Témalaboratórium dokumentáció
Czeglédi Levente – AJRUP9

Az első hetekben a megfelelő tudás megalapozásán dolgoztam, hisz sem ML-lel, sem Pythonnal nem foglalkoztam korábban.

Python alapok elsajátításához az alábbi forrásokat használtam:

https://infopy.eet.bme.hu/ - előadás anyagok bejelentkezés nélkül is hozzáférhetők, nagyon alapos, részletes

https://www.youtube.com/watch?v=JJmcL1N2KQs&ab_channel=TraversyMedia – ez a videó nagyon jó alapot adott, ez után konkrétabb témákra már könnyedén megtaláltam StackOverflow-n a válaszokat. Nézegettem pár további videót is a csatornáról és egyéb indiai csávóktól is, azok se voltak rosszak, de ahhoz sajnos nem tudtam megtalálni a linket ☹

Fejlesztőkörnyezet:

Alapvetően Jupyter Notebookot használtam a félév során végig, de Pythonhoz egy másik tárgy kapcsán használtam a PyCharm-ot is. A notebook mellett talán azért is érdemes dönteni, mert lépésenként lehet futtatni a kódot, sokkal könnyebb benne lépésekben haladni, de a PyCharm is nagyon kényelmes, főleg annak, aki korábban használt IntelliJ-t vagy Android Studiot.

https://www.youtube.com/watch?v=HW29067qVWk&ab_channel=CoreySchafer – Jupyter Notebook tutorial, a használathoz szükséges dolgokat ebből mind meg tudtam tanulni, azóta is elégnek bizonyult

ML alapok:

https://www.youtube.com/watch?v=gmvvaobm7eQ&list=PLeo1K3hjS3uvCeTYTeyfe0-rN5r8zn9rw&ab_channel=codebasics – első 17 videót megnéztem, az első pár példát le is kódoltam magam is, de mindig szenvedni kellett azzal, hogy az Excel nem akarta a csv-t megfelelően kezelni. Ez alapján szerintem rengeteg dolgot meg tudtam tanulni, elmagyarázta érthetően a fickó a dolgokat, modelleket, valamint Jupyter Notebook gyakorlásra nagyon jól jött.

https://medium.com/@ageitgey/machine-learning-is-fun-80ea3ec3c471 - talán ez hozta meg a kedvem az egész témához a leginkább, végig is olvastam mindegyik fejezetét. Segített abban, hogy nagyvonalakban értsem azt, mi hogy működik, nem konkrét kóddal leírva, hanem konyhanyelven vannak megfogalmazva a dolgok benne.

Deep learning alapok:

https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi ez a videósorozat főleg a matematikai részét és az egész lényegét szemlélteti vizuálisan, ugyan még nem deep learninggel foglalkoztunk, de nagyon meghozta a kedvem hozzá ez a playlist.

Szivacsos képek




Konzultáción beszéltük, hogy következő alkalomra esetleg viszünk lego-t és abból építünk modelleket, de nagyon bepörögtünk páran és szivacsokból csináltunk pár nagyon alap modellt, amin az eddigieket ki tudtuk próbálni. Ezekről készítettem fényképeket és feltöltöttem a Teamsen megosztott Drive mappába. (Minden fázisról van oldalt egy kép)

A képek készítésénél tulajdonképpen csak azt reméltem, hogy egyáltalán a modell tud valami értelmeset mondani ennyi (fázisonként 70) kép alapján, ezért nagyon hasonló képeket csináltam, mindet egy eszközzel, azonos háttér és fényviszonyok mellett, a kamerát alig mozgatva. Próbáltam a lehető legélesebb képeket csinálni és nagy felbontásban, azt hittem, ez sokat fog számítani. Mint kiderült, nem nagyon számít egyik sem, már 100x100-as mérettel is nagyon pontosan meg tudja mondani a modell – bár ez csak a későbbiekben derült ki.

Ezekkel a képekkel próbáltunk egy random forestes modellt de én a beolvasással megakadtam, így azt a kódot használtam és formáztam át, amit Mesterházi Marci mutatott konzultáción és ott még tovább javították Istvánnal (konzulens).

Ezzel a modellel végül sikerült nekem is tesztelnem, de sajnos a szivacsos képek eredményeit nem találtam meg. Marcinak nagyon jó eredményei voltak (közel 100%, de akár 100% is), itt arra jutottunk, hogy valószínűleg a képek egyszerűsége miatt tudott ennyire pontos lenni.

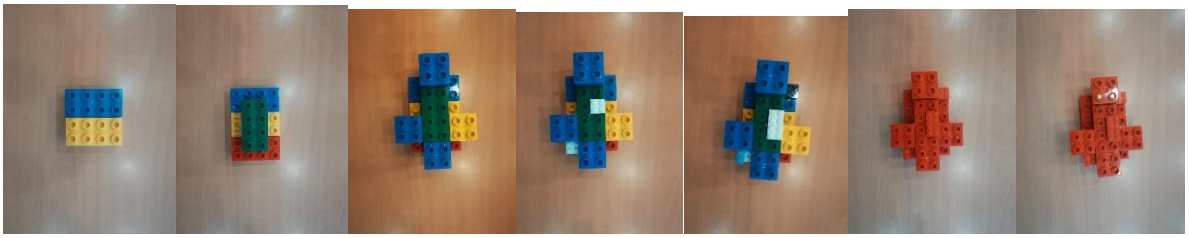
A Random_forest.ipynb fájlban van a beolvasás és maga a modell is. Jelen pillanatban a fájlban a legújabb, hibadetektációs képek beolvasása és kiértékelése van bent, confusion_matrixszal és az eltévesztett képek kirajzolásával.

Lego-s modellek

Végül megcsináltuk a lego-s modelleket, ezekről készítettem képet és felraktam a drive mappába. Itt többféle galéria készült:

Az egyik, amiben 7 lépésben „szerelünk” össze egy modellt, itt vannak nagy és kicsi eltérések is egyes fázisok között – ki szerettük volna próbálni, mennyire érzékeny a modell arra, ha nem csupán egy újabb alkatrész kerül rá, hanem egy meglévő máshova kerül vagy szabad szemmel is alig észrevehető elsőre a változás. Erről a galériáról többféle eszközzel is készült kép – még profibb géppel fotós kolléga által is - de hátránya maradt, hogy egyféle háttér előtt készült mind (bár ezt a különböző eszközök kicsit különböző színűvé tették más fehéregyensúly-beállítások miatt). Igyekeztem úgy csinálni, hogy ne pontosan ugyanonnan fotózzak, forgattam, döntögettem a telefonom, esetenként bele is nyúltam a képbe (hátha a gyárban is előfordulhat ilyen) vagy odatettem valamit a kép szélére. (~1000 kép, minden fázisról látható egy lejjebb). A 7. lépésben kíváncsiak voltunk, fel tudunk-e ismerni hibákat, így egy kis airsoft-golyót rátettünk a modellre (különböző helyekre néhány képenként – itt szórakoztam azzal is picit,

hogy a fényt ott csillantom meg rajta, ahol a golyó is van, hátha így nem veszi észre, illetve a nem hibás képen hátha a fényt golyónak hiszi).



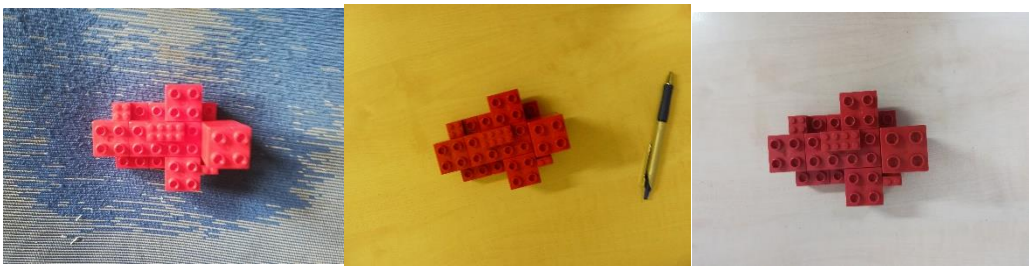
A másik, amelyik csak szinte ad hoc módon jött, hogy közel ugyanazok a fázisok, csak egyetlen apróság kerül rá pluszban, ez okozza a fázisok közti különbséget. (~200 kép) Ez Norbi ötlete volt alapvetően, ő foglalkozott ezekkel a képekkel mélyebben.

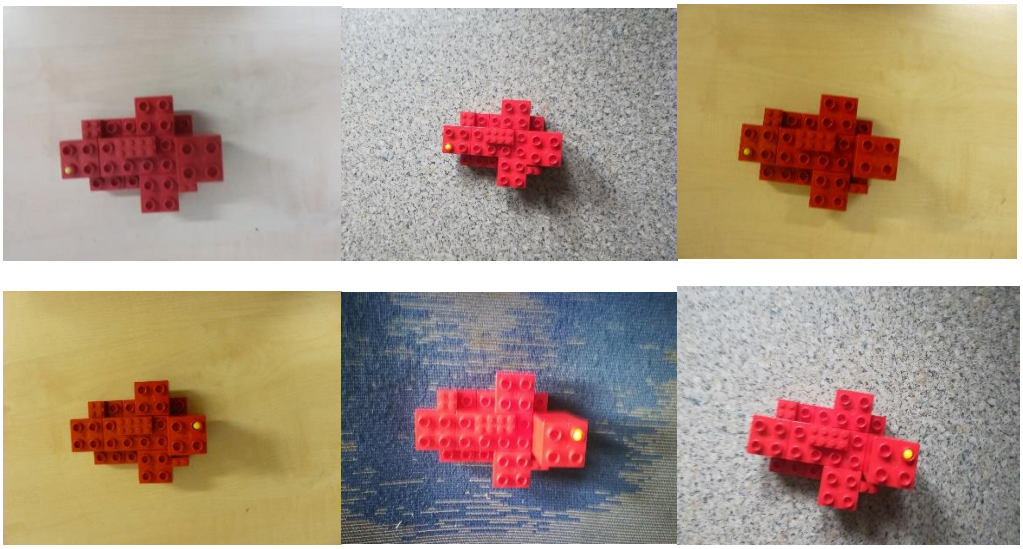
Kipróbáltam mindkét fajta képpel a Random forestes modellt, ~90-95% pontosságú volt (sajnos nincs a futás eredményéről képem ☹)

Végső modell

Nem sokkal ezután jött az a feladat, hogy piros pöttyöket kéne észrevenni a modellen és ez alapján jelezni hibát, ha kell. Ennek tesztelésére készítettünk egy eddigieknél is nagyobb datasetet, mindössze 3 fázisból. 4 különböző háttér előtt, 4 különböző eszközzel, hátterenként különböző mennyiségű képet (de fázisonként ugyanannyit és egyes fázisok között is közel ugyanannyi kép van) csináltam, majd ezeket feltöltöttem drivera.

A fázisok között csak annyi a különbség, hogy az alapszíntől elütő színű airsoft golyót hova rakunk a modellen (illetve van-e rajta egyáltalán). Összesen 1760 képet csináltam, hogy pontosabb lehessen az eredmény. Ezek közül pár látható alul, fázisonként 3.





Ezeket a képeket kipróbáltam a Random Forestes modellen, az alábbi eredményre jutottam (több tesztet csináltam, ezeket általgoltam):

100x100	RGB			Grayscale		
test nagyság	0.1	0.2	0.3	0.1	0.2	0.3
accuracy:	91%	89%	86%	83%	82%	78%

Feltűnt az, hogy ha több képünk van tanítani, jobb lesz az eredmény is. Ugyanakkor sokszor előfordult, hogy a 3-4 tesztfutam pontossága között 5-6%-os különbségek mutatkoztak, amit kicsit nagynak gondolok.

Készítettem teszteket 50x50, 75x75 és 200x200-as méretben is, ezekből a tapasztalatok:

- 50x50 már számottevően rosszabb volt, 75x75 már közel a 100x100-assal megegyező
- 200x200-as alig jobb, mint a 100x100, de lényegesen tovább tartott a beolvasás és minden más művelet is, így a 100x100-nál maradtam

Ezután kezdtem el deep learninggel foglalkozni.

Deep Learning

Alapból én TensorFlow-t választottam akkor, hogy mivel szeretnénk foglalkozni a továbbiakban, de sok nehézségem volt a telepítésével.

https://www.youtube.com/watch?v=wQ8BIBpya2k&list=PLQVvvaa0QuDfhTox0AjmQ6tvTgMBZBEXN&index=1&ab_channel=sentdex – ezt a playlistet végignéztem, habár nem minden részével foglalkoztam ténylegesen (pl. TensorBoardos optimalizációval nem foglalkoztam).

Mivel nagyon érdekelt a dolog, már azelőtt egészen sokat néztem belőle, hogy ténylegesen elkezdtem volna írni a programot.

Ahhoz, hogy a telepítés sikerüljön, újra kellett húznom a Pythont is (valahogy összeakadt egy 2.7-es verzió és egy 3.valami-s a gépemen). Ekkor jött még az a probléma, hogy a TensorFlow nem minden Python verzióval kompatibilis, így az akkori legújabb verzió Pythonból nem volt jó (3.9, előtte pár nappal jött ki), de itt már gyorsan sikerült újrarakni amit kellett és 3.8-cal ment tökéletesen.

Azelőtt, hogy ezt megtettem volna, gondoltam, megnézem a PyTorch-ot, hátha azzal nem lesz ilyen jellegű probléma.

https://www.youtube.com/watch?v=BzcBsTou0C0&list=PLQVvvaa0QuDdeMyHEYc0gxFpYwHY2Qfdh&index=1&ab_channel=sentdex – ezt a videósorozatot néztem végig belőle, de annyira nem tetszett. A TensorFlow-s tutorial sorozat sokkal jobbnak tűnt és ugyanolyan pontosan elmond mindent, így vissza is álltam arra és a TensorFlow-ra, amikor azt sikerült feltelepíteni.

A PyCharm nekem nagyon furcsának is tűnt, sok hibába is ütköztem, úgyhogy a következő konzultációig nem foglalkoztam vele, hanem áttértem az immáron újrahúzott TensorFlow-ra.

A PyTorchos próbálkozásom megtalálható a PyTorch vol.1.ipynb-ben, a végére sikerült kicsikarni talán belőle valamit az mnist-es datasetre (a beolvasást nem tudtam megcsinálni, ezt láttam a tutorialban), de sok hiba volt/van benne.

Az első TensorFlow-s modellem valamiért nem akart tanulni, minden epoch után ugyanazokat az értékeket írta ki. Itt több probléma is volt a modellel, gondoltuk, hogy a categorical_crossentropy és a softmax activation akadhatott össze, de ezek későbbi modellekben teljesen jól működik azóta is nekem együtt. Probléma volt még, hogy az utolsó layernél a kategóriák számát valamiért nem akarta elfogadni, mondván, hogy a labelek nem terjednek akkora számig, mint a kategóriák az utolsó layerben.

Mivel nem sikerült nekem megoldani, hogy működjön, Norbinak pedig már volt egy működő modellje, abból próbáltam rájönni és azt átírni megfelelően, de nem jártam sikerrel.

A 'végleges' próbálkozás a tensorflow.vol1.ipynb-ben van, de maga a kód nem működik (valami jól).

```
Epoch 17/30
29/29 [==============================] - 1s 51ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 18/30
29/29 [==============================] - 2s 52ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 19/30
29/29 [==============================] - 1s 50ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 20/30
29/29 [==============================] - 1s 52ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 21/30
29/29 [==============================] - 1s 51ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 22/30
29/29 [==============================] - 1s 50ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 23/30
29/29 [==============================] - 1s 51ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 24/30
29/29 [==============================] - 1s 50ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 25/30
29/29 [==============================] - 1s 51ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 26/30
29/29 [==============================] - 1s 50ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 27/30
29/29 [==============================] - 2s 52ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 28/30
29/29 [==============================] - 2s 52ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 29/30
29/29 [==============================] - 1s 51ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
Epoch 30/30
29/29 [==============================] - 1s 51ms/step - loss: 0.0000e+00 - accuracy: 0.3243 - val_loss: 0.0000e+00 - val_accura
cy: 0.3456
```

A megoldás végül egy másik tutorial alapján írt modell volt, ott működött minden, a szivacsos képeken 100%-os eredményt produkált, a Lego-s modelleken pedig 90 körül, de még valószínűleg tudott volna hova tanulni a rendszer további epochokkal és optimalizációval sem foglalkoztam semmit, csak örültem, hogy végre van egy működő hálóm.. 😊

Sajnos a következő hetekben nem volt időm foglalkozni a témalaborral pár videó megtekintésén kívül ☹. Annyit mégis foglalkoztam a dologgal, hogy megnéztem pár előadást az AutMI szemináriumból és többnyire végignéztem az NVIDIA deep learning konferenciát. Utóbbi nagyon érdekes volt szerintem, meghozta a kedvem, hogy foglalkozzak a témával, de még másfél-két hétig nem tudtam, mert ZH hegyek jöttek, majd hazaköltözés.

Ezekben a hetekben megnézegettem a Laci által küldött anyagokból párat:

https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python - többnyire alapabb dolgok vannak benne, amikkel korábban találkoztam, de kimondottan jól el van magyarázva és mutatott újdonságokat még bőven. Az overfitting kimondottan jól el van magyarázva, bemutatja a Dropout megoldást is, amivel korábban én nem találkoztam (de később nem is használtam végül).

https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063 - ez igazából már előfordult korábbi videókban, de szerintem hasznos volt elolvasni.

Megnéztem továbbá ezeket is, itt a működés van elmagyarázva részletesen és nagyon érthetően:

https://www.youtube.com/watch?v=CqOfi41LfDw&list=TLPQMjMxMTIwMjDPYpMbyVR5KA&index=1 és a Backpropagation videót is megnéztem.

A következő konzultáción azt beszéltük, hogy GPU alapú TensorFlow-t fogok kipróbálni és esetleg data augmentationt, ebből végül az utóbbi lett.

Augmentáció

Az augmentáció módszereit kerestem először, erre a Kerasnak van elég jó API-ja, ezt próbáltam megtanulni, olyan képeket létrehozni augmentációval, amik „valósághűek” és akár mi is készíthettük volna őket, ha több fényképet csinálunk. Erre az alábbi beállításokat találtam a legjobbnak:

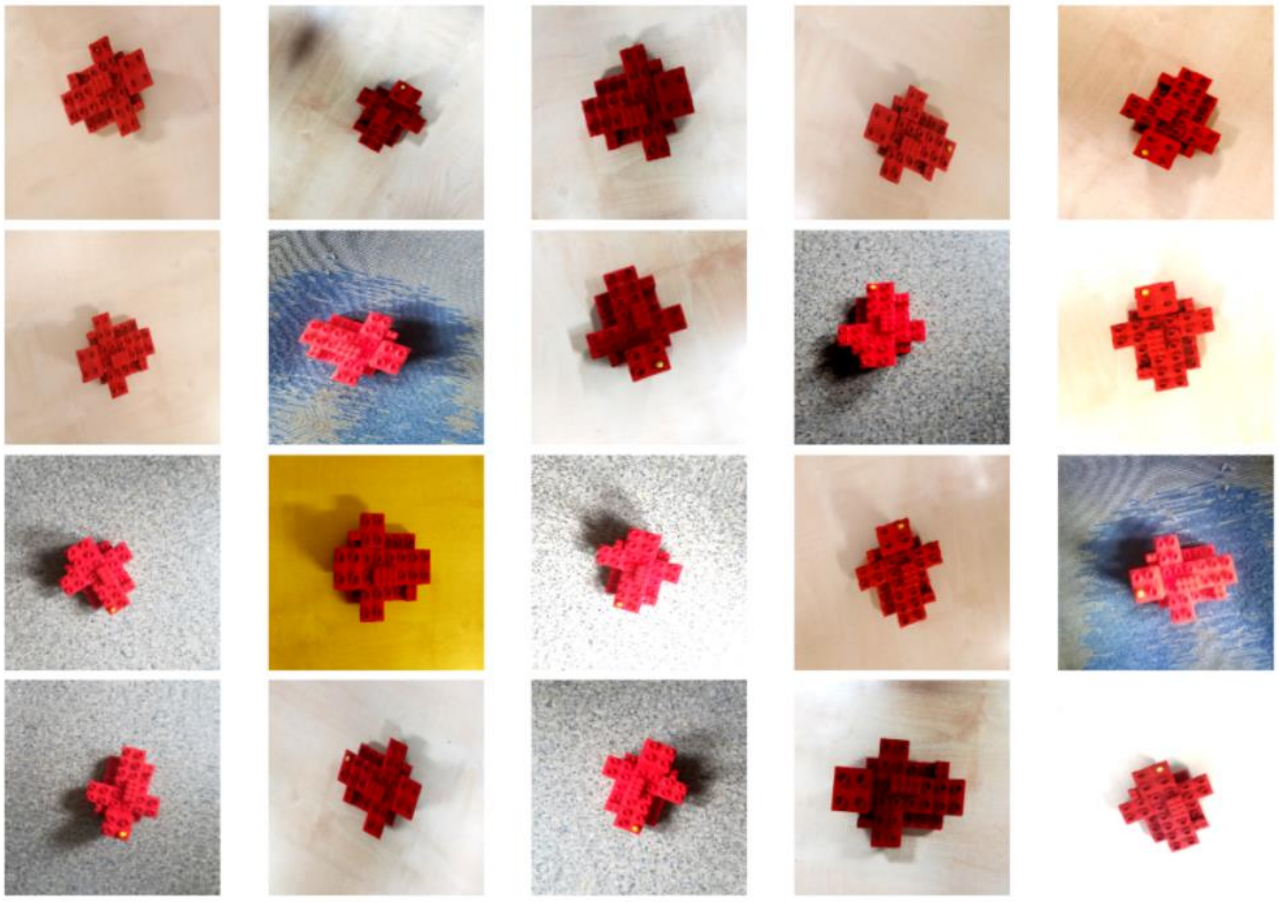
```
train_datagen = ImageDataGenerator(rescale = 1./255, rotation_range= 30, width_shift_range = 0.05, height_shift_range  =0.05,
                        zoom_range = [0.8, 1.2], brightness_range = [0.8, 1.2],
                        channel_shift_range= 60, horizontal_flip = True, vertical_flip = True, fill_mode = "reflect")
```

Van pár olyan paraméter, ami nem annyira magáért beszélős, ezeket leírom, miért választottam arra, amire:

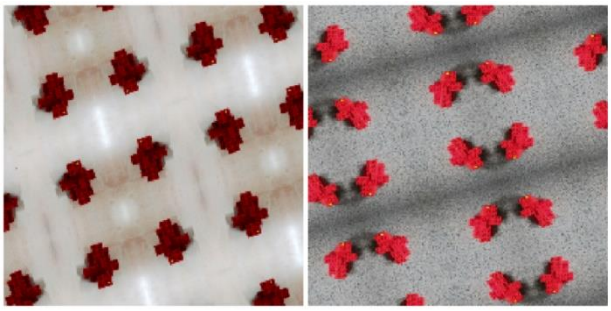
- A rescale hatására lesznek korrektek a színek, ha nem tesszük ezt meg, akkor kb. agyonexponált képeket kapunk
- A width és height shift azért lett pici, mert az eredeti képeken is elég változatos helyeken bukkan fel a modell, ne toljam le róla véletlen se (főleg azért, mert a forgatás és zoom még változtat a képen)
- A zoom_range egy [min, max] formátumot követ, de fontos, hogy nem tartja meg a képarányt, a két tengely mentén véletlenszerűen zoomol az adott tartományból (ez mondjuk szerintem elég rossz, hogy így van, bár kétségkívül többféle kép állítható elő vele)
- a channel_shift_range meg kell mondjam, nem jöttem rá, pontosan mit csinál – azt írták róla, a színcsatornákat manipulálja, de én azt vettem csak észre, hogy a hátteret tudja változtatni, többek közt blank fehér hátteret is csinált.
- a fill_mode határozza meg, mi legyen akkor, ha a képbe olyan rész is bekerül, ami előtte nem volt ott (shift, rotation, zoom > 1.0 esetén). Erre a reflectet ítéltem meg a legjobbnak, ami ilyenkor tükröz a kép határára, ezáltal természetesnek tűnik a kép (bár vannak anomáliák, ha nagy a zoom)

Az augmentációs kísérletek kódját az augmentation.ipynb tartalmazza.

Pár kép mintaként látható az augmentálás után a következő oldalon:



Le is lehet menteni az így generált képeket – konkrétan csináltam egy modellt, aminek a 'tanítása' menti le a képeket (a generator flow-ja csak akkor indul meg, ha én léptetem manuálisan az iterátort, vagy tanítom a hálót, utóbbi egyszerűbbnek tűnt). Ezt biztos lehet másképp is, de a felhasználási célnak megfelel ez is 😊.



Fill_mode: 'reflect' magic (zoom_range: [3:5])

Ehhez tartozó kódrészlet:

```
train_generator = train_datagen.flow_from_directory(directory=path, target_size = (IMG_SIZE,IMG_SIZE),
                        classes = ['1','2','3'], batch_size = 10, subset='training',
                        color_mode = 'rgb', shuffle = True,
                        save_to_dir=savepath+'augmented_fulltrain', save_format="png", save_prefix = "aug")
```

Konzultáción beszéltük, hogy én haladok ebben az irányban tovább. Mielőtt folytattam volna, megnéztem pár videót és kutakodtam a témában:

https://www.youtube.com/channel/UC4UJ26WkceqONNF5S26OiVw - ezen a csatornán találtam rengeteg hasznos dolgot, nagyon jó magyarázatokkal (innen volt belinkelve a Conv2D és MaxPool magyarázat is Teamsre)

https://www.youtube.com/watch?v=qFJeN9V1ZsI&t=4043s – ez egy az előző csatornát nyomó csaj által készített crash course, ebben is találtam sok hasznos dolgot, többek között, hogy hogyan is működnek ezek az ImageDataGeneratorok.

Írások:

https://heartbeat.fritz.ai/overcoming-overfitting-in-image-classification-using-data-augmentation-9858c5cee986

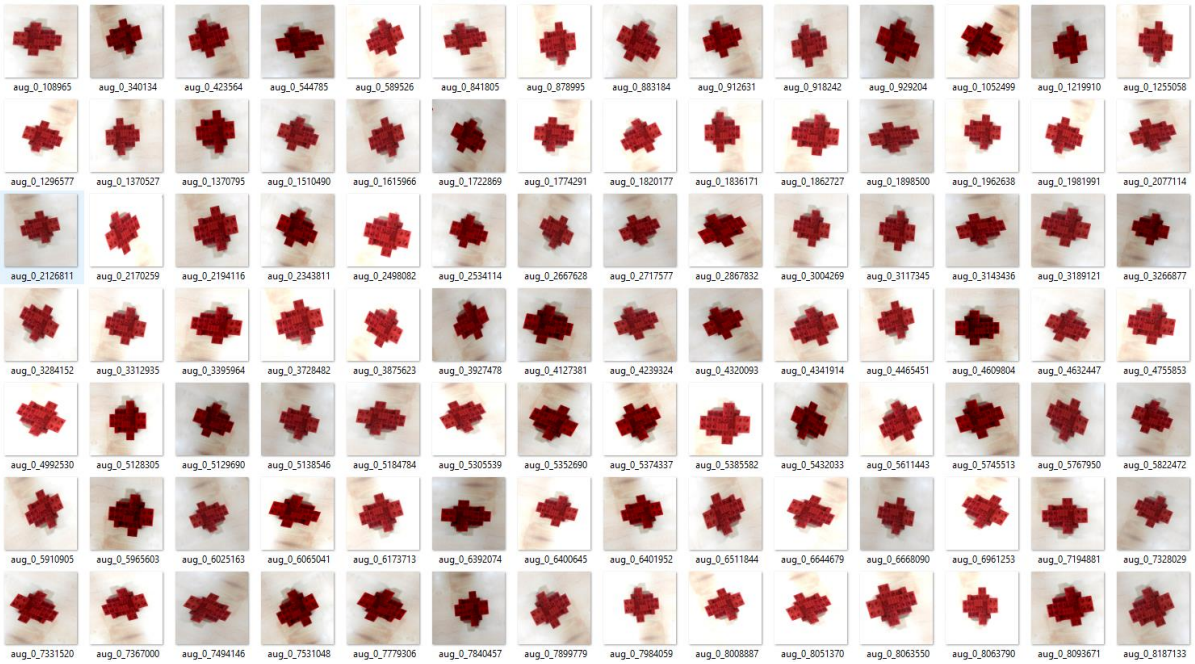
https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

https://keras.io/api/preprocessing/image/

Adathalmazok

Jó képek

Az elsőnél kiválogattam pár jó képet a fotók közül (körülbelül 300-at), a feltétele a kiválasztásnak az volt, hogy fázisonként egyenlő mennyiségű legyen (hátterekre figyelve), lehetőleg teljesen felülről készüljenek, vagy csak attól nagyon kis eltéréssel és ne lógjon bele semmi. Ezeket a képeket utána augmentáltam, csináltam belőlük összesen kb. 43 000 200x200-as verziót.



Ez a sok kép mind ugyan annak a kezdeti képnek az augmentációi (a kezdeti kép nincs köztük). Képenként pontosan 350-et készítettem 350 epoch segítségével (ez a folyamat körülbelül 2 órán át futott).

A kód a CNN_augmented.ipynb-ben található, maga a modell implementáció lényegtelen, csak képgenerálásra használtam – bár előtte ezt akartam betanítani is, de új modellt kezdtem, mert ez nem akart megfelelően működni (bár ennek csak a validation_split volt az oka, lásd később)

Ezeket az augmentált képeket egy mappába öntöttem be, de szerencsére könnyű volt utána szétválogatni, mert a Keras ad egy számot a képeknek annak megfelelően, hogy a beolvasáskor hányadikként következtek (itt a 0. kép szerepel).

Elkészítettem egy egyszerű hálót, ennek a kódja van alább:

```
model = Sequential([
    Conv2D(filters = 32, kernel_size = (3,3), activation='relu', padding = 'same', input_shape = (IMG_SIZE,IMG_SIZE,3)),
    MaxPool2D(pool_size=(2,2), strides = 2),
    Conv2D(filters = 64, kernel_size = (3,3), activation='relu', padding = 'same'),
    MaxPool2D(pool_size=(2,2), strides = 2),
    Conv2D(filters = 64, kernel_size = (3,3), activation='relu', padding = 'same'),
    MaxPool2D(pool_size=(2,2), strides = 2),
    Flatten(),
    Dense(units=3, activation='softmax')
])
model.summary()
```

A kékkel jelölt sorok néha ki voltak kommentezve a tesztek során, néha nem, kíváncsi voltam, egy olyan kis bonyolultságú háló mire képes, ami a kékkel jelölt sorok nélküli. Mielőtt tesztelhettem volna, beleütköztem egy újabb problémába:

Lehet validation splitet csinálni az ImageDataGeneratorokkal az alábbi módon:

```
train_datagen = ImageDataGenerator(rescale = 1./255, validation_split = 0.1)


valid_generator = train_datagen.flow_from_directory(directory=path, target_size = (IMG_SIZE,IMG_SIZE),
                    batch_size = 30, classes = ['1','2','3'], subset='validation', color_mode = 'rgb', shuffle = True)
train_generator = train_datagen.flow_from_directory(directory=path, target_size = (IMG_SIZE,IMG_SIZE),
                    batch_size = 30, classes = ['1','2','3'], subset='training', color_mode = 'rgb', shuffle  =True)
```

Ez nagyon szép lenne, ha így lenne, de az a baj, a validation split konkrétan elosztja a mappát keverés nélkül két részre, így a mappa végén lévő képek lesznek a validálók, az elején lévők pedig a train állomány (hiába van a *shuffle=True*, az arra vonatkozik, hogy a már kettévágott datasetből hogy adogat vissza elemeket). Ez azért problémás, mert a mappákon belül a képek olyan sorrendben voltak, hogy az azonos fajta hátterűek egymás után - így bizonyos hátterű nem is került a train halmazba, vagy csak kevés belőle, de cserébe azzal tesztelt végig.

Ebből következően „kézzel” (kódból) kellett randomizálni a képeket:

```
IMG_SIZE =100
path = 'D:/Data/BME/felev_5/temalab_tanulas/datasets/augmented_fulltrain'


os.chdir(path+ '/1/')

i = 1
for c in random.sample(glob('*.png'),17944):
    print(c)
    os.rename(path+'/1/'+c,path+'/1/'+str(i)+'.png')
    shutil.move(str(i)+'.png', 'valid1')
    i+=1
```

Kiválasztottam egy random képet, átneveztem, majd áttettem egy másik mappába, amiben immár a kiválasztás sorrendjében vannak (ezt mindhárom fázis mappáira). A Keras erre nem volt képes, sajnos csak én is vagy 8 óra ötleteléssel azután, hogy láttam, a validation_split nem az elvárt módon működik.

A teljes kód a WokringCNN.ipynb-ben van. Ebben van a mappákba szétosztás is, de azt csak az elején kell megcsinálni, így a tényleges tanításhoz kikommenteztem őket. (Nem túl szép megoldás, de működött :D)

```
Found 4287 images belonging to 3 classes.
Found 38602 images belonging to 3 classes.
```

Ezután már működik a validation_split is. Pár eredmény, amit produkáltak hosszú trainelés után:

```
Epoch 1/15
1287/1287 [==============================] - 753s 585ms/step - loss: 0.5565 - a
ccuracy: 0.7485 - val_loss: 0.2738 - val_accuracy: 0.8941
Epoch 2/15
1287/1287 [==============================] - 735s 571ms/step - loss: 0.1740 - a
ccuracy: 0.9345 - val_loss: 0.0695 - val_accuracy: 0.9757
Epoch 3/15
1287/1287 [==============================] - 730s 567ms/step - loss: 0.0594 - a
ccuracy: 0.9799 - val_loss: 0.0415 - val_accuracy: 0.9879
Epoch 4/15
1287/1287 [==============================] - 727s 565ms/step - loss: 0.0288 - a
ccuracy: 0.9911 - val_loss: 0.0512 - val_accuracy: 0.9839
Epoch 5/15
1287/1287 [==============================] - 651s 506ms/step - loss: 0.0202 - a
ccuracy: 0.9934 - val_loss: 0.0127 - val_accuracy: 0.9956
Epoch 6/15
1287/1287 [==============================] - 678s 527ms/step - loss: 0.0156 - a
ccuracy: 0.9956 - val_loss: 0.0197 - val_accuracy: 0.9949
Epoch 7/15
1287/1287 [==============================] - 701s 545ms/step - loss: 0.0169 - a
ccuracy: 0.9947 - val_loss: 0.0151 - val_accuracy: 0.9946
Epoch 8/15
1287/1287 [==============================] - 759s 590ms/step - loss: 0.0077 - a
ccuracy: 0.9977 - val_loss: 0.0197 - val_accuracy: 0.9946
Epoch 9/15
1287/1287 [==============================] - 611s 474ms/step - loss: 0.0105 - a
ccuracy: 0.9970 - val_loss: 0.0348 - val_accuracy: 0.9911
Epoch 10/15
1287/1287 [==============================] - 655s 509ms/step - loss: 0.0094 - a
ccuracy: 0.9974 - val_loss: 0.0676 - val_accuracy: 0.9774
Epoch 11/15
1287/1287 [==============================] - 666s 517ms/step - loss: 0.0065 - a
ccuracy: 0.9980 - val_loss: 0.0340 - val_accuracy: 0.9909
Epoch 12/15
1287/1287 [==============================] - 611s 475ms/step - loss: 0.0073 - a
ccuracy: 0.9977 - val_loss: 0.0118 - val_accuracy: 0.9963
Epoch 13/15
1287/1287 [==============================] - 623s 484ms/step - loss: 0.0024 - a
ccuracy: 0.9993 - val_loss: 0.0168 - val_accuracy: 0.9946
Epoch 14/15
1287/1287 [==============================] - 630s 490ms/step - loss: 0.0068 - a
ccuracy: 0.9980 - val_loss: 0.0237 - val_accuracy: 0.9932
Epoch 15/15
1287/1287 [==============================] - 628s 488ms/step - loss: 0.0091 - a
ccuracy: 0.9975 - val_loss: 0.0084 - val_accuracy: 0.9979

Out[6]: <tensorflow.python.keras.callbacks.History at 0x2ca82f86fd0>
```

A fenti eredmény 100x100-as képekkel, kékkel jelölt sorokkal.

```
In [7]: model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

        #model.fit(x = batches, epochs = 10, verbose = 2)
        model.fit(train_generator, validation_data = valid_generator, epochs = 10)

        Epoch 1/10
        1287/1287 [==============================] - 489s 380ms/step - loss: 0.5672 - accuracy: 0.7487 - val_loss: 0.2589 - val_accurac
        y: 0.8892
        Epoch 2/10
        1287/1287 [==============================] - 381s 296ms/step - loss: 0.2141 - accuracy: 0.9189 - val_loss: 0.1505 - val_accurac
        y: 0.9517
        Epoch 3/10
        1287/1287 [==============================] - 396s 308ms/step - loss: 0.1205 - accuracy: 0.9574 - val_loss: 0.0957 - val_accurac
        y: 0.9697
        Epoch 4/10
        1287/1287 [==============================] - 459s 357ms/step - loss: 0.0748 - accuracy: 0.9750 - val_loss: 0.0834 - val_accurac
        y: 0.9727
        Epoch 5/10
        1287/1287 [==============================] - 352s 274ms/step - loss: 0.0511 - accuracy: 0.9835 - val_loss: 0.0570 - val_accurac
        y: 0.9816
        Epoch 6/10
        1287/1287 [==============================] - 261s 203ms/step - loss: 0.0412 - accuracy: 0.9865 - val_loss: 0.1393 - val_accurac
        y: 0.9491
        Epoch 7/10
        1287/1287 [==============================] - 287s 223ms/step - loss: 0.0308 - accuracy: 0.9897 - val_loss: 0.0432 - val_accurac
        y: 0.9841
        Epoch 8/10
        1287/1287 [==============================] - 271s 210ms/step - loss: 0.0211 - accuracy: 0.9934 - val_loss: 0.0279 - val_accurac
        y: 0.9900
        Epoch 9/10
        1287/1287 [==============================] - 260s 202ms/step - loss: 0.0275 - accuracy: 0.9915 - val_loss: 0.0511 - val_accurac
        y: 0.9839
        Epoch 10/10
        1287/1287 [==============================] - 251s 195ms/step - loss: 0.0145 - accuracy: 0.9955 - val_loss: 0.0403 - val_accurac
        y: 0.9886

Out[7]: <tensorflow.python.keras.callbacks.History at 0x1a9a2b0fbe0>
```

100x100 kék sorok nélkül.

```
Epoch 1/5
1287/1287 [==============================] - 950s 738ms/step - loss: 0.7422 - a
ccuracy: 0.6713 - val_loss: 0.4113 - val_accuracy: 0.8374
Epoch 2/5
1287/1287 [==============================] - 900s 699ms/step - loss: 0.3177 - a
ccuracy: 0.8821 - val_loss: 0.2248 - val_accuracy: 0.9207
Epoch 3/5
1287/1287 [==============================] - 872s 677ms/step - loss: 0.1769 - a
ccuracy: 0.9394 - val_loss: 0.1699 - val_accuracy: 0.9473
Epoch 4/5
1287/1287 [==============================] - 924s 718ms/step - loss: 0.1156 - a
ccuracy: 0.9638 - val_loss: 0.1386 - val_accuracy: 0.9510
Epoch 5/5
1287/1287 [==============================] - 992s 771ms/step - loss: 0.0867 - a
ccuracy: 0.9738 - val_loss: 0.1205 - val_accuracy: 0.9578

: <tensorflow.python.keras.callbacks.History at 0x1f9ea6bfd30>
```

200x200 kék sorok nélkül.

```
#model.fit(x = batches, epochs = 10, verbose = 2)
model.fit(train_generator, validation_data = valid_generator, epochs = 10)
```

```
Epoch 1/10
1287/1287 [==============================] - 1161s 902ms/step - loss: 0.6576 - accuracy: 0.6975 - val_loss: 0.4307 - val_accura
cy: 0.8234
Epoch 2/10
1287/1287 [==============================] - 1095s 851ms/step - loss: 0.2428 - accuracy: 0.9103 - val_loss: 0.0967 - val_accura
cy: 0.9671
Epoch 3/10
1287/1287 [==============================] - 1079s 839ms/step - loss: 0.0877 - accuracy: 0.9721 - val_loss: 0.0862 - val_accura
cy: 0.9680
Epoch 4/10
1287/1287 [==============================] - 1066s 828ms/step - loss: 0.0513 - accuracy: 0.9823 - val_loss: 0.0524 - val_accura
cy: 0.9806
Epoch 5/10
1287/1287 [==============================] - 1103s 857ms/step - loss: 0.0360 - accuracy: 0.9889 - val_loss: 0.0511 - val_accura
cy: 0.9844
Epoch 6/10
1287/1287 [==============================] - 1089s 846ms/step - loss: 0.0272 - accuracy: 0.9915 - val_loss: 0.0417 - val_accura
cy: 0.9837
Epoch 7/10
1287/1287 [==============================] - 1083s 841ms/step - loss: 0.0201 - accuracy: 0.9937 - val_loss: 0.0204 - val_accura
cy: 0.9942
Epoch 8/10
1287/1287 [==============================] - 1078s 837ms/step - loss: 0.0144 - accuracy: 0.9954 - val_loss: 0.0220 - val_accura
cy: 0.9935
Epoch 9/10
1287/1287 [==============================] - 1066s 829ms/step - loss: 0.0102 - accuracy: 0.9968 - val_loss: 0.0189 - val_accura
cy: 0.9970
Epoch 10/10
1287/1287 [==============================] - 1222s 950ms/step - loss: 0.0096 - accuracy: 0.9972 - val_loss: 0.0145 - val_accura
cy: 0.9963

<tensorflow.python.keras.callbacks.History at 0x20d56fd3a60>
```

200x200 kék sorokkal.

Az eredmény alapján azt tudom állítani, hogy ha makulátlanul jó képekkel tudunk dolgozni, akkor 99,5% nem elérhetetlen. Illetve mégis, hisz ilyen jó képeink nem valószínű, hogy lesznek.

A másik észrevétel, hogy a 100x100-as méret bőven elegendő, még a bonyolultabb háló esetén is (ahol a végére 12x12-es képeket kapunk). Itt fontos lehet, hogy az 50x50-es esetén a végén 6x6-os marad a kép, amiből már elég nehéz kiszülni bármit is (MaxPooling felezi, de a Conv2D is csökkenthet a méreten, ha nem *padding = 'same'* van).

A további teszteket 100x100-as méret és a kékkel jelölt sorokat tartalmazó hálóval csináltam, ez bizonyult a legjobbnak, ha nem csak a pontosság szempont, hanem az is, hogy egy nap alatt esetleg 2 tesztet le tudjak futtatni.

```
model = Sequential([
    Conv2D(filters = 32, kernel_size = (3,3), activation='relu', padding = 'same', input_shape = (IMG_SIZE,IMG_SIZE,3)),
    MaxPool2D(pool_size=(2,2), strides = 2),
    Conv2D(filters = 64, kernel_size = (3,3), activation='relu', padding = 'same'),
    MaxPool2D(pool_size=(2,2), strides = 2),
    Conv2D(filters = 64, kernel_size = (3,3), activation='relu', padding = 'same'),
    MaxPool2D(pool_size=(2,2), strides = 2),
    Flatten(),
    Dense(units=3, activation='softmax')
])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 100, 100, 32)      896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 50, 50, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 50, 50, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 25, 25, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 25, 25, 64)        36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 27651
=================================================================
Total params: 83,971
Trainable params: 83,971
Non-trainable params: 0
_________________________________________________________________
```

Teljes adathalmaz augmentációja

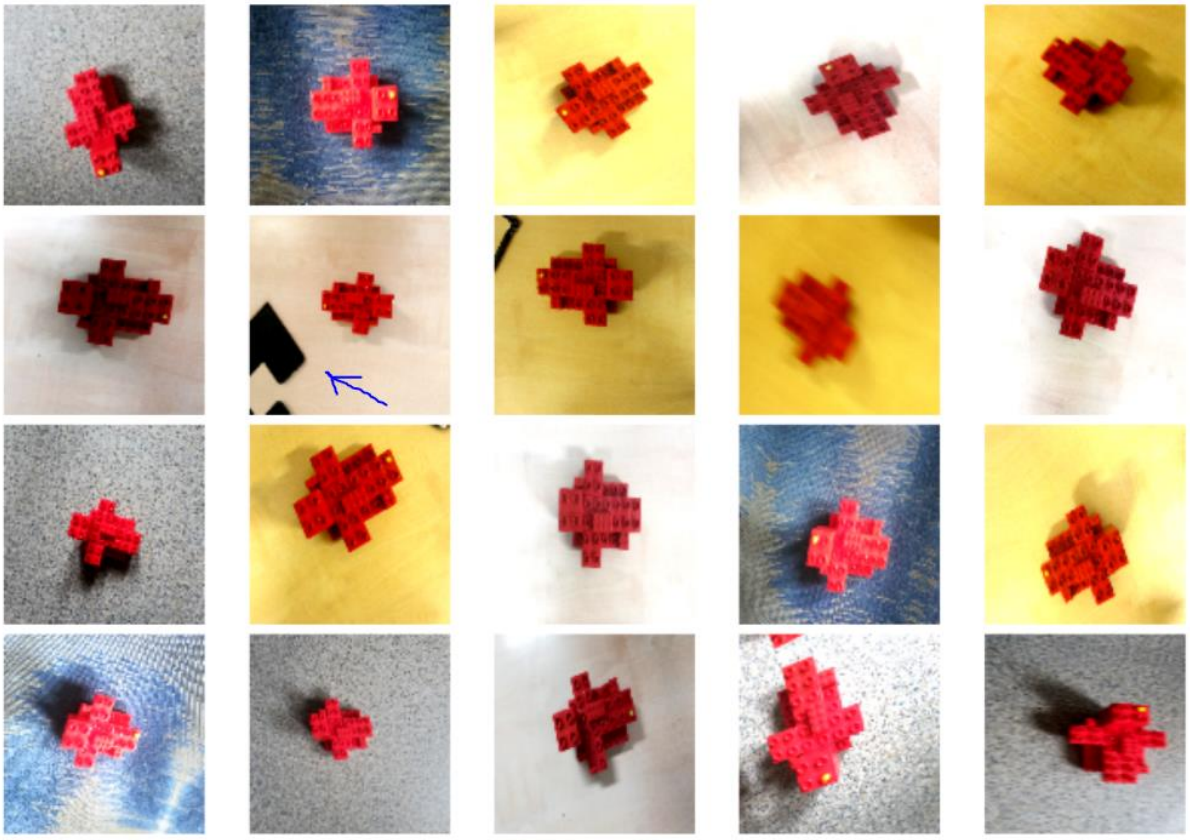
Visszatérve a valósághű világba, nem csak makulátlan képeink lesznek, így legeneráltam 51 000 képet a fent részletezett módon, aminek az alapja mind az 1762 kép, amit készítettünk.

```
train_datagen = ImageDataGenerator(rescale = 1./255, validation_split = 0.1)


valid_generator = train_datagen.flow_from_directory(directory=path, target_size = (IMG_SIZE,IMG_SIZE), batch_size = 30, classes
train_generator = train_datagen.flow_from_directory(directory=path, target_size = (IMG_SIZE,IMG_SIZE), batch_size = 30, classes
```

```
Found 5286 images belonging to 3 classes.
Found 47584 images belonging to 3 classes.
```

Pár kép mintaként:



A kék nyilas képnél épp a fill_mode= 'reflection' csodáját láthatjuk, a telefonomból csinált valami szépet

A nagy kérdés nyilván az, hogy (mennyivel) jobbak azok az eredmények, amik az augmentált adathalmazzal lettek tanítva az eredetihez képest.

```
Epoch 1/10
1587/1587 [==============================] - 993s 626ms/step - loss: 0.8010 - accuracy: 0.5896 - val_loss: 0.5634 - val_accurac
y: 0.7331
Epoch 2/10
1587/1587 [==============================] - 954s 601ms/step - loss: 0.4282 - accuracy: 0.8143 - val_loss: 0.1943 - val_accurac
y: 0.9253
Epoch 3/10
1587/1587 [==============================] - 948s 597ms/step - loss: 0.1272 - accuracy: 0.9559 - val_loss: 0.0910 - val_accurac
y: 0.9692
Epoch 4/10
1587/1587 [==============================] - 941s 593ms/step - loss: 0.0635 - accuracy: 0.9795 - val_loss: 0.0572 - val_accurac
y: 0.9822
Epoch 5/10
1587/1587 [==============================] - 939s 592ms/step - loss: 0.0443 - accuracy: 0.9856 - val_loss: 0.0479 - val_accurac
y: 0.9826
Epoch 6/10
1587/1587 [==============================] - 925s 583ms/step - loss: 0.0314 - accuracy: 0.9900 - val_loss: 0.0334 - val_accurac
y: 0.9896
Epoch 7/10
1587/1587 [==============================] - 912s 574ms/step - loss: 0.0280 - accuracy: 0.9906 - val_loss: 0.0408 - val_accurac
y: 0.9862
Epoch 8/10
1587/1587 [==============================] - 862s 543ms/step - loss: 0.0196 - accuracy: 0.9935 - val_loss: 0.0384 - val_accurac
y: 0.9892
Epoch 9/10
1587/1587 [==============================] - 967s 609ms/step - loss: 0.0203 - accuracy: 0.9935 - val_loss: 0.0317 - val_accurac
y: 0.9894
Epoch 10/10
1587/1587 [==============================] - 959s 604ms/step - loss: 0.0181 - accuracy: 0.9941 - val_loss: 0.0415 - val_accurac
y: 0.9854

<tensorflow.python.keras.callbacks.History at 0x20d01477610>
```

Ez a train eredménye az augmentált adathalmazra. Látható, hogy tulajdonképpen már a 6-7. epoch környékén elérte azt a szintet, amire beállt. Fun fact: amióta megvan a laptopom, nem hallottam annyira felpörögni a hűtést, mint ez alatt.

```
09
Epoch 3/10
53/53 [==============================] - 191s 4s/step - loss: 0.5965 - accuracy: 0.7508 - val_loss: 0.5074 - val_accuracy: 0.78
74
Epoch 4/10
53/53 [==============================] - 179s 3s/step - loss: 0.4353 - accuracy: 0.8202 - val_loss: 0.4082 - val_accuracy: 0.81
61
Epoch 5/10
53/53 [==============================] - 182s 3s/step - loss: 0.3717 - accuracy: 0.8549 - val_loss: 0.3591 - val_accuracy: 0.82
76
Epoch 6/10
53/53 [==============================] - 177s 3s/step - loss: 0.2680 - accuracy: 0.9060 - val_loss: 0.2720 - val_accuracy: 0.88
51
Epoch 7/10
53/53 [==============================] - 177s 3s/step - loss: 0.2316 - accuracy: 0.9066 - val_loss: 0.2951 - val_accuracy: 0.87
93
Epoch 8/10
53/53 [==============================] - 176s 3s/step - loss: 0.2396 - accuracy: 0.8978 - val_loss: 0.2730 - val_accuracy: 0.87
93
Epoch 9/10
53/53 [==============================] - 176s 3s/step - loss: 0.1631 - accuracy: 0.9426 - val_loss: 0.2090 - val_accuracy: 0.90
80
Epoch 10/10
53/53 [==============================] - 176s 3s/step - loss: 0.1518 - accuracy: 0.9432 - val_loss: 0.3127 - val_accuracy: 0.86
78

<tensorflow.python.keras.callbacks.History at 0x136cd44a790>
```

```
model.fit(train_generator, validation_data = valid_generator, epochs = 10)
```

```
Epoch 1/10
53/53 [==============================] - 177s 3s/step - loss: 0.1606 - accuracy: 0.9388 - val_loss: 0.1814 - val_accuracy: 0.89
66
Epoch 2/10
53/53 [==============================] - 177s 3s/step - loss: 0.1564 - accuracy: 0.9338 - val_loss: 0.1180 - val_accuracy: 0.96
55
Epoch 3/10
53/53 [==============================] - 181s 3s/step - loss: 0.1098 - accuracy: 0.9590 - val_loss: 0.1482 - val_accuracy: 0.93
68
Epoch 4/10
53/53 [==============================] - 178s 3s/step - loss: 0.1030 - accuracy: 0.9634 - val_loss: 0.1130 - val_accuracy: 0.94
83
Epoch 5/10
53/53 [==============================] - 177s 3s/step - loss: 0.0844 - accuracy: 0.9716 - val_loss: 0.1109 - val_accuracy: 0.93
10
Epoch 6/10
53/53 [==============================] - 178s 3s/step - loss: 0.0820 - accuracy: 0.9710 - val_loss: 0.1333 - val_accuracy: 0.94
83
Epoch 7/10
53/53 [==============================] - 182s 3s/step - loss: 0.0634 - accuracy: 0.9798 - val_loss: 0.0893 - val_accuracy: 0.94
83
Epoch 8/10
53/53 [==============================] - 178s 3s/step - loss: 0.0694 - accuracy: 0.9773 - val_loss: 0.1493 - val_accuracy: 0.95
40
Epoch 9/10
53/53 [==============================] - 176s 3s/step - loss: 0.0875 - accuracy: 0.9653 - val_loss: 0.1428 - val_accuracy: 0.95
40
Epoch 10/10
53/53 [==============================] - 183s 3s/step - loss: 0.1052 - accuracy: 0.9653 - val_loss: 0.1391 - val_accuracy: 0.96
55
```

Ez az eredménye az eredeti adathalmazzal tanított hálónak. Itt úgy éreztem, 10 epoch nem volt elég a tanításra, ezért lefuttattam még egyszer annyit rajta – az eredmények pedig sokkal jobbak lettek így.

További 10-et is futtattam kíváncsiságból (meg mert viszonylag gyorsan megvan, nem úgy, mint az előző képmennyiségnél) de már nem javult az eredmény, sőt, elkeződött az overfitting.

Konklúzió:

Sok képpel lényegesen jobb eredményt érhetünk el. Hiába nem tűnik soknak a pár százalék javulás, de az augmentált képekkel tanított modell harmadannyit rontott el, mint a másik, így pedig egészen szembetűnő az eredmény. Augmentációval a tanítást is tovább tudjuk folytatni és elérni, hogy generalizáljon a háló, ne legyen overfitting (ezt főleg úgy lehet elérni, hogyha a képek beolvasásakor augmentáljuk meg azt, és egy olyan képpel tanítunk, amit következő epochok alatt sem fog újra látni a háló, vagy csak szimplán nagyon sok augmentált képpel tanítunk). Ha overfitting lépne fel, lehet próbálkozni a dropouttal is, ami tulajdonképpen egyes epochok alatt pár neuront „befagyaszt” véletlenszerűen, ezzel is gátolva, hogy a háló megjegyezze a képeket.

Az az eredmény, amit az augmentált képekkel elért a modell (~98,5% 👌)még tovább javítható lehet (szerintem) ha bonyolultabb modellt használunk és jobb minőségű képeket (mert azért ezek között volt pár olyan, ami – lássuk be – nem volt túl fair).