מסמך מסכם – זיהוי תמונות של מחלות בכותנה

מוטיבציה:

**בפרויקט הגמר שלנו אנו משתמשים ברשת נוירונים מסוג CNN לזיהוי תמונה ולכן בחרנו בדאטה המתאים לרשת מסוג זה בכדי שנוכל להתנסות בקנה מידה קטן ולהיחשף לקוד ולרעיונות של אנשים אחרים שהשתמשו ברשתות מסוג זה.**

סט המידע שנבחר :

CNN for Cotton disease Prediction ~84 % val Accuracy

[**https://www.kaggle.com/code/anuragupadhyay6212/cotton-disease-prediction-cnn-for-beginners**](https://www.kaggle.com/code/anuragupadhyay6212/cotton-disease-prediction-cnn-for-beginners)

**תהליך החקירה:**

סט הנתונים – סט תמונות מהמאגר של Kaggle הסט מחולק ל4 קטגוריות עלה עם מחלה, עלה ללא מחלה. שיח עם מחלה, שיח ללא מחלה.

סך הכל 2300 תמונות מתוכן 1270 כ55% עם מחלות השאר ללא מחלות. אימון 80% ולדציה 10% טסט 10%,כאשר שומרים על יחס של רבע בקירוב בכל קבוצה .

התמונות צבעוניות וגודלן גובה-256 אורך -256 שיטת צבעים-RGB(3).

היה שימוש בדאטה אוגמנטיישן סוגי האומנטציה הינם שינוי גובה,אורך, רוטציה, זום מריחת צבעים והפיכה.

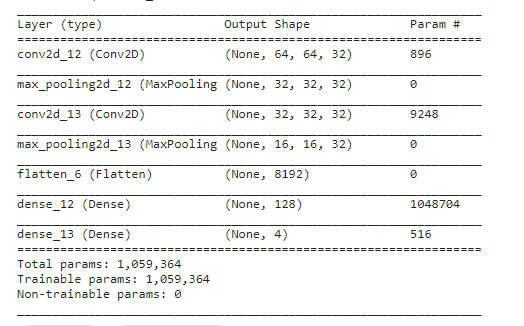
**קישור למחברת Kaggle**:

<https://www.kaggle.com/code/agrinurture/cotton-disease-prediction-cnn>

**REPORT**

Cotton disease Prediction CNN

Architecture of the model:



starting point:

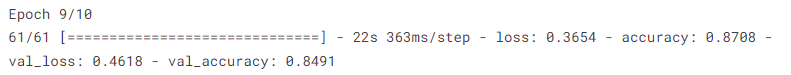
Epochs:10

Batch size = 32(on data augmentation)

Last Epoch:

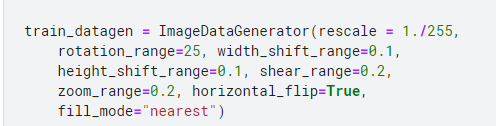


Best Epoch:



First attempt: (without evaluate the model this run was for sanity test)

After a little research we decided to add data augmentation and increase number of epochs(15):



And to change our training function to **keras.fit\_generator()** due to our data augmentation:

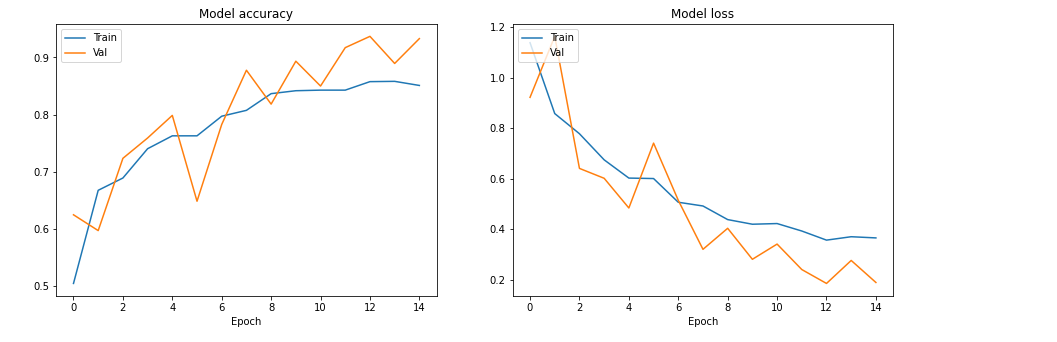
**.fit** is used when the entire training dataset can fit into the memory and no data augmentation is applied.  
**.fit\_generator** is used when either we have a huge dataset to fit into our memory or when data augmentation needs to be applied.

Last Epoch:



Best Epoch:





Conclusion:

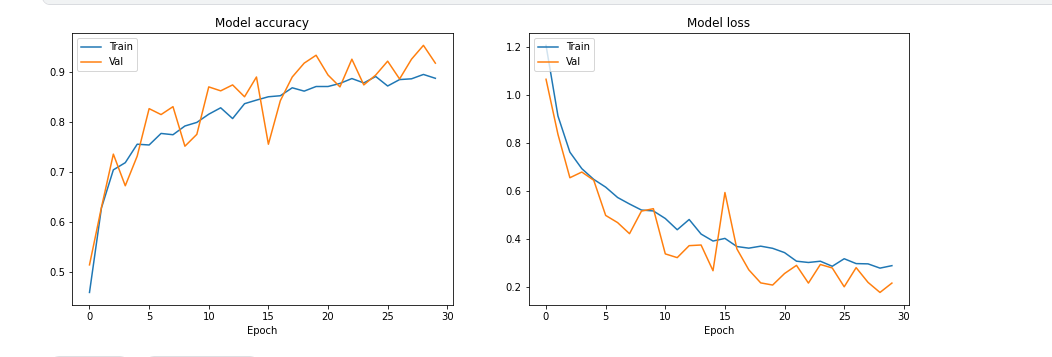
Better Result

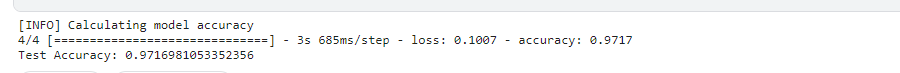
Second attempt: (saved version)

In this try we only increase to Epochs to 30 with same configuration

Last Epoch:

  
Best Epoch:

  
  
  
  
Evaluate:



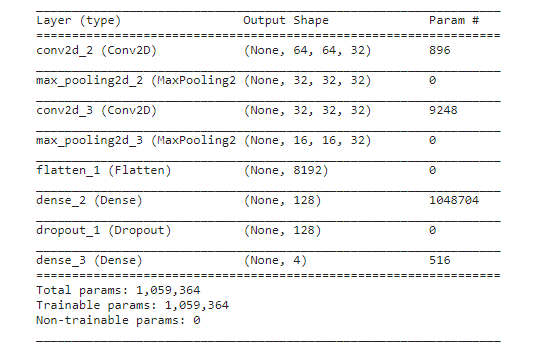
Conclusion:  
got on our best epoch to 0.95 acc on validation little bit better

On the **Test** we got better results than the validation 🡪 0.97

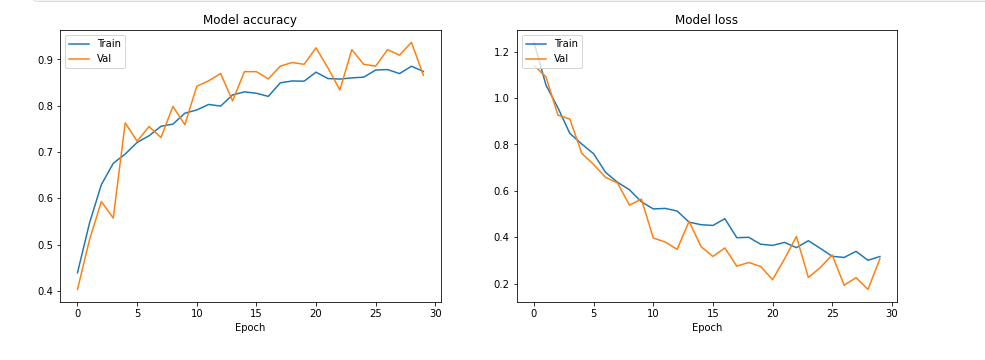
Third attempt:

In this attempt we stayed with same configuration and add dropout of 0.5

Architecture of a model:



Last Epoch:

  
Best Epoch:  
  


Evaluate:  
  
Conclusion:  
Got worse result on the test 0.83

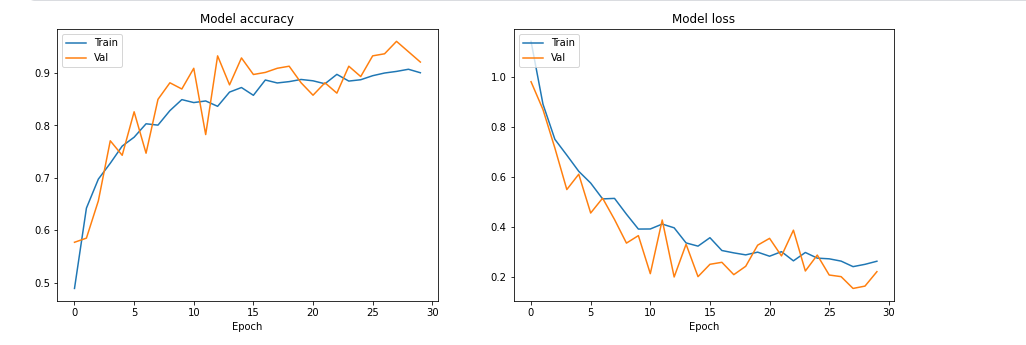
Fourth attempt:

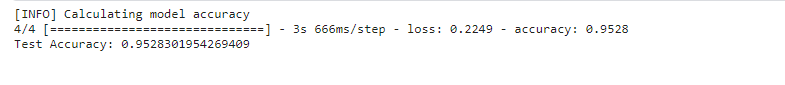
In this attempt we stayed with same configuration and Change dropout to 0.2

Last Epoch:



Best Epoch:

Evaluate:  
  
Conclusion:

Got better result than the last attempt 0.95 on the test

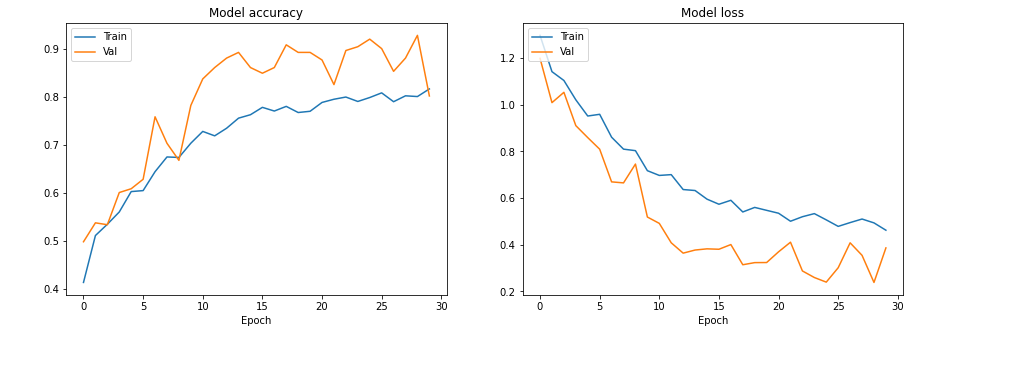
Fifth attempt:

In this attempt we stayed with same configuration and Change dropout to 0.8

Last Epoch:

  
Best Epoch:





Evaluate:

A picture containing text

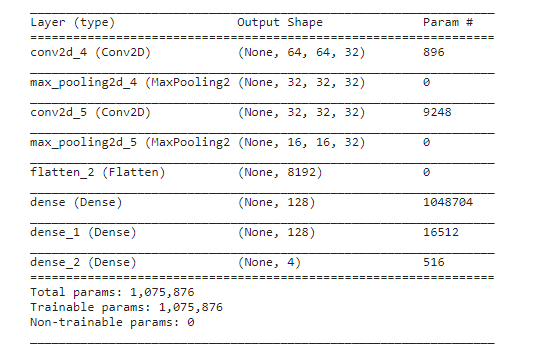
Description automatically generated  
Conclusion:

Got worse result, large drop out Doesn't serve the purpose little drop-out sometimes can do better.

Sixth attempt:

In this attempt we stayed with same configuration and add L1 Regularization layer with value of 0.0005.

Architecture of a model:



Last Epoch:



Best Epoch:  
  
Chart, line chart, histogram

Description automatically generated

Evaluate:

Graphical user interface

Description automatically generated with medium confidence

Conclusion:  
Got worse result

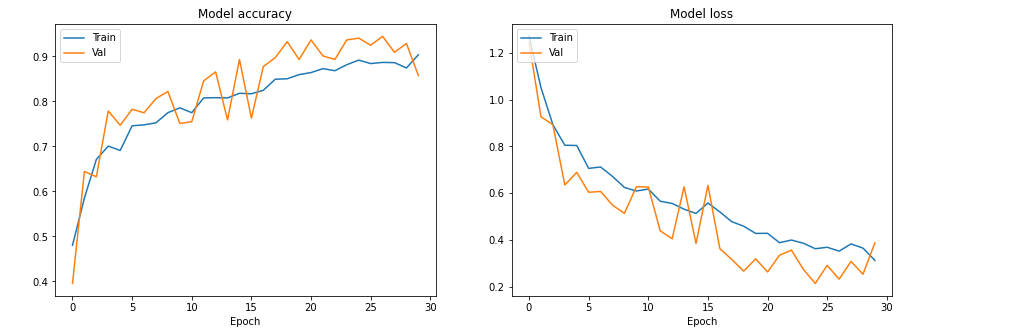
Seventh attempt:

In this attempt we stayed with same configuration and add L2 Regularization layer with value of 0.0005.(same architecture of a model as the last attempt).

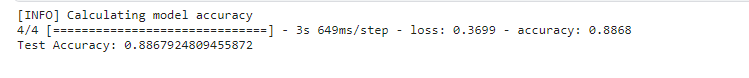
Last Epoch:



Best Epoch:

Evaluate:

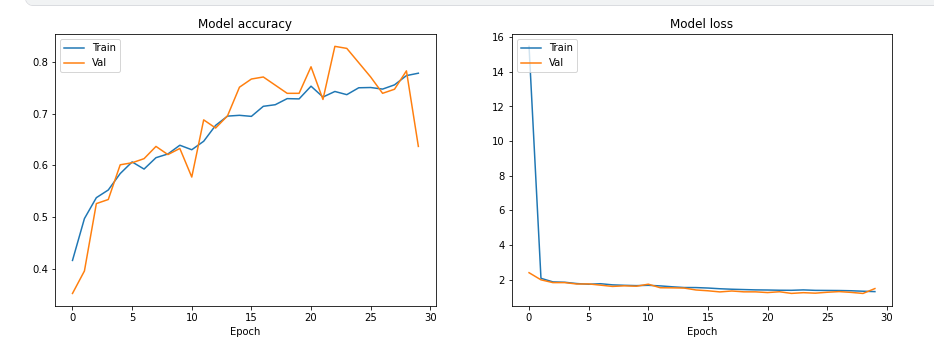
  
Conclusion:  
Got worse result

Eighth attempt:

In this attempt we stayed with same configuration and add L1 Regularization layer with value of 0.005.(same architecture of a model as the last attempt).

Last Epoch:

  
Best Epoch:

  
  
Evaluate:

Graphical user interface, Word

Description automatically generated

Conclusion:

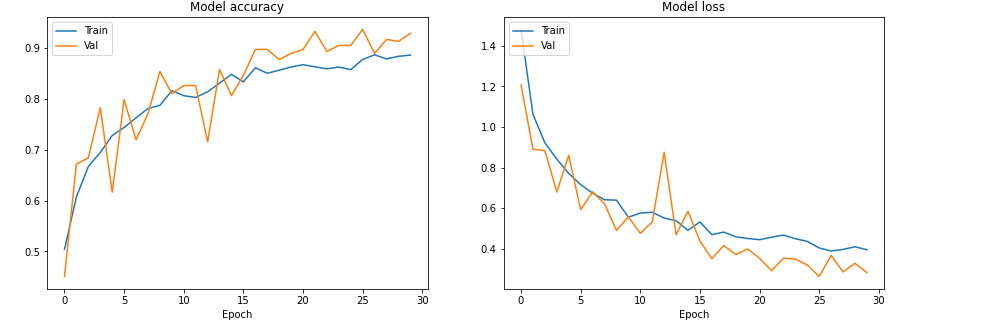
With 10 times bigger value of Regularization we got very bad result

Ninth attempt:

In this attempt we stayed with same configuration and add L2 Regularization layer with value of 0.005.(same architecture of a model as the last attempt).

Last Epoch:

  
Best Epoch:

Evaluate:

Logo

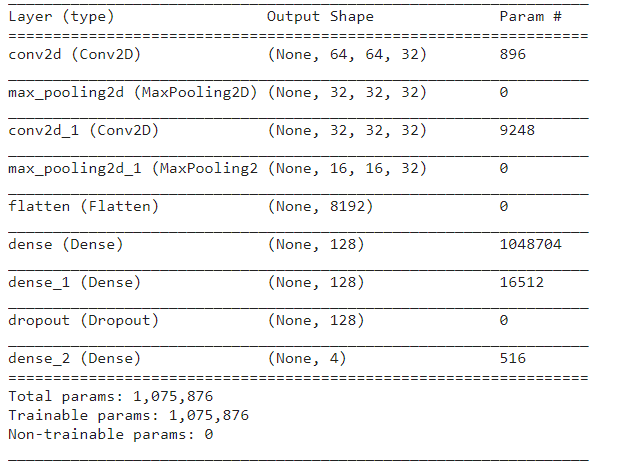
Description automatically generated with medium confidence  
Conclusion:

With 10 times bigger value of Regularization value we got very nice result one of the best so far.

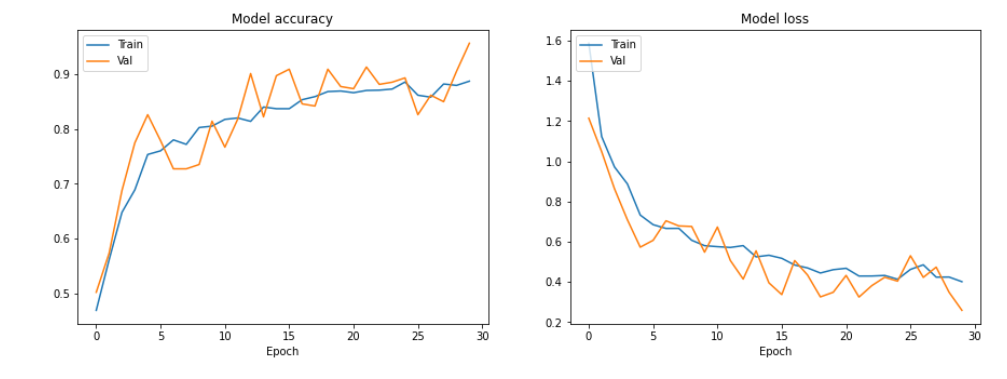
Tenth attempt:

In this attempt we stayed with same configuration, and additional to the L2 Regularization layer we add a drop-out layer of 0.2 which was our best attempt with drop-out.

Architecture of a model:



Last & Best Epoch:  


  
Evaluate:



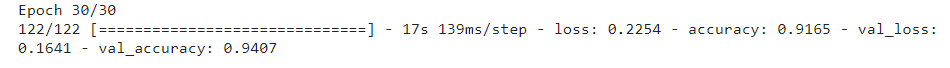
Conclusion:

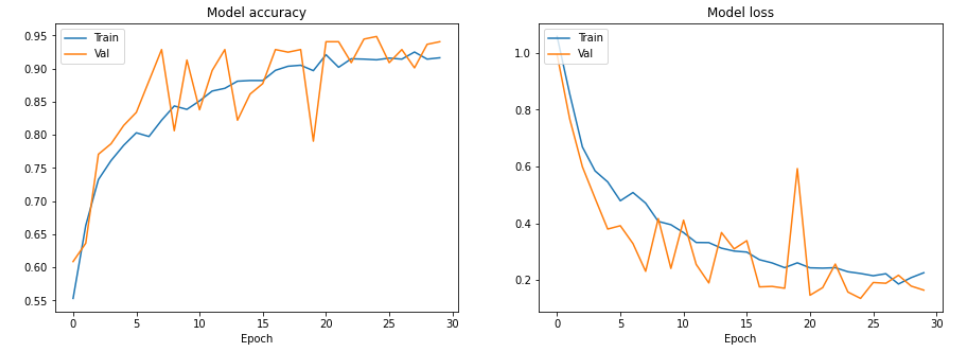
With L2 Regularization layer and drop-out of 0.2 layer we got nice result.

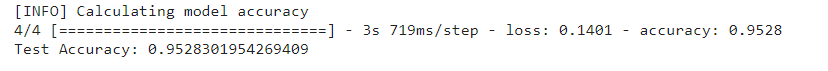
Eleventh attempt:

In this attempt we changed our architecture to the original architecture with change of the batch size to 16.

Last Epoch:



Best Epoch:  
  


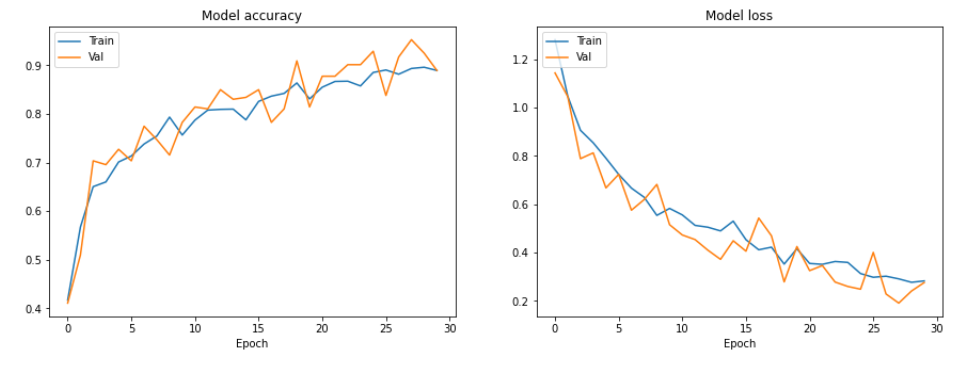
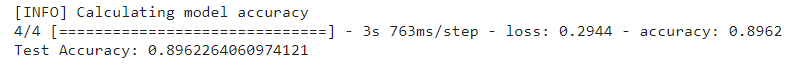
Evaluate:  
  
Conclusion:

Got very good result but still not our best result.

Twelfth attempt:

In this attempt we changed our architecture to the original architecture with change of the batch size to 64.

Last Epoch:

  
Best Epoch:  
  
  
Evaluate:  
  
Conclusion:  
Got nice result on the test but still not the best.

Thirteen attempt :

In this attempt we stayed with the same architecture and changed the batch size to 32 but and changed our strides to None.

Last & Best Epoch:

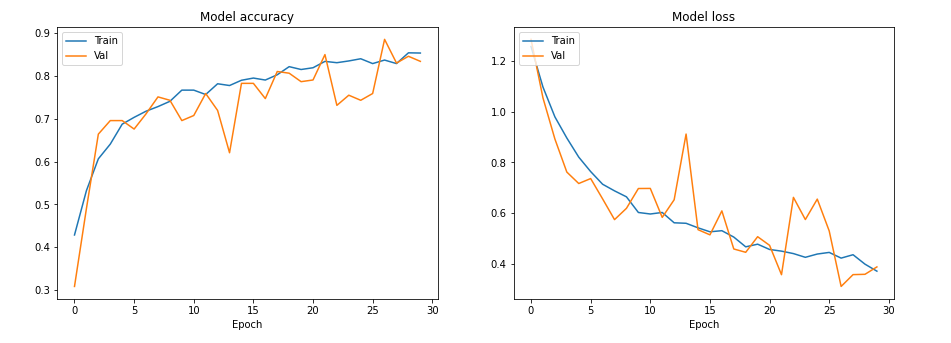
  
Chart, line chart

Description automatically generated  
Evaluate:  
Graphical user interface, Word

Description automatically generated with medium confidence  
Conclusion:  
Got Pretty good result on the test but still not the best.

Fourteen attempt:

In this attempt we stayed with the same architecture and changed the strides to 4.

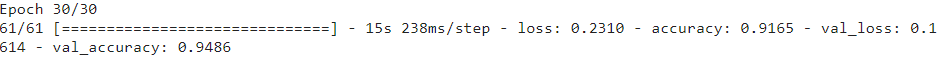
Last Epoch:  
  
Best Epoch:  
  
  
Evaluate:  
Logo

Description automatically generated  
Conclusion:  
we got very bad result

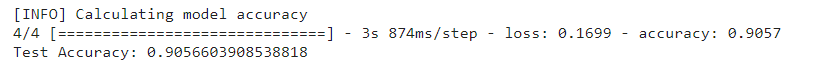
Fifteen attempt:

In this attempt we changed the activation in the Convolution layer to LeakyReLU(alpha=0.1) and down the strides back to 2 .

Last Epoch:

  
Best Epoch:  
  
Chart, line chart

Description automatically generated

Evaluate:  
  
Conclusion:

Got Pretty good result on the test but still not the Best.

Sixteen attempt:

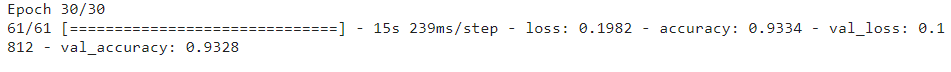
In this attempt we add another Convolution layer with the same configuration.

Architecture of a model:

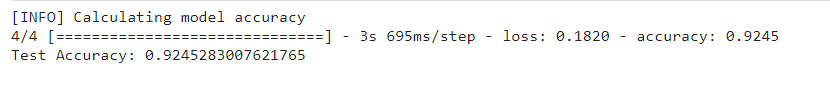
Table

Description automatically generated

Last & Best Epoch:



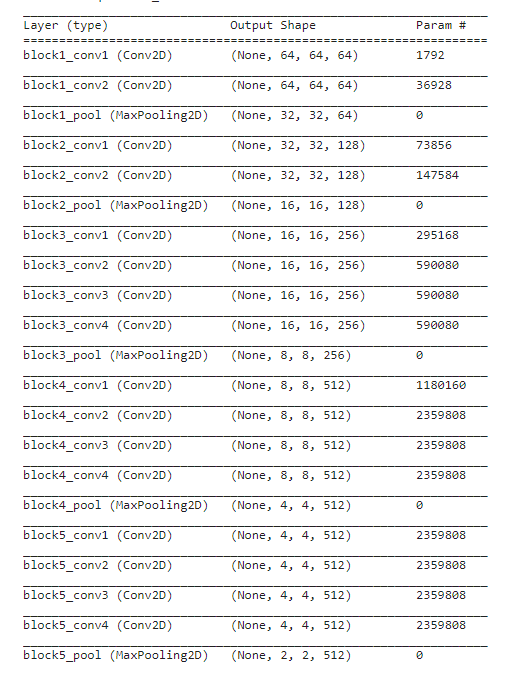
Chart, line chart

Description automatically generated  
Evaluate:  
  
Conclusion:  
Got Pretty good result on the test but still not the best, best so we will try another approach in the next attempts and will move to Transfer learning approach.

Seventeen attempt: (saved version)

In this attempt we applied the **VGG-19** model with the weights of IMAGENAET and the optimizer was SGD(lr=0.001, momentum=0.9).(epoch=12)

Architecture of a model:

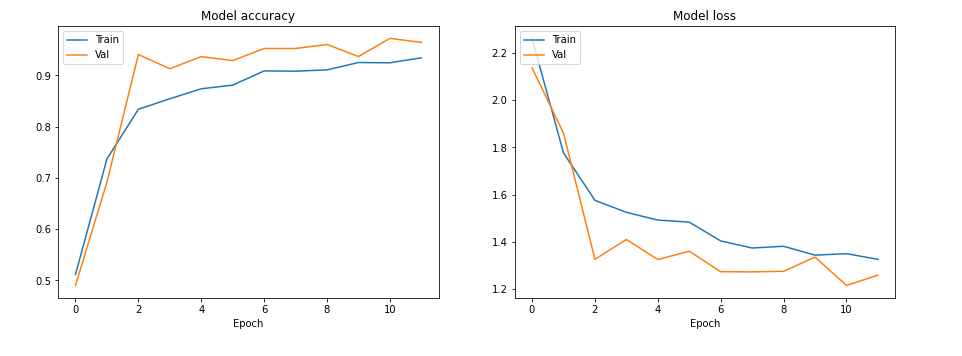


Last Epoch:



Best Epoch:





Evaluate:

A picture containing text

Description automatically generated

Conclusion:

On the **Test** we got better results than the validation 🡪 0.99

Eighteenth attempt:

In this attempt we applied the **VGG-16** model with the weights of IMAGENAET and the optimizer was SGD(lr=0.001, momentum=0.9). (epoch=12)

Architecture of a model:

Table

Description automatically generated

Last Epoch:

  
Best Epoch:

  
Chart, line chart

Description automatically generated  
Evaluate:

A picture containing text

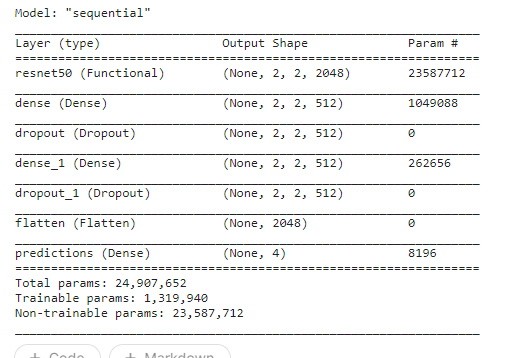
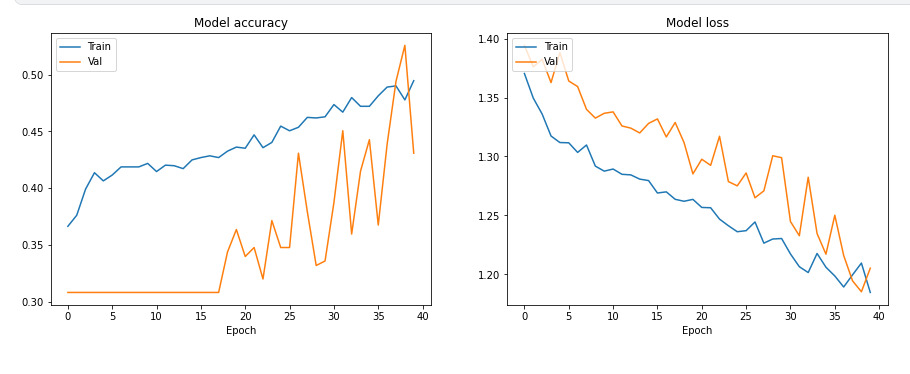
Description automatically generated  
Conclusion:

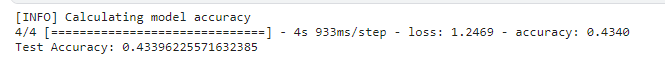
On the **Test** we got🡪 0.99.

Nineteen attempt:

In this attempt we applied the **ResNet50** model with the weights of IMAGENAET and the optimizer was SGD(lr=0.001, momentum=0.9). (epoch=40) (batch size=64)

Architecture of a model:

  
Last Epoch:  
  
Best Epoch:  
  
  
Evaluate:



Conclusion:

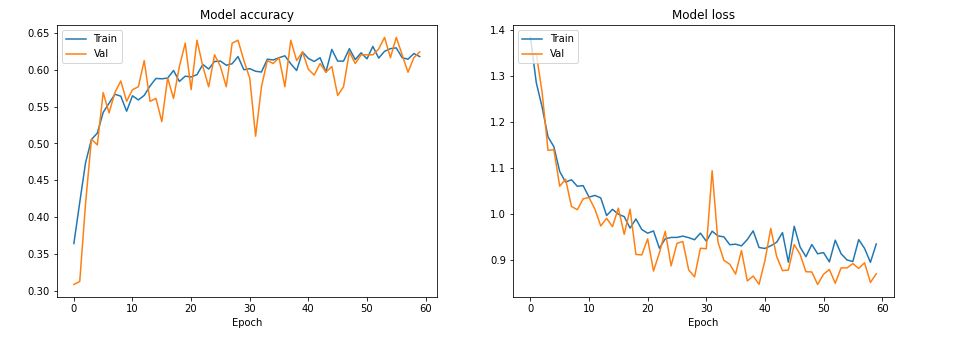
On the **Test** we got very bad result🡪 0.43, we can see that the validation accuracy jump up and down maybe it's time to change optimizer.

Twenty attempt :

In this attempt we applied the **ResNet50** model with the weights of IMAGENAET and the optimizer was Adam with default values same Architecture as before. (epoch=60) (batch size=32)

Last Epoch:

  
Best Epoch:

Evaluate:

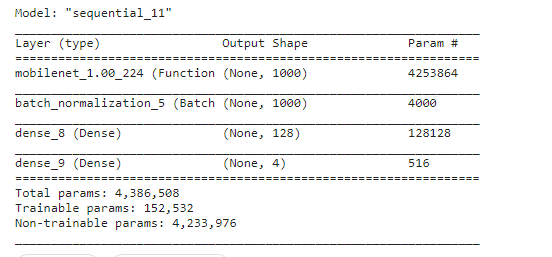
  
Conclusion:

On the **Test** we got very bad result🡪 0.61, Still the validation accuracy jump up and down got slightly better results but still it seems that it is not right model for this task.

Twenty-one attempt :

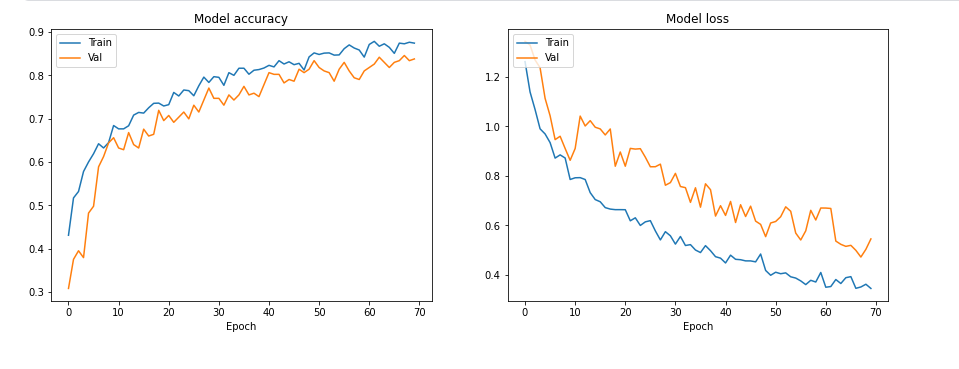
In this attempt we applied the **MobileNet** model with the weights of IMAGENAET and the optimizer was Adam with default values. (epoch=70) (batch size=32)

Architecture of a model:



Last Epoch:

  
Best Epoch:

Evaluate:

Word

Description automatically generated with medium confidence

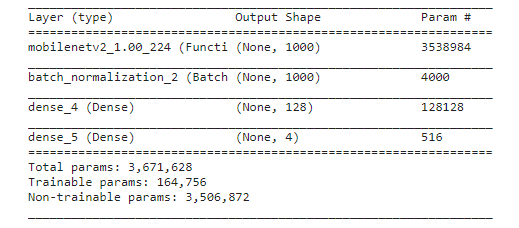
Conclusion:

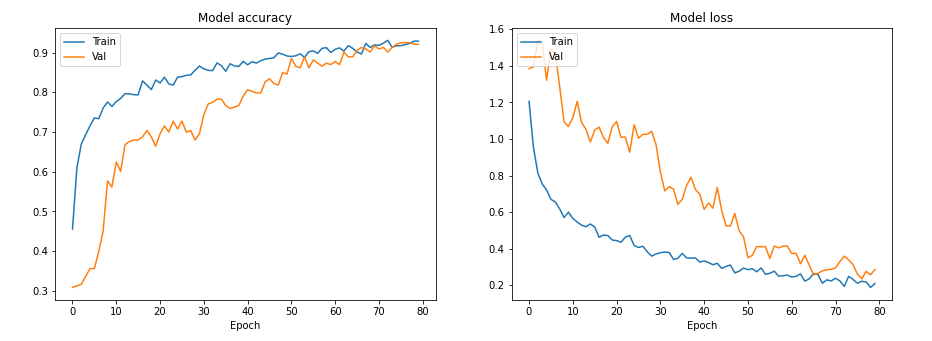
On the **Test** we got good result🡪 0.84

Twenty-two attempt :

In this attempt we applied the **MobileNetV2** model with the weights of IMAGENAET and the optimizer was Adam with default values. (epoch=80) (batch size=32)

Architecture of a model:

  
Last & Best Epoch:

  
  
Evaluate:

Logo

Description automatically generated  
Conclusion:

On the **Test** we got excellent result🡪 0.94, this model it is a version 2 of mobilenet so we can see that after properly fit the model we can use it on mobile applications

**סיכום**:

לסיכום ניתן לראות שגם המודלים הפשוט ביותר בעל 2 שכבות קונבולוציה נותן לנו תוצאות של מעל 95 אחוז גם בבדיקה וגם בולידציה.

אך המודל בעל התוצאות הטובות ביותר הינו VGG-16/19 בעל אופטימיזר מסוג SGD עם עקומת למידה של 0.001 ובעל מומנטום של 0.9 על מנת להתגבר על בעיית המינימום המקומי.

מודל זה הינו המודל אשר נתן את תוצאות הטובות ביותר של 99% על סט האימון אמנם סט האימון קטן אך התוצאות תמיד נעו בטווח של 0.97 ל0.99 כלומר שהמודל בצורה נותן תוצאות אמינות בצורה עקבית .

זהו מודל אשר עבר FINE-TUNE ונעשה שימוש במשקולות של . IMAGENET

**מקורות**:

<https://www.tensorflow.org/>  
<https://openai.com/blog/chatgpt/>  
<https://datascience.stackexchange.com/>

<https://towardsdatascience.com/transfer-learning-with-vgg16-and-keras-50ea161580b4>

<https://medium.com/@godeep48/an-overview-on-mobilenet-an-efficient-mobile-vision-cnn-f301141db94d>  
  
<https://stackoverflow.com/questions/4195453/how-to-resize-an-image-with-opencv2-0-and-python2-6>

<https://towardsdatascience.com/deep-learning-using-transfer-learning-python-code-for-resnet50-8acdfb3a2d38>

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| הערות | Epochs | Dense | Model | Batch Size | Changed | Added | Validation Accuracy | Validation Loss | Model Name |
| ~1 minutes | 10 | 128 | Conv+pool \*2 | 32 | - | - | 0.832 | 0.4829 | Base |
| ~3 minutes | 30 | 128 | Conv+pool \*2 | 32 | 30 epochs | data augmentation | 0.9526 | 0.1773 | Base -data augmentation 30 epochs |
| ~7.5minutes | 30 | 128 | Conv+pool \*2 | 32 | - | Dropout(0.5) | 0.8656 | 0.3070 | Dropout(0.5) |
| ~8 minutes ~3 minutes | 30 | 128 | Conv+pool \*2 | 32 | Dropout(0.2) | - | 0.9605 | 0.1546 | Dropout(0.2) |
| ~7.5 minutes ~3 minutes | 30 | 128 | Conv+pool \*2 | 32 | Dropout(0.8) | - | 0.8024 | 0.3865 | Dropout(0.8) |
| ~7.5 minutes ~3 minutes | 30 | 128 | Conv+pool \*2 | 32 | regularization | L1(0.005) | 0.8814 | 0.4675 | L1 |
| ~7.2 minutes ~3 minutes | 30 | 128 | Conv+pool \*2 | 32 | Regularization | L2(0.005) | 0.8577 | 0.3872 | L2 |
| ~7.5 minutes ~3 minutes | 30 | 128 | Conv+pool \*2 | 32 | Regularization, Dropout(0.2) | dropout(0.2) | 0.9565 | 0.2588 | L2(0.005)+dropout(0.2) |
| ~8.5minutes ~3 minutes | 30 | 128 | Conv+pool \*2 | 16 | batch | - | 0.9407 | 0.1614 | Base-Batch 16 |
| ~6.5 minutes ~3 minutes | 30 | 128 | Conv+pool \*2 | 64 | batch | - | 0.8893 | 0.2760 | Base-Batch 16 |
| ~7.5 minutes ~3 minutes | 30 | 128 | Conv+pool \*3 | 32 | Architecture | Convolution layer | 0.9328 | 0.1812 | Base + new convolution layer |
| ~3.5  minutes ~3 minutes | 12 | 512 | VGG19 | 32 | Architecture | - | 0.9723 | 1.2144 | VGG19 |
| ~3 minutes ~3 minutes | 12 | 512 | VGG16 | 32 | Architecture | - | 0.9802 | 1.2050 | VGG16 |
| ~10 minutes ~3 minutes | 40 | 2048 | Resnet50 | 64 | Architecture | - | 0.4308 | 1.2053 | Resnet50 |
| ~16 minutes ~3 minutes | 60 | 2048 | Resnet50 | 64 | epochs | - | 0.6245 | 0.8698 | Resnet50+epochs |
| ~17.5minutes ~3 minutes | 70 | 128 | MobileNetV1 | 32 | Architecture | - | 0.8379 | 0.5456 | MobileNetV1 |
| ~19minutes ~3 minutes | 80 | 128 | MobileNetV2 | 32 | Architecture | - | 0.9209 | 0.2857 | MobileNetV2 |