מכון טכנולוגי חולון
Holon Institute of Technology

למידה עמוקה ליישומי ראייה ממוחשבת

מטלה 2

אופיר גוריאל, ת.ז. 322975871
אור כהן, ת.ז. 325046969

מספר קורס:
51283
ימי שישי , 11:00-15:00

סמסטר א תשפ"ב – 2022

סעיף ז

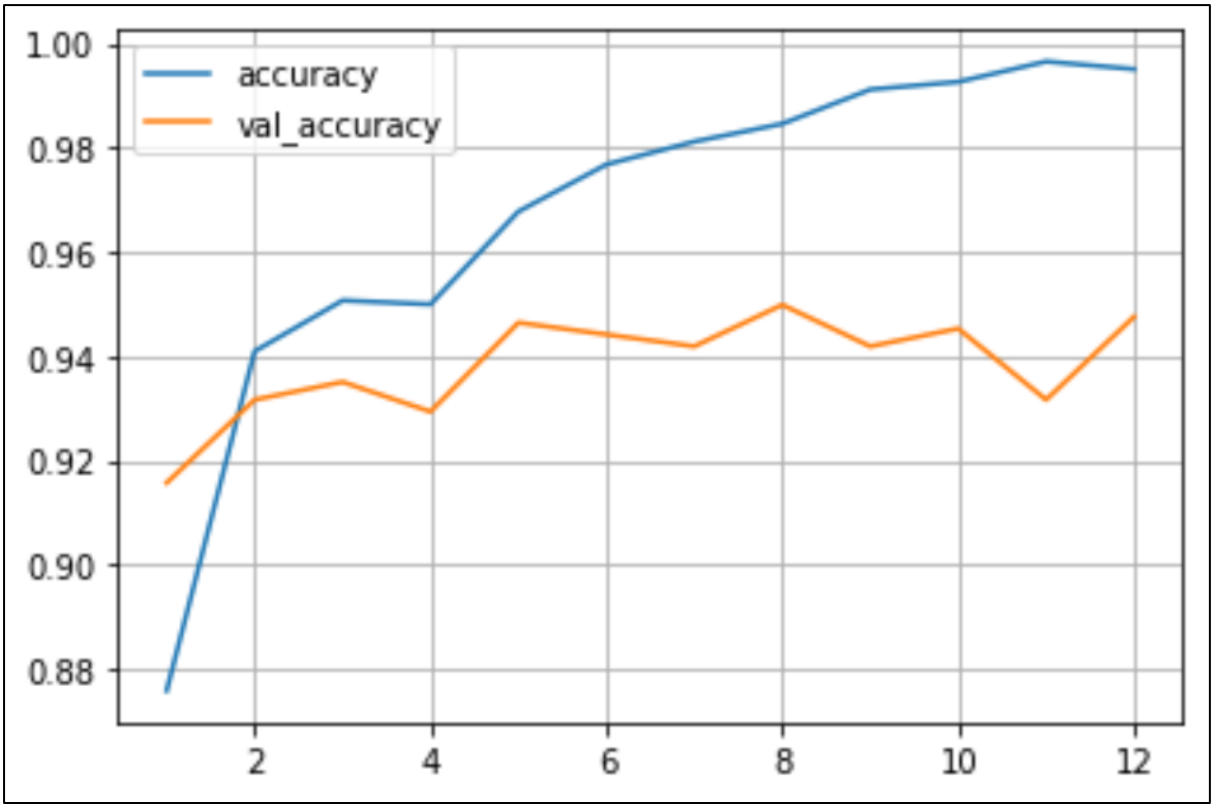
בסעיף זה נשתמש ברשתות מוכנות ומאומנות על מנת לבצע את הקלסיפיקציה של התמונות.

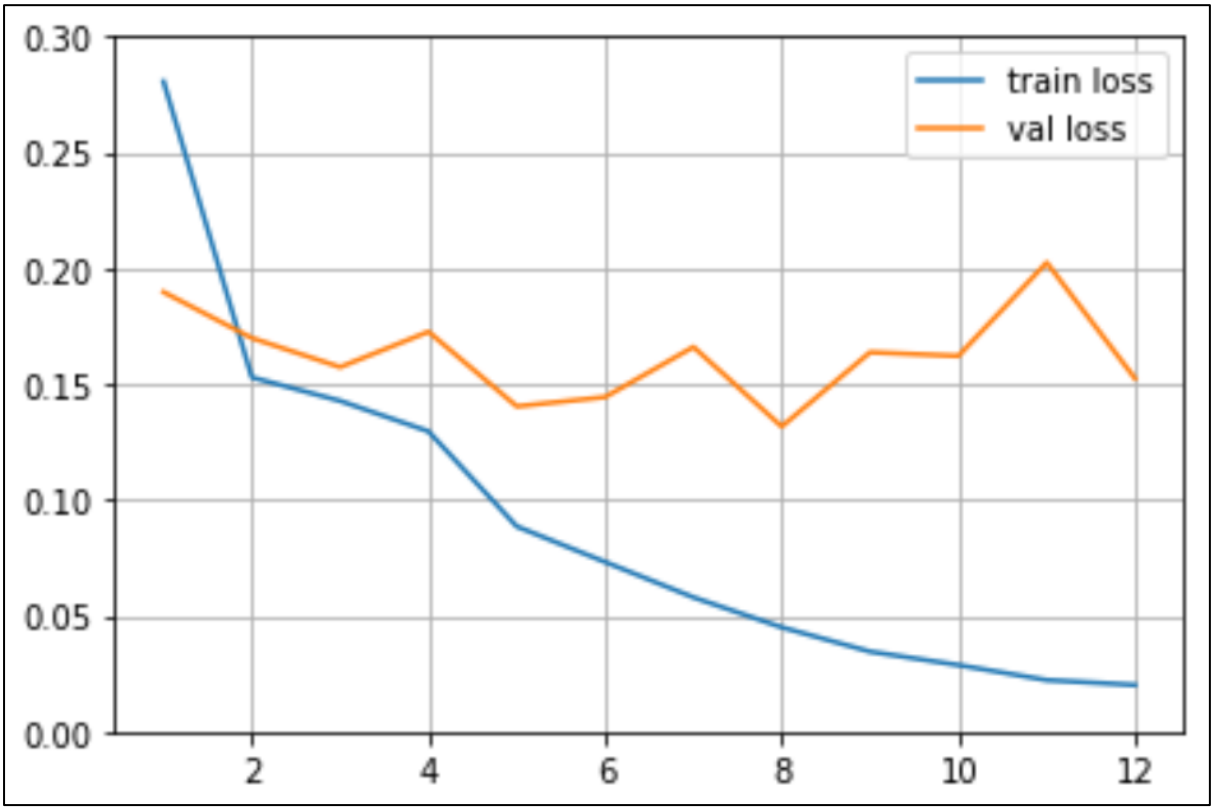
נתחיל מארכיטקטורה בשם: Inception – Version 3. הארכיטקטורה היא מאוד מסובכת, ומכילה למעלה מ 300 שכבות מסוגים שונים: קונבולוציה, Batch Normalization, שרשור, max pooling, אקטיבציה, average pooling, ומספר מקומות של feed forward. במקור, גם רשת זו הייתה מיועדת למשימת image classification, ולכן היא תתאים גם למשימה הבינארית שלנו ככל הנראה. בסופו של מודל זה נוסיף רשת fully connected בעל מספר יחסית קטן של נוירונים: 10 בשכבה הראשונה, 6 בשנייה, ועוד שכבה אחרונה של output. סה"כ נקבל שהאימון יתבצע על 20990 פרמטרים, הרבה מכמות הפרמטרים ברשת DNN רגילה. נשים לב שאת הרשת עצמה נשאיר כמו שהיא ולא נאמן כרגע אף שכבה בתוכה.

להלן סיכום הרשת:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 255, 255, 3)       0

 inception_v3 (Functional)   (None, 6, 6, 2048)        21802784

 dense (Dense)               (None, 6, 6, 10)          20490

 dense_1 (Dense)             (None, 6, 6, 6)           66

 flatten (Flatten)           (None, 216)               0

 dense_2 (Dense)             (None, 2)                 434

=================================================================
Total params: 21,823,774
Trainable params: 20,990
Non-trainable params: 21,802,784
_________________________________________________________________
```

להלן התוצאות שנקבל מתהליך האימון:





והתוצאות שמתקבלות מההרצה על סט המבחן:

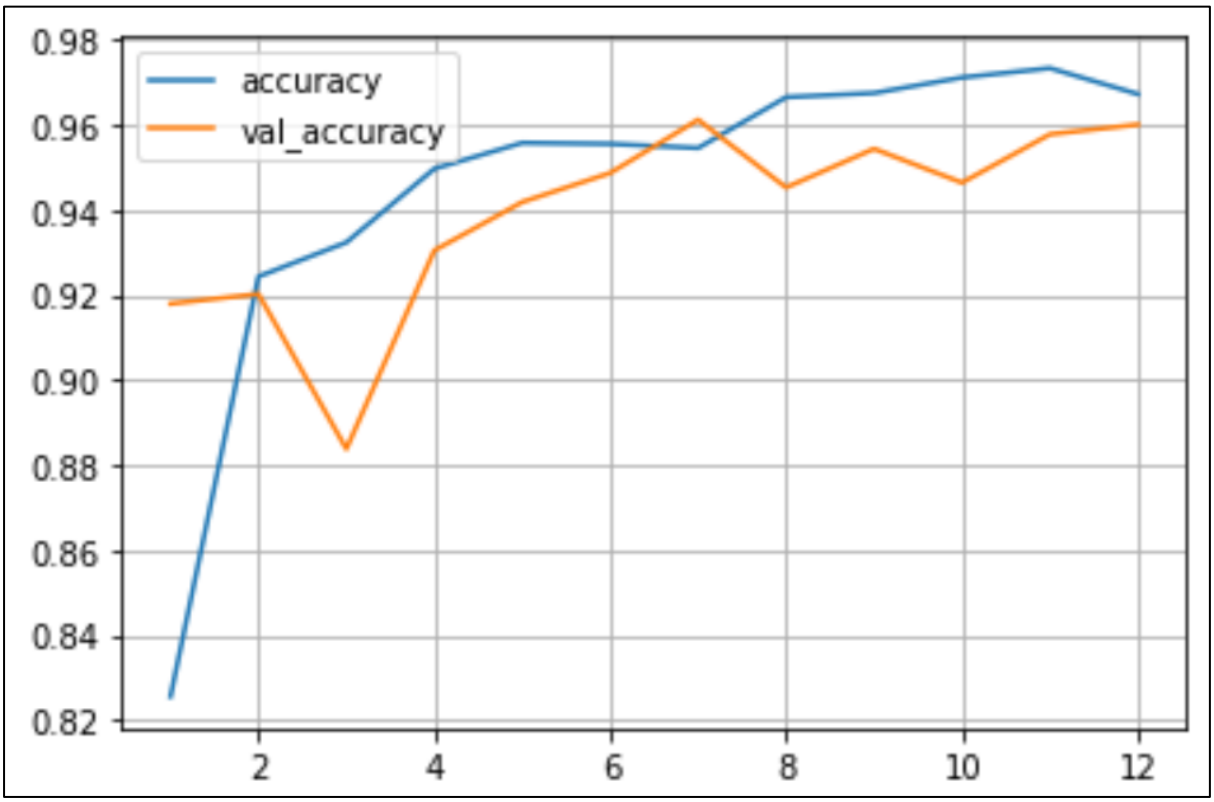
```
loss, accuracy = model.evaluate(TestData)
print("Loss :", loss)
print("Accuracy :", accuracy)
```

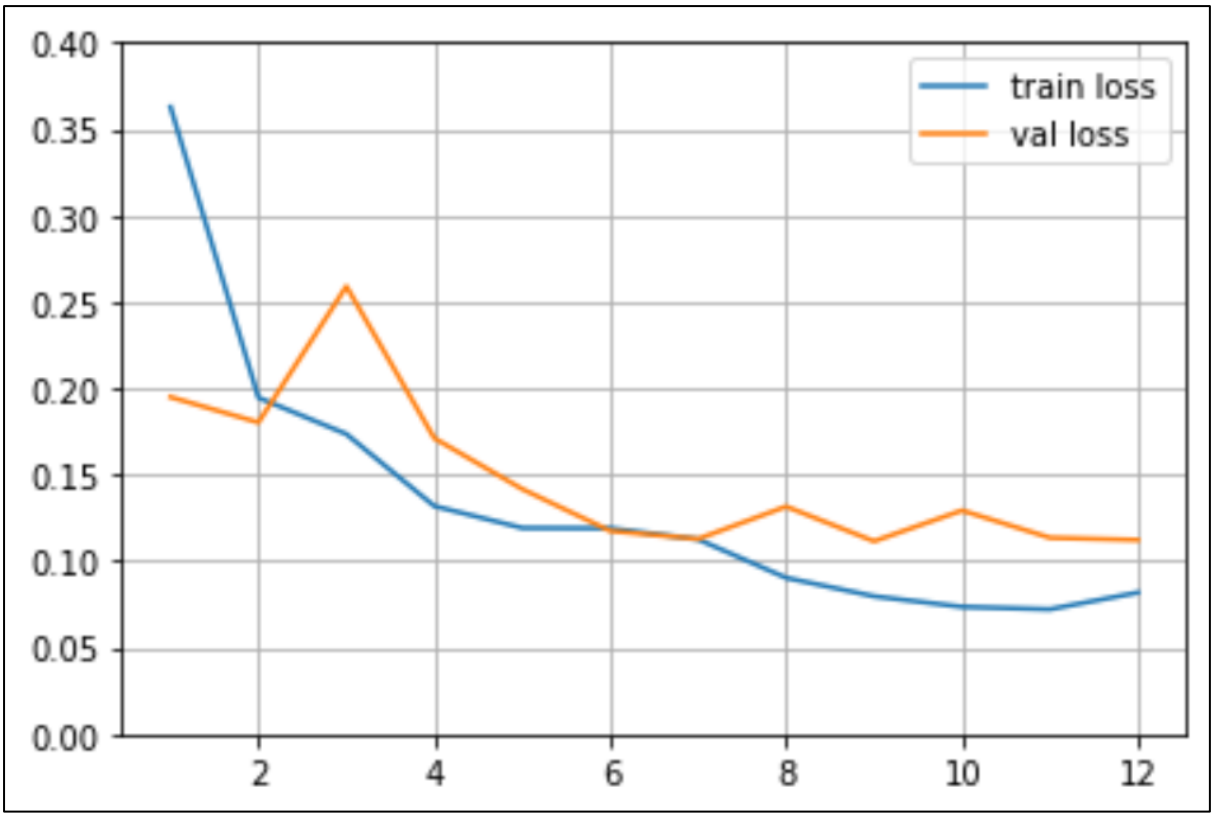
```
1/1 [==============================] - 139s 139s/step - loss: 0.1098 - accuracy: 0.9647
Loss : 0.10978132486343384
Accuracy : 0.9647326469421387
```

```
Precision: 0.9663608562691132
Recall: 0.9859594383775351
confusion matrix: tf.Tensor(
[[632   9]
 [ 22 216]], shape=(2, 2), dtype=int32)
```

נבדוק כעת את ההשפעה של הוספת dropout על הרשת. נוסיף שיעור dropout של 0.2 אחרי כל אחת מהשכבות fully connected שאנחנו הוספנו.

להלן גרפי האימון שנקבל כעת:





והבדיקה על סט המבחן:

```
loss, accuracy = model.evaluate(TestData)
print("Loss :", loss)
print("Accuracy :", accuracy)
```

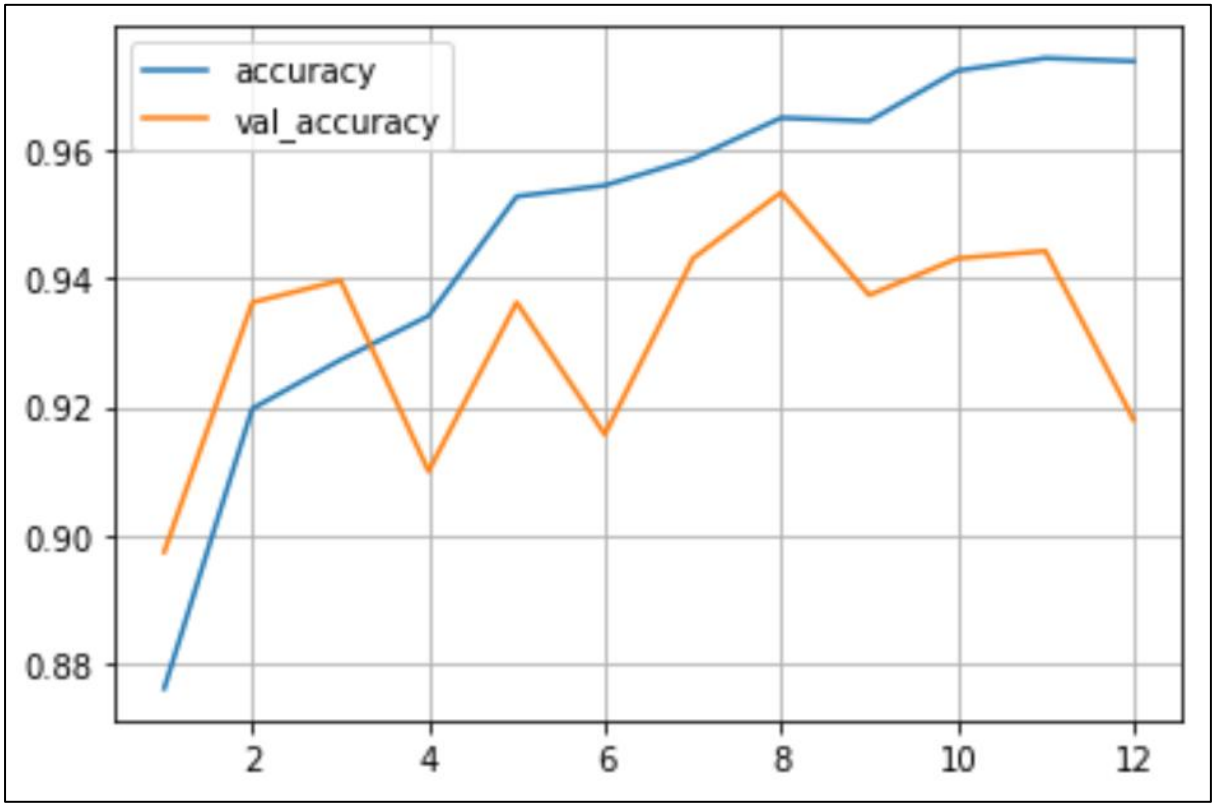
```
1/1 [==============================] - 149s 149s/step - loss: 0.1643 - accuracy: 0.9579
Loss : 0.1643354296684265
Accuracy : 0.9579067230224609
```

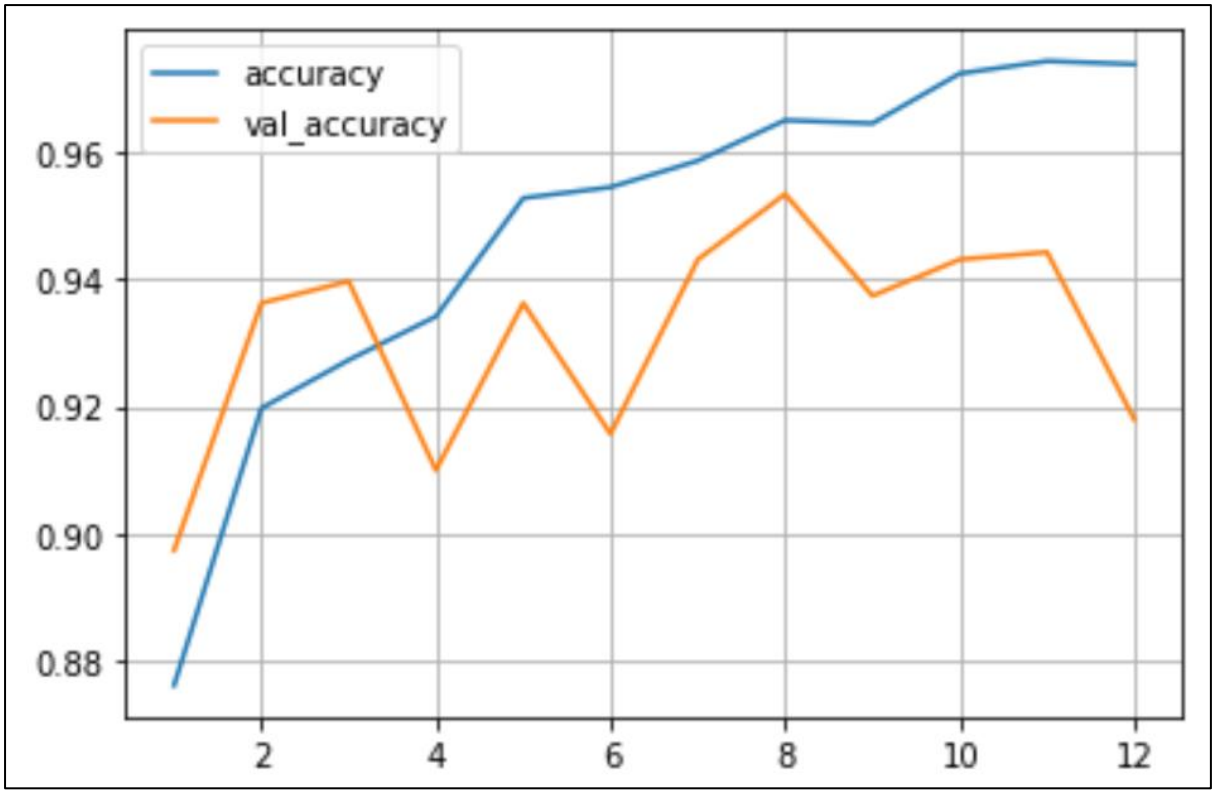
```
Precision: 0.9520958083832335
Recall: 0.9921996879875195
confusion matrix: tf.Tensor(
[[636   5]
 [ 32 206]], shape=(2, 2), dtype=int32)
```

ראשית, ניתן לראות שתהליך האימון יחד עם ה dropout היה יותר רציף, ורמת הדיוק בקבוצת ה validation עלתה עם פחות נדנודים. לעומת זאת, באימון ללא ה dropout קיבלנו דיוק סופי בקבוצת המבחן של 96.5%, לעומת 95.8% כאשר האימון נעשה עם ה dropout. אומנם זהו לא שינוי

משמעותי במיוחד, אך כן ניתן להסיק ממנו שיש השפעה לשכבות אלו. עדות נוספת להשפעה היא השינוי במטריצת המבוכה, שבה קיבלנו ערכים שונים בשני המקרים.

נבדוק כעת את ההשפעה של הוספת אוגמנטציות, להלן מה שנקבל:





```
loss, accuracy = model.evaluate(TestData)
print("Loss :", loss)
print("Accuracy :", accuracy)
```

```
1/1 [==============================] - 142s 142s/step - loss: 0.3219 - accuracy: 0.9158
Loss : 0.3218936622142792
Accuracy : 0.9158134460449219
```

```
Precision: 0.8987341772151899
Recall: 0.9968798751950078
confusion matrix: tf.Tensor(
[[639   2]
 [ 72 166]], shape=(2, 2), dtype=int32)
```

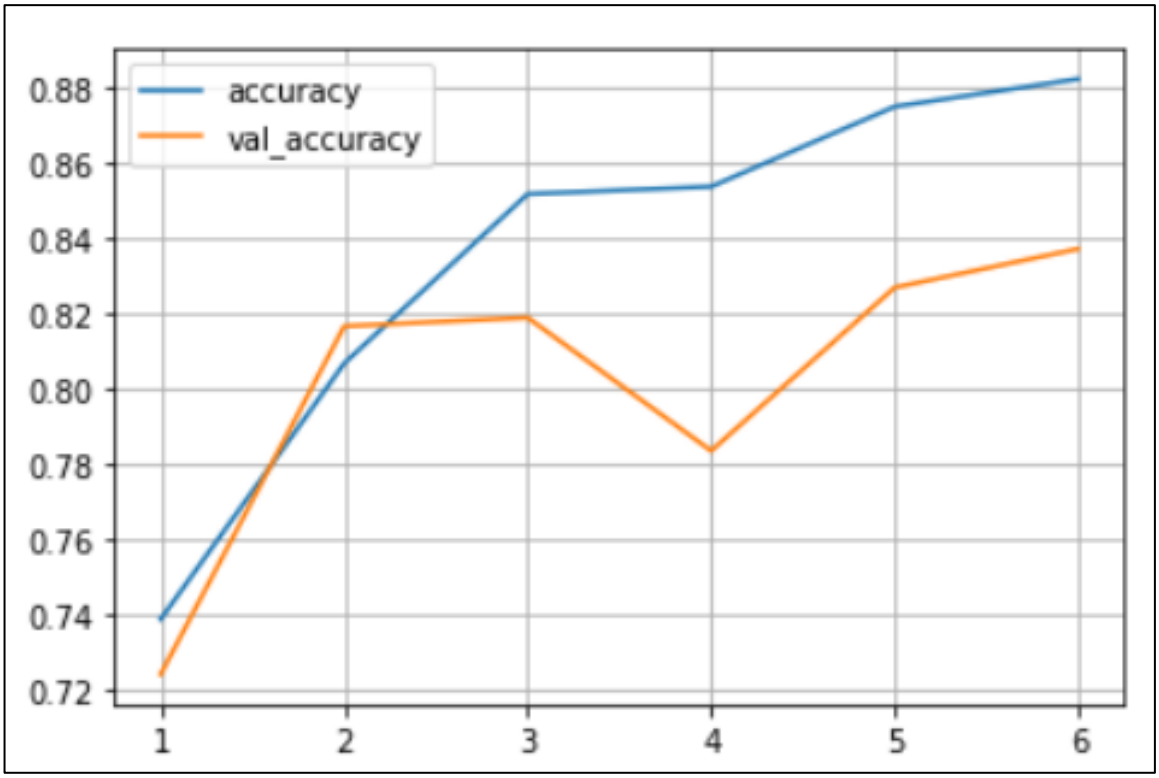
ניתן לראות שהאוגמנטציות גרמו להרעה בביצועי המערכת, בכך שהדיוק על קבוצת המבחן ירדה מ 95.8% ל 91.6%. עם זאת, ה recall הגבוה שהיה נשמר, והירידה היא דווקא במדד ה precision.

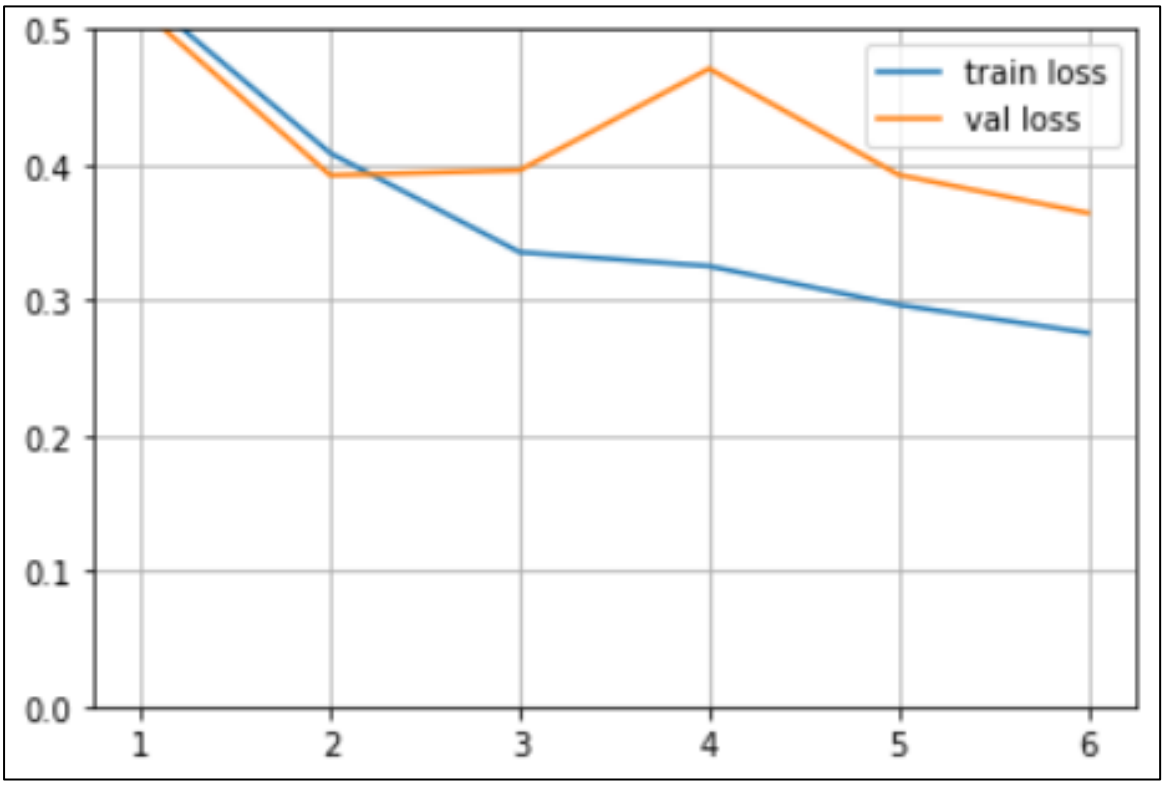
נמשיך כעת לארכיטקטורה בשם Resnet50. הפעם נקבל שהאופטימיזציה מתבצעת רק על 17079 פרמטרים.

המודל שבנינו:

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_5 (Rescaling)     (None, 255, 255, 3)       0

 resnet50 (Functional)       (None, 8, 8, 2048)        23587712

 dense_15 (Dense)            (None, 8, 8, 8)           16392

 dense_16 (Dense)            (None, 8, 8, 5)           45

 flatten_5 (Flatten)         (None, 320)               0

 dense_17 (Dense)            (None, 2)                 642

=================================================================
Total params: 23,604,791
Trainable params: 17,079
Non-trainable params: 23,587,712
_________________________________________________________________
```

תוצאות האימון שנקבל:

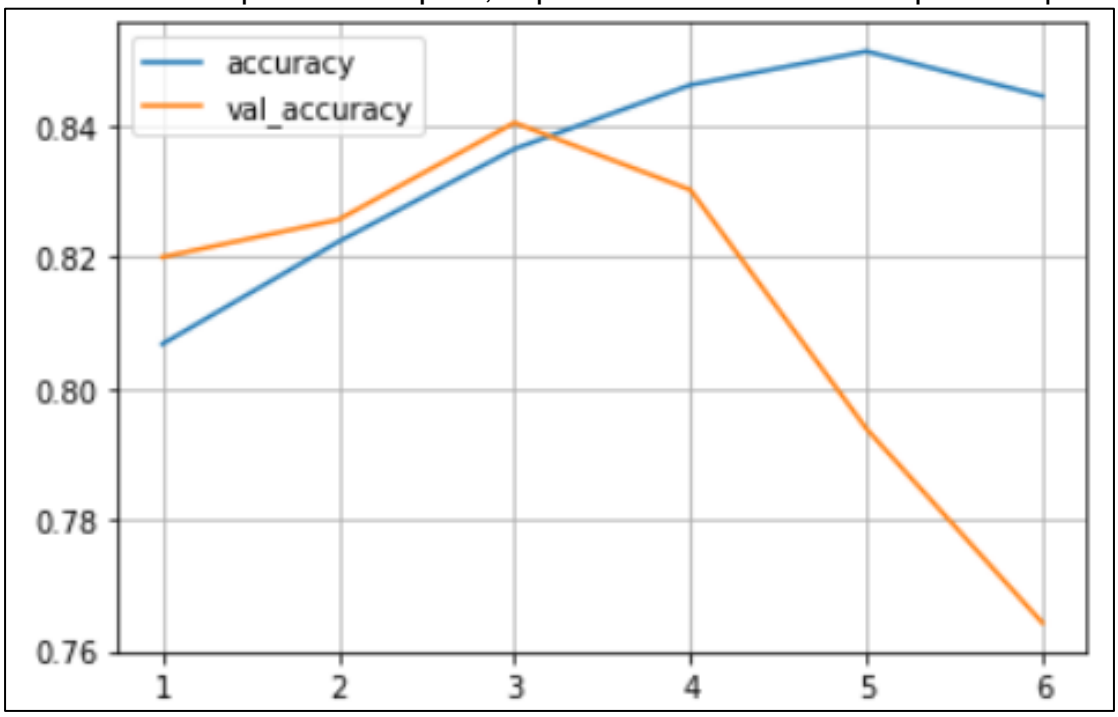


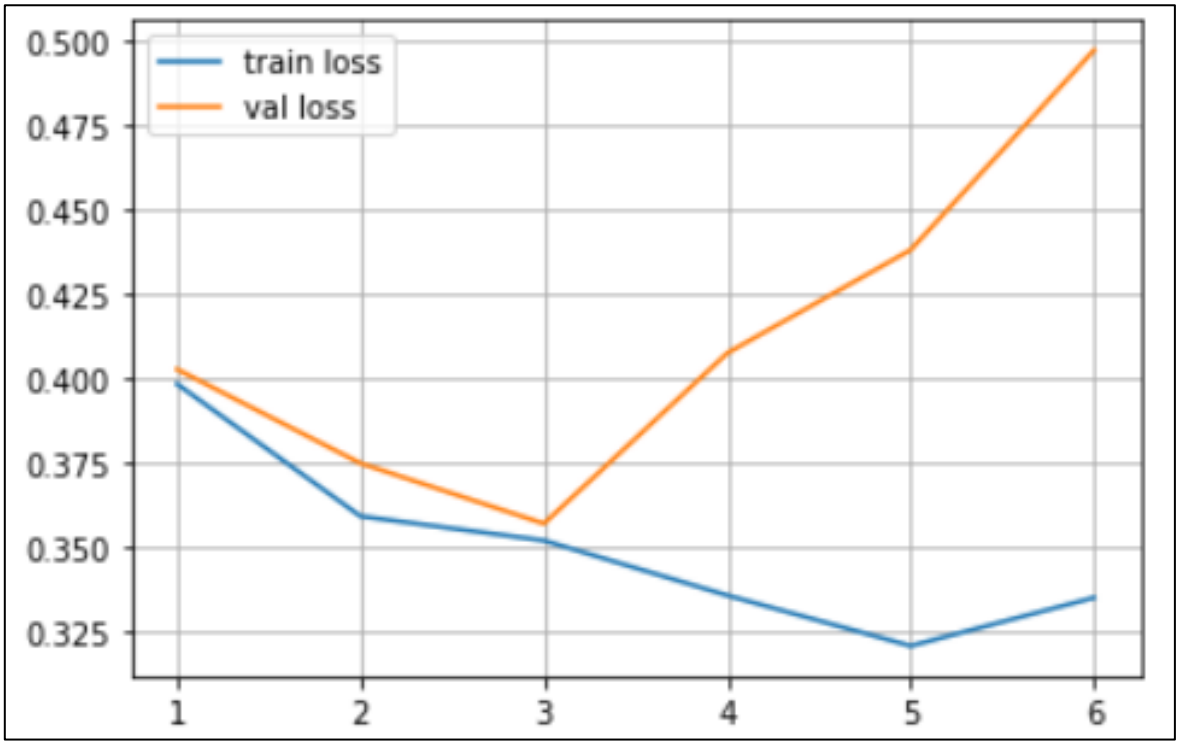


והתוצאות שמתקבלות מההרצה על סט המבחן:

```
1/1 [==============================] - 177s 177s/step - loss: 0.2227 - accuracy: 0.9295
Loss : 0.2226855307817459
Accuracy : 0.9294652938842773
```

ניתן לראות שהתוצאות שהתקבלו בשימוש ב resnet פחות טובות מן התוצאות שהתקבלו מהרצה על inception. נוסיף כעת אוגמנטציה של סיבוב אקראי, להלן התוצאות שנקבל:





והתוצאות על סט המבחן:

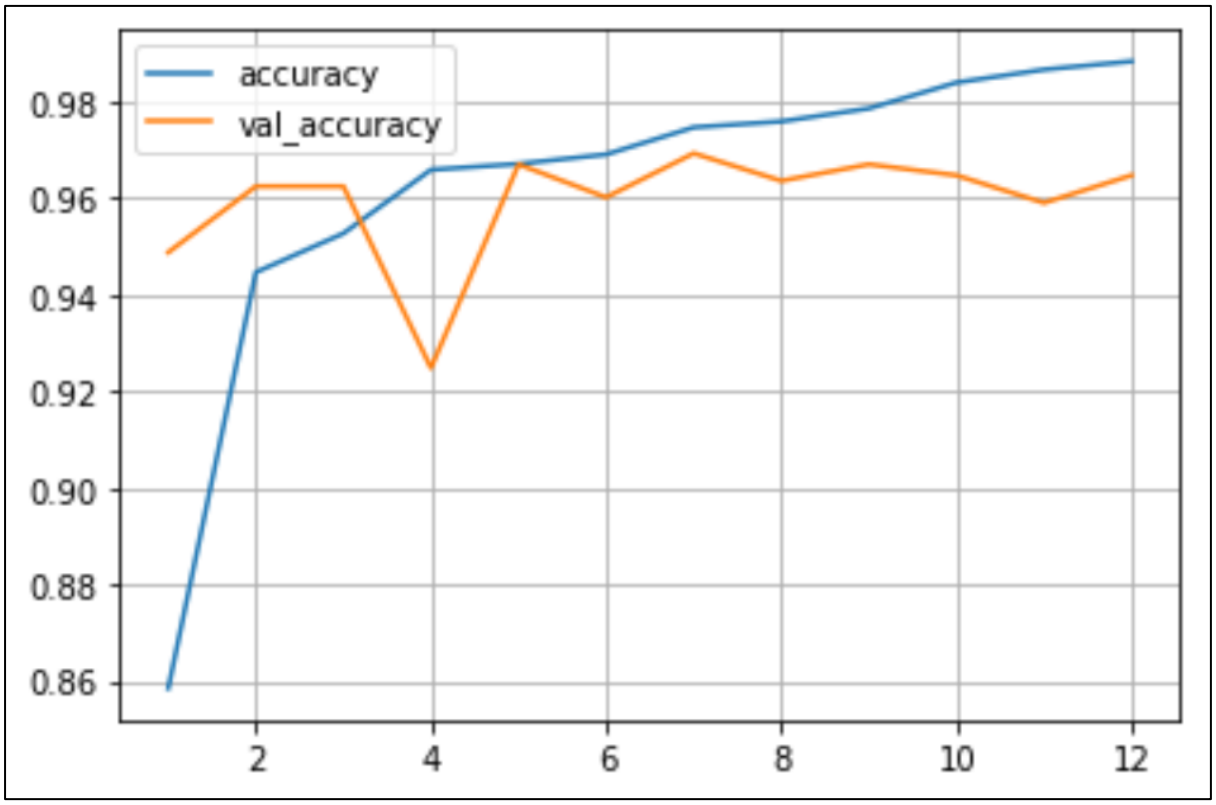
```
1/1 [==============================] - 174s 174s/step - loss: 0.3068 - accuracy: 0.9317
Loss : 0.30677762627601624
Accuracy : 0.9317406415939331
```

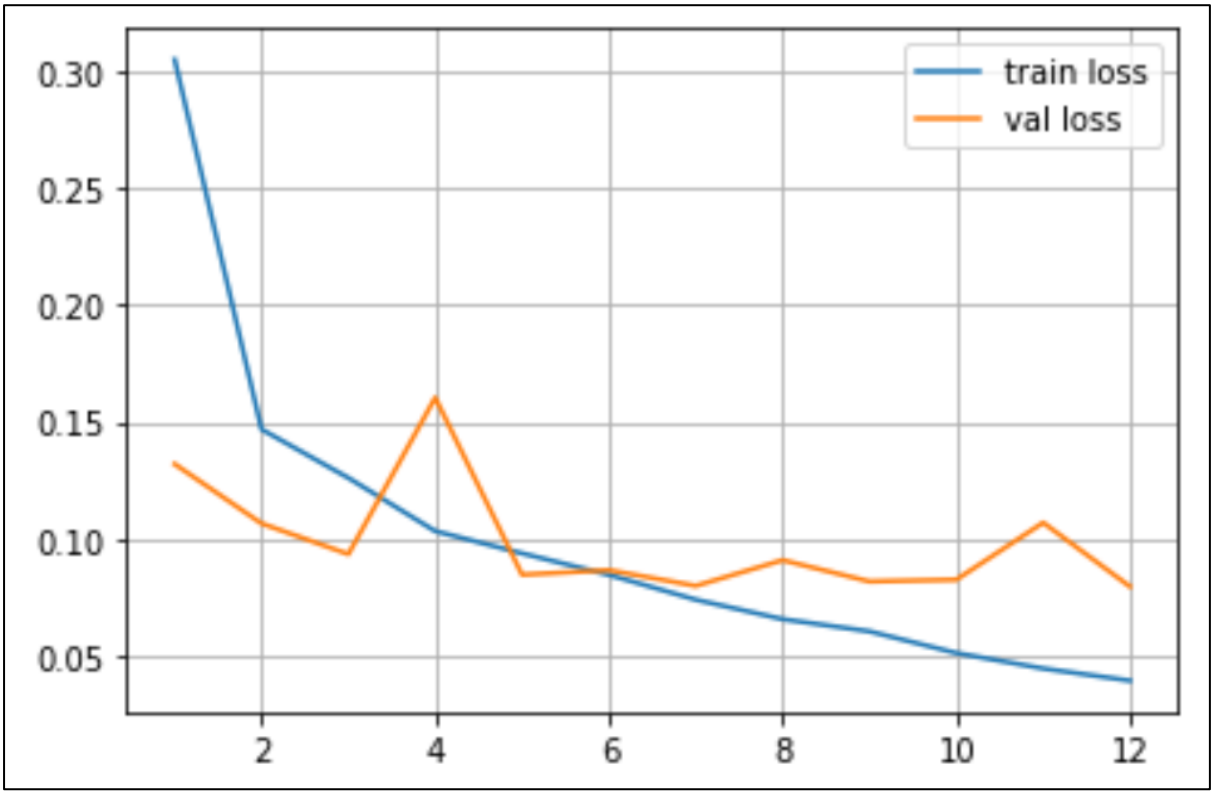
```
Precision: 0.9677938808373591
Recall: 0.9375975039001561
confusion matrix: tf.Tensor(
[[601  40]
 [ 20 218]], shape=(2, 2), dtype=int32)
```

ניתן לראות שהסיבוב האקראי הפריע לרשת ללמוד, ובאפוקים האחרונים שלה קיבלנו עלייה בפונקציית המחיר. למעשה, כל התוצאות שקיבלנו ב resnet היו פחות טובות מאלו שקיבלנו כאשר השתמשנו ב Inception.

נמשיך כעת לארכיטקטורה בשם DenseNet121. כמו Inception, גם היא מכילה כמות גדולה במיוחד של שכבות מסוגים שונים, קונבולוציה, שרשור, Batch Normalization, וכו'. הפעם, נוסיף בסוף הארכיטקטורה 2 שכבות fully connected אך עם פחות נוירונים. סה"כ נקבל שהאופטימיזציה תתבצע רק על 8887 פרמטרים (פחות ממחצית מכמות הפרמטרים שמאמנים).

להלן התוצאות שנקבל מתהליך האימון:





והתוצאות שנקבל מסט המבחן:

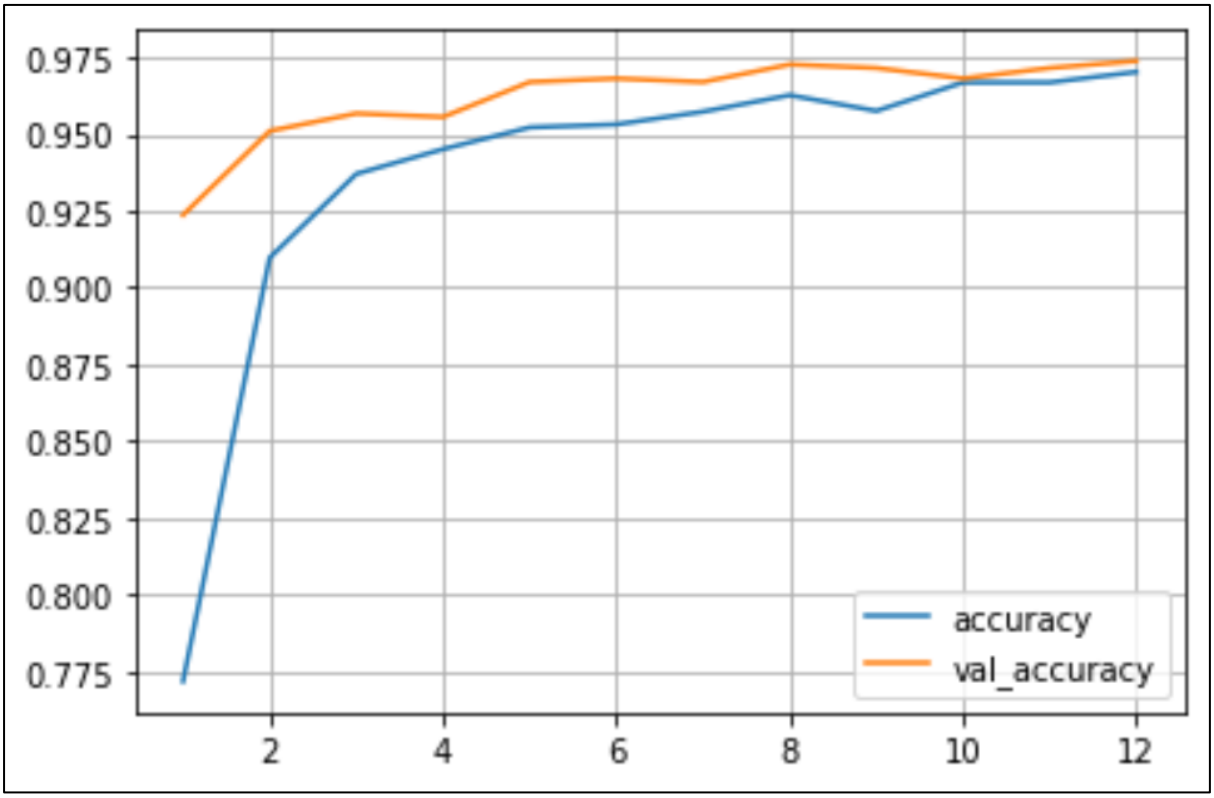
```
loss, accuracy = model.evaluate(TestData)
print("Loss :", loss)
print("Accuracy :", accuracy)
```

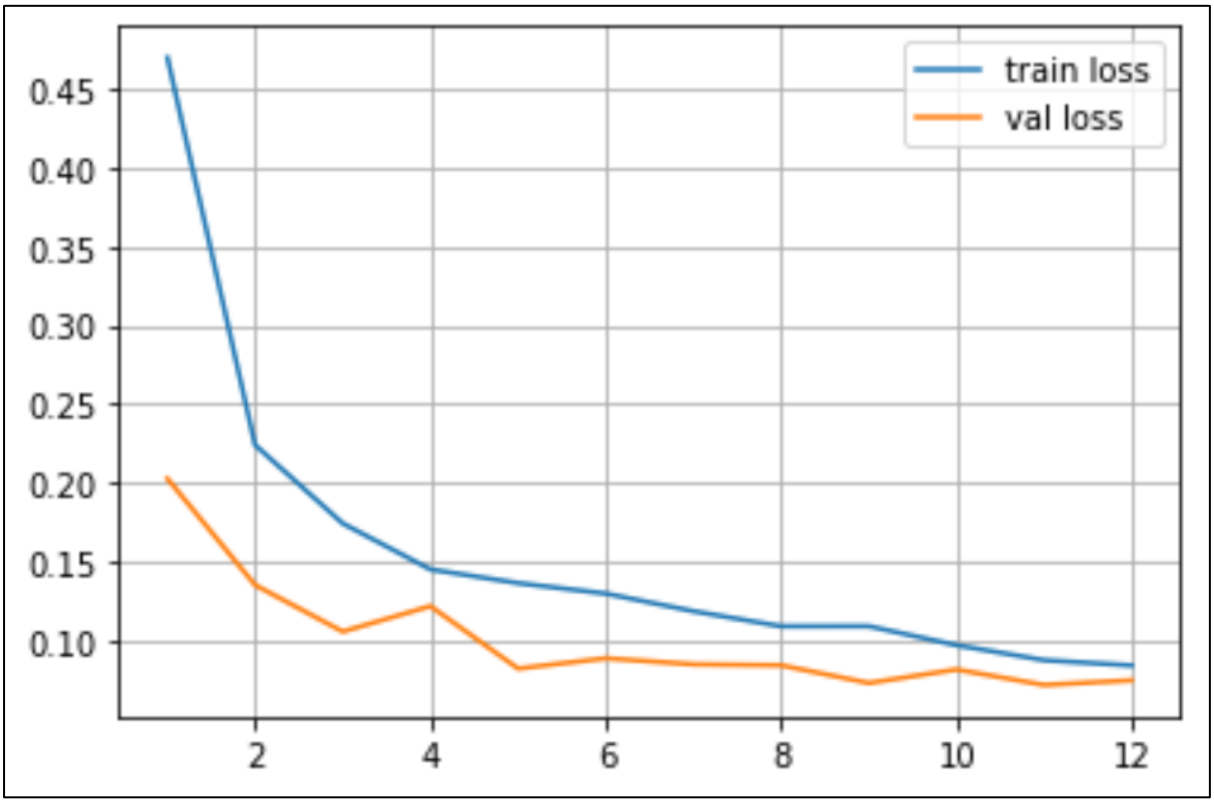
```
1/1 [==============================] - 186s 186s/step - loss: 0.1067 - accuracy: 0.9704
Loss : 0.10669291019439697
Accuracy : 0.9704209566116333
```

```
Precision: 0.9680365296803652
Recall: 0.9921996879875195
confusion matrix: tf.Tensor(
[[636   5]
 [ 21 217]], shape=(2, 2), dtype=int32)
```

כעת נבחן הוספת שכבות dropout אחרי כל אחת מהשכבות fully connected בשיעור של 0.3.

להלן התוצאות שנקבל בתהליך האימון:





והתוצאות שנקבל מסט המבחן:

```
loss, accuracy = model.evaluate(TestData)
print("Loss :", loss)
print("Accuracy :", accuracy)
```

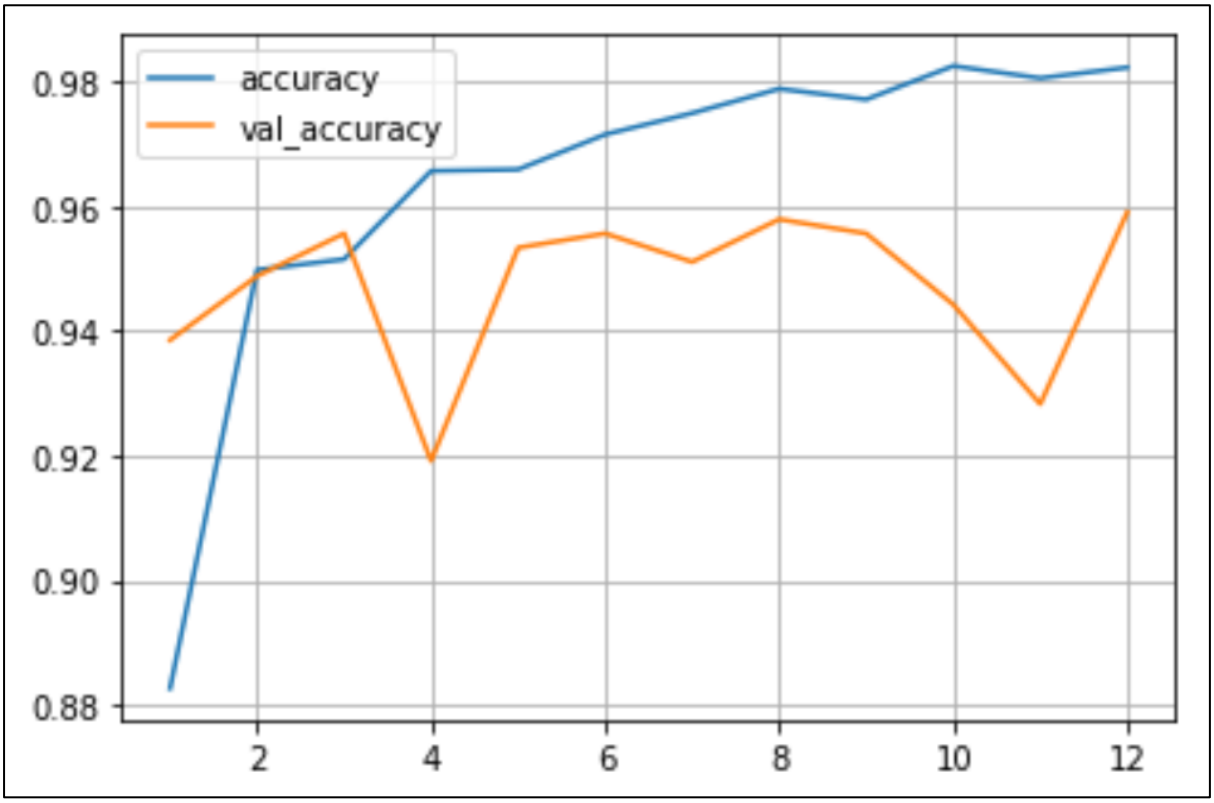
```
1/1 [==============================] - 201s 201s/step - loss: 0.0970 - accuracy: 0.9681
Loss : 0.09695755690336227
Accuracy : 0.9681456089019775
```

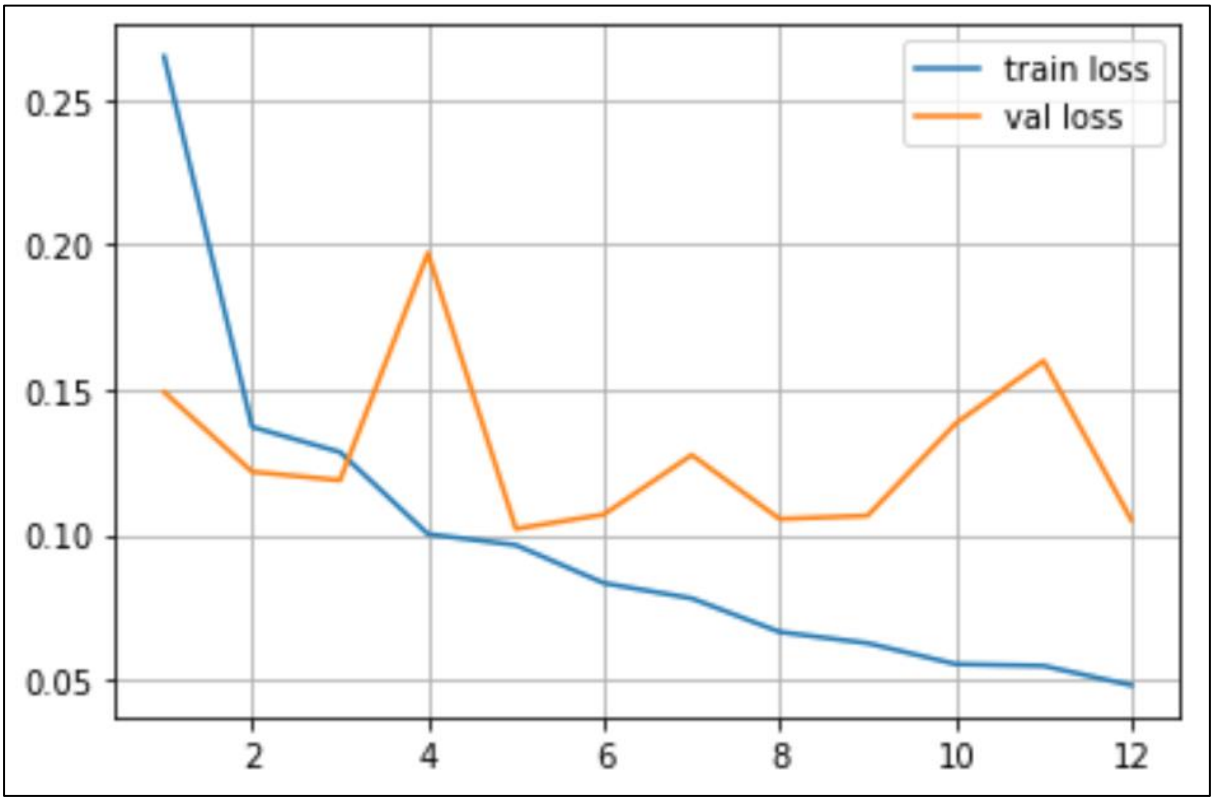
```
Precision: 0.9622926093514329
Recall: 0.9953198127925117
confusion matrix: tf.Tensor(
[[638   3]
 [ 25 213]], shape=(2, 2), dtype=int32)
```

ראשית, גם פה ניתן לראות שהוספת שכבות ה dropout עזרה ללמידה להיות יותר רציפה ועם פחות נדנוד, שכן שכבות אלו דואגות לכך שההחלטה והאימון לא תהיה סביב משקולים בודדים אלא בעזרת כל הפרמטרים.

שנית, ניתן לראות שהגענו לביצועים טובים במיוחד בשני הניסויים, של יותר מ 96.7% דיוק בקבוצת המבחן. זהו שיפור ביחס לשימוש ב Inception, במיוחד תוך התחשבות שבמקרה זה השתמשנו ברשת עם הרבה פחות נוירונים. נשים לב שבכל אחד מהניסויים קיבלנו שיעור Recall גבוה במיוחד, מה שמעיד על כך שבזמן אמת בהחלט ניתן להשתמש ברשתות אלו, שכן תהיה הסתברות נמוכה מאוד לקבל תוצאה שלילית שגויה.

נבדוק כעת את ההשפעה של הוספת אוגמנטציות, להלן מה שנקבל:





והתוצאות שמתקבלות מסט המבחן:

```
loss, accuracy = model.evaluate(TestData)
print("Loss :", loss)
print("Accuracy :", accuracy)
```

```
1/1 [==============================] - 191s 191s/step - loss: 0.0968 - accuracy: 0.9659
Loss : 0.0968031957745552
Accuracy : 0.9658703207969666
```

```
Precision: 0.9649923896499238
Recall: 0.9890795631825273
confusion matrix: tf.Tensor(
[[634   7]
 [ 23 215]], shape=(2, 2), dtype=int32)
```

ניתן לראות שהפעם קיבלנו שביצועי המערכת כמעט ולא השתנו (קיבלנו ירידה קטנה של פחות מאחוז ברמת הדיוק – ולכן ניתן להזניח אותה). נשים לב שזה עומד בניגוד למה שראינו בחלק הראשון של העבודה, שם ראינו הרעה בביצועי הרשת בעקבות הוספת האוגמנטציות. מגמה זו ראינו בניסיון עם הרשת Inception, ולכן יכול להיות שהשוני בין ההשפעות שנובעות מהאוגמנטציות מקורו בשוני בארכיטקטורות.

סעיף ח

בסעיף זה נבחן את המשמעות של אימון שכבות מסוימות ברשת, תהליך אשר נקרא fine tuning, כלומר הסבה של ארכיטקטורה מסוימת לצרכים שלנו על ידי אימון מספר מסוים של שכבות בלבד. לצורך הניסוי נשתמש ברשת Inception V3, וליתר ביטחון נוסיף בינה ובין שכבת ה output שכבה אחת של fully connected עם 10 נוירונים ושכבת dropout בשיעור של 0.2. באופן תיאורטי, התוצאות אליהן אנחנו מצפים הם שאימון השכבות האחרונות בארכיטקטורה יועיל הרבה יותר מאשר אימון השכבות הראשונות, שכן שכבות הקונבולוציה האחרונות הן אלו המזהות את הפרטים הקטנים בתמונות.

בשביל סעיף זה נתעמק יותר בסוגי השכבות שנמצאות בארכיטקטורה. נמצא את שכבות הקונבולוציה ואת המיקומים שלהן, ולפי זה נבחר לאמן. בניסוי הראשון נקפיא את כל השכבות ונאמן רק את שכבת הקונבולוציה השלישית מההתחלה (לא רצינו לקחת את הראשונה על מנת להימנע מתופעות מעבר למיניהן שיכולות לקרות בתחילת הרשת). ביחד עם שכבת הקונבולוציה, נגדיר את פרמטר ה trainable ל true גם לשכבות ה Batch Normalization, Activation ו Max Pooling שאחריה ולפני שכבת הקונבולוציה הבאה.

```
for i, layer in enumerate(BasicModel.layers):
  layer.trainable = 7 <= i <= 11
```

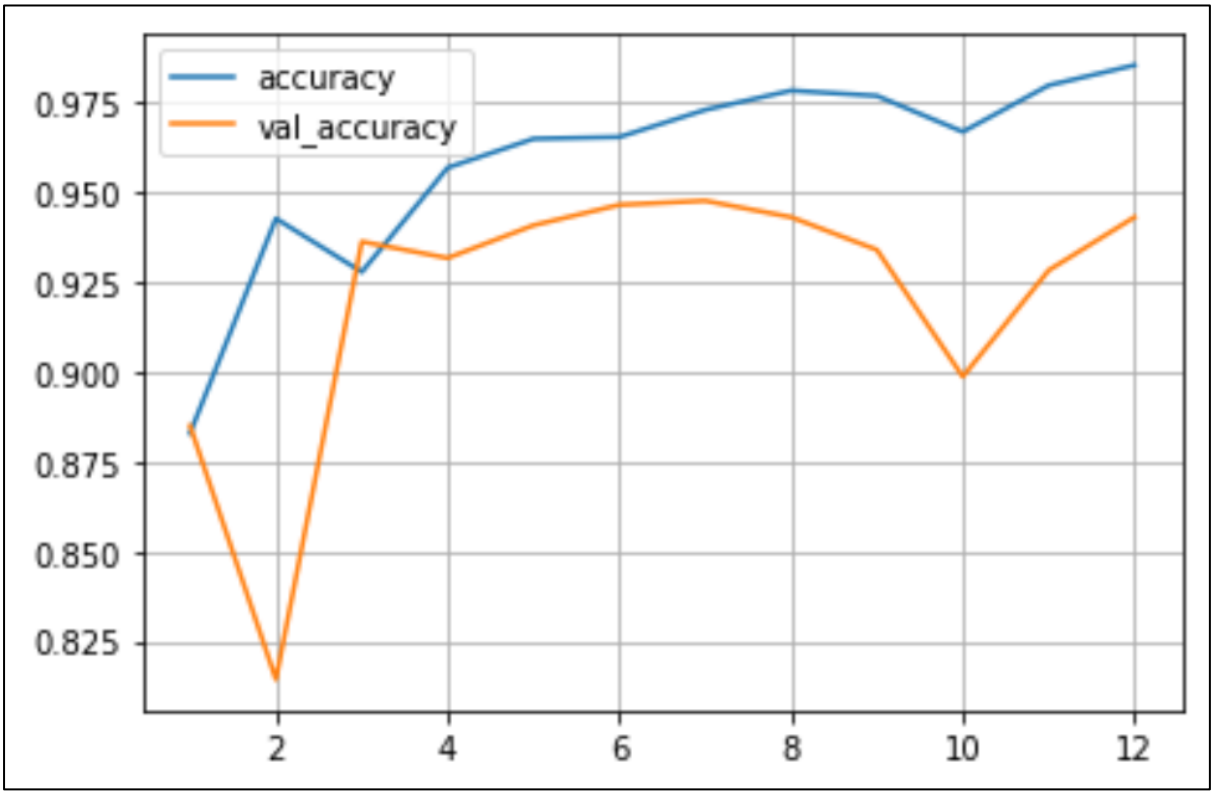
```
BasicModel.layers[7:11]
```

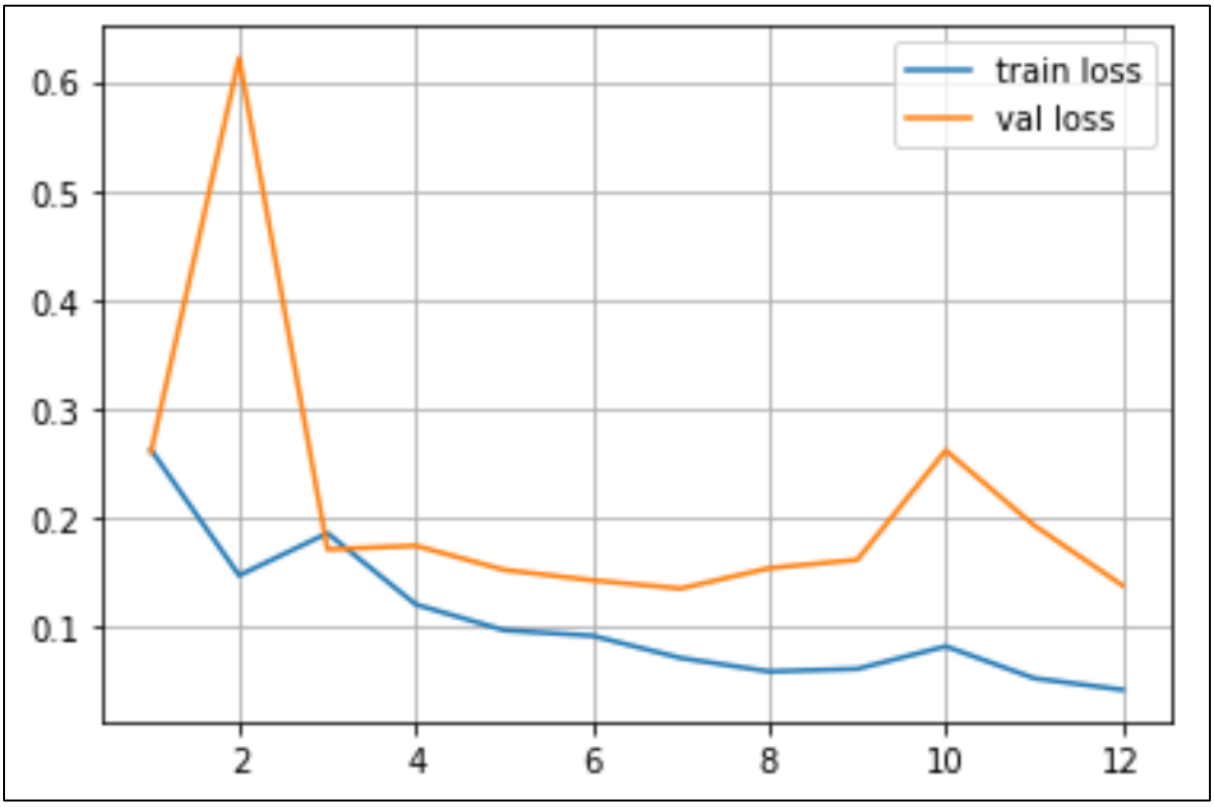
```
[<keras.layers.convolutional.Conv2D at 0x7fa3c810b710>,
 <keras.layers.normalization.batch_normalization.BatchNormalization at 0x7fa3c8119290>,
 <keras.layers.core.activation.Activation at 0x7fa3c8120090>,
 <keras.layers.pooling.MaxPooling2D at 0x7fa3c8120490>]
```

נקבל אם כן את סיכום המודל:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 255, 255, 3)       0

 inception_v3 (Functional)   (None, 6, 6, 2048)        21802784

 dense (Dense)               (None, 6, 6, 10)          20490

 dropout (Dropout)           (None, 6, 6, 10)          0

 flatten (Flatten)           (None, 360)               0

 dense_1 (Dense)             (None, 2)                 722

=================================================================
Total params: 21,823,996
Trainable params: 44,828
Non-trainable params: 21,779,168
_________________________________________________________________
```

קיבלנו סה"כ 44828 פרמטרים שאותם מאמנים. להלן התוצאות שנקבל:





ומסט המבחן:

```
loss, accuracy = model.evaluate(TestData)
print("Loss :", loss)
print("Accuracy :", accuracy)
```

```
1/1 [==============================] - 135s 135s/step - loss: 0.1087 - accuracy: 0.9545
Loss : 0.10874897241592407
Accuracy : 0.9544937610626221
```

```
Precision: 0.9573820395738204
Recall: 0.9812792511700468
confusion matrix: tf.Tensor(
[[629  12]
 [ 28 210]], shape=(2, 2), dtype=int32)
```

כעת נבחן את האימון של שכבת הקונבולוציה האחרונה בארכיטקטורה. נשים לב שבשני המקרים בחרנו אימון מספר זהה של שכבות קונבולוציה בשביל שתהיה התאמה. גם פה, לקחנו שכבות נוספות שבאות אחרי שכבת הקונבולוציה עצמה, ובמקרה זה אלו Batch Normalization

```
for i, layer in enumerate(BasicModel.layers):
  layer.trainable = 299 <= i <= 301
```

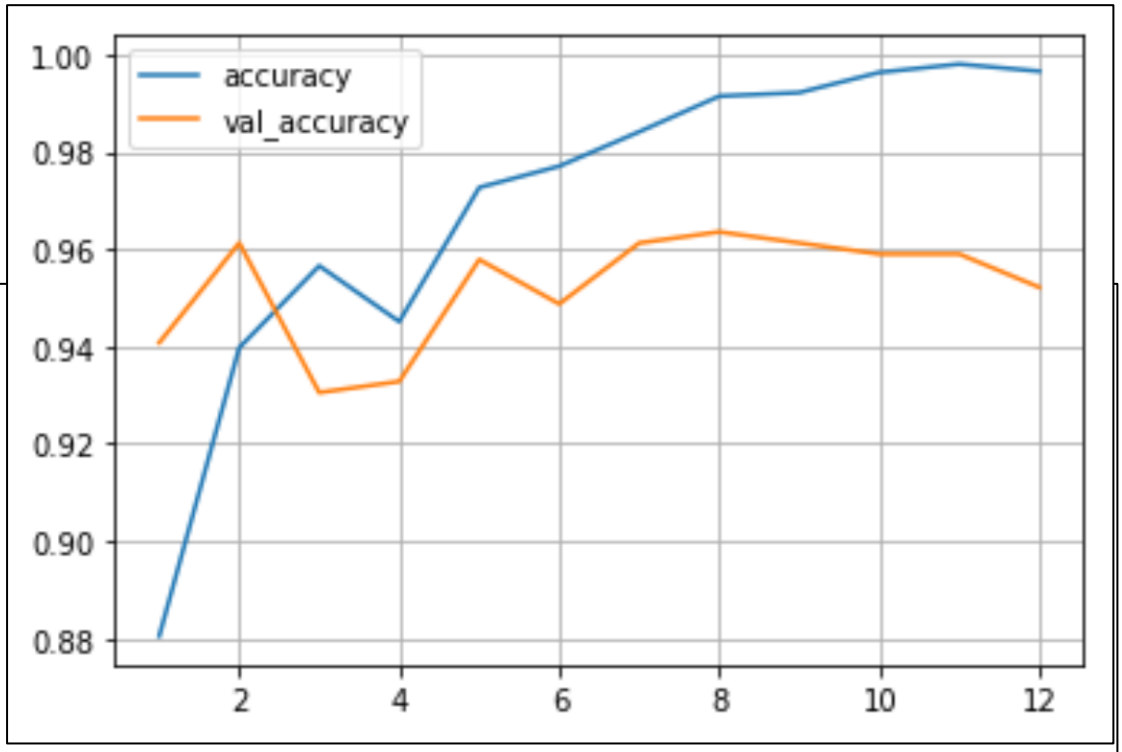
```
BasicModel.layers[299:302]
```

```
[<keras.layers.convolutional.Conv2D at 0x7f760329b290>,
 <keras.layers.normalization.batch_normalization.BatchNormalization at 0x7f76032c9fd0>,
 <keras.layers.core.activation.Activation at 0x7f760335ef10>]
```

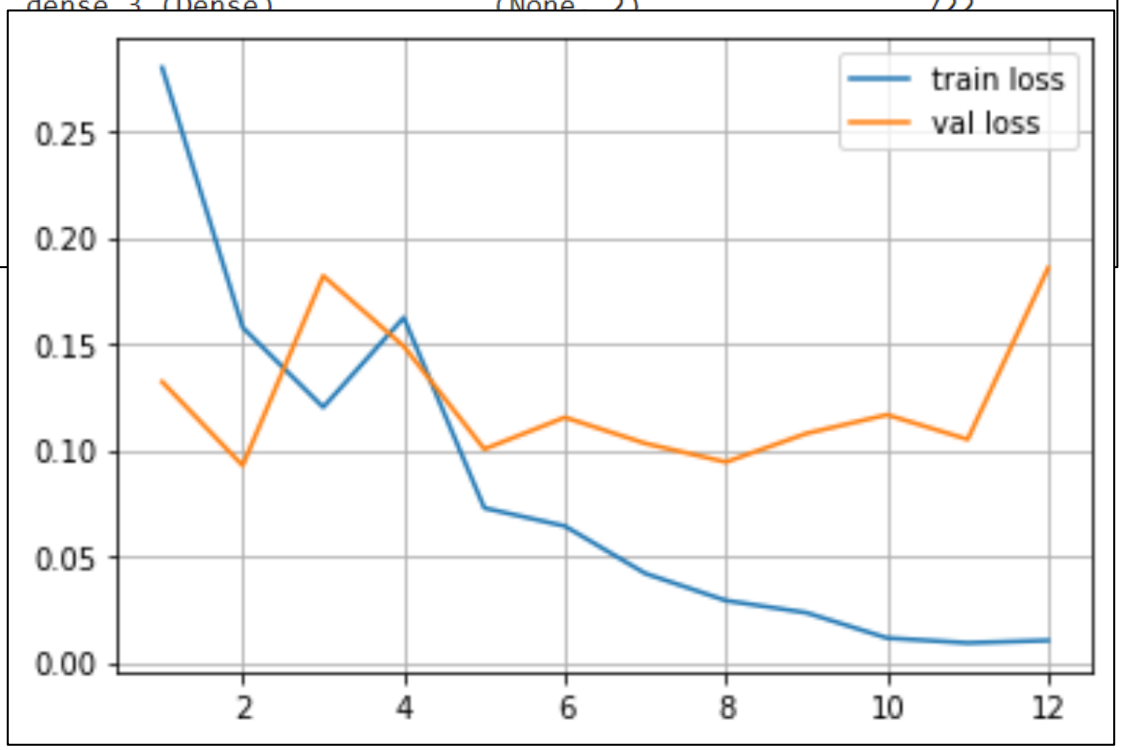
ואקטיבציה.

את שכבות ה fully connected נשאיר כמו שהן, והפעם נקבל שסיכום המודל יהיה:

כלומר הפעם האופטימיזציה תהיה על 414748 פרמטרים. להלן התוצאות שנקבל:



dense_3 (Dense) (None, 2) 722



ודיוק המודל על סט המבחן:

```
loss, accuracy = model.evaluate(TestData)
print("Loss :", loss)
print("Accuracy :", accuracy)
```

```
1/1 [==============================] - 144s 144s/step - loss: 0.1337 - accuracy: 0.9716
Loss : 0.13368432223796844
Accuracy : 0.9715585708618164
```

```
Precision: 0.98125
Recall: 0.9797191887675507
confusion matrix: tf.Tensor(
[[628  13]
 [ 12 226]], shape=(2, 2), dtype=int32)
```

קיבלנו שרמת הדיוק באימון בשכבת הקונבולוציה האחרונה עלתה, אם כי בשיעור קטן יחסית. עם זאת, יכול להיות שהשינוי אינו ניכר בגלל רשת ה fully conneceted שהוספנו בסוף הארכיטקטורה, שעוזרת בעצמה להביא את כל המודל לביצועים גבוהים במיוחד בכל מקרה.

למרות זאת, נוכל לאשר את הטענה התיאורטית, לפיה אימון השכבות האחרונות בארכיטקטורה יותר מתאים לביצוע fine tuning מאשר האימון של השכבות האחרונות. מסקנה נוספת שעולה מסעיף זה הוא שהוספת שכבות fully connected בסוף ארכיטקטורה, ממש משפר את ביצועי המודל במקרה בו צריך להשתמש ב transfer learning.

סעיף ט

כידוע, סט הנתונים שלנו אינו מאוזן. הוא מכיל הרבה יותר תמונות עם תוצאה חיובית (כלומר חולה בדלקת ריאות) מאשר תמונות עם תוצאה שלילית. דבר זה יכול לגרום להטיה בתוצאות הרשת, ויכול להיות שהיא תתמודד פחות טוב עם העולם האמיתי, בו השכיחות של דלקת ריאות הרבה פחות גבוהה. לשם כך עלינו לאזן את ה dataset, וניתן לעשות זאת במספר דרכים.

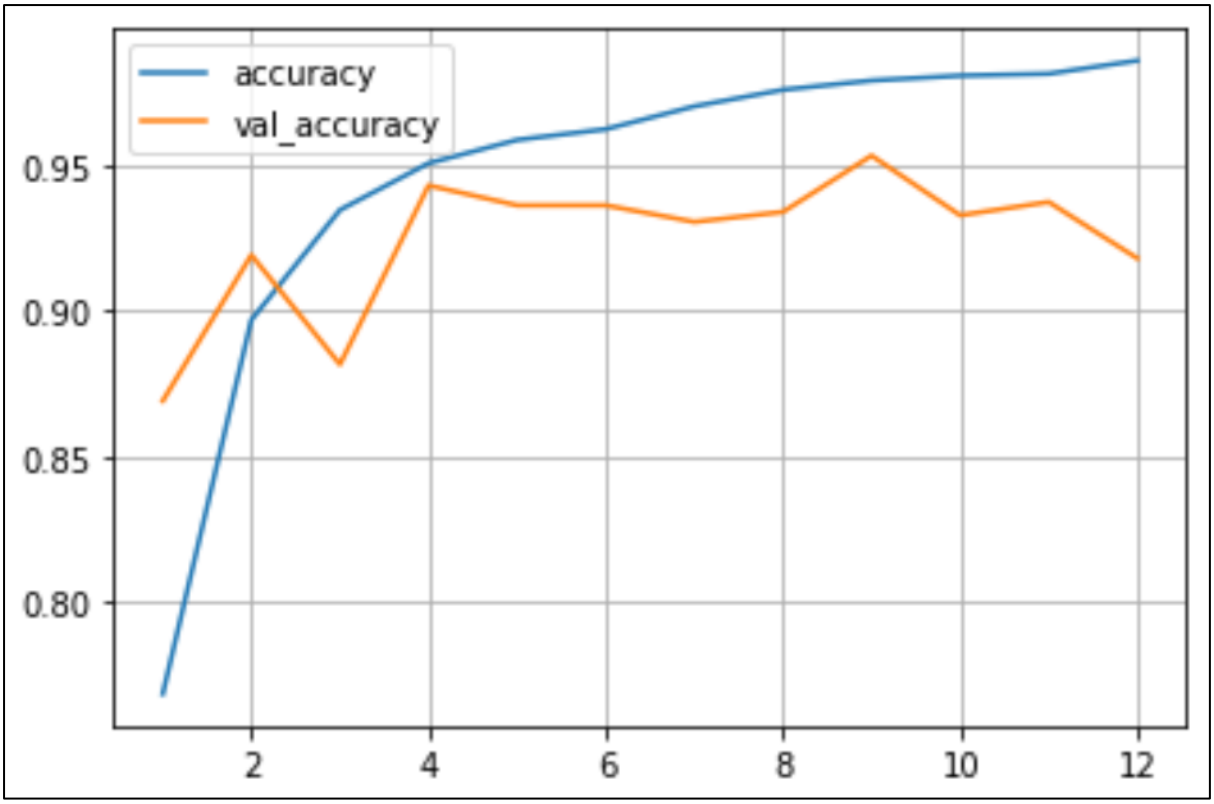
1. הוספת תמונות של תוצאה שלילית – ניתן להוסיף תמונות עד לרמה שבה כמות התמונות בסט הנתונים תהיה שווה או גדולה מכמות התמונות שמתויגות כתוצאה חיובית (לפי היחס במציאות). הוספת התמונות יכולה להתבצע במספר דרכים:
   a. שכפול תמונות באופן אקראי – כלומר ניתן לבחור תמונות מסוימות בסט הנתונים וליצור להן עותקים.
   b. הוספת אוגמנטציות – כלומר לבחור תמונות באופן אקראי ולהוסיף להן עותק עם אוגמנטציות מסוימות, כמו למשל הזזה (לכיוון אקראי), סיבוב (בזווית כלשהי), צביעה מחדש, כיווץ ומתיחה וכו'.
   c. במקרים מסוימים משתמשים ב GAN על מנת לייצר תמונות חדשות מאפס, אבל נשים לב שבשביל אפשרות כזאת צריך כמות גדולה של תמונות מלכתחילה.

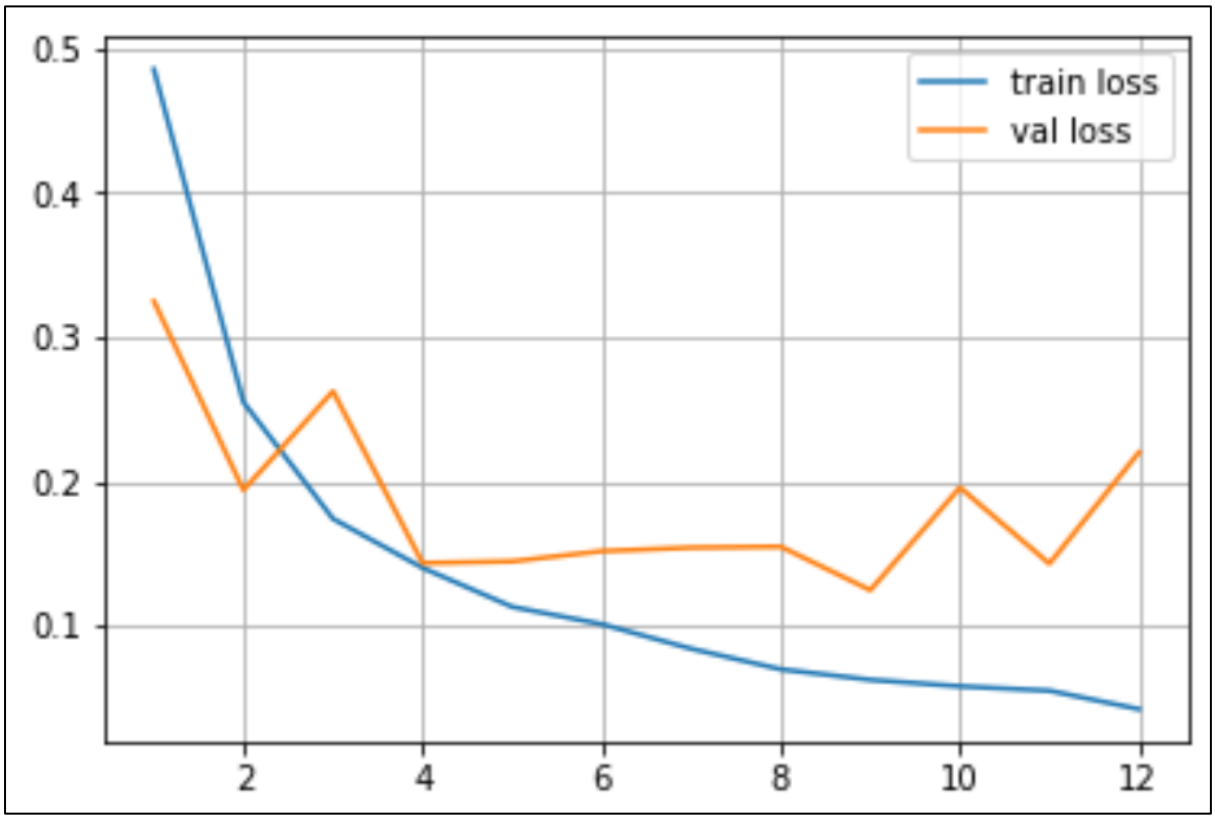
   נשים לב שתמיד כאשר אנחנו מוסיפים תמונות שאינן חדשות ומקוריות אנחנו מסתכנים בכך שהרשת תיכנס לתהליך של overfitting במהלך תהליך האימון, ולכן כדאי להיזהר עם כלים אלו ולהשתמש בהם כאשר חוסר ההתאמה בכמות התמונות הוא משמעותי.

2. הורדה/מחיקה של תמונות המתויגות עם תוצאה חיובית – עד לרמה שבה ייווצר איזון בין התמונות המתויגות כ-"חיובי" והתמונות המתויגות כ-"שלילי". בשיטה זו אנחנו נמנעים יותר מ overfitting מכיוון שאנחנו לא מוסיפים תמונות אלא עובדים עם פחות ממה שקיים.

דרך נוספת בה ניתן להתמודד עם סט נתונים שאינו מאוזן הוא לערוך שינויים במשקלים אותם הרשת לומדת. בעצם זה שנסתכל על המשקלים וההתפלגויות נוכל לראות את ההטיות שנוצרו ברשת, ולתקן אותם ע"י שיטוח ריכוזיות מסוימת שנוצרה. כלומר כאשר רואים מהמשקלים שהתפלגות מסוימת מציגה הטיה בניחושים ניתן לשנות אותה ידנית כך שהמשקלים יסתדרו לתוצאות לא מוטות.

בסעיף זה ניסינו לבדוק את ההשפעה של הוספת תמונות ע"י שכפול באופן אקראי. הוספנו לסט האימון כמות גדולה של תמונות המתויגות כ Normal, כך שכעת בסט האימון יש כמות זהה של תמונות המייצגות תוצאה חיובית ותמונות המייצגות תוצאה שלילית. הרשת שבחרנו היא הארכיטקטורה Inception V3 ללא אימון מחדש ואחריה שתי שכבות fully connected ושכבות dropout. להלן התוצאות שנקבל מתהליך האימון:





והתוצאות המתקבלות מסט המבחן:

```
loss, accuracy = model.evaluate(TestData)
print("Loss :", loss)
print("Accuracy :", accuracy)
```

```
1/1 [==============================] - 148s 148s/step - loss: 0.1418 - accuracy: 0.9534
Loss : 0.14181996881961823
Accuracy : 0.9533560872077942
```

```
Precision: 0.9838709677419355
Recall: 0.9516380655226209
confusion matrix: tf.Tensor(
[[610  31]
 [ 10 228]], shape=(2, 2), dtype=int32)
```

נוכל כעת להשוות את התוצאות שקיבלנו לאלו שהתקבלו בתוצאות הראשונות שמוצגות בסעיף ז, שם עשינו את אותו אימון אבל על ה dataset המקורי. ראשית נוכל לשים לב לירידה בדיוק בערך באחוז, שאומנם לא משמעותי אבל כן יכול להיות שהוא נובע מהשינוי בסט הנתונים. כפי שאמרנו, בגלל שכפול התמונות קיים סיכון גבוה במיוחד ל overfitting, ולכן יכול להיות שגם אם זה לא קרה במקרה הזה, השינוי בסט האימון הקשה על הרשת להתאמן בלי ללמוד בע"פ.

בנוסף, נוכל לשים לב לעלייה ב Precision. נזכיר את הנוסחה ל Precision:

$$Precision = \frac{\#\ True\ Positive}{\#\ True\ Positive + \#\ False\ Positive}$$

מהנוחה ניתן להבין ש precision הוא מדד לגודל ה false positive בסט המבחן, כלומר precision גבוה אומר שלא יהיו הרבה מטופלים שיקבלו תוצאה חיובית כאשר בפועל הם בריאים.

נשים לב שבעקבות שכפול התמונות של Normal, קיבלנו עלייה ב precision מ 96.6% ל 98.3%, עלייה משמעותית שבהחלט ניתן לומר שהיא נובעת משינוי שהכנסנו בסט הנתונים. העלייה ב precision הגיונית לחלוטין: הגדלנו את כמות התמונות המתויגות כתוצאה שלילית, ולכן הרשת לומדת לתת תוצאה חיובית בשכיחות קטנה יותר (כי עכשיו היה בסט האימון יותר תמונות המתויגות כתוצאה שלילית), מה שגורם לכמות ה false positive לרדת. נסייג ונאמר שזה אומנם תוצאה טובה ביחס לציפייה, אך במימוש של מערכת כזו במציאות דווקא ה recall הוא זה שיותר חשוב ופחות ה precision ולכן לא בטוח שהשכפול של התמונות הוא אידאלי למערכת מציאותית.

לינק לקוד ב GitHub:

https://github.com/OfirGuriel/DeepLearning2.git