יונתן מנחם – 203772611
אלון מצרי – 311503841

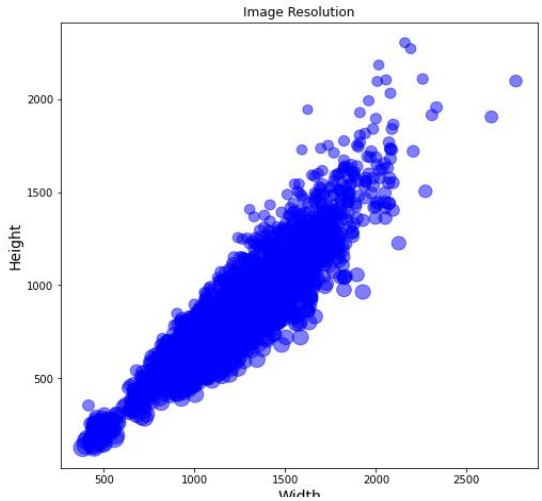
למידה עמוקה ליישומי ראיה ממוחשבת – זיהוי דלקת ריאות

בעבודה זו נדרשנו לבנות מודל רשת נוירונים אשר ידע להבחין בין שני סוגים של צילומי דלקת ריאות, תקין ושאינו תקין.

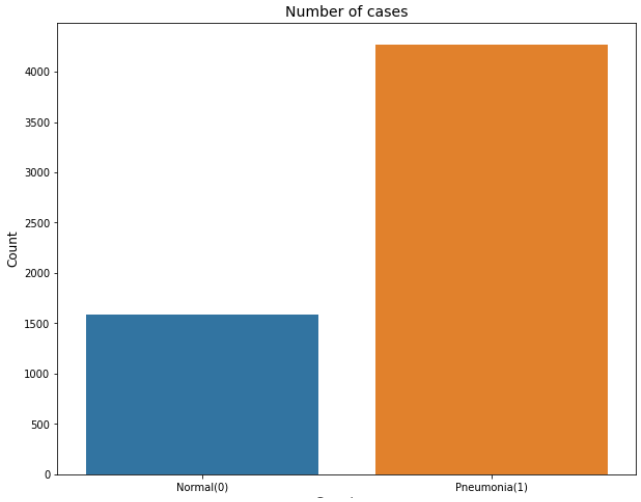
למעבר לגיט לחצ/י כאן.

חשוב לציין כי נעשתה עבודה רבה באופטימיזציה של המודל והמענה על הסעיפים השונים לא בהכרח יוצג בסדר המתאים (למשל, כבר בסעיף א' ברשת הפשוטה, הוספנו שכבת "Dropout" "Learning Rate" אשר שייכת לסעיף ה').

ראשית, ביצענו אקספלורציה למידע על מנת להבין כמה תמונות קיימות לנו במאגר ובאיזו רזולוציה [איור 2]. כמו כן, היה חשוב להבין מהי התפלגות התמונות בקטגוריות השונות [איור 1] (דבר שיתברר כחשוב מאוד ונתייחס אליו בהמשך).



איור 1: התפלגות רזולוציית התמונות לפי אורך ורוחב



איור 2: התפלגות התמונות לפי קטגוריה

לאחר טעינת התמונות לתוך משתנים מקומיים, ביצענו הקטנה של התמונות לטובת התאמתם למודל. לאחר ביצוע האקספלורציה, (ומעט ניסוי וטעיה עם המודל) הגענו למסקנה שהגודל האידאלי יהיה 256x256 פיקסלים. בגודל זה, התמונות מצד אחד, לא יאבדו יותר מדי מידע אשר חשוב למודל ומצד שני שיהיו כמה שפחות תמונות שקטנות מהגודל הזה ויעברו הגדלה.

בנוסף, מאחר והתמונות כולן בצבעי שחור לבן כיאה לצילומי רנטגן בחרנו להמיר את כולן למימד אחד לטובות הקלה על המודל בשעת האימון.

בתחילה, נדרשנו לבנות רשת נוירונים Fully Connected ללא שכבות קונבולוציה ("המודל הבסיסי") [איור 3].

במוצא המודל בחרנו בשכבת אקטיבציה מסוג "Sigmoid" משום שיש לנו רק שני סוגי תוצאה, צילום תקין ושאינו תקין.

כמות הפרמטרים במודל זה הייתה: 786,596

```
num_classes = 2
input_shape = (256, 256, 1)

basic_model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Dense(10, activation="relu"),
        layers.Dense(8, activation="relu"),
        layers.Dense(6, activation="relu"),
        layers.Flatten(),
        layers.Dropout(0.15),
        layers.Dense(num_classes, activation="sigmoid"),
```

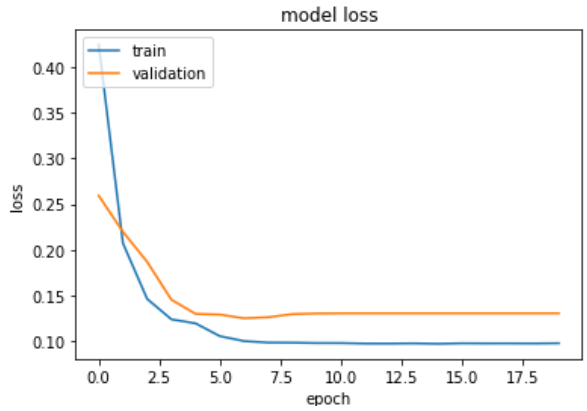
איור 3: מודל Fully Connected ללא שכבות קונבולוציה

יונתן מנחם – 203772611
אלון מצרי – 311503841

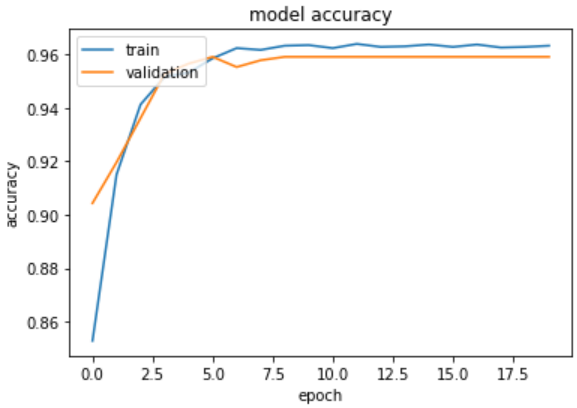
לאחר אימון הרשת ניתן לראות כי הרשת התכנסה [גרפים 1,2] אך דיוק הרשת ("Accuracy") עומד על כ- 77% [איור 4].

כאמור, קיים חוסר איזון בהתפלגות הנתונים בקטגוריות השונות בעיה זו נקראת בעיית "Imbalance" והיא נפוצה בין היתר בתחומי הרפואה.

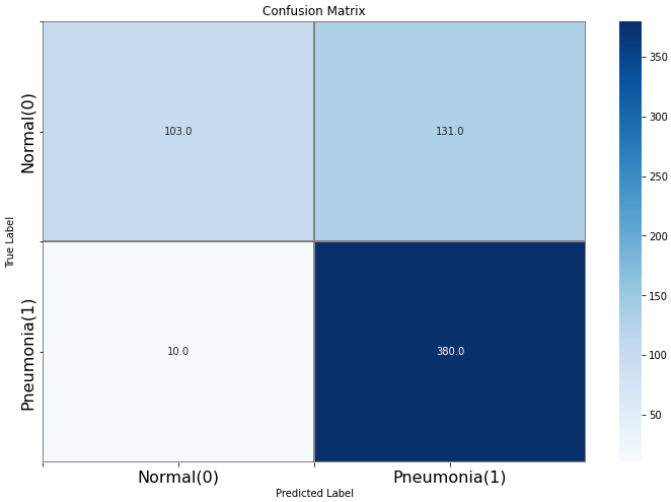
ניתן לראות על פי מטריצת המבוכה [איור 5] את דיוק המודל ומשם ניתן לגזור מספר מטריקות ביצועים (Accuracy, Recall, Precision).



גרף 1: גרף ה-Loss כפונקציה של ה-epoch



גרף 2: דיוק המודל על סט האימון והוולידציה כפונקציה של ה-epoch



איור 5: מטריצת המבוכה עבור המודל הבסיסי

```
loss, accuracy = basic_model.evaluate(test_ds)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
✓ 0.7s
20/20 [==============================] - 1s 19ms/step
Test loss: 0.758037269115448
Test accuracy: 0.7740384340286255
```

איור 4: דיוק המודל הבסיסי

לפי התפלגות התמונות חישבנו כי כ-73% מכלל התמונות שייכות לצילום לא תקין. כלומר, קיימת הסתברות של 73% שנבחר בצילום לא תקין גם אם הבחירה הייתה שרירותית.

על מנת להתגבר על בעיה זו, קיימות מספר דרכים. בחרנו לחשב משקולים התחלתיים שונים בהתאם לכמות התמונות בכל קטגוריות. בפעולה זו בעצם ניסינו ללמד את המודל כי התמונות התקינות "חשובות יותר" מהתמונות שאינן תקינות [איור 6].

בנוסף, ניתן לבצע אוגמנטציות שונות על מנת להגדיל את מספר התמונות התקינות כדי שתהיה תואמת למספר התמונות שאינן תקינות.

מאחר ומדובר בבעיית סיווג של מידע רפואי קיימת חשיבות לסוג הדיוק של המודל.

כאן בא לידי ביטוי השימוש במטריקות שונות להערכת ביצועי המודל.

```
weight_for_0 = (1 / COUNT_NORMAL)*(TRAIN_IMG_COUNT)/2.0
weight_for_1 = (1 / COUNT_PNEUMONIA)*(TRAIN_IMG_COUNT)/2.0

class_weight = {0: weight_for_0, 1: weight_for_1}

print('Weight for class 0: {:.2f}'.format(weight_for_0))
print('Weight for class 1: {:.2f}'.format(weight_for_1))
✓ 0.3s
Weight for class 0: 1.85
Weight for class 1: 0.69
```

איור 6: הגדרת המשקולים ההתחלתיים למודל

יונתן מנחם – 203772611
אלון מצרי – 311503841

מטריצת המבוכה עוזרת לנו להציג את המידע בצורה ויזואלית ולהבין מתי המודל טועה. בנוסף היא עוזרת לנו לחלץ מטריקות שונות למדידת הביצועים. במקרה הבינארי, קיימת מטריצה בגודל 2x2 ובה ארבעה אלמנטים באופן הבא:

		Predicted	
		Positive	Negative
Ground-Truth	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

מטריקת Accuracy מתארת באופן כללי את ביצועי המודל ושימושית כאשר לכל הקטגוריות חשיבות זהה. החישוב מתבסס על היחס בין מספר התחזיות הנכונות (TP+TN) לבין סה"כ התחזיות (TP+TN+FP+FN).

מטריקת Precision מתארת את דיוק המודל בסיווג תמונה כחיובית, במילים אחרות ניתן לומר כי זהו מדד האמינות של המודל. החישוב מתבסס על היחס בין מספר הדגימות החיוביות שזוהו בצורה נכונה (TP) לבין סכום הדגימות החיוביות (TP+FP).

מטריקת Recall מתארת את היכולת של המודל לזהות תמונות חיוביות ללא קשר למספר התמונות שהוגדרו כשליליות (ללא תלות ב-Precision). החישוב מתבסס על היחס בין מספר הדגימות החיוביות שזוהו בצורה נכונה (TP) לבין סכום הדגימות שזוהו בצורה נכונה והדגימות החיוביות שזוהו כשליליות (TP+FN).

הבחירה בין המטריקות השונות תלויה בבעיה אותה מנסים לפתור. במקרה שלנו נעדיף לבחור ב-Recall משום שהמטרה שלנו היא לזהות דגימות שזוהו כחיוביות (המטופל חולה והוא נדרש לקבל טיפול) ואנחנו פחות דואגים מדגימות שליליות שיסווגו כחיוביות (מטופל בריא שבטעות יזוהה כחולה). ניתן לראות את תוצאות המטריקות השונות באיור 7.

```
# prescision and recall calculation

from sklearn.metrics import classification_report
print(classification_report(test_labels, predictions))
✓ 0.3s
              precision    recall  f1-score   support

         0.0       0.91      0.44      0.59       234
         1.0       0.74      0.97      0.84       390

    accuracy                           0.77       624
   macro avg       0.83      0.71      0.72       624
weighted avg       0.81      0.77      0.75       624
```

איור 7: תוצאות המטריקות השונות על המודל הבסיסי

ניתן לראות כי המודל הבסיסי קיבל ציון Recall של 97% (כאשר חיובי=דלקת ושלילי=בריא). מצד שני, תוצאת ציון ה-Precision היה 74% והמודל סיווג בשוגג מטופלים בריאים כחולים.

בהמשך נרצה להוסיף שכבות נוספות למודל על מנת להגדיל את הדיוק גם במטריקת ה-Precision.

```
num_classes = 2
input_shape = (256, 256, 1)

advanced_model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu", strides=(2, 2), padding='same'),
        layers.Conv2D(128, kernel_size=(3, 3), activation="relu", strides=(1, 1), padding='same'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(128, kernel_size=(3, 3), activation="relu", strides=(2, 2), padding='same'),
        layers.Conv2D(256, kernel_size=(3, 3), activation="relu", strides=(1, 1), padding='same'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.2),
        layers.Dense(50, activation="relu"),
        layers.Dense(15, activation="relu"),
        layers.Dense(num_classes, activation="softmax"),
    ]
```

איור 8: המודל המתקדם - CNN

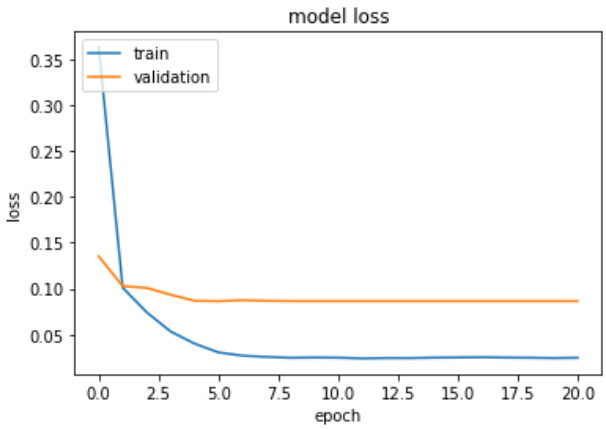
לאחר הוספת שכבות הקונבולוציה למודל ("המודל המתקדם") [איור 8] קיבלנו כמות פרמטרים גדולה יותר: 3,794,895.

יונתן מנחם – 203772611
אלון מצרי – 311503841

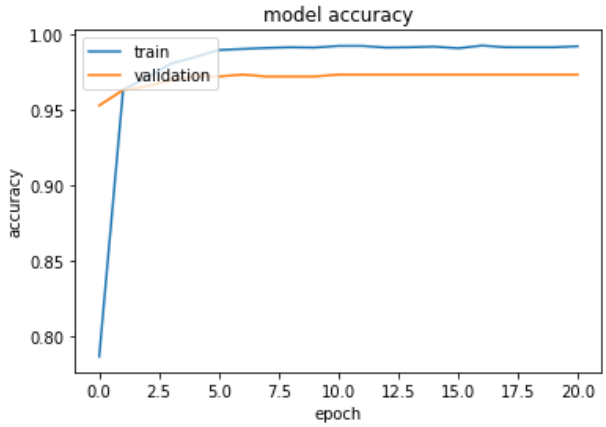
לאחר אימון הרשת ניתן לראות כי הרשת התכנסה [גרפים 3,4] אך דיוק הרשת ("Accuracy") עדיין ללא שיפור ועומד על כ- 77% [איור 9].

```
loss, accuracy = advanced_model.evaluate(test_ds)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
✓ 0.3s
20/20 [==============================] - 0s 12ms/step
Test loss: 1.323726773262024
Test accuracy: 0.7724359035491943
```

איור 9: דיוק המודל המתקדם

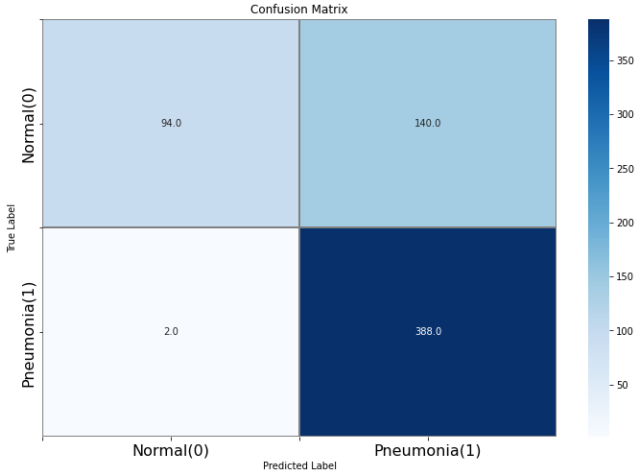


גרף 3: גרף ה-Loss כפונקציה של ה-epoch



גרף 4: דיוק המודל על סט האימון והוולידציה כפונקציה של ה-epoch

על בסיס מטריצת המבוכה [איור 10] ניתן לראות כי חל שיפור במודל המתקדם מבחינת מטריקת ה-Recall (רק 2 דוגמאות מתוך סט הטסט לא זוהו בהצלחה)



איור 10: מטריצת המבוכה עבור המודל המתקדם

```
# prescision and recall calculation

from sklearn.metrics import classification_report
print(classification_report(test_labels, predictions))
✓ 0.3s
              precision    recall  f1-score   support

         0.0       0.98      0.40      0.57       234
         1.0       0.73      0.99      0.85       390

    accuracy                           0.77       624
   macro avg       0.86      0.70      0.71       624
weighted avg       0.83      0.77      0.74       624
```

איור 11: תוצאות המטריקות השונות על המודל המתקדם

ניתן לראות [איור 11] כי המודל המתקדם קיבל ציון Recall של 99% (באשר חיובי=דלקת ושלילי=בריא). מצד שני, תוצאת ה-Precision הייתה 73% והמודל סיווג בשוגג מטופלים בריאים כחולים.

בהמשך נרצה להוסיף אוגמנטציות שונות לסט האימון על מנת לתת מגוון רחב יותר של שונות בין התמונות השונות.

חשוב לציין כי אסור בשום שלב לגעת בסט הטסט ולבצע עליו שינויים. מותר לחשוף את סט הטסט למודל אך ורק בסוף האימון בצורה המקורית שלו.

קיימות מספר דרכים לבצע אוגמנטציות לתמונות הן בשלב טעינת ועיבוד המידע והן בשלב האימון. מאחר ולא נדרשנו בשלב זה לביצוע ספציפי, החלטנו להכניס שכבות אוגמנטציה בתוך המודל עצמו ("מודל האוגמנטציות") [איור 12].

יונתן מנחם – 203772611
אלון מצרי – 311503841

```python
num_classes = 2
input_shape = (256, 256, 1)

augmentation_model = keras.Sequential(
    [
        keras.Input(shape=input_shape),

        # Data Augmentation
        preprocessing.RandomContrast(0.3),
        preprocessing.RandomRotation(0.18),
        preprocessing.RandomZoom(0.21),

        # body
        layers.BatchNormalization(),

        layers.Conv2D(64, kernel_size=(3,3) , strides=(1, 1) , padding = 'same' , activation = 'relu'),
        layers.MaxPooling2D(pool_size=(2,2) , strides=(2, 2) , padding = 'same'),

        layers.Conv2D(64, kernel_size=(3,3) , strides=(1, 1) , padding = 'same' , activation = 'relu'),
        layers.Dropout(0.2),
        layers.MaxPooling2D(pool_size=(2,2) , strides=(2, 2) , padding = 'same'),

        layers.Conv2D(64, kernel_size=(3,3) , strides=(1, 1) , padding = 'same' , activation = 'relu'),
        layers.MaxPooling2D(pool_size=(2,2) , strides=(2, 2) , padding = 'same'),
        layers.Dropout(0.1),

        layers.Conv2D(128, kernel_size=(3,3) , strides=(1, 1) , padding = 'same', activation = 'relu'),
        layers.MaxPooling2D(pool_size=(2,2) , strides=(2, 2) , padding = 'same'),

        # Head
        layers.Flatten(),
        layers.Dense(units = 128, activation = 'relu'),
        layers.Dropout(0.1),
        layers.Dense(num_classes, activation = 'sigmoid'),
    ]
)
```

איור 12: המודל המתקדם – CNN יחד עם אוגמנטציות שונות

```python
from keras.callbacks import LearningRateScheduler

# This is a sample of a scheduler I used in the past
def lr_scheduler(epoch, lr):
    decay_rate = 0.85
    decay_step = 1
    if epoch % decay_step == 0 and epoch:
        return lr * pow(decay_rate, np.floor(epoch / decay_step))
    return lr
```

איור 13: פונקציית ה-Learning Rate

בנוסף, השתמשנו לאורך כל הדרך ב-Learning Rate אשר משתנה בהתאם לקצב הלמידה [איור 13] אשר סייעה בהתכנסות מהירה יותר של המודל ובפונקציית Early Stopping אשר הפסיקה את אימון המודל לאחר שהתוצאות לא השתפרו.

לאחר אימון הרשת ניתן לראות כי הרשת התכנסה [גרפים 5,6] ודיוק הרשת ("Accuracy") עומד על כ- 88% [איור 14].

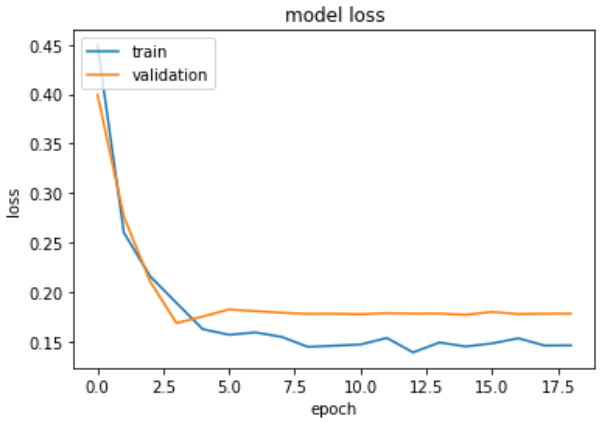
```python
loss, accuracy = augmentation_model.evaluate(test_ds)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
```

✓ 0.4s

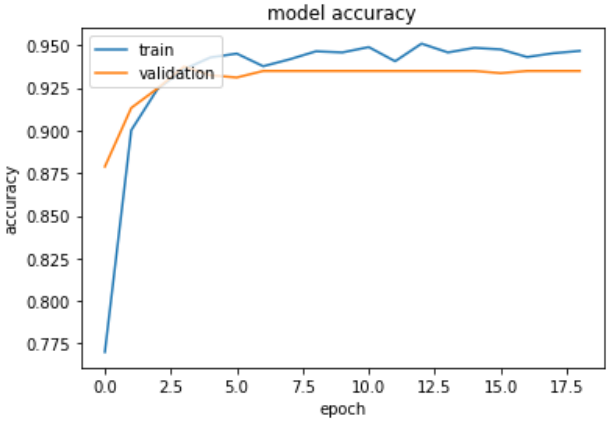
```
20/20 [==============================] - 0s 15ms/step -
Test loss: 0.32091212272644043
Test accuracy: 0.8782051205635071
```

איור 14: דיוק המודל האוגמנטציות

יונתן מנחם – 203772611
אלון מצרי – 311503841



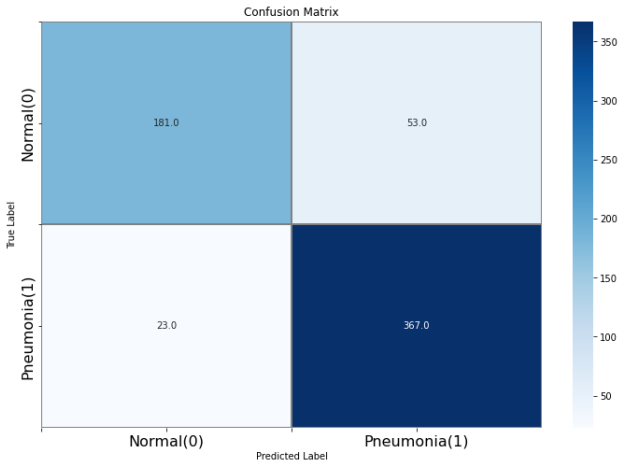
גרף 5: גרף ה-Loss כפונקציה של ה-epoch



גרף 6: דיוק המודל על סט האימון והוולידציה כפונקציה של ה-epoch

על פי מטריצת המבוכה [איור 15] ניתן לראות כי חל שיפור ניכר במטריקת ה – Precision (סיווג אדם בריא כחולה קטן משמעותית) אך חלה ירידה במטריקת ה- Recall.

ניתן לראות [איור 16] כי מודל האוגמנטציות קיבל ציון Recall של 94% (באשר חיובי=דלקת ושלילי=בריא). מצד שני, תוצאת ה-Precision הייתה 87%.



איור 15: מטריצת המבוכה עבור מודל האוגמנטציות

```
# prescision and recall calculation

from sklearn.metrics import classification_report
print(classification_report(test_labels, predictions))
✓ 0.4s
              precision    recall  f1-score   support

         0.0       0.89      0.77      0.83       234
         1.0       0.87      0.94      0.91       390

    accuracy                           0.88       624
   macro avg       0.88      0.86      0.87       624
weighted avg       0.88      0.88      0.88       624
```

איור 16: תוצאות המטריקות השונות על מודל האוגמנטציות

לסיכום, במהלך אימון ואופטימיזציית המודלים השונים, ראינו שקיימים הבדלים גדולים בשימוש בהיפר פרמטרים שונים ובערכים שלהם. אין ספק שעם עוד זמן ויכולת חישובית ניתן היה להגיע לביצועים טובים יותר בכל המטריקות השונות.

במהלך האימונים הרבים שעשינו כן יצא לנו להיתקל ב-Overfitting בהתחלה בגלל באג בקוד (סט הטסט היה בטעות חופף באופן חלקי עם סט הוולידציה), לאחר מכן בכלל רזולוציית תמונה נמוכה מדי ובנוסף, בגלל כמות פרמטרים גבוהה מדי וללא Early Stopping למשך יותר מדי epoch-ים.