

# **Témalabor dokumentáció**

## **Machine Learning**

### **Nagyobb témakörök:**

- Ismerkedés a témával, kapott anyagok feldolgozása
- A szivacsok fázisának megállapítása
- A legok fázisának megállapítása:
  1. nagyobb változások
  2. kisebb változások
  3. bináris keresés a minimális változásra

## Ismerkedés a témával, kapott anyagok feldolgozása:

Az elején a félévnek én még nem foglalkoztam a Machine Learning témakörével, tehát először mindenképpen el kellett kezdeni ismerkedni az adott témakörrel, amihez kaptunk nagyon jó segédanyagok. Ezeknek köszönhetően tisztába lettem az alapfogalmakkal és azzal, hogy pontosan melyik részével is akarunk foglalkozni és milyen témával ezen belül.

### Python:

Az első feladat a Pythonnal való megismerkedés volt, mivel még azzal se foglalkoztam. Ehhez kaptunk egy remek másfél órás összefoglaló videót. Gyakorlásképpen a MI háziat is ebbe írtam meg így már elég szilárd alapokkal rendelkeztem ebből.

Jupyter notebookot és Pycharmot használtam a Pythonhoz.

- A notebookkal nagyon könnyű a kimeneteket szépen megjeleníteni, tehát szerintem ezt akkor érdemes használni amikor, gyakran akarsz kimenetet megnézni vagy kiíratni, tehát a gép betanítására tökéletes.
- Pycharmot pedig, amikor egy több függvénnyel rendelkező programot akarsz írni, vagy használni az egyik modelledet. Notebookba én már nem bírtam követni ezeket.

### ML:

Itt kaptunk egy 8 részes ismertetőt / kedvcsinálót, különböző problémák megoldásáról, bemutatásáról: <https://medium.com/@ageitgey/machine-learning-is-fun-80ea3ec3c471>

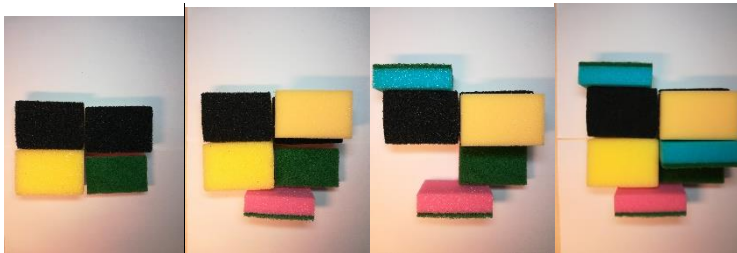
Ez nagyon érdekes volt szerintem, ezért meg is hozta a kedvem még jobban a témakörrel kapcsolatosan.

A következő egy videósorozat volt, ahol minden videóban más problémákat oldottak meg, különböző módszerekkel és azok bemutatásával: [https://www.youtube.com/watch?v=gmvvaobm7eQ&list=PLeo1K3hjS3uvCeTYTeyfe0-rN5r8zn9rw&ab\\_channel=codebasics](https://www.youtube.com/watch?v=gmvvaobm7eQ&list=PLeo1K3hjS3uvCeTYTeyfe0-rN5r8zn9rw&ab_channel=codebasics)

Ez a videó sorozat egy nagyon jó alaptudással látott el engem és így már neki is tudtam állni az első nagyobb feladatnak a szivacsok felismerésének.

## A szivacsok fázisának megállapítása:

Az első ötlet az volt, hogy legokkal kezdjünk el fázisokat készíteni, de mivel legot nem tudtunk elég gyorsan szerezni, ezért hárman csináltunk képeket különböző szivacsokkal más-más helyzetben. A következő állapotok születtek:



Erre az interneten talált videó segítségével építettem egy neurális hálót és deep learning segítségével a modelletem betanítottam a kód a következő:

```
In [1]: from tensorflow.keras.preprocessing.image import ImageDataGenerator
        from tensorflow.keras.preprocessing import image
        from tensorflow.keras.optimizers import RMSprop
        import matplotlib.pyplot as plt
        import tensorflow as tf
        import numpy as np
        import cv2
        import os
```

Ezek a szükséges importok (később lehet az adatbeolvasásra még kipróbálok egy másik megoldást és változni fog.) Tensorflowval és kerassal dolgozok alapvetően, tehát ezek az alap importok mint látható. Ezeken kívül még pár a ML-ben népszerűen használt library található.

```
In [2]: train = ImageDataGenerator(rescale = 1/255)
        validation = ImageDataGenerator(rescale = 1/255)

In [3]: train_dataset = train.flow_from_directory('F:\datasets\szivacsok_bont/train', target_size= (200,200),
        batch_size = 3, class_mode = 'categorical')
        validation_dataset = train.flow_from_directory('F:\datasets\szivacsok_bont/validate', target_size= (200,200),
        batch_size = 3, class_mode = 'categorical')

Found 253 images belonging to 4 classes.
Found 236 images belonging to 4 classes.
```

Az első blokkban a 0-255es intervallum 0-1 közé szorítása található (ML szemponjából 0 és 1 közötti értékekkel sokkal jobb eredményt lehet elérni.) Ezután pedig a kép átméretezése található, mert ezek az állapotok

megkülönböztetéséhez nem kell full hd kép. 200x200-as van beállítva de szerintem még ez is sok egy 50x50-es is elég. Mivel több mint 2 állapotunk van, ezért categorical a mode. Alatta ki is írja mennyi képet talált és hány osztályt ez a jóságot jelzi (mert tényleg ennyi osztály és kép van.) Kettő adatállományra pedig azért van szükség, mert neurális háló készítésénél két adatcsoport kell. Egy amivel traineljük, egy amivel pedig az overfitting jelenséget ellenőrizzük.

```
In [4]: model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(16,(3,3),activation = 'relu', input_shape =(200,200,3)),
                                             tf.keras.layers.MaxPool2D(2,2),
                                             #
                                             tf.keras.layers.Conv2D(32,(3,3),activation = 'relu'),
                                             tf.keras.layers.MaxPool2D(2,2),
                                             #
                                             tf.keras.layers.Conv2D(64,(3,3),activation = 'relu'),
                                             tf.keras.layers.MaxPool2D(2,2),
                                             #
                                             tf.keras.layers.Flatten(),
                                             #
                                             tf.keras.layers.Dense(512, activation = 'relu'),
                                             #
                                             tf.keras.layers.Dense(4,activation = 'softmax')
                                             ])

In [12]: model.compile(loss = 'categorical_crossentropy', optimizer = RMSprop(lr=0.001), metrics = ['accuracy'])
```

Az első blokkban a neurális háló felépítése található. Ez ad hoc módon jött, ezt a részét próbálgatni kell melyik illik éppen a modellre. Egyedül az utolsó layernek a neuronjainak a száma fix, ami 4 jelen esetben, mert 4 kategóriánk van.

A következő blokkban a modellnek a loss számolási módszerét állítjuk be, ami categorical, mert kategóriákat nézünk, és optimizert pedig szintén érzésre választunk, ezzel jól működött.

```
model_fit = model.fit(train_dataset,
                      epochs=10,
                      validation_data= validation_dataset,)
```

```
Epoch 1/10
105/105 [=====] - 157s 1s/step - loss: 0.5661 - accuracy: 0.7651 - val_loss: 0.0198 - val_accuracy: 1.0000
Epoch 2/10
105/105 [=====] - 140s 1s/step - loss: 0.0397 - accuracy: 0.9841 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 3/10
105/105 [=====] - 142s 1s/step - loss: 0.0117 - accuracy: 0.9968 - val_loss: 0.1384 - val_accuracy: 0.9425
Epoch 4/10
105/105 [=====] - 149s 1s/step - loss: 0.0275 - accuracy: 0.9937 - val_loss: 1.4959e-05 - val_accuracy: 1.0000
Epoch 5/10
105/105 [=====] - 166s 2s/step - loss: 2.9840e-04 - accuracy: 1.0000 - val_loss: 5.9661e-06 - val_accuracy: 1.0000
Epoch 6/10
105/105 [=====] - 152s 1s/step - loss: 1.5307e-05 - accuracy: 1.0000 - val_loss: 1.6991e-07 - val_accuracy: 1.0000
Epoch 7/10
105/105 [=====] - 144s 1s/step - loss: 1.9679e-08 - accuracy: 1.0000 - val_loss: 5.4809e-09 - val_accuracy: 1.0000
Epoch 8/10
105/105 [=====] - 141s 1s/step - loss: 3.7844e-10 - accuracy: 1.0000 - val_loss: 1.3702e-09 - val_accuracy: 1.0000
Epoch 9/10
105/105 [=====] - 99s 946ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 1.3702e-09 - val_accuracy: 1.0000
Epoch 10/10
105/105 [=====] - 68s 645ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 6.8511e-10 - val_accuracy: 1.0000
```

Aktiválja a Wi

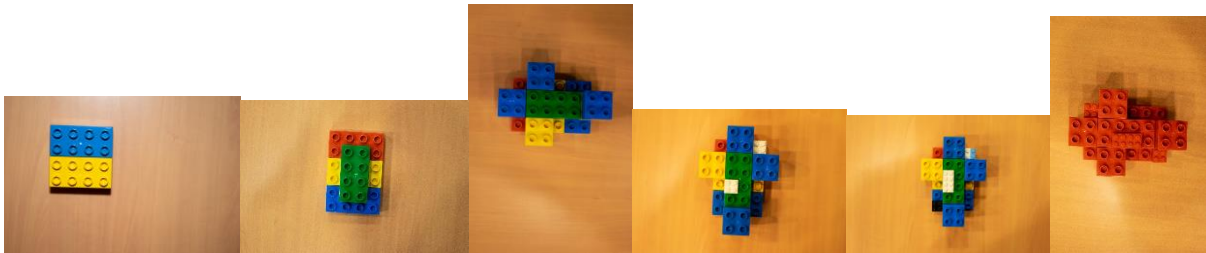
A modellünket traineljük az adatainkkal a képen. a végeredmény a legalsó sorban található, ami 100%, tehát ez azt jelenti a végére a modellünk, már mindig megtudta mondani melyik állapotról van a kép ami elég jó eredmény, valójában tökéletes.

Azt beszéltük, ez azért lehet mivel a képeket megkülönböztetni, elég könnyű lehetett, mivel nagy változások mentek végbe 1-1 állapot közt.

## A legok fázisának megállapítása:

### 1. nagyobb változások:

Most már tudtunk legot szerezni és így, már tudtunk kisebb változásokat is modellezni. Ez alatt található a 6 fázis:



Az első két fázisváltásnál még itt is nagyobb változások voltak. Az utána levőknél viszont, már minimálisak, az utolsó állapot változás konkrétan egy csak más színű, mint az előző, ami már kötelezővé tette a színek használatát a modellben.

```
In [4]: model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(16,(3,3),activation = 'relu', input_shape = (200,200,3)),
      tf.keras.layers.MaxPool2D(2,2),
      #
      tf.keras.layers.Conv2D(32,(3,3),activation = 'relu'),
      tf.keras.layers.MaxPool2D(2,2),
      #
      tf.keras.layers.Conv2D(64,(3,3),activation = 'relu'),
      tf.keras.layers.MaxPool2D(2,2),
      #
      tf.keras.layers.Flatten(),
      #
      tf.keras.layers.Dense(512, activation = 'relu'),
      #
      tf.keras.layers.Dense(6,activation = 'softmax')
    ])

In [5]: model.compile(loss = 'categorical_crossentropy', optimizer = RMSprop(lr=0.001), metrics = ['accuracy'])
```

Az előző feladathoz képest a kód nem változott még sokat. a 4-esből 6 lett, mert itt már 6 állapot található. Ezenkívül a legalul aláhúzott optimizerrel végeztem kísérleteket.

## RMSprop optimizer:

```
model.compile(loss = 'categorical_crossentropy', optimizer = RMSprop(lr=0.001), metrics = ['accuracy'])

model_fit = model.fit(train_dataset,
                      steps_per_epoch = 50,
                      epochs=15,
                      validation_data= validation_dataset)
```

```
Epoch 1/15
50/50 [=====] - 115s 2s/step - loss: 2.1381 - accuracy: 0.5120 - val_loss: 0.8725 - val_accuracy: 0.58
64
Epoch 2/15
50/50 [=====] - 103s 2s/step - loss: 0.7677 - accuracy: 0.6880 - val_loss: 0.6016 - val_accuracy: 0.77
78
Epoch 3/15
50/50 [=====] - 108s 2s/step - loss: 0.5495 - accuracy: 0.8240 - val_loss: 0.4743 - val_accuracy: 0.82
10
Epoch 4/15
50/50 [=====] - 101s 2s/step - loss: 0.3399 - accuracy: 0.8988 - val_loss: 0.3683 - val_accuracy: 0.84
57
Epoch 5/15
50/50 [=====] - 105s 2s/step - loss: 0.2613 - accuracy: 0.9200 - val_loss: 0.3408 - val_accuracy: 0.90
74
Epoch 6/15
50/50 [=====] - 108s 2s/step - loss: 0.3726 - accuracy: 0.9240 - val_loss: 0.2842 - val_accuracy: 0.91
36
Epoch 7/15
50/50 [=====] - 99s 2s/step - loss: 0.1749 - accuracy: 0.9400 - val_loss: 0.2703 - val_accuracy: 0.913
6
Epoch 8/15
50/50 [=====] - 105s 2s/step - loss: 0.2118 - accuracy: 0.9480 - val_loss: 0.2501 - val_accuracy: 0.93
21
Epoch 9/15
50/50 [=====] - 103s 2s/step - loss: 0.1171 - accuracy: 0.9595 - val_loss: 0.3365 - val_accuracy: 0.91
36
Epoch 10/15
50/50 [=====] - 102s 2s/step - loss: 0.1139 - accuracy: 0.9800 - val_loss: 0.2895 - val_accuracy: 0.94
44
Epoch 11/15
50/50 [=====] - 115s 2s/step - loss: 0.1074 - accuracy: 0.9800 - val_loss: 0.6061 - val_accuracy: 0.84
57
Epoch 12/15
50/50 [=====] - 112s 2s/step - loss: 0.1226 - accuracy: 0.9680 - val_loss: 0.2125 - val_accuracy: 0.95
06
Epoch 13/15
50/50 [=====] - 112s 2s/step - loss: 0.0180 - accuracy: 0.9919 - val_loss: 0.3363 - val_accuracy: 0.91
36
Epoch 14/15
50/50 [=====] - 116s 2s/step - loss: 0.2039 - accuracy: 0.9800 - val_loss: 0.2449 - val_accuracy: 0.95
68
Epoch 15/15
50/50 [=====] - 103s 2s/step - loss: 0.0987 - accuracy: 0.9919 - val_loss: 0.2266 - val_accuracy: 0.93
21
```

15 epochig futtattam. Itt a legnagyobb validation accuracy az 0.95 volt, de utána mindig volt egy visszaesés. Én azt mondanám ez alapján, hogy lehet kicsit már overtrained a modellt, mivel a val\_loss ugrál összevissza, ezért szerintem a valós accuracy kevesebre tehető, körülbelül 90-93%-ra. Szerintem ez elég jó eredmény, mivel a képek nem is voltak feltétlen beforgatva (mint a fenti képeken látszik), a fényviszonyok is különböznek illetve, az képek mennyisége se volt túl sok csak olyan 100 körüli / fázis. Az itt felsorolt feltételek mind javíthatók szerintem mivel, a gyártáson is egy szögből egy fényviszonyban fogunk fényképezni nem fog a kamera mozogni stb.. Tehát szerintem egy magabiztos 95%+ elérhető lenne ezekre a fázisokra.

## Adam optimizer:

```

In [5]: model.compile(loss = 'categorical_crossentropy', optimizer = "adam", metrics = ['accuracy'])

In [6]: model_fit = model.fit(train_dataset,
                             steps_per_epoch = 50,
                             epochs=15,
                             validation_data= validation_dataset)

```

```

Epoch 1/15
50/50 [=====] - 112s 2s/step - loss: 1.6123 - accuracy: 0.3800 - val_loss: 0.9875 - val_accuracy: 0.61
11
Epoch 2/15
50/50 [=====] - 102s 2s/step - loss: 0.8568 - accuracy: 0.6842 - val_loss: 0.7522 - val_accuracy: 0.69
75
Epoch 3/15
50/50 [=====] - 108s 2s/step - loss: 0.7166 - accuracy: 0.7773 - val_loss: 0.6264 - val_accuracy: 0.70
99
Epoch 4/15
50/50 [=====] - 111s 2s/step - loss: 0.5270 - accuracy: 0.8200 - val_loss: 0.3679 - val_accuracy: 0.88
27
Epoch 5/15
50/50 [=====] - 107s 2s/step - loss: 0.2417 - accuracy: 0.9433 - val_loss: 0.4148 - val_accuracy: 0.85
19
Epoch 6/15
50/50 [=====] - 109s 2s/step - loss: 0.2555 - accuracy: 0.9312 - val_loss: 0.3671 - val_accuracy: 0.90
12
Epoch 7/15
50/50 [=====] - 100s 2s/step - loss: 0.1146 - accuracy: 0.9560 - val_loss: 0.5198 - val_accuracy: 0.82
72
Epoch 8/15
50/50 [=====] - 98s 2s/step - loss: 0.1845 - accuracy: 0.9480 - val_loss: 0.2480 - val_accuracy: 0.913
6
Epoch 9/15
50/50 [=====] - 107s 2s/step - loss: 0.1336 - accuracy: 0.9800 - val_loss: 0.2672 - val_accuracy: 0.92
59
Epoch 10/15
50/50 [=====] - 95s 2s/step - loss: 0.0452 - accuracy: 0.9879 - val_loss: 0.5328 - val_accuracy: 0.895
1
Epoch 11/15
50/50 [=====] - 107s 2s/step - loss: 0.1150 - accuracy: 0.9717 - val_loss: 0.2706 - val_accuracy: 0.91
98
Epoch 12/15
50/50 [=====] - 99s 2s/step - loss: 0.0554 - accuracy: 0.9757 - val_loss: 0.1510 - val_accuracy: 0.944
4
Epoch 13/15
50/50 [=====] - 98s 2s/step - loss: 0.0521 - accuracy: 0.9920 - val_loss: 0.1505 - val_accuracy: 0.944
4
Epoch 14/15
50/50 [=====] - 102s 2s/step - loss: 0.0474 - accuracy: 0.9838 - val_loss: 0.1720 - val_accuracy: 0.95
68
Epoch 15/15
50/50 [=====] - 98s 2s/step - loss: 0.0118 - accuracy: 1.0000 - val_loss: 0.1410 - val_accuracy: 0.969
1

```

15 epochig futtattam ezt is. Itt a val\_accuracy végig nőtt és a val\_loss végig csökkent, tehát ez a modell biztos nincs overtrainerve. Az elején kicsit lassabban növelte az accuracyt, de a végére sokkal magabiztosabban haladt előre nem voltak ugrálások, még valószínűleg tovább is lehetne trainelni. Ezzel a optimizerrel, tehát egy magabiztos 96%+-t el lehet érni, azok az optimalizálások nélkül, amiről az előző optimizernél beszéltem. Azokat megcsinálva lehet akár egy erősen 100% közeli állapot is elérhető.

Szerintem a fent leírtak alapján egyértelműen az Adam oldotta meg jobban ezt az adott problémát.



Kisebb fázisok:

Legokból csináltunk olyan fázisokat is ahol még kisebb volt a változás, itt láthatóak a fázisok:



Itt mindegyik lépésben egy nagyon kicsi lego került rá az eddigi nagy legokhoz képest, tehát az összes hasonlít kicsit az összesre. Mivel öt kimenet van, ezért ugyanúgy az előbb említett részt korrigáltam csak megfelelően. Ugyan úgy kipróbáltam mind a két optimizerrel.

RMSprop optimizer:

```
model_fit = model.fit(train_dataset,
                      steps_per_epoch = 6,
                      epochs=50,
                      validation_data= validation_dataset)
```

Epoch 1/50  
6/6 [=====] - 9s 1s/step - loss: 13.6089 - accuracy: 0.2222 - val\_loss: 1.5785 - val\_accuracy: 0.3947  
Epoch 2/50  
6/6 [=====] - 8s 1s/step - loss: 1.7203 - accuracy: 0.3889 - val\_loss: 2.2369 - val\_accuracy: 0.2632  
Epoch 3/50  
6/6 [=====] - 8s 1s/step - loss: 1.6092 - accuracy: 0.4375 - val\_loss: 1.5714 - val\_accuracy: 0.2368  
Epoch 4/50  
6/6 [=====] - 8s 1s/step - loss: 1.3301 - accuracy: 0.3889 - val\_loss: 4.1504 - val\_accuracy: 0.1053  
Epoch 5/50  
6/6 [=====] - 8s 1s/step - loss: 1.4822 - accuracy: 0.3333 - val\_loss: 1.6138 - val\_accuracy: 0.5526  
Epoch 6/50  
6/6 [=====] - 8s 1s/step - loss: 1.2821 - accuracy: 0.6111 - val\_loss: 1.1197 - val\_accuracy: 0.6053  
Epoch 7/50  
6/6 [=====] - 8s 1s/step - loss: 1.8817 - accuracy: 0.4444 - val\_loss: 1.1930 - val\_accuracy: 0.5000  
Epoch 8/50  
6/6 [=====] - 8s 1s/step - loss: 0.9862 - accuracy: 0.6111 - val\_loss: 0.9696 - val\_accuracy: 0.7105  
Epoch 9/50  
6/6 [=====] - 8s 1s/step - loss: 0.7897 - accuracy: 0.8333 - val\_loss: 0.8279 - val\_accuracy: 0.7632  
Epoch 10/50  
6/6 [=====] - 9s 1s/step - loss: 0.8384 - accuracy: 0.6667 - val\_loss: 0.5778 - val\_accuracy: 0.7632  
Epoch 11/50  
6/6 [=====] - 9s 1s/step - loss: 0.6282 - accuracy: 0.7778 - val\_loss: 0.7275 - val\_accuracy: 0.6053  
Epoch 12/50  
6/6 [=====] - 8s 1s/step - loss: 1.5710 - accuracy: 0.6667 - val\_loss: 0.5448 - val\_accuracy: 0.8947  
Epoch 13/50  
6/6 [=====] - 9s 1s/step - loss: 0.5159 - accuracy: 0.7778 - val\_loss: 0.7316 - val\_accuracy: 0.7895  
Epoch 14/50  
6/6 [=====] - 8s 1s/step - loss: 0.2828 - accuracy: 0.9444 - val\_loss: 0.6360 - val\_accuracy: 0.7895  
Epoch 15/50  
6/6 [=====] - 8s 1s/step - loss: 1.5106 - accuracy: 0.5556 - val\_loss: 0.6713 - val\_accuracy: 0.8421  
Epoch 16/50  
6/6 [=====] - 8s 1s/step - loss: 0.2318 - accuracy: 0.9444 - val\_loss: 1.5283 - val\_accuracy: 0.7105  
Epoch 17/50  
6/6 [=====] - 8s 1s/step - loss: 0.5550 - accuracy: 0.7222 - val\_loss: 0.7031 - val\_accuracy: 0.7895  
Epoch 18/50  
6/6 [=====] - 8s 1s/step - loss: 0.6202 - accuracy: 0.7778 - val\_loss: 0.6501 - val\_accuracy: 0.7368  
Epoch 19/50  
6/6 [=====] - 8s 1s/step - loss: 0.1777 - accuracy: 1.0000 - val\_loss: 0.5487 - val\_accuracy: 0.8421  
Epoch 20/50  
6/6 [=====] - 8s 1s/step - loss: 0.1685 - accuracy: 0.9444 - val\_loss: 0.4530 - val\_accuracy: 0.8421  
Epoch 21/50  
6/6 [=====] - 8s 1s/step - loss: 0.1347 - accuracy: 0.9444 - val\_loss: 0.4067 - val\_accuracy: 0.8684  
Epoch 22/50  
6/6 [=====] - 8s 1s/step - loss: 0.4968 - accuracy: 0.8889 - val\_loss: 0.4822 - val\_accuracy: 0.8158  
Epoch 23/50  
6/6 [=====] - 8s 1s/step - loss: 0.0403 - accuracy: 1.0000 - val\_loss: 0.4214 - val\_accuracy: 0.8947  
Epoch 24/50  
6/6 [=====] - 8s 1s/step - loss: 0.0547 - accuracy: 1.0000 - val\_loss: 0.5191 - val\_accuracy: 0.8947  
Epoch 25/50  
6/6 [=====] - 8s 1s/step - loss: 0.0169 - accuracy: 1.0000 - val\_loss: 0.6923 - val\_accuracy: 0.8421

```

Epoch 26/50
6/6 [=====] - 9s 2s/step - loss: 1.5481 - accuracy: 0.7778 - val_loss: 1.3346 - val_accuracy: 0.7105
Epoch 27/50
6/6 [=====] - 10s 2s/step - loss: 0.2301 - accuracy: 0.8889 - val_loss: 0.3765 - val_accuracy: 0.9211
Epoch 28/50
6/6 [=====] - 9s 1s/step - loss: 0.1559 - accuracy: 0.8750 - val_loss: 0.5035 - val_accuracy: 0.8947
Epoch 29/50
6/6 [=====] - 8s 1s/step - loss: 0.1387 - accuracy: 0.9444 - val_loss: 0.4570 - val_accuracy: 0.8947
Epoch 30/50
6/6 [=====] - 9s 2s/step - loss: 0.2261 - accuracy: 0.9444 - val_loss: 0.7611 - val_accuracy: 0.7632
Epoch 31/50
6/6 [=====] - 8s 1s/step - loss: 0.1923 - accuracy: 0.8889 - val_loss: 1.2622 - val_accuracy: 0.6842
Epoch 32/50
6/6 [=====] - 8s 1s/step - loss: 0.2142 - accuracy: 0.9375 - val_loss: 0.3715 - val_accuracy: 0.9211
Epoch 33/50
6/6 [=====] - 8s 1s/step - loss: 0.3724 - accuracy: 0.9444 - val_loss: 0.2830 - val_accuracy: 0.8947
Epoch 34/50
6/6 [=====] - 9s 1s/step - loss: 0.0279 - accuracy: 1.0000 - val_loss: 0.2371 - val_accuracy: 0.8947
Epoch 35/50
6/6 [=====] - 9s 1s/step - loss: 0.0413 - accuracy: 1.0000 - val_loss: 0.3369 - val_accuracy: 0.8947
Epoch 36/50
6/6 [=====] - 8s 1s/step - loss: 0.0974 - accuracy: 0.9444 - val_loss: 0.2816 - val_accuracy: 0.9211
Epoch 37/50
6/6 [=====] - 8s 1s/step - loss: 0.2523 - accuracy: 0.8750 - val_loss: 0.3710 - val_accuracy: 0.8947
Epoch 38/50
6/6 [=====] - 8s 1s/step - loss: 0.0197 - accuracy: 1.0000 - val_loss: 0.3557 - val_accuracy: 0.8947
Epoch 39/50
6/6 [=====] - 8s 1s/step - loss: 9.2263e-04 - accuracy: 1.0000 - val_loss: 0.3623 - val_accuracy: 0.89
47
Epoch 40/50
6/6 [=====] - 8s 1s/step - loss: 0.0257 - accuracy: 1.0000 - val_loss: 0.4150 - val_accuracy: 0.8947
Epoch 41/50
6/6 [=====] - 8s 1s/step - loss: 0.0465 - accuracy: 0.9444 - val_loss: 7.4097 - val_accuracy: 0.2105
Epoch 42/50
6/6 [=====] - 8s 1s/step - loss: 2.2226 - accuracy: 0.6875 - val_loss: 0.9723 - val_accuracy: 0.8421
Epoch 43/50
6/6 [=====] - 8s 1s/step - loss: 0.1790 - accuracy: 0.9444 - val_loss: 0.6201 - val_accuracy: 0.8684
Epoch 44/50
6/6 [=====] - 8s 1s/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.5942 - val_accuracy: 0.8684
Epoch 45/50
6/6 [=====] - 9s 1s/step - loss: 8.2100e-04 - accuracy: 1.0000 - val_loss: 0.5963 - val_accuracy: 0.86
84
Epoch 46/50
6/6 [=====] - 8s 1s/step - loss: 0.0023 - accuracy: 1.0000 - val_loss: 0.5401 - val_accuracy: 0.8684
Epoch 47/50
6/6 [=====] - 8s 1s/step - loss: 0.1417 - accuracy: 0.9444 - val_loss: 0.4763 - val_accuracy: 0.8421
Epoch 48/50
6/6 [=====] - 8s 1s/step - loss: 0.0048 - accuracy: 1.0000 - val_loss: 0.4271 - val_accuracy: 0.8947
Epoch 49/50
6/6 [=====] - 8s 1s/step - loss: 0.0113 - accuracy: 1.0000 - val_loss: 0.3690 - val_accuracy: 0.8947
Epoch 50/50
6/6 [=====] - 8s 1s/step - loss: 6.3232e-04 - accuracy: 1.0000 - val_loss: 0.3652 - val_accuracy: 0.89
47

```

Röviden összefoglalva:

- 10. epochig növekedtünk 76%ig
- 20. epochig le/fölfele ugrálva elértük a 84%-ot
- 23. epochig elértük a 89%-ot
- utána ez az érték körül ugráltunk fel le egészen a végéig

Itt a fényviszonyok viszonylag egységesek voltak és mindig ugyanabból a szögből is volt fényképezve. A kamera mozgott, de az teljesen szabályos mivel a gyártó soron se biztos, hogy ugyanoda teszik vissza. Ennek a két paraméternek köszönhető szerintem ez a nagyon jó 89% körüli eredmény, ami az előzőekhez képest kevésnek hangozhat, de ha figyelembe vesszük, hogy fázisonként átlagosan 30 kép volt és két fázisban csak 20, akkor teljesen ártékelődik és szerintem egy teljesen meglepően jó eredménynek számít már így. Ha lett volna legalább 100 kép mindről 90% fölött lennénk az biztos, ha nem 95% fölött.

## Adam optimizer:

```

Epoch 1/50
6/6 [=====] - 9s 1s/step - loss: 3.7860 - accuracy: 0.2778 - val_loss: 1.5717 - val_accuracy: 0.2105
Epoch 2/50
6/6 [=====] - 8s 1s/step - loss: 1.6287 - accuracy: 0.2778 - val_loss: 1.5789 - val_accuracy: 0.3947
Epoch 3/50
6/6 [=====] - 8s 1s/step - loss: 1.5440 - accuracy: 0.5000 - val_loss: 1.5888 - val_accuracy: 0.1053
Epoch 4/50
6/6 [=====] - 8s 1s/step - loss: 1.4605 - accuracy: 0.5000 - val_loss: 1.5385 - val_accuracy: 0.3421
Epoch 5/50
6/6 [=====] - 8s 1s/step - loss: 1.5832 - accuracy: 0.3125 - val_loss: 1.4365 - val_accuracy: 0.2368
Epoch 6/50
6/6 [=====] - 8s 1s/step - loss: 1.4034 - accuracy: 0.3333 - val_loss: 1.3935 - val_accuracy: 0.3947
Epoch 7/50
6/6 [=====] - 8s 1s/step - loss: 1.2787 - accuracy: 0.3889 - val_loss: 1.1453 - val_accuracy: 0.5263
Epoch 8/50
6/6 [=====] - 8s 1s/step - loss: 0.8806 - accuracy: 0.6875 - val_loss: 0.8167 - val_accuracy: 0.6842
Epoch 9/50
6/6 [=====] - 8s 1s/step - loss: 0.3168 - accuracy: 1.0000 - val_loss: 0.5763 - val_accuracy: 0.8421
Epoch 10/50
6/6 [=====] - 8s 1s/step - loss: 0.3803 - accuracy: 0.7778 - val_loss: 1.1318 - val_accuracy: 0.6316
Epoch 11/50
6/6 [=====] - 8s 1s/step - loss: 0.6241 - accuracy: 0.7778 - val_loss: 0.8196 - val_accuracy: 0.7368
Epoch 12/50
6/6 [=====] - 8s 1s/step - loss: 0.6374 - accuracy: 0.8333 - val_loss: 0.4681 - val_accuracy: 0.8947
Epoch 13/50
6/6 [=====] - 8s 1s/step - loss: 0.3205 - accuracy: 0.8333 - val_loss: 0.4504 - val_accuracy: 0.8684
Epoch 14/50
6/6 [=====] - 8s 1s/step - loss: 0.4830 - accuracy: 0.8333 - val_loss: 0.9601 - val_accuracy: 0.6579
Epoch 15/50
6/6 [=====] - 8s 1s/step - loss: 0.6022 - accuracy: 0.7222 - val_loss: 0.7022 - val_accuracy: 0.7895
Epoch 16/50
6/6 [=====] - 8s 1s/step - loss: 0.1924 - accuracy: 0.9444 - val_loss: 0.7557 - val_accuracy: 0.6842
Epoch 17/50
6/6 [=====] - 8s 1s/step - loss: 0.5778 - accuracy: 0.8333 - val_loss: 0.4951 - val_accuracy: 0.8947
Epoch 18/50
6/6 [=====] - 8s 1s/step - loss: 0.2867 - accuracy: 0.8889 - val_loss: 0.8832 - val_accuracy: 0.5789
Epoch 19/50
6/6 [=====] - 8s 1s/step - loss: 0.1145 - accuracy: 0.9444 - val_loss: 0.5686 - val_accuracy: 0.7368
Epoch 20/50
6/6 [=====] - 8s 1s/step - loss: 0.3184 - accuracy: 0.8333 - val_loss: 0.4665 - val_accuracy: 0.7632
Epoch 21/50
6/6 [=====] - 8s 1s/step - loss: 0.0586 - accuracy: 1.0000 - val_loss: 0.4431 - val_accuracy: 0.8947
Epoch 22/50
6/6 [=====] - 8s 1s/step - loss: 0.1497 - accuracy: 0.9444 - val_loss: 0.4329 - val_accuracy: 0.8684
Epoch 23/50
6/6 [=====] - 8s 1s/step - loss: 0.0700 - accuracy: 1.0000 - val_loss: 0.3721 - val_accuracy: 0.8158
Epoch 24/50
6/6 [=====] - 8s 1s/step - loss: 0.0855 - accuracy: 1.0000 - val_loss: 0.3166 - val_accuracy: 0.8684
Epoch 25/50
6/6 [=====] - 8s 1s/step - loss: 0.0538 - accuracy: 1.0000 - val_loss: 0.3799 - val_accuracy: 0.9211
Epoch 26/50
6/6 [=====] - 8s 1s/step - loss: 0.0133 - accuracy: 1.0000 - val_loss: 0.5086 - val_accuracy: 0.9211
Epoch 27/50
6/6 [=====] - 8s 1s/step - loss: 0.0232 - accuracy: 1.0000 - val_loss: 0.4586 - val_accuracy: 0.9211

Epoch 28/50
6/6 [=====] - 8s 1s/step - loss: 0.0290 - accuracy: 1.0000 - val_loss: 0.3098 - val_accuracy: 0.9211
Epoch 29/50
6/6 [=====] - 8s 1s/step - loss: 0.0422 - accuracy: 1.0000 - val_loss: 0.2612 - val_accuracy: 0.8947
Epoch 30/50
6/6 [=====] - 8s 1s/step - loss: 0.1449 - accuracy: 0.9444 - val_loss: 2.6834 - val_accuracy: 0.3684
Epoch 31/50
6/6 [=====] - 8s 1s/step - loss: 0.1393 - accuracy: 0.8889 - val_loss: 0.2287 - val_accuracy: 0.9211
Epoch 32/50
6/6 [=====] - 8s 1s/step - loss: 0.0125 - accuracy: 1.0000 - val_loss: 0.5340 - val_accuracy: 0.8947
Epoch 33/50
6/6 [=====] - 8s 1s/step - loss: 0.1758 - accuracy: 0.9375 - val_loss: 0.3808 - val_accuracy: 0.8947
Epoch 34/50
6/6 [=====] - 8s 1s/step - loss: 0.0280 - accuracy: 1.0000 - val_loss: 0.3969 - val_accuracy: 0.8684
Epoch 35/50
6/6 [=====] - 8s 1s/step - loss: 0.1198 - accuracy: 0.9375 - val_loss: 0.5008 - val_accuracy: 0.8947
Epoch 36/50
6/6 [=====] - 8s 1s/step - loss: 0.1993 - accuracy: 0.8889 - val_loss: 0.2036 - val_accuracy: 0.9211
Epoch 37/50
6/6 [=====] - 8s 1s/step - loss: 0.0201 - accuracy: 1.0000 - val_loss: 0.1725 - val_accuracy: 0.9474
Epoch 38/50
6/6 [=====] - 8s 1s/step - loss: 0.0612 - accuracy: 1.0000 - val_loss: 0.4536 - val_accuracy: 0.8684
Epoch 39/50
6/6 [=====] - 8s 1s/step - loss: 0.0330 - accuracy: 1.0000 - val_loss: 0.2615 - val_accuracy: 0.8947
Epoch 40/50
6/6 [=====] - 8s 1s/step - loss: 0.0190 - accuracy: 1.0000 - val_loss: 0.1865 - val_accuracy: 0.9474
Epoch 41/50
6/6 [=====] - 8s 1s/step - loss: 0.0209 - accuracy: 1.0000 - val_loss: 0.2369 - val_accuracy: 0.9211
Epoch 42/50
6/6 [=====] - 8s 1s/step - loss: 6.8878e-04 - accuracy: 1.0000 - val_loss: 0.2928 - val_accuracy: 0.9
11
Epoch 43/50
6/6 [=====] - 8s 1s/step - loss: 0.0076 - accuracy: 1.0000 - val_loss: 0.3046 - val_accuracy: 0.9211
Epoch 44/50
6/6 [=====] - 8s 1s/step - loss: 0.0162 - accuracy: 1.0000 - val_loss: 0.2526 - val_accuracy: 0.9211
Epoch 45/50
6/6 [=====] - 8s 1s/step - loss: 0.0079 - accuracy: 1.0000 - val_loss: 0.1585 - val_accuracy: 0.9474
Epoch 46/50
6/6 [=====] - 8s 1s/step - loss: 0.0103 - accuracy: 1.0000 - val_loss: 0.2041 - val_accuracy: 0.9474
Epoch 47/50
6/6 [=====] - 8s 1s/step - loss: 0.1055 - accuracy: 0.9444 - val_loss: 0.3456 - val_accuracy: 0.9211
Epoch 48/50
6/6 [=====] - 9s 1s/step - loss: 0.0327 - accuracy: 1.0000 - val_loss: 0.4836 - val_accuracy: 0.9211
Epoch 49/50
6/6 [=====] - 9s 1s/step - loss: 0.2696 - accuracy: 0.9444 - val_loss: 0.1691 - val_accuracy: 0.9474
Epoch 50/50
6/6 [=====] - 8s 1s/step - loss: 0.0223 - accuracy: 1.0000 - val_loss: 0.1744 - val_accuracy: 0.9737

```

Röviden összefoglalva:

- 12. epochig növekedtünk 89%ig
- utána elkezdünk vissza fele menni, majd megint fel, de a 25. epochra elértünk egy fix 92%ot
- utána általában le és visszaugráltunk erre az értékre vagy a közelébe
- 50. epochra pont elértük a legjobb eredményt 97%ot

Ugyanazok az előnyök, illetve hátrányok elmondhatóak az adathalmazról, mint az előzőben. A 92%ot mondanám átlagos teljesítménynek. itt is szinte biztos vagyok benne, hogy legalább 100 képpel fázisonként elérhető lett volna a 95% fölötti teljesítmény.

Összefoglalva:

Szerintem ilyen kevés képpel nem volt jó modellt csinálni, de legalább ezt is megtudtuk és ugyanakkor ennek ellenére is jól teljesített a modell.

Mindenképpen kéne még több adat, és esetleg egy gép nagyon jó GPU-val, mivel nekem elégé lassan futnak le a kiértékelések.

Bináris keresés a minimális változásra:

Itt két állapot található a legkisebb dologgal módosítva, amit találtunk, hogy vajon ezt felismeri-e.

