Témalabor:

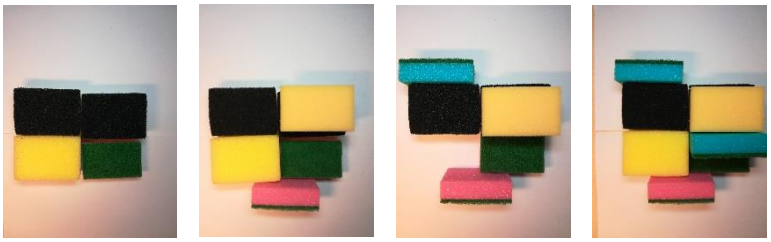
Gépi tanulás, MI alkalmazása a gyártóiparban (predictive maintenance)

Mesterházi Marcell (TN0VU7)

Dokumentáció

ML_Szivacs_classification:

A megbeszéltek alapján a **szivacs képek osztályozását** valósítottam meg Jupyterben:



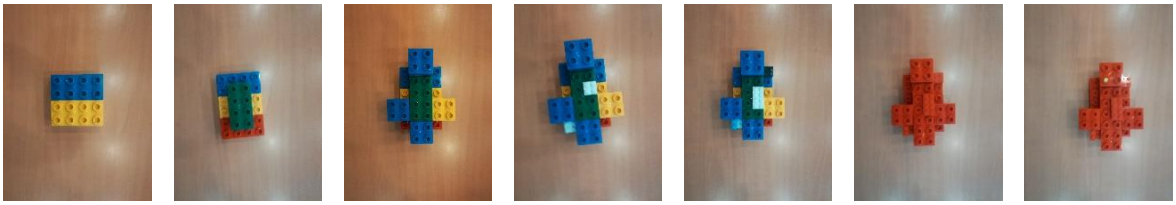
Egyszerű machine learninggel osztályoztam, ahol a legjobb eredményt **Random Forest**-tel értem el, ami meglepően ügyesen, **97-100% pontosság**gal prediktált ((`szivacs_calssification_regi`))

```
In [18]: model.fit(X_train,Y_train)
Out[18]: RandomForestClassifier()

In [19]: model.score(X_test, Y_test) #pontosság
Out[19]: 1.0
```

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

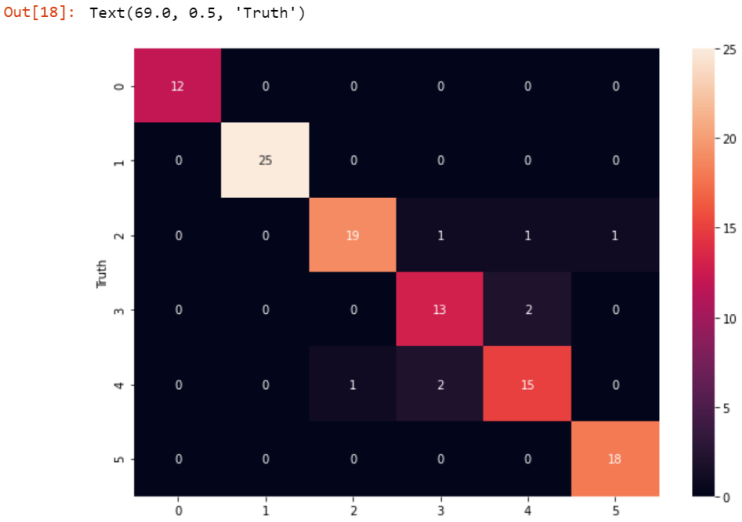
ML_Lego_classification:

Állapotok:

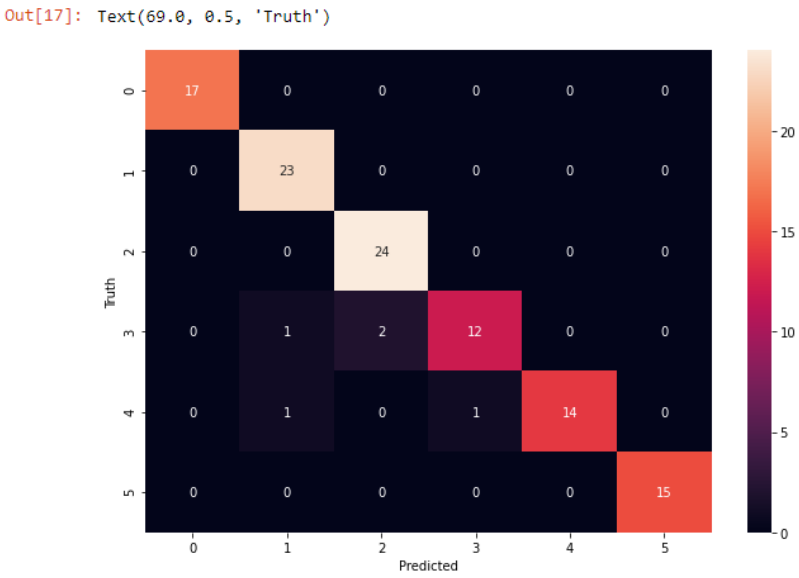


A legokról készül képeket elforgattam, kibalanszoltam az állapotokat (minden állapotról kb 90 kép), most a már megírt szivacson alkalmazott **Random Forest**-es programmal futtattam a Lego-s adatokra. Először csak az első 6 állapotot osztályoztam.

- A pontosság 50x50-es képekkel, **Random Forest 92%** lett, confusion matrix:

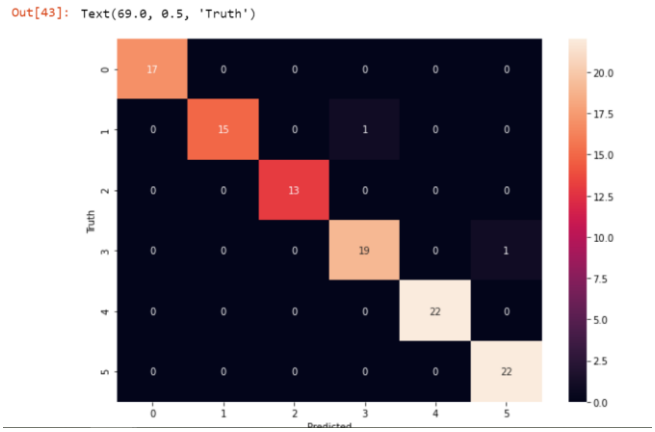


- A pontosság 100x100-as képekkel **93,4%** lett(utána már csak 90%)

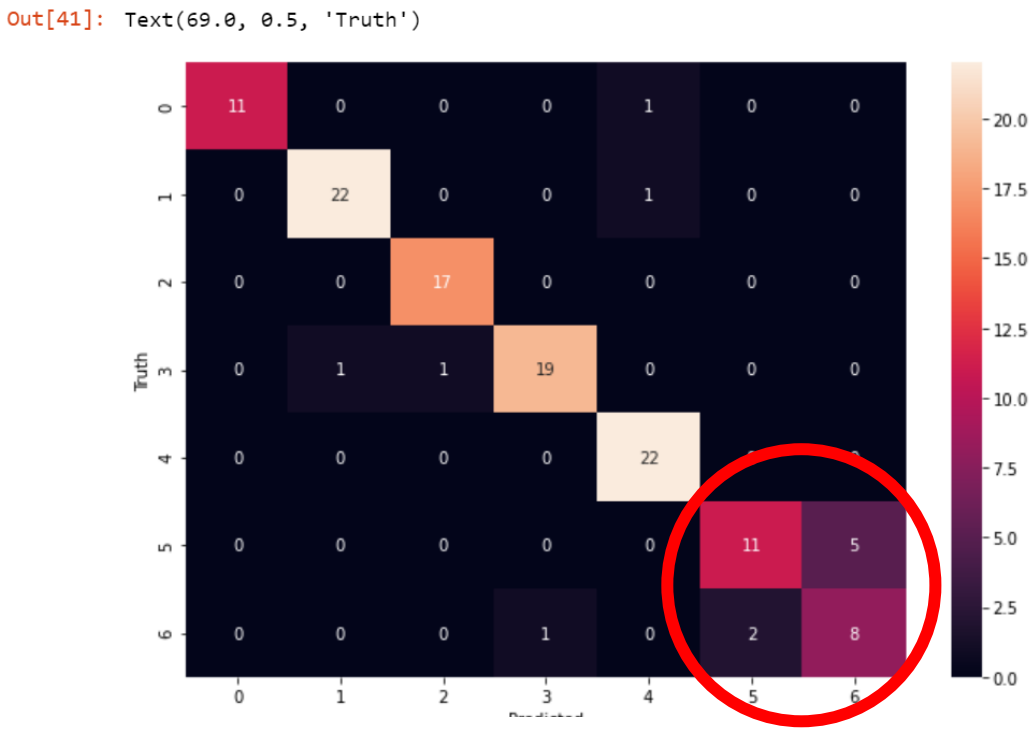


ML_Lego_classification:

- 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:

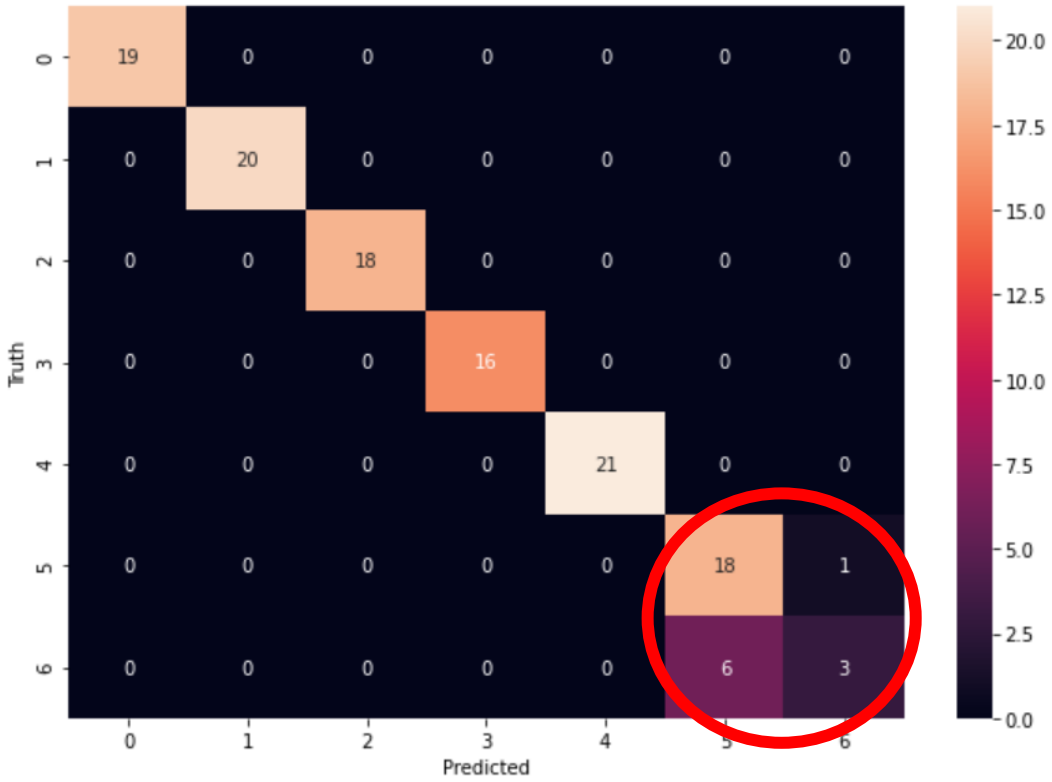


Látható, hogy az utolsó két állapot már szemmel láthatóan nehezebb megkülönböztetni, ami a programnak is nehéz feladat volt, lááthatóan itt elég sok tévedés történt.

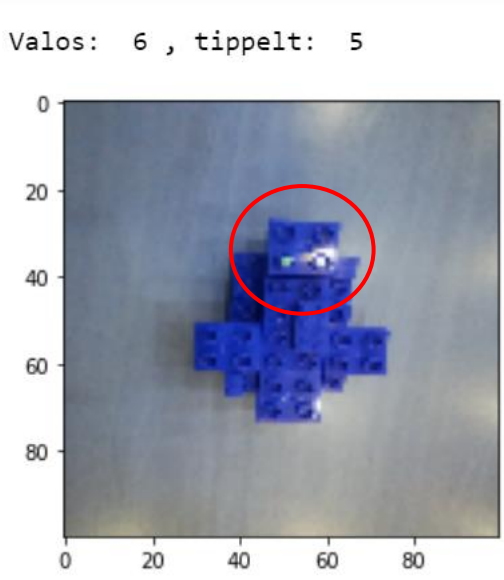
Deep learning:

A témalaboron az osztályozást **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')
```

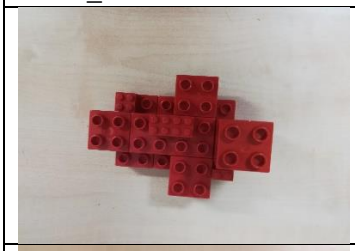
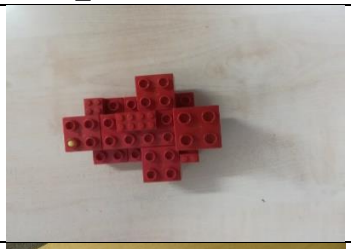
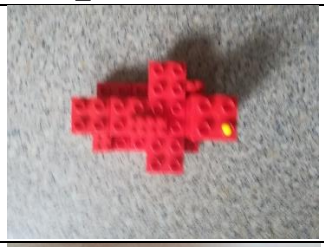
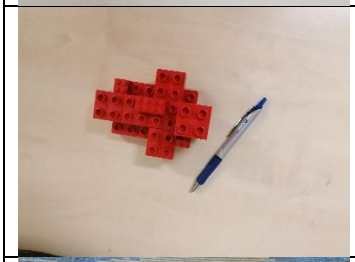
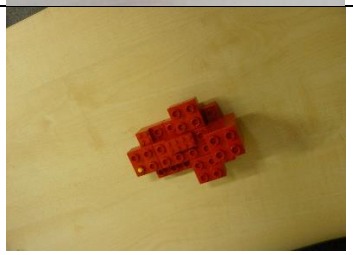
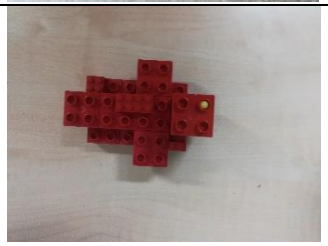
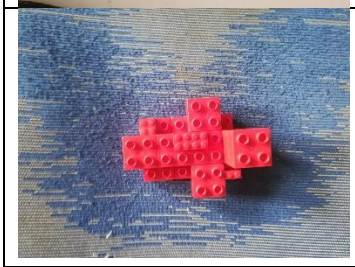
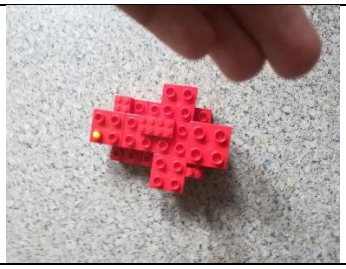
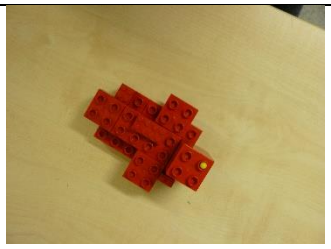


Érdekes volt, hogy amikor a tesztek közül kiírattam a rosszul megtippelt, látszott, hogy néha miért is tévedett a programunk, itt például a visszacsillanó fény megzavarta.



Lego error classification:

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

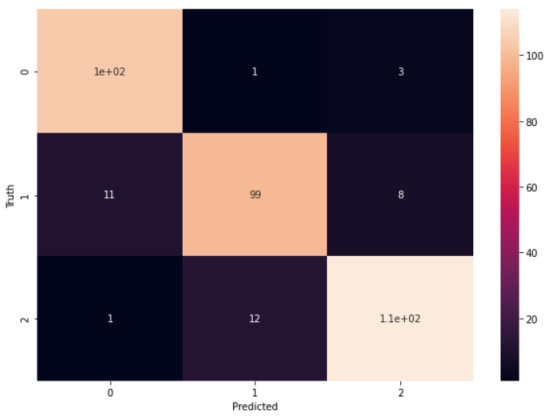
Phase_1	Phase_2	Phase_3
		
		
		

Meglepő módon a **Randomforest** alig van lemaradva a Deep Learning programoktól, 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))

```
Out[18]: Text(69.0, 0.5, 'Truth')
```



Pytorch-os NN

A neurális háló felépítése:

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 5)
        self.conv2 = nn.Conv2d(32, 64, 5)
        self.conv3 = nn.Conv2d(64, 128, 5)

        #random adat
        x = torch.randn(100,100,3).view(-1,3,100,100)
        self._to_linear = None
        self.convs(x) #átküldjük

        self.fc1 = nn.Linear(self._to_linear, 512)
        self.fc2 = nn.Linear(512, 3) #itt az osztályok száma

    def convs(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2,2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2,2))
        x = F.max_pool2d(F.relu(self.conv3(x)), (2,2))

        if self._to_linear is None:
            self._to_linear = x[0].shape[0]*x[0].shape[1]*x[0].shape[2]
        return x

    def forward(self, x):
        x = self.convs(x)
        x = x.view(-1, self._to_linear)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.softmax(x, dim=1)


net = Net()
print(net)
```

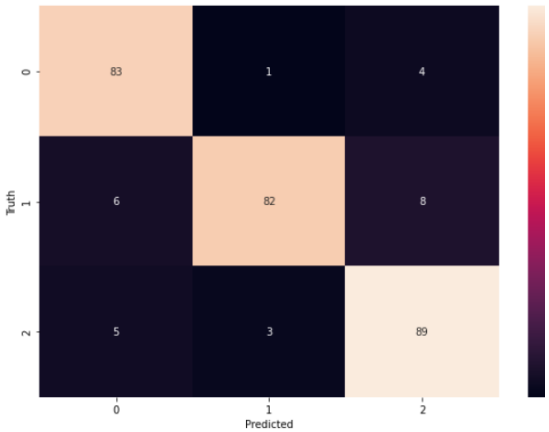
A neurális háló epoch után **90%** pontosságú lett:

```
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]
```

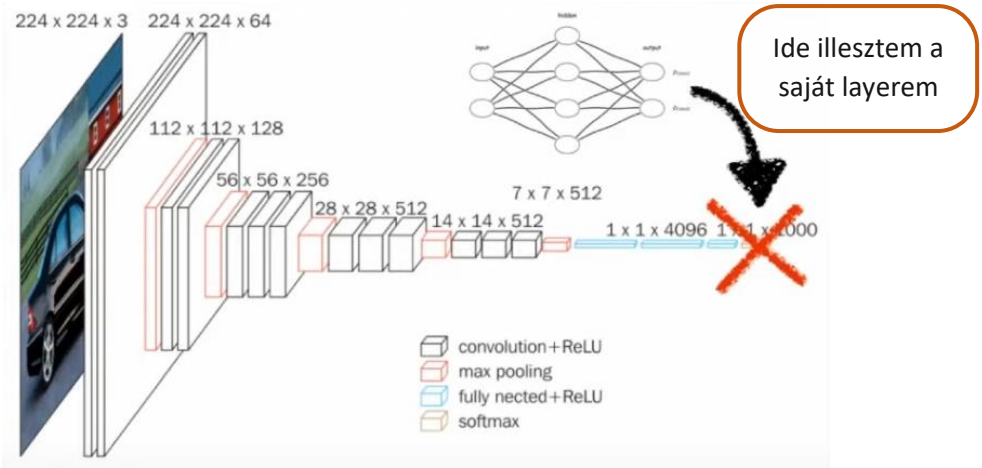
```
print("Accuracy: ", round(correct/total, 3))
```

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

Conf. matrix:



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, 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.

Itt állítottam be a saját layeremet a `model`-nek:

```
#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 már 95%-os pontossággal dolgozott:

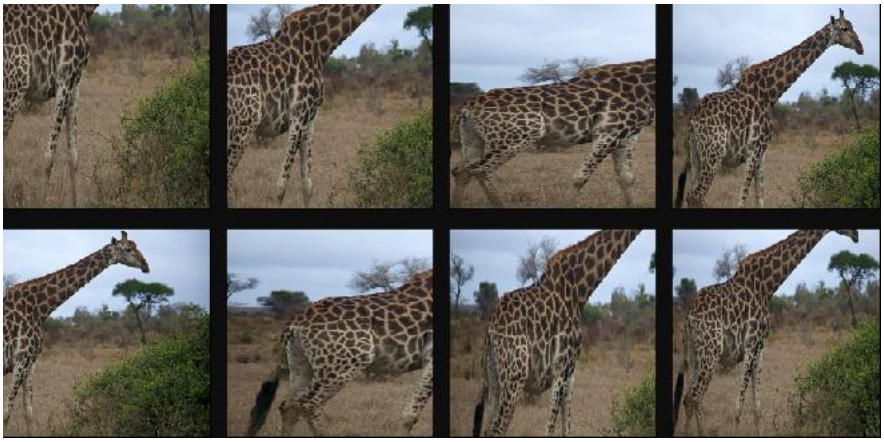
```
Háló pontossága a test képeken 94 %
Finished Training
```

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 módon 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 <u>lényegesen lassabb lesz</u>, de így tudott a pontosságán javítani:

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

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
```

Képbeolvasás művelete:

```
def make_training_data(self):
    for label in self.LABELS:
        for f in tqdm(os.listdir(label)):
            if "jpg" in f:
                path = os.path.join(label, f)
                img = cv2.imread(path, cv2.IMREAD_COLOR)
                img = cv2.resize(img, (self.IMG_S, self.IMG_S))
                img = zoom_in(img)
                self.training_data.append([np.array(img), np.eye(3)[self.LABELS[label]]]) # állapotszám beáll.

    if augmentation:
        for i in range(10):  #hányszorosan augmentáljuk a training datát
            for label in self.LABELS:
                for f in tqdm(os.listdir(label)):
                    if "jpg" in f:
                        path = os.path.join(label, f)
                        img = cv2.imread(path, cv2.IMREAD_COLOR)
                        img = cv2.resize(img, (self.IMG_P_SIZE, self.IMG_P_SIZE))
                        aug_img = get_random_crop(img, 100)
                        aug_img = cv2.flip(aug_img, r)
                        self.training_data.append([np.array(aug_img), np.eye(3)[self.LABELS[label]]]) # állapotszám beáll.

    np.random.shuffle(self.training_data)
    np.save("training_data100zoom.npy",self.training_data)
```

Adat-augmentáció

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
```

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.
```

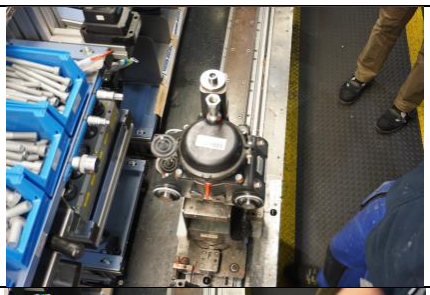
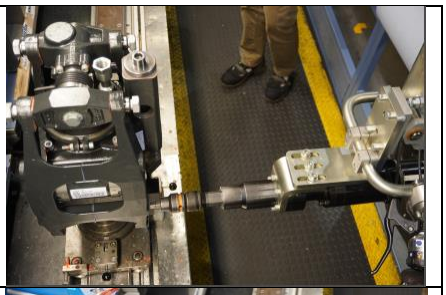
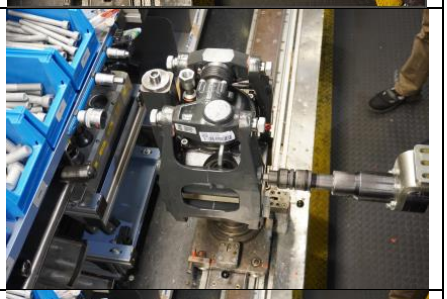
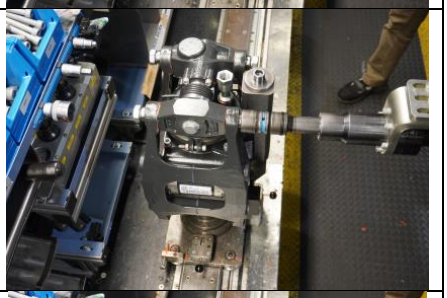
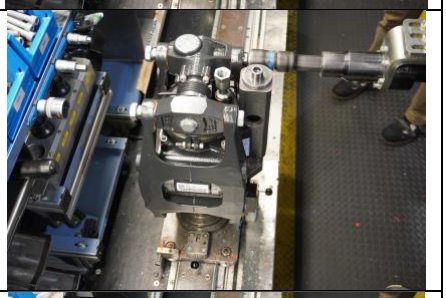
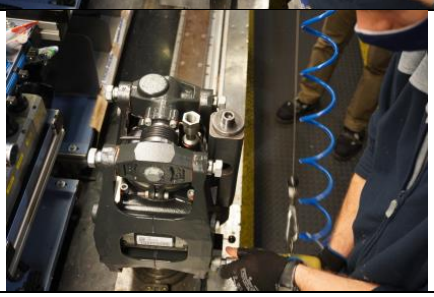
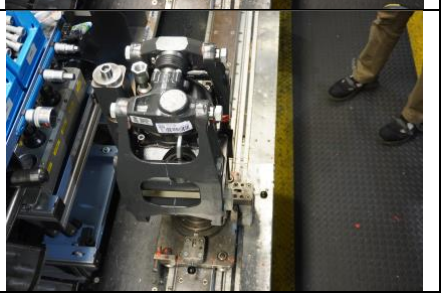
Az fenti adatokból kiszűrhető egyfajta „optimális tipp”, ahol még az adatok nagy részét osztályozzuk, de a pontosságon is javítunk.

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.

Knorr gyártósor:

Az egyik állomáson található képeket elemeztem, amit a következő 10 osztályra bontottam:

1		6	
2		7	
3		8	
4		9	
5		10	

Fényképek előkészítése:

A kapott fényképeknél probléma volt, hogy az egyes folyamatok nem ugyan olyan hosszúak, ami miatt a képhalmaz aránytalan volt, egyes állomásokról sokkal több kép készült. Ahhoz, hogy a képhalmazom osztályaiban az elemekből körülbelül ugyanannyi legyen (a neurális hálók tanításánál fontos a kiegyensúlyozottság) augmentációt használtam. Az általam írt adataugmentációs függvény kicsit belezoomol a képekbe.

Augmentációhóz használt függvény és használata:

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

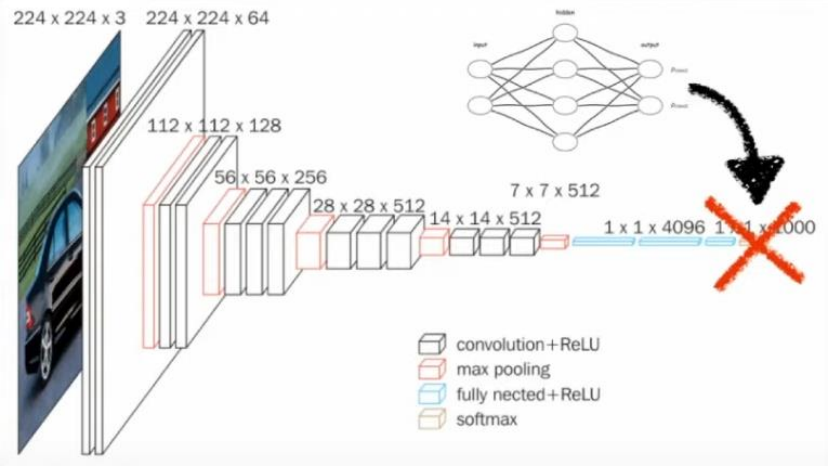
```
if "JPG" in f:
    path = os.path.join(label, f)
    img = cv2.imread(path, cv2.IMREAD_COLOR)
    img = cv2.resize(img, (720, 480))
    aug_img = get_random_crop(img, 400, 600)
    self.training_data.append([np.array(aug_img),
```

Például ebből a képből ezt a hármat készíti:

Amint az látható, csak kicsit vágok le a képből, a lényeg mindenképpen látszani fog a képen, csak kicsi belezoomolások történtek.

Ezzel a technikával a képhalmazt kiegészítettem ~50-50 képesre, hozzá 6-6 képpel teszteltem.

A kész adathalmazt a már elkészített Pytorch-s konvolúciós neurális hálómon futtatam. A neurális hálóm egy **squeezeNet**-es hálón alapul, aminek az utolsó rétegét változtatttam meg.



A háló forráskódjának főbb részei:

Adatbeolvasás – itt importálom a *train* és *test*-ben található képeket, a train képeken továbbá látható, hogy minden beolvasásnál végzünk augmentációt, amivel minden olvasáskor mindig kicsit más képeket kap inputnak a háló.

```
[ ]  from torchvision import transforms
     from torchvision.datasets import ImageFolder
     normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225])


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

     test_transform = transforms.Compose([
         transforms.Resize(256),
         transforms.CenterCrop(224),
         transforms.ToTensor(),
         normalize,
     ])

     train_set = ImageFolder("/content/drive/MyDrive/gyartosor/train", transform = train_transform)
     test_set = ImageFolder("/content/drive/MyDrive/gyartosor/test", transform = test_transform)
```

Importálom a SqueezeNet-et:

```
]  #importáljuk a már kész NN-t

   from torchvision.models import squeezenet1_0

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

Saját layerem, amit hozzáadok, és a hozzáadás lépése:

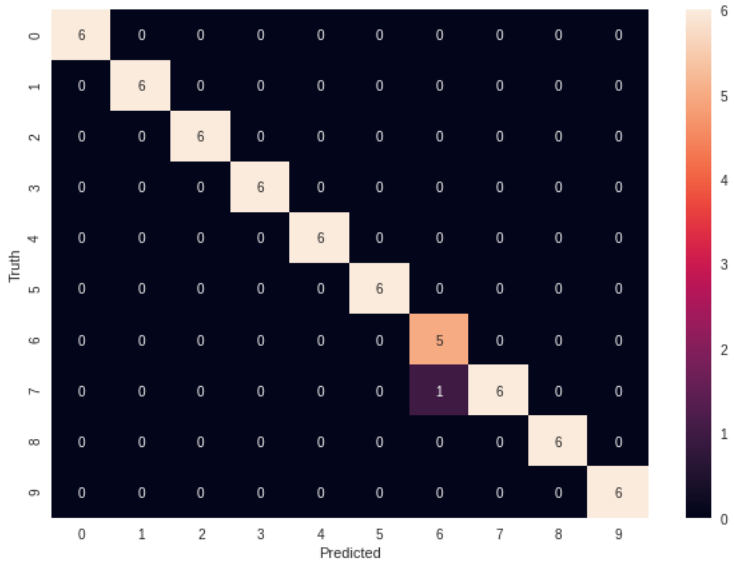
```
Net(
  (conv): Conv2d(3, 18, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=4608, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=3, bias=True)
)
```

```
#módosítjuk az utolsó rétegét a saját layeremre

n_classes = 10

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

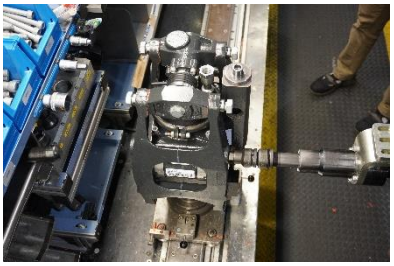
A háló tanulás, 20 epoch után **98%** pontosságú lett a test képeken:

```
18 . epoch
Háló pontossága a test képeken 98 %
19 . epoch
Háló pontossága a test képeken 98 %
Finished Training
```

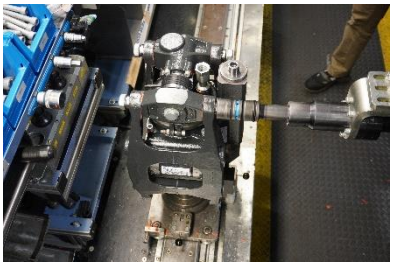
Az eredményhez tatorzó confusion mátrix:



A confusion mátrixból leolvasható, hogy csak egyet tippelt rosszul, a hetedik osztály egy elemét keverte össze a nyolcadik osztály egy elemével. A bal oldali a kép, amit eltévesztett, a jobb oldali pedig egy nyolcadik állapotú kép, amilyennek „nézte” a háló (különbség ugye a csavarhúzó? helyzete)



az eltévesztett kép



egy kép a nyolcadik osztályból, aminek prediktálta a neurhálónk