Témalabor:

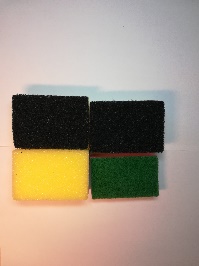
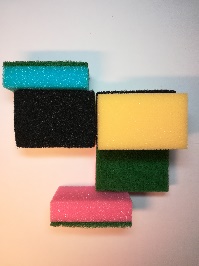
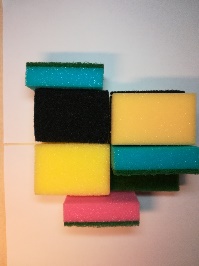
**Gépi tanulás, MI alkalmazása a gyártóiparban (predictive maintenance)**

Mesterházi Marcell (TN0VU7)

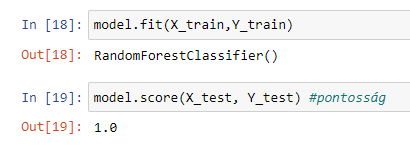
Dokumentáció

ML\_Szivacs\_classification:

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

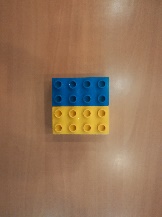
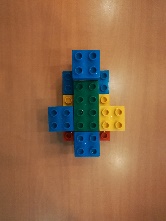
Egyszerű machine learninggel osztályoztam, ahol a legjobb eredményt **Random Forest**-tel értem el, ami meglepően ügyesen, **97-100% pontosság**gal prediktált ((szivacs\_calssification\_regi))



A témalaboron kicsit egyszerűsítettünk a képek tárolásán, a *pandas dataframe*-mel, ((szivacs\_calssification\_temalab\_jav))

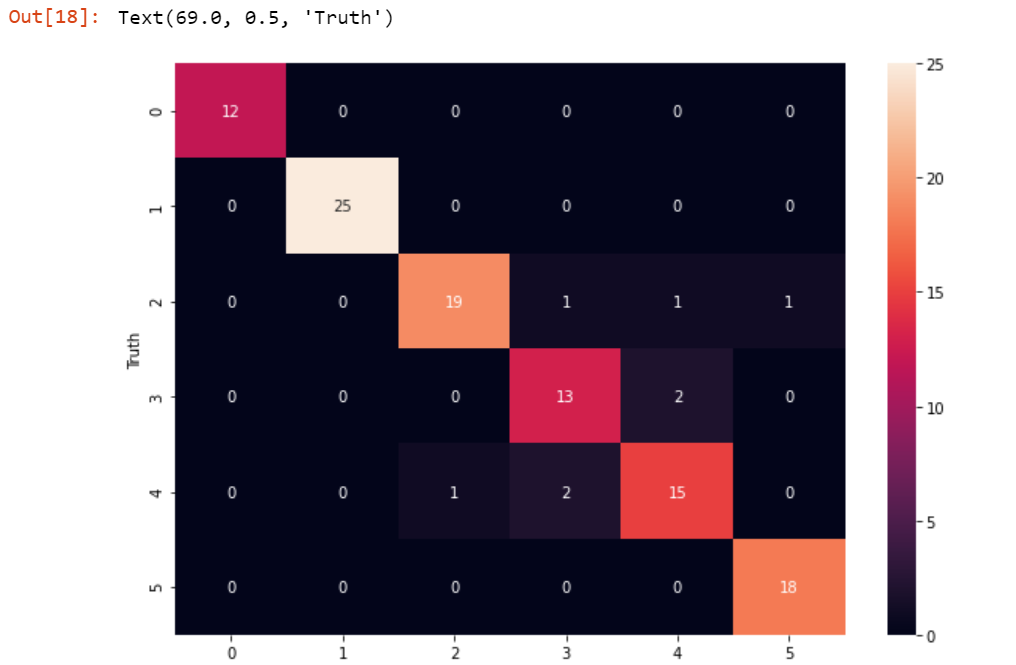
**ML\_Lego\_classification:**

Állapotok:

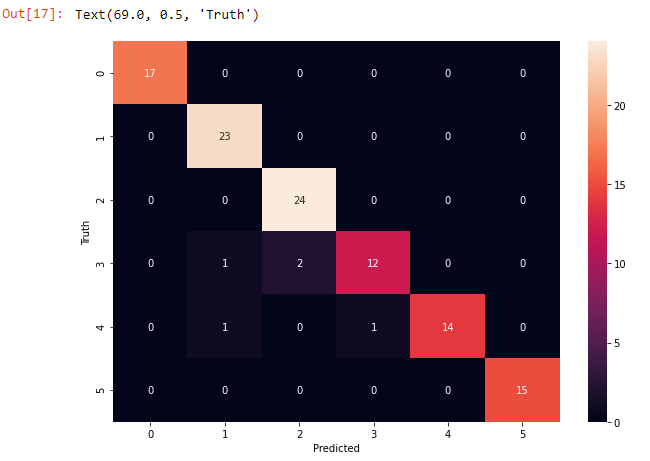
      

A legokról készül képeket elforgattam, kibalanszoltam az állapotokat (minden állapotról kb 90 kép), most a már megírt szivacson alkalmazott **Random Forest**-es programmal futtattam a Lego-s adatokra. Először csak az első 6 állapotot osztályoztam.

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

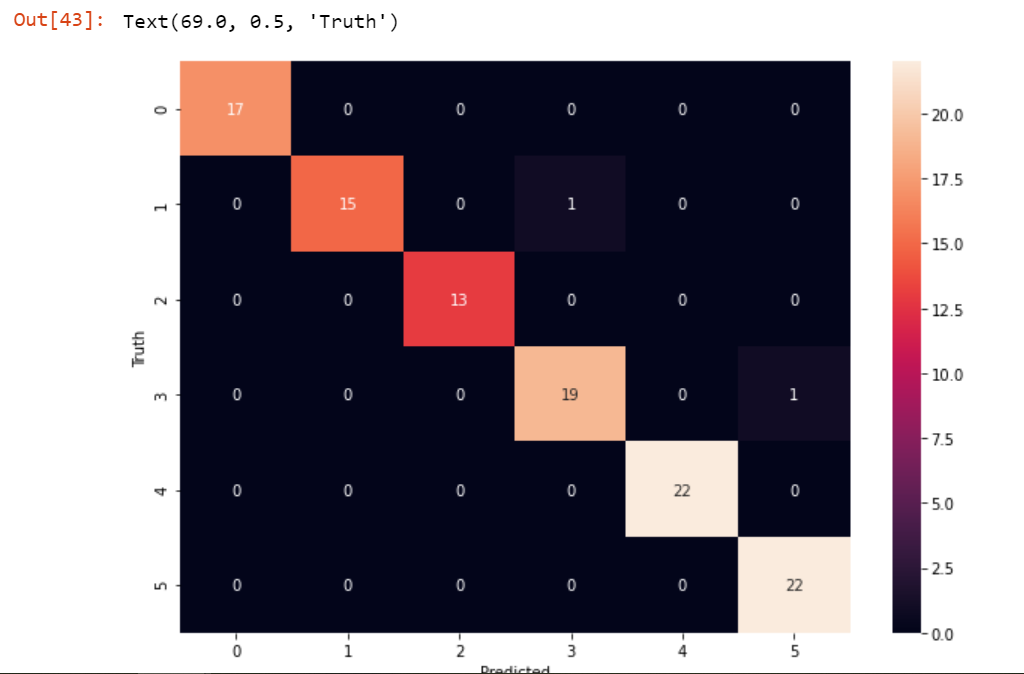


* A pontosság 100x100-as képekkel **93,4%** lett(utána már csak 90%)



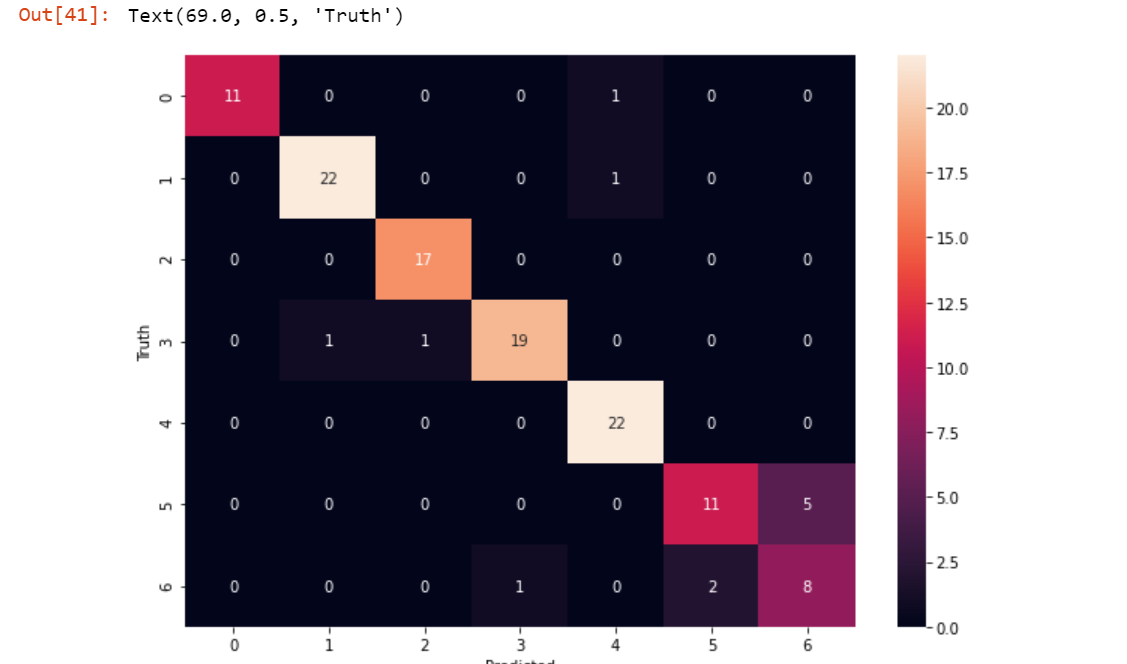
ML\_Lego\_classification:

* 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:



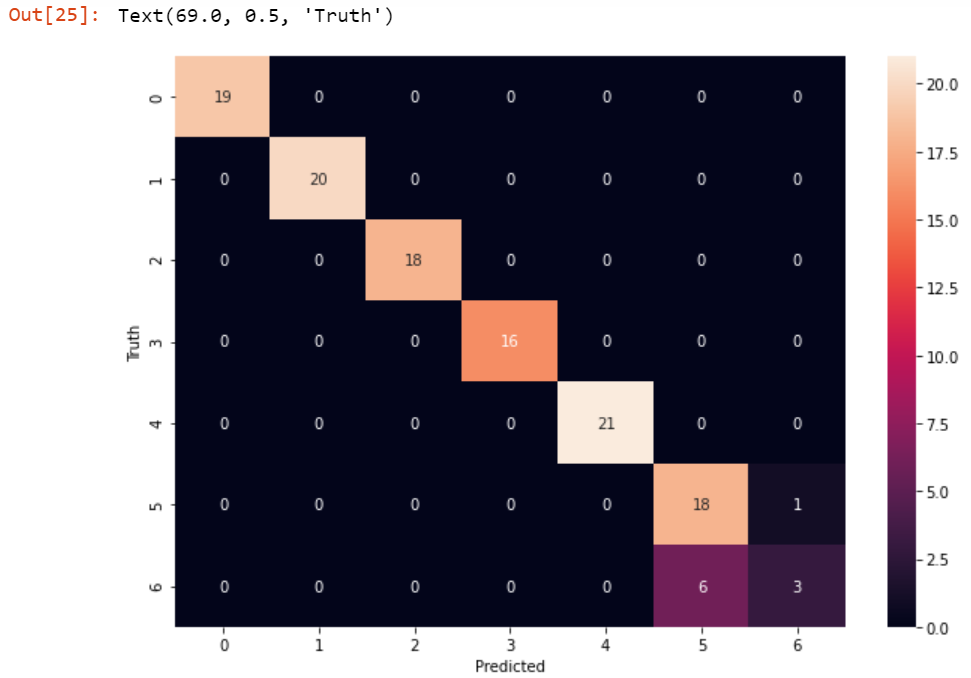


Látható, hogy az utolsó két állapot már szemmel láthatóan nehezebb megkülönböztetni, ami a programnak is nehéz feladat volt, lááthatóan itt elég sok tévedés történt.

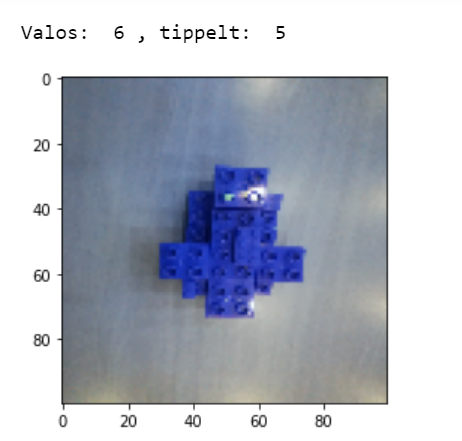
Deep learning:

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



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.

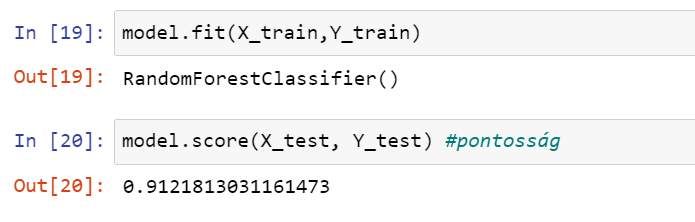


Lego error classification:

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.



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



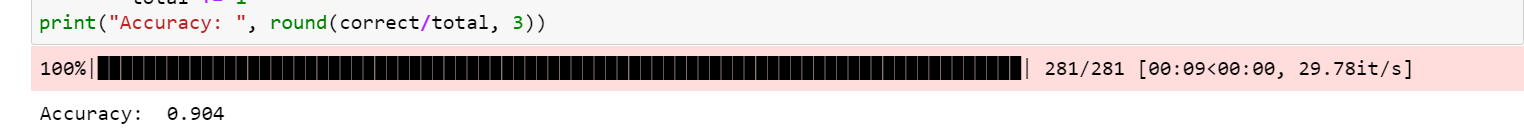
**Pytorch**-os NN

A neurális háló felépítése:

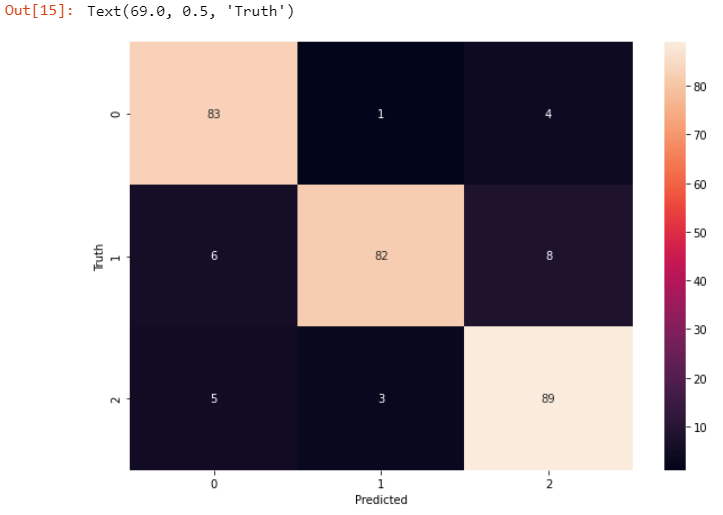


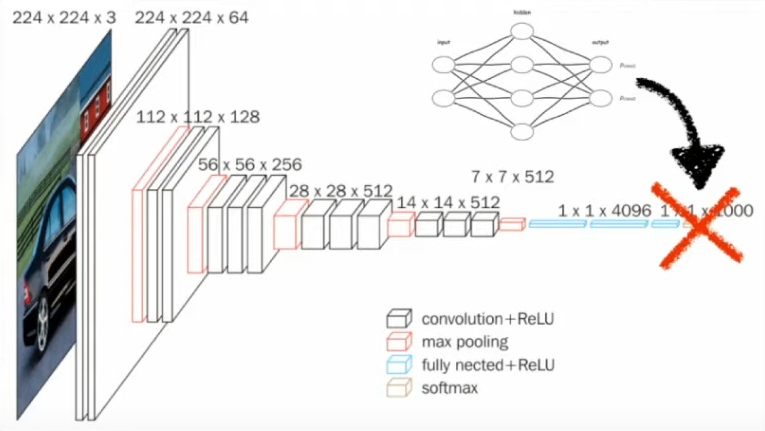
A neurális háló epoch után **90%** pontosságú lett:



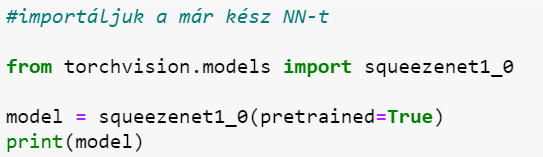


Conf. matrix:



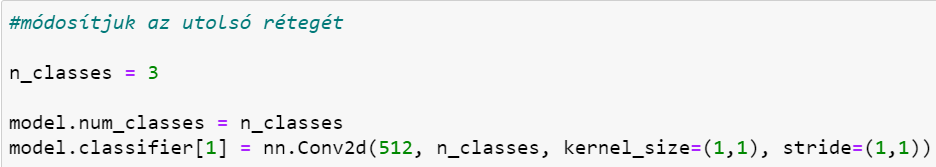
**Pre-trained Neural Network** használata, aminek a lényege az, hogy egy előre, rengeteg képpel betanított neurális hálót használunk fel, amelynek az utolsó layerét változtatjuk a sajátunkra, és ezen a hálón tanítjuk a képek alapján.

Ide illesztem a saját layerem

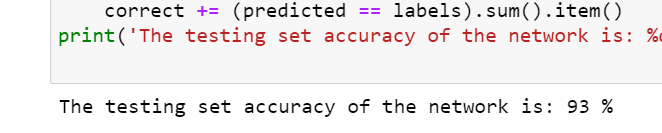


**SqueezNet**: pre-trained neural network, kisebb, mint sok más pre-trained NN, például az AlexNet-ben 50szer több paraméter van, és sokkal gyorsabb. Ez volt a célja a fejlesztésének, jelenleg az önvezető autóknál is használják.

Itt állítottam be a saját layeremet a model-nek:



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

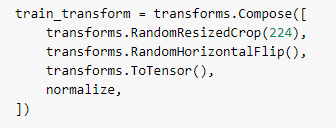


20 epoch után már 95%-os pontossággal dolgozott:



**Augmentáció és SqueezeNet:**

Adat aumentáció:



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 lényegesen lassabb lesz, de így tudott a pontosságán javítani:

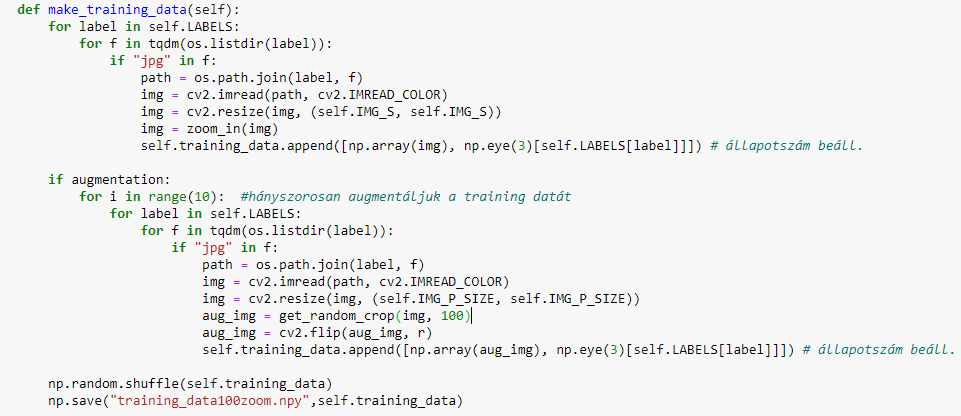




Írtam egy *RandomResizedCrop* szerű függvényt (itt ugye nem volt használható ez, mert másképpen olvastuk be a képeket) és kipróbáltam a másik, teljesen saját neurális hálón, így ~1500 helyett ~7000 képen tanulhat a háló:

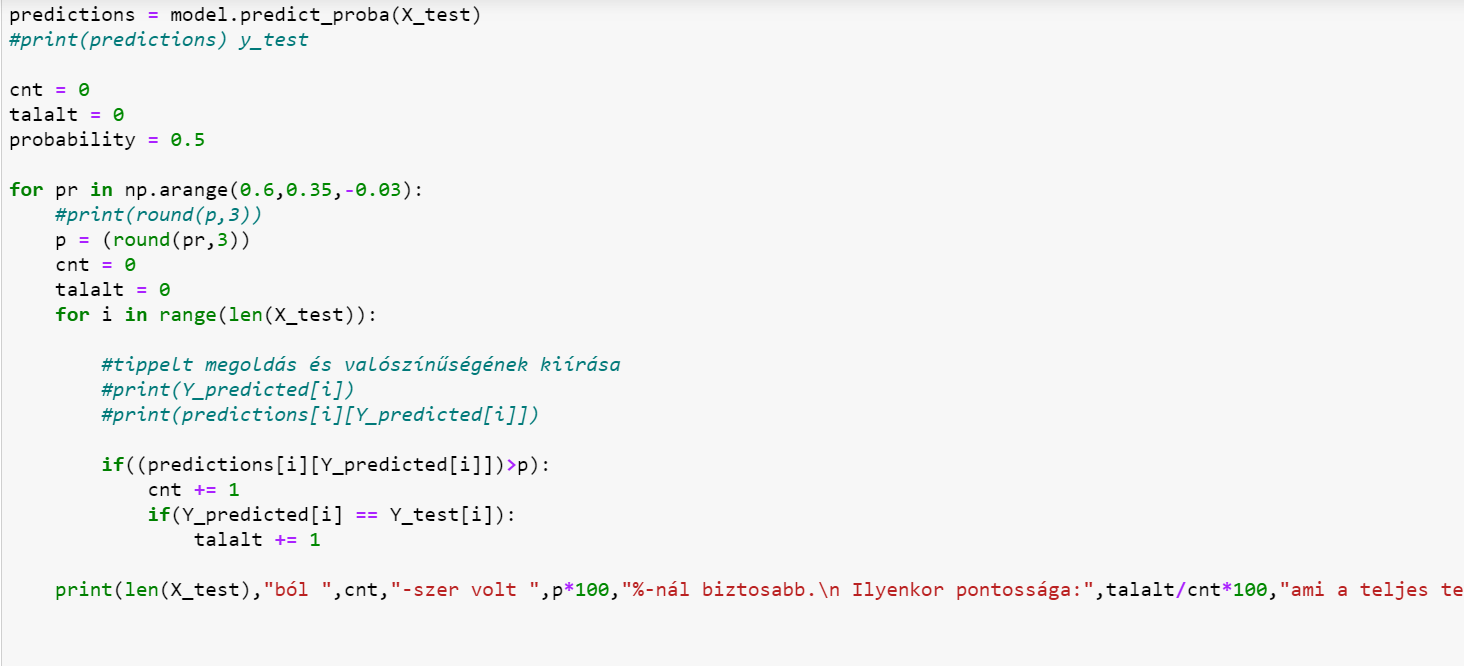


Képbeolvasás művelete:

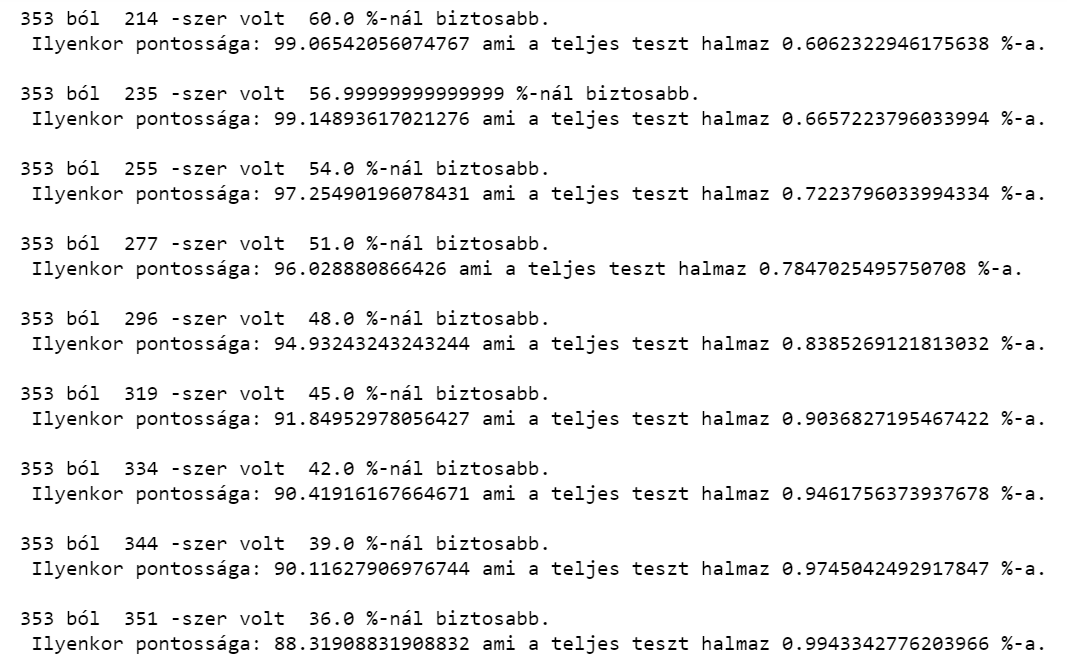


Adat-augmentáció

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:



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



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:**

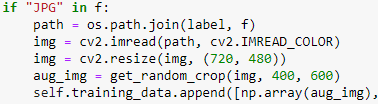
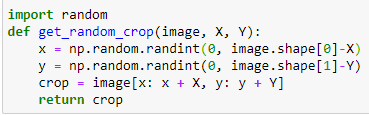
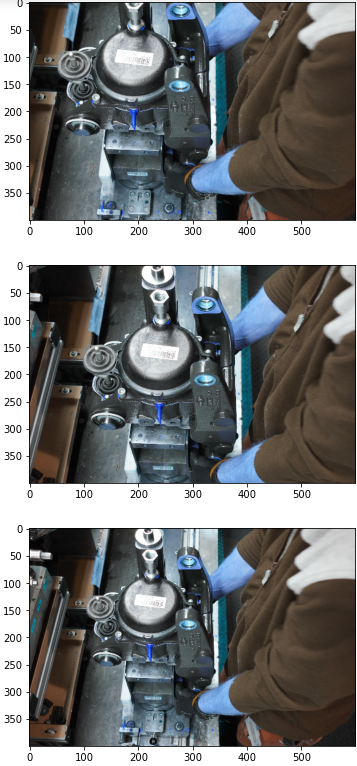
Az egyik állomáson található képeket elemeztem, amit a következő 10 osztályra bontottam:

|  |  |  |  |
| --- | --- | --- | --- |
| 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:

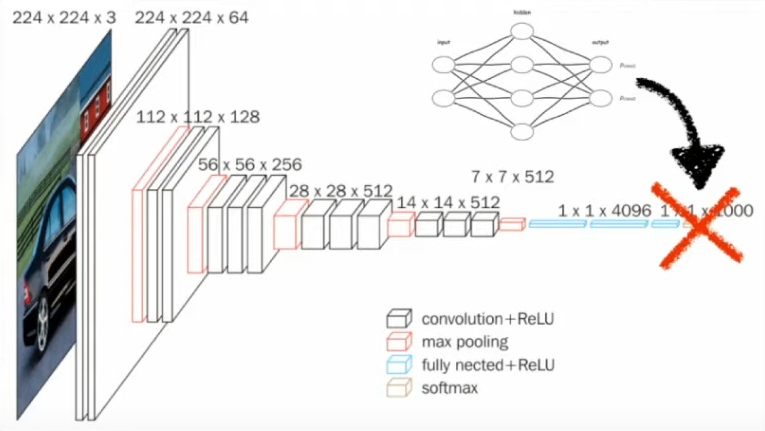


Például ebből a képből ezt a hármat készíti:

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.

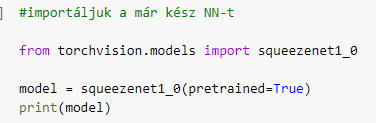


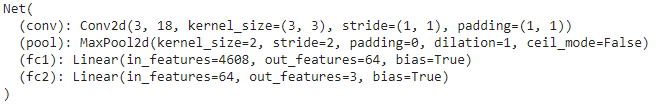
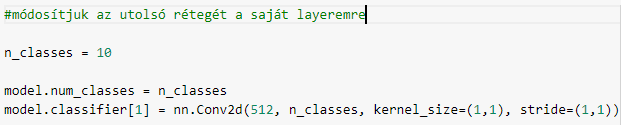
A háló forráskódjának főbb részei:

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.



Importálom a SqueezeNet-et:



Saját layerem, amit hozzáadok, és a hozzáadás lépése:

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

