signature verification with triplet loss

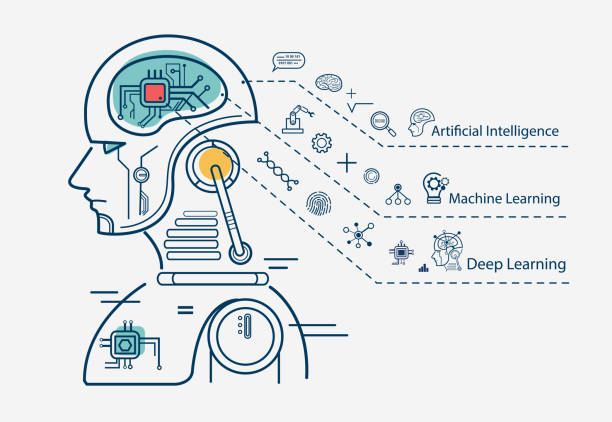
מגיש: לירן ייני

ת"ז: 325707156

שם מנחה: שי פרח

חלופה: למידת מכונה

תאריך : 19.06.2022

****

[**מבוא:**](#_weh4t9x6n9pl) **3**

[רקע](#_rwtufbivjyee) 3

[תהליך המחקר](#_dkh1r2x39myp) 4

[אתגרים מרכזיים](#_f9h265ebn65s) 5

[**מבנה / ארכיטקטורה של הפרויקט**](#_etvensrux36w) **5**

[איסוף הכנה וניתוח הנתונים](#_r9o43t4i4t6j) 5

[שלב בנייה ואימון המודל](#_k7jor912xgcr) 6

[תיאור גרפי של המוגל](#_3yonnow7yfl0) 6

[שכבות שונות ברשת](#_ps2zbok8xjp2) 9

[דוחות וגרפים המתארים את תוצאות שלב האימון](#_cp8ni39nbf4a) 11

[דוח ריכוז ה - Hyperparameters](#_d5gp3b6n2i8u) 12

[הסברHyper parameters:](#_ooireva14d6x) 13

[תיעוד והסבר של פונקציית השגיאה](#_qkx8t6zg7s6j) 13

[תיעוד והסבר של ייעול ההתכנסות](#_a6kqvigvw9z7) 14

[תיעוד והתמודדות עם הטיה ושונות](#_gctuoop162vx) 14

שלב הישום 14

[תיאור והסבר כיצד היישום משתמש במודל](#_t9zoytogpvfk) 15

[תיאור קוד הקולט את ה DATA שעליו יבוצע החיזוי והתאמתו למבנה נתונים המתאים לחיזוי](#_lcxte1mrcq18) 15

[**מדריך למפתח**](#_jzuybld79gsl) **16**

[אימון המודלים](#_8lum41os7fni) 16

[בדיקת ביצועים של המודלים](#_3aped82k865t) 25

[**מדריך למשתמש**](#_i0icm07m9fs5) **35**

[תרשים מסוג Screen flow diagram](#_oj0ftgp5csuw) 35

[תפקיד כל חלון באפליקציה](#_7oo80aus4zwi) 35

[תפקיד אלמנטים](#_5nxc7lfck0fb) 36

[**סיכום אישי / רפלקציה**](#_yzyfeu59mjaz) **37**

[**ביבליוגרפיה**](#_v94ggs3n4kz) **37**

[**נספחים**](#_gzkephg8qxdu) **38**

[קוד האפליקציה](#_j1lfrophznjd) 38

[מודלים קודמים](#_ls4uo8ifb3e0) 42

[המודל של FRmodel](#_uyieee68ylks) 43

[המודל של ResNet50](#_e1nsjao6o6f0) 54

# מבוא:

## רקע-

כשנודע לי שישנו פרויקט שאני יכול לבחור את הנושא שלו התלבטתי מה יהיה נושא הפרויקט. זהו היה אחד השלבים הקשים. עלו לי לא מעט אופציות לנושאי העבודה, חלקם היו מאוד מענינים אך מסובכים,

לבסוף חשבתי לעשות פרויקט המנסה לזהות זיופי חתימות. סיווג חתימות היה נשמע לא מאתגר. לאחר מחשבה ארוכה חשבתי להשתמש ב- triplet loss בשביל "להוסיף קצת פלפל" לפרויקט.

הנושא הסופי של הפרויקט שלי הוא זיהוי אמינות של חתימות כלליות - המודל מקבל תמונה של חתימה כלשהי ושם של האדם לו שייכת החתימה (החתימה לא חייבת להתבצע על ידי האדם שאליו משוכיית החתימה- זיוף), המחשב שולף את התמונה של החתימה המקורית של אותו אדם בעזרת מילון המסייג חתימה לאדם. לאחר מכן הוא מריץ את המודל על התמונה שהוכנסה ומחזיר את המרחק בין התמונה למאגר התמונות הנתונות של אותו האדם. ככל שהמרחק קצר יותר כך הסיכוי שהחתימה מזויפת קטן יותר.

מלבד הכנת מודל מקורי, איסוף תמונות מקוריות של חתימות וכתיבת קוד מקורי מאפס, אני בודק האם ניתן להמיר מודל מוכן של FRmodel המשמש לזיהוי פנים עם triplet loss, למודל המשמש לזיהוי חתימות. יתר על כן, אני בודק איך מודל עם ה weights של imagenet יזהה את אמינות החתימות לאחר אימון עם -triplet loss. הרעיון מאחורי פעולה עם triplet loss הוא שלא אצטרך לאמן מחדש את המודל כל פעם שאדם יצטרף למאגר הנתונים הקיים. לכן, כאשר נוסיף אדם חדש למאגר הנתונים, המודל ידע לזהות זיופים של החתימה של האדם החדש.

קהל היעד של המוצר הוא בנקים. הבנקים דורשים חתימה מכל אדם כדי לזהות שאכן עסקאות בחשבון מסוים נעשו על ידי בעל החשבון. כיום, במידה וחתימה מספיק דומה לחתימה של בעל החשבון, הבנק לא מטיל ספק בחותם. הדבר יכול לגרום שבעל החשבון ירומה ופעילות חשבונו לא תבוצע על ידו. אני מציע מודל אשר יזהה זיוף של חתימות אפילו אם הן דומות. כלומר, כל עוד אדם שונה חתם המודל יזהה זאת. עם זאת, הפרויקט שלי הוא רק הוכחה ליעילות של triplet loss מכיוון שאין בידי את כוח החישוב או את כמות הנתונים הנדרשת בכדי לאמן מודל מוכן לשיווק.

לאחר בניית מודל מאומן יהיה ניתן לבנות מודל שידע לזייף חתימות רק על ידי תמונה שלהן. המחשב ישרבט, המודל יבדוק כמה השרבוט קרוב לחתימה המקורית ולאר מכן, המחשב ינסה להתקרב לחתימה המקורית (לעשות זיוף טוב יותר) , כך המחשב ילמד לזייף חתימות. כמובן שהכנת דבר כזה היא אינה מוסרית, אך בעזרת בניית מודל שכזה יהיה ניתן לבסוף לתת למודלים לאמן אחד את השני. המחשב יצייר זיוף טוב והמודל ילמד לזהות זיופים קשים, ובכך הם ישפרו אחד את השני במחזוריות.

## תהליך המחקר-

כאשר התחלתי לעבוד על הפרויקט שלי ידעתי מראש כי בנקים כבר כיום משתמשים בתוכנת זיהוי חתימות. לכן, לפני שהתחלתי לחפש עבודות עם triplet loss, עשיתי חיפוש כללי על signature verification ולא מצאתי אף פרויקט המשלב בין השניים. בהמשך המחקר, מצאתי עבודות רבות שעובדות עם סיווג חתימות, אך לא מצאתי שילוב של triplet loss. הבנתי כי הפרויקט שלי מקורי לגמרי, דבר הגרם לי לחשוש כי אולי הניסוי מיועד לכישלון. למרות זאת, החלטתי להמשיך לנסות. בשביל לעבוד על הפרויקט המקורי שלי הייתי צריך לנסות למצוא מידע באינטרנט על זיהוי חתימות וגם על triplet loss, הקושי בדבר היה כי הפרויקטים שמצאתי בנושא היו מיועדים לזיהוי פנים, או לסיווג. מכיוון שאף אחד מן הפרויקטים לא מתאים לי ב100% הייתי צריך לאסוף מידע מכמה מקורות ולעשות את השילוב בעצמי. מטרת הפרויקט היא בדיקה ישומית של השילוב בין השניים, כמובן כי אין לי את כוח החישוב בשביל להגיע לתשובה חד משמעית אך הרעיון הוא לתת דוגמה והוכחה ראשונית לחידוש שלי.

בשביל ליישם את הרעיון הייתי צריך לעבוד עצמאית, למדתי איך ליצור generator המספק למודל קבוצות בכל שלב באימון, איך להכניס למודל פונקציית loss מקורית, איך לעבד תמונות וכיצד לאמן מודל על ידי בניית מודל אחר.

## אתגרים מרכזיים-

במהלך פיתוח הפרויקט תמיד צפוי הלומד להסתבך ולהיתקל בבעיות שונות. הבעיות הנפוצות ביותר הם ארורים (Error) שונים אשר יכולים להופיע בכל מיני מקומות. יכול להיות ארור בארגון הנתונים, הכנסתם למודל, אימון המודל וכמעט בכל מקום בפרויקט. הקושי הוא להתמודד ולפתור אותם. בנוסף על כך ישנם בעיות שלא נראות לעין. לדוגמה, כשמודל מחזיר תשובה לא הגיונית, אך על הדף נראה כי הכל בסדר (ללא error).

# מבנה / ארכיטקטורה של הפרויקט

## איסוף הכנה וניתוח הנתונים -

את מבנה הנתונים שלי בניתי באופן עצמאי. בשביל לבנות את מסד הנתונים, הייתי צריך מספר חתימות אמיתיות ומספר חתימות שמנסות לזייף את החתימה. לשם כך, גייסתי 8 אנשים (כולל אותי) תחילה כל אחד מן המשתתפים חתם 6 חתימות מקוריות בשלושה צבעים שונים (שחור, כחול ואדום).

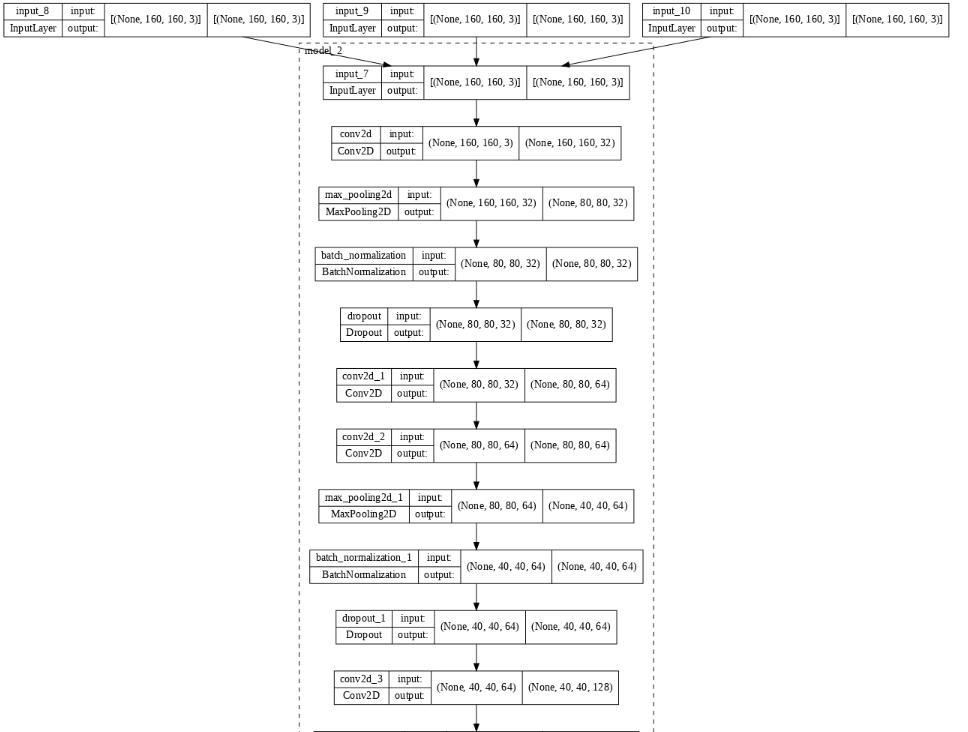
לאחר מכן, כל משתתף ניסה לזייף את החתימות של שאר המשתתפים, אחת מכל צבע. כך לאחר שכל המשתתפים חתמו היו לי 216 תמונות שונות של חתימות כשאר חלקן אמיתיות וחלקן מזויפות. למרות שמספר זה נראה קטן ביחס למה שנדרש בכדי לאמן מודל, חשוב לזכור כי כאשר עובדים עם triplet loss חלוקת הנתונים היא בשלשות (anchor, positive, negative). לכן, כאשר נחלק את הנתונים, על כל שתי תמונות זהות ישנן מספר רב של אפשרויות אימון שונות - מספר הנתונים/אפשרויות הוא 52896 . הנתונים אינם מאורגנים כdataset מפני שאני רוצה לעבוד ללא קטגוריות וסיווג. את הנתונים חילקתי על פי אדם, ועל פי זיופים, לאחר מכן הכנסתי את הכל לרשימות של anchor, negative ו positive ומהרשימות בניתי generator המכניס לmodel קבוצת שלשות על פי הbatch\_size.

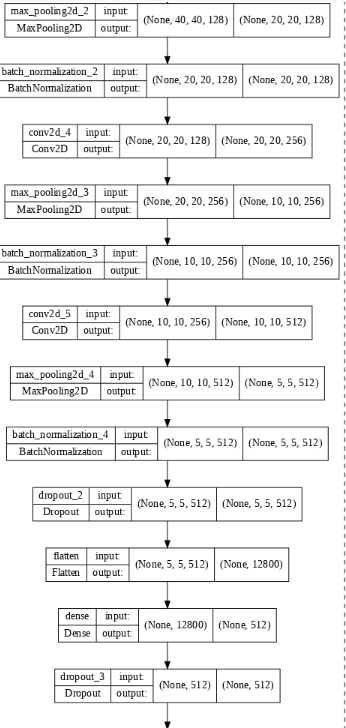
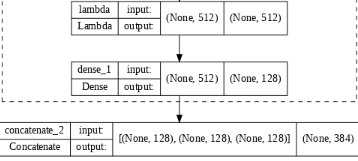
## שלב בנייה ואימון המודל -

הרעיון של הפרויקט שלי היה בניית מודל מקורי משלי, עבודה עם המשקלים של imageNet ובחינה של FRmodel על חתימות. FRmodel לא עבד כלל. הרשת של FRmodel גדולה מאוד ומאומנת שעות רבות על זיהוי פנים, ולכן האימון הרב עליה לא השפיע כלל. התוצאות לא השתנו הרבה כלומר, המודל נשאר די זהה והתצאות היו מספרים גדולים בהרבה מ 0.4 (ההפרש המבוקש בין זיוף לאמת).

כאשר המודל אומן על imageNet התוצאות היו בטווח הרצוי אך היו מאוד קרובות אחת לשנייה. הזיופים הניבו מספר בין 0.2 ל0.6 והמקוריים הניבו מספר בין 0 ל0.4. ניתן לראות שיש חפיפה די ברורה ומשמעותית, כלומר המודל לא הצליח לסווג ולהבחין בצורה מספיק טובה בין החתימות. המודל המקורי שלי היה החלק הכי קשה מכיוון שהכל נכתב מאפס. המודל שונה כמה פעמים בין אם בשורות בודדות או לגמרי. המודל שונה משתי סיבות, תיקון בעיות ושיפור ביצועים. לדוגמה מודל אחר החזיר מרחקים בין תמונות בגודל 100 (כאשר נדרש להיות קטן מ -1), ומודל אחר החזיר מרחקים בטווח הרצוי אך רנדומליים. 

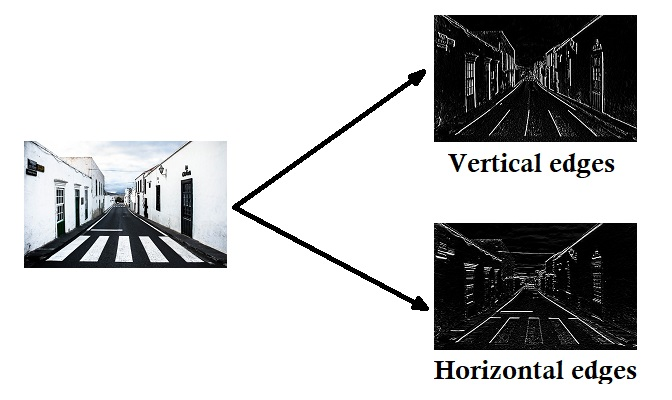
### תיאור גרפי של המוגל



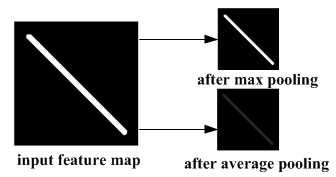


### **שכבות שונות ברשת**:

**conv2D** - שכבת הקונבולוציה פועלת בשביל לקבל ייצוג אחר ופשוט יותר של הדאטה. השכבה מנסה למצוא דפוסים ומאפיינים החוזרים על עצמם בתמונות והיא עושה זאת על ידי כפל מטריצות. כפל המטריצות נעשה על ידי מסננים. השכבה מקבלת input ומפעילה עליו מסנן (filter), המסנן מוכפל בדאטה ומתקבלת מטריצה חדשה עם פיצ'ר עיקרי מסוים. לרוב בכל רשת קונבולוציה יהיו כמה מסננים, אשר כל אחד מהם אמור ללמוד מאפיין אחר בתמונה. ככל שהרשת הולכת ומעמיקה, כך המאפיינים בתמונה אמורים להיות מובחנים באופן חד יותר אחד מהשני, ולכן המסננים בשכבות העמוקות אמורים להבדיל בין דברים מורכבים יותר. לדוגמה בתחילת הרשת המסנן עשוי לזהות קווים אנכיים, ובסוף הרשת המסננים יזהו דברים מסובכים יותר כמו עיניים. בנוסף על כך, לאחר כפל המטריצות התמונה מקבלת ייצוג פשוט יותר ולכן כמות המשקלים והמשתנים קטנה.



**MaxPooling2D** - מטרת הפונקציה היא להקטין את כמות המשתנים ולהקל על רשת הקונבולוציה. פעמים רבות דאטה מאופיין בכך שאיברים קרובים דומים אחד לשני, למשל פיקסלים סמוכים לרוב יהיו בעלי אותו ערך. ניתן לנצל עובדה זו בכדי להוריד את מספר החישובים הדרוש. לאחר כל קונבולוציה הפונקציה תעבור על התמונות ותיקח את המספר המקסימלי מבין המיימדים שהוכנסו לה. לדוגמה אם הוכנס 2\*2, הפונקציה תרוץ על התמונה ותחזיר את הערך המקסימלי מבין כל ריבוע בעל המימדים האלו.



**BatchNormalization** - טכניקה לאימון רשתות עמוקות, השכבה שומרת על ממוצע התוצאות קרוב ל - 0. יש לכך השפעה על תהליך הלמידה והפחתה דרמטית של מספר האיפוקים הנדרשות לאימון רשתות עמוקות.

**Dense** - שכבה אשר דוחסת את התוצאה משכבה קודמת למספר מסוים של נוירונים, ניתן להפעיל על השכבה פונקציית אקטיבציה.

**Dropout** - השכבה מורידה אחוז מסוים מהנתונים (במקרה שלנו תמונות רנדומליות) בשביל למנוע overfitting.

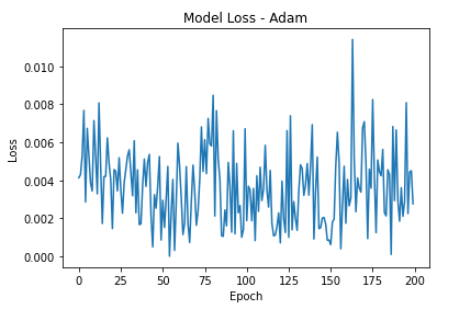
**Flatten** - השכבה משטחת את התוצאה משכבה קודמת למימד אחד.

**Concatenate** - מחבר בין רשימה של תוצאות בעלות אותו גודל ומחזיר תוצאה אחת משולבת שלהם.

### 

### דוחות וגרפים המתארים את תוצאות שלב האימון -

לאחר בניית מודלים שונים, המודל הסופי שעבד בצורה הטובה ביותר הוא זה שבחרתי להשקיע בו את רוב זמן העבודה של המחשב. המודל אומן כ - 600 epoch עם 500 צעדים (steps) עבור כל אחד. הגרף מייצג את הלמידה ב200 ה - epoch האחרונים.



הגרף מציג את השינוי בloss על גבי 200 epoch. ככל שהloss מתקרב ל - 0 המרחק בין התמונות מתקרב למה שדרשנו מהמחשב. בגרף הנוכחי מוצגים 200 ה- epoch האחרונים ולכן הloss כבר נמוך מתחילתו של הגרף.

מכיוון ש -triplet loss מקבלת 3 תמונות ומנסה להחזיר הפרש גדלים מסוים אין תשובה מדויקת שהמודל חייב להחזיר. למודל אין מטריצה של דיוק (accuracy) ולכן אנו מחויבים לבדוק את דיוק המודל בעצמנו. לשם כך בניתי קוד אשר בוחן את היכולות של המודל. תחילה, בחנתי את התוצאות שהמודל מניב כאשר מקיים אינטראקציה עם תמונות של חתימות זהות ושונות. לאחר מכן, בדקתי האם המודל מצליח לזהות חתימות ללא קשר לצבע שלהם. לדוגמה, בדקתי האם הוא מזהה כי חתימה שחורה וכחולה מסוגלות להיות אותה חתימה. בהמשך, בדקתי איך המודל מסתדר בזיהוי של זיוף חתימות כאשר הזיוף קל או קשה. לבסוף בדקתי את דיוק המודל בעצמי.

בהרצת הקוד לבדיקת הדיוק בדקתי איך המודל מצליח להסתדר עם הדאטה עליה הוא התאמן (train). לאחר כל הבדיקות הבנתי כי המודל מניב תוצאות בטווח המספרים הרצוי וכי ניתן לראות הפרש ברור בין חתימות זהות לשונות. בנוסף המודל אכן מתקשה יותר לזהות שוויון בין חתימות כאשר הצבע שונה, אך לאחר אימון מספק הוא מצליח לזהות אותם. בנוגע לזיוף חתימות המודל הסתדר והצליח לזהות כי מדובר בזיוף גם כאשר הזיוף היה קל לזיהוי וגם כאשר היה קשה. לבסוף המודל הצליח לזהות 81% מתוך החתימות האותנטיות ו 83% מתוך החתימות המזויפות. בסך הכל המודל סיים עם 82% דיוק על הדאטה עליו התאמן.

השלב הבא היה לבדוק איך המודל מצליח להסתדר עם דאטה שאותו הוא לא ראה מעולם (test). לאחר כל הבדיקות הבנתי כי המודל מניב תוצאות בטווח המספרים הרצוי, אך הן אינן מדויקות באותה המידה. המודל מציג לפעמים זיופים כאותנטים ולפעמים אותנטיים כזיוף. המודל הצליח לזהות 55% מתוך החתימות האותנטיות ו- 80% מתוך החתימות המזויפות.לבסוף המודל סיים עם 78% דיוק על הדאטה עליו התאמן.

## דוח ריכוז ה - Hyperparameters

| **(160,160)** | **Input shape** |
| --- | --- |
| **Adam** | **optimizer** |
| **Triplet loss** | **Loss function** |
| **64** | **Batch size** |
| **Sigmoid** | **Activation function** |
| **600** | **Epochs** |
| **0.3** | **Dropout rate** |
| **L2** | **regularization** |
| **מאותחל אקראית על ידי keras** | **Weight** |

## הסבר Hyper parameters:

**input shape** - הגודל של התמונה נקבע לפי הגודל ששומש ב- FRmodel.

**Loss function** - פונקציית השגיאה היא הנושא המרכזי ומה שמייחד את הפרויקט שלי, ולכן היא חובה.

**Batch size -** השתמשתי בגודל של 64 כיוון שהגודל מאפשר למידה טובה ואפקטיבית על הנתונים מבלי להקשות על כוח החישוב.

**Activation function -** הפונקציה של sigmoid עובדת טוב כאשר מנסים להגיע למספר בין אפס לאחד מספר מצוין למרחק אני מנסה להציג.

**Epoches -** מספר הepoches שונה בין 0 ל 1000, לבסוף התוצאות קיבלו צורה טובה לאחר 600 ומאז לא השתנו יותר מידי. בנוסך פחדתי לקבל overfit ולכן נשארתי עם 600.

**Dropout rate -**  מכיוון שאין לי מספר גדול מאוד של נתונים, ואין לי כוח חישוב רב או זמן לאמן רשת גדולה כמו שצריך הגדרתי dropout של 30 אחוז בכדי למנוע overfit.

### תיעוד והסבר של פונקציית השגיאה-

**triplet loss -** פונקציה המקבלת שלושה פרמטרים**:**

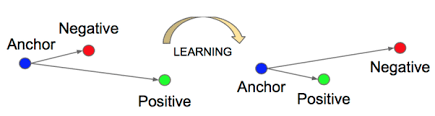
anchor - תמונה של חתימה מסוימת. (a)

positive - תמונה אחרת של אותה חתימה מסוימת.(p)

negative - תמונה של חתימה שונה השתיים הקודמות.(n)

הפונקציה מודדת את המרחק בין (a) ל (p) - המרחק החיובי, ובין (a) ל (n) - המרחק השלילי. מבצעת חיסור בין המרחקים ומוסיפה אלפא - מספר קבוע שנועד למנוע את איפוס כל המשתנים. לאחר מכן היא עושה לתוצאה מקסימום עם 0 ומחזירה את loss.

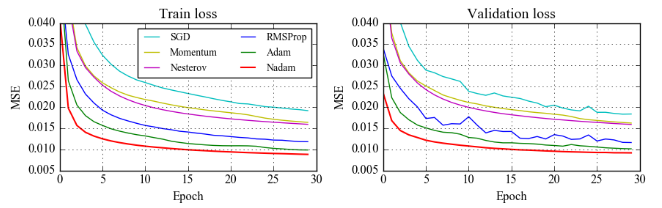
המחשב ישאף להקטין את הloss ובשביל לעשות זאת הוא יצטרך שהמרחק בין התמונות יהיה לפחות אלפא.



### 

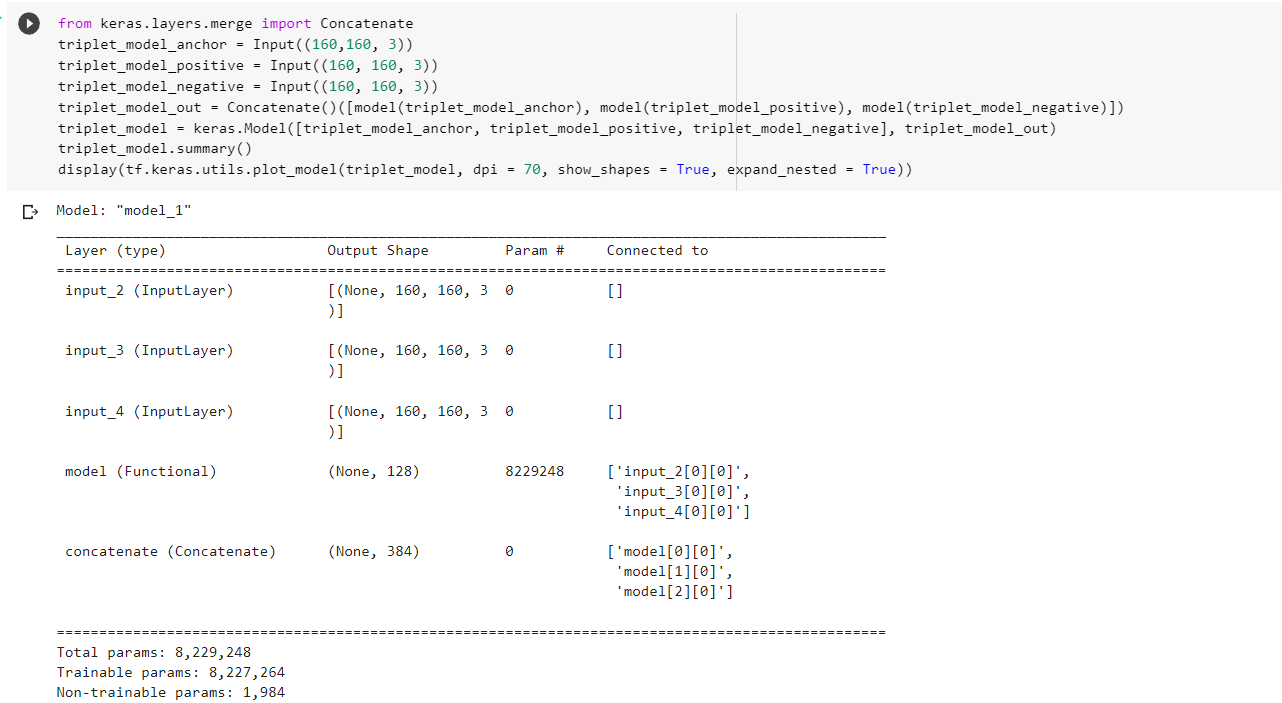
### תיעוד והסבר של ייעול ההתכנסות-

בפרויקט השתמשתי באחד מן האלגוריתם ייעול (optimizer) הידועים והטובים ביותר, Adam optimizer. האופטימייזר יעיל מאוד ובעל יתרונות משמעותיים על אחרים ביניהם, עבודה יעילה וטובה עם מסד נתונים גדול ומספר פרמטרים גדול, דורש מעט זיכרון, יעיל חישובית, מהיר, אינטואיטיבי ועוד. בנוסף האלגוריתם כולל שילוב של שתי שיטות של gradient descent, מומנטום - משמש להאצת האלגוריתם gradient descent על ידי שינוי קצב הלמידה בהתאם לממוצע המשקלים. ו RMSP.

.

### תיעוד והתמודדות עם הטיה ושונות-

הפרויקט שלי בא לתת אפשרות לזיהוי של זיוף חתימות בצורה נוחה, יעילה, ומסודרת וללא תלות בסיווג. בזמן העבודה על הפרויקט נתקלתי בארורים רבים חלקם פשוטים וחלקם מסובכים. אחת הבעיות שהיו לי כאשר התחלתי לכתוב את הקוד של המודל היא למצוא דרך לאמן מודל שמקבל 3 תמונות שונות, מריץ על התמונות פונקציית loss מקורית ולבסוף מחזיר מודל מאומן שיכול לפעול על תמונה יחידה. בשביל לעשות זאת נדרשתי ללמוד כי כאשר מאמנים מודל המשלב מודל אחר, המודל האחר מסוגל ללמוד. לאחר בניית המודל בהרצת הקוד קיבלתי ארור ולא ידעתי מה לעשות - לא הבנתי מה המודל עושה לא נכון. רק לאחר זמן רב, הבנתי כי הloss לא מצפה לאותו input כמו שהמודל מוציא. לבסוף תיקנתי והגעתי למודל המוכן:



בעיה נוספת הייתה עם batch generator כאשר במשך זמן רב לא הבנתי מדוע המודל לא לומד. רק בשלב מאוחר יותר גיליתי כי הפונקציה שלי אינה עובדת, ושהיא "מאכילה" את המודל מספרים ריקים כלומר, מטריצות של אפסים.

## שלב הישום -

### תיאור והסבר כיצד היישום משתמש במודל

היישום שלי הוא אפלקציה שנכתבה בפייטון בעזרת הספרייה kivy. הישום שלי מקבל תמונה מהמצלמה שמחוברת למחשב, ושם של האדם לו שייכת החתימה. לאחר מכן הוא נכנס לתקייה של אותו אדם המאוחסנת על המחשב שלי, מריץ את המודל המאומן שהורדתי למחשב, מחשב את מחרק הממוצע של החתימות. על פי המרחק הוא אומר האם החתימה היא זיוף או לא.

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

תיאור פעולות הקוד -

תחילה בניתי פונקציה שמקבלת image\_path ומריצה עליו את המודל. לאחר מכן, יצרתי תקייה במחשב לכל משתתף במאגר הנתונים, עם החתימות שלו. לבסוף כאשר נרצה לבדוק האם חתימה היא מזויפת או אותנטית, נכניס לפונקציה "verify" שם של בעל החתימה ואת התמונה של החתימה אותה אנו בודקים. הפונקציה תריץ את המודל על התמונה החדשה ותבדוק את המרחק שלה מול כל אחת מן החתימות המקוריות. בנוסף, היא תבדוק אם ממוצע התוצאות מהווה תוצאה מספיק טובה בשביל לאשר את החתימה. לאחר הבדיקה הפונקציה תחזיר True - אם החתימה אותנטית או False - אם היא זיוף.

### 

**def img\_to\_encoding(image\_path, model):**

**img2 = tf.keras.preprocessing.image.load\_img(image\_path, target\_size = (160,160))**

**img2 = np.around(np.array(img2) / 255.0, decimals=12)**

**img2 = np.expand\_dims(img2, 0)**

**return model.predict(img2)**

בשורה השניה השתמשתי בפונקציה של keras אשר מקבלת קישור למיקום של תמונה, מוציאה את המידע מהקישור ומחזירה תמונה בעלת גודל מסוים. בשורה השלישת הפכתי את התמונה למטריצה של מספרים על פי rgb (בין 0-255) ולאחר מכן חילקתי את המטריצה ב255 בכדי לקבל מספרים בין 0-1. בשורה הרביעית הרחבתי את המימדים של התמונה בכדי שהתמונה תתאים למודל. בשורה החמישית החזרתי וקטור של 128, שהוא החיזוי שהמודל ביצע על התמונה.

def verify(image\_path, identity, database, model):

encoding = img\_to\_encoding(image\_path, model)

dist = 0

for i in range(len(database[identity]))):

dist += np.linalg.norm(encoding -database[identity][i])

dist = dist/len(database[identity]))

if dist<0.4:

print("It's "+str(identity) + " verified")

verify = True

10. else:

11. print("It's not " + str(identity) + " Eror")

12. verify = False

13. return dist, verify

בשורה השניה קראתי לפונקציה אשר אותה הצגתי לפני, והפעלתי אותה על התמונה שאותה אני רוצה לבדוק. בשורה הרביעית והחמישית הפעלתי לולאה שעוברת על כל החתימות של אדם מסוים ממאגר הנתונים, ומחשבת את סכום המרחקים בין כל חתימה מקורית לחתימה הנבדקת. בשורה השישית חלקתי את סכום המרחקים במספר התמונות כדי לקבל את המרחק הממוצע. בין השורה השביעית לשנים עשר, בדקתי האם המרחק הממוצע גדול מ0.4 (המרחק שנדרש מהמודל), אם המרחק קטן מ0.4 החתימה אותנטית ואם לא היא מזויפת. בשורה האחרונה החזרתי את המרחק הממוצע והאם החתימה שהוכנסה מאושרת.

# מדריך למפתח

## אימון המודלים:

שם הקובץ הוא signature.ipynb והוא ממוקם בdrive שלי במחברת של - google colab. הקובץ מכיל את אימון המודל המקורי שלי, ואת נסיונות האימון על ResNet ועל FRmodel.

תחילה התחברתי ל - google drive שלי.

from google.colab import drive

drive.mount('/content/drive')

ברוב הפרויקטים נראה שימושים שונים בספריות שונות, פה יבאתי כמה ספריות שונות בינהן numpy, pandas, tensorflow, matplotlib,keras.

import numpy as np

import os

import glob

import tensorflow as tf

import cv2

from matplotlib import pyplot as plt

import matplotlib.image as mpimg

%matplotlib inline

import keras

from keras import layers

from keras.preprocessing.image import ImageDataGenerator, load\_img

from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D,Dropout, Flatten, Activation, RandomFlip, RandomRotation, Resizing, Rescaling, experimental

from keras.layers import concatenate, Lambda, Embedding, Input

import random

פה עשיתי את הצעד הראשון בבניית הdatabase שלי, בניתי מילון המכיל את כל המשתתפים שחתמו וזיפו חתימות בשבילי, והגדרתי בעבור כל אחד מהם רשימה משלו.

true\_signature = {}

true\_signature["Itzik Yeyni"] = []

true\_signature["Or Yeyni"] = []

true\_signature["Shalev Yeyni"] = []

true\_signature["Tamar Founis"] = []

true\_signature["Eyal Moskowitz"] = []

true\_signature["Roni Waltman"] = []

true\_signature["Adina Waltman"] = []

true\_signature["Liran Yeyni"] = []

הפונקציה הבאה מקבלת שם של משתתף, ניגשת אל קובץ החתימות שלו ומשם לוקחת את כל התמונות בעלות הסיומת 'jpeg' ומוסיפה אותם אל הרשימה של אותו משתתף.

def make\_data(name,lst):# משתנה הפונקציה הוא שם פרטי ושם משפחה.

name2 = name.split(' ')# (ניצור רשימה לכל שם (פרטי\משפחה

a = '/content/drive/MyDrive/'+name2[0]+name2[1]+'/\*.jpeg'

for img in glob.glob(a): # בעבור כל קובץ עם סיומת 'jprg'

img2 = tf.keras.preprocessing.image.load\_img(img, target\_size = (160,160))# טעינת התמונה

img2 = np.around(np.array(img2) / 255.0, decimals=12)#הפיכה של התמונה למטריצה

lst[name].append(img2)# הוספה של מריצת התמונה לבעל השם

הכנסת התמונות האותנטיות של כל אדם אל רשימתו.

for i in true\_signature:

make\_data(i)

הכנסת הזיופים לרשימה משלהם.

fake\_signature = []

for img in glob.glob('/content/drive/MyDrive/FakeSignature2/\*.jpeg'):

img2 = tf.keras.preprocessing.image.load\_img(img, target\_size = (160,160))

img2 = np.around(np.array(img2) / 255.0, decimals=12)

fake\_signature.append(img2)

הצגת החתימות האותנטיות של כל אדם במאגר נתונים.

x = 1

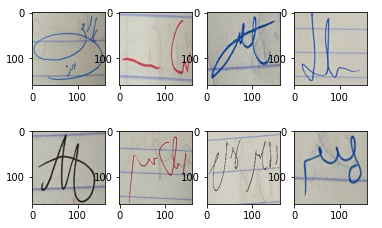
for i in true\_signature:

num = random.randint(0,5)#תמונה רנדומלית מכל אדם

ax = plt.subplot(3, 4, x)#מיקום התמונה על המסך

plt.imshow(true\_signature[i][num]) #הצגת התמונה

x+=1



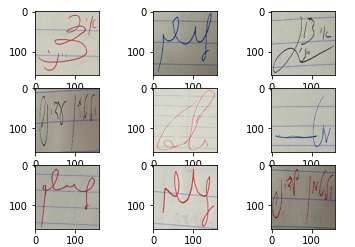
הצגת החתימות המזויפות.

for i in range(9):

ax = plt.subplot(3, 3, 1+i)#מיקום התמונה על המסך

num = random.randint(0,len(fake\_signature)-1)

plt.imshow(fake\_signature[num])#הצגת התמונה



הקוד הבא מחלק את התמונות לשלוש רשימות כך שבכל רשימה במיקום מסוים ההתמונה מ - anchor, והתמונה מ - positive, יהיו שתי חתימות אותנטיות של אותו אדם. בעוד שהתמונה ב - negative תהיה של חתימה שונה או מזויפת.

anchor = []

positive = []

negative = []

for per in true\_signature: # בעבור כל אדם

for i in range(6): # מעבר על התמונות האישיות

for j in range(6):# מעבר על התמונות האישיות

if j != i:

for g in range(len(fake\_signature)):# מעבר על התמונות המזויפות

anchor.append(np.expand\_dims(true\_signature[per][i], 0))#הוספה

positive.append(np.expand\_dims(true\_signature[per][j], 0))#של

negative.append(np.expand\_dims(fake\_signature[g],0))#התמונות

for per2 in true\_signature:

if per2 != per:

for p in range(6):# הוספה של התמונות השונות מאנשים שונים

anchor.append(np.expand\_dims(true\_signature[per][i], 0))

positive.append(np.expand\_dims(true\_signature[per][j], 0))

negative.append(np.expand\_dims(true\_signature[per2][p], 0))

הסבר הקוד:

תחילה שלוש רשימות מוגדרות (anchor, negative, positive). לאחר מכן עבור כל אדם במאגר הנתונים שלנו נעבור על התמונות האותנטיות שלו ובעבור כל זוג תמונות אותנטיות שונות נוסיף אותן לרשימות המתאימות עם חתימה שונה מהם.

הפונקציה הבאה אחראית על חשיפת המודל ל - triplets בזמן האימון. הפונקציה לוקחת דוגמה רנדומלית מהשלישיות ומוסיפה את הדוגמה לשלוש רשימות, היא חוזרת על התהליך לפי מספר ה - batch\_size. לאחר מכן היא שולחת את הקבוצה אל המודל, כל פעם לפי בקשתו.

def gen\_batch(batch\_size = 64):

while True:

a = []

p = []

n = []

for \_ in range(batch\_size):

num = random.randint(0, len(anchor)-1)

a.append(anchor[num])

p.append(positive[num])

n.append(negative[num])

yield (np.array(a), np.array(p), np.array(n)), np.zeros((batch\_size, 1)).astype("float32")[0]

זאת היא פונקציית הloss המיוחדת לtriplet loss שהסברתי עליה בפירוט מקודם. הפונקציה מקבלת את את הanchor, positive, negative כוקטור משולב של שלושתם ומפרידה אותו לאחד .לאחר מכן הפונקציה מחשבת את המרחק בין המשתנים החיוביים והשליליים מחסרת, עושה מקסימום עם 0 ומחזירה את התוצאה.

def triplet\_loss(y\_true, y\_pred, margin =0.4):

anchor, positive, negative = y\_pred[:, 0:128], y\_pred[:, 128:256], y\_pred[:, 256:384]#פיצול הוקטור

pos\_dist = tf.reduce\_sum(tf.square(tf.subtract(anchor,positive)),axis=-1)# חישוב מרחק חיובי

neg\_dist = tf.reduce\_sum(tf.square(tf.subtract(anchor,negative)),axis=-1)# חישוב מרחק שלילי

basic\_loss = pos\_dist - neg\_dist + margin

loss = tf.reduce\_mean(tf.maximum(basic\_loss,0))

return loss

זה המודל הסופי שהכנתי בעצמי והפעלתו (המשתנים הוסברו ב"שכבות שונות ברשת").

def make\_model(input\_shape):

inputs = keras.Input(shape = input\_shape)

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)

x = layers.MaxPooling2D((2, 2))(x)

x = layers.BatchNormalization()(x)

x = layers.Dropout(0.2)(x)

x = layers.Conv2D(64, (3, 3),strides = 1, padding = 'same', activation = 'relu')(x)

x = layers.Conv2D(64, 3,strides = 1, padding = 'same', activation = 'relu')(x)

x = layers.MaxPooling2D((2, 2))(x)

x = layers.BatchNormalization()(x)

x = layers.Dropout(0.2)(x)

x = layers.Conv2D(128, (3, 3),strides = 1, padding = 'same', activation = 'relu')(x)

x = layers.MaxPooling2D((2, 2))(x)

x = layers.BatchNormalization()(x)

x = layers.Conv2D(256, (3, 3),strides = 1, padding = 'same', activation = 'relu')(x)

x = layers.MaxPooling2D((2, 2))(x)

x = layers.BatchNormalization()(x)

x = layers.Conv2D(512, 3,strides = 1, padding = 'same', activation = 'relu')(x)

x = layers.MaxPooling2D((2, 2))(x)

x = layers.BatchNormalization()(x)

x = layers.Dropout(0.2)(x)

x = layers.Flatten()(x)

x = layers.Dense(512, activation='sigmoid')(x)

x = layers.Dropout(0.2)(x)

x = tf.keras.layers.Lambda(lambda x: tf.math.l2\_normalize(x, axis=1))(x)

outputs = layers.Dense(128, activation='softmax')(x)

return keras.Model(inputs, outputs)

model = make\_model((160, 160, 3))

model.summary()

הקוד הבא מייצר את התבנית של המודל שאיתו יבוצע האימון. תחילה מוגדרים שלושה משתנים המצפים לקבל input בגודל מסוים, לאחר מכן נכין את ה - output של המודל על ידי הרצת המודל מהקוד הקודם על כל אחד וחיבור שלושת הוקטורים לאחד בעזרת הפונקציה - Concatenate. לבסוך נקרא לפונקציה של keras המכינה לנו מודל ונכניס את הinput וה output.

from keras.layers.merge import Concatenate

triplet\_model\_anchor = Input((160,160, 3))

triplet\_model\_positive = Input((160, 160, 3))

triplet\_model\_negative = Input((160, 160, 3))

triplet\_model\_out = Concatenate()([model(triplet\_model\_anchor), model(triplet\_model\_positive), model(triplet\_model\_negative)])# חיבור המשתנים יחדיו

triplet\_model = keras.Model([triplet\_model\_anchor, triplet\_model\_positive, triplet\_model\_negative], triplet\_model\_out)# הכנת המודל

triplet\_model.summary()

display(tf.keras.utils.plot\_model(triplet\_model, dpi = 70, show\_shapes = True, expand\_nested = True))

עכשיו קימפלתי את המודל בעזרת האופטימיזר אדאם ופונקציית הloss שלי.

triplet\_model.compile(optimizer = tf.keras.optimizers.Adam(), loss = triplet\_loss)

נריץ ונאמן את המודל בקבוצות של 64, 300 epochs ו1000 צעדים בכל epoch.

history = triplet\_model.fit(gen\_batch(64), steps\_per\_epoch = 1000, epochs = 300)

נציג את ההתקדמות של המודל בלמידת הloss במהלך הepochs.

plt.plot(history.history['loss'])

plt.title('Model Loss - Adam')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.show()

שמירת המודל לתקייה בדרייב שלי.

model.save("/content/drive/MyDrive/best\_model")

ייבאתי מהגוגל דרייב שלי את המודל השמור של FRmodel (מודל האומן על ידי אותה פונקציית loss אך על פנים) תוך הקפאת השכבות של המודל.

FRmodel = tf.keras.models.load\_model('/content/drive/MyDrive/facenet\_keras.h5')

FRmodel.load\_weights('/content/drive/MyDrive/facenet\_keras\_weights2.h5')

FRmodel.trainable = False

FRmodel.summary()

חזרה על תהליך בניית המודל שעליו יבוצע האימון וקימפולו אך עם אפשרות למידה קטנה.

from keras.layers.merge import Concatenate

triplet\_model\_out1 = Concatenate()([FRmodel(triplet\_model\_anchor), FRmodel(triplet\_model\_positive), FRmodel(triplet\_model\_negative)])

triplet\_model1 = keras.Model([triplet\_model\_anchor, triplet\_model\_positive, triplet\_model\_negative], [triplet\_model\_out1])

triplet\_model1.summary()

display(tf.keras.utils.plot\_model(triplet\_model1, dpi = 70, show\_shapes = True, expand\_nested = True))

triplet\_model1.compile(optimizer = tf.keras.optimizers.Adam(1e-5), loss = triplet\_loss)

אימון קטן של המודל בכדי לחשוף את המודל לחתימות.

history = triplet\_model1.fit(gen\_batch(batch\_size = 64), steps\_per\_epoch = 150, epochs = 15)

ביטול הקפאת שכבות המודל ואיפשור למידה.

FRmodel.trainable = True

triplet\_model1.compile(optimizer = tf.keras.optimizers.Adam(), loss = triplet\_loss)

אימון המודל.

history = triplet\_model1.fit\_generator(gen\_batch(batch\_size = 64), steps\_per\_epoch = 500, epochs = 32)

שמירת המודל לדרייב.

FRmodel.save("/content/drive/MyDrive/FRmodel")

פה חזרתי על תהליך האימון שוב עם המודל ResNet50ועם הweights של- imagenet:

יבוא של ResNet50 והקפאת השכבות.

base\_model = tf.keras.applications.ResNet50(weights="imagenet", input\_shape=(160,160,3),include\_top=False)

base\_model.trainable = False

הכנת המודל לקבלת תמונות מגודל מסוים וקביעת output.

from tensorflow.python.ops.gen\_dataset\_ops import optional\_get\_value

from tensorflow.python import training

inputs = keras.Input(shape = (160,160,3))

x = base\_model(inputs)

x = keras.layers.GlobalAveragePooling2D()(x)

outputs = keras.layers.Dense(128, activation = 'sigmoid')(x)

modelX = keras.Model(inputs, outputs)

modelX.summary()

בניית המודל המשמש לאימון.

from keras.layers.merge import Concatenate

triplet\_model\_out1 = Concatenate()([modelX(triplet\_model\_anchor), modelX(triplet\_model\_positive), modelX(triplet\_model\_negative)])

triplet\_modelX = keras.Model([triplet\_model\_anchor, triplet\_model\_positive, triplet\_model\_negative], [triplet\_model\_out1])

triplet\_modelX.summary()

display(tf.keras.utils.plot\_model(triplet\_modelX, dpi = 70, show\_shapes = True, expand\_nested = True))

triplet\_modelX.compile(optimizer = tf.keras.optimizers.Adam(), loss = triplet\_loss)

אימון המודל.

history = triplet\_modelX.fit(gen\_batch(), steps\_per\_epoch = 250, epochs = 15)

ביטול ההקפאה והמשך אימון.

base\_model.trainable = True

triplet\_modelX.compile(optimizer = tf.keras.optimizers.Adam(1e-5), loss = triplet\_loss)

history = triplet\_modelX.fit\_generator(gen\_batch(), steps\_per\_epoch = 500, epochs = 32)

שמירת המודל

modelX.save("/content/drive/MyDrive/Xmodel")

## בדיקת ביצועים של המודלים:

שם הקובץ הוא testing.ipynb והוא ממוקם בdrive שלי במחברת של - google colab. בקובץ ישנם פונקציות שונות הבוחנות את הביצועים של המודלים השונים.

הקובץ מתחיל כמו הקודם (אותו קוד), תחילה נתחבר לדרייב, נייבא ספריות שונות ונטען את מסד הנתונים עליו בוצע האימון.

from google.colab import drive

drive.mount('/content/drive')

import pandas as pd

import numpy as np

import os

import glob

import tensorflow as tf

import cv2

from matplotlib import pyplot as plt

import matplotlib.image as mpimg

import random

%matplotlib inline

import keras

from keras import layers

from keras.preprocessing.image import ImageDataGenerator, load\_img

from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D,Dropout, Flatten, Activation, RandomFlip, RandomRotation, Resizing, Rescaling, experimental

from keras.layers import concatenate, Lambda, Embedding, Input

import random

import cv2

import pathlib

true\_signature = {}

true\_signature["Itzik Yeyni"] = []

true\_signature["Or Yeyni"] = []

true\_signature["Shalev Yeyni"] = []

true\_signature["Tamar Founis"] = []

true\_signature["Eyal Moskowitz"] = []

true\_signature["Roni Waltman"] = []

true\_signature["Adina Waltman"] = []

true\_signature["Liran Yeyni"] = []

def make\_data(name, lst):

name2 = name.split(' ')

a = '/content/drive/MyDrive/'+name2[0]+name2[1]+'/\*.jpeg'

for img in glob.glob(a):

img2 = tf.keras.preprocessing.image.load\_img(img, target\_size = (160,160))

img2 = np.around(np.array(img2) / 255.0, decimals=12)

lst[name].append(img2)

for i in true\_signature:

make\_data(i, true\_signature)

fake\_signature = []

for img in glob.glob('/content/drive/MyDrive/FakeSignature2/\*.jpeg'):

img2 = tf.keras.preprocessing.image.load\_img(img, target\_size = (160,160))

img2 = np.around(np.array(img2) / 255.0, decimals=12)

fake\_signature.append(img2)

לאחר השלבים הראשונים נייבא את המודלים שנרצה לבחון.

model\_triplet2 = tf.keras.models.load\_model('/content/drive/MyDrive/model\_triplet2', compile=False)

FRmodel = tf.keras.models.load\_model('/content/drive/MyDrive/FRmodel', compile=False)

imagenet\_model = tf.keras.models.load\_model('/content/drive/MyDrive/Xmodel', compile=False)

החוברת בנויה כך שהיא בוחנת מודל יחיד בשם model לכן כל פעם שנרצה לבחון מודל מסוים נגדיר אותו תחת השם model.

model = model\_triplet2

קודם נבדוק את התוצאות שהמודל מחזיר על תמונות שונות ואותנטיות עלהים הוא התאמן.

x = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][0], 0))

y = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][1], 0))

z = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][0], 0))

print(np.linalg.norm(x - y), np.linalg.norm(x - z))

print(np.sum(np.square(x - y), axis=-1), np.sum(np.square(x - z), axis=-1))

x = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][5], 0))

y = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][3], 0))

z = model.predict(np.expand\_dims(true\_signature['Tamar Founis

'][2], 0))

print(np.linalg.norm(x - y), np.linalg.norm(x - z))

print(np.sum(np.square(x - y), axis=-1), np.sum(np.square(x - z), axis=-1))

x = model.predict(np.expand\_dims(true\_signature['Liran Yeyni

'][2], 0))

y = model.predict(np.expand\_dims(true\_signature['Liran Yeyni

'][5], 0))

z = model.predict(np.expand\_dims(true\_signature['Roni Waltman

'][2], 0))

print(np.linalg.norm(x - y), np.linalg.norm(x - z))

print(np.sum(np.square(x - y), axis=-1), np.sum(np.square(x - z), axis=-1))

בשביל לבחון איך המודל עובד עם זיהוי חתימות כאשר מדובל בצבעים שונים, נדפיס את החתימות של אדם מסוים ממסד הנתונים נסדר את החתימות לפי צבע ונראה איך המודל מגיד כאשר האינטרקציה היא בין תמונות זהות מצבע שונה וזהה.

count = 0

for i in range(6):

ax = plt.subplot(2, 3, 1+count)

plt.imshow(true\_signature['Itzik Yeyni'][i])# הדפסת כל התמונות של אדם

count += 1

תחילה נבדוק את תוצאות המודל על כל אחד מן הצבעים כנגד עצמו על ידי חישוב מרחק בין התמונות בעזרת המודל.

print("the color black")

x = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][0], 0))

y = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][4], 0))

print(np.linalg.norm(x - y))

print(np.sum(np.square(x - y), axis=-1))

print("the color blue")

x = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][1], 0))

y = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][3], 0))

print(np.linalg.norm(x - y))

print(np.sum(np.square(x - y), axis=-1))

print("the color red")

x = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][2], 0))

y = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][5], 0))

print(np.linalg.norm(x - y))

print(np.sum(np.square(x - y), axis=-1))

פה נבחן את תוצאות המודל על חתימות אותנטיות כאשר הצבעים שונים על ידי חישוב מרחקים בין תמונות בעזרת המודל.

print("the color red and black")

x\_red = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][2], 0))

y\_red = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][5], 0))

x\_black = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][0], 0))

y\_black = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][4], 0))

lst\_red = [x\_red, y\_red]

lst\_black = [x\_black, y\_black]

for i in lst\_red:

for j in lst\_black:

print(np.linalg.norm(i - j))

print(np.sum(np.square(i - j), axis=-1))

print()

print("the color red and blue")

x\_red = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][2], 0))

y\_red = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][5], 0))

x\_blue = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][1], 0))

y\_blue = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][3], 0))

lst\_red = [x\_red, y\_red]

lst\_blue = [x\_blue, y\_blue]

for i in lst\_red:

for j in lst\_blue:

print(np.linalg.norm(i - j))

print(np.sum(np.square(i - j), axis=-1))

print()

print("the color black and blue")

x\_black = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][0], 0))

y\_black = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][4], 0))

x\_blue = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][1], 0))

y\_blue = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][3], 0))

lst\_red = [x\_black, y\_black]

lst\_blue = [x\_blue, y\_blue]

for i in lst\_red:

for j in lst\_blue:

print(np.linalg.norm(i - j))

print(np.sum(np.square(i - j), axis=-1))

print()

נחזור על אותן פעולות על אדם נוסך בכדי לבדוק את עקביות התוצאות.

count = 0

for i in range(6):

ax = plt.subplot(2, 3, 1+count)

plt.imshow(true\_signature['Or Yeyni'][i])

count += 1

print("the color black")

x = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][1], 0))

y = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][5], 0))

print(np.linalg.norm(x - y))

print(np.sum(np.square(x - y), axis=-1))

print("the color blue")

x = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][0], 0))

y = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][2], 0))

print(np.linalg.norm(x - y))

print(np.sum(np.square(x - y), axis=-1))

print("the color red")

x = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][3], 0))

y = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][4], 0))

print(np.linalg.norm(x - y))

print(np.sum(np.square(x - y), axis=-1))

print("the color red and black")

x\_red = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][3], 0))

y\_red = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][4], 0))

x\_black = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][1], 0))

y\_black = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][5], 0))

lst\_red = [x\_red, y\_red]

lst\_black = [x\_black, y\_black]

for i in lst\_red:

for j in lst\_black:

print(np.linalg.norm(i - j))

print(np.sum(np.square(i - j), axis=-1))

print()

print("the color red and blue")

x\_red = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][3], 0))

y\_red = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][4], 0))

x\_blue = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][0], 0))

y\_blue = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][2], 0))

lst\_red = [x\_red, y\_red]

lst\_blue = [x\_blue, y\_blue]

for i in lst\_red:

for j in lst\_blue:

print(np.linalg.norm(i - j))

print(np.sum(np.square(i - j), axis=-1))

print()

print("the color black and blue")

x\_black = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][1], 0))

y\_black = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][5], 0))

x\_blue = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][0], 0))

y\_blue = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][2], 0))

lst\_red = [x\_black, y\_black]

lst\_blue = [x\_blue, y\_blue]

for i in lst\_red:

for j in lst\_blue:

print(np.linalg.norm(i - j))

print(np.sum(np.square(i - j), axis=-1))

פה נעשת בדיקה בכדי לראות איך המודל מתמודד עם זיופים קלים ולאחר מכן קשים על שני אנשים שונים.

תחילה נגדיר חתימות מזויפות לבחינה בעבור שני משתתפים.

print("lets see if the computer can identify a fake, the first will be easy and the second hard")

itz\_fake = [fake\_signature[29], fake\_signature[0]]

or\_fake = [fake\_signature[76], fake\_signature[1]]

נבחן איך המודל מחשב את המרחק בין החתימות המזויפות לבין כל החתימות האותנטיות של המשתתפים.

flag = True

for i in or\_fake:

x = model.predict(np.expand\_dims(i, 0))

for j in range(6):

if flag:

print("easy")

else:

print("hard")

y = model.predict(np.expand\_dims(true\_signature['Or Yeyni'][j], 0))

print(np.linalg.norm(x - y))

print(np.sum(np.square(x - y), axis=-1))

print()

flag = False

flag = True

for i in itz\_fake:

x = model.predict(np.expand\_dims(i, 0))

for j in range(6):

if flag:

print("easy")

else:

print("hard")

y = model.predict(np.expand\_dims(true\_signature['Itzik Yeyni'][j], 0))

print(np.linalg.norm(x - y))

print(np.sum(np.square(x - y), axis=-1))

print()

flag = False

לאחר הבדיקות השונות על המודל נבחן את אחוזי הדיוק של המודל כל חתימות אותנטיות, מזויפות ולבסוף שילוב של השניים על החתימות שהתאמן עליהן (train) .

הקוד הבא משווה בין כל חתימה של משתתף לבין שאר החתימות האותנטיות שלו, אם התוצאה קטנה מ0.45 היא נכונה. לבסוך נחזיר את אחוז הדיוק (כמות הניחושים הנכונים חלקי הכמות הכוללת של השוואות).

count = 0

count2 = 0

for i in true\_signature:

for j in range(5):

x = model.predict(np.expand\_dims(true\_signature[i][j], 0))

for p in range(6):

if p -j >0:

count2+=1

y = model.predict(np.expand\_dims(true\_signature[i][p], 0))

if np.linalg.norm(x - y) < 0.45:

count+=1

print(count/count2)

הקוד הבא משווה בין כל החתימות של כל משתתף לכל החתימות המזויפות והשונות, אם התוצאה גדולה מ0.45 המודל צדק. לבסוף נחזיר את אחוז הדיוק (כמות הניחושים הנכונים חלקי הכמות הכוללת של השוואות).

count3 = 0

for t in true\_signature:

for i in range(6):

x = model.predict(np.expand\_dims(true\_signature[t][i], 0))

for j in range(len(fake\_signature)):

y = model.predict(np.expand\_dims(fake\_signature[j], 0))

if np.linalg.norm(x - y) > 0.45:

count3+=1

print(count3/(6\*len(fake\_signature)\*len(true\_signature)))

לבסוף נחשב את אחוז הדיוק הכולל (כמות הניחושים הנכונים חלקי הכמות הכוללת של השוואות).

print((count+count3)/(count2 +6\*len(fake\_signature)\*len(true\_signature)))

לאחר העבודה עם החתימות עליהם התאמנו נבנה את ה - val\_data.

val\_signature = {}

val\_signature["Jonathan Assa2"] = []

val\_signature["Oren Faust"] = []

val\_signature["Rachel Founis2"] = []

for i in val\_signature:

make\_data(i, val\_signature)

fake\_val\_signature = []

for img in glob.glob('/content/drive/MyDrive/FakeValidation2/\*.jpeg'):

img2 = tf.keras.preprocessing.image.load\_img(img, target\_size = (160,160))

img2 = np.around(np.array(img2) / 255.0, decimals=12)

fake\_val\_signature.append(img2)

כעת נבדוק בעזרת אותן הפונקציות כמו מקודם את אחוז הדיוק של המודל על חתימות שהוא לא ראה אף פעם.

count = 0

count2 = 0

for i in val\_signature:

for j in range(5):

x = model.predict(np.expand\_dims(val\_signature[i][j], 0))

for p in range(6):

if p -j >0:

count2+=1

y = model.predict(np.expand\_dims(val\_signature[i][p], 0))

if np.linalg.norm(x - y) < 0.45:

count+=1

print(count/count2)

count3 = 0

for t in val\_signature:

for i in range(6):

x = model.predict(np.expand\_dims(val\_signature[t][i], 0))

for j in range(len(fake\_val\_signature)):

y = model.predict(np.expand\_dims(fake\_val\_signature[j], 0))

if np.linalg.norm(x - y) > 0.45:

count3+=1

print(count3/(6\*len(fake\_val\_signature)\*len(val\_signature)))

print((count+count3)/(count2 +6\*len(fake\_val\_signature)\*len(val\_signature)))

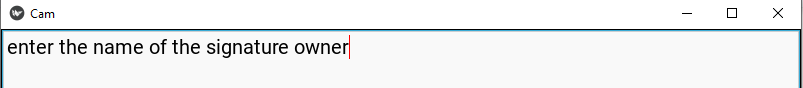
# מדריך למשתמש

## תרשים מסוג Screen flow diagram

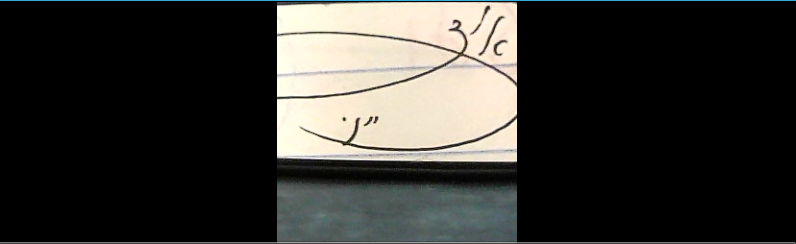


## תפקיד כל חלון באפליקציה

החלון הראשון - הכנסת שם בעל החתימה.



החלון השני - פה מוצג פלט המצלה במחשב.

החלון השלישי - מוצג כפתור המורה למצלמה לצלם ולבדוק את התמונה והפלט של המודל.

## 

## תפקיד אלמנטים

**כפתורים:** ישנו כפתור אחד בשם "verify" וכאשר הוא נלחץ תמונה תשמרת בתקייה מיוחדת, השם שהוכנס נשמר והמודל משווה בין החתימה לחתימות האמתיות.

**תיבת טקסט:** בתחילת המסך יש תיבת טקסט שם שמו של בעל החתימה יוכנס. ובתחתית ישנה תיבה שמציגה את תוצאות החיזוי (אמיתי או זיוף).

# סיכום אישי / רפלקציה

העבודה על הפרויקט הייתה קשה, אינטנסיבית ומלמדת בשבילי. הפרויקט היה שונה לגמרי מכל הפרויקטים שעבדנו עליהם בכיתה. כשהתחלתי את הפרויקט לא מיהרתי או חששתי ממנו מפני חשבתי שהוא יהיה כמו הפרויקט ששי (המנחה שלי) נתן לנו. חשבתי שאצטרך לעשות זיהוי תמונות פשוט, אולי להשתמש במודל מהאינטרנט ולא יותר מידי חוץ מזה. אך ההפך הוא הנכון, גיליתי שהפרויקט הנוכחי הוא לא סתם פרויקט, אי אפשר להעזר במצגות או בקודים פשוטים שניתנו מראש. אי אפשר לבקש עזרה אמיתית מהחברים מפני שהם עובדים על פרויקטים שונים בנושאים שונים. זוהי פעם ראשונה שעבדתי על פרויקט שונה מכל מה שידעתי לעשות עד כה, לגמרי לבד, ללא עזרה אמיתית וזה היה קשה. לילות רבים בהם נשארתי ער עד 3:00 כשאני אומר לעצמי כל פעם מחדש, רק עוד נסיון אחד לתקן את הארור ואני הולך לישון. כאשר התחלנו לעבוד על הפרויקט שי נתן לנו חופש עבודה - לעבוד מהבית, אך הכריח אותנו לקום ולעבוד בכל השיעורים. הוא אמר שאנחנו צריכים לעבוד לפחות 10 שעות בשבוע על הפרויקט. בהתחלה זלזלתי ואפילו בניתי קוד עם חבר שיענה לו במקומי כאשר הוא כתב לנו בוקר טוב, כל זה בשביל לישון בבוקר. אך לאחר זמן קצר ראיתי כי הפרויקט קשה ממה שחשבתי, והתחלתי לעבוד אף יותר מ10 שעות בשבוע על הפרויקט מיוזמתי. למרות העבודה הקשה נהנתי לעבוד על הפרויקט, ואני עוד יותר נהנה מהעובדה שעשיתי את העבודה לגמרי לבד וכעת, שיש לי ספר פרויקט בעל 30+ עמודים שאני כתבתי לבד בעצמי, אני פשוט מתרגש. הפרויקט הקנה לי כלים רבים, ולמדתי ממנו המון. במהלך הפרויקט למדתי מה זאת עבודה אמיתית על פרויקט, נדרשתי לחפש פרויקטים רבים דומים לשלי בכדי שאוכל לפתור בעיות, פניתי לstack overflow בשביל לחפש עזרה, מצאתי עובד הייטק שיעזור לי לנסות לפתור בעיות מסוימות, חשבתי בעצמי על פתרונות ואלגורתמים שונים ועוד. אפילו הייתי קרוב ליאוש מוחלט וויתור על הפרויקט מספר פעמים. אך בסוף המשכתי ופתרתי בעיה אחר בעיה עד שהצלחתי לסיים בדקה ה - 90. למדתי להסתדר, לשאול, לחפש, להעזר והכי חשוב לא להתייאש גם כשקשה. לפני הפרויקט לא הייתי בטוח כמה אהנה לעסוק במקצוע בעתיד, זאת בעיקר כי כל מה שחוויתי בשנת הלימודים היה למידה וישום פשוט ומודרך. החוויה של הפרויקט היא זאת שחשפה אותי לעולם האמיתי והמעניין וגרמה לי להתאהב בו.

# ביבליוגרפיה

<https://www.kaggle.com/code/stoicstatic/face-recognition-siamese-w-triplet-loss>

<https://www.youtube.com/watch?v=LKispFFQ5GU>

<https://keras.io/examples/vision/siamese_network/>

<https://www.tensorflow.org/addons/tutorials/losses_triplet>

<https://www.coursera.org/lecture/convolutional-neural-networks/triplet-loss-HuUtN>

<https://arxiv.org/pdf/1503.03832.pdf>

<https://www.youtube.com/watch?v=dG8le1YWUI8>

<https://www.youtube.com/watch?v=50cBvizRuM0>

# נספחים

## קוד האפליקציה

# Import kivy dependencies first

from kivy.app import App

from kivy.uix.boxlayout import BoxLayout

# Import kivy UX components

from kivy.uix.image import Image

from kivy.uix.button import Button

from kivy.uix.label import Label

# Import other kivy stuff

from kivy.clock import Clock

from kivy.graphics.texture import Texture

from kivy.logger import Logger

# Import other dependencies

import cv2

import tensorflow as tf

from layers import L1Dist

import os

import numpy as np

import PIL

from kivy.uix.textinput import TextInput

# Build app and layout

class CamApp(App):

def build(self):

# Main layout components

self.names = TextInput(text = "enter the name of the signature owner", multiline=False, font\_size='20sp', )

self.web\_cam = Image(size\_hint=(1, .8))

self.button = Button(text="Verify", on\_press=self.verify, size\_hint=(1, .1))

self.verification\_label = Label(text="Verification Uninitiated", size\_hint=(1, .1))

# Add items to layout

layout = BoxLayout(orientation='vertical')

layout.add\_widget(self.names)

layout.add\_widget(self.web\_cam)

layout.add\_widget(self.button)

layout.add\_widget(self.verification\_label)

# Load tensorflow/keras model

self.model = tf.keras.models.load\_model('drive-download-20220501T141246Z-001', compile = False)# custom\_objects={'L1Dist': L1Dist}

# Setup video capture device

self.capture = cv2.VideoCapture(2)

Clock.schedule\_interval(self.update, 1.0 / 33.0)

return layout

# Run continuously to get webcam feed

def update(self, \*args):

#Read frame from opencv

ret, frame = self.capture.read()

frame = frame[120:120 + 250, 200:200 + 250, :]

# Flip horizontall and convert image to texture

buf = cv2.flip(frame, 0).tostring()

img\_texture = Texture.create(size=(frame.shape[1], frame.shape[0]), colorfmt='bgr')

img\_texture.blit\_buffer(buf, colorfmt='bgr', bufferfmt='ubyte')

self.web\_cam.texture = img\_texture

# Load image from file and conver to 160x160px

def preprocess(self, file\_path):

# Read in image from file path

img2 = tf.keras.preprocessing.image.load\_img(file\_path, target\_size=(160, 160))

img2 = np.around(np.array(img2) / 255.0, decimals=12)

img2 = np.expand\_dims(img2, 0)

return img2

#def submit(self):

# Verification function to verify person

def verify(self, \*args):

verification\_threshold = 0.75

# Capture input image from our webcam

SAVE\_PATH = os.path.join('application\_data', 'input\_image', 'input\_image.jpg')

ret, frame = self.capture.read()

frame = frame[120:120 + 250, 200:200 + 250, :]

cv2.imwrite(SAVE\_PATH, frame)

name = self.names.text

# Build results array

results = []

for image in os.listdir(os.path.join('application\_data', 'verification\_images', name)):

input\_img = self.preprocess(os.path.join('application\_data', 'input\_image', 'input\_image.jpg'))

validation\_img = self.preprocess(os.path.join('application\_data', 'verification\_images', name, image))

# Make Predictions

result1 = self.model.predict(input\_img)

result2 = self.model.predict(validation\_img)

results.append(np.linalg.norm(result1 - result2))

# Detection Threshold: Metric above which a prediciton is considered positive

detection = np.sum(results)

# Verification Threshold: Proportion of positive predictions / total positive samples

verification = detection / len(os.listdir(os.path.join('application\_data', 'verification\_images', name)))

verified = verification < verification\_threshold

# Set verification text

print(verification)

self.verification\_label.text = 'Verified' if verified == True else 'Unverified'

# Log out details

Logger.info(results)

Logger.info(detection)

Logger.info(verification)

Logger.info(verified)

return results, verified

if \_\_name\_\_ == '\_\_main\_\_':

CamApp().run()

## מודלים קודמים

def make\_model(input\_shape):

inputs = keras.Input(shape = input\_shape)

x = layers.Conv2D(64, 3,strides = 1, padding = 'same', activation = 'relu')(inputs)

x = layers.Conv2D(64, 3,strides = 1, padding = 'same', activation = 'relu')(x)

x = layers.MaxPooling2D()(x)

x = layers.BatchNormalization()(x)

x = layers.Dropout(0.2)(x)

x = layers.Conv2D(128, 3,strides = 1, padding = 'same', activation = 'relu')(x)

x = layers.Conv2D(128, 3,strides = 1, padding = 'same', activation = 'relu')(x)

x = layers.MaxPooling2D()(x)

x = layers.BatchNormalization()(x)

x = layers.Dropout(0.2)(x)

x = layers.Conv2D(256, 3,strides = 1, padding = 'same', activation = 'relu')(x)

x = layers.Conv2D(256, 3,strides = 1, padding = 'same')(x)

x = layers.MaxPooling2D()(x)

x = layers.BatchNormalization()(x)

x = layers.Conv2D(512, 3,strides = 1, padding = 'same', activation = 'relu')(x)

x = layers.Conv2D(512, 3,strides = 1, padding = 'same')(x)

x = layers.MaxPooling2D()(x)

x = layers.BatchNormalization()(x)

x = layers.Dense(1024, activation='sigmoid')(x)

x = layers.Dropout(0.2)(x)

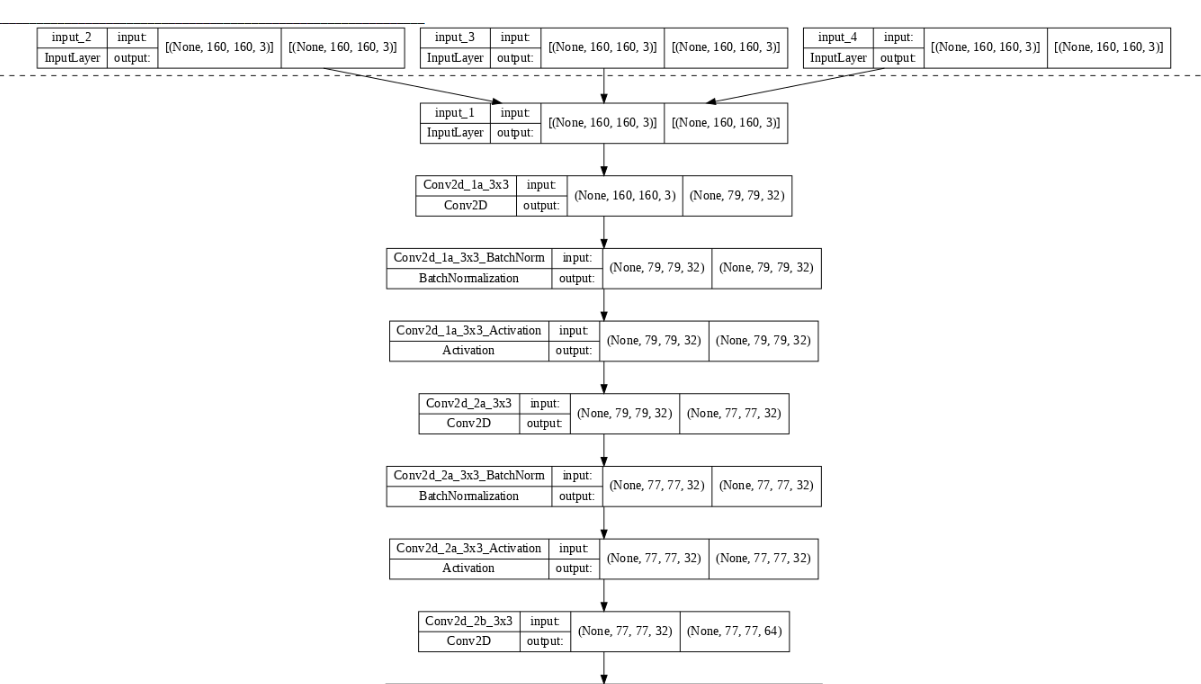
x = layers.MaxPooling2D()(x)

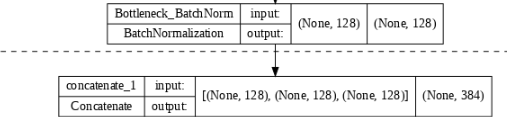
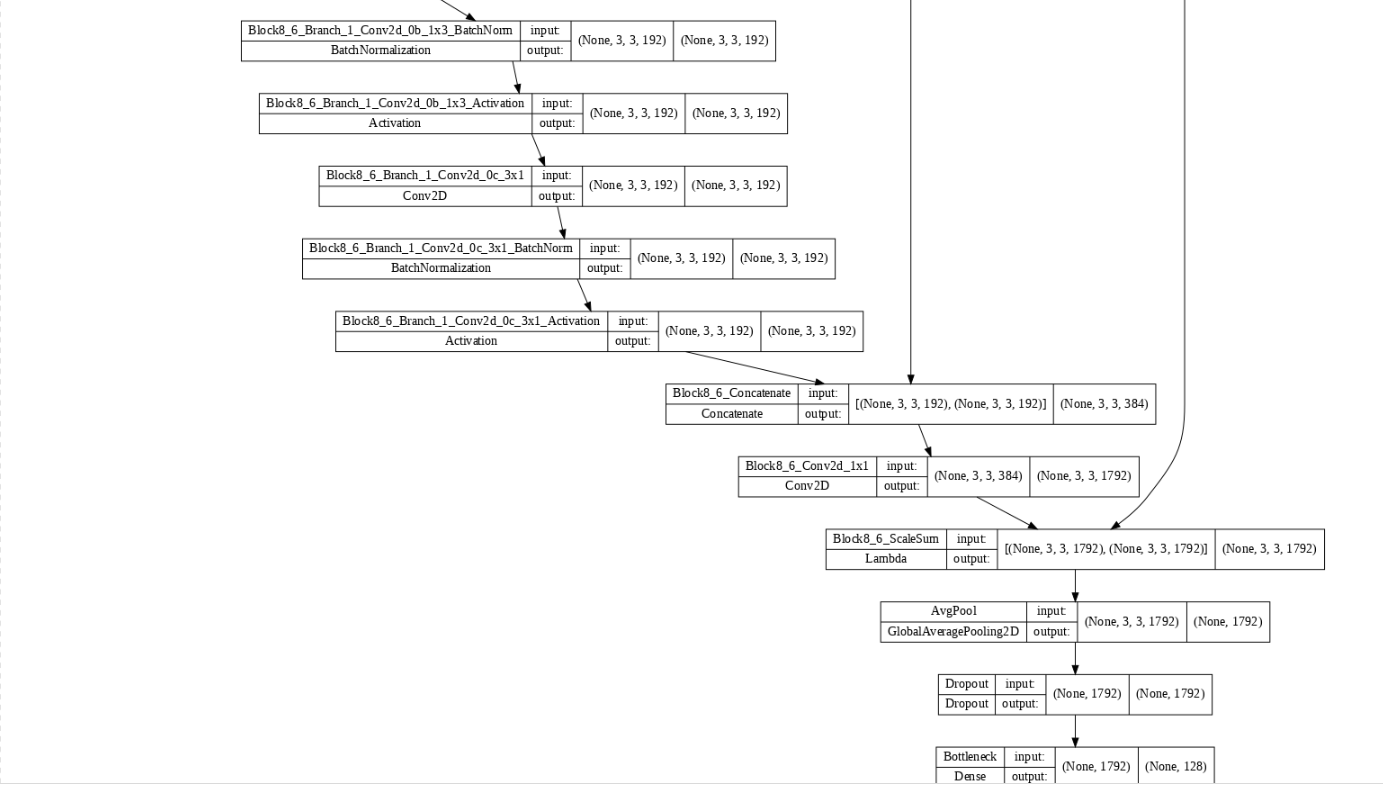
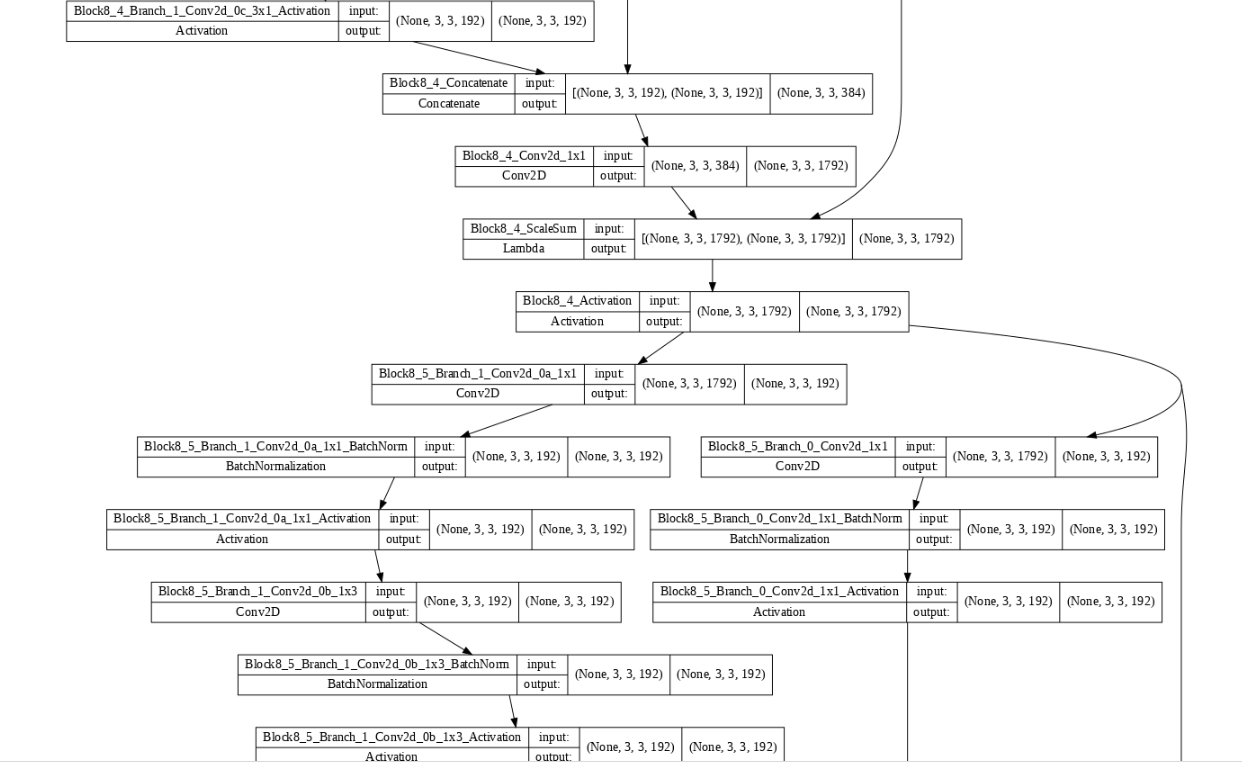
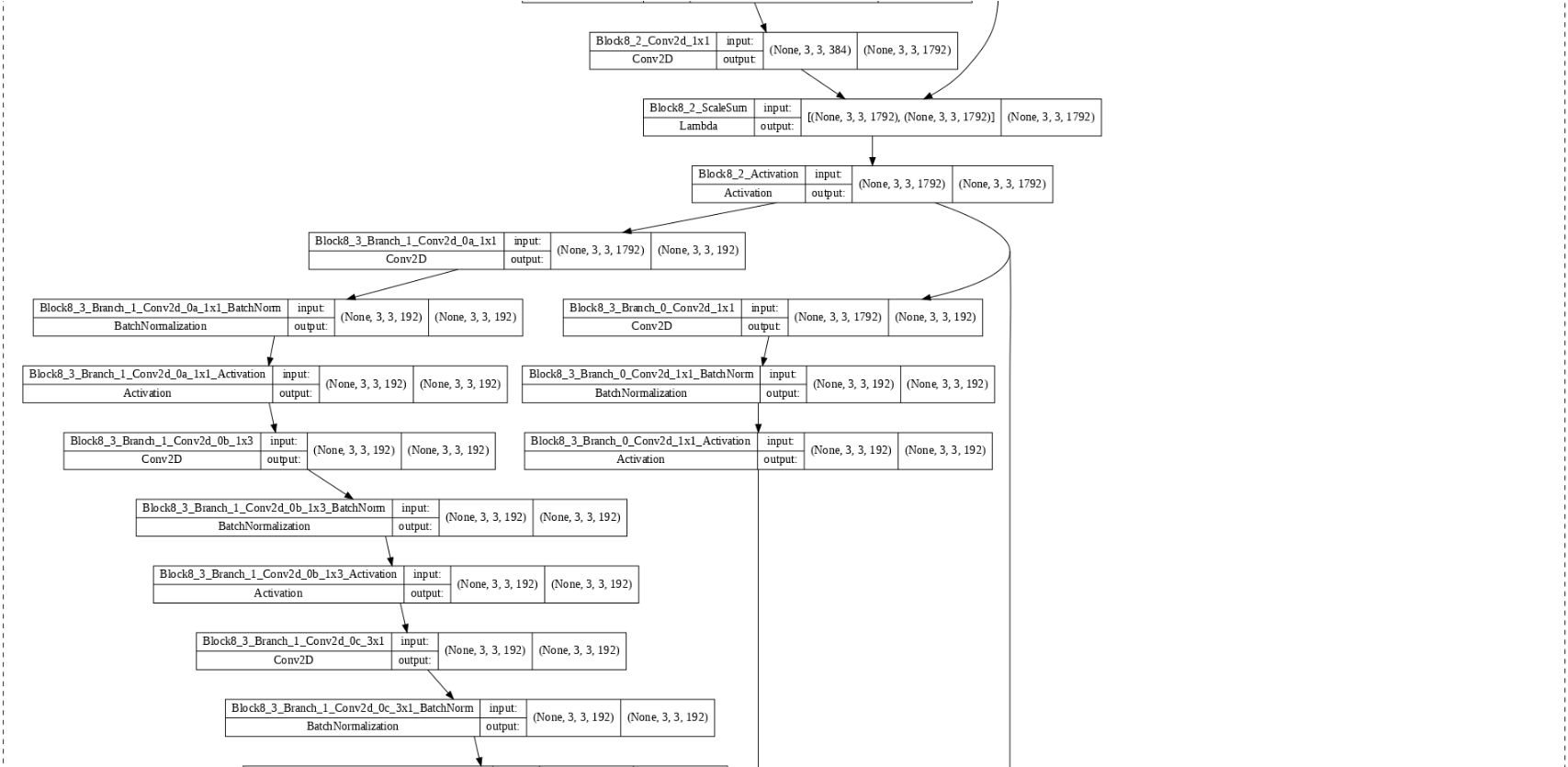
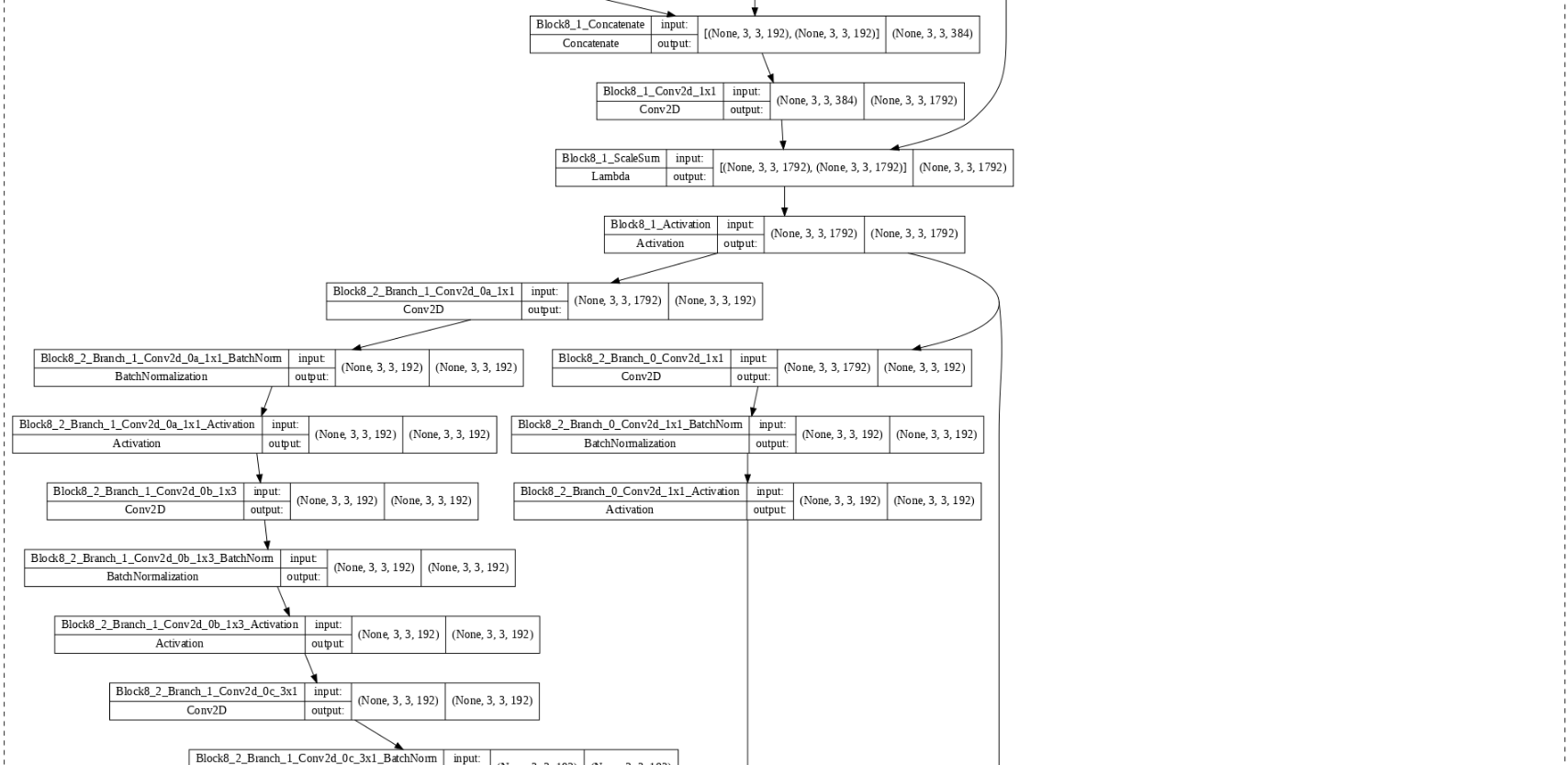
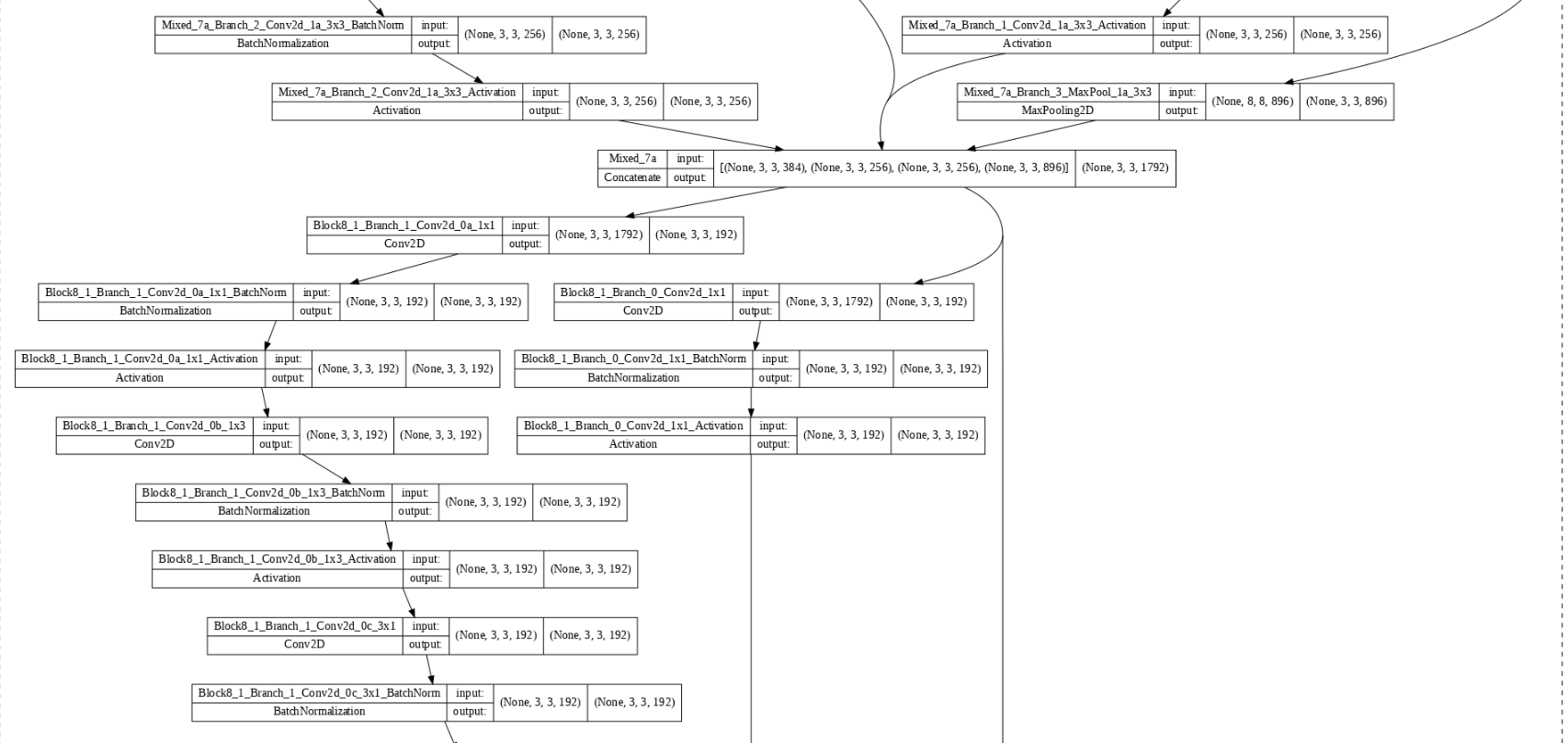
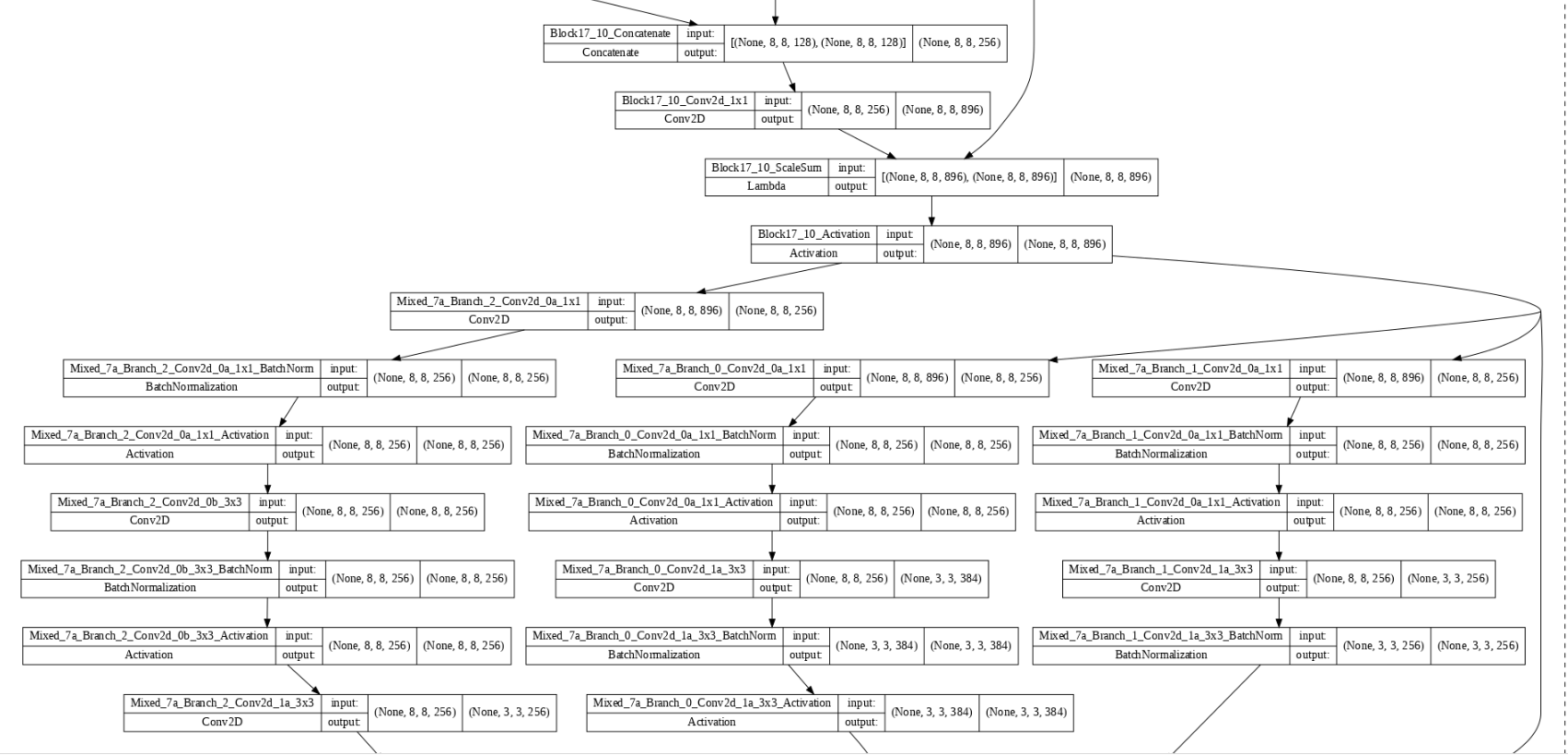
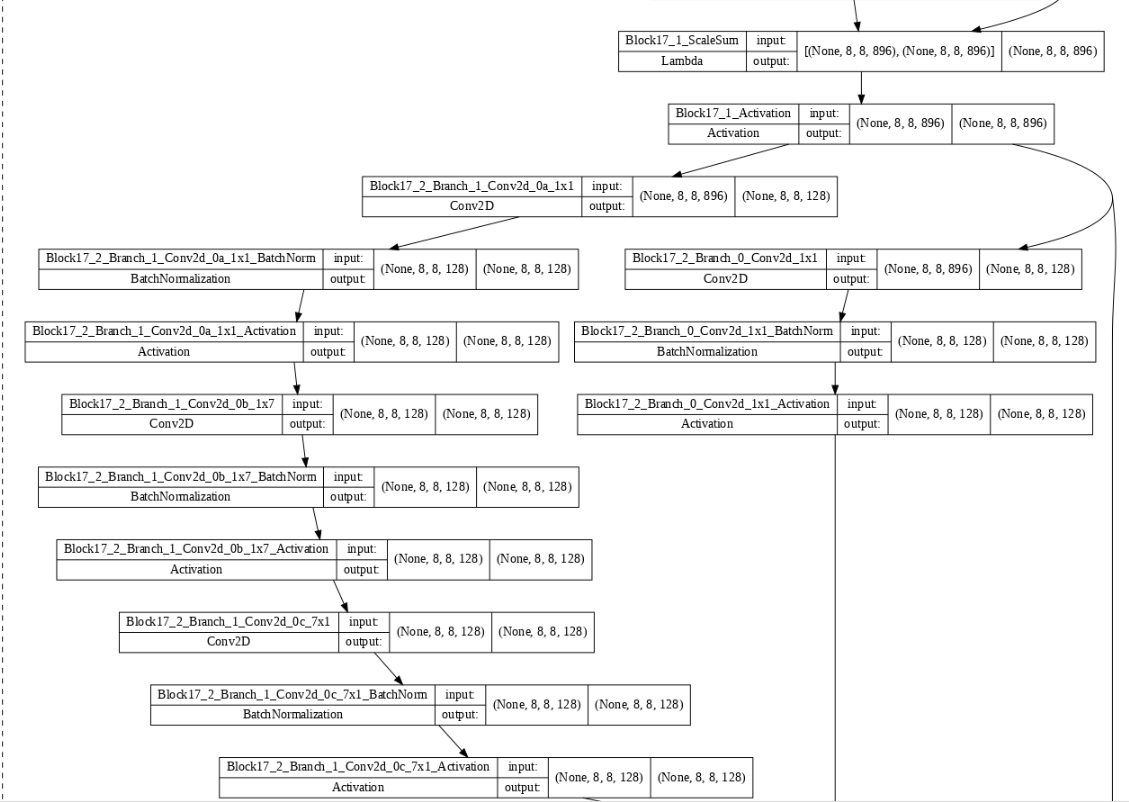
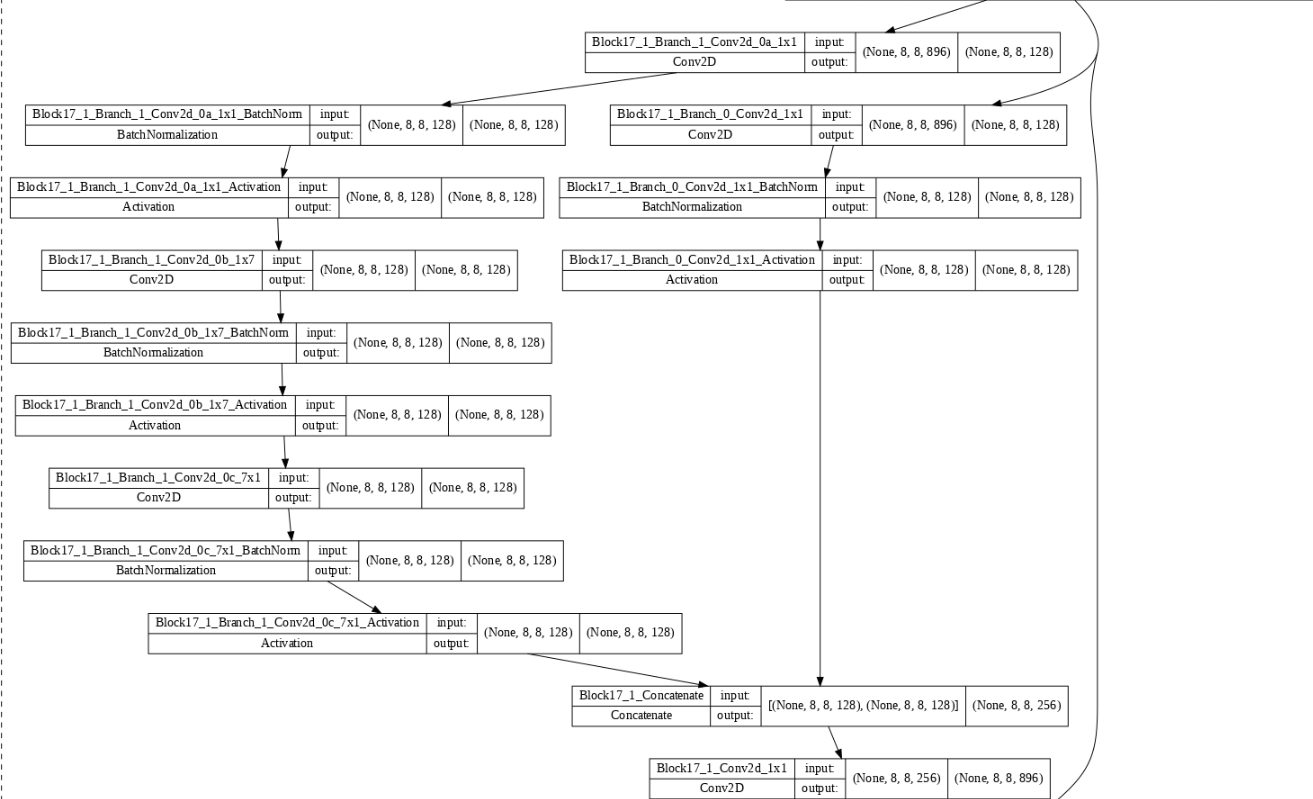
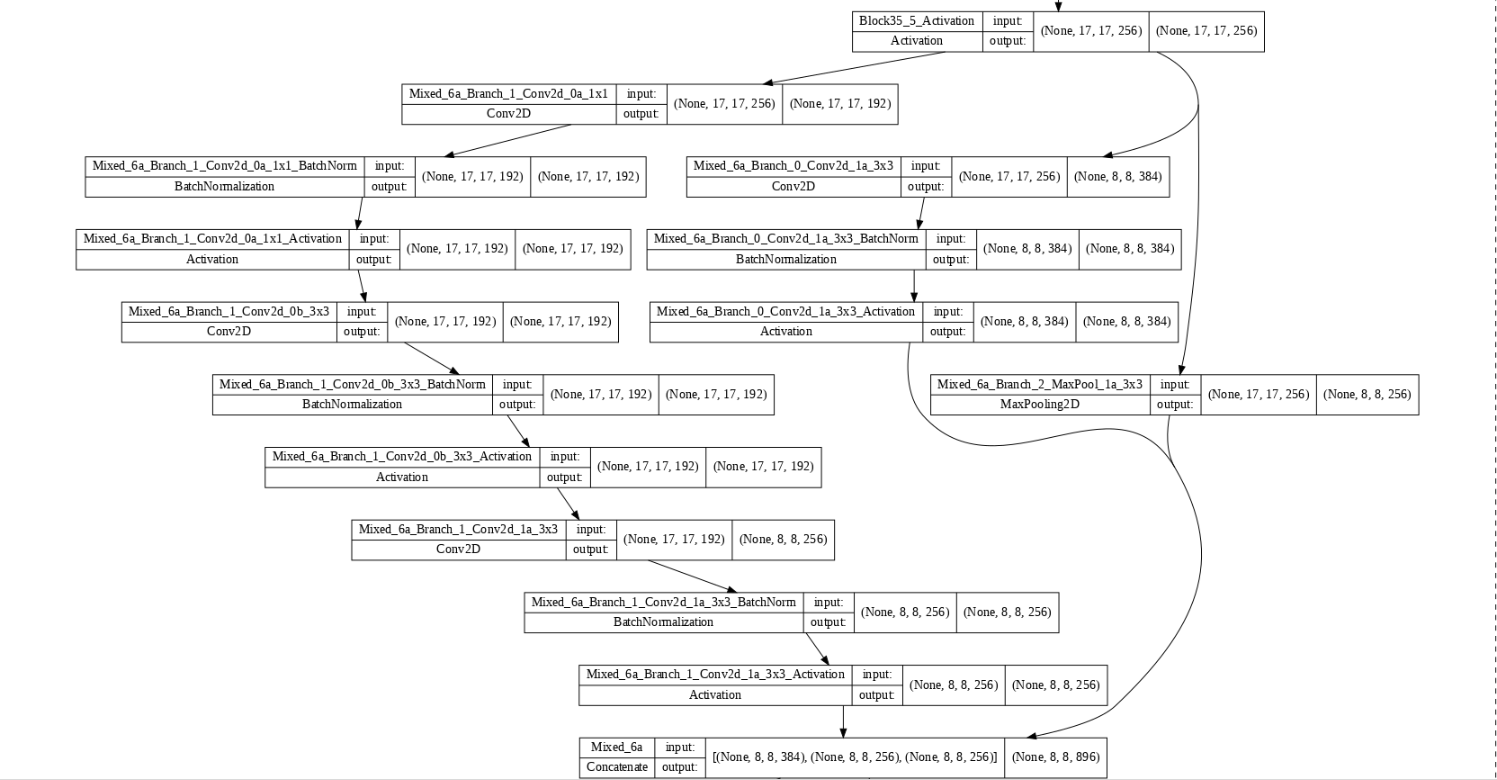
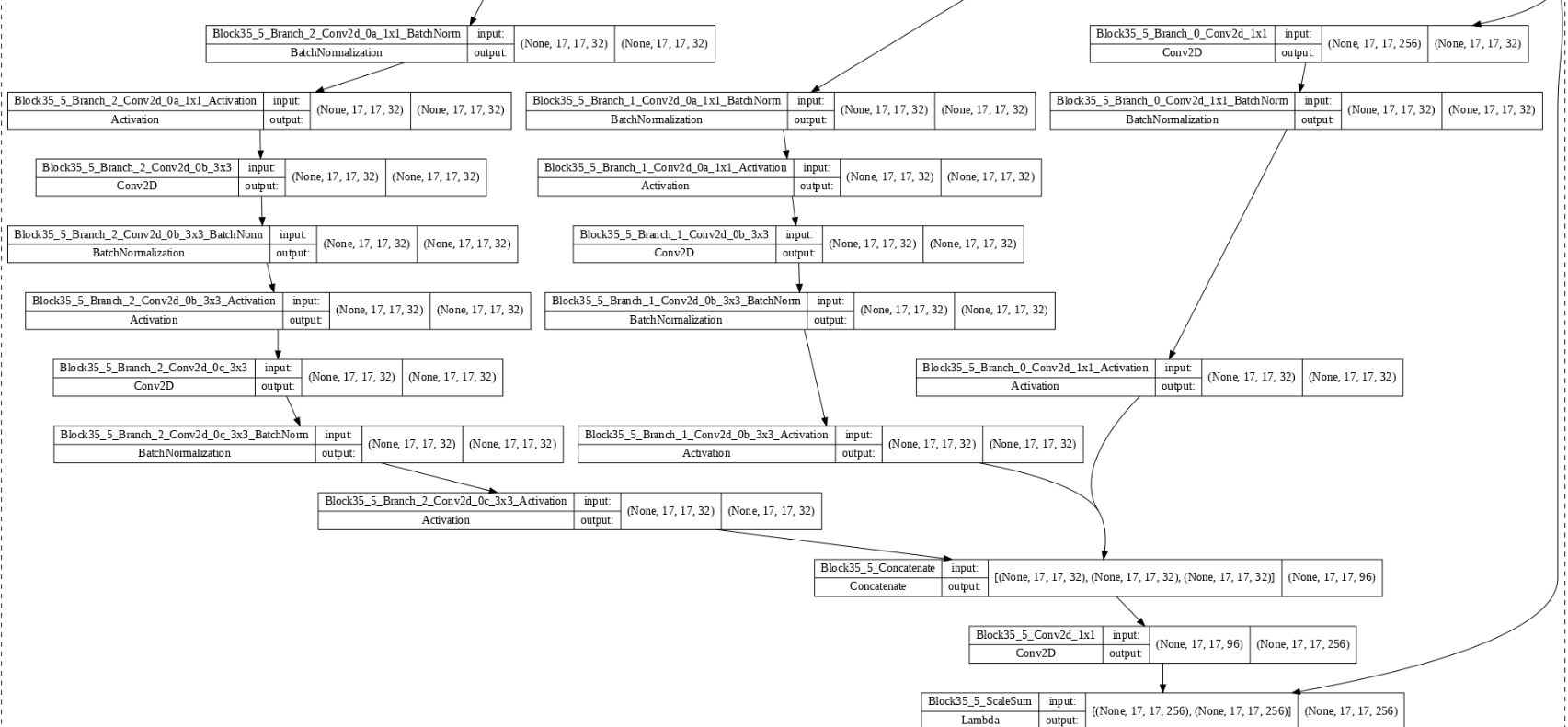
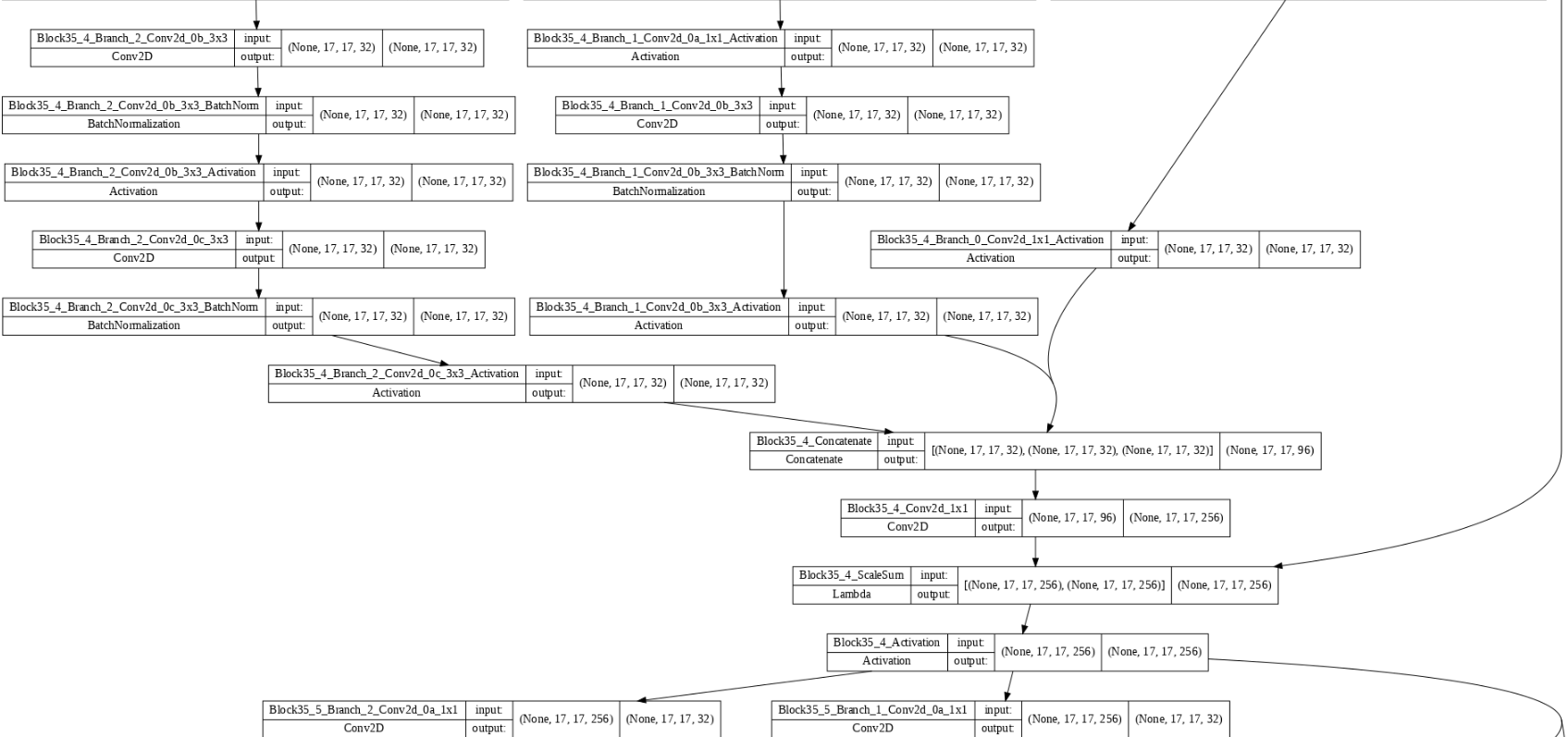
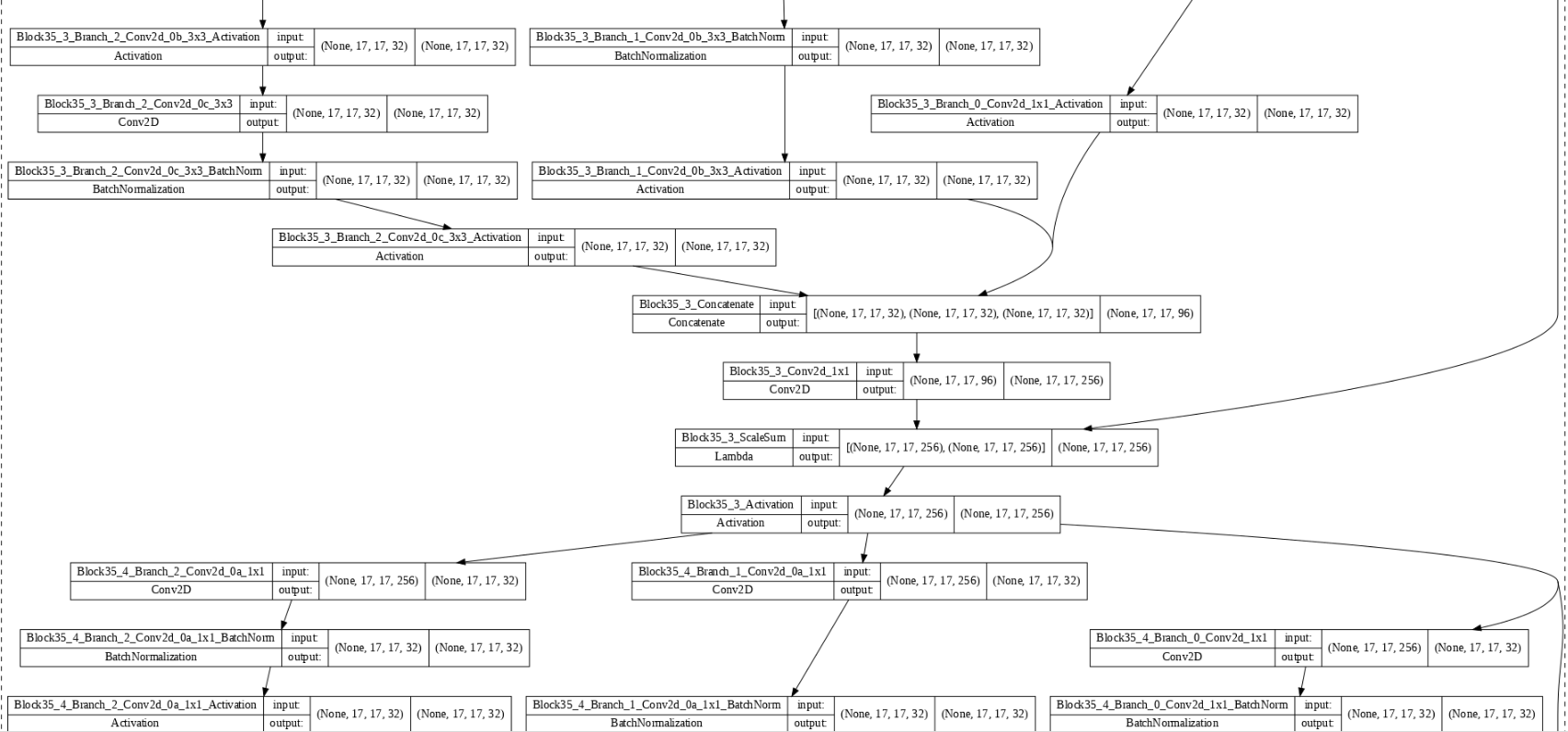
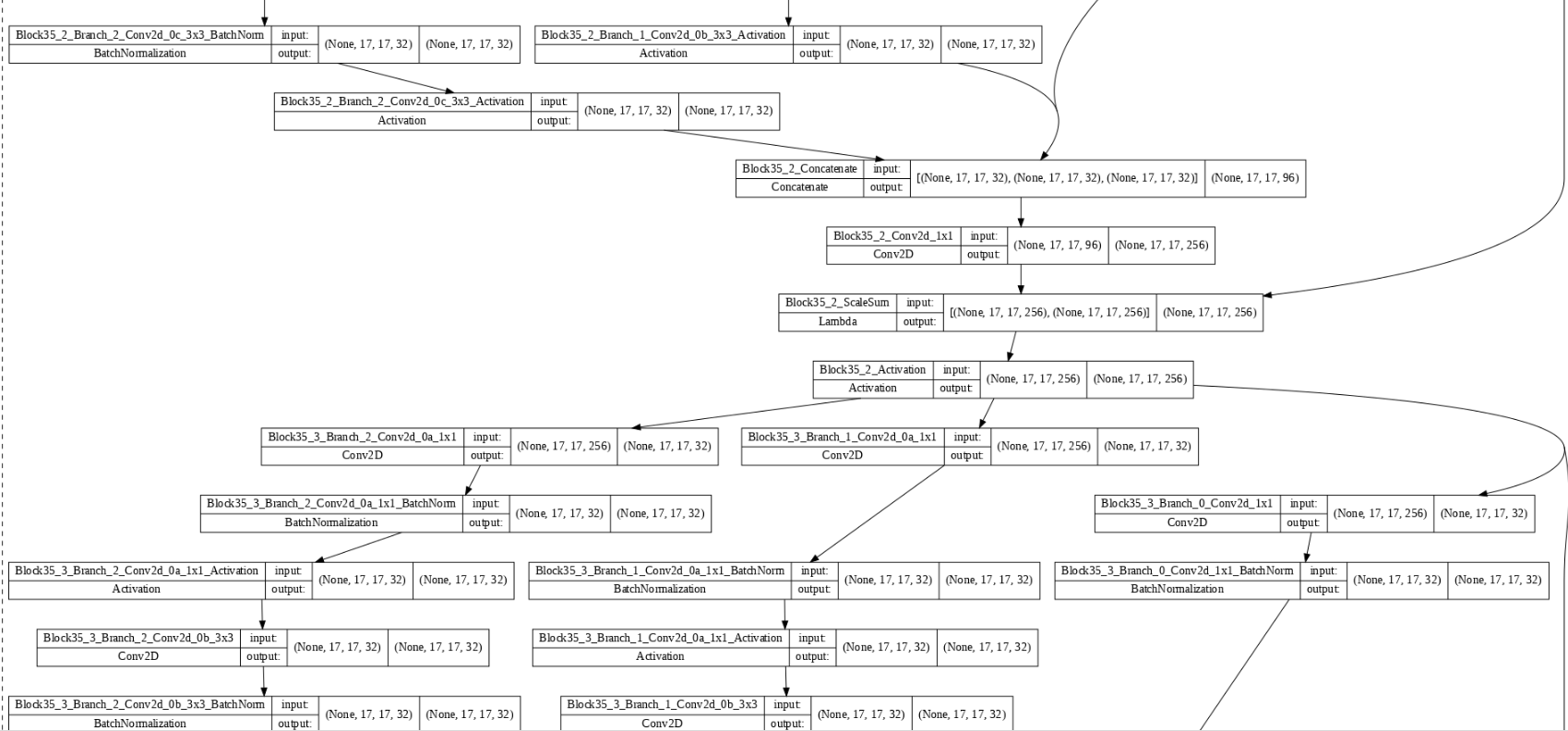
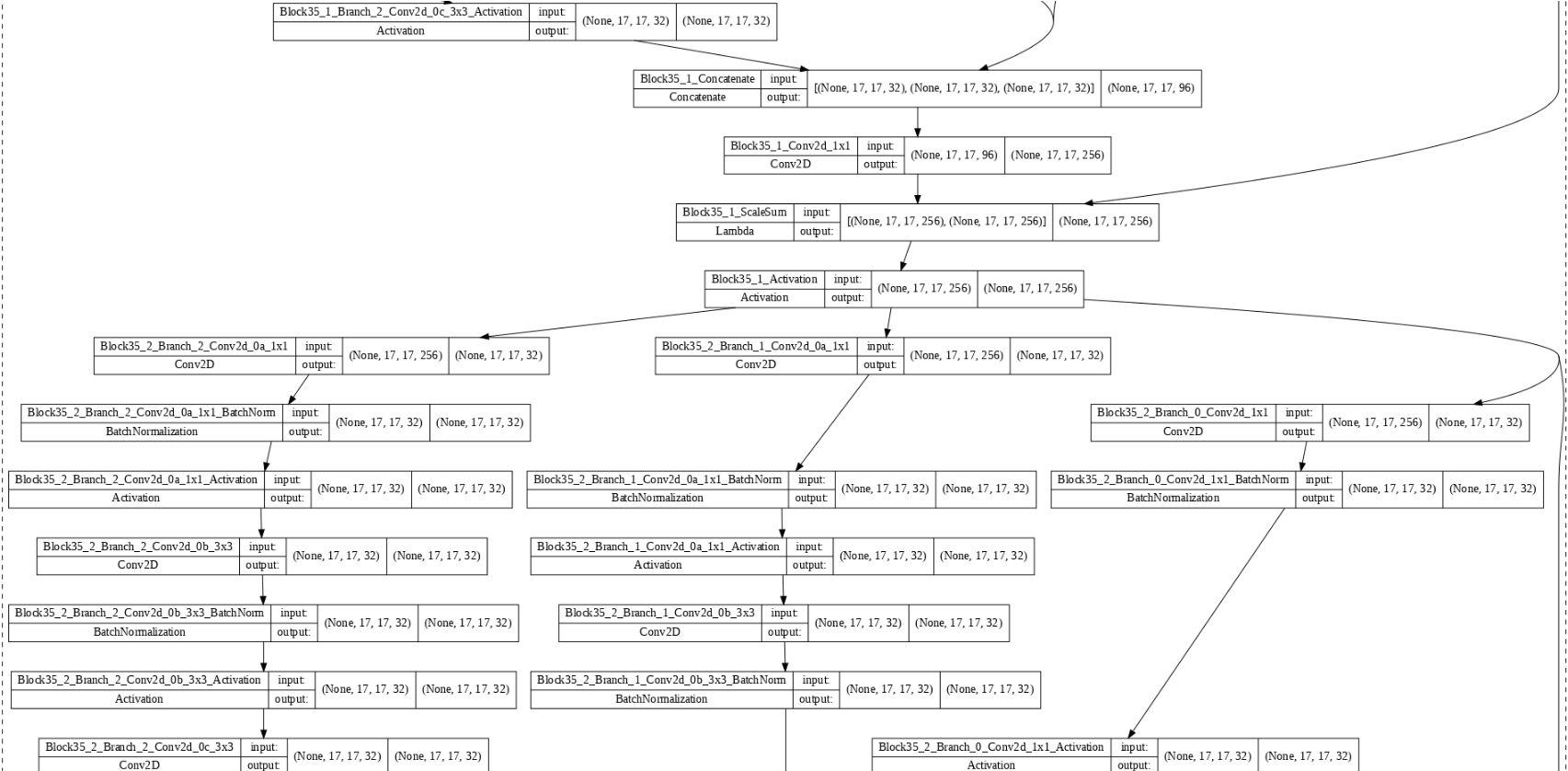
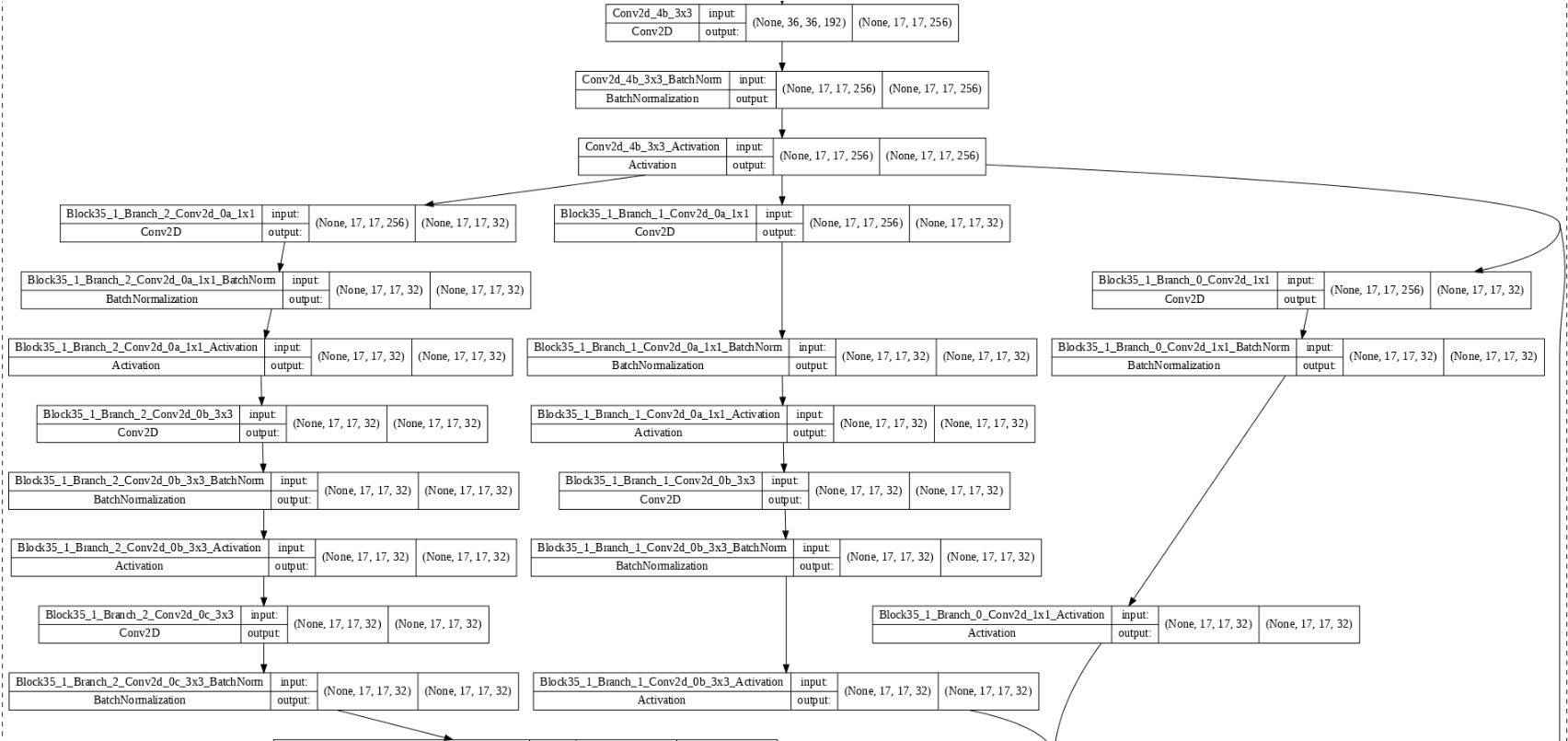
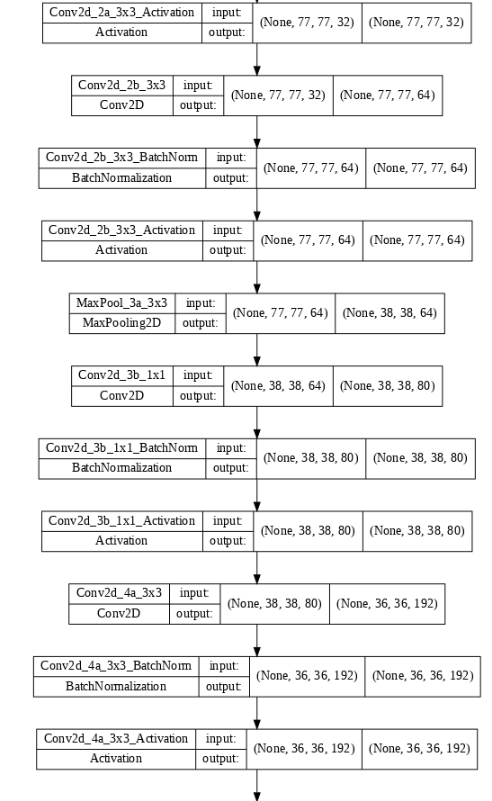
x = Flatten()(x)

outputs = layers.Dense(128, activation = 'sigmoid')(x)

return keras.Model(inputs, outputs)

## המודל של FRmodel





## המודל של ResNet50

