD' # client requests images that have been recognized by digit

RETURN_IMAGES = 'RIHL' # server returns images from database (starts return process)
IMAGE_FILE_RETURN = 'RILF' # returns a image file (the "RETURN_IMAGES" send the amount of "IMAGE_FILE_RE
RETURN_FILES_END = 'RIHE' # end of the return process

SIGN_UP_REQUEST = 'CRSU' # client requests to sign up
SIGN_UP_APPROVED = 'CRSA' # server approved the sign up request
SIGN_UP_DENIED = 'CRSD' # server denied the sign up request
LOG_IN_REQUEST = 'CRSI' # client requests to sign in
LOG_IN_APPROVED = 'CRLA' # server approved the sign in request
LOG_IN_DENIED = 'CRFD' # server denied the sign in request

# Error Codes
error_messages = {"1": "Image format not recognized/supported",
                  "2": "File is too large",
                  "3": "Invalid request",
                  "4": "General_error"}

DEBUG_FLAG = True

def __recv_amount(sock, size=SIZE_OF_SIZE):
    """
    Receive a specific amount of data from a socket.
    :param sock: a socket to receive data from
    :param size: size of the data to receive, default is SIZE_OF_SIZE
    :return: data received from the socket as bytes
    """
    buffer = b''
    while size:
        new_bufffer = sock.recv(size)
        if not new_bufffer:
            return b''
        buffer += new_bufffer
        size -= len(new_bufffer)
    return buffer

def recv_by_size(sock, return_type="bytes"):
    """
    Receive data from a socket by size.
    :param sock: socket to receive data from
    :param return_type: type of data to return, either "bytes" or "string"
    :return: data received from the socket, either as bytes or string
    """
    data = b''
    data_len = int(__recv_amount(sock, SIZE_OF_SIZE))
    # code handle the case of data_len 0
    data = __recv_amount(sock, data_len)
    __log("Receive", data)
    if return_type == "string":
        return data.decode()
    return data
```

```

def send_by_size(sock, data, max_chunk_size=4096):
    """
    Send data to a socket by size.
    :param sock: socket to send data to
    :param data: data to send, can be a string or bytes
    :param max_chunk_size: chunk size to send data in, default is 4096 bytes
    :return:
    """
    if len(data) == 0:
        return
    if type(data) != bytes:
        data = data.encode()
    len_data = str(len(data)).zfill(SIZE_OF_SIZE).encode()
    data = len_data + data
    total_sent = 0
    while total_sent < len(data):
        end = min(total_sent + max_chunk_size, len(data))
        chunk = data[total_sent:end]
        sent = sock.send(chunk)
        if sent == 0:
            raise RuntimeError("socket connection broken")
        total_sent += sent
    __log("Sent", data)

def format_message(args):
    """
    Format a message by encoding each argument in base64 and joining them with SEPERATOR.
    :param args: arguments to format, can be a list or tuple of strings or bytes
    :return: string with base64 encoded parts separated by SEPERATOR
    """
    args = list(args)
    for i in range(len(args)):
        if type(args[i]) == str:
            args[i] = args[i].encode()

    base64_args = [base64.b64encode(arg).decode() for arg in args]
    return SEPERATOR.join(base64_args)

def unformat_message(msg):
    """
    Unformat a message that was formatted with format_message.
    :param msg: the message to unformat, should be a string with base64 encoded parts separated by SEPERATOR
    :return: a list of bytes, each element is the decoded base64 string from the message
    """
    split_msg = msg.split(SEPERATOR)
    return [(base64.b64decode(s.encode())) for s in split_msg]

def __log(prefix, data, max_to_print=100):
    """
    Log data to the console if DEBUG_FLAG is set.
    :param prefix: prefix for the log message
    :param data: data to log, can be a string or bytes
    :param max_to_print: maximum number of characters to print from the data
    :return:
    """
    if not DEBUG_FLAG:
        return
    data_to_log = data[:max_to_print]
    if type(data_to_log) == bytes:
        try:
            data_to_log = data_to_log.decode()
        except (UnicodeDecodeError, AttributeError):
            pass
    print(f"\n{prefix} ({len(data)})>>>{data_to_log}")

```



```

        self.connected = False
        self.gui_callback.display_result("Server disconnected", message_type="error")
        break

def recv_files_process(self, amount_of_files):
    """
    This function is used to receive files from the server
    :param amount_of_files: the number of files to receive
    :return: None
    """
    images = []
    finished_process = False
    for i in range(amount_of_files):
        msg = self.recv()
        opcode = msg[0].decode()
        if opcode == protocol.RETURN_FILES_END:
            finished_process = True
            break
        elif opcode == protocol.IMAGE_FILE_RETURN:
            image_list = msg[1:]
            id = image_list[0].decode()
            image_content = image_list[1]
            image_file = io.BytesIO(image_content)
            image = Image.open(image_file)
            digit = image_list[2].decode()
            confidence = float(image_list[3].decode())
            images.append((id, image, digit, confidence))
        else:
            raise ValueError(f"Unexpected opcode: {opcode}")
    if not finished_process:
        msg = self.recv()
        opcode = msg[0].decode()
        if opcode == protocol.RETURN_FILES_END:
            finished_process = True
    return images

def business_logic(self, response):
    """ This function processes the response from the server based on the opcode. """
    opcode = response[0].decode()
    if opcode == protocol.IMAGE_IDENTIFIED:
        self.gui_callback.display_result("The server predicts: " + response[1].decode() + "\nwith con
    if opcode == protocol.RETURN_IMAGES:
        images = self.recv_files_process(int(response[1].decode()))

        self.gui_callback.display_images(images)
    if opcode == protocol.ERROR:
        error_code = response[1].decode()
        if error_code in protocol.error_messages:
            error_message = protocol.error_messages[error_code]
        else:
            error_message = "Unknown error"
        self.gui_callback.display_result(error_message, message_type="error")
    if opcode == protocol.LOG_IN_APPROVED:
        self.gui_callback.gui_set_logged_in_user(response[1].decode())
    if opcode == protocol.LOG_IN_DENIED:
        self.gui_callback.display_result("Log in denied", message_type="error")
    if opcode == protocol.SIGN_UP_APPROVED:
        self.gui_callback.display_result("Sign up approved", message_type="result")
    if opcode == protocol.SIGN_UP_DENIED:
        self.gui_callback.display_result("Sign up denied- username is already taken", message_type="e

@staticmethod
def convert_image_string_to_tuple(image_string_list):
    """
    converts the list of strings representing images to a list of tuples, that makes it easier to wor
    """
    img_lst = []
    for i in range(0, len(image_string_list), 4):
        id = image_string_list[i].decode()

```

```

        image_content = image_string_list[i+1]
        image_file = io.BytesIO(image_content)
        image = Image.open(image_file)
        digit = image_string_list[i+2].decode()
        confidence = float(image_string_list[i+3].decode())
        img_lst.append((id, image, digit, confidence))
    return img_lst

def send_file(self, file_path):
    """puts a task to the request queue to send a file to the server."""
    self.queue_task(protocol.REQUEST_IMAGE, file_path)

def request_sign_up(self, username, password):
    """puts a task to the request queue to sign up a new user."""
    self.queue_task(protocol.SIGN_UP_REQUEST, username, password)

def request_log_in(self, username, password):
    """puts a task to the request queue to log in a user."""
    self.queue_task(protocol.LOG_IN_REQUEST, username, password)

def request_images(self, digit=None):
    """puts a task to the request queue to request images from the server."""
    if digit is None:
        self.queue_task(protocol.REQUEST_IMAGES)
    else:
        self.queue_task(protocol.REQUEST_IMAGES_BY_DIGIT, digit)

def handle_task(self, task_code, args):
    """
    Handle the task based on the task code and arguments.
    :param task_code: a string representing the task code
    :param args: the arguments for the task
    :return: should the client recv a message
    """
    if task_code == protocol.REQUEST_IMAGE:
        file_name = args[0].split('/')[-1]
        if not os.path.exists(args[0]):
            self.gui_callback.display_result(f"File {args[0]} does not exist.", message_type="error")
            return False
        with open(args[0], 'rb') as file:
            file_content = file.read()
        if len(file_content) > protocol.MAX_FILE_SIZE:
            self.gui_callback.display_result(f"File {args[0]} is too large", message_type="error")
            return False

        self.send(protocol.REQUEST_IMAGE, file_name, file_content)
    if task_code == protocol.REQUEST_IMAGES:
        self.send(protocol.REQUEST_IMAGES)
    if task_code == protocol.REQUEST_IMAGES_BY_DIGIT:
        digit = args[0]
        self.send(protocol.REQUEST_IMAGES_BY_DIGIT, digit)
    if task_code == protocol.SIGN_UP_REQUEST:
        username = args[0]
        password = args[1]
        if len(username) == 0 or len(password) == 0:
            self.gui_callback.display_result("Username or password cannot be empty", message_type="error")
            return False
        self.send(protocol.SIGN_UP_REQUEST, username, password)
    if task_code == protocol.LOG_IN_REQUEST:
        username = args[0]
        password = args[1]
        if len(username) == 0 or len(password) == 0:
            self.gui_callback.display_result("Username or password cannot be empty", message_type="error")
            return False
        self.send(protocol.LOG_IN_REQUEST, username, password)
    return True

def is_connected(self):
    return self.connected

def close(self):

```



```

        self.connected = False
        self.request_queue.put(None)
        self.sock.close()

    def send(self, *msg):
        """ Sends a message to the server after formatting and encrypting it."""
        msg = protocol.format_message(msg)

        protocol.send_by_size(self.sock, self.crypto.encrypt(msg))

    def recv(self):
        """ Receives a message from the server, decrypts it, and unformats it."""
        return protocol.unformat_message(self.crypto.decrypt(protocol.recv_by_size(self.sock)))

class ClientCrypto:
    """ A class to handle the encryption and decryption of messages using AES and RSA."""
    def __init__(self):
        self.aes_key = os.urandom(32)
        self.aes_iv = os.urandom(16)

    def encrypted_key_iv(self, rsa_key):
        """ Encrypts the AES key and IV using the provided RSA public key."""
        rsa_key = RSA.import_key(rsa_key)
        cipher_rsa = PKCS1_OAEP.new(rsa_key)
        encrypted_aes_key = cipher_rsa.encrypt(self.aes_key)
        aes_iv = self.aes_iv
        return encrypted_aes_key, aes_iv

    def encrypt(self, plaintext):
        """ Encrypts the plaintext using AES in CBC mode with padding."""
        cipher = AES.new(self.aes_key, AES.MODE_CBC, self.aes_iv)
        padded_data = pad(plaintext.encode(), AES.block_size)
        ciphertext = cipher.encrypt(padded_data)

        return ciphertext

    def decrypt(self, encrypted_text):
        """ Decrypts the encrypted text using AES in CBC mode with unpadding."""
        cipher = AES.new(self.aes_key, AES.MODE_CBC, self.aes_iv)
        decrypted_padded = cipher.decrypt(encrypted_text)
        plaintext = unpad(decrypted_padded, AES.block_size)

        return plaintext.decode()

```

■ client\gui.py

```

import tkinter as tk
from tkinter import filedialog, messagebox, ttk
import zipfile, io, csv
from PIL import Image, ImageTk
from client import Client

```

```

class ClientGUI:

    def __init__(self):
        self.root = tk.Tk()
        self.exit = False
        self.root.title("DigitNet Client")
        self.root.geometry("400x300")
        self.client = None
        self.logged_in_user = None

        self.load_icons()
        self.no_internet_img = None
        self.connected_img = None
        self.connection_label = None
        self.status_frame = None

```

```

self.login_status_label = None

self.load_images()
self.create_main_screen()
self.file_label = None
self.root.protocol("WM_DELETE_WINDOW", self.exit_gui)

def load_icons(self):
    self.browse_icon = ImageTk.PhotoImage(Image.open("static/browse.png").resize((20, 20)))
    self.upload_icon = ImageTk.PhotoImage(Image.open("static/upload.png").resize((20, 15)))
    self.exit_icon = ImageTk.PhotoImage(Image.open("static/exit.png").resize((20, 20)))
    self.login_icon = ImageTk.PhotoImage(Image.open("static/login.png").resize((20, 20)))

def load_images(self):
    no_net = Image.open("static/no_connection.png")
    connected = Image.open("static/connected.png")
    no_net.thumbnail((30, 30))
    connected.thumbnail((30, 30))
    self.no_internet_img = ImageTk.PhotoImage(no_net)
    self.connected_img = ImageTk.PhotoImage(connected)

def update_connection_status(self):
    if self.connection_label and self.connection_label.winfo_exists():
        if self.client is None or not self.client.is_connected():
            self.connection_label.config(image=self.no_internet_img)
            self.update_login_status(None)
        else:
            self.connection_label.config(image=self.connected_img)

def start_connection_polling(self, interval_ms=2000):
    self.update_connection_status()
    self.root.after(interval_ms, self.start_connection_polling)

def create_status_frame(self):
    if self.status_frame:
        self.status_frame.destroy()

    self.status_frame = tk.Frame(self.root)
    self.status_frame.place(x=5, y=5)

    self.connection_label = tk.Label(self.status_frame)
    self.connection_label.pack(side=tk.LEFT, padx=5)

    self.login_status_label = tk.Label(self.status_frame, text="Not logged in", font=("Arial", 10))
    self.login_status_label.pack(side=tk.LEFT, padx=5)

    self.update_login_status(self.logged_in_user)
    self.update_connection_status()

def update_login_status(self, username):
    self.logged_in_user = username
    if self.login_status_label and self.login_status_label.winfo_exists():
        if username:
            self.login_status_label.config(text=f"Logged in as: {username}")
        else:
            self.login_status_label.config(text="Not logged in")

def create_main_screen(self):
    self.file_path = None
    for widget in self.root.winfo_children():
        widget.destroy()

    self.create_status_frame() # Moved to top

    # Now the title label is pushed below the status bar
    tk.Label(self.root, text="Main Menu", font=("Arial", 16)).pack(pady=(40, 20))

    ttk.Button(self.root, text="Upload Image", image=self.upload_icon, compound="left",
               command=self.open_upload_screen, width=20).pack(pady=5)
    ttk.Button(self.root, text="Browse", image=self.browse_icon, compound="left",
               command=self.open_browse_screen, width=20).pack(pady=5)

```

```

        ttk.Button(self.root, text="Log in", image=self.login_icon, compound="left",
                    command=self.open_login_screen, width=20).pack(pady=5)
        ttk.Button(self.root, text="Exit", image=self.exit_icon, compound="left",
                    command=self.exit_gui, width=20).pack(pady=5)

    self.create_status_frame()

def open_login_screen(self):
    for widget in self.root.winfo_children():
        widget.destroy()

    tk.Label(self.root, text="Login / Sign Up", font=("Arial", 16)).pack(pady=(40,20))

    tk.Label(self.root, text="Username:").pack()
    username_entry = tk.Entry(self.root)
    username_entry.pack(pady=5)

    tk.Label(self.root, text="Password:").pack()
    password_entry = tk.Entry(self.root, show="*")
    password_entry.pack(pady=5)

    def login():
        username = username_entry.get().strip()
        password = password_entry.get().strip()
        if self.client:
            self.client.request_log_in(username, password)
        else:
            messagebox.showerror("Error", "Client not initialized.")

    def signup():
        username = username_entry.get().strip()
        password = password_entry.get().strip()
        if self.client:
            self.client.request_sign_up(username, password)
        else:
            messagebox.showerror("Error", "Client not initialized.")

    tk.Button(self.root, text="Login", command=login).pack(pady=5)
    tk.Button(self.root, text="Sign Up", command=signup).pack(pady=5)
    tk.Button(self.root, text="Back", command=self.create_main_screen).pack(pady=5)

    self.create_status_frame()

def gui_set_logged_in_user(self, username):
    self.update_login_status(username)

def exit_gui(self):
    self.exit = True
    if self.client is not None:
        self.client.close()
    self.root.quit()

def open_upload_screen(self):
    for widget in self.root.winfo_children():
        widget.destroy()

    tk.Label(self.root, text="Upload Image", font=("Arial", 16)).pack(pady=(40,20))

    self.upload_button = tk.Button(self.root, text="Choose File", command=self.upload_image)
    self.upload_button.pack(pady=10)

    self.file_label = tk.Label(self.root, text="No file selected", wraplength=350)
    self.file_label.pack(pady=5)

    self.submit_button = tk.Button(self.root, text="Send", command=self.send_image)
    self.submit_button.pack(pady=10)

    tk.Button(self.root, text="Back", command=self.create_main_screen).pack(pady=20)

    self.create_status_frame()
def upload_image(self):

```

```

        self.file_path = filedialog.askopenfilename(filetypes=[("Image Files", "*.png;*.jpg;*.jpeg")])
    if self.file_path:
        self.file_label.config(text=f"Selected: {self.file_path}")

def send_image(self):
    if hasattr(self, 'file_path') and self.file_path:
        if self.client is not None:
            self.client.send_file(self.file_path)
        else:
            messagebox.showerror("Error", "Client not initialized.")
    else:
        messagebox.showerror("Error", "No file selected!")

def open_browse_screen(self):
    for widget in self.root.winfo_children():
        widget.destroy()

    tk.Label(self.root, text="Browse Images", font=("Arial", 16)).pack(pady=(40,20))

    filter_frame = tk.Frame(self.root)
    filter_frame.pack(pady=10)

    tk.Label(filter_frame, text="Filter:").pack(side=tk.LEFT, padx=(0, 5))

    self.filter_var = tk.StringVar(value="All Images")
    options = ["All Images"] + [str(i) for i in range(10)]

    self.filter_dropdown = ttk.Combobox(filter_frame, textvariable=self.filter_var, values=options,
                                         state="readonly")
    self.filter_dropdown.pack(side=tk.LEFT)

    tk.Button(self.root, text="Browse", command=self.browse_images, width=20).pack(pady=10)
    tk.Button(self.root, text="Back", command=self.create_main_screen).pack(pady=10)

    self.create_status_frame()

def browse_images(self):
    if self.client is not None:
        digit = self.filter_var.get()
        if digit == "All Images":
            self.client.request_images()
        else:
            self.client.request_images(digit=digit)
    else:
        messagebox.showerror("Error", "Client not initialized.")

def display_images(self, images):
    self.current_images = images

    for widget in self.root.winfo_children():
        widget.destroy()

    self.image_refs = []

    container = tk.Frame(self.root)
    container.grid(row=0, column=0, sticky="nsew")
    self.root.grid_rowconfigure(0, weight=1)
    self.root.grid_columnconfigure(0, weight=1)

    tk.Label(container, text="Images", font=("Arial", 16)).grid(row=0, column=0, columnspan=2, pady=10)

    canvas = tk.Canvas(container)
    scrollbar = tk.Scrollbar(container, orient="vertical", command=canvas.yview)
    scrollable_frame = tk.Frame(canvas)

    def update_scrollregion_delayed():
        canvas.configure(scrollregion=canvas.bbox("all"))

    scrollable_frame.bind("<Configure>", lambda e: self.root.after(100, update_scrollregion_delayed))

    canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")

```

```

canvas.configure(yscrollcommand=scrollbar.set)

canvas.grid(row=1, column=0, sticky="nsew")
scrollbar.grid(row=1, column=1, sticky="ns")
container.grid_rowconfigure(1, weight=1)
container.grid_columnconfigure(0, weight=1)

max_width, max_height = 100, 100
min_canvas_width = 100
columns = 3
row = 0
col = 0

for idx, (img_id, img_pil, digit, confidence) in enumerate(images):
    img_copy = img_pil.copy()
    img_copy.thumbnail((max_width, max_height))
    img_tk = ImageTk.PhotoImage(img_copy)
    self.image_refs.append(img_tk)

    short_id = img_id if len(img_id) <= 5 else img_id[:5] + "..."
    info_text = f"{short_id} | {digit} | {confidence:.2f}"

    frame = tk.Frame(scrollable_frame, padx=5, pady=5)
    frame.grid(row=row, column=col, padx=5, pady=5, sticky="n")

    canvas_width = max(min_canvas_width, img_copy.width)

    canvas_img = tk.Canvas(frame, width=canvas_width, height=img_copy.height + 30, highlightthickness=1)
    canvas_img.pack()
    canvas_img.create_image(0, 0, image=img_tk, anchor="nw")
    canvas_img.create_rectangle(0, img_copy.height, canvas_width, img_copy.height + 30, fill="black",
                                outline="black")
    canvas_img.create_text(5, img_copy.height + 15, anchor="w", text=info_text,
                           fill="white", font=("Arial", 9, "bold"))

    col += 1
    if col >= columns:
        col = 0
        row += 1

def _on_mousewheel(event):
    try:
        if canvas.winfo_exists():
            canvas.yview_scroll(int(-1 * (event.delta / 60)), "units")
    except Exception:
        pass

def bind_scroll():
    self.root.bind_all("<MouseWheel>", _on_mousewheel)

def unbind_scroll():
    self.root.unbind_all("<MouseWheel>")

bind_scroll()
self.root.bind("<Destroy>", lambda e: unbind_scroll())

btn_frame = tk.Frame(self.root)
btn_frame.grid(row=1, column=0, pady=10)

tk.Button(btn_frame, text="Back", command=lambda: [unbind_scroll(), self.create_main_screen()]).pack(
    side=tk.LEFT, padx=10)
tk.Button(btn_frame, text="Download ZIP", command=self.download_zip).pack(side=tk.RIGHT, padx=10)

self.create_status_frame()

def download_zip(self):
    if not hasattr(self, "current_images") or not self.current_images:
        messagebox.showinfo("Info", "No images to export.")
        return

    zip_path = filedialog.asksaveasfilename(

```

```

        defaultextension=".zip",
        filetypes=[("ZIP files", "*.zip")],
        title="Save ZIP Archive"
    )

    if not zip_path:
        return

    try:
        with zipfile.ZipFile(zip_path, 'w') as zipf:
            csv_buffer = io.StringIO()
            csv_writer = csv.writer(csv_buffer)
            csv_writer.writerow(["filename", "label", "confidence"])

            for img_id, img_pil, digit, confidence in self.current_images:
                filename = f"{img_id}.png"
                img_bytes = io.BytesIO()
                img_pil.save(img_bytes, format="PNG")
                img_bytes.seek(0)
                zipf.writestr(f"images/{filename}", img_bytes.read())
                csv_writer.writerow([filename, digit, f"{confidence:.4f}"])

            zipf.writestr("labels.csv", csv_buffer.getvalue())

            messagebox.showinfo("Success", f"ZIP file saved to:\n{zip_path}")
    except Exception as e:
        messagebox.showerror("Error", f"Failed to save ZIP: {str(e)}")

    def display_result(self, message, message_type="result"):
        if message_type == "result":
            messagebox.showinfo("Result", message)
        elif message_type == "error":
            messagebox.showerror("Error", message)

    def handle_server_response(self, response):
        self.display_result(response)

    def activate(self):
        self.root.after(0, self.start_connection_polling)
        self.root.mainloop()

```

■ client\main.py

```

import client
import gui
from time import sleep
import threading
from sys import argv

def main():
    dest_ip = "127.0.0.1"
    if len(argv) > 1:
        dest_ip = argv[1]
    finished = False
    app = gui.ClientGUI()

    def start_client():
        nonlocal finished
        while not finished:
            connection = client.Client(dest_ip)
            try:
                connection.connect()
            except Exception as e:
                print(f"could not connect to server: {e}")
                sleep(0.5)
                continue

        connection.start()

    start_client()

```

```

        connection.gui_callback = app
        app.client = connection
        connection.join()
        finished = app.exit
    threading.Thread(target=start_client, daemon=True).start()
    app.activate()

```

```

if __name__ == '__main__':
    main()

```

■ server\img_db_orm.py

```

import sqlite3
import os
import uuid
import hashlib
import secrets
from PIL import Image
import io

image_limit = 100

class ImagesORM:
    """A class to handle image storage and retrieval using SQLite ORM."""
    def __init__(self, db_path='images.db', image_dir='saved_images'):
        self.db_path = db_path
        self.image_dir = image_dir
        self.conn = None
        self.cursor = None
        os.makedirs(self.image_dir, exist_ok=True)

    def open_DB(self):
        """Open a connection to the SQLite database."""
        self.conn = sqlite3.connect(self.db_path)
        self.cursor = self.conn.cursor()

    def close_DB(self):
        """Close the database connection if it is open."""
        if self.conn:
            self.conn.close()

    def commit(self):
        """Commit the current transaction to the database."""
        self.conn.commit()

    def create_tables(self):
        """Create the necessary tables in the database if they do not exist."""
        self.open_DB()

        # Create Users table
        self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS Users (
            user_id TEXT PRIMARY KEY,
            username TEXT UNIQUE,
            password_hash TEXT,
            salt TEXT
        )
        ''')

        # Create Images table with user_id column
        self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS Images (
            image_id TEXT PRIMARY KEY,
            digit TEXT,
            path TEXT,
            confidence REAL,
            hash TEXT UNIQUE,

```

```

        user_id TEXT
    )
    '')

    self.commit()
    self.close_DB()

def register_user(self, username, password):
    """Register a new user with a username and password, returning the user_id if successful."""
    self.open_DB()
    salt = secrets.token_hex(16)
    password_hash = hashlib.sha256((password + salt).encode()).hexdigest()
    user_id = str(uuid.uuid4())

    try:
        self.cursor.execute('''
            INSERT INTO Users (user_id, username, password_hash, salt)
            VALUES (?, ?, ?, ?)
            ''', (user_id, username, password_hash, salt))
        self.commit()
    except sqlite3.IntegrityError:
        user_id = None # Username already exists
    self.close_DB()
    return user_id

def authenticate_user(self, username, password):
    """Authenticate a user by checking the username and password against the database."""
    self.open_DB()
    self.cursor.execute('SELECT user_id, password_hash, salt FROM Users WHERE username = ?', (username,))
    result = self.cursor.fetchone()
    self.close_DB()

    if result:
        user_id, stored_hash, salt = result
        computed_hash = hashlib.sha256((password + salt).encode()).hexdigest()
        if computed_hash == stored_hash:
            return user_id
    return None

def save_image_file(self, image_bytes, max_size=256):
    """Resize an image, compute its SHA-256 hash, and save it to the disk."""
    img = Image.open(io.BytesIO(image_bytes))
    img = img.convert("RGB")
    img.thumbnail((max_size, max_size), Image.Resampling.LANCZOS)

    output = io.BytesIO()
    img.save(output, format='PNG')
    resized_bytes = output.getvalue()

    # Compute SHA-256 hash of resized image
    hash_val = hashlib.sha256(resized_bytes).hexdigest()

    image_id = str(uuid.uuid4())
    filename = f"{image_id}.png"
    path = os.path.join(self.image_dir, filename)

    with open(path, 'wb') as f:
        f.write(resized_bytes)

    return image_id, path, hash_val

def insert_image(self, image_id, digit, path, confidence, hash_val, user_id):
    """Insert an image record into the database, checking for duplicates based on hash."""
    self.open_DB()

    # Check for duplicate based on hash
    self.cursor.execute('SELECT image_id, path FROM Images WHERE hash = ?', (hash_val,))
    result = self.cursor.fetchone()

    if result:
        existing_image_id, existing_path = result

```



```

        if not os.path.exists(existing_path):
            # File is missing, remove old database record
            self.cursor.execute('DELETE FROM Images WHERE image_id = ?', (existing_image_id,))
        else:
            self.close_DB()
            return # File exists, skip insertion

self.cursor.execute('''
INSERT INTO Images (image_id, digit, path, confidence, hash, user_id)
VALUES (?, ?, ?, ?, ?, ?)
''', (image_id, digit, path, confidence, hash_val, user_id))

# Keep only the most recent `image_limit` entries
self.cursor.execute('''
DELETE FROM Images
WHERE image_id NOT IN (
    SELECT image_id FROM Images
    ORDER BY rowid DESC
    LIMIT ?
)
''', (image_limit,))

self.commit()
self.close_DB()

def delete_old_files(self):
    """Delete old image files that are not in the most recent `image_limit` entries."""
    self.open_DB()
    self.cursor.execute('SELECT image_id, path FROM Images ORDER BY rowid DESC LIMIT ?', (image_limit,))
    rows = self.cursor.fetchall()
    self.close_DB()

    recent_paths = set()
    missing_image_ids = []

    for image_id, path in rows:
        if os.path.exists(path):
            recent_paths.add(path)
        else:
            missing_image_ids.append(image_id)

    all_paths = set(os.path.join(self.image_dir, f) for f in os.listdir(self.image_dir))
    to_delete = all_paths - recent_paths

    for path in to_delete:
        try:
            os.remove(path)
        except FileNotFoundError:
            pass

    if missing_image_ids:
        self.open_DB()
        self.cursor.executemany('DELETE FROM Images WHERE image_id = ?',
                                [(img_id,) for img_id in missing_image_ids])

        self.commit()
        self.close_DB()

def process_and_store(self, image_bytes, digit, confidence, user_id):
    """Process an image, resize it, compute its hash, and store it in the database."""
    image_id, path, hash_val = self.save_image_file(image_bytes)
    self.insert_image(image_id, digit, path, confidence, hash_val, user_id)
    self.delete_old_files()

def get_all_images_files(self):
    """Retrieve all image files from the database."""
    self.open_DB()
    self.cursor.execute('SELECT image_id, digit, path, confidence FROM Images ORDER BY rowid DESC')
    rows = self.cursor.fetchall()
    self.close_DB()
    files = Files(rows)
    return files

```

```

def get_image_by_digit_files(self, digit):
    """Retrieve image files by digit from the database."""
    self.open_DB()
    self.cursor.execute('SELECT image_id, digit, path, confidence FROM Images WHERE digit = ?', (digit,))
    rows = self.cursor.fetchall()
    self.close_DB()
    files = Files(rows)
    return files

def get_files_by_rows(rows):
    """Retrieve image files based on database rows."""
    files = []
    for row in rows:
        image_id, digit, path, confidence = row
        if os.path.exists(path):
            with open(path, 'rb') as f:
                image_bytes = f.read()
            files.append((image_id, image_bytes, digit, confidence))
    return files

class Files:
    """A class to handle a collection of image files stored in a database."""
    def __init__(self, rows):
        self.rows = rows
        self.index = 0

    def __len__(self):
        return len(self.rows)

    def __iter__(self):
        self.index = 0
        return self

    def __next__(self):
        if self.index >= len(self.rows):
            raise StopIteration
        file_path = self.rows[self.index][2]
        self.index += 1
        if os.path.exists(file_path):
            with open(file_path, 'rb') as f:
                image_bytes = f.read()
            return self.rows[self.index - 1][0], image_bytes, self.rows[self.index - 1][1], self.rows[self.index - 1][3]
        return None

```

■ server/main.py

```

import server
if __name__ == "__main__":
    # Initialize the server
    serv = server.Server()
    serv.activate_server()

```

■ server/server.py

```

import socket

import threading

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

import sys
import os

```

```

import io
import pickle
from PIL import Image
import numpy as np
import img_db_orm

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '../..')))

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), 'CNNall')))
from CNNall.CNN import models

import app.protocol as protocol

import cv2

from keyboard import on_press_key

__author__ = 'Yuval Mendel'

db_lock = threading.Lock()

class Server:
    def __init__(self):
        self.rsa_key = RSA.generate(2048)
        self.online = False
        self.sock = socket.socket()
        self.sock.bind(('0.0.0.0', protocol.PORT))
        self.sock.listen(10)
        self.clients = []
        self.sock.settimeout(0.1)
        self.server_lock = threading.Lock()
        on_press_key('q', lambda _: self.close())

    def activate_server(self):
        """
        Activate the server to listen for incoming connections.
        This method will run in a loop, accepting new clients and starting a handler for each client.
        :return: None
        """
        self.online = True
        while self.online:
            try:
                with self.server_lock:
                    s, _ = self.sock.accept()
                    handler = ClientHandler(s, self.rsa_key)
                    self.clients.append(handler)
                    handler.start()
            except socket.timeout:
                pass
            except socket.error as e:
                print(f"Socket error: {e}")

    def close(self):
        """
        Close the server and all its clients.
        :return: None
        """
        with self.server_lock:
            self.online = False

            for client in self.clients:
                client.connected = False
            for client in self.clients:
                client.join()
        print("Server closed")

class ClientHandler(threading.Thread):
    """

```

This class handles a single client connection.

```
"""
def __init__(self, soc, rsa_key):
    """
    Initialize the client handler.
    :param soc: the socket of the client
    :param rsa_key: the RSA key for encryption
    """
    super().__init__()
    self.crypto = ServerCrypto(rsa_key)
    self.soc = soc
    self.user_id = None
    self.connected = True
    self.ai = pickle.load(open('main_model.pkl', 'rb')) # Load the model
    self.db_orm = img_db_orm.ImagesORM()
    with db_lock:
        self.db_orm.create_tables()

def run(self):
    """
    Run the client handler.
    This method will perform the handshake with the client and then start the business logic.
    :return: None
    """
    try:
        self.handshake()
    except Exception as e:
        print(f"Handshake failed: {e}")
        self.connected = False
        return
    self.soc.settimeout(0.1)
    self.business_logic()

def handshake(self):
    """ Perform the handshake with the client. """
    protocol.send_by_size(self.soc, self.crypto.get_public())
    self.crypto.decrypt_aes_key(protocol.recv_by_size(self.soc), protocol.recv_by_size(self.soc))
    # Get aes key and aes iv (for cbc) and give them to crypto object
    self.send(protocol.ACK_START)

def send(self, *msg):
    """ Send a message to the client. """
    msg = protocol.format_message(msg)
    protocol.send_by_size(self.soc, self.crypto.encrypt(msg))

def recv(self):
    """
    Receive a message from the client.
    :return: decoded message
    """
    rdata = protocol.recv_by_size(self.soc)
    decrypted_data = self.crypto.decrypt(rdata)
    return protocol.unformat_message(decrypted_data)

def business_logic(self):
    """
    The main business logic of the server.
    :return:
    """
    while self.connected:
        try:
            request = self.recv()
        except ValueError:
            # Handle the case where the data is not valid
            request = b''
        except ConnectionResetError:
            request = b''
        if request == b'':
            print("Client disconnected")
            self.connected = False
```

```

        continue
opcode = request[0].decode()
to_send = (protocol.ERROR, "4")
if opcode == protocol.REQUEST_IMAGE:
    # request[1] is the image name
    # request[2] is the image content
    if len(request) != 3:
        to_send = (protocol.ERROR, "3")
    elif len(request[2]) > protocol.MAX_FILE_SIZE:
        to_send = (protocol.ERROR, "2")
    elif not is_valid_image(request[2]):
        to_send = (protocol.ERROR, "1")
    else:
        num, confidence = self.identify_num(request[2], user_id=self.user_id)
        to_send = (protocol.IMAGE_IDENTIFIED, num, str(confidence))
elif opcode == protocol.REQUEST_IMAGES:
    files = self.db_orm.get_all_images_files()
    # send the images
    self.send_files_process(files)
    to_send = (protocol.RETURN_FILES_END,)
elif opcode == protocol.REQUEST_IMAGES_BY_DIGIT:
    digit = request[1].decode()
    files = self.db_orm.get_image_by_digit_files(digit)
    self.send_files_process(files)
    to_send = (protocol.RETURN_FILES_END,)
elif opcode == protocol.SIGN_UP_REQUEST:
    username = request[1].decode()
    password = request[2].decode()
    with db_lock:
        id_of_created_user = self.db_orm.register_user(username, password)
        if id_of_created_user is None:
            to_send = (protocol.SIGN_UP_DENIED,)
        else:
            to_send = (protocol.SIGN_UP_APPROVED,)
elif opcode == protocol.LOG_IN_REQUEST:
    username = request[1].decode()
    password = request[2].decode()
    self.user_id = self.db_orm.authenticate_user(username, password)
    if self.user_id is None:
        to_send = (protocol.LOG_IN_DENIED,)
    else:
        to_send = (protocol.LOG_IN_APPROVED, username)
    self.send(*to_send)
except socket.timeout:
    pass

def send_files_process(self, files):
    """
    Send the files to the client.
    :param files: the files to send
    :return:
    """
    self.send(protocol.RETURN_IMAGES, str(len(files)))
    # send the images
    for file in files:
        if file is None:
            continue
        image_id, image_bytes, digit, confidence = file
        if image_bytes is None:
            continue
        if len(image_bytes) > protocol.MAX_FILE_SIZE:
            continue
        self.send(protocol.IMAGE_FILE_RETURN, image_id, image_bytes, digit, str(confidence))

@staticmethod
def build_return_images_msg(files):
    """ Build the return images message."""
    msg_lst = []
    for file in files:
        msg_lst.append(file[0])
        msg_lst.append(file[1])

```

```

        msg_lst.append(file[2])
        msg_lst.append(str(file[3]))
    msg_lst = [protocol.RETURN_IMAGES] + msg_lst
    return msg_lst

def identify_num(self, picture_content, user_id=None):
    """
    Identify the number in the image using the AI model.
    Save the image to the database.
    :param picture_content: the image content
    :param user_id: the user id that sent the image
    :return:
    """
    image_array = image_to_2d_array(picture_content)
    prediction = self.ai.forward(image_array)
    class_index = np.argmax(prediction[0])
    confidence = float(prediction[0][class_index])
    if user_id is not None:
        with db_lock:
            self.db_orm.process_and_store(picture_content, str(class_index), confidence, user_id)

    return str(class_index), confidence

def is_valid_image(bytes_data):
    """ Check if the given bytes data is a valid image. """
    try:
        with Image.open(io.BytesIO(bytes_data)) as img:
            img.verify() # Verifies it is an image, doesn't decode full data
        return True
    except (IOError, SyntaxError):
        return False

def thicken_digit_pil(pil_img, kernel_size=(2, 2), iterations=1):
    """ Thicken the digit in a PIL image using dilation. """
    # Convert to grayscale NumPy array
    img_np = np.array(pil_img.convert("L"))

    # Binarize if not already (thresholding)
    _, binary_img = cv2.threshold(img_np, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    # Create kernel and apply dilation
    kernel = np.ones(kernel_size, np.uint8)
    dilated = cv2.dilate(binary_img, kernel, iterations=iterations)

    # Convert back to PIL Image
    return Image.fromarray(dilated)

def image_to_2d_array(image_content):
    """
    Convert the image content to a 2D NumPy array suitable for the model.
    also does preprocessing on the image
    :param image_content:
    :return:
    """
    # Load the image from the file content
    image_file = io.BytesIO(image_content)
    image = Image.open(image_file)

    # Convert the image to grayscale
    image = image.convert('L')
    # Resize the image to 28x28 pixels
    image = image.resize((28, 28))
    # Invert the image (255 - pixel value)
    image = Image.eval(image, lambda x: 255 - x)

    image = thicken_digit_pil(image, kernel_size=(2, 2), iterations=1)

    # Save the grayscale image to the Downloads directory
    downloads_path = os.path.join(os.path.expanduser('~'), 'Downloads', 'try.png')
    image.save(downloads_path)

```

```

# Convert the image to a NumPy array
image_array = np.array(image)

# Normalize the pixel values to be between 0 and 1
image_array = image_array / 255.0
image_array = image_array.reshape(1, 1, 28, 28) # Reshape to (1, 1, 28, 28) for the model

return image_array

class ServerCrypto:
    """ This class handles the encryption and decryption of messages using RSA and AES."""
    def __init__(self, rsa_key):
        self.rsa_key = rsa_key
        self.aes_key = None
        self.aes_iv = None

    def get_public(self):
        """ Get the public key of the RSA key."""
        return self.rsa_key.publickey().export_key()

    def decrypt_aes_key(self, aes_key, aes_iv):
        """ Decrypt the AES key and IV using the RSA key."""
        decipher_rsa = PKCS1_OAEP.new(self.rsa_key)
        self.aes_key = decipher_rsa.decrypt(aes_key)
        self.aes_iv = aes_iv

    def encrypt(self, plaintext):
        """ Encrypt the plaintext using AES encryption."""
        if not self.aes_key or not self.aes_iv:
            raise ValueError("AES key and IV must be set before encryption.")

        cipher = AES.new(self.aes_key, AES.MODE_CBC, self.aes_iv)
        padded_data = pad(plaintext.encode(), AES.block_size)
        ciphertext = cipher.encrypt(padded_data)

        return ciphertext

    def decrypt(self, encrypted_text):
        """ Decrypt the encrypted text using AES decryption."""
        if not self.aes_key or not self.aes_iv:
            raise ValueError("AES key and IV must be set before decryption.")

        cipher = AES.new(self.aes_key, AES.MODE_CBC, self.aes_iv)
        decrypted_padded = cipher.decrypt(encrypted_text)
        plaintext = unpad(decrypted_padded, AES.block_size)

        return plaintext.decode()

```

■ server\CNNall\init.py

```

from . import CNN

__all__ = ['CNN']

```

■ server\CNNall\main.py

```

import pickle
from CNN.models import CNN
from CNN.layers.activations import Softmax, ReLU
from CNN.layers import MaxPool2D, Conv2D, Flatten, Dense, Input
from CNN.data import DataLoader
from CNN.data.datasets import load_mnist, load_cifar10
from CNN.utils import xavier_initializer, SGD
from CNN.losses import CategoricalCrossEntropy
from CNN.trainers import Trainer, Tester
from CNN.utils import augment_mnist
if __name__ == '__main__':

```

```

model_mnist = CNN([
    Input((1, 28, 28)),

    Conv2D(1, 32, (3, 3), xavier_initializer, stride=1, padding=1),
    ReLU(),
    MaxPool2D((2, 2), stride=2),

    Conv2D(32, 64, (3, 3), xavier_initializer, stride=1, padding=1),
    ReLU(),
    MaxPool2D((2, 2), stride=2),

    Conv2D(64, 64, (3, 3), xavier_initializer, stride=1, padding=1),
    ReLU(),

    Flatten(),
    Dense(128, 64 * 7 * 7, xavier_initializer),
    ReLU(),
    Dense(10, 128, xavier_initializer),
    Softmax()
])

optimizer = SGD(model_mnist)
loss = CategoricalCrossEntropy()
train_dataset, test_dataset = load_mnist(flatten=False)
images, labels = train_dataset
train_dataset = (images[:50000], labels[:50000])
validation_dataset = (images[50000:], labels[50000:])
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, transform=augment_mnist)
validation_loader = DataLoader(validation_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
epochs = 30
trainer = Trainer(model_mnist, optimizer, loss, learning_rate=0.005, decay_epochs=[10, 20],
                  decay_rate=0.3, validator=Tester(model_mnist, loss),
                  , early_stopping=True, patience=5)
trainer.train(train_loader, epochs, val_data_loader=validation_loader)
tester = Tester(model_mnist, loss)
loss, acc = tester.test(test_loader)
print(f"Test Loss = {loss:.4f}, Accuracy = {acc:.2%}")
with open('model.pkl', 'wb') as f:
    pickle.dump(model_mnist, f)

```

■ server\CNNall\CNN__init__.py

```
__all__ = ["layers", "models", "trainers", "data", "losses", "utils"]
```

■ server\CNNall\CNN\data\dataloader.py

```

import numpy as np
from multiprocessing.pool import ThreadPool

class DataLoader:
    def __init__(self, dataset, batch_size, shuffle=True, transform=None):
        self.images, self.labels = dataset
        self.batch_size = batch_size
        self.shuffle = shuffle
        self.indices = np.arange(len(self.images), dtype=np.int32)
        self.current_index = 0
        self.transform = transform
        self.pool = ThreadPool(processes=4) if transform else None

    def __iter__(self):
        if self.shuffle:
            np.random.shuffle(self.indices)
        self.current_index = 0
        return self

    def __next__(self):

```



```

        """Return the next batch of images and labels."""
        if self.current_index >= len(self.images):
            raise StopIteration
        start = self.current_index
        end = min(self.current_index + self.batch_size, len(self.indices))
        batch_indices = self.indices[start:end]
        batch_images_data = self.images[batch_indices]
        batch_labels_data = self.labels[batch_indices]
        if self.transform:
            batch_images_data = np.stack(self.pool.map(self.transform, batch_images_data))
        self.current_index = end
        return batch_images_data, batch_labels_data

    def __len__(self):
        return len(self.images)

    def __del__(self):
        if self.pool is not None:
            self.pool.close()
            self.pool.join()

```

■ server\CNNall\CNN\data__init__.py

```

from .data_loader import DataLoader
from . import datasets

__all__ = ["DataLoader", "datasets"]

```

■ server\CNNall\CNN\data\datasets\cifar10.py

```

import os
import urllib.request
import tarfile
import numpy as np
import pickle

def download(url, filepath):
    """Downloads a file from a URL if it doesn't exist."""
    if not os.path.exists(filepath):
        print(f"Downloading {url}...")
        urllib.request.urlretrieve(url, filepath)

def load_batch(filepath):
    """Loads a single CIFAR-10 batch."""
    with open(filepath, 'rb') as f:
        batch = pickle.load(f, encoding='bytes')
        data = batch[b'data'] # shape (10000, 3072)
        labels = batch[b'labels']
        data = data.reshape(-1, 3, 32, 32) # (batch_size, 3, 32, 32)
        return data, np.array(labels)

def one_hot_encode(labels, num_classes=10):
    """One-hot encodes the labels."""
    one_hot = np.zeros((labels.shape[0], num_classes), dtype=np.float32)
    one_hot[np.arange(labels.shape[0]), labels.astype(int)] = 1
    return one_hot

def load_cifar10(data_dir="data_files/cifar10", normalize=True, one_hot=True):
    """Downloads and loads the CIFAR-10 dataset."""
    os.makedirs(data_dir, exist_ok=True)

    url = "https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz"
    archive_path = os.path.join(data_dir, "cifar-10-python.tar.gz")
    extracted_folder = os.path.join(data_dir, "cifar-10-batches-py")

    # Download and extract if needed
    if not os.path.exists(extracted_folder):
        download(url, archive_path)

```

```

        print(f"Extracting {archive_path}...")
        with tarfile.open(archive_path, 'r:gz') as tar:
            tar.extractall(path=data_dir)

# Load training batches
train_data = []
train_labels = []
for i in range(1, 6):
    batch_path = os.path.join(extracted_folder, f"data_batch_{i}")
    data, labels = load_batch(batch_path)
    train_data.append(data)
    train_labels.append(labels)
train_images = np.concatenate(train_data, axis=0)
train_labels = np.concatenate(train_labels, axis=0)

# Load test batch
test_batch_path = os.path.join(extracted_folder, "test_batch")
test_images, test_labels = load_batch(test_batch_path)

if normalize:
    train_images = train_images.astype(np.float32) / 255.0
    test_images = test_images.astype(np.float32) / 255.0

if one_hot:
    train_labels = one_hot_encode(train_labels, num_classes=10)
    test_labels = one_hot_encode(test_labels, num_classes=10)

# Pack datasets
train_dataset = (train_images, train_labels)
test_dataset = (test_images, test_labels)

return train_dataset, test_dataset

```

■ server\CNNall\CNN\data\datasets\mnist.py

```

import os
import urllib.request
import gzip
import numpy as np

def download(url, filepath):
    """Downloads a file from a URL if it doesn't exist."""
    if not os.path.exists(filepath):
        print(f"Downloading {url}...")
        urllib.request.urlretrieve(url, filepath)

def load_images(filepath):
    """Loads MNIST image file."""
    with gzip.open(filepath, 'rb') as f:
        _ = int.from_bytes(f.read(4), 'big') # magic number
        num_images = int.from_bytes(f.read(4), 'big')
        rows = int.from_bytes(f.read(4), 'big')
        cols = int.from_bytes(f.read(4), 'big')
        images = np.frombuffer(f.read(), dtype=np.uint8)
        images = images.reshape(num_images, rows * cols)
        return images

def load_labels(filepath):
    """Loads MNIST label file."""
    with gzip.open(filepath, 'rb') as f:
        _ = int.from_bytes(f.read(4), 'big') # magic number
        num_labels = int.from_bytes(f.read(4), 'big')
        labels = np.frombuffer(f.read(), dtype=np.uint8)
        return labels

def one_hot_encode(labels, num_classes=10):
    """One-hot encodes the labels."""
    one_hot = np.zeros((labels.shape[0], num_classes), dtype=np.float32)
    one_hot[np.arange(labels.shape[0]), labels.astype(int)] = 1

```

```

return one_hot

def load_mnist(data_dir="data_files/mnist", normalize=True, flatten=True):
    """Downloads and loads the MNIST dataset."""
    os.makedirs(data_dir, exist_ok=True)

    urls = {
        "train_images": "https://storage.googleapis.com/cvdf-datasets/mnist/train-images-idx3-ubyte.gz",
        "train_labels": "https://storage.googleapis.com/cvdf-datasets/mnist/train-labels-idx1-ubyte.gz",
        "test_images": "https://storage.googleapis.com/cvdf-datasets/mnist/t10k-images-idx3-ubyte.gz",
        "test_labels": "https://storage.googleapis.com/cvdf-datasets/mnist/t10k-labels-idx1-ubyte.gz",
    }
    files = {key: os.path.join(data_dir, url.split('/')[-1]) for key, url in urls.items()}

    # Download if needed
    for key in files:
        download(urls[key], files[key])

    # Load data
    train_images = load_images(files["train_images"])
    train_labels = load_labels(files["train_labels"])
    test_images = load_images(files["test_images"])
    test_labels = load_labels(files["test_labels"])

    if normalize:
        train_images = train_images.astype(np.float32) / 255.0
        test_images = test_images.astype(np.float32) / 255.0

    if not flatten:
        train_images = train_images.reshape(train_images.shape[0], 1, 28, 28)
        test_images = test_images.reshape(test_images.shape[0], 1, 28, 28)

    # ■ NEW: One-hot encode the labels
    train_labels = one_hot_encode(train_labels, num_classes=10)
    test_labels = one_hot_encode(test_labels, num_classes=10)

    # Pack datasets
    train_dataset = (train_images, train_labels)
    test_dataset = (test_images, test_labels)

    return train_dataset, test_dataset

```

■ server\CNNall\CNN\data\datasets__init__.py

```

from .mnist import load_mnist
from .cifar10 import load_cifar10
__all__ = ["load_mnist", "load_cifar10"]

```

■ server\CNNall\CNN\layers\base.py

```

from abc import ABC, abstractmethod

class Layer(ABC):
    @abstractmethod
    def forward(self, input_data):
        pass

    @abstractmethod
    def backward(self, output_grad):
        pass

class TrainableLayer(Layer):
    def __init__(self):
        self.weights = None
        self.biases = None

```

```

        self.weights_gradient = None
        self.biases_gradient = None

    def update_parameters(self, learning_rate):
        self.weights -= learning_rate * self.weights_gradient

        self.biases -= learning_rate * self.biases_gradient

```

■ server\CNNall\CNN\layers\conv.py

```

import numpy as np
from .base import TrainableLayer
from ..utils import im2col, col2im

class Conv2D(TrainableLayer):
    def __init__(self, in_channels, out_channels, kernel_size, initialization, stride=1, padding=0):
        """Initializes a Conv2D layer with weights and biases.
        - in_channels: Number of input channels.
        - out_channels: Number of output channels (filters).
        - kernel_size: Size of the convolutional kernel (height, width).
        - initialization: Function to initialize weights.
        - stride: Stride of the convolution.
        - padding: Padding added to the input.
        """
        super().__init__()
        kh, kw = kernel_size

        # The biases have the shape (out_channels) (one bias per filter)
        self.biases = np.zeros(out_channels)

        # The Weights have the shape (out_channels, in_channels, kh, kw)
        # for each filter, we have (in_channels * kh * kw) weights (the shape of the kernel)
        self.weights = initialization((out_channels, in_channels, kh, kw))

        self.weights_gradient = np.zeros_like(self.weights)
        self.biases_gradient = np.zeros_like(self.biases)

        self.in_channels = in_channels
        self.out_channels = out_channels

        self.stride = stride
        self.kh = kh
        self.kw = kw

        self.padding = padding

        self.input_data = None
        self.output = None
        self.output_shape = None
        self.patches = None

    def forward(self, input_data):
        """Computes the convolution operation.
        - input_data: The input from the previous layer.
        - Saves calculation and returns it.
        """
        self.input_data = np.pad(input_data, ((0,0), (0,0), (self.padding, self.padding), (self.padding, self.padding)), weights_for_multiplication = self.weights.reshape(self.out_channels, -1).T

        self.patches = im2col(self.input_data, (self.kh, self.kw), self.stride)
        self.output = np.dot(self.patches, weights_for_multiplication) + self.biases
        N, _, H, W = self.input_data.shape
        out_h = (H - self.kh) // self.stride + 1
        out_w = (W - self.kw) // self.stride + 1
        self.output_shape = (N, self.out_channels, out_h, out_w)
        self.output = self.output.reshape(N, out_h, out_w, self.out_channels).transpose(0, 3, 1, 2)
        return self.output

    def backward(self, output_grad):

```

```

"""Computes the gradient of the loss with respect to the input (and with respect to the weights a
- output_grad: The gradient of the loss with respect to the output.
- Returns the gradient of the loss with respect to the input.
"""

# reshape the output gradient to match the shape of the patches
output_grad_flat = output_grad.transpose(0, 2, 3, 1).reshape(-1, self.out_channels)
# Calculate the gradient of the loss with respect to weights
self.weights_gradient = np.dot(output_grad_flat.T, self.patches)
self.weights_gradient = self.weights_gradient.reshape(self.out_channels, self.in_channels, self.kh, self.kw)

# Calculate the gradient of the loss with respect to bias
# sum of the output gradient (batch-wise)
self.biases_gradient = np.sum(output_grad_flat, axis=0)

# Calculate the gradient of the loss with respect to the input (for the next layer)
# -----
# Gradient w.r.t. the input ( $\partial L / \partial \text{input}$ ) - Intuition:
#
# For each input pixel:
# - Look at all output pixels that were computed using it.
# - Each of those output pixels has a gradient telling how much it wants to change.
# - The input pixel affected that output pixel through a specific weight in the filter.
# - So we multiply the output gradient by that weight (the strength of the connection).
# - Then we sum up all of these contributions.
#
# From the output's perspective:
# - Each output pixel "spreads" its gradient back to the input patch it came from.
# - It does so proportionally to the filter weights used during the forward pass.
#
# The final result tells each input pixel: "Here's how much you need to change to reduce the loss"
# -----
N, C, H, W = self.input_data.shape
out_h = (H - self.kh) // self.stride + 1
out_w = (W - self.kw) // self.stride + 1
input_grad_patches = np.dot(output_grad_flat, self.weights.reshape(self.out_channels, -1))
input_grad_patches = input_grad_patches.reshape(N, out_h, out_w, C, self.kh, self.kw)
input_grad_patches = input_grad_patches.transpose(0, 3, 4, 5, 1, 2)
# input_grad_patches holds the gradient of the loss with respect to each input patch.
# Shape: (N, C, kh, kw, out_h, out_w)
#
# For each image in the batch (N), each input channel (C), and each output location (out_h, out_w)
# this array tells us how much each pixel in the receptive field (defined by the kernel window kh, kw)
# should change to reduce the loss.
#
# This will be scattered back into the full input gradient image, summing overlapping contributions
# -----
input_grad = col2im(input_grad_patches, self.input_data.shape, (self.kh, self.kw), self.stride, self.stride)
return input_grad

```

■ server\CNN\CNN\layers\dense.py

```

import numpy as np
from .base import TrainableLayer

```

```

class Dense(TrainableLayer):
    def __init__(self, output_size, input_size, initialization):
        """Initializes a layer with weights and biases.
        - output_size: Number of neurons in this layer.
        - input_size: Number of neurons in the previous layer.
        - The weights matrix is initialized using Xavier initialization.
        """
        super().__init__()
        self.weights = initialization((input_size, output_size))
        self.biases = np.zeros((1, output_size))
        self.output = None
        self.input_data = None
        self.weights_gradient = None

```

```

        self.biases_gradient = None

    def forward(self, input_data):
        """Computes the affine transformation: output = input * W + b.
        - input_data: The input from the previous layer.
        - Saves calculation and returns it.
        """
        self.input_data = input_data
        z = np.dot(input_data, self.weights) + self.biases
        self.output = z
        return self.output

    def backward(self, output_grad):

        # output_grad is the gradient of the loss with respect to the output

        # Calculate the gradient of the loss with respect to weights
        # dot product of the input data transposed with the output gradient
        self.weights_gradient = np.dot(self.input_data.T, output_grad)

        # Calculate the gradient of the loss with respect to bias
        # sum of the output gradient (batch-wise)
        self.biases_gradient = np.sum(output_grad, axis=0, keepdims=True)

        # Calculate the gradient of the loss with respect to the input (for the next layer)
        # the gradient of the loss with respect to the input:
        # the gradient of the loss with respect to the output dot the gradient of the output with respect to the input
        # (chain rule)
        # we have the gradient of the loss with respect to the output (output_grad)
        # the gradient of the output with respect to the input is just the weights
        # output = input*W + B
        # so, it is just the dot product of output_grad and weights.T
        input_grad = np.dot(output_grad, self.weights.T)
        return input_grad

```

■ server\CNNall\CNN\layers\flatten.py

```

import numpy as np
from .base import Layer

class Flatten(Layer):
    def __init__(self):
        super().__init__()
        self.input_shape = None

    def forward(self, input_data):
        self.input_shape = input_data.shape
        return input_data.reshape(input_data.shape[0], -1)

    def backward(self, output_grad):
        return output_grad.reshape(self.input_shape)

```

■ server\CNNall\CNN\layers\input_layer.py

```

from .base import Layer

class Input(Layer):
    def __init__(self, input_shape):
        super().__init__()
        self.input = None
        self.input_shape = input_shape

    def forward(self, input_data):
        self.input = input_data
        return input_data

    def backward(self, output_grad):

```

```
return output_grad
```

■ server\CNN\layers\maxpool.py

```
import numpy as np
from .base import Layer
```

```
class MaxPool2D(Layer):
    def __init__(self, pool_size, stride=None, padding=0):
        """Initializes a MaxPool2D layer.
        - pool_size: Size of the pooling window (height, width).
        - stride: Stride of the pooling operation. If None, it defaults to pool_size.
        - padding: Padding added to the input.
        """
        super().__init__()
        if isinstance(pool_size, int):
            self.ph, self.pw = pool_size, pool_size
        else:
            self.ph, self.pw = pool_size

        if stride is None:
            self.stride = self.ph # default to pool size
        else:
            self.stride = stride
        self.padding = padding
        self.input_data = None
        self.output = None
        self.output_shape = None
        self.mask = None

    def forward(self, input_data):
        self.input_data = np.pad(input_data, ((0, 0), (0, 0), (self.padding, self.padding), (self.padding, self.padding)), mode='constant')
        N, C, H, W = self.input_data.shape
        out_h = (H - self.ph) // self.stride + 1
        out_w = (W - self.pw) // self.stride + 1
        self.output_shape = (N, C, out_h, out_w)
        shape = (N, C, out_h, out_w, self.ph, self.pw)
        s0, s1, s2, s3 = self.input_data.strides
        strides = (s0, s1, s2 * self.stride, s3 * self.stride, s2, s3)
        patches = np.lib.stride_tricks.as_strided(self.input_data, shape=shape, strides=strides)
        self.output = np.max(patches, axis=(4, 5))
        self.mask = (patches == self.output[..., np.newaxis, np.newaxis])
        return self.output

    def backward(self, output_grad):
        N, C, H, W = self.input_data.shape
        out_h = (H - self.ph) // self.stride + 1
        out_w = (W - self.pw) // self.stride + 1

        input_grad = np.zeros_like(self.input_data)

        # Same strides as in forward
        shape = (N, C, out_h, out_w, self.ph, self.pw)
        s0, s1, s2, s3 = input_grad.strides
        strides = (s0, s1, s2 * self.stride, s3 * self.stride, s2, s3)
        input_patches = np.lib.stride_tricks.as_strided(input_grad, shape=shape, strides=strides)

        # Direct broadcasted accumulation
        input_patches += self.mask * output_grad[..., np.newaxis, np.newaxis]

        if self.padding > 0:
            return input_grad[:, :, self.padding:-self.padding, self.padding:-self.padding]
        return input_grad
```

■ server\CNN\layers__init__.py

```
from .base import Layer, TrainableLayer
```

```
from .conv import Conv2D
from .dense import Dense
from .flatten import Flatten
from .maxpool import MaxPool2D
from .input_layer import Input
from . import activations
```

```
__all__ = ["Layer", "TrainableLayer", "Conv2D", "Dense", "Flatten", "MaxPool2D", "activations", "Input"]
```

■ server\CNNall\CNN\layers\activations\relu.py

```
from ..base import Layer
import numpy as np
```

```
class ReLU(Layer):
    def forward(self, z):
        self.input = z
        return np.maximum(0, z)

    def backward(self, output_grad):
        return output_grad * (self.input > 0).astype(float)
```

■ server\CNNall\CNN\layers\activations\sigmoid.py

```
import numpy as np
from ..base import Layer
```

```
class Sigmoid(Layer):
    def forward(self, z):
        self.output = 1 / (1 + np.exp(-z))
        return self.output

    def backward(self, output_grad):
        return output_grad * self.output * (1 - self.output)
```

■ server\CNNall\CNN\layers\activations\softmax.py

```
import numpy as np
from ..base import Layer
```

```
class Softmax(Layer):
    def forward(self, z):
        # z: shape (batch_size, num_classes)
        exps = np.exp(z - np.max(z, axis=1, keepdims=True)) # stability trick
        self.output = exps / np.sum(exps, axis=1, keepdims=True)
        return self.output

    def backward(self, output_grad):
        # Assumes softmax used with cross-entropy loss
        # and that loss.backward() returned (softmax_output - true_labels)
        return output_grad
```

■ server\CNNall\CNN\layers\activations__init__.py

```
from .relu import ReLU
from .sigmoid import Sigmoid
from .softmax import Softmax
```

```
__all__ = ["ReLU", "Sigmoid", "Softmax"]
```


■ server\CNNall\CNN\losses\base.py

```
import numpy as np
from abc import ABC, abstractmethod

# Loss functions for neural networks

class Loss(ABC):
    def __init__(self):
        self.predictions = None
        self.targets = None

    def forward(self, predictions, targets):
        """Computes the loss value."""
        self.predictions = predictions
        self.targets = targets
        return self._compute_loss()

    def backward(self):
        """Computes the gradient of the loss wrt predictions."""
        return self._compute_grad()

    @abstractmethod
    def _compute_loss(self):
        raise NotImplementedError

    @abstractmethod
    def _compute_grad(self):
        raise NotImplementedError
```

■ server\CNNall\CNN\losses\cce.py

```
import numpy as np
from .base import Loss

class CategoricalCrossEntropy(Loss):
    def __init__(self):
        super().__init__()

    def _compute_loss(self):
        # add epsilon to avoid log(0)
        return -np.sum(self.targets * np.log(self.predictions + 1e-8)) / self.targets.shape[0]

    def _compute_grad(self):
        # assumes predicted already passed through softmax
        return self.predictions - self.targets
```

■ server\CNNall\CNN\losses\mse.py

```
import numpy as np
from .base import Loss

class MeanSquaredError(Loss):
    def __init__(self):
        super().__init__()

    def _compute_loss(self):
        return np.mean((self.predictions - self.targets) ** 2)

    def _compute_grad(self):
        return 2 * (self.predictions - self.targets) / self.targets.size
```

■ server\CNNall\CNN\losses__init__.py

```
from .cce import CategoricalCrossEntropy
from .mse import MeanSquaredError
from .base import Loss

__all__ = ["MeanSquaredError", "CategoricalCrossEntropy", "Loss"]
```

■ server\CNNall\CNN\models\cnn.py

```
from ..layers import TrainableLayer, Layer

class CNN:
    def __init__(self, layers):
        """Initializes the CNN model with a list of layers.
        - layers: List of layers to be added to the model.
        """
        self.layers = layers

    def forward(self, input_data):
        """Performs a forward pass through the model.
        - input_data: Input data to the model.
        Returns the output of the last layer.
        """
        for layer in self.layers:
            input_data = layer.forward(input_data)
        return input_data

    def backward(self, output_grad):
        """Performs a backward pass through the model.
        - output_grad: Gradient of the loss with respect to the output.
        Returns the gradient of the loss with respect to the input.
        """
        for layer in reversed(self.layers):
            output_grad = layer.backward(output_grad)
        return output_grad

    def update_parameters(self, learning_rate):
        """Updates the parameters of the model using the gradients.
        - learning_rate: Learning rate for the update.
        """
        for layer in self.layers:
            if isinstance(layer, TrainableLayer):
                layer.update_parameters(learning_rate)
```

■ server\CNNall\CNN\models__init__.py

```
from .cnn import CNN

__all__ = ["CNN"]
```

■ server\CNNall\CNN\trainers\tester.py

```
import numpy as np

class Tester:
    def __init__(self, model, loss_function=None):
        self.model = model
        self.loss_function = loss_function

    def test(self, dataloader):
```

```

loss = 0
correct = 0
for inputs, targets in dataloader:
    # Forward pass
    output = self.model.forward(inputs)
    if self.loss_function is not None:
        loss += self.loss_function.forward(output, targets)
    predictions = np.argmax(output, axis=1)
    true_labels = np.argmax(targets, axis=1)
    correct += np.sum(predictions == true_labels)
if self.loss_function is not None:
    loss /= len(dataloader)
accuracy = correct / len(dataloader)
return loss, accuracy

```

■ server\CNNall\CNN\trainers\trainer.py

```

from time import time
import copy
import numpy as np

class Trainer:
    def __init__(self, model, optimizer, loss_function,
                 learning_rate=0.01, decay_epochs=None,
                 decay_rate=None, validator=None,
                 early_stopping=False, patience=5):
        """Initializes the Trainer with a model, optimizer, and loss function.
        - model: The neural network model to be trained.
        - optimizer: The optimizer to be used for training.
        - loss_function: The loss function to be used for training.
        """
        self.model = model
        self.best_model_state = None
        self.optimizer = optimizer
        self.loss_function = loss_function
        self.learning_rate = learning_rate
        self.decay_epochs = decay_epochs or []
        self.decay_rate = decay_rate or 0.1
        self.validator = validator
        self.early_stopping = early_stopping
        self.patience = patience

    def train(self, dataloader, num_epochs, val_dataloader=None):
        best_val_acc = -np.Inf
        epochs_no_improve = 0
        for epoch in range(num_epochs):
            total_loss = 0
            t1 = time()
            if epoch in self.decay_epochs:
                self.learning_rate *= self.decay_rate
                print(f"Learning Rate Decayed to {self.learning_rate}")
                print(f"Epoch {epoch + 1}/{num_epochs}", end=", ")

            for inputs, targets in dataloader:
                # Forward pass
                outputs = self.model.forward(inputs)
                loss = self.loss_function.forward(outputs, targets)
                # Backward pass
                output_grad = self.loss_function.backward()
                self.model.backward(output_grad)
                self.optimizer.step(self.learning_rate)
                total_loss += loss
            if len(dataloader) > 0:
                print(f"Loss: {(total_loss/len(dataloader)):.4f}")
            if val_dataloader is not None and self.validator is not None:
                val_loss, val_acc = self.validator.test(val_dataloader)
                print(f"Validation Loss = {val_loss:.4f}, Accuracy = {val_acc:.2%}")
                if self.early_stopping:

```

```

        if val_acc > best_val_acc:
            best_val_acc = val_acc
            self.best_model_state = copy.deepcopy(self.model)
            epochs_no_improve = 0
        else:
            epochs_no_improve += 1
            print(f" (no improvement for {epochs_no_improve} epoch(s))")

        if epochs_no_improve >= self.patience:
            print("Early stopping triggered!")
            self.model = self.best_model_state
            break

    t2 = time()
    print(f"Time taken: {t2 - t1:.2f} seconds")

```

■ server\CNNall\CNN\trainers__init__.py

```

from .trainer import Trainer
from .tester import Tester
__all__ = ["Trainer", "Tester"]

```

■ server\CNNall\CNN\utils\augment.py

```

import numpy as np
import cv2

def augment_mnist(img):
    """
    Apply random transformations to MNIST-like image.
    Expects img of shape (1, 28, 28) or (28, 28) with float32 in [0,1].
    Returns transformed image with shape (1, 28, 28)
    """
    if img.ndim == 3:
        img = img[0] # squeeze channel dimension

    # Ensure float32 and range [0, 1]
    img = img.astype(np.float32)
    img = np.clip(img, 0, 1)

    # Convert to 8-bit for OpenCV ops
    img_cv = (img * 255).astype(np.uint8)

    # Random rotation between -15 and +15 degrees
    angle = np.random.uniform(-15, 15)
    M = cv2.getRotationMatrix2D((14, 14), angle, 1.0)
    img_cv = cv2.warpAffine(img_cv, M, (28, 28), borderMode=cv2.BORDER_CONSTANT, borderValue=0)

    # Random erosion or dilation
    if np.random.rand() < 0.5:
        kernel = np.ones((2, 2), np.uint8)
        if np.random.rand() < 0.6: # make it slightly more likely to dilate
            img_cv = cv2.dilate(img_cv, kernel, iterations=1)
        else:
            img_cv = cv2.erode(img_cv, kernel, iterations=1)

    # Small random translation (shift x/y by -2 to 2 pixels)
    tx = np.random.randint(-2, 3)
    ty = np.random.randint(-2, 3)
    M = np.float32([[1, 0, tx], [0, 1, ty]])
    img_cv = cv2.warpAffine(img_cv, M, (28, 28), borderMode=cv2.BORDER_CONSTANT, borderValue=0)

    # Add small Gaussian noise
    img = img_cv.astype(np.float32) / 255.0
    noise = np.random.normal(0, 0.05, img.shape)
    img = np.clip(img + noise, 0, 1)

    return img[np.newaxis, :, :] # shape (1, 28, 28)

```

■ server\CNNall\CNN\utils\initializers.py

```
import numpy as np

def xavier_initializer(shape):
    """
    Xavier initializer for any shape of weights.
    - shape: tuple, the shape of the weight tensor
    """
    if len(shape) == 2:
        # Dense layer: (out_features, in_features)
        fan_in, fan_out = shape[1], shape[0]
    elif len(shape) == 4:
        # Conv2D: (out_channels, in_channels, kernel_height, kernel_width)
        fan_in = shape[1] * shape[2] * shape[3]
        fan_out = shape[0] * shape[2] * shape[3]
    else:
        raise ValueError(f"Unsupported shape for Xavier initialization: {shape}")

    limit = np.sqrt(6 / (fan_in + fan_out))
    return np.random.uniform(-limit, limit, size=shape)
```

■ server\CNNall\CNN\utils\optimizer.py

```
from ..layers import Layer, TrainableLayer

class SGD:
    def __init__(self, model):
        self.model = model

    def step(self, learning_rate=0.01):
        for layer in self.model.layers:
            if isinstance(layer, TrainableLayer):
                layer.update_parameters(learning_rate)
```

■ server\CNNall\CNN\utils\tensor_patches.py

```
import numpy as np

def im2col(input_data, kernel_size, stride): # I am too cool
    """
    Extracts sliding local blocks (patches) from a padded input tensor and flattens them into rows.

    This operation is used to convert a 4D image tensor into a 2D matrix suitable for efficient
    convolution via matrix multiplication.

    Args:
        input_data (ndarray): Input tensor of shape (N, C, H, W), assumed to be padded already.
        kernel_size (tuple): Tuple of (kh, kw), the kernel height and width.
        stride (int): Stride of the convolution.

    Returns:
        patches (ndarray): A 2D array of shape (N * out_h * out_w, C * kh * kw),
                           where each row is a flattened receptive field from the input.
    """
    N, C, H, W = input_data.shape
    kh, kw = kernel_size

    # Calculate output spatial dimensions
    out_h = (H - kh) // stride + 1
    out_w = (W - kw) // stride + 1

    # Define output shape and strides for as_strided
```

```

shape = (N, C, out_h, out_w, kh, kw)
s0, s1, s2, s3 = input_data.strides
strides = (s0, s1, s2 * stride, s3 * stride, s2, s3)

# Extract and reshape patches
patches = np.lib.stride_tricks.as_strided(input_data, shape=shape, strides=strides)
patches = patches.transpose(0, 2, 3, 1, 4, 5) # → (N, out_h, out_w, C, kh, kw)
patches = patches.reshape(N * out_h * out_w, C * kh * kw)

return patches

def col2im(patch_grads, input_shape, kernel_size, stride, padding):
    """
    Reconstructs the full input gradient tensor from local patch gradients (reverse of im2col).

    This function takes the per-output-location patch gradients (e.g., from backpropagation)
    and scatters them back into the full input tensor, summing overlapping contributions.

    Args:
        patch_grads (ndarray): Gradient patches of shape (N, C, kh, kw, out_h, out_w),
                                where each patch is a (C × kh × kw) gradient for an output pixel.
        input_shape (tuple): Shape of the padded input tensor, e.g. (N, C, H, W).
        kernel_size (tuple): Tuple (kh, kw), height and width of the kernel.
        stride (int): Stride used during the forward convolution.
        padding (int): Padding used during the forward convolution.

    Returns:
        input_grad (ndarray): Reconstructed input gradient tensor of shape:
                               (N, C, H - 2 * padding, W - 2 * padding) if padding > 0,
                               or (N, C, H, W) if no padding.
    """
    N, C, H, W = input_shape
    kh, kw = kernel_size
    out_h = patch_grads.shape[4]
    out_w = patch_grads.shape[5]
    input_grad_padded = np.zeros(input_shape, dtype=patch_grads.dtype)
    # Calculate the strided view shape and strides
    shape = (N, C, out_h, out_w, kh, kw)
    s0, s1, s2, s3 = input_grad_padded.strides
    strides = (s0, s1, s2 * stride, s3 * stride, s2, s3)

    # Create a writeable strided view into input_grad_padded
    patch_view = np.lib.stride_tricks.as_strided(
        input_grad_padded,
        shape=shape,
        strides=strides,
        writeable=True
    )

    # Transpose patches to match patch_view shape
    patch_reshaped = patch_grads.transpose(0, 1, 4, 5, 2, 3)
    np.add.at(patch_view, (...), patch_reshaped)
    if padding > 0:
        return input_grad_padded[:, :, padding:-padding, padding:-padding]
    return input_grad_padded

```

■ server\CNNall\CNN\utils__init__.py

```

from .initializers import xavier_initializer
from .optimizer import SGD
from .tensor_patches import im2col, col2im
from .augment import augment_mnist
__all__ = ["SGD", "xavier_initializer", "im2col", "col2im", "augment_mnist"]

```