Témalabor:

Gépi tanulás, MI alkalmazása a gyártóiparban (predictive maintenance)

Mesterházi Marcell (TN0VU7)

Dokumentáció

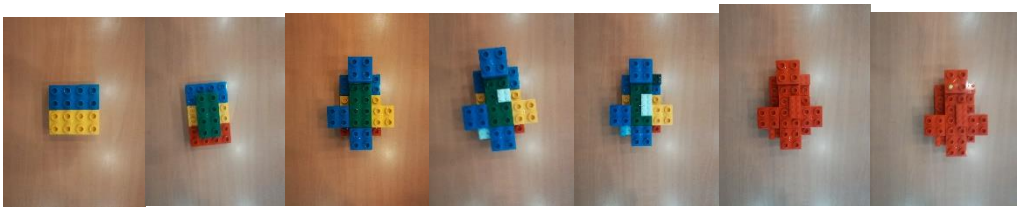
ML_Szivacs_classification:

A megbeszéltek alapján a szivacs képek osztályozását valósítottam meg Jupyterben, egyszerű machine learninggel, random foresttel, ami meglepően ügyesen, 98% pontossággal ((`szivacs_calssification_regi`))

A témalaboron kicsit egyszerűsítettünk a képek tárolásán, a pandas dataframe-mel, ((`szivacs_calssification_temalab_jav`))

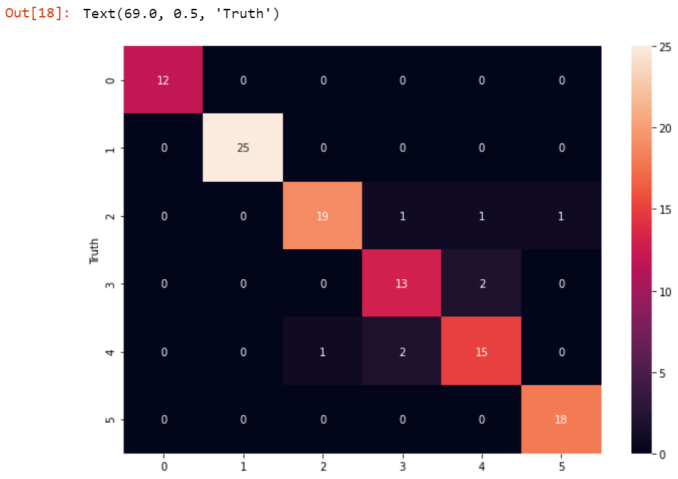
ML_Lego_classification:

Állapotok:



A legokról készül képeket elforgattam, kibalanszoltam az állapotokat (minden állapotról kb 90 kép), most a már megírt szivacson alkalmazott random forestes programmal futtattam a Lego-s adatokra. Először csak az első 6 állapotot osztályoztam.

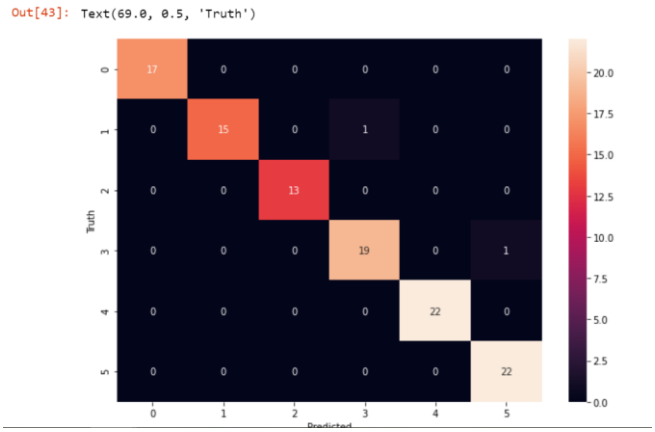
- A pontosság 50x50-es képekkel, random foresttel 92% lett, confusion matrix:



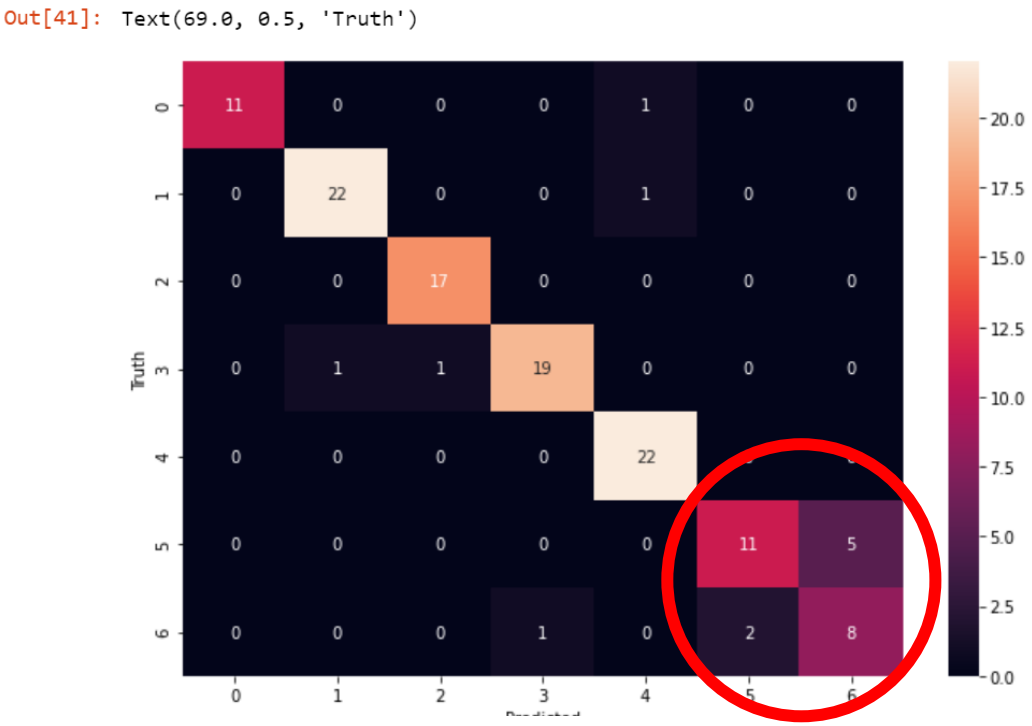
- A pontosság 100x100-as képekkel 93,4% lett(utána már csak 90%), confusion matrix

ML_Lego_classification:

- A pontosság 100x100-as képekkel, logistic regressionnel 98% lett, confusion matrix



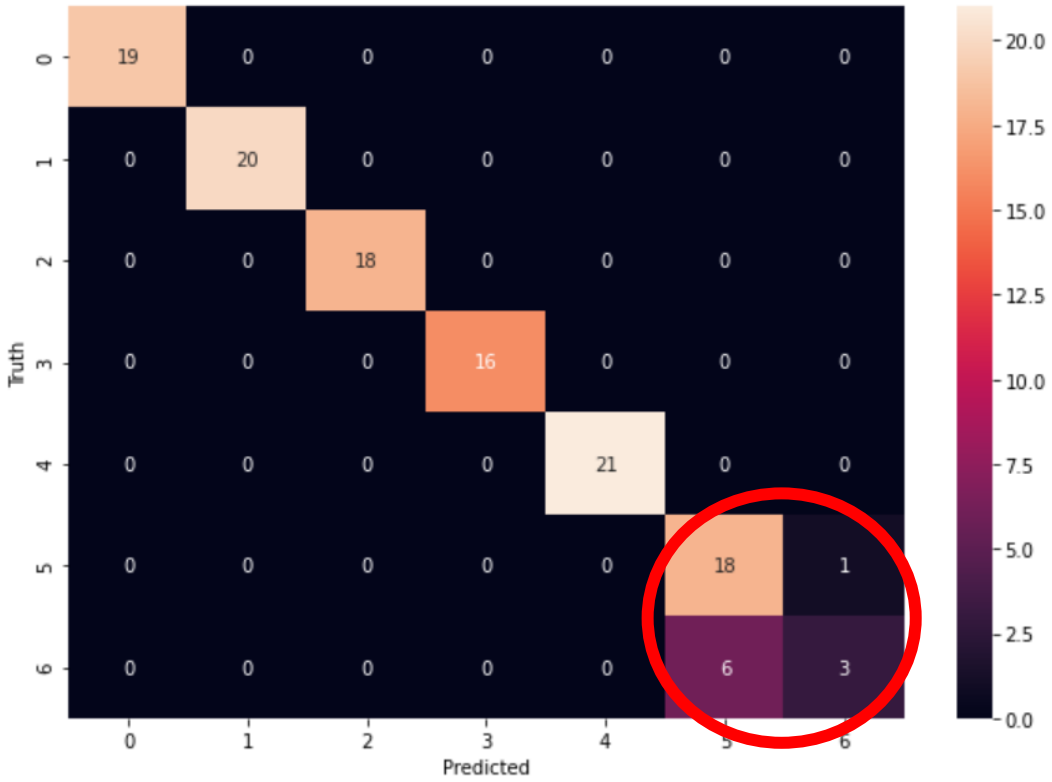
- ha hozzáteszem az phase_6+errors részt, már csak 90%, látszik is, hogy az utolsó két állapotnál pontatlanabb, akkor sem javul, ha megemeljük a képek felbontását, vagy színes képeket olvasunk be:



Deep learning:

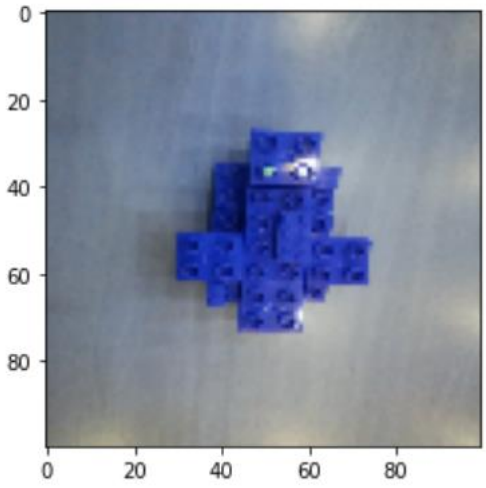
A programot pytorch-chal folytattam, ez 100x100-as fekete-fehér képekkel, 20 batch-al a pontossága hét állapotra tekintve 94%-os volt.
Látható módon, ő is megküzdött az utolsó kettő, nehezebben megkülönböztethető állapottal, de nem annyira, mint machine learning esetén

```
Out[25]: Text(69.0, 0.5, 'Truth')
```



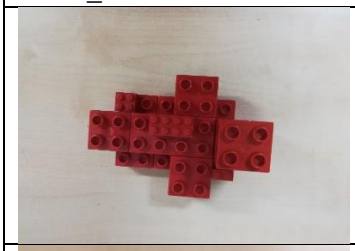
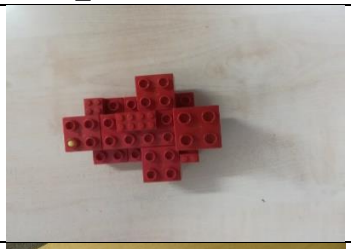
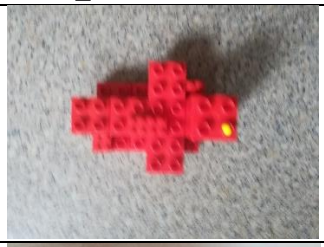
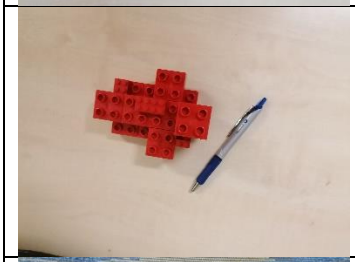
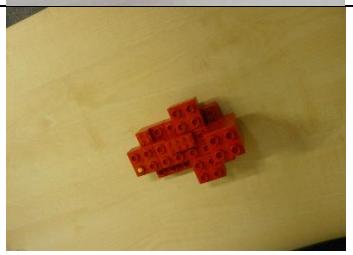
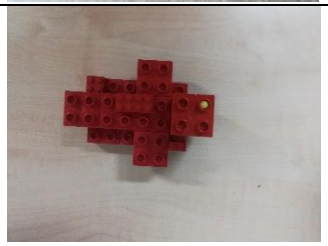
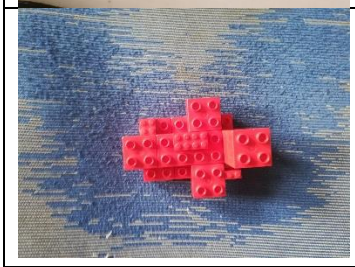
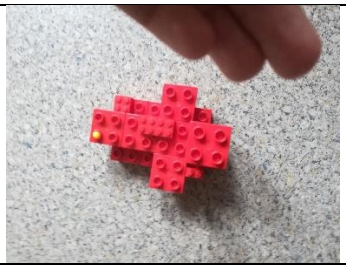
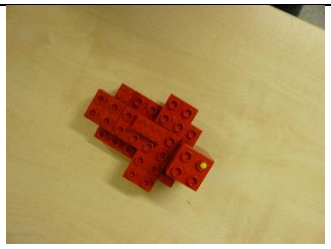
Érdekes volt, hogy amikor a tesztek közül kiírattam a rosszul megtippelt, látszott, hogy néha miért is tévedett a programunk, itt például a visszacsillanó fény megzavarta.

```
Valos:  6 , tippelt:  5
```



Lego error classification:

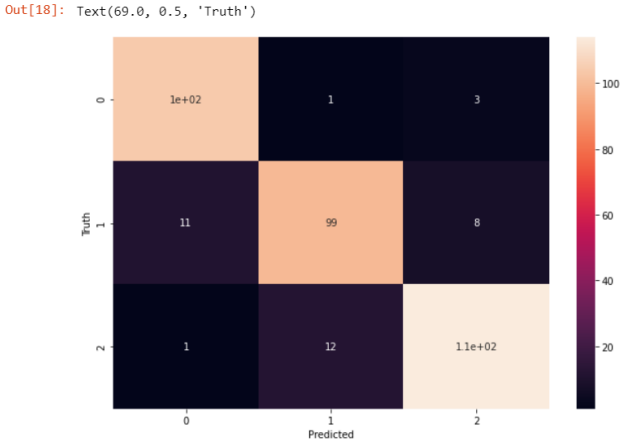
Ezeknél a képeknél már sokat változtak a szögek, a fények, a háttér is, különbség ugye a sárga pont helyzete.

Phase_1	Phase_2	Phase_3
		
		
		

Meglepő módon a **Randomforest** tartja a versenyt a Deep Learning programokkal, többszöri futtatás után is körülbelül **90%**-os eredményt kapunk.

```
In [19]: model.fit(X_train,Y_train)
Out[19]: RandomForestClassifier()

In [20]: model.score(X_test, Y_test) #pontosság
Out[20]: 0.9121813031161473
```

((további futtatások után ez az eredmény inkább átlagosan 86% körül volt))



A **Pytorch**-os Neural Network 54 epoch után **90%** pontosságú lett:

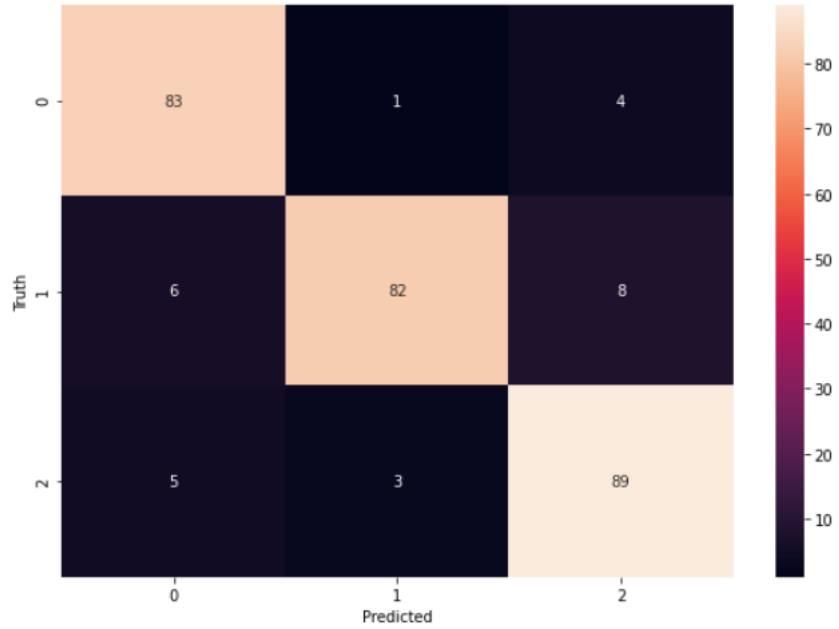
```
Epoch: 51. Loss: 2.0176346879452467e-05
100%|██████████████████████████████████████████████████████████████████████████| 23/23 [01:25<00:00,  3.71s/it]
  0%|                                                                            | 0/23 [00:00<?, ?it/s]
Epoch: 52. Loss: 1.8091088804794708e-06
100%|██████████████████████████████████████████████████████████████████████████| 23/23 [01:23<00:00,  3.64s/it]
  0%|                                                                            | 0/23 [00:00<?, ?it/s]
```

```
print("Accuracy: ", round(correct/total, 3))
```

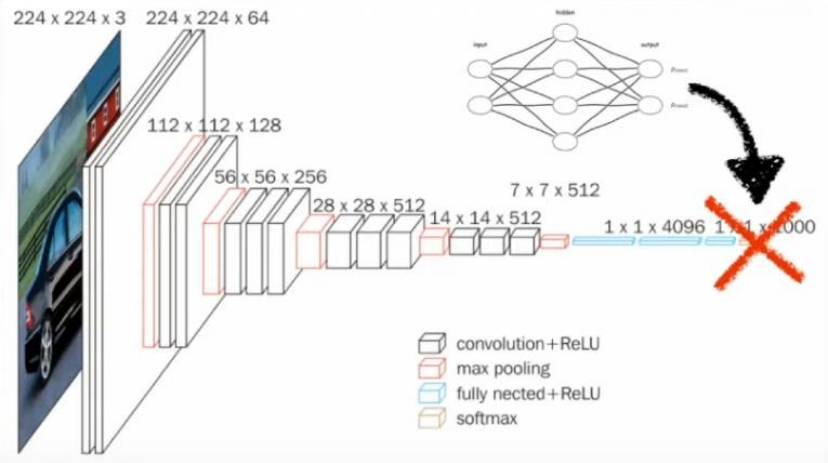
```
100%|██████████████████████████████████████████████████████████████████████████| 281/281 [00:09<00:00, 29.78it/s]
Accuracy:  0.904
```

conf. matrix:

```
Out[15]: Text(69.0, 0.5, 'Truth')
```



Pre-trained Neural Network használata, aminek a lényege az, hogy egy előre, rengeteg képpel betanított neurális hálót használunk fel, amelynek az utolsó layerét változtatjuk a sajátunkra, és ezen a hálón tanítjuk a képek alapján.




```
#importáljuk a már kész NN-t

from torchvision.models import squeezenet1_0

model = squeezenet1_0(pretrained=True)
print(model)
```

SqueezNet: pre-trained neural network, gyorsabb, kisebb, mint sok más pre-trained NN, például az AlexNet-ben 50szer több paraméter van, és sokkal gyorsabb, ez volt a célja a fejlesztésének, jelenleg az önvezető autóknál is használják.

```
#módosítjuk az utolsó rétegét

n_classes = 3

model.num_classes = n_classes
model.classifier[1] = nn.Conv2d(512, n_classes, kernel_size=(1,1), stride=(1,1))
```

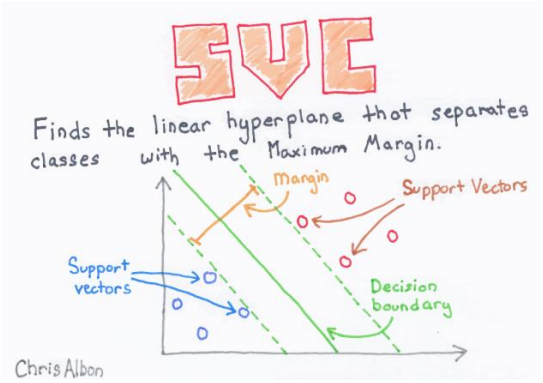
A neurális háló 10 epoch után **93%**-os pontosságú lett.

```
The testing set accuracy of the network is: 93 %
```

20 epoch után sem lett pontosabb, 92%-os pontossággal dolgozott:

```
The testing set accuracy of the network is: 92 %
```

A következő hétre különböző más megoldásokat is megpróbáltam:



SVC - Support Vector Classifier kipróbálása:

```
In [28]: from sklearn.svm import SVC
         model = SVC(C=1, kernel='poly', gamma = 'auto')
         model.fit(X_train,Y_train)

Out[28]: SVC(C=1, gamma='auto', kernel='poly')
```

Adat

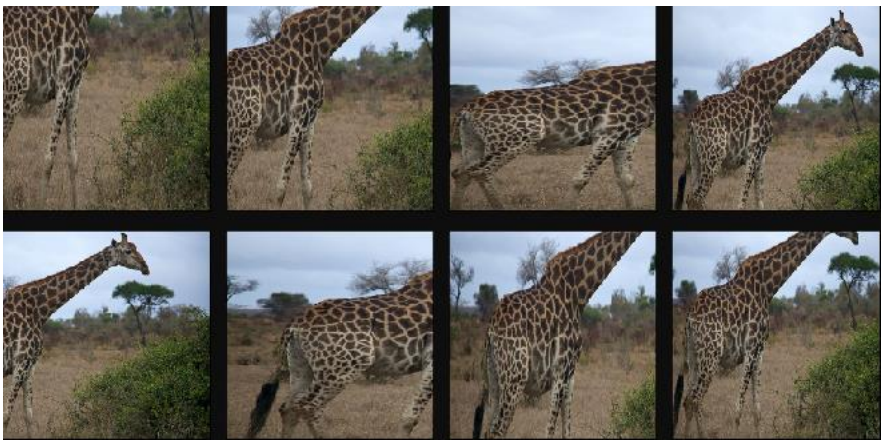
```
In [29]: model.score(X_test, Y_test) #pontosság

Out[29]: 0.7903682719546742
```

Augmentáció és SqueezeNet:

Adat aumentáció:

```
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    normalize,
])
```

A háló felépítéséhez használt (train) képeket nem csak egyszerűen átadom, hogy abból tanuljon a háló, hanem a fent látható módonm kétféleképpen is; az utóbbi (*RandomHorizontalFlip*) egyszerűen csak tükröz, a *RandomResizedCrop* kicsit belezoomolhat, kizoomolhat, egy kicsit balra zoomol, kicsit jobbra, lényege, hogy a kép egy részét adja vissza, de emiatt minden egyes epochnál végülis „más” képekkel tanítjuk:



RandomResizedCrop működése

A módszer emiatt minden egyes epochnál újra olvassa a képhalmazt, ami miatt lényegesen lassabb lesz, de így tudott a pontosságán javítani:

```
Háló pontossága a test képeken 90 %
12 . epoch
Háló pontossága a test képeken 95 %
```

Nem biztos, hogy ez a legjobb megoldás, meglátom még a másik fajta, VGG16 előretanított neurhálóval, de ennek a 20 epochnak a futtatása is 1,5 óra volt Colab-os szuper GPU-val.

Írtam egy *RandomResizedCrop* szerű függvényt (itt ugye nem volt használható ez, mert másképpen olvastuk be a képeket) és kipróbáltam a másik, teljesen saját neurális hálón, így ~1500 helyett ~7000 képen tanulhat a háló:

```
def get_random_crop(image, crop_size):
    x = np.random.randint(0, image.shape[1]-crop_size)
    y = np.random.randint(0, image.shape[1]-crop_size)
    crop = image[y: y + crop_size, x: x + crop_size]
    return crop
```

```
training_data = np.load("training_data.npy", allow_pickle=True)
print(len(training_data))
```

```
7040
```

A Knorr-os képek érkezéséig megvizsgáltam még az egyes esetekben vizsgált valószínűségeket, hogy milyen pontos a háló olyan esetben, amikor feltételhez kötjük, hogy mennyire biztos a háló a tippjében:

```
predictions = model.predict_proba(X_test)
#print(predictions) y_test

cnt = 0
talalt = 0
probability = 0.5

for pr in np.arange(0.6,0.35,-0.03):
    #print(round(p,3))
    p = (round(pr,3))
    cnt = 0
    talalt = 0
    for i in range(len(X_test)):

        #tippelt megoldás és valószínűségének kiírása
        #print(Y_predicted[i])
        #print(predictions[i][Y_predicted[i]])

        if((predictions[i][Y_predicted[i]])>p):
            cnt += 1
            if(Y_predicted[i] == Y_test[i]):
                talalt += 1

    print(len(X_test),"ból ",cnt,"-szer volt ",p*100,"%-nál biztosabb.\n Ilyenkor pontossága:",talalt/cnt*100,"ami a teljes te
```

Ugye az adott három osztály esetén vizsgáltam meg 60 és 35% között:

```
353 ból  214 -szer volt  60.0 %-nál biztosabb.
 Ilyenkor pontossága: 99.06542056074767 ami a teljes teszt halmaz 0.6062322946175638 %-a.

353 ból  235 -szer volt  56.99999999999999 %-nál biztosabb.
 Ilyenkor pontossága: 99.14893617021276 ami a teljes teszt halmaz 0.6657223796033994 %-a.

353 ból  255 -szer volt  54.0 %-nál biztosabb.
 Ilyenkor pontossága: 97.25490196078431 ami a teljes teszt halmaz 0.7223796033994334 %-a.

353 ból  277 -szer volt  51.0 %-nál biztosabb.
 Ilyenkor pontossága: 96.028880866426 ami a teljes teszt halmaz 0.7847025495750708 %-a.

353 ból  296 -szer volt  48.0 %-nál biztosabb.
 Ilyenkor pontossága: 94.93243243243244 ami a teljes teszt halmaz 0.8385269121813032 %-a.

353 ból  319 -szer volt  45.0 %-nál biztosabb.
 Ilyenkor pontossága: 91.84952978056427 ami a teljes teszt halmaz 0.9036827195467422 %-a.

353 ból  334 -szer volt  42.0 %-nál biztosabb.
 Ilyenkor pontossága: 90.41916167664671 ami a teljes teszt halmaz 0.9461756373937678 %-a.

353 ból  344 -szer volt  39.0 %-nál biztosabb.
 Ilyenkor pontossága: 90.11627906976744 ami a teljes teszt halmaz 0.9745042492917847 %-a.

353 ból  351 -szer volt  36.0 %-nál biztosabb.
 Ilyenkor pontossága: 88.31908831908832 ami a teljes teszt halmaz 0.9943342776203966 %-a.
```

Az fenti adatokból kiszűrhető egyfajta optimális tipp, ahol még az adatok nagy részét osztályozzuk, de a pontosságon is javítunk.

Itt a randomforest alapból 85%-on teljesített, a levont tanulságaim:

- ha már 39%-ra állítjuk a minimum tippszázalékot, **90%-ra javul**, és a teszt képek **97.5%-át felhasználta**.
- ha ezt a százalékot 48%-ra tesszük majdnem 95%-ra javul, itt a teszt képek 84%-át felhasználva.

tervezem még megnézni, hogy melyik képeknél tippel rosszul (mondjuk 40% alatt) és megcsinálni ezt a dataloaderes olvasásnál is, csak ott bonyolultabb a batchek és tensorok miatt, de meglesz! :D