Documentatie

1. <u>Naive Bayse</u>

Prima incercare, nu prea eficienta, a fost cu (Gaussian) Naive Bayse oferit de libraria *sklearn.* Nu am avut ce hiperparametrii sa modific la el, iar imaginile nu au fost prelucrate in niciun fel, doar rescrise intr-o dimensiune pentru a se potrivi cu inputul dorit de model. Iata rezultatele:

```
class     precision    recall  f1-score   support

    0       0.00       0.00       0.00        13
    1       0.20       0.15       0.17        13
    2       0.55       0.43       0.48        14
    3       0.42       0.77       0.54        13
    4       0.33       0.09       0.14        11
    5       0.64       0.64       0.64        11
    6       0.46       0.67       0.55         9
    7       0.17       0.20       0.18        10
    8       0.50       0.15       0.24        13
    9       0.44       0.57       0.50         7
   10        0.29       0.36       0.32        11
   11        0.38       0.25       0.30        12
   12        0.60       0.50       0.55        12
   13        0.20       0.60       0.30         5
   14        0.12       0.17       0.14        12
   15        0.00       0.00       0.00        14
   16        0.09       0.10       0.10        10
   17        0.27       0.67       0.38         6
   18        0.22       0.60       0.32        15
   19        0.17       0.31       0.22        13
   20        0.67       0.13       0.22        15
   21        0.57       0.33       0.42        12
   22        0.63       0.92       0.75        13
   23        0.43       0.60       0.50        10
   24        0.08       0.09       0.09        11
   25        0.00       0.00       0.00        10
   26        0.00       0.00       0.00        14
   27        0.67       0.22       0.33        36
   28        0.65       0.79       0.71        14
   29        0.25       0.29       0.27         7
   30        0.22       0.50       0.31         4
```

31	0.14	0.56	0.23	9
32	0.09	0.14	0.11	7
33	0.50	0.18	0.27	11
34	0.18	0.33	0.24	6
35	0.14	0.22	0.17	9
36	1.00	0.11	0.20	9
37	0.00	0.00	0.00	8
38	0.50	0.23	0.31	22
39	0.40	0.22	0.29	9
40	0.33	0.11	0.17	9
41	0.20	0.08	0.11	13
42	0.00	0.00	0.00	6
43	1.00	0.38	0.55	8
44	0.25	0.36	0.30	11
45	0.20	0.25	0.22	8
46	0.00	0.00	0.00	12
47	0.33	0.33	0.33	9
48	0.43	0.27	0.33	11
49	0.22	0.14	0.17	14
50	0.00	0.00	0.00	10
51	0.33	0.25	0.29	4
52	0.11	0.43	0.17	7
53	0.09	0.17	0.11	12
54	0.00	0.00	0.00	8
55	0.20	0.08	0.12	12
56	0.20	0.22	0.21	9
57	0.27	0.43	0.33	7
58	0.22	0.31	0.26	13
59	0.36	0.62	0.45	8
60	0.50	0.31	0.38	13
61	0.18	0.18	0.18	11
62	0.14	0.12	0.13	8
63	0.00	0.00	0.00	6
64	0.14	0.10	0.12	10
65	0.00	0.00	0.00	6
66	0.18	0.25	0.21	8
67	0.22	0.33	0.27	6
68	0.00	0.00	0.00	7
69	0.43	0.21	0.29	14
70	0.56	0.71	0.63	7
71	0.80	0.67	0.73	12
72	1.00	0.20	0.33	5
73	0.23	0.21	0.22	14
74	0.13	0.14	0.14	14

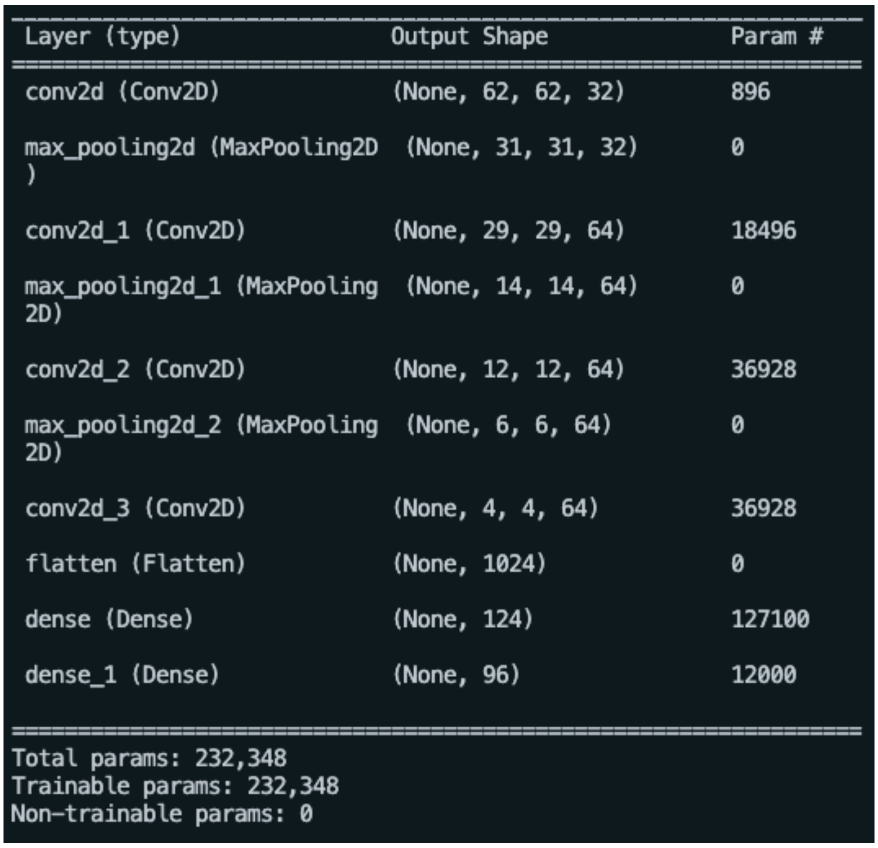
```
          75       0.22      0.33      0.27        12
          76       0.33      0.56      0.42         9
          77       0.00      0.00      0.00         8
          78       1.00      0.30      0.46        10
          79       0.25      0.14      0.18         7
          80       0.00      0.00      0.00        11
          81       0.67      0.40      0.50        15
          82       0.00      0.00      0.00         8
          83       0.14      0.11      0.12         9
          84       0.18      0.27      0.21        11
          85       0.30      0.55      0.39        11
          86       0.40      0.25      0.31         8
          87       0.50      0.42      0.45        12
          88       0.10      0.08      0.09        12
          89       0.27      0.50      0.35         8
          90       0.33      0.14      0.20         7
          91       0.00      0.00      0.00        11
          92       0.14      0.14      0.14         7
          93       0.37      0.47      0.41        15
          94       0.25      0.09      0.13        11
          95       0.20      0.17      0.18         6

    accuracy                           0.27
```

2. Convolutional Neural Network (CNN)

Pentru acest model, am folosit libraria TensorFlow, deoarece sklearn nu avea CNN.

Aici am incercat mai multe strategii. Pentru prima incercare am prelucrat pozele putin, impartind pixelii la 255 ca sa am doar valori curinse intre 0 si 1 pentru a facilita invatarea modelului:

Pentru arhitectura modelului am inceput cu ceva simplu: 4 straturi de convolutie cu window de (3, 3) si 64 de filtre(exceptie primul cu 32), fiecare urmat de un maxPool de (2, 2), dupa *flatten* avand un strat cu activare relu urmat de un strat cu activare softmax, pentru a obtine probabiliatile claselor:



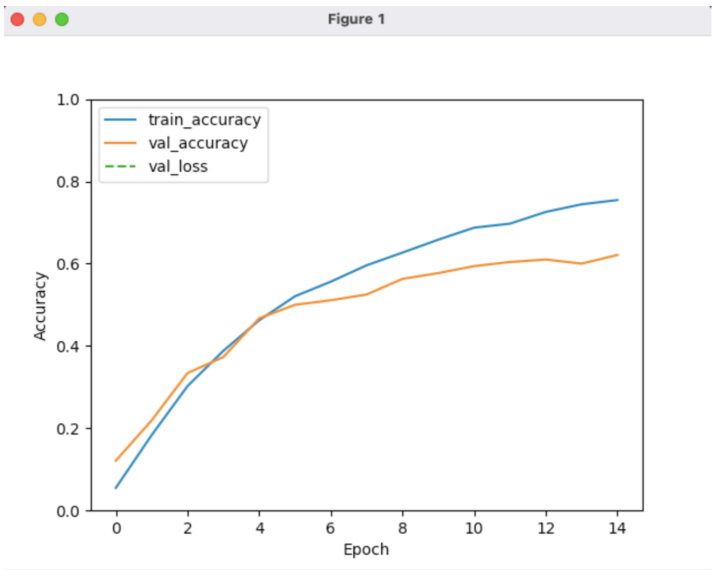
```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_3 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 124)               127100

 dense_1 (Dense)             (None, 96)                12000

=================================================================
Total params: 232,348
Trainable params: 232,348
Non-trainable params: 0
```

Functia de optimizare folosita initial a fost Adam, iar hiperparametrii modificati au fost doar numarul de epoci si batch size:15 epoci cu batch size 150. Iata rezultatele:

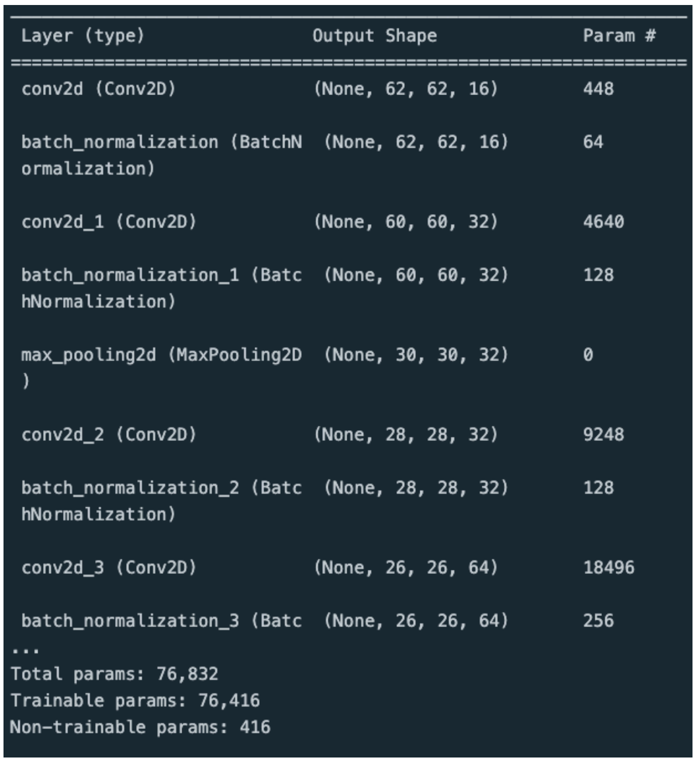
class	precision	recall	f1-score	support
0	0.33	0.38	0.36	13
1	0.38	0.23	0.29	13
2	0.91	0.71	0.80	14
3	0.80	0.92	0.86	13
4	0.50	0.18	0.27	11
5	0.35	0.55	0.43	11
6	0.60	0.33	0.43	9
7	0.40	0.40	0.40	10
8	0.55	0.46	0.50	13
9	0.44	1.00	0.61	7
10	0.55	0.55	0.55	11
11	0.30	0.25	0.27	12
12	0.75	0.75	0.75	12
13	0.33	0.20	0.25	5
14	0.60	0.75	0.67	12
15	0.60	0.64	0.62	14
16	0.50	0.80	0.62	10
17	0.75	1.00	0.86	6
18	0.59	0.67	0.62	15
19	0.82	0.69	0.75	13

20	0.82	0.60	0.69	15
21	0.92	0.92	0.92	12
22	0.76	1.00	0.87	13
23	0.90	0.90	0.90	10
24	1.00	0.18	0.31	11
25	0.62	0.50	0.56	10
26	0.60	0.21	0.32	14
27	0.68	0.69	0.68	36
28	0.79	0.79	0.79	14
29	0.08	0.14	0.10	7
30	0.50	0.75	0.60	4
31	0.67	0.67	0.67	9
32	0.50	0.14	0.22	7
33	0.58	0.64	0.61	11
34	1.00	0.50	0.67	6
35	0.00	0.00	0.00	9
36	0.42	0.56	0.48	9
37	0.20	0.12	0.15	8
38	0.32	0.36	0.34	22
39	0.17	0.11	0.13	9
40	0.42	0.56	0.48	9
41	0.70	0.54	0.61	13
42	0.80	0.67	0.73	6
43	0.50	0.62	0.56	8
44	0.62	0.91	0.74	11
45	0.64	0.88	0.74	8
46	1.00	0.58	0.74	12
47	0.80	0.89	0.84	9
48	0.71	0.45	0.56	11
49	0.50	0.50	0.50	14
50	0.50	0.30	0.37	10
51	1.00	0.50	0.67	4
52	0.33	0.43	0.38	7
53	0.54	0.58	0.56	12
54	1.00	0.75	0.86	8
55	0.73	0.67	0.70	12
56	0.67	0.44	0.53	9
57	0.40	0.86	0.55	7
58	0.60	0.69	0.64	13
59	0.83	0.62	0.71	8
60	0.30	0.23	0.26	13
61	0.50	0.27	0.35	11
62	0.62	0.62	0.62	8
63	0.31	0.67	0.42	6

```
      64       0.70      0.70      0.70        10
      65       0.50      0.67      0.57         6
      66       0.46      0.75      0.57         8
      67       0.33      0.67      0.44         6
      68       0.45      0.71      0.56         7
      69       0.87      0.93      0.90        14
      70       0.78      1.00      0.88         7
      71       0.90      0.75      0.82        12
      72       0.36      0.80      0.50         5
      73       0.57      0.86      0.69        14
      74       0.90      0.64      0.75        14
      75       0.44      0.58      0.50        12
      76       0.86      0.67      0.75         9
      77       0.54      0.88      0.67         8
      78       0.89      0.80      0.84        10
      79       0.75      0.43      0.55         7
      80       0.60      0.27      0.37        11
      81       0.70      0.47      0.56        15
      82       0.00      0.00      0.00         8
      83       0.60      0.67      0.63         9
      84       0.55      0.55      0.55        11
      85       0.89      0.73      0.80        11
      86       0.57      0.50      0.53         8
      87       0.67      0.33      0.44        12
      88       0.70      0.58      0.64        12
      89       0.58      0.88      0.70         8
      90       0.27      0.43      0.33         7
      91       0.35      0.73      0.47        11
      92       0.30      0.43      0.35         7
      93       0.69      0.73      0.71        15
      94       1.00      0.82      0.90        11
      95       0.50      0.17      0.25         6

accuracy                        0.58
```



Am crescut acuratetea, dar se poate mai bine. De data aceasta am stanadardizat pozele, scazand media din fiecare poza si impartind la deviatia standard, pastrand valorile in intervalul [0, 1].



```
Layer (type)                 Output Shape              Param #
================================================================
conv2d (Conv2D)              (None, 62, 62, 16)        448

batch_normalization (BatchN  (None, 62, 62, 16)        64
ormalization)

conv2d_1 (Conv2D)            (None, 60, 60, 32)        4640

batch_normalization_1 (Batc  (None, 60, 60, 32)        128
hNormalization)

max_pooling2d (MaxPooling2D  (None, 30, 30, 32)        0
)

conv2d_2 (Conv2D)            (None, 28, 28, 32)        9248

batch_normalization_2 (Batc  (None, 28, 28, 32)        128
hNormalization)

conv2d_3 (Conv2D)            (None, 26, 26, 64)        18496

batch_normalization_3 (Batc  (None, 26, 26, 64)        256
...
Total params: 76,832
Trainable params: 76,416
Non-trainable params: 416
```

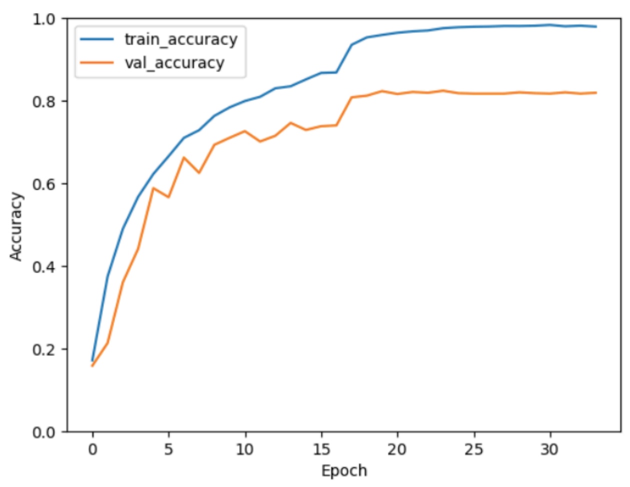
Referitor la arhitectura modelului, aceasta a suferit mai multe schimbari, printre care:
- am mai adaugat un strat de convolutie
- am lasat maxPool doar pe al doilea strat si am adaugat averagePool pe ultimul strat
- am adaugat batch normalization pe fiecare layer
- am variat numarul de filtre: 16-32-32-64-64
- am eliminat primul layer de dupa *flatten*

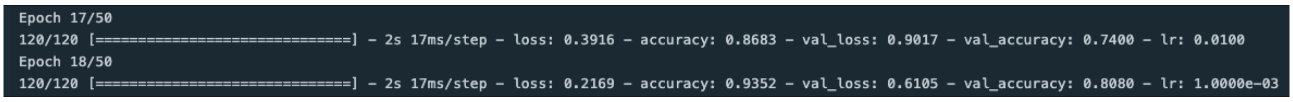
Scopul acestei variatii a fost in principal sa reduc numarul de parametrii pe care modelul va trebui sa ii modifice in procesul de antrenare, si am reusit sa obtin ~76k, de la ~230k

Hiperparametrii au fost si ei modificati dupa cum urmeaza:
- am marit learning rate la 0.01, default fiind 0.001
- am marit numarul de epoci la 50, dar am adaugat si o functie de early stopping in caz ca acuratetea pe validare nu se imbunatateste in 10 epoci
- am scazut batch size la 100
- si cel mai important, am adaugat o functie de reduce lr on plateau, care micsoreaza learning rate daca nu facem progrese dupa 3 epoci:

new_lr = 0.1 x old_lr

Programul s-a oprit dupa 34 de epoci, ajungand la o acuratete de 81%:

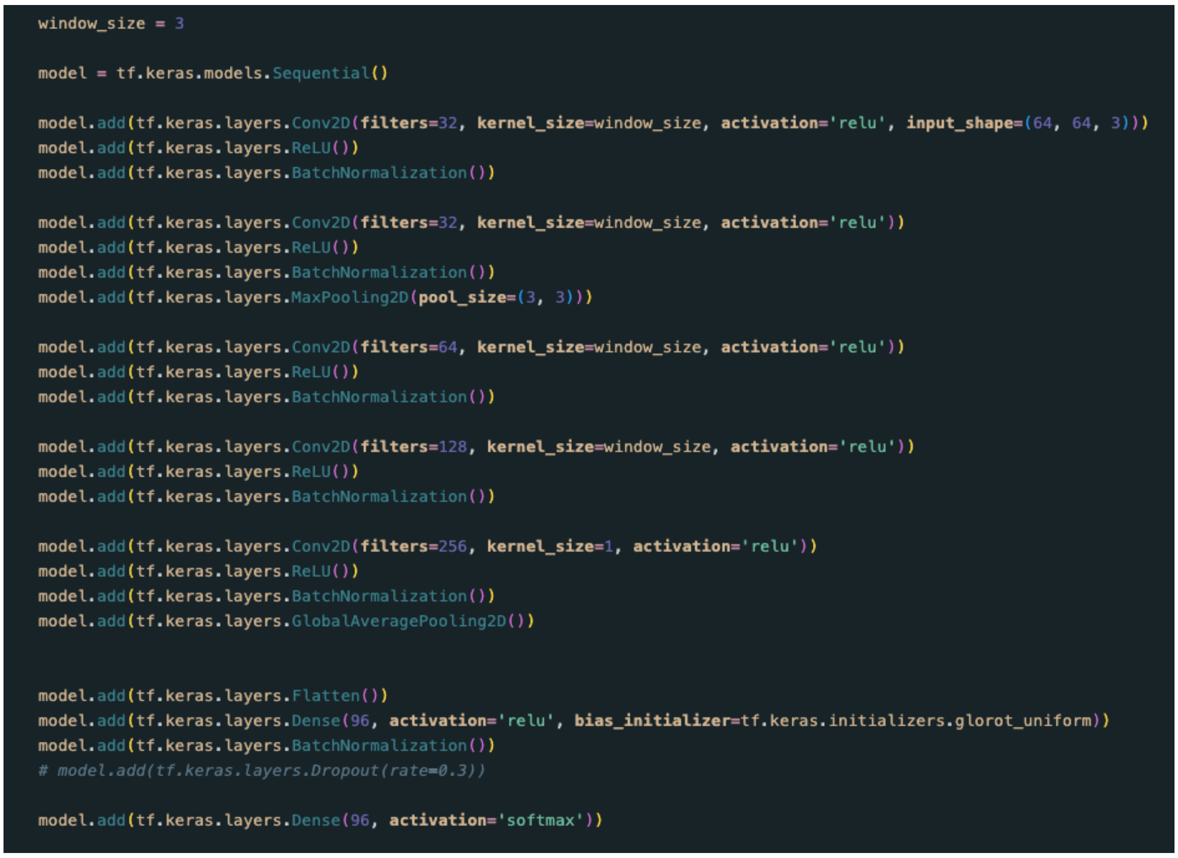




```
Epoch 17/50
120/120 [==============================] - 2s 17ms/step - loss: 0.3916 - accuracy: 0.8683 - val_loss: 0.9017 - val_accuracy: 0.7400 - lr: 0.0100
Epoch 18/50
120/120 [==============================] - 2s 17ms/step - loss: 0.2169 - accuracy: 0.9352 - val_loss: 0.6105 - val_accuracy: 0.8080 - lr: 1.0000e-03
```

Dupa cum se vede si in grafic, in momentul in care reducem learning rate prima data, obtinem o imbunatatire semnificativa cand vine vorba de acuratete. Totusi, learning rate va mai scadea de 4 ori, dar acuratetea pe validare nu se mai imbunatateste.

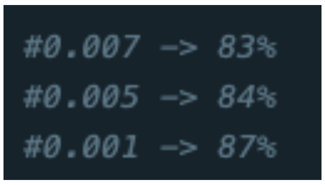
Avem accuracy de 81%; se poate oare mai bine? Da!



```python
window_size = 3

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=window_size, activation='relu', input_shape=(64, 64, 3)))
model.add(tf.keras.layers.ReLU())
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=window_size, activation='relu'))
model.add(tf.keras.layers.ReLU())
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(3, 3)))

model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=window_size, activation='relu'))
model.add(tf.keras.layers.ReLU())
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=window_size, activation='relu'))
model.add(tf.keras.layers.ReLU())
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=1, activation='relu'))
model.add(tf.keras.layers.ReLU())
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.GlobalAveragePooling2D())


model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(96, activation='relu', bias_initializer=tf.keras.initializers.glorot_uniform))
model.add(tf.keras.layers.BatchNormalization())
# model.add(tf.keras.layers.Dropout(rate=0.3))

model.add(tf.keras.layers.Dense(96, activation='softmax'))
```

Arhitectura de mai sus nu difera cu mult fata de cea precedenta, dar totusi este mai buna. Diferenta esentiala consta in numarul mai mare de filtre: 32-32-64-128-256. Totodata, am readus in joc primul strat dupa *flatten,* de data asta insotit si el de un batch normalization.

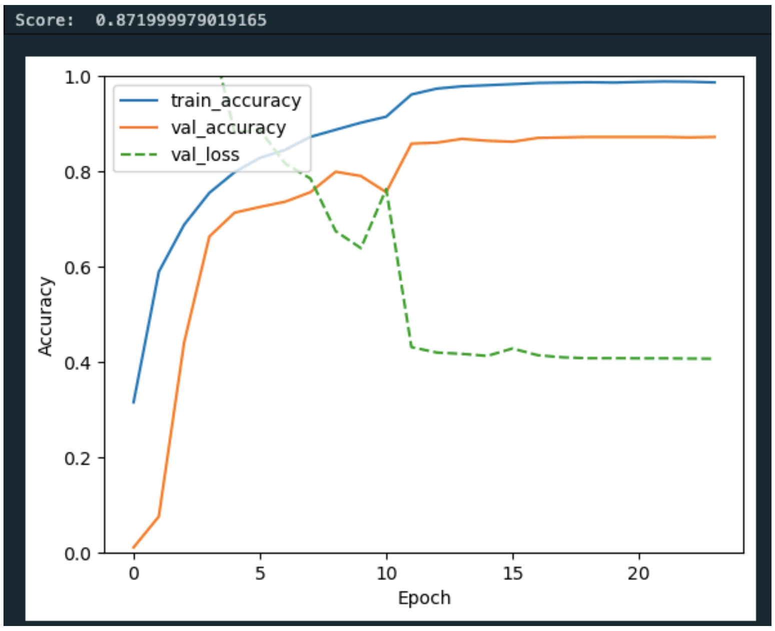
Legat de hiperparametrii, am rulat un mic ‘grid search’ manual pentru learning rate si am obtinut valorile din dreapta. Prin urmare am ales learning rate de 0.001 de data asta.



```
#0.007 -> 83%
#0.005 -> 84%
#0.001 -> 87%
```

Am redus si numarul de epoci la 80 si patiente pentru early stopping la 5 epoci in loc de 10.

Cu aceste schimbari am reusit sa ating o acuratete de 87%. Din nou, se poate observa saltul cauzat de micsorarea learning rate-ului, care ne duce intr-un overfit cinstit pe datele de antrenare.

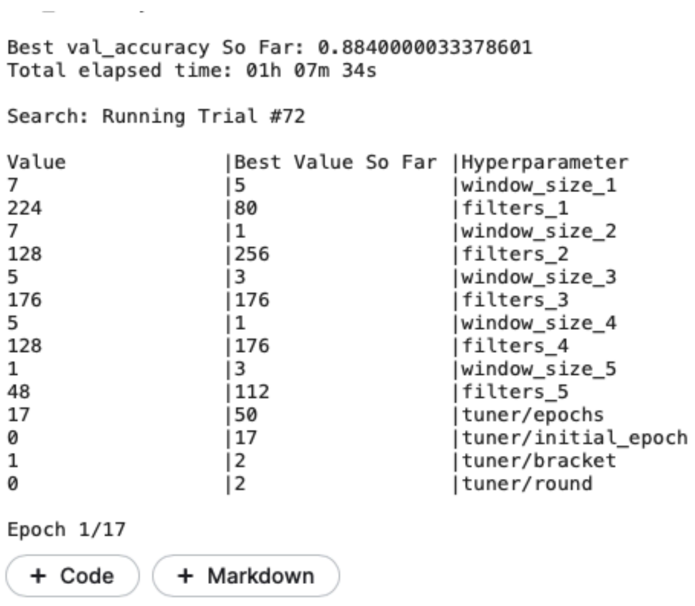


Urmatoarea varianta nu este cu mult mai buna, ba din contra, mi se pare putin mai haotica, dar cum fiecare procent reprezinta un progres, am pastrat si aceasta versiune.

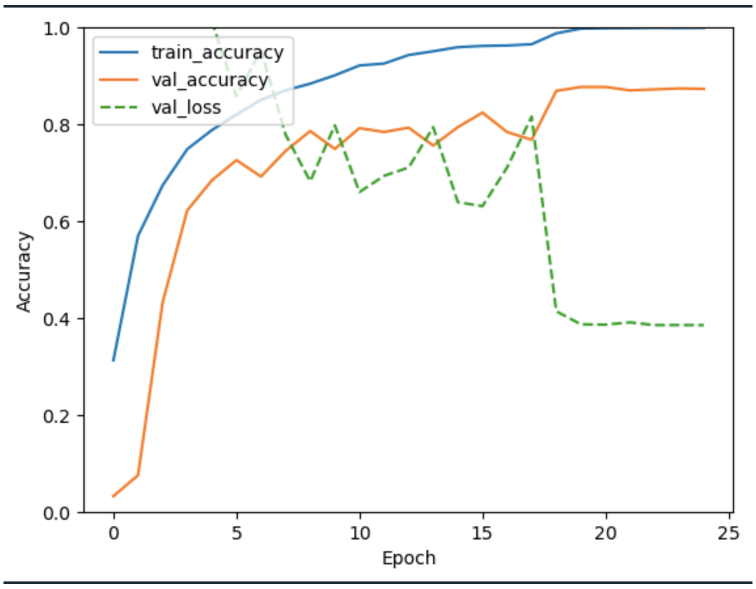
Am decis sa rulez un grid search cautand numerele magice pentru filtre/ window size, si am obtinut urmatoarele valori:

Cam ciudate daca ma intrebati pe mine, pentru ca stiam ca de regula numarul de filtre creste odata cu adancimea modelulu.

*hiperparametrii sunt la fel ca la versiunea precedenta

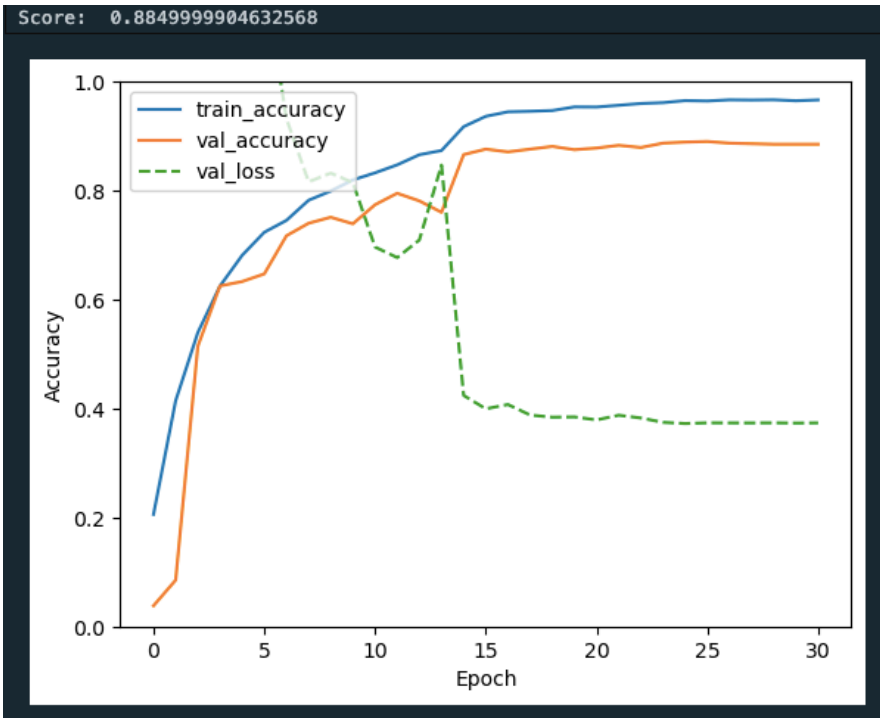


```
Best val_accuracy So Far: 0.8840000033378601
Total elapsed time: 01h 07m 34s

Search: Running Trial #72

Value             |Best Value So Far |Hyperparameter
7                 |5                 |window_size_1
224               |80                |filters_1
7                 |1                 |window_size_2
128               |256               |filters_2
5                 |3                 |window_size_3
176               |176               |filters_3
5                 |1                 |window_size_4
128               |176               |filters_4
1                 |3                 |window_size_5
48                |112               |filters_5
17                |50                |tuner/epochs
0                 |17                |tuner/initial_epoch
1                 |2                 |tuner/bracket
0                 |2                 |tuner/round

Epoch 1/17
```



Asa arata acum un grafic cu procesul de invatare al modelului. Putem observa ca a crescut overfitul pe datele de antrenare.

Prin urmare, in noua versiune a modelului, scopul a fost sa reduc acest overfit. Am redus learning rate la 0.0009 si am adaugat un dropout de 33% inainte ultimului layer al modelului. Rezultattul arata in felul urmator:



```
Score:  0.8849999904632568
```

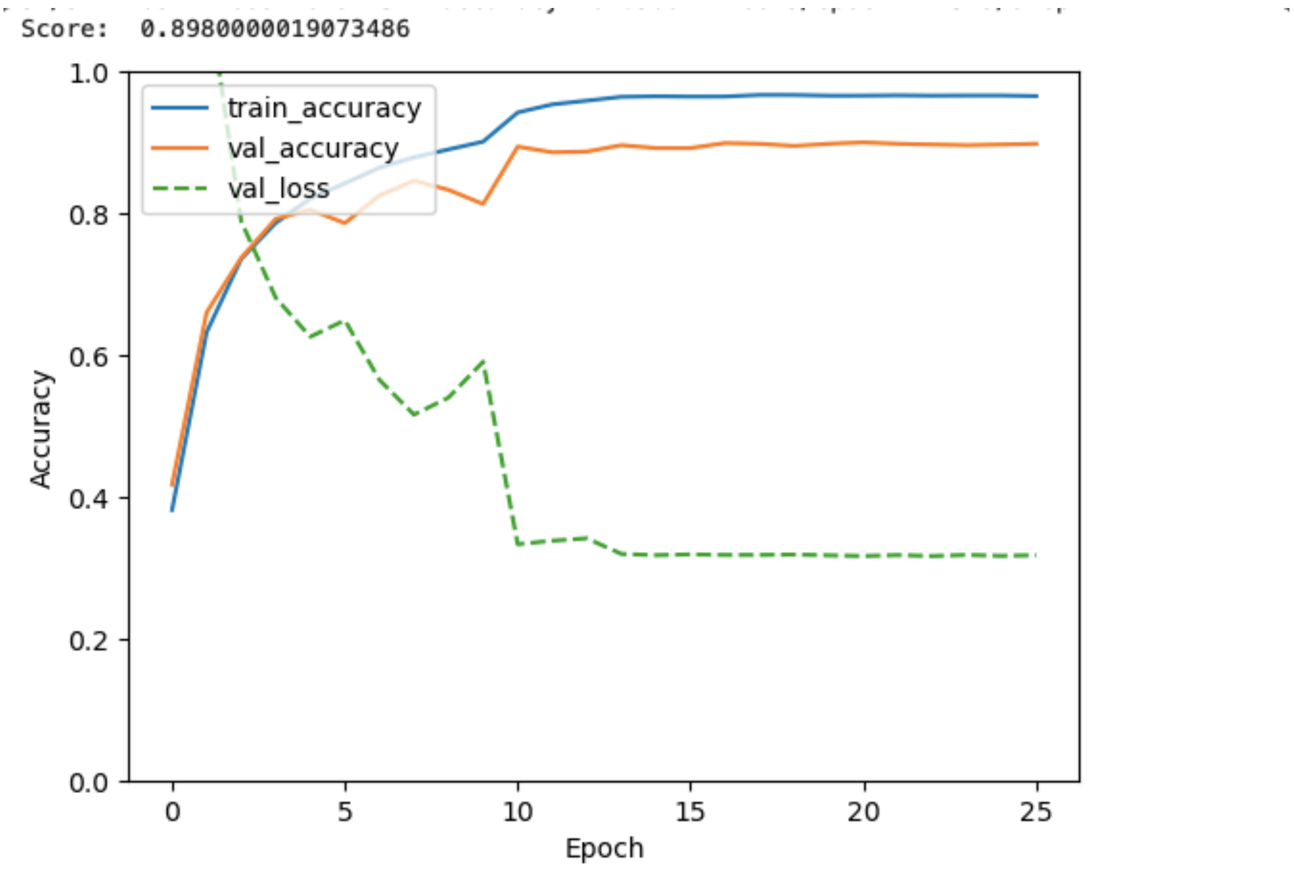
Nu numai ca am redus semnificativ overfitul, am reusit chiar sa imbunatatesc putin si acuratetea pe datele de validare cu 1 procent.

Am ajuns la 88%, dar inca mai aveam cateva ore de GPU pe kaggle asa ca am mai incercat sa imbunatatesc modelul. Doar ca de data asta nu am schimbat nimic la model :)

Ce am facut acum a fost sa mai prelucrez pozele ca pana acum erau doar standardizate. Acum am marit setul de date de antrenare la 36k poze dupa cum urmeaza:

- 12k pozele originale
- 12k pozele cu flip orizontal
- 12k pozele cu flip vertical

Cu aceasta prelucrare a pozelor, am ajuns la o acuratete de datele de validare de 89%:



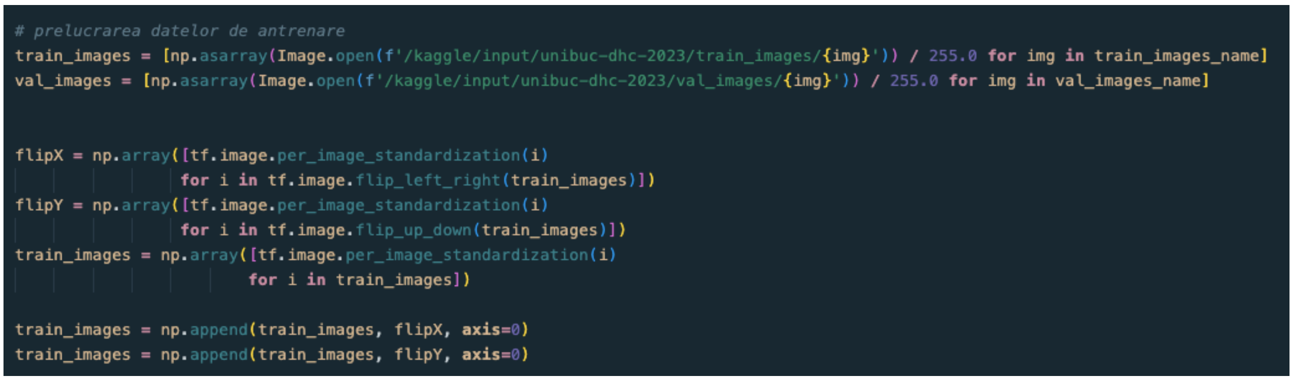
```
Score:  0.8980000019073486
```

Se poate observa si o mai buna invatare inca din primele epoci, pornind cu o acuratete de ~40% dupa prima epoca.

In final, am ajuns la ultima varianta a modelului CNN, care atinge 90% pe validare. Fiind un mix intre ultimele versiuni, iata un rezumat cu intreaga structura a modelului:

Prelucrarea Datelor:

Imaginile sunt intai standardizate, avand valori in intervalul [0, 1], iar apoi sunt inversate pe orizontala si verticala, ajungand la un set de date de antrenare de 36.000 de poze



```python
# prelucrarea datelor de antrenare
train_images = [np.asarray(Image.open(f'/kaggle/input/unibuc-dhc-2023/train_images/{img}')) / 255.0 for img in train_images_name]
val_images = [np.asarray(Image.open(f'/kaggle/input/unibuc-dhc-2023/val_images/{img}')) / 255.0 for img in val_images_name]


flipX = np.array([tf.image.per_image_standardization(i)
                    for i in tf.image.flip_left_right(train_images)])
flipY = np.array([tf.image.per_image_standardization(i)
                    for i in tf.image.flip_up_down(train_images)])
train_images = np.array([tf.image.per_image_standardization(i)
                          for i in train_images])

train_images = np.append(train_images, flipX, axis=0)
train_images = np.append(train_images, flipY, axis=0)
```

Arhitectura Modelului:

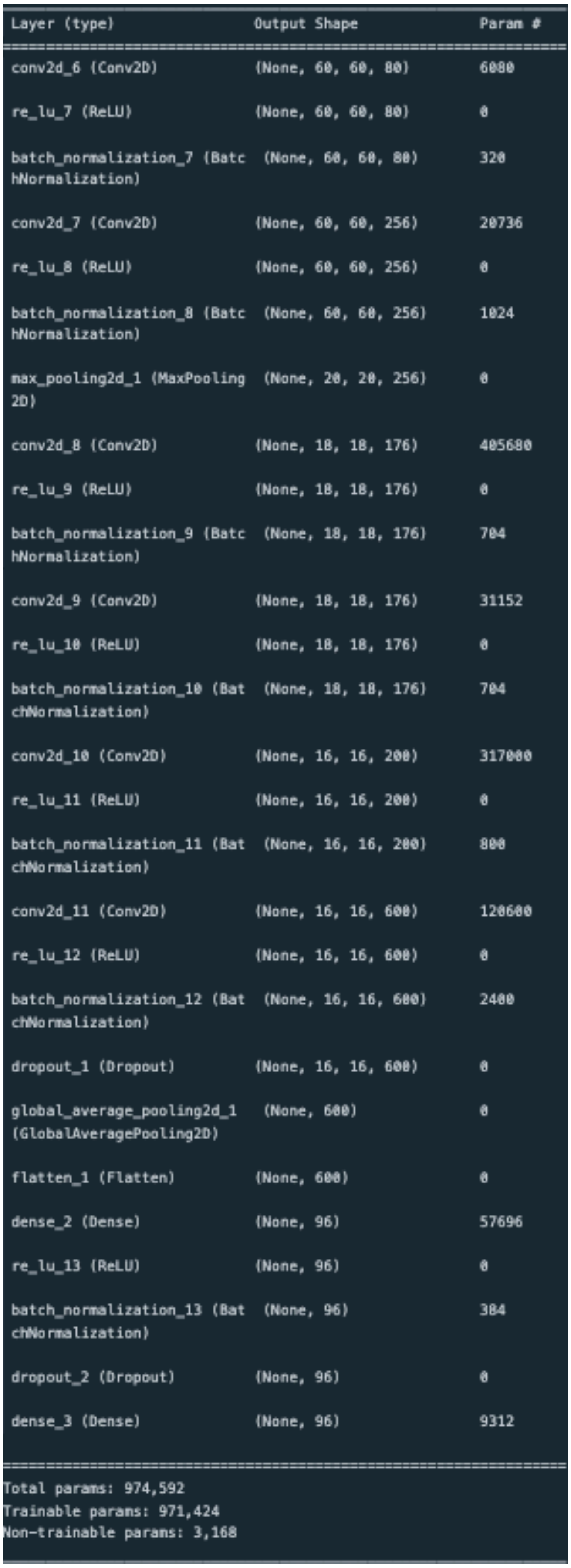
Avem un model cu 6 layere de convolutie, fiecare insotit de batch normalization, al doilea avand un max pool de (3, 3) iar ultimul un average pool si un dropout de 50%. Am crescut si numarul de filtre: 80-256-176-176-200-600.

Dupa *flatten* avem 2 straturi fully connected:

- primul are 96 de neuroni, o activare relu insotita de batch normalization si un dropout de 50%.

-al doilea si ultimul layer din model are tot 96 de neuroni si o activare softmax, pentru a returna probabilitatea fiecarei clase

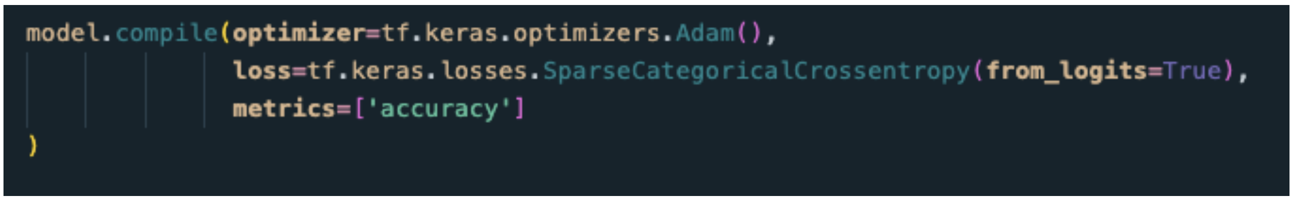
Astfel, am ajuns de la un model cu ~70k parametrii la unul cu aproape 1 milion de parametrii… Dar este si mai eficient, desi probabil se putea mai optim.



```
Layer (type)                Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)           (None, 60, 60, 80)        6080

re_lu_7 (ReLU)              (None, 60, 60, 80)        0

batch_normalization_7 (Batc  (None, 60, 60, 80)       320
hNormalization)

conv2d_7 (Conv2D)           (None, 60, 60, 256)       20736

re_lu_8 (ReLU)              (None, 60, 60, 256)       0

batch_normalization_8 (Batc  (None, 60, 60, 256)      1024
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 20, 20, 256)      0
2D)

conv2d_8 (Conv2D)           (None, 18, 18, 176)       405680

re_lu_9 (ReLU)              (None, 18, 18, 176)       0

batch_normalization_9 (Batc  (None, 18, 18, 176)      704
hNormalization)

conv2d_9 (Conv2D)           (None, 18, 18, 176)       31152

re_lu_10 (ReLU)             (None, 18, 18, 176)       0

batch_normalization_10 (Bat  (None, 18, 18, 176)      704
chNormalization)

conv2d_10 (Conv2D)          (None, 16, 16, 200)       317000

re_lu_11 (ReLU)             (None, 16, 16, 200)       0

batch_normalization_11 (Bat  (None, 16, 16, 200)      800
chNormalization)

conv2d_11 (Conv2D)          (None, 16, 16, 600)       120600

re_lu_12 (ReLU)             (None, 16, 16, 600)       0

batch_normalization_12 (Bat  (None, 16, 16, 600)      2400
chNormalization)

dropout_1 (Dropout)         (None, 16, 16, 600)       0

global_average_pooling2d_1   (None, 600)              0
(GlobalAveragePooling2D)

flatten_1 (Flatten)         (None, 600)               0

dense_2 (Dense)             (None, 96)                57696

re_lu_13 (ReLU)             (None, 96)                0

batch_normalization_13 (Bat  (None, 96)               384
chNormalization)

dropout_2 (Dropout)         (None, 96)                0

dense_3 (Dense)             (None, 96)                9312

=================================================================
Total params: 974,592
Trainable params: 971,424
Non-trainable params: 3,168
```

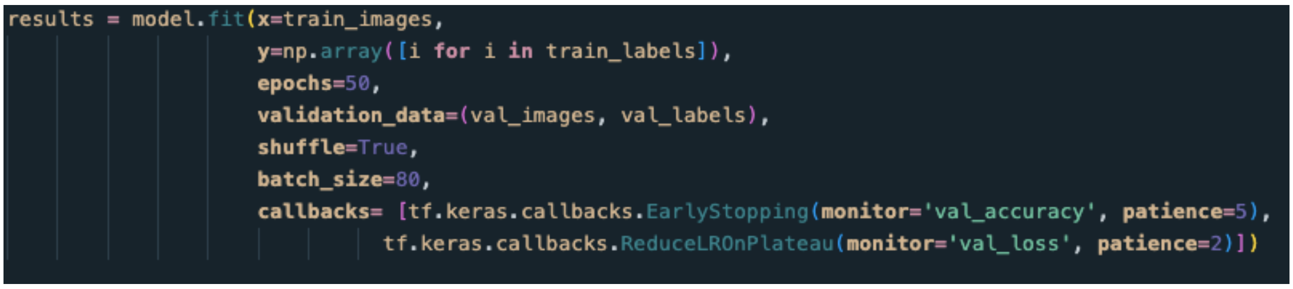
Hiperparametrii:

Modelul nostru foloseste optimizatorul Adam cu functia de loss Categorical Crossentropy. Learning rate initial este 0.0005, si scade daca validation loss nu se imbunatateste dupa 2 epoci. Noul lr va fi vechiul lr x 0.1.



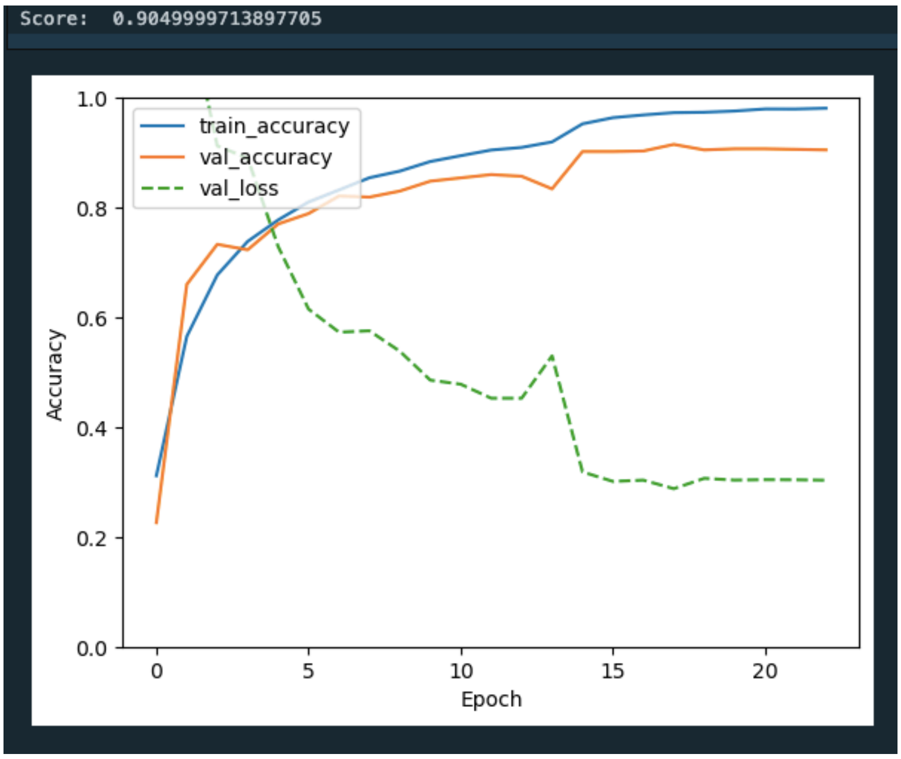
```
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy']
)
```

Antrenarea se desfasoara in maxim 50 de epoci, cu un early stopping daca nu se inbunatateste validation accuracy dupa 5 epoci. Batch size a ramas 80, iar imaginile au fost amestecate din nou pentru fiecare epoca.



```
results = model.fit(x=train_images,
                    y=np.array([i for i in train_labels]),
                    epochs=50,
                    validation_data=(val_images, val_labels),
                    shuffle=True,
                    batch_size=80,
                    callbacks= [tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5),
                              tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', patience=2)])
```

Rezultatul final:



```
Score:  0.9049999713897705
```

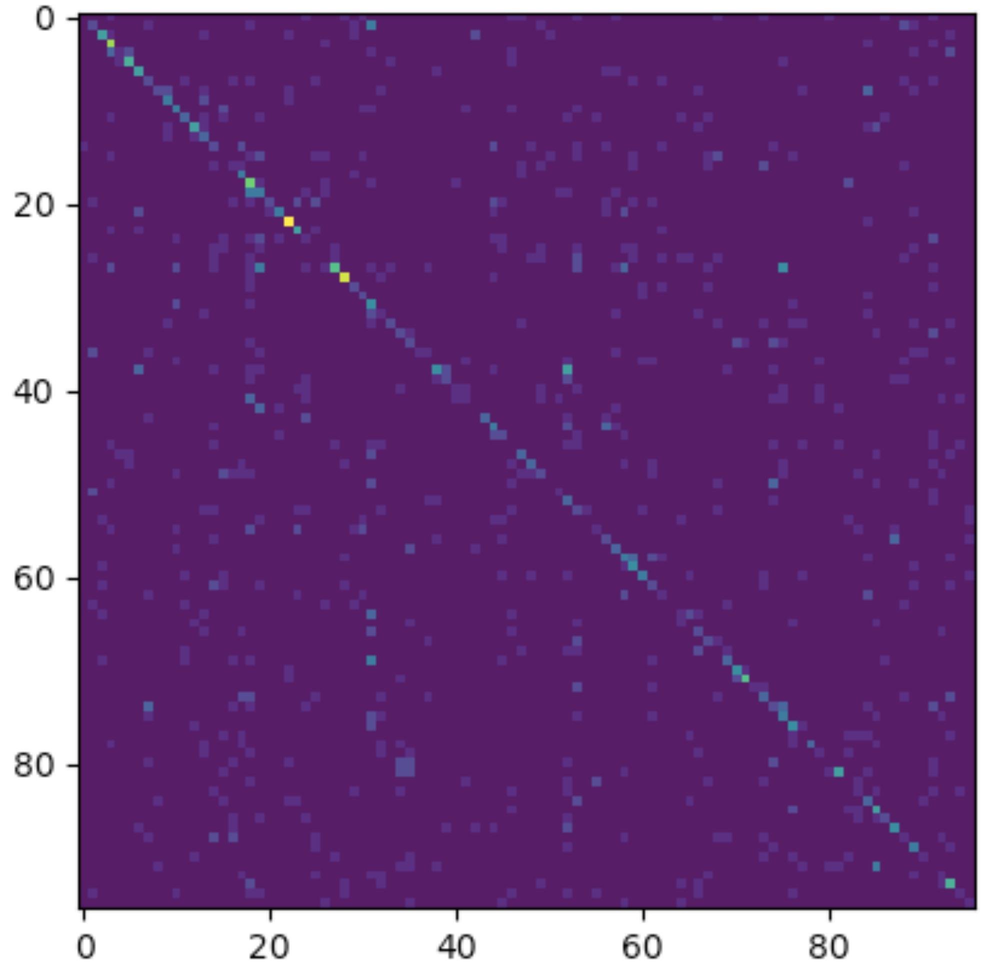
```
class    precision    recall  f1-score   support

  0        1.00       0.92      0.96        13
  1        0.86       0.92      0.89        13
  2        0.93       1.00      0.97        14
  3        1.00       0.92      0.96        13
  4        0.71       0.45      0.56        11
  5        0.60       0.82      0.69        11
  6        1.00       1.00      1.00         9
  7        1.00       0.90      0.95        10
  8        0.88       0.54      0.67        13
  9        0.67       0.86      0.75         7
 10        0.73       1.00      0.85        11
 11        1.00       0.83      0.91        12
 12        1.00       0.92      0.96        12
 13        0.83       1.00      0.91         5
 14        0.92       0.92      0.92        12
 15        0.80       0.57      0.67        14
 16        1.00       0.90      0.95        10
 17        1.00       1.00      1.00         6
 18        1.00       0.93      0.97        15
 19        1.00       1.00      1.00        13
 20        1.00       1.00      1.00        15
 21        1.00       0.92      0.96        12
 22        1.00       1.00      1.00        13
 23        1.00       0.90      0.95        10
 24        0.89       0.73      0.80        11
 25        0.64       0.90      0.75        10
 26        0.92       0.79      0.85        14
 27        0.97       1.00      0.99        36
 28        1.00       0.93      0.96        14
 29        0.56       0.71      0.63         7
 30        1.00       1.00      1.00         4
 31        0.89       0.89      0.89         9
 32        0.70       1.00      0.82         7
 33        1.00       0.91      0.95        11
 34        0.75       0.50      0.60         6
 35        0.80       0.89      0.84         9
 36        0.90       1.00      0.95         9
 37        1.00       0.88      0.93         8
 38        0.81       1.00      0.90        22
 39        0.83       0.56      0.67         9
 40        0.58       0.78      0.67         9
 41        0.83       0.77      0.80        13
 42        1.00       0.83      0.91         6
 43        0.70       0.88      0.78         8
 44        0.92       1.00      0.96        11
 45        1.00       0.88      0.93         8
```

46	1.00	0.83	0.91	12
47	1.00	1.00	1.00	9
48	0.91	0.91	0.91	11
49	0.92	0.86	0.89	14
50	1.00	0.80	0.89	10
51	0.67	1.00	0.80	4
52	1.00	0.86	0.92	7
53	0.85	0.92	0.88	12
54	1.00	0.88	0.93	8
55	0.92	1.00	0.96	12
56	1.00	0.89	0.94	9
57	1.00	1.00	1.00	7
58	1.00	0.92	0.96	13
59	0.89	1.00	0.94	8
60	0.85	0.85	0.85	13
61	1.00	1.00	1.00	11
62	1.00	1.00	1.00	8
63	0.50	0.83	0.62	6
64	1.00	1.00	1.00	10
65	0.83	0.83	0.83	6
66	1.00	1.00	1.00	8
67	0.75	1.00	0.86	6
68	1.00	0.86	0.92	7
69	1.00	1.00	1.00	14
70	1.00	1.00	1.00	7
71	1.00	0.92	0.96	12
72	1.00	1.00	1.00	5
73	0.93	1.00	0.97	14
74	1.00	0.71	0.83	14
75	0.73	0.92	0.81	12
76	0.90	1.00	0.95	9
77	1.00	1.00	1.00	8
78	1.00	1.00	1.00	10
79	1.00	1.00	1.00	7
80	0.77	0.91	0.83	11
81	0.76	0.87	0.81	15
82	1.00	1.00	1.00	8
83	1.00	1.00	1.00	9
84	0.83	0.91	0.87	11
85	1.00	0.91	0.95	11
86	0.88	0.88	0.88	8
87	0.91	0.83	0.87	12
88	1.00	0.75	0.86	12
89	0.73	1.00	0.84	8
90	0.71	0.71	0.71	7
91	0.90	0.82	0.86	11
92	0.86	0.86	0.86	7
93	1.00	0.93	0.97	15

```
          94       1.00      1.00      1.00        11
          95       0.83      0.83      0.83         6

    accuracy                           0.90
```

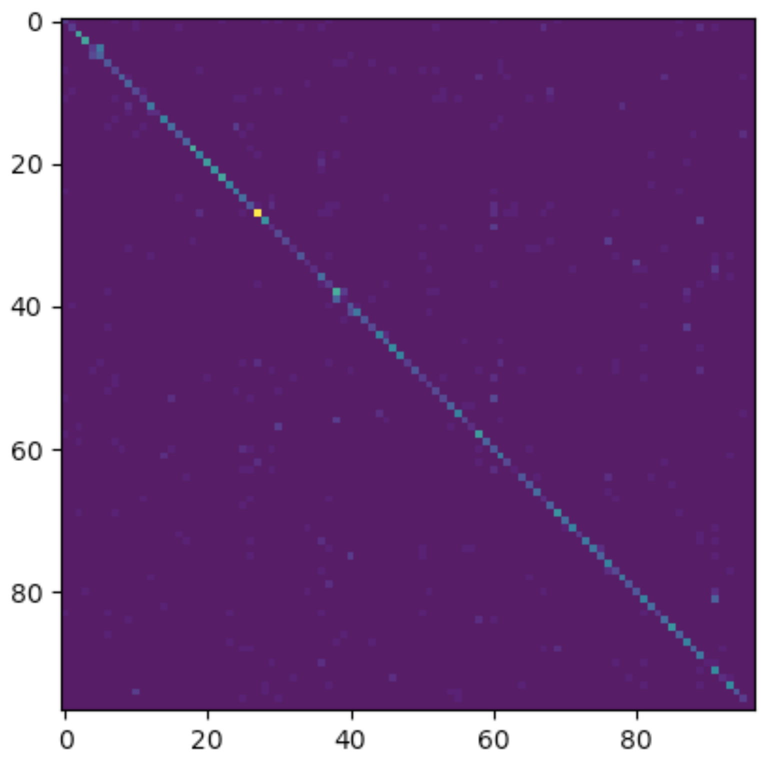
Matricile de confuzie:

Aceasta este matricea pentru Naive Baise:



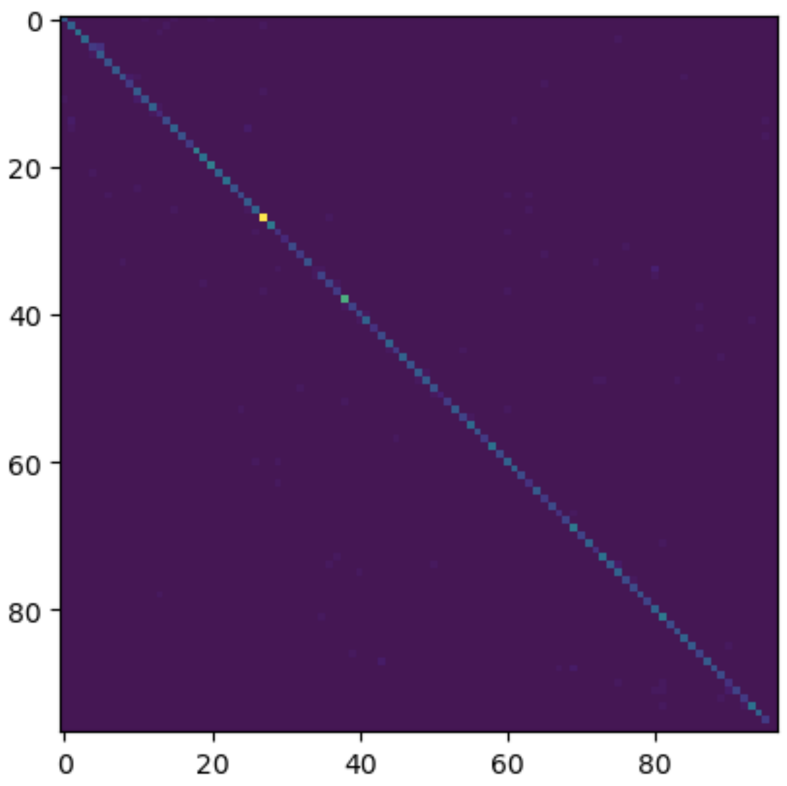
Putem oberva ca are destul de multe erori, dar pentru un NB, este rezonabil. Se justifica acuratetea de 27% :)

Matricea de confuzie pentru prima versiune de CNN:



Se oberva o matrice mult mai curata pentru acest model, justificand acuratetea de 57%.

Acum matricea pentru versiunea finala de CNN, cu acuratetea de 90%:



0
20
40
60
80
0
20
40
60
80