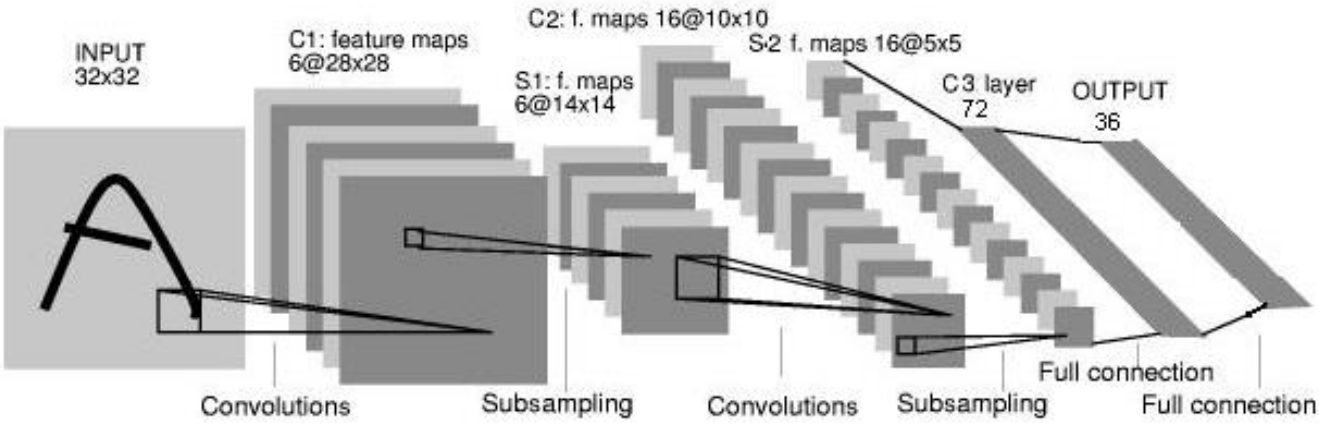
Témalabor

Knorr képosztályozás neurális hálókkal

Nagyobb témakörök

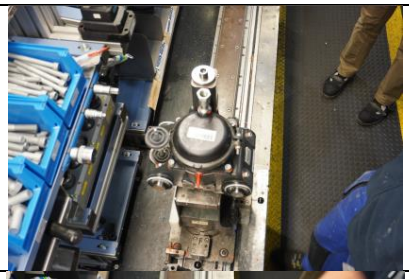
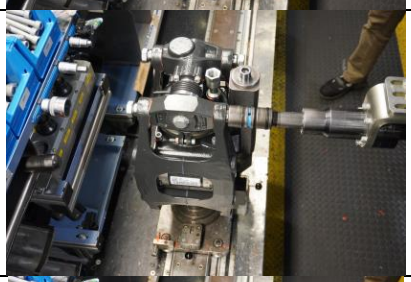
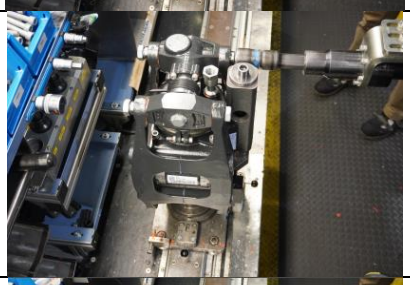
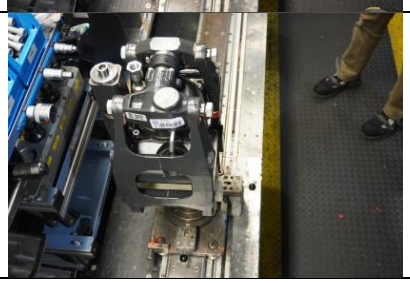
- Feladat bemutatása
- Képfeldolgozás, augmentáció
- Képosztályozás pretrained neurális hálóval (SqueezeNet)
- Képosztályozás saját neurális hálóval, optimalizálás



Feladat bemutatása:

A félévben képosztályozással foglalkoztunk, a félév korábbi szakaszában saját képhalmazokkal dolgoztunk, például szivacs-modellel és Lego-modellekkel. Ezekkel kapcsolatban ismerkedünk meg a neurális hálókkal és a Deep Learninggel. Megismerésükre azért volt szükség, mert az elején az egyszerű Machine Learning nyújtotta megoldások nem voltak elég precízek. Ezután áttértünk a neurális hálókra, amivel nagyobb pontossággal tudtuk megoldani az osztályozást. A félév során sok Deep Learninget kiegészítő technikával (például augmentáció, pre-trained neural network, konvolúciós neurális hálók, optimalizálás menete) megismerkedtünk. A félév végén megkaptuk a Knorr által elvégzett munkafolyamatok képeit, amin az eddig szerzett tudásunkat hasznosítottuk.

A Knorr öt állomáson végez összeszerelést, ebből az egyik állomáson található képeket elemeztük, amit a következő 10 osztályra bontottuk:

1		6	
2		7	
3		8	
4		9	
5		10	

Képfeldolgozás:

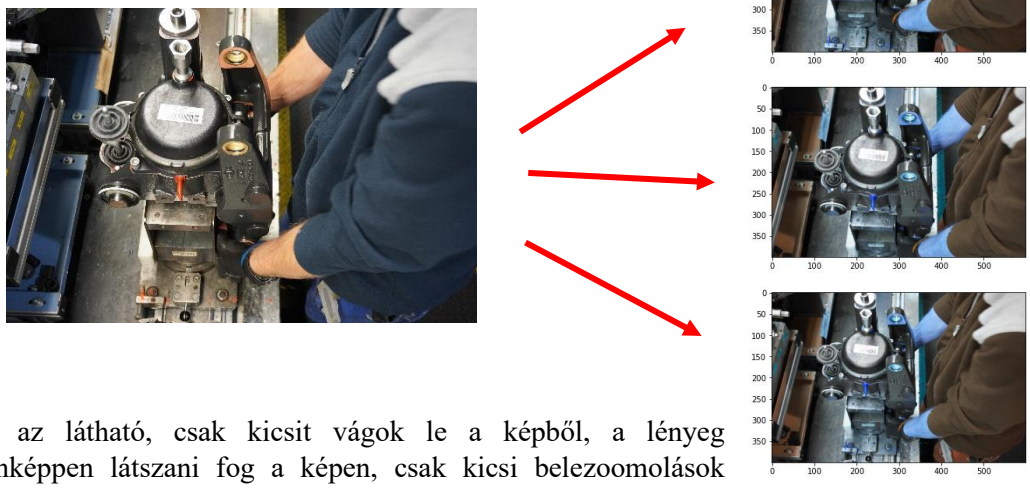
A kapott fényképeknél probléma volt, hogy az egyes osztályokról nem ugyan olyan mennyiségű képet kaptunk, ami miatt a képhalmaz aránytalan volt. Ahhoz, hogy a képhalmaz osztályaiban az elemekből körülbelül ugyanannyi legyen (a neurális hálók tanításánál fontos a kiegyensúlyozottság), ehhez többféle augmentációs megoldást is használtunk. Az augmentálást nem csak a képhalmaz kiegészítéséhez érdemes használni, hanem ahhoz is, hogy kisméretű adathalmaz esetén megnöveljük az adathalmaz méretét, így növelve az általánosító képességet és a pontosságot is.

Az általunk írt egyik adataugmentációs függvény például kicsit belezoomol (`get_random_crop`) a képekbe, ami itt látható:

A függvény megkapja az átalakítandó képet, a várt szélességet, magasságot és visszaadja a zoomolt képet.

```python
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
```

Például ebből a képből ezt a hármat készíti:

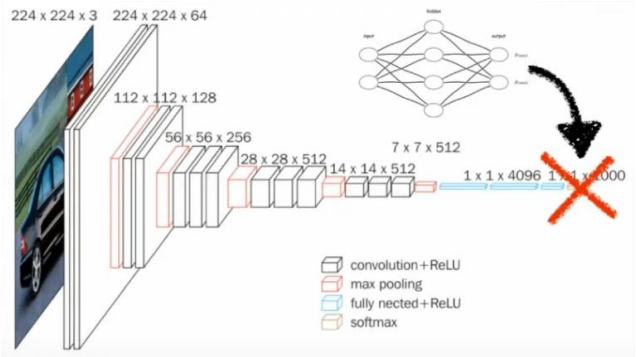


Amint az látható, csak kicsit vágok le a képből, a lényeg mindenképpen látszani fog a képen, csak kicsi belezoomolások történtek (a kép nem kerül átszínezésre, ez csak a megjelenítő felület sajátossága).

Ezzel a technikával a képhalmaz osztályait kiegészítettem ~50-50 képesre.

Képosztályozás pretrained neurális hálóval (SqueezeNet):

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



A háló forráskódjának főbb részei:

Adatbeolvasás – itt importáljuk 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, ami egy előre több tízmilló képen tanult neurális háló, ami a képfelismerési hálónk alapja:

```
]  #importáljuk a már kész NN-t

   from torchvision.models import squeezenet1_0

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

A kész neurális háló utolsó rétegére beillesztem az általam készített saját layert, ami erre a feladatra egyedi (osztályok száma, kapott képméret, kernel méret, padding, max-pooling) :

```
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, ami azt jelenti, hogy nagyjából 1-2 képet tévedett:

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

Képosztályozás saját neurális hálóval, optimalizálás:

A pre-trained neurális háló lassabban tanul, mert rengeteg layerből áll, ezért megnéztünk, hogy egy általunk írt háló az, hogy teljesíti ezt a feladatot. Ezért a korábban megtanult optimalizálási lépéseken végig mentünk, hogy egy általunk készített optimális hálót kapjunk. A lépések a következők, aminek a végrehajtását hasznosnak/fontosnak találtuk:

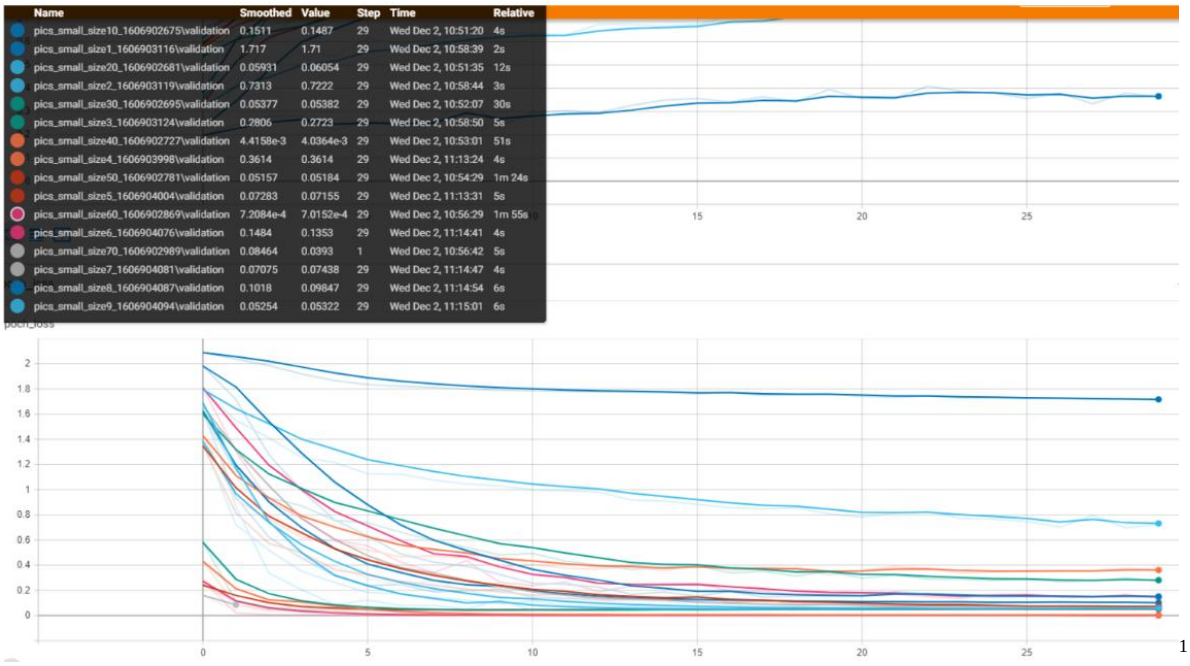
1. Képméret ellenőrzése
2. Batch size mérete
3. Legjobban teljesítő egyedüli layer megkeresése
4. A layer többi paraméterének beállítása

1. Képméret ellenőrzése:

A képméret ellenőrzése arra szolgál, hogy megtalálja a legkisebb képméretet, amin a feladat még megoldható jó megoldással. Ennek előnyei:

- A neurális hálónak kisebb adatot kell végig vinnie a hálóján, ezért gyorsabb lesz a következtetés és a tanítás is
- A neten akarjuk a kamerával készített képeket átküldeni és ha előtte redukáljuk a méretét a képnek, akkor a hálózati forgalmat se terheljük túl.
- Tárolás esetén nem kell gigabájtokon tárolni pár ezer darab képet.

Az alábbi eredmények jöttek ki az optimalizálás futtatása során:

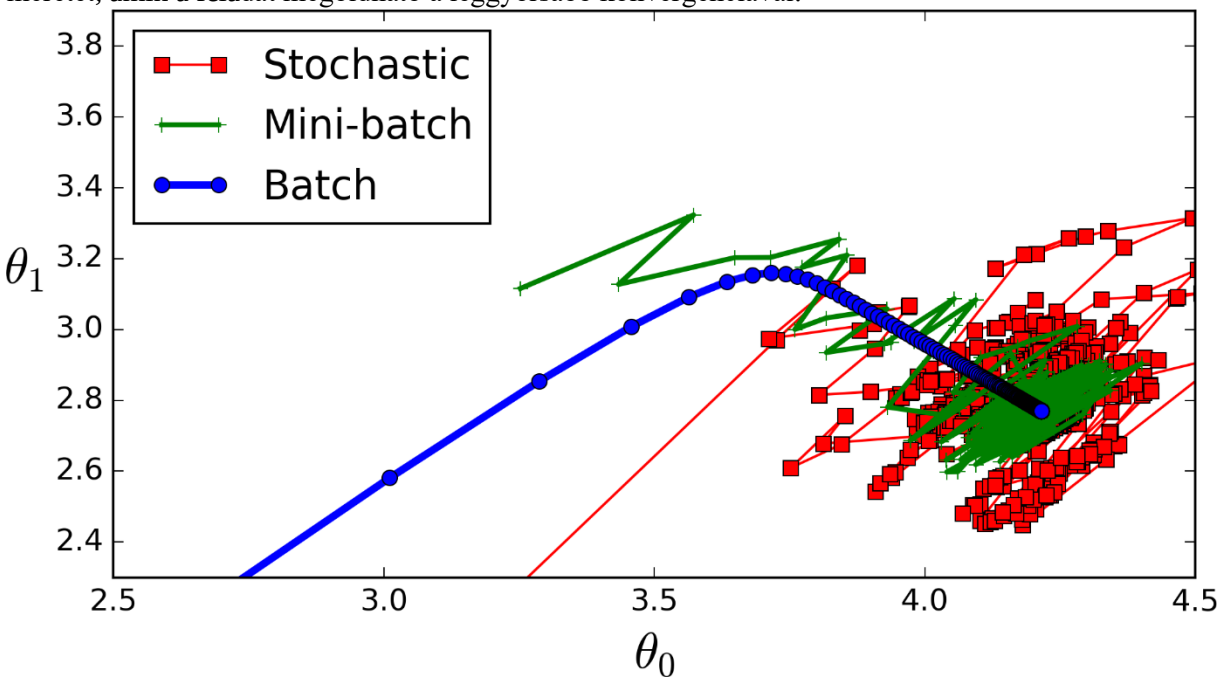


Az 5x5-ös képtől már az eredmény egész jó lett, de a biztonság kedvéért ennél magasabb felbontást választottunk végül. Ennél a képnél az ideális méretnek az előbb írtak alapján 10x10 pixeles képet választottuk, mivel a beszúrt képen látszik, hogy ezzel már elég jó eredményt lehetett produkálni.

[1] A képen a loss-t tüntetjük fel, tehát minél kisebb az érték annál jobb.

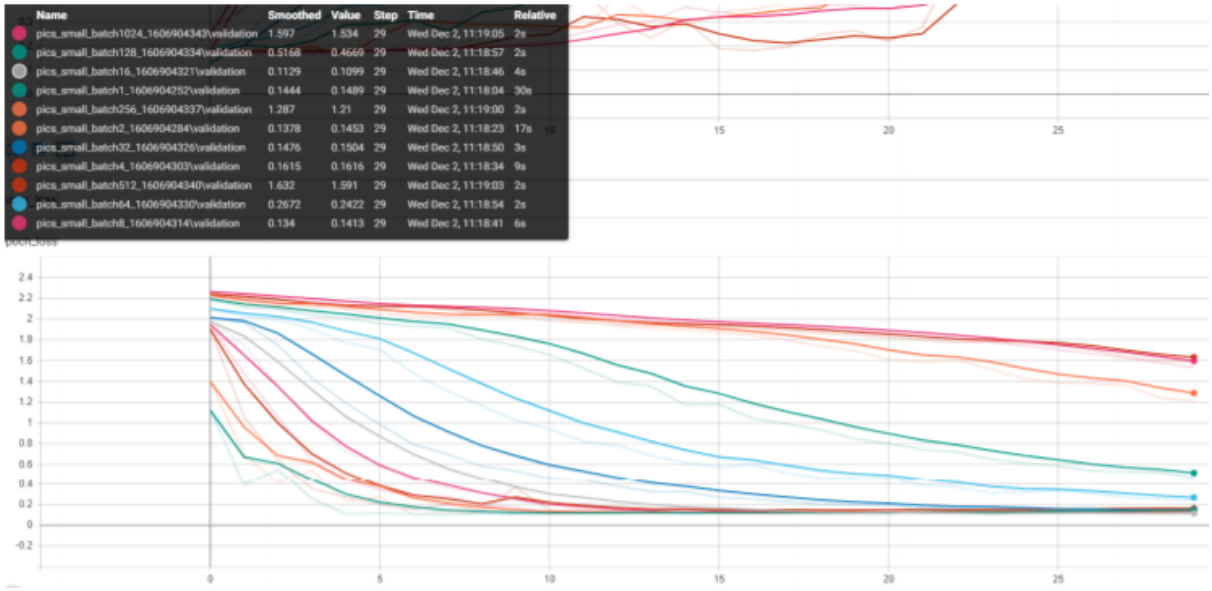
2. Batch size mérete:

A Batch size méretének optimalizálása arra szolgál, hogy megtalálja a leghatékonyabban batch méretet, amin a feladat megoldható a leggyorsabb konvergenciával.



A fent levő képen látszik, hogy különböző fajta (színű) vonalak láthatóak, amik ugyanaz a pont felé tartanak. A ponthoz való tartás típusai például: lassú, összevissza ugráló, egyenletes, stb..

Tapasztalataink alapján általában az adathalmaz méretétől függ az optimális batch size és sokat segít. A tesztek alapján végül a 16-ost választottuk, ami itt látható:



3. Legjobban teljesítő egyedüli layer megkeresése:

A jó layerek megtalálása arra szolgál, hogy megtalálja a legjobban használható layereket, amin a feladat legjobban teljesít. Ennek az az előnye, hogy ezután a jól teljesítő layerek kombinációit használhatjuk, és nem csak vaktába kell találhatnunk melyik lehet a jó.

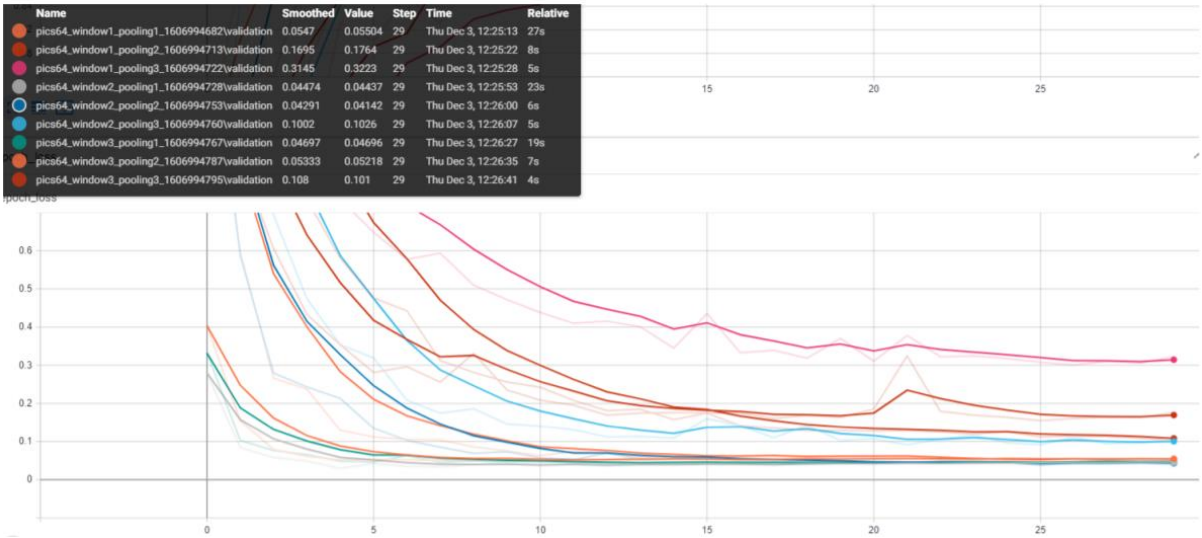
Összefoglalása az eredményeknek:

- 2-2048 iteráció számokkal dolgoztam (kettő hatványait használva)
- összeségében kettőt leszámítva az összes elég jól teljesített ez a kettő a 2 és a 4-es layer volt.
- A legjobban teljestő az 1024-es volt és a nagyobbak jobban teljesítettek általában.
- A 64-es layerrel dolgoztam tovább, mivel elég jónak bizonyult és gyors is, mert nem kell annyiszor megcsinálni, mint az 1024-esnél.
- Végül a pontosság itt 98% lett, tehát egy layerrel is ezek szerint elég jó eredményt lehet elérni nem csak kettővel ennél a feladatnál. Ezért nem használtam több layert a továbbiakban, hogy minél gyorsabb és egyszerűbb legyen a neurális háló. (Vannak olyan feladatok amikor érdemes azokat is megnézni)

4. A layer többi paraméterének beállítása

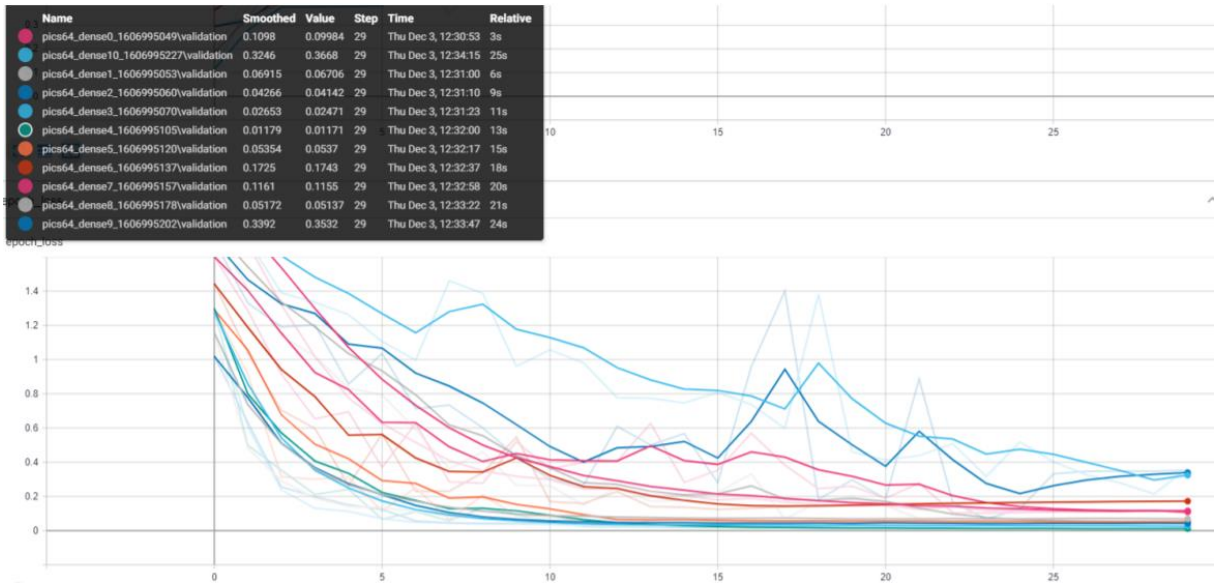
A többi paraméter beállítása arra szolgál, hogy megtalálja a maradék paraméterek közül a legjobbakat, amik javítanak a háló pontosságán. Ennek előnye, hogy a háló pontosabb lesz és jobban használható. Az általam módosított paraméterek a window és a maxpooling mérete volt, ezenkívül a dense layerek számát is idesorolnám, mivel képfeldolgozásnál jellemzően konvolúciós layereket használunk nem dense layereket, mert sokkal nagyobb a hatásuk.

Az eredmények a következők lettek:



Összefoglalása a window - maxpooling paraméternek:

- Itt már jelentősebb eltérések voltak köztük, de a 2x2-es window és a 2x2 polling volt a legjobb.
- A legrosszabb a 1 window és 3 pooling lett, amin kicsit meglepődtünk. Utána pedig a 2 window és 3 pooling. Ebből arra következtettünk, hogy valószínűleg nem éri meg nagyobb poolingot adni, mint windowt valószínűleg.
- Ami még meglepett minket, hogy az 1 window és 1 pooling ilyen jól teljesített. Valószínűleg azért lett ilyen jó, mert a kép is maga kicsi, tehát lehet most ennek annyira „nincs értelme”.



Összefoglalása a dense layernek:

- A legjobban teljesítő végül a 4 dense layer lett.
- A legrosszabb helyért pedig a 10 és a 9 darab dense layer vívott egy csatát.
- Úgy látszik ebből, hogy a 4 volt a legjobb tőle pedig jobbra, balra egyre rosszabbul teljesítettek.

Az optimalizáció végeredményének összefoglalása:

A végeredmény hálónak a pontossága **99%** lett (pár epochban a 100%-ot is elérte). Ez az eredmény jobb, mint a pre-trained háló eredménye. Az optimalizáció hosszadalmas és fáradalmas munka és még ezenfelül is biztos lehetne optimalizálni más feladatoknál (vagy akár ennél) de erre szerintünk elég volt és a végeredményt látva meg érte a fáradtságot.