

Széchenyi István Egyetem
Gépészmérnöki, Informatikai és Villamosmérnöki Kar
Informatika Tanszék

SZAKDOLGOZAT

Németh Bendegúz
Mérnök Informatikus BSc szak

2019



SZÉCHENYI
ISTVÁN
EGYETEM



SZAKDOLGOZAT

Kiterjesztett valóságot támogató gépi tanulás algoritmus fejlesztése

Németh Bendegúz

Mérnök Informatikus BSc szak

2019

FELADAT-KIÍRÓ LAP SZAKDOLGOZATHOZ

Hallgató adatai

Név: Németh Bendegúz

Neptun-kód: AORBOJ

Szak: Mérnök informatikus BSc

Specializáció: -

Cím: Kiterjesztett valóságot támogató gépi tanulás algoritmus fejlesztése

Feladatok leírása:

A szakdolgozatban egy gépi tanulás algoritmus kerül megvalósításra amely becslést ad egy mobiltelefon elmozdulásának mértékére a kamera képe alapján. A tanítóhalmaz a mobiltelefon kamerájának képéből és egy milliméter pontosságú infrakamerás tracker rendszer adataiból áll. Ez alapján egy konvolúciós neurális háló kerül betanításra. Az eredmény összevetésre kerül az iOS operációs rendszerű készülékeken elérhető ARKit keretrendszer által nyújtott pontossággal.

Részfeladatok: szakirodalmi áttekintés, tanítóhalmaz és teszhalmaz összeállítása, ARKit alapú feldolgozás az adathalmazon történő benchmarkolhatóság céljából, konvolúciós háló felállítása / betanítása, eredmények kiértékelése.

Győr,

Dr. Csapó Ádám Balázs
Egyetemi docens

Dr. Hatwágner F. Miklós
Egyetemi docens, mb. tanszékvezető

SZAKDOLGOZAT ÉRTÉKELŐ LAP

Hallgató adatai

Név: Németh Bendegúz
Szak: Mérnök informatikus BSc
Tagozat: Nappali

Neptun-kód: AORBOJ

A szakdolgozat adatai

Cím: Kiterjesztett valóságot támogató gépi tanulás algoritmus fejlesztése
Nyelv: Magyar
Típus: Nyilvános

A szakdolgozat bírálatra bocsátható

A bíráló adatai

Név:
Munkahely:
Beosztás:

Dátum

Dr. Hatwágner F. Miklós
Egyetemi docens, mb. tanszékvezető

A bíráló javaslata:

Dátum

Érdemjegy

Dr. Hatwágner F. Miklós
Egyetemi docens, mb. tanszékvezető

A belső konzulens javaslata:

Dátum

Érdemjegy

Dr. Csapó Ádám Balázs
Egyetemi docens

A ZVB döntése:

Dátum

Érdemjegy

Aláírás (ZVB elnök)

NYILATKOZAT

Alulírott, Németh Bendegúz (AORBOJ), Mérnök Informatikus, BSc szakos hallgató kijelentem, hogy az Kiterjesztett valóságot támogató gépi tanulás algoritmus fejlesztése című szakdolgozat feladat kidolgozása a saját munkám, abban csak a megjelölt forrásokat, és a megjelölt mértékben használtam fel, az idézés szabályainak megfelelően, a hivatkozások pontos megjelölésével.

Eredményeim saját munkán, számításokon, kutatáson, valós méréseken alapulnak, és a legjobb tudásom szerint hitelesek.

Győr,

hallgató

KIVONAT

Kiterjesztett valóságot támogató gépi tanulás algoritmus fejlesztése

A kiterjesztett valóság a valós világba helyezett virtuális elemek elnevezése. Működéséhez szükség van egy digitális eszköz térbeli, pontos nyomon követésére. A kiterjesztett valóság mai legelterjedtebb megvalósítása okostelefon segítségével történik. A kamera képe a készülék kijelzőjén rávetített digitális tartalommal együtt jelenik meg. A dolgozat célja egy olyan gépi tanulás algoritmus fejlesztése amely a kamerából származó egymás utáni képek vizsgálatával, összehasonlításával becslést ad a képek készítése között történt elmozdulás mértékére, így támogatni képes egy rendszert amely a kiterjesztett valósághoz szükséges térbeli követést valósítja meg.

Az 1. fejezet átfogó képet nyújt a dolgozat fő részeinek elméleti háttéréről, mint a kiterjesztett valóság, a mesterséges neurális hálózatok és az optical flow, mindezt a szakirodalom áttekintésével együtt.

A 2. fejezetben a dolgozat készítése során alkalmazott főbb technológiák kerülnek bemutatásra.

A 3. fejezet részletezi a megvalósítás menetét, lépéseit. Ebben a fejezetben bemutatásra kerül a tanító- és tesztalmazok összeállítása, a modellek felállítása, betanítása, optical flow számítás végzése mobilkészüléken, illetve az alkalmazás megvalósítása az ARKit keretrendszer használatával.

A 4. fejezet ismerteti a dolgozat készítése során elért eredményeket.

ABSTRACT

Development of machine learning algorithm supporting augmented reality

Augmented Reality (AR) consists of virtual elements that are placed into the real world. AR requires the spatially accurate tracking of a digital device. Today's most widespread applications of AR are implemented on smartphones, such that the camera image appears on the display of the device with digital content projected onto it.

The aim of this thesis is to develop a machine learning algorithm that estimates the degree of displacement of the device by sequentially scanning and comparing images from the camera, thereby supporting a system that implements spatial tracking for Augmented Reality.

In Chapter 1 a comprehensive overview is given on the theoretical background of the thesis, through a review of the literature on AR, artificial neural networks and optical flow.

In Chapter 2, the main technologies used in the thesis are introduced.

Chapter 3 details the process and steps of the implementation. This chapter details how the training and test datasets were compiled, how the models developed in the thesis were set up and trained, how optical flow calculations were performed on the mobile device, and how the application was implemented using the ARKit framework.

Chapter 4 summarizes the results of the thesis.

TARTALOMJEGYZÉK

Feladat-kiíró lap szakdolgozathoz	i
Szakdolgozat értékelő lap	ii
Nyilatkozat	iii
Kivonat	iv
Abstract.....	v
Tartalomjegyzék	vi
Ábrajegyzék.....	viii
Forráskódjegyzék	x
Bevezetés	1
1 Irodalomkutatás	2
1.1 Kiterjesztett valóság.....	2
1.2 Mesterséges neurális hálózatok	3
1.2.1 Konvolúciós neurális hálózatok	6
1.2.1.1 Konvolúciós réteg	7
1.2.1.2 Padding.....	9
1.2.1.3 Aktivációs réteg (nonlinear layer).....	9
1.2.1.4 Összevonó réteg (pooling layer)	10
1.2.1.5 Dropout réteg.....	11
1.2.1.6 Network in Network réteg.....	12
1.2.1.7 Teljesen összekötött réteg (fully connected layer)	13
1.2.2 Tanítás.....	13
1.2.3 Transfer learning.....	15
1.3 Optical flow	16
2 Alkalmazott technológiák.....	17
2.1 Python	17

2.2	Project Jupyter	18
2.3	Tensorflow	19
2.4	OpenCV	20
2.5	Turi Create	21
2.6	Az iOS operációs rendszer.....	22
2.7	Swift.....	23
2.8	ARKit.....	24
2.9	CocoaPods	25
3	A megvalósítás menete	26
3.1	Képek osztályokba sorolása.....	26
3.1.1	Adatok gyűjtése, előkészítése.....	27
3.1.2	Konvolúciós háló felállítása, betanítása	29
3.2	Optical flow számítása.....	34
3.3	Készülék elmozdulásának becslése.....	38
3.3.1	Tanító adatok összeállítása	38
3.3.2	Regressziós modell elkészítése.....	43
3.3.3	Megvalósítás ARKit használatával	46
4	Eredmények kiértékelése.....	49
	Irodalomjegyzék	51
	Mellékletek	57

ÁBRAJEGYZÉK

1. ábra – Kiterjesztett valóság[2].	2
2. ábra – Milgram-féle Valóság-Virtualitás Kontinuum[3].	3
3. ábra – Kiterjesztett valóság mobilkészülékeken[4][5][6].	3
4. ábra – Neuron felépítése[7].	4
5. ábra – Aktivációs függvények[7].	4
6. ábra – Neurális hálózat[7].	5
7. ábra – Konvolúciós neurális hálózat[10].	6
8. ábra – Konvolúciós réteg[11].	7
9. ábra – Szorzat[9].	8
10. ábra – Padding[9].	9
11. ábra – Aktivációs függvények[12].	9
12. ábra – Összevonó réteg (pooling layer)[9].	10
13. ábra – Dropout réteg[13].	11
14. ábra – Network in Network réteg[14].	12
15. ábra – Teljesen összekötött réteg[16].	13
16. ábra – Költség minimalizálás[18].	14
17. ábra – Transfer learning[19].	15
18. ábra – Optical flow[32].	16
19. ábra – Python[23].	17
20. ábra – Project Jupyter [38].	18
21. ábra – Tensorflow[26].	19
22. ábra – OpenCV[41].	20
23. ábra – Turi Create[42].	21
24. ábra – Az iOS operációs rendszer[21].	22
25. ábra – Swift[29].	23

26. ábra – ARKit[31].	24
27. ábra – CocoaPods[45].	25
28. ábra – image_downloader.py futtatása.	27
29. ábra – searchlist.txt tartalma.	27
30. ábra – Felismert tárgyak.	28
31. ábra – Automator folyamat.	29
32. ábra – Virtuális környezet létrehozása.	30
33. ábra – Adatok vizualizációja.	31
34. ábra – Resnet tanítási folyamat.	32
35. ábra – Squeezenet tanítási folyamat.	33
36. ábra – Modell tesztelése.	34
37. ábra – Modell be- és kimenetei.	34
38. ábra – Wrapper és Bridging fájlok.	35
39. ábra – OptiTrack infrakamerák[55].	40
40. ábra – OptiTrack Motive alkalmazás[56].	40
41. ábra – iOS alkalmazásból származó adatok.	41
42. ábra – OptiTrack rendszerből származó adatok.	42
43. ábra – Tanításra használt adatok végleges formája.	43
44. ábra – Túltanulás (Overfitting).	45
45. ábra – Modell tesztelése.	46
46. ábra – iOS alkalmazás képernyőképek.	49

FORRÁSKÓDJEGYZÉK

1. forráskód – SFrame formátumra alakítás.	31
2. forráskód – Tanító és teszt adat bontás.	32
3. forráskód – Konkrét modell meghatározása.	33
4. forráskód – Modell tesztelése.	33
5. forráskód – Modell mentése.	34
6. forráskód – Podfile.	35
7. forráskód – Mat formátumra alakítás.	36
8. forráskód – Sarok detektálás.	36
9. forráskód – Optical flow számítás.	37
10. forráskód – data.csv exportálása.	39
11. forráskód – Adatok beolvasása.	43
12. forráskód – Adatok bontása.	44
13. forráskód – Regresszió automatikus modell választással.	44
14. forráskód – Döntési Fa mélység meghatározással.	44
15. forráskód – Boosted Regression Trees paraméterekkel.	45
16. forráskód – Regressziós modell tesztelése.	46
17. forráskód – Regressziós modell mentése.	46
18. forráskód – ARKit használata.	47

BEVEZETÉS

A kiterjesztett valóság (augmented reality, AR) a valós világba helyezett virtuális elemek elnevezése. Működéséhez szükség van egy digitális eszköz térbeli, pontos nyomon követésére. Az AR mai legelterjedtebb megvalósítása okostelefon segítségével történik. A kamera képe a készülék kijelzőjén rávetített digitális tartalommal együtt jelenik meg. A készülék térbeli követéséhez általában több szenzorból, mint giroszkópból, gyorsulásmérőből származó adatokat használnak, illetve a kamera képének felhasználása is elterjedt erre a célra.

A dolgozat célja egy olyan gépi tanulás algoritmus fejlesztése amely a kamerából származó egymás utáni képek vizsgálatával, összehasonlításával becslést ad a képek készítése között történt elmozdulás mértékére, így támogatni képes egy rendszert amely a kiterjesztett valósághoz szükséges térbeli követést valósítja meg.

A tanuló algoritmus megvalósításához a kezdeti elképzelések szerint a Google által fejlesztett nyílt forrású TensorFlow szoftverkönyvtár került volna felhasználásra, viszont a dolgozat írása során a Turi Create keretrendszer alkalmasabbnak bizonyult erre a célra. A tanítóhalmaz a mobiltelefon kamerájának képéből és egy milliméter pontosságú infrakamerás tracker rendszer adataiból áll. Az eredmények összevetésre kerülnek az iOS operációs rendszerű készülékeken elérhető ARKit keretrendszer által nyújtott pontossággal. Az elméleti háttérrel és a főbb technológiákról részletesebb leírás található a következő két fejezet alpontjaiban.

1 IRODALOMKUTATÁS

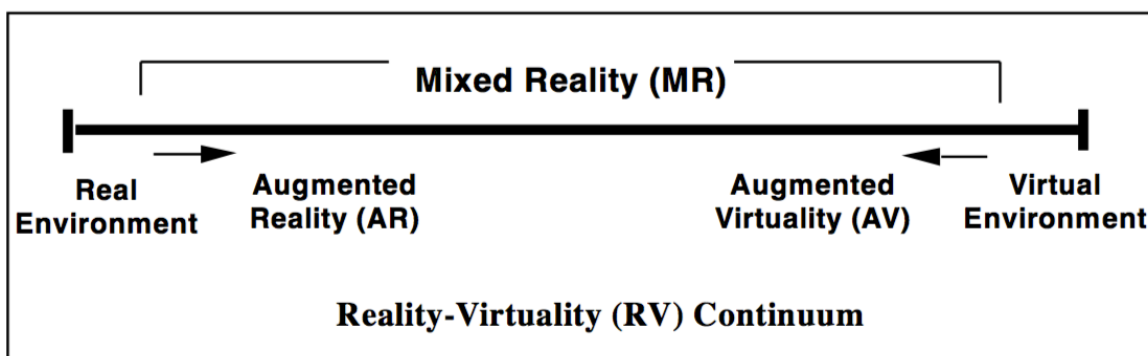
1.1 Kiterjesztett valóság

A kiterjesztett valóság gondolata egészen az első számítógépek idejére nyúlik vissza. Az „Augmented Reality” kifejezést először Tom Caudell használta 1992-ben, aki a Boeing számára készített oktató alkalmazást. Ahhoz, hogy a mindennapi felhasználókat is elérje a technológia azonban még jóval költséghatékonyabbnak kellett lennie. A kiterjesztett valóság, digitális információ és a valós környezet egymásra helyezése. Egyik első gyakorlati felhasználása ami tömegeket is el tudott érni, a sportközvetítésekkor a kamera képére helyezett extra információ volt[1].



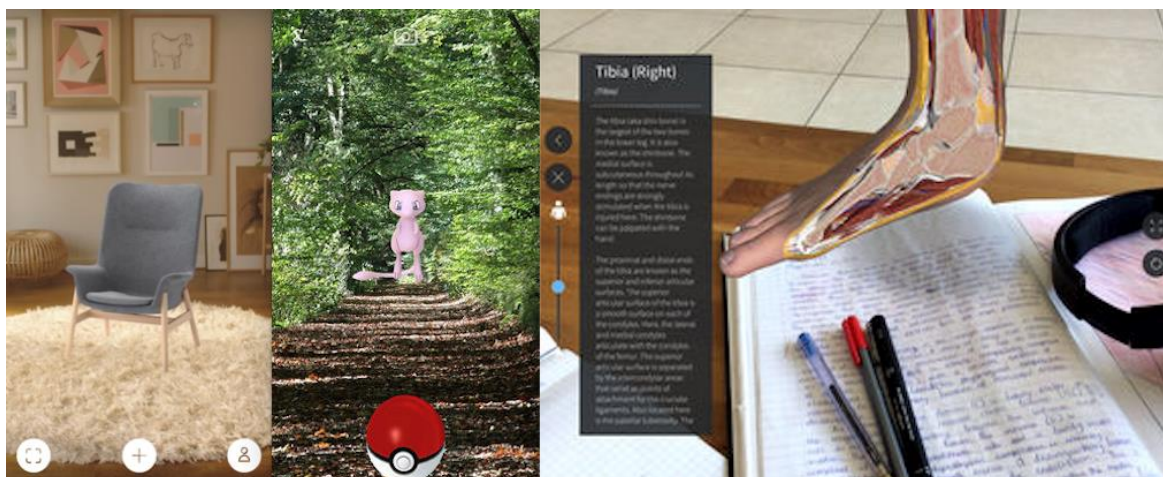
1. ábra – Kiterjesztett valóság[2].

Több definíció is létezik a kiterjesztett valóságra. Ronald Azuma szerint digitális eszközök segítségével, virtuális elemek a valós világra történő rétegezésével jön létre[1].



2. ábra – Milgram-féle Valóság-Virtualitás Kontinuum[3].

Paul Milgram és Fumio Kishino egy Valóság-Virtualitás Kontinuumot javasoltak a következő fogalmak definiálásához. A két végpontban helyezkedik el a valós illetve a virtuális környezet. Ezen két véglet között található a kevert valóság amelynek a valós környezethez közelebbi szakasza a kiterjesztett valóság, a virtuális környezethez közelebbi pedig a kiterjesztett virtualitás[1].



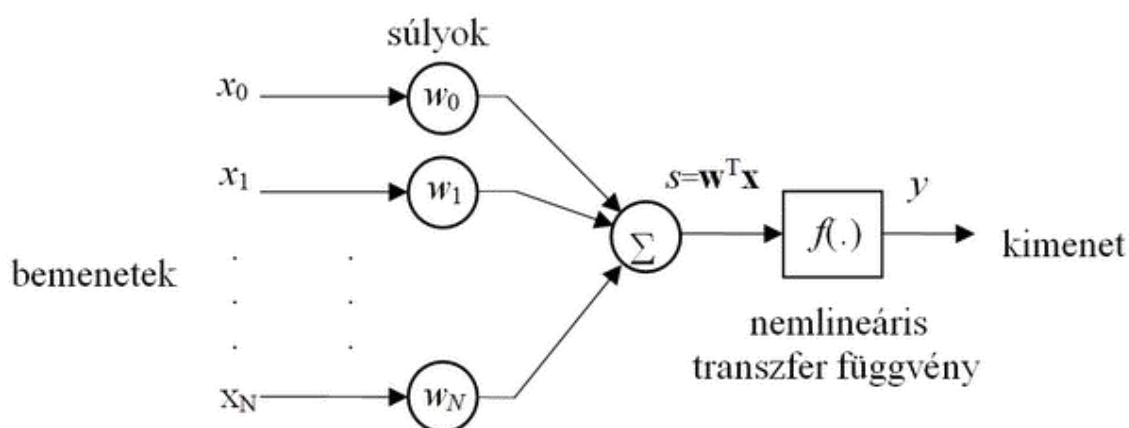
3. ábra – Kiterjesztett valóság mobilkészülékeken[4][5][6].

Napjainkban, kiterjesztett valóság bárki számára hozzáférhető amennyiben rendelkezik egy okostelefonnal. A kamera veheti a valós világot, míg a készülék kijelzőjén már egy virtuális elemekkel kiegészített kép látható. Ez a technológia egyaránt használható marketing, szórakoztató és oktatási célokra is[1].

1.2 Mesterséges neurális hálózatok

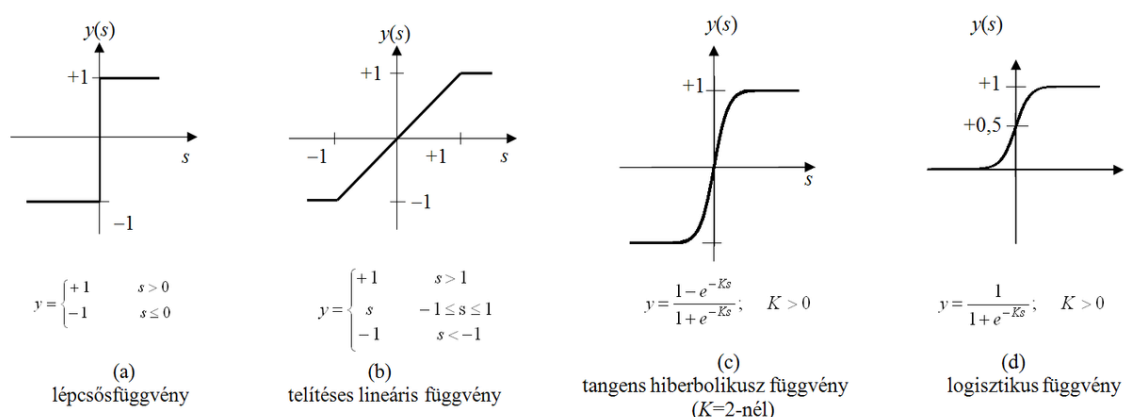
Évtizedekkel ezelőtt, főleg biológiai kutatások eredményeképpen merült fel az a

gondolat, hogy a természetes neurális hálózatok mintájára létrehozható mind hardver, mind szoftver. Ezek olyan rendszerek amelyek mintákból, példákból képesek tanulni, általánosítani majd feladatmegoldó képességet kialakítani. A neurális hálózatok számos feladat megoldásánál akár jobbnak is bizonyulnak a hagyományos algoritmikus rendszereknél. Nagymértékben párhuzamos felépítésük és tanulási képességük különbözteti meg őket leginkább a korábban használtaktól. A neurális hálózat általában nagyszámú, lokális feldolgozást végző műveleti elem, rendezett, összekapcsolt rendszere illetve rendelkezik tanulási és előhívási algoritmussal[7].



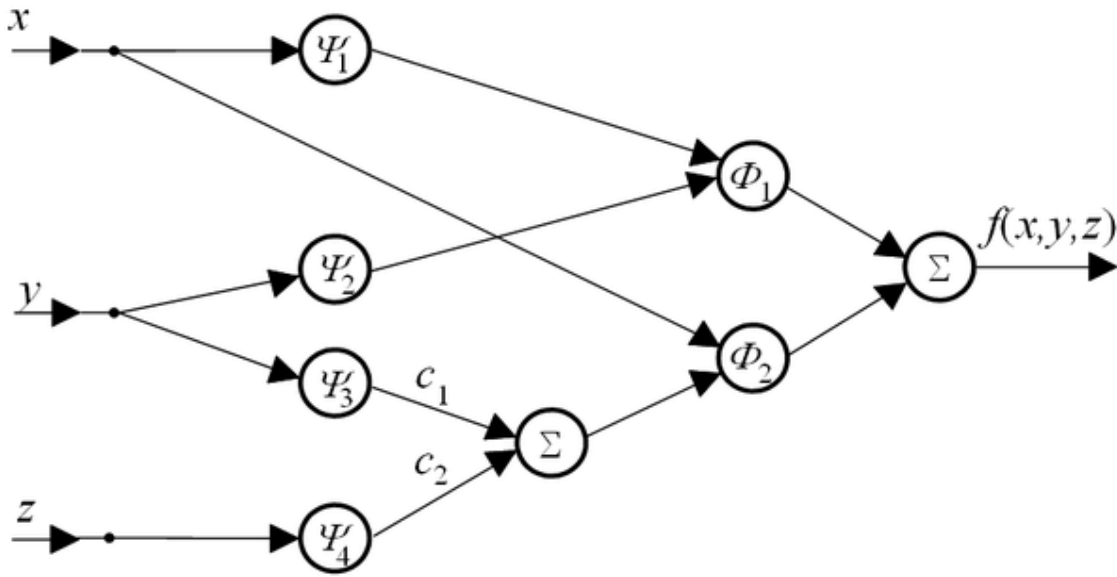
4. ábra – Neuron felépítése[7].

Egy műveleti elem vagy neuron egy több-bemenetű, egy-kimenetű eszköz, amely a bemenetek és a kimenet között általában valamilyen nemlineáris leképezést valósít meg[7].



5. ábra – Aktivációs függvények[7].

A bemeneti értékekből az aktuális kimeneti értéket egy tipikusan nemlineáris függvény alkalmazásával hozza létre, melyet aktivációs függvénynek nevezünk[7].



6. ábra – Neurális hálózat[7].

A neuronok összekapcsolásával hozhatóak létre a neurális hálók amelyeknek méretét, elrendezését a megoldandó feladat határozza meg. Az egyes változókhoz tartozó szorzó tényezők a súlyok, illetve egy változó nélküli tag végzi a középponttól való eltolást, ezt nevezzük bias-nak[7].

A gépi látásra használt neurális hálózatok bemenete általában egy három dimenziós mátrix ahol két dimenzió a pixeleket, a harmadik pedig az RGB színcsatornákat reprezentálja. Ennek a mátrixnak minden eleme 0 és 255 közötti értéket vehet fel az RGB kódolásnak megfelelően. Ha minden egyes elemet, egy adott pixel egy adott színcsatornájának értékét egy szabadságfokként értelmezzük akkor egy kellően sokdimenziós térben egy pont reprezentálja minden képünket. Belátható hogyha egyetlen dimenziónak az értékét eggyel változtatjuk meg akkor az egyik szomszédos pontot kapjuk. Ebből következik, hogy egy kép egy pixelének egyetlen színcsatornájának értékét eggyel megváltoztatva pont a szomszédos pontot fogjuk kapni az adott térben.

$$\sqrt{\sum_{i=1}^k (x_i - y_i)^2}$$

1. képlet – Euklidészi távolság.

$$\sum_{i=1}^k |x_i - y_i|$$

2. képlet – Manhattan távolság.

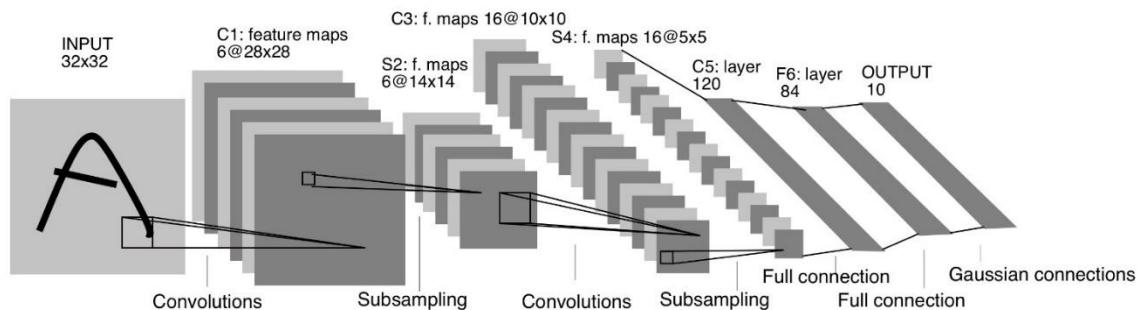
$$\left(\sum_{i=1}^k (|x_i - y_i|)^q \right)^{1/q}$$

3. képlet – Minkowski távolság.

Sokdimenziós terek pontjai közötti távolság mérésére több módszer is létezik például a Manhattan-távolság, euklideszi távolság, Minkowski távolság[8].

1.2.1 Konvolúciós neurális hálózatok

Az eddig leírtaknál fennáll az a probléma hogy minden egyes bemenet értéket, egy pixel egy színcsatornájának értékét, önmagában vizsgáltuk. Ha a bemeneti mátrixunkat mindig ugyanazzal a módszerrel egy vektorrá alakítanánk, akkor is ugyanezt az eredményt érnénk el. Ezzel a módszerrel az azonos tárgyakat tartalmazó képek egyáltalán nem biztos, hogy közel esnek egymáshoz az adott vektortérben. Sok egyéb mellett ezt a problémát is megoldják a konvolúciós neurális hálózatok. Annak köszönhetően, hogy a képet nem egészében, hanem részenként vizsgálják és primitívekből, köztes tulajdonságokból építik fel a tárgyakat. Az ötlet biológiai hátterét Hubel és Wiesel 1962-es kísérlete jelentette. Bebizonyították hogy a látóközpont kis részei figyelnek különböző formákra, mintázatokra, élekre, egy adott szögű élre mindig ugyanazok a neuronok aktiválódnak illetve hogy vannak amik a látótér különböző részeire érzékenyek[9].



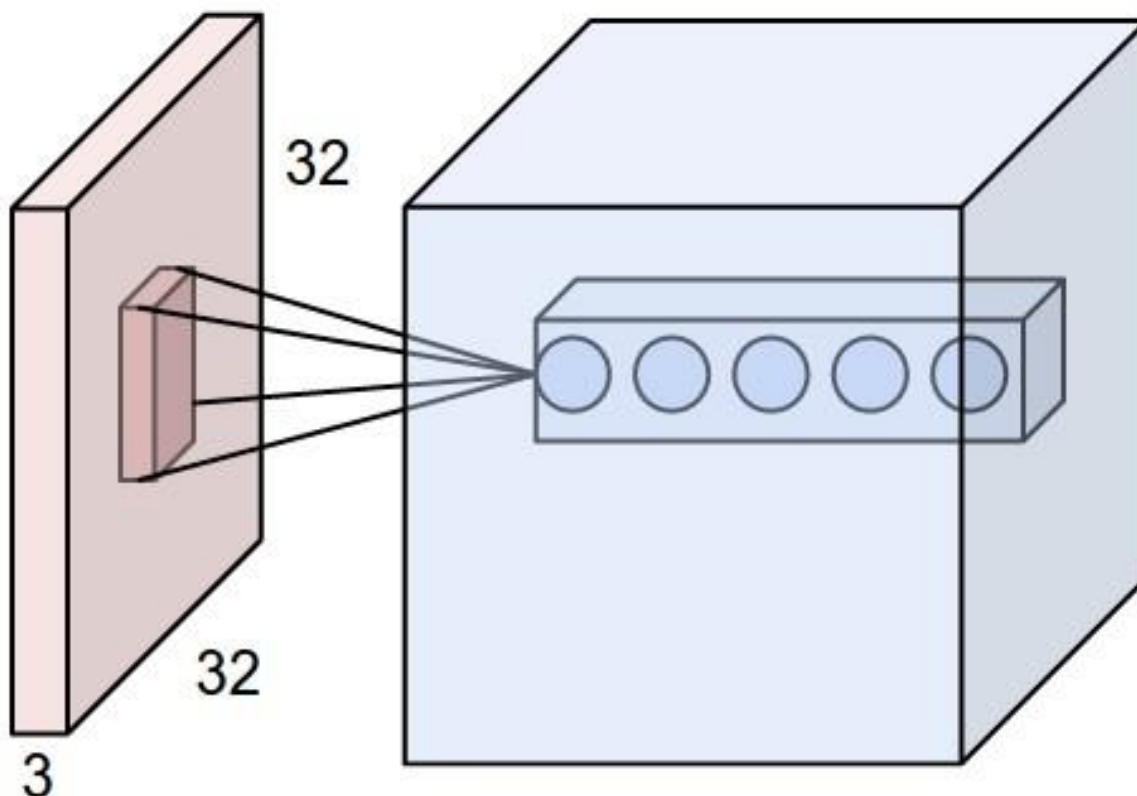
7. ábra – Konvolúciós neurális hálózat[10].

Az első, gyakorlatban is használt ilyen háló Yann LeCun nevéhez fűződik, a LeNet-5

32x32 pixeles képeken különböztetett meg számokat. Számos bank, postahivatal használta csekkek, irányítószámok értelmezésére[10].

A neurális hálók ezen fajtája több különböző funkciójú és felépítésű réteget tartalmaz.

1.2.1.1 Konvolúciós réteg

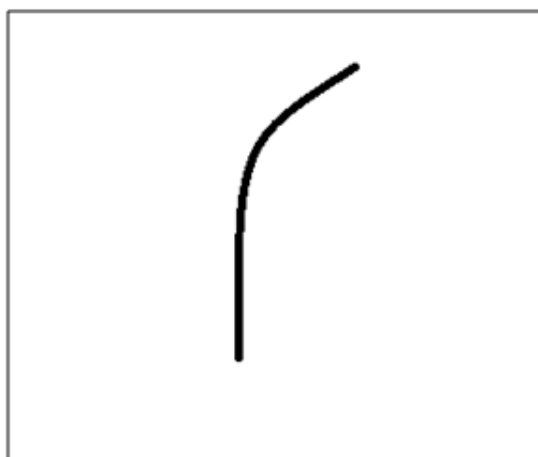


8. ábra – Konvolúciós réteg[11].

Az első réteg egy konvolúciós neurális hálózatban mindig egy konvolúciós réteg. Az ábrán látható 32x32x3 méretű mátrix maga a bemeneti kép. Ennek kiemelt része a filter vagy más néven kernel, ez pásztázza végig az egész képet. Mérete egyéni döntés kérdése, minél nagyobb, a kép annál nagyobb területéről fog információt gyűjteni az első réteg esetében. Mélyebben elhelyezkedő rétegeknél is több információt jelent egy nagyobb filter, de ott már az előző réteg által megállapított tulajdonságok kombinációját figyeli. A filter által egyszerre figyelt értékek mindegyikéhez tartozik egy további érték ami lesz a változó, a súly vagy más néven paraméter[9].

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

9. ábra – Szorzat[9].

A figyelt érték és a súly szorzatából derül ki mennyire fontos ez az érték ebben a konkrét esetben, például ha a súlyunk nulla, akkor a szorzat is nulla. Ezzel a módszerrel képes a háló az első rétegekben például éleket, nagyobb filter esetén mintákat, mélyebben elhelyezkedő rétegek esetén pedig magasabb szintű, bonyolultabb tulajdonságokat megjegyezni[9].

Belátható hogy egy ilyen súlymátrix-al egy bizonyos tulajdonságot képes keresni a háló, ezért van több ilyen pásztázásra szükség, eltérő súlyokkal amik más-más tulajdonságot keresnek. Egy ilyen pásztázás adja az eredmény mátrix egy rétegét, a pásztázások számával lesz megegyező a mátrix z dimenziójában található értékek száma.

1.2.1.2 Padding

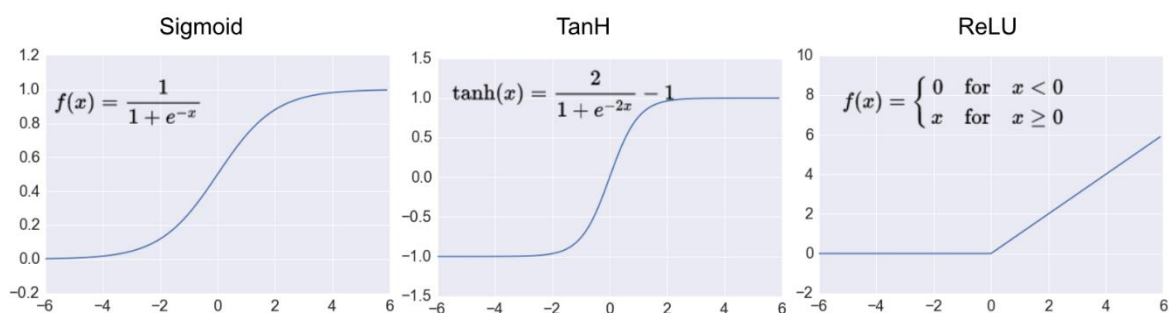
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0							0	0
0	0							0	0
0	0							0	0
0	0							0	0
0	0							0	0
0	0							0	0
0	0							0	0
0	0							0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

10. ábra – Padding[9].

Minél nagyobb a filter mérete illetve a stride (a lépések mértéke) annál kevesebb értékéből fog állni a kimenet. Ennek megakadályozására alkalmas a padding, aminek a lényege, hogy a bemenetet körben nullákkal egészítjük ki.

1.2.1.3 Aktivációs réteg (nonlinear layer)

Az aktivációs rétegek szüntetik meg a linearitást a hálóban, eddig lényegében minden műveletünk kimenete lineárisan következett a bemenetéből.

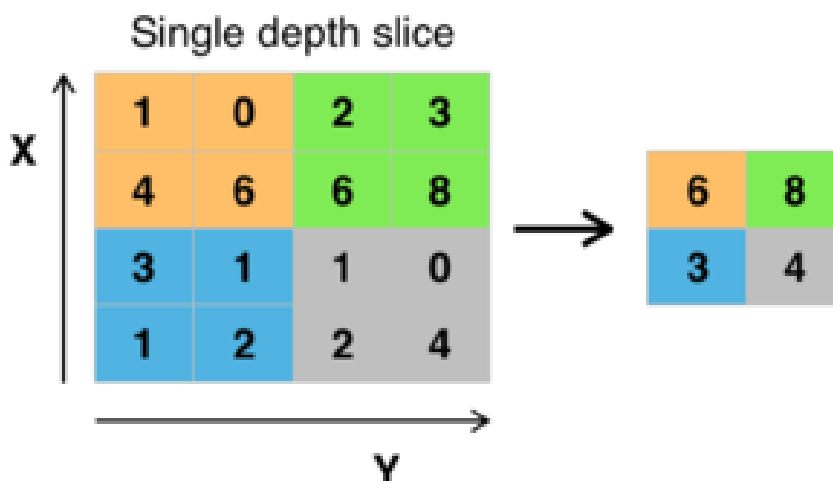


11. ábra – Aktivációs függvények[12].

Több függvény is létezik ami alkalmas erre a célra. A legújabb és egyben legelterjedtebb a ReLU (Rectified Linear Unit). Lényegében ahol a konvolúciós réteg

kimenete negatív, ott nulla lesz, ahol pozitív, ott megtartja eredeti értékét. Előnye a többivel szemben, hogy a pontosság megtartása mellett sokkal kisebb a számítási igénye, ezáltal gyorsabban lehetséges a háló tanítása.

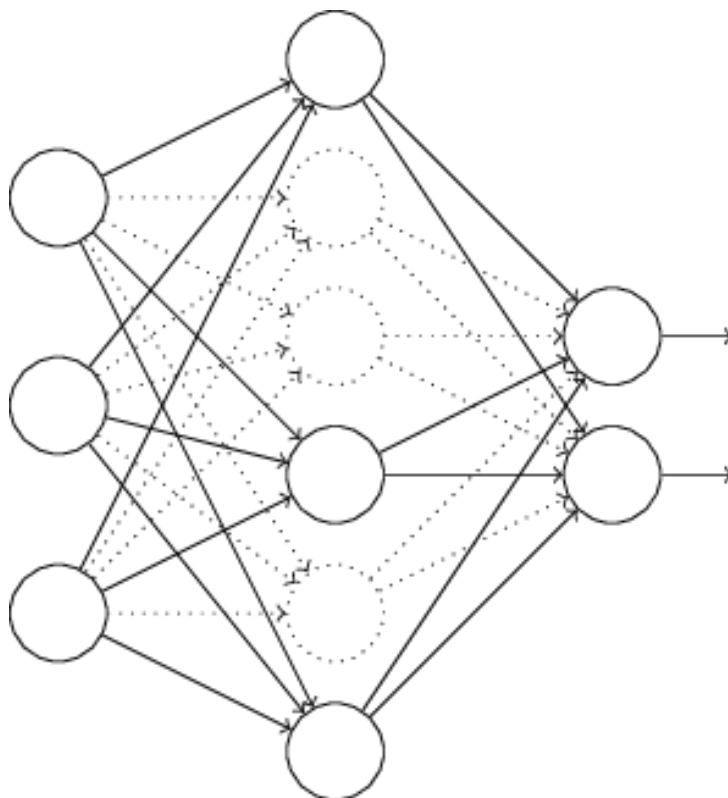
1.2.1.4 Összevonó réteg (pooling layer)



12. ábra – Összevonó réteg (pooling layer)[9].

Itt a filter a konvolúciós réteghez hasonlóan végigmegy az értékeken, viszont ez egy térben alulmintavételező réteg, ami diszkrét ugrások szerint vizsgálja a bemenet egy-egy diszjunkt területét és a területen lévő értékek maximumát veszi. Az előzőekben figyelembe vett érték melletti értéktől folytatja a mintavételt. A legelterjedtebb eset amikor a maximális értéket viszi tovább, ez látható a képen is. Az elképzelés e mögött, hogy ha már tudjuk, hogy egy tulajdonság szerepel a képen, akkor nem olyan fontos, hogy pontosan hol. Így például a képen látható 2x2 filterekkel 75%-al lehet csökkenteni a paraméterek számát, ami a számítási igényben is nagy változást eredményez. Léteznek más pooling rétegek is, vehetjük akár az értéket átlagát.

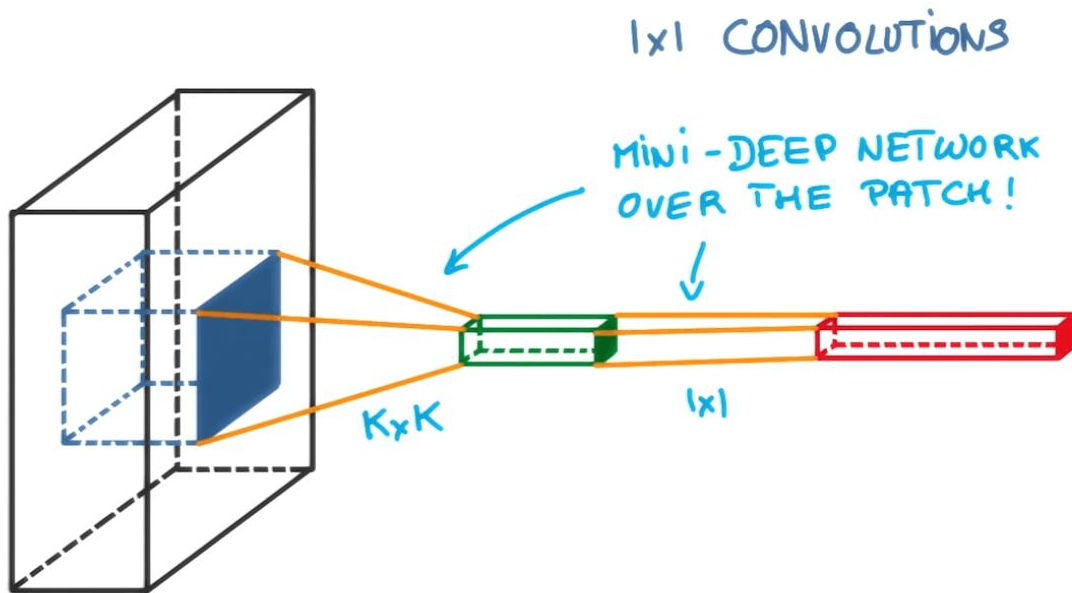
1.2.1.5 Dropout réteg



13. ábra – Dropout réteg[13].

A dropout réteg lényege, hogy random neuronokat hagyunk el. Ezen döntés mögött áll, hogy rákényszerítsük a hálót az általánosításra. Ha kevés adattal tanítunk egy nagyon összetett hálót akkor extrém esetben akár arra is képes, hogy egy az egyben megjegyezze a bemenetére kapott adatokat. Ebben az esetben a tesztelés során, amikor eddig ismeretlen adatot kap, rosszul fog teljesíteni, ezt nevezzük túltanulásnak. A dropout technikát általában a fully connected rétegeknél alkalmazzák, mivel ott a háló többi részéhez képest rengeteg neuron és kapcsolat található.

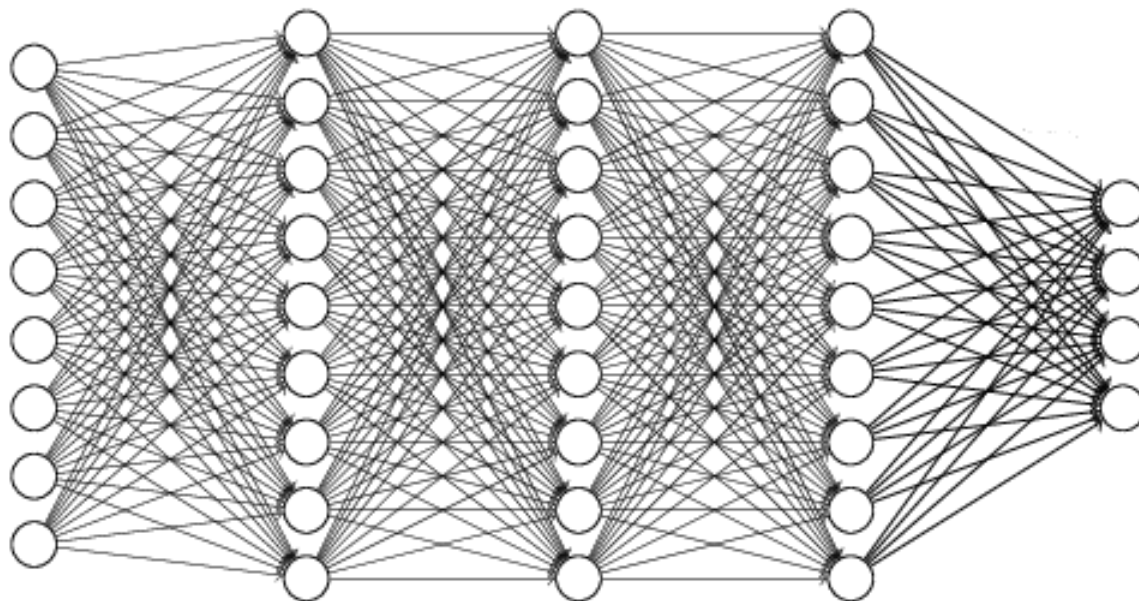
1.2.1.6 Network in Network réteg



14. ábra – Network in Network réteg[14].

A network in network réteg lényegében egy 1×1 -es konvolúciós réteg. Ennek a rétegnek a megalkotása adta az inspirációt a Google mérnökei számára az Inception hálók elkészítéséhez. A pooling réteghez hasonlóan a paraméterek számát csökkenti, csak egy másik dimenzió mentén. Az 1×1 konvolúció bemenete lesz egy $1 \times 1 \times K$ vektor ahol K a mátrix mélysége, ez ad egy súlyvektor segítségével egy értéket a kimenetben, majd ahogy az 1×1 -es filter végigpáztázza a bemenetet, képződik a kimeneti mátrix egy rétege. Több súlyvektor segítségével, a páztázást többször lefuttatva, épül fel a mátrix a harmadik dimenzió mentén. Ha a bemenet ezen dimenziója 24 értékből állt, akkor 12 páztázással a paraméterek száma felére csökkent[15].

1.2.1.7 Teljesen összekötött réteg (fully connected layer)



15. ábra – Teljesen összekötött réteg[16].

Egy fully connected réteg minden neuronja csatlakozik az előtte lévő réteg minden neuronjához. Általában egy, maximum három ilyen réteg helyezkedik el a háló utolsó rétegeiként mivel rengeteg erőforrást igényelnek. Az utolsó réteg pontosan annyi neuront tartalmaz amennyi kategóriát meg akarunk egymástól különböztetni. A funkciója eldönteni, hogy a legmagasabb szintű tulajdonságok közül melyik mennyire fontos egy kategória megállapításához.

1.2.2 Tanítás

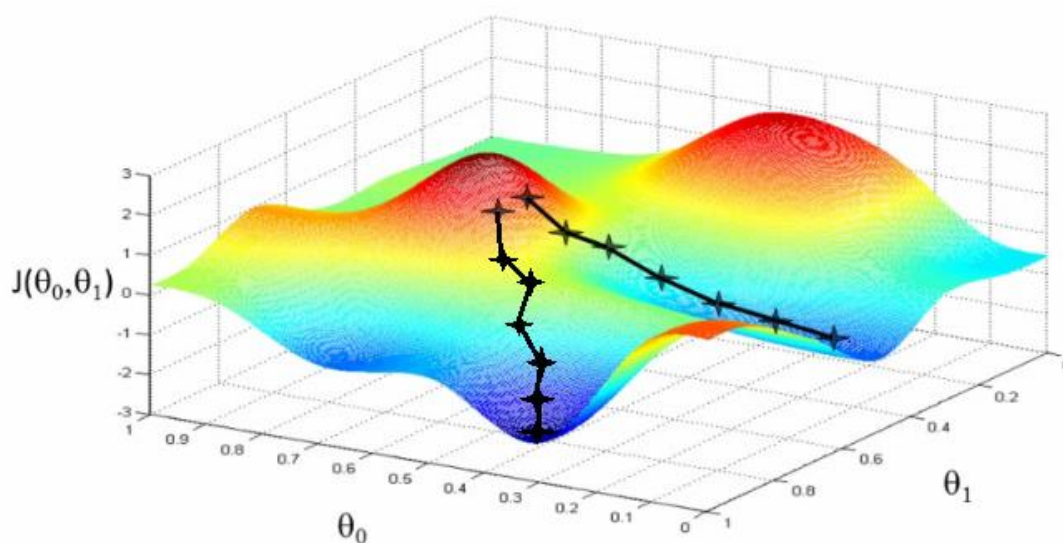
A betanítás célja a háló paramétereinek, súlyainak olyan értékre történő beállítása, hogy a feladatra jellemző bemenetekre a célnak megfelelő kimenetet adja. Léteznek felügyelt és nem felügyelt tanítási feladatok. Felügyelt esetben a backpropagation algoritmus használata a legelterjedtebb. A kiinduló értékek általában random választás eredményei.

A backpropagation folyamata több részre bontható. A hálóban előre irányuló rész során a bemeneti értékekből kiindulva, azon a megfelelő műveleteket elvégezve (például a súlyokkal való szorzás majd szummázás, az aktivációs függvényekbe való behelyettesítés) megkapjuk a kimenetek értékeit (például ennek az első lezajlásakor valószínűleg ugyanaz a valószínűség lesz minden kimeneten a random kezdőértékek miatt). Ez alapján lehet számítani egy költséget. Kategorikus kimeneteknél a leggyakrabban használt

költségfüggvény a keresztentrópia[17], ahol $p(x)$ az x bemenetre elvárt, $q(x)$ pedig az x bemenetre kapott valószínűség (így ahol az elvárt valószínűség relatíve magas, ott relatíve nagyobb büntetés jár az alacsony kapott valószínűségért).

$$H(p, q) = - \sum_x p(x) \log q(x)$$

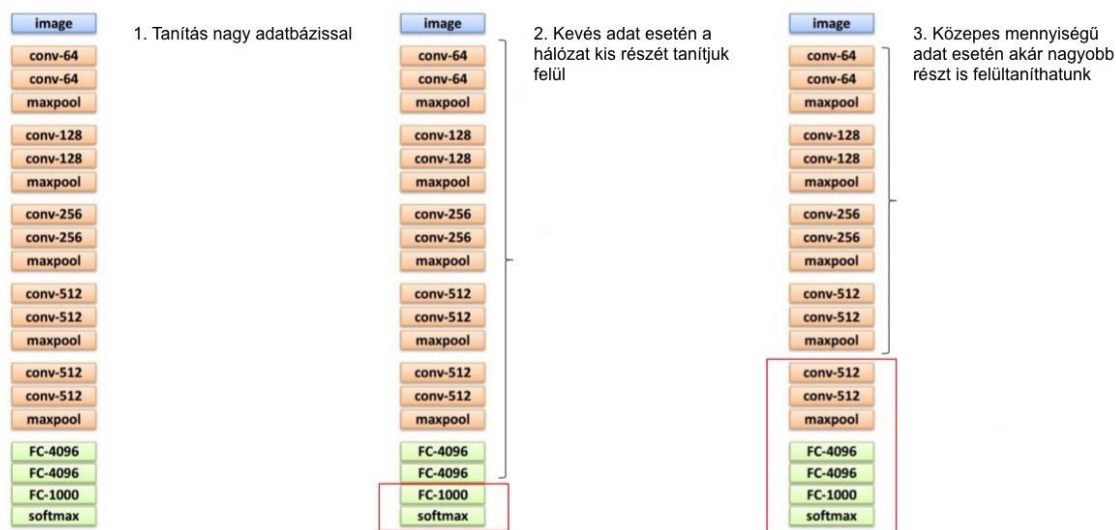
4. képlet – Keresztentrópia.



16. ábra – Költség minimalizálás[18].

A súlyok olyan módosítása a cél, hogy a költség minimalizálódjon. Ezen az ábrán Θ -k a súlyok (az egyszerűség kedvéért csak kettő van belőle), $J()$ pedig a költségfüggvény. A matematikában a deriválás pontosan arra alkalmas amire itt szükség van. Ennek a síknak egy pontjához húzott érintő meredekségéből megállapítható, merre érdemes változtatni a súlyokat ahhoz, hogy a költség csökkenjen, ez a gradiens módszer. A learning rate egy egyszerű szorzótényező amivel ennek a lépésnek a mértéke változtatható.

1.2.3 Transfer learning



17. ábra – Transfer learning[19].

Egy gyakorlatban is használható hálózat kiinduló állapotból történő betanítása rengeteg adatot és hatalmas erőforrásokat igényel. Ezt a problémát oldja meg a transfer learning módszere. Kiinduló állapota egy már korábban betanított modell. Általában érdemes olyan modellt választani ami a saját feladatunkhoz hasonló adattal lett tanítva. Nagy általánosságban jó kiindulópont egy a 14 millió képből és több mint ezer kategóriából álló ImageNet adatbázissal betanított modell. A transfer learning eljárás azon alapul, hogy alacsony szintű tulajdonságok mint élek, azonosak lehetnek más kategóriákban is, így ha egy háló első rétegei már megtanulták felismerni ezeket, akkor felhasználhatóak más célból is. Egy másik előnye ennek a módszernek, hogy jóval kisebb adatbázis is elegendő a tanításhoz. Gyakorlatban általában elegendő az utolsó, vagy az utolsó pár réteg tanítása. Az eljárás lehetővé teszi, hogy egy viszonylag nagy hálózat mint az inception v3 betanítási kísérlete fél órán belül végbe menjen egy átlagos laptop processzorral. Ez már elegendően rövid idő ahhoz, hogy ki lehessen kísérletezni a saját feladathoz megfelelő paramétereket[9][11][20].

1.3 Optical flow



18. ábra – Optical flow[32].

A mozgás detektálásának egyik közkedvelt és széleskörűen használható eszköze az optical flow, mely egy képsorozat két egymást követő képkockája között végbemenő mozgások vektorait tartalmazó mező[33]. Az optical flow felületek, élek, sarkok mozgásának mintája, ami a megfigyelő és a látottak közötti relatív mozgásból következik[34]. Az ábrán látható példán egy autópályán haladó járművek mozgása könnyen megfigyelhető. Az optical flow detektálásról elsők között K. Prazdny 1979-ben megjelent írásában olvashatunk[35]. Ezek után több módszer is született optical flow alkalmazására, ilyen például B. K. P. Horn és B. G. Schunk 1981-ben kiadott publikációja[36], illetve a még ma is közkedvelt és sokszor alkalmazott módszer amit B. D. Lucas és T. Kanade 1981-ben publikált cikkükben mutattak be[37].

2 ALKALMAZOTT TECHNOLÓGIÁK

2.1 Python



19. ábra – Python[23].

A Python egy nyílt forrású, általános célú, magas szintű programozási nyelv. Alapjait Guido van Rossum holland programozó kezdte el fejleszteni 1989 végén, majd 1991-ben hozták nyilvánosságra. Elsődleges célja az olvashatóság és a programozói munka megkönnyítése. A Python támogatja az objektumorientált, a procedurális, a funkcionális, és az imperatív programozási paradigmákat is. Az alkalmazások készítéséhez és futtatáshoz szükséges programok és dokumentációk ingyenesen letölthetők a www.python.org weboldalról. Felhasználása, módosítása és terjesztése korlátozás nélkül szabadon megtehető. Üzleti céllal is szabadon használható. A Python nyelven készített programok portábilisek, minden olyan operációs rendszeren futtathatók, amelyre készült interpreter, mint például Unix változatok, macOS és különböző Windows verziók[24][25].

2.2 Project Jupyter



20. ábra – Project Jupyter [38].

A Project Jupyter egy nonprofit szervezet, amely teljes mértékben nyílt forráskódú szoftverek fejlesztésével foglalkozik. 2014-ben alapította Fernando Pérez. Jupyter Notebook nevű termékük interaktív módon teszi lehetővé kód írását és futtatását számos programozási nyelven. A céljaik közé tartozik, hogy termékeik minden programozási nyelvet támogassanak, teljesen nyílt forráskódúak és ingyenesek maradjanak[39]. A Jupyter Notebook segítségével Python kódok és futási eredmények egy weboldalba ágyazhatók. Egy oldal cellákból épül fel. A cella tartalmazhat fejléct (header), leírást (markdown) és kódot (code). A cellák egymás után végrehajthatók. Kód esetén az eredményt az oldalba beágyazva kapjuk meg[40].

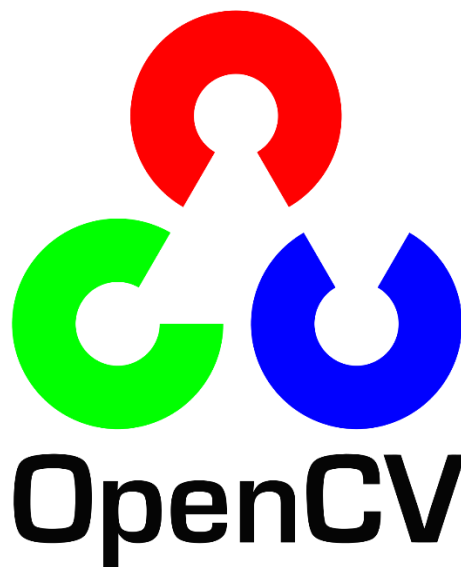
2.3 Tensorflow



21. ábra – Tensorflow[26].

A TensorFlow egy Google által fejlesztett, 2015. november 9-én nyilvánosságra hozott nyílt forráskódú dataflow programozási paradigmára épülő szoftverkönyvtár. A TensorFlow-val fejlesztett algoritmusok reprezentálhatóak egy gráffal aminek csomópontjai matematikai műveletek, míg élei a közöttük közölt többdimenziós adatrácsok (tenzorok). Az architektúra lehetővé teszi gépi tanuláshoz és deep learning-hez szükséges számítások végrehajtását egy vagy több processzoros rendszerekben és GPU-n egyaránt. Mind szerver, mind mobileszközön egyetlen API segítségével használható. Eredetileg a Google mesterséges intelligenciát kutató szervezetének Google Brain nevű csapata fejlesztette. Kiindulópontjának a Geoffrey Hinton által vezetett csapat 2009-es backpropagation implementációja tekinthető, majd a 2011-ben megjelent neurális hálókra épülő DistBelief gépi tanulás rendszer amely a TensorFlow elődjének nevezhető. 2016 májusában a Google külön hardver elemet, TPU (tensor processing unit) mutatott be ami kifejezetten TensorFlow felhasználásával írt algoritmusok futtatására nagyságrendekkel hatékonyabbnak bizonyult a korábban használt videokártyáknál. A TPU-k 2018 február óta elérhetőek a Google Cloud Platform-on[26][27][28].

2.4 OpenCV



22. ábra – OpenCV[41].

Az OpenCV egy nyílt forráskódú számítógépes látásra és gépi tanulásra kifejlesztett szoftverkönyvtár, amely több mint 2500 optimalizált, C és C++ nyelveken írt, rendkívül rugalmasan alkalmazható algoritmust tartalmaz. A projekt 1999-ben az Intel orosz kutatói központjában indult, eredetileg C interface-t használt, azonban az OpenCV 2.0. (2009) óta a fő fejlesztési irány a C++ interface. Az indulásától kezdve cél volt a valós idejű feldolgozás. A későbbiek során a Willow Garage és az Itseez is támogatta, majd az OpenCV.org non-profit alapítvány 2012-ben vette át a projektet. C++, Python és Java interface érhető el hozzá és elérhető Windows, Linux, macOS, iOS és Android operációs rendszereken is[41].

2.5 Turi Create



23. ábra – Turi Create[42].

A Turi egy főleg gépi tanulásra létrehozott keretrendszer. Teljes egészében nyílt forráskódú és C++ nyelven írt, viszont Python interface is elérhető hozzá. Korábbi neve Dato, előtte pedig GraphLab volt. A GraphLab projektet Carlos Guestrin indította a Carnegie Mellon Egyetemen 2009-ben, majd 2013-ban hozta létre a GraphLab Inc. vállalatot, hogy biztosítsa a keretrendszer további fejlesztését. 2016 augusztus 5.-én került az Apple Inc. tulajdonába[43]. A 2018-ban elérhető verzió többek között alkalmas klaszterező, osztályozó, regressziót végző és ajánló algoritmusok létrehozására is. A betanított modellek átalakíthatóak az Apple által létrehozott mlmodel formátumra, így közvetlenül használhatóak kiértékelésre iOS operációs rendszeren[44].

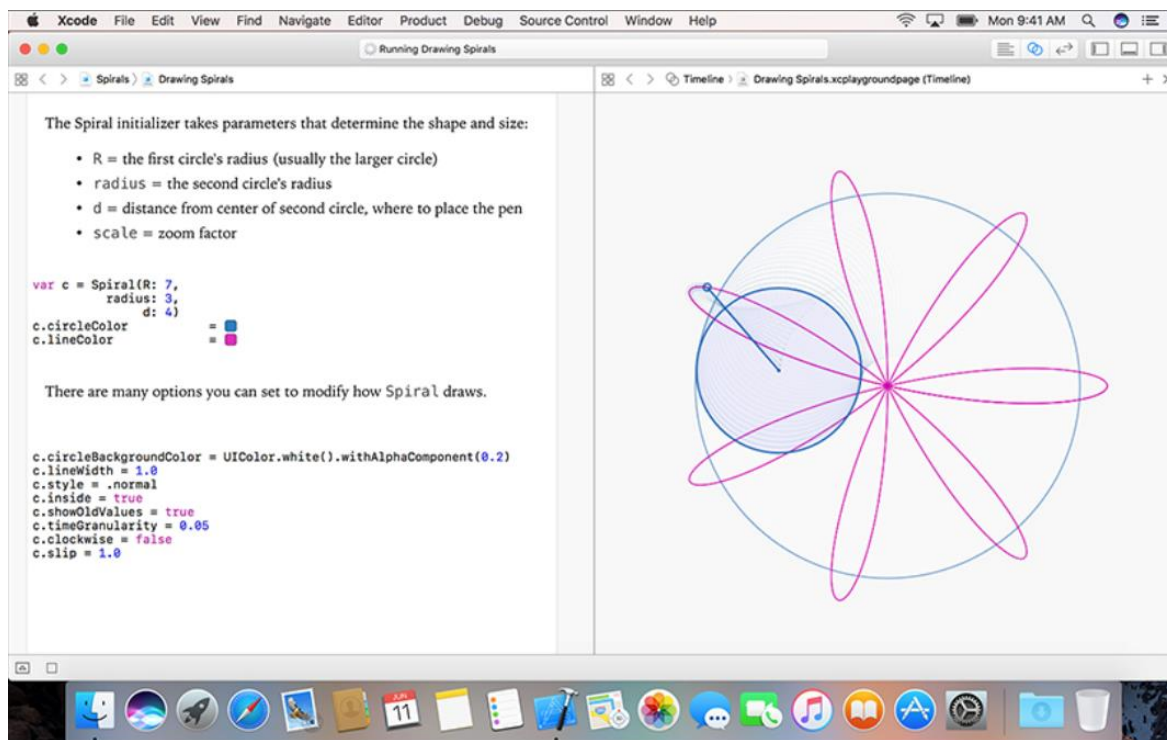
2.6 Az iOS operációs rendszer



24. ábra – Az iOS operációs rendszer[21].

Az iOS az Apple mobil operációs rendszere amit kizárólag saját hardverekre, mint az iPhone, iPad és iPod Touch készítettek, valamint teljesen zárt forráskódú. iOS operációs rendszerű készülékekre alkalmazás kizárólag az App Store-ból telepíthető. Az alkalmazásoknak szigorú ellenőrzésen kell átesniük mielőtt elérhetőek lesznek a felhasználók számára. Az iOS az Android után a második legnépszerűbb mobil operációs rendszer a világon. 2007-ben mutatták be iPhone OS néven, az első iPhone megjelenésével együtt, majd 2009-ben kapta meg az iOS nevet. Az Apple Mac OS X asztali operációs rendszerből származtatták le, amely UNIX alapokra épül. Megjelenése óta évente mutatnak be nagyobb frissítést hozzá. Az iOS négy rétegből áll, ezek a Core OS, Core Services, Media és a Cocoa Touch. A kezdetektől érintőkijelzős készülékekre lett tervezve, több ujjas gesztusok támogatásával[21][22].

2.7 Swift



25. ábra – Swift[29].

A Swift egy Apple által fejlesztett, 2014 június 2.-án bemutatott, magas szintű, általános célú programozási nyelv. Eleinte zárt forráskódú volt és az Apple saját operációs rendszereire készített alkalmazások fejlesztésének céljából hozták létre, majd 2015. december 3.-án nyílt forráskódúvá tették és azóta nyilvánosan GitHub-on fejlesztik. Cégen belül már 2010-ben dolgoztak a swift nyelven, szabad szoftverré tétele óta felgyorsult a fejlődése amibe nagy beleszólást engednek a fejlesztői közösségnek. A Swift támogatja a strukturált, a funkcionális, az objektumorientált, és a procedurális-imperatív programozási paradigmákat is. A nyelv számos régi és modern programozási nyelvből merít ötleteket, mint például az Objective-C, a Haskell, a Rust, a Python, a Ruby és a C#[29][30].

2.8 ARKit



26. ábra – ARKit[31].

Az ARKit az Apple iOS operációs rendszerre készített kiterjesztett valóság keretrendszere. Képes mind 2D mind 3D elemeket a kamera képére vetíteni. A készülék térben való nyomon követésére VIO (Visual Inertial Odometry) technológiát használ, amely mozgásérzékelő szenzorokból származó adat és a kamera képére egyaránt támaszkodik, hogy kalibrálás nélküli működést eredményezzen. Mind horizontális, mind vertikális síkokra képes virtuális objektumokat elhelyezni amelyeket folyamatosan az aktuális fényviszonyokhoz alakít[31].

2.9 CocoaPods



27. ábra – CocoaPods[45].

A CocoaPods macOS operációs rendszeren Xcode fejlesztői környezetben használható függőség menedzser Objective-C és Swift nyelvekhez. A külső könyvtárak kezelését biztosítja, így elkerülve azt az esetet amikor kézzel szükséges forrásfájlokat a projekt könyvtárába másolni. A harmadik féltől származó kód egyszerű használatára, a saját alkalmazásunkba való integrációjára összpontosít. A Terminalból (macOS parancssor) lehetséges a futtatása, illetve létezik grafikus kezelőfelülettel rendelkező macOS operációs rendszerre elérhető alkalmazás is hozzá[46]. A hivatalos cocoapods.org weboldalon jelenleg több mint 53 ezer könyvtár érhető el, amit körülbelül 3 millió alkalmazásban használtak fel eddig[45].

3 A MEGVALÓSÍTÁS MENETE

Az alkalmazás iOS operációs rendszeren történő futtatásra lett tervezve, így az Apple XCode fejlesztői környezet 10.0 verziójának használata vált szükségessé. Az XCode csak Apple MacOS operációs rendszerekre telepíthető. A fejlesztésre használt számítógép a MacOS High Sierra operációs rendszert futtatja. Az alkalmazásban használt technológiák megkövetelik az iOS operációs rendszer 11.0 vagy újabb verziójának használatát.

Az alkalmazás elkészítése különböző lépésekre bontható. A gondolatmenet, ami kiindulási pontként szolgál a következő:

- a mobil készülék kamerája által látottakon tárgyak felismerése konvolúciós neurális hálózat használatával,
- ugyanezen a képen könnyen követhető pontok keresése,
- majd az időben egymás után következő képeken optical flow számítás segítségével ezen pontok követése,
- végül egy regressziós modell bemenetét képezik a felismert tárgy, illetve a követett pontok koordinátái, ez a modell becslést ad a készülék elmozdulására.

3.1 Képek osztályokba sorolása

A látott tárgyak felismerésére a szabadon elérhető előre betanított neurális hálózatok pontatlannak bizonyultak, illetve ezek a modellek általában az ImageNet adatbázis 1000 kategóriáját különböztetik meg és a végső regressziót végző modell megfelelő betanításához minden felismert kategória esetében szükség van tanító adat gyűjtésére. Ezen problémák elkerülése céljából egy kevesebb kategóriát tartalmazó saját modell betanítása vált szükségessé ami adott környezetben (irodai környezet) való használatra lett felkészítve.

3.1.1 Adatok gyűjtése, előkészítése

```
[bendeguzs-mbp:Image_Downloader nemethbendeguz$ python image_downloader.py 200
Total images: 400

Downloading image 1 : https://static.techspot.com/articles-info/1179/images/2016-05-24-thumb-2.jpg
Downloading image 2 : https://media.4rgos.it/s/Argos/8209566_R_SET?$Web$&$Main350$&w=238&h=238&qlt=70
Downloading image 3 : https://cdn.arstechnica.net/wp-content/uploads/2018/08/fnmobilemouse-800x600.jpg
Downloading image 4 : https://brain-images-ssl.cdn.dixons.com/4/5/10118454/u_10118454.jpg
Downloading image 5 : https://assets.pcmag.com/media/images/352614-das-keyboard-4.jpg?width=1000&height=414
Downloading image 6 : https://images-na.ssl-images-amazon.com/images/I/41jXe9dmQmL._SX425_.jpg
Downloading image 7 : https://www.dsi-keyboards.com/wp-content/uploads/2015/03/KA-US-20874.jpg
```

28. ábra – image_downloader.py futtatása.

A képek osztályozását végző konvolúciós neurális hálózat betanításához szükséges adatok gyűjtését több eszköz felhasználásával végeztem, mint például a Google képkereső segítségével. Az adatok gyűjtésének folyamatát automatizáltam egy Python script megírásával. A futtatáskor lehetőség van paraméter megadására amivel beállítható, hogy osztályonként hány darab képet kívánunk gyűjteni. A Firefox böngésző script általi kezelését a Selenium WebDriver segítségével valósítottam meg. Ez egy teljesen nyílt forráskódú, szabadon bárki által használható keretrendszer amit főleg web alkalmazások tesztelésére használnak[47]. Több programozási nyelvet is támogat, ezen dolgozat keretein belül Python nyelven kerül használatra. A Selenium segítségével webes technológiákkal készített felületeken megoldható például a görgetés és a gombokra való kattintás, így tökéletesen alkalmazható a Google képkereső találati listájának automatizált mentésére. Ezen listán a képek kis felbontású formában szerepelnek, így egy további lépés is szükségessé vált. A script futásakor először a képekhez tartozó linkek kerülnek lementésre, majd ezek alapján az eredeti, jó felbontású kép kerül letöltésre.

```
mineral water
office chair
pen
keyboard
monitor
```

29. ábra – searchlist.txt tartalma.

A program a searchlist.txt szöveges fájl beolvasásával állapítja meg a keresendő osztályok neveit, amiknek egymás alatt, új sorban kell elhelyezkedniük. A szóközök kezelése megoldott. A gyűjtött képeket a dataset nevű könyvtárban, mappákba rendezve

kapjuk meg.

Az image_downloader.py script futtatásának feltételei:

- macOS operációs rendszer,
- Firefox böngésző,
- Python 2.0 futtatási környezet,
- Selenium Python csomag,
- Geckodriver[48].



keyboard.jpg



mineral_water.jpg



office_chair.jpg



pen.jpg



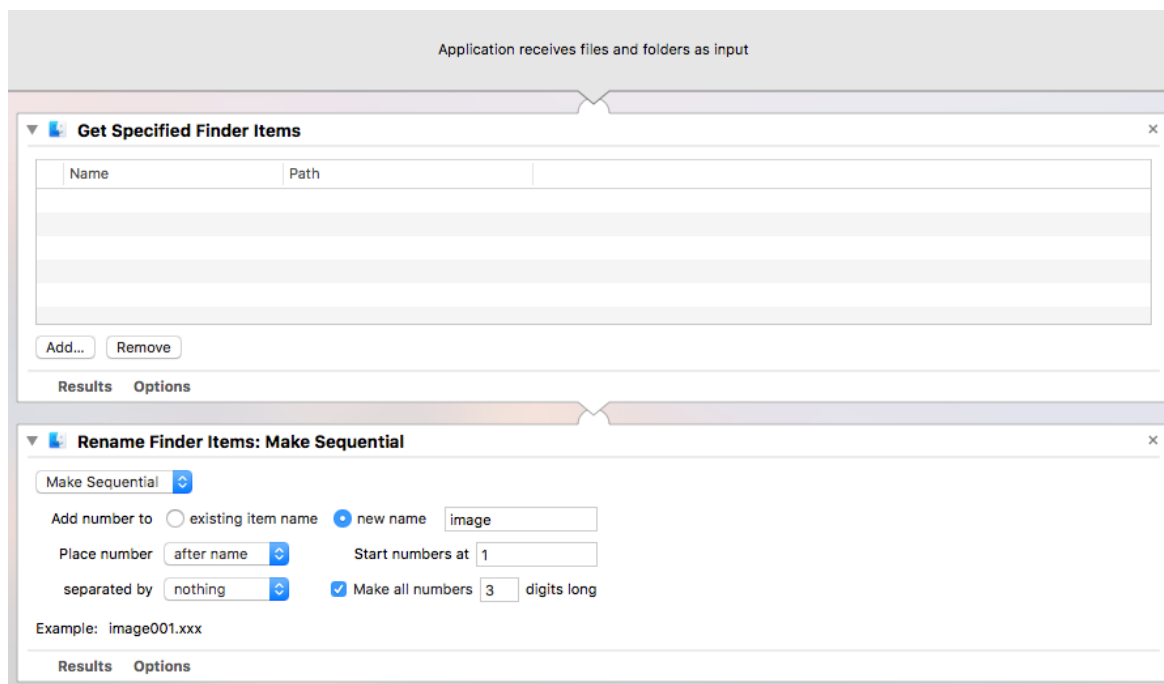
monitor.jpg

30. ábra – Felismert tárgyak.

A felismerni kívánt osztályok esetében a következőkre esett a választásom:

- billentyűzet,
- monitor,
- ásványvíz,
- irodai szék,
- toll.

A felsorolt kategóriák mindegyike gyakran előfordul irodai környezetben, ezzel biztosítva az alkalmazás használhatóságát. A képek automatizált gyűjtése azt eredményezte, hogy kerülhetnek az osztályokba nem illő elemek is az adatbázisba. Ennek megoldásaként kézzel válogattam ki a nem megfelelő kategóriában lévő képeket.



31. ábra – Automator folyamat.

A képek könyvtárának rendezett formára hozásában nagy segítség volt az Automator alkalmazás használata. Az Automator alkalmazás macOS operációs rendszeren elérhető alap alkalmazásokkal végezhető műveletek automatizálására, folyamatokba rendezésére alkalmas. A dolgozat keretein belül fájlok rendezésére, elnevezésére használtam.

3.1.2 Konvolúciós háló felállítása, betanítása

A képek osztályozásához szükséges konvolúciós neurális háló felállítására, betanítására és iOS alkalmazásba való integrálására használt eszközök a kezdeti elképzeléshez képest teljes mértékben lecserélődtek a dolgozat írása során. Az eredeti elképzelés szerint a neurális háló a TensorFlow szoftverkönyvtár használatával kerül felállításra illetve betanításra, majd a TensorFlow to CoreML Converter[49] segítségével kerül konvertálásra az iOS alkalmazásban használható mlmodel formátumra.

A végső megoldást a Turi Create gépi tanulásra létrehozott keretrendszer használatával valósítottam meg. Ezen keretrendszer használata melletti döntést több szempont

figyelembevételével hoztam meg. Használata egyszerűbb, ugyanazon feladat elkészítéséhez kevesebb kód megírása szükséges. A keretrendszer tartalmi elrendezése nem az egyes gépi tanulás algoritmusok szerint történik, hanem a felhasználás célja szerint, mint például ajánló algoritmusok, objektum detektálás, kép osztályozás, képek hasonlóságát megállapító algoritmusok vagy aktivitást osztályozó algoritmusok. Az adatok és az eredmények vizualizációja, illetve a különböző adattípusok, mint a szöveg, kép, hang, videó és szerzőkől származó adatok használata egyszerűbbnek bizonyult. A keretrendszeren belül van lehetőség a végső betanított modell közvetlenül mlmodel formátumban történő mentésére.

A Turi Create használatának követelményei:

- Python 2.7, 3.5, 3.6
- x86_64 architektúra
- macOS 10.12, Linux GNU C könyvtár 2.12-vel vagy Windows 10 operációs rendszer
- legalább 4 gigabyte elérhető memória

```
pip install virtualenv

# Create a Python virtual environment
cd ~
virtualenv venv

# Activate your virtual environment
source ~/venv/bin/activate
```

32. ábra – Virtuális környezet létrehozása.

A Turi Create használatához ajánlott virtuális környezet létrehozása. A dolgozat keretein belül ezt a Virtualenv segítségével valósítottam meg, mely alkalmas teljesen elszigetelt virtuális Python futtatási környezetek létrehozására. A telepítés a „pip install virtualenv”, a virtuális környezet létrehozása a „virtualenv venv”, majd az aktiválása a „source ~/venv/bin/activate” Terminal (macOS parancssor) paranccsal történik. A virtuális futtatási környezetben belül kód írásához a Jupyter Notebook-ot használtam fejlesztői környezetként. Ennek telepítése szintén a pip Python package manager segítségével lehetséges, majd futtatása a „jupyter notebook” paranccsal.

```

import turicreate as tc
import os

# Load images
data = tc.image_analysis.load_images('dataset',
                                     with_path=True)

# Create label column based on folder name
data['label'] = data['path'].apply(lambda path:
os.path.basename(os.path.dirname(path)))




# Save as .sframe
data.save('data.sframe')

# Explore
data.explore()

```

1. forráskód – SFrame formátumra alakítás.

A betanítás elvégzéséhez az első szükséges lépés az adatok Turi Create által használt SFrame táblázatos formára hozása volt. A korábban könyvtár struktúrába rendezett képek esetében a mappák neveit használtam fel osztályokként.

	path	image	label
0	/Users/nemethbendeguz/Library/...		keyboard
1	/Users/nemethbendeguz/Library/...		keyboard
2	/Users/nemethbendeguz/Library/...		keyboard

33. ábra – Adatok vizualizációja.

Az adatok vizualizálhatóak az .explore() függvény meghívásával. Ez nagy segítséget nyújthat annak ellenőrzésében, hogy az SFrame formátumra hozás sikeresen megtörtént e.

```
import turicreate as tc

# Load the data
data = tc.SFrame('data.sframe')

# Split to train and test data
train_data, test_data = data.random_split(0.8)

# Create model
model = tc.image_classifier.create(train_data,
                                   target='label')
```

2. forráskód – Tanító és teszt adat bontás.

A Turi Create importálása és az adatok beolvasása után az adatok két részre bontása következett. A képek 80% került felhasználásra a betanításhoz és a maradék 20% a teszteléshez. A modell elkészítéséhez elegendő akár csak annak meghatározása, hogy képek osztályozására van szükségünk. Ebben az esetben az eredmény javításához szükséges paraméterek megtalálása, illetve a konkrét modell kiválasztása automatikusan történik. Az adatok 5% kerül felhasználásra validálás céljából a tanítás során. A tanító- és validációs halmaztól teljesen elkülönülő teszhalmazon csak a tanítás végeztével értékeljük ki a megoldást, hogy szimulálni tudjuk a megoldás általános használhatóságát, és a teszhalmazon kapott teljesítmény ne informálhassa magának a megoldásnak a kialakítását.

Iteration	Passes	Step size	Elapsed Time	Training-accuracy	Validation-accuracy
1	4	0.000647	1.354263	0.821475	0.800000
2	6	1.000000	1.578969	0.847348	0.857143
3	7	1.000000	1.742404	0.978008	0.971429
4	8	1.000000	1.889910	0.985770	0.971429
5	9	1.000000	2.030474	0.997413	1.000000
6	10	1.000000	2.172180	0.998706	1.000000
10	14	1.000000	2.773519	1.000000	1.000000

34. ábra – Resnet tanítási folyamat.

Iteration	Passes	Step size	Elapsed Time	Training-accuracy	Validation-accuracy
1	6	0.000049	1.232502	0.255659	0.150000
2	8	1.000000	1.350768	0.559254	0.500000
3	9	1.000000	1.423698	0.740346	0.725000
4	10	1.000000	1.496254	0.717710	0.675000
5	11	1.000000	1.576413	0.800266	0.750000
6	12	1.000000	1.655594	0.805593	0.800000
11	18	1.000000	2.046216	0.962716	0.975000
25	33	1.000000	3.070595	1.000000	0.975000
50	68	1.000000	5.224137	1.000000	1.000000
51	69	1.000000	5.309906	1.000000	1.000000

35. ábra – Squeezenet tanítási folyamat.

Az automatikus tanítás során a Resnet50 modell került kiválasztásra. Ez a modell az 50 réteggel és több mint 25.6 millió paraméterével már kezdésként nagyon jó eredményeket ért el, viszont 94.2 megabyte-os méretével lassulást okozhat mobil készülékeken. Mivel nagyon kevés osztály megkülönböztetésére van szükségünk, így érdemesnek bizonyult megpróbálkozni kisebb méretű modellekkel.

```
# Create model, squeezenet, iteration=100
model = tc.image_classifier.create(train_data,
                                   target='label',
                                   model='squeezenet_v1.1',
                                   max_iterations=100)
```

3. forráskód – Konkrét modell meghatározása.

A mindössze 5 megabyte méretű, 69 rétegből álló Squeezenet v1.1 modell a nála jóval nagyobb Resnet50-el azonos eredményt ért el, mindezt alig több mint 1.2 millió paraméterével, így egyértelműen a Squeezenet használatát tartottam célszerűnek.

```
# Evaluate the model and show metrics
metrics = model.evaluate(test_data)
print(metrics['accuracy'])
```

4. forráskód – Modell tesztelése.

```
Resizing images...
Performing feature extraction on resized images...
Completed 210/210
1.0
```

36. ábra – Modell tesztelése.

A modell kiértékelése a tesztelésre szánt 210 kép mindegyikénél helyes eredményt hozott, így 100%-ot érve el.

```
# Save the model
model.save('squeezenet.model')

# Export to CoreML format
model.export_coreml('squeezenet.mlmodel')
```

5. forráskód – Modell mentése.

A modell mentésére lehetőség van a Turi Create által használt model, illetve az Apple platformokon, így az iOS operációs rendszeren is használt mlmodel formátumban is.

Name	Type	Description
▼ Inputs		
image	Image (Color 227 x 227)	Input image
▼ Outputs		
labelProbability	Dictionary (String → Double)	Prediction probabilities
label	String	Class label of top prediction

37. ábra – Modell be- és kimenetei.

A kapott végső modell bemenete egy 227x227 pixeles színes kép. Kimenatként több lehetőség is adott, vagy a legvalószínűbb osztály neve, vagy a felismert osztályok nevei és valószínűségük.

3.2 Optical flow számítása

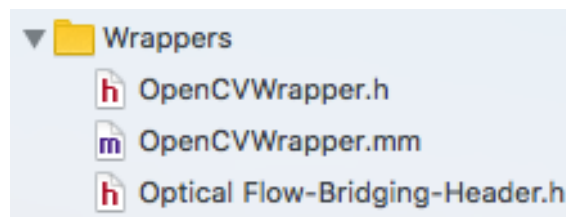
A cél eléréséhez felvázolt elképzelés alapján, következő lépésként a látottakon pontok elmozdulásának követésére volt szükségünk. Ennek megvalósítása az OpenCV szoftverkönyvtár használatával történt, így megoldandó feladatként merült fel, hogy hogyan

lehetséges OpenCV használata iOS alkalmazásban.

```
target 'Optical Flow' do  
  
    use_frameworks!  
  
    # Pods for Optical Flow  
    pod 'OpenCV'  
  
end
```

6. forráskód – Podfile.

iOS alkalmazásban a külső függőségek kezelésének legegyszerűbb módja a CocoaPods függőség menedzser használata. A menedzser ruby alapú konzolos alkalmazásként a „sudo gem install cocoapods” terminál paranccsal telepíthető. Telepítés után a szoftverprojektünk mappájába navigálva az „init pod” paranccsal inicializálható egy alap podfile a projektünkben. A podfile-ban felsorolásszerűen szükséges megadni a függőségeket, mint esetünkben pod OpenCV, majd a „pod install” parancs meghívásával lehetséges a függőségek projekthez adása.



38. ábra – Wrapper és Bridging fájlok.

Ahhoz, hogy OpenCV-t egy Swift nyelven írt iOS alkalmazásban használni lehessen több lépés is szükséges. Objective-C fájlok létrehozása szükséges a Wrapper osztályhoz, amin belül lehetséges az OpenCV használata. A Swift nyelvvel ellentétben, itt külön szükséges megírni az interface-t amit a „h” kiterjesztésű fájl tartalmaz és az implementációt a „m” kiterjesztésű fájlban. Azt, hogy mi nem kizárólag Objective-C hanem Objective-C++ nyelvet kívánunk használni a „m” kiterjesztés „mm” kiterjesztésre történő átírásával jelezhetjük az XCode-nak. A bridging header fájl teszi lehetővé, hogy a megírt osztályok elérhetőek legyenek a Swift nyelvű fájlokban is[50].

```

- (cv::Mat) greyscaleMatFromSampleBuffer:(CMSampleBufferRef)buf {

    CVImageBufferRef imgBuf = CMSampleBufferGetImageBuffer(buf);

    // lock the buffer
    CVPixelBufferLockBaseAddress(imgBuf, 0);

    // get the address to the image data
    void *imgBufAddr = CVPixelBufferGetBaseAddressOfPlane(imgBuf, 0);

    // get image properties
    int w = (int)CVPixelBufferGetWidth(imgBuf);
    int h = (int)CVPixelBufferGetHeight(imgBuf);

    // create the cv mat
    cv::Mat mat;

    // 8 bit unsigned chars for grayscale data
    mat.create(h, w, CV_8UC1);

    // the first plane contains the grayscale data
    memcpy(mat.data, imgBufAddr, w * h);
    // therefore we use <imgBufAddr> as source

    // unlock again
    CVPixelBufferUnlockBaseAddress(imgBuf, 0);

    return mat;
}

```

7. forráskód – Mat formátumra alakítás.

A következő lépésként a készülék kamerája által látottaknak az átalakítása szükséges az OpenCV keretein belül használható Mat formátumra. A Mat egy C++ nyelven írt osztály amely két fő részre bontható. A fejrész az egész mátrixra vonatkozó adatokat tartalmazza, mint a méretek, tárolási eljárások. A másik rész tartalmazza a tárolt pixelek adatait[51].

```

- (void) cornerDetector:(CMSampleBufferRef)buf {

    previousMat = [self greyscaleMatFromSampleBuffer:buf];

    cv::goodFeaturesToTrack(previousMat, previousFeatures, 6, 0.01, 20);
}

```

8. forráskód – Sarok detektálás.

A követésre alkalmas pontok keresése, élék, sarkok detektálásával történik. A sarokpontok speciális kulcspontok, azaz különböző objektumokat jól jellemző struktúrák.

Lehetnek például az objektumok sarkai, a fényesség szélsőértékei, vonalvégek, görbék legnagyobb görbületei. A sarkokat leggyakrabban leegyszerűsítik két él találkozására, hogy megkönnyítsék a felismerést. A lényeg, hogy hasonló, de nem azonos képeken megtalálják az egymásnak megfelelő sarkokat, tehát megismételhetőek legyenek[52]. Az OpenCV `goodFeaturesToTrack` függvénye teljes mértékben alkalmas erre a célra. Ez a függvény a Shi-Tomasi[53] sarok detektáló algoritmust használja, ami egy minimálisan módosított változata a Harris és Stephens[54] által 1988-ban készített algoritmusnak.

```
- (NSMutableArray *) opticalFlowTracker:(CMSampleBufferRef)buf {
    cv::Mat mat = [self greyscaleMatFromSampleBuffer:buf];
    std::vector<cv::Point2f> features;
    cv::calcOpticalFlowPyrLK(previousMat,
                             mat,
                             previousFeatures,
                             features,
                             status,
                             err);

    NSMutableArray *array = [[NSMutableArray alloc] init];
    NSMutableArray *previousPoints = [[NSMutableArray alloc] init];
    NSMutableArray *currentPoints = [[NSMutableArray alloc] init];

    for(unsigned int n = 0; n < previousFeatures.size(); n++) {
        int x = previousFeatures[n].x;
        int y = previousFeatures[n].y;

        [previousPoints addObject:[NSArray arrayWithObjects:@(x), @(y), nil]];
    }

    for(unsigned int n = 0; n < features.size(); n++) {
        int x = features[n].x;
        int y = features[n].y;

        [currentPoints addObject:[NSArray arrayWithObjects:@(x), @(y), nil]];
    }

    [array addObject:previousPoints];
    [array addObject:currentPoints];

    cv::goodFeaturesToTrack(mat, previousFeatures, 6, 0.01, 20);
    previousMat = mat;

    return array;
}
```

9. forráskód – Optical flow számítás.

Az alkalmazás indításakor a `cornerDetector` függvény kerül meghívásra, mivel ekkor még nem állnak rendelkezésre korábbi pontok, így nem tudunk optical flow-t számolni. A második iterációtól már az `opticalFlowTracker` fut le. Minden iterációnál a 6 darab legkönnyebben követhető pont koordinátái kerülnek eltárolásra. Az `OpenCV` `calcOpticalFlowPyrLK` függvénye segítségével történik az előző lefutáskor eltárolt pontok új helyzetének megtalálása. Ez a függvény a Lucas-Kanade[37] eljárást használja az optical flow számítás megvalósítására.

3.3 Készülék elmozdulásának becslése

3.3.1 Tanító adatok összeállítása

A készülék elmozdulását becslő végső regressziós modell betanításához szükséges adatok gyűjtése több szakaszra bontható. Az adatok egy részét csv formátumban exportáltam a mobil alkalmazásból. Ez tartalmazta a kimentés időpontját ezred másodperc pontossággal, illetve a követni kívánt hat pont koordinátáit a jelenlegi és az előző mentéskor.

```

@IBAction func exportCSV(_ sender: UIButton) {

    if (captureSession.isRunning) {
        captureSession.stopRunning()

        let path =
            NSURL(fileURLWithPath:
                NSTemporaryDirectory()).appendingPathComponent(fileName)

        do {
            //Puts file in the path
            try csvText.write(to: path!,
                               atomically: true,
                               encoding: String.Encoding.utf8)

            //Displays export options
            let vc = UIActivityViewController(activityItems: [path!],
                                               applicationActivities: [])

            present(vc, animated: true, completion: nil)
        } catch {
            print("Failed to create file")
            print("\(error)")
        }
    } else {
        first = true

        csvText = "time,p1t0x,p1t0y,p2t0x,p2t0y,p3t0x,p3t0y,p4t0x,p4t0y,
                    p5t0x,p5t0y,p6t0x,p6t0y,p1t1x,p1t1y,p2t1x,p2t1y,p3t1x,
                    p3t1y,p4t1x,p4t1y,p5t1x,p5t1y,p6t1x,p6t1y\n"

        captureSession.startRunning()
    }
}

```

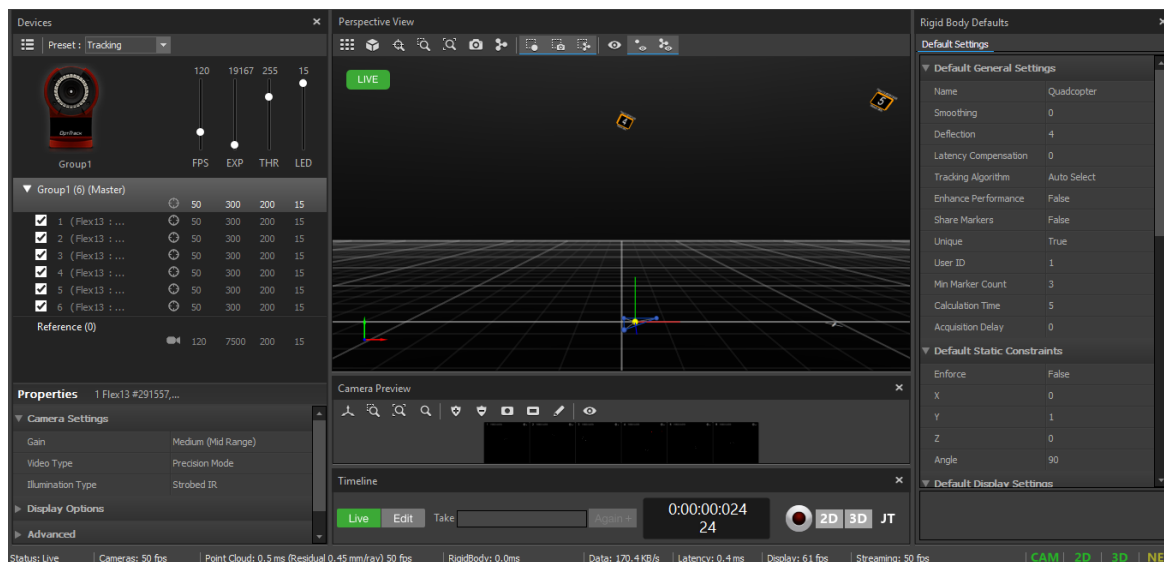
10. forráskód – data.csv exportálása.

Az adatgyűjtés során folyamatosan feljegyeztem melyik vizsgált objektumot látja a kamera és ezt az információt kézzel fűztem hozzá az adatokhoz, ezzel kiküszöbölve az előző modell általi esetleges rossz osztályba sorolásból eredő hibákat.



39. ábra – OptiTrack infrakamerák[55].

A készülék térbeli elmozdulásának meghatározására szolgáló adatokat az OptiTrack infrakamerás tracker rendszer segítségével gyűjtöttem. Itt az időpontok mentése a mobil alkalmazással ellentétben a következőképpen történt: a felvétel indulása a nulla időpont, majd minden mentésnél az induláshoz képest eltelt idő kerül kiírásra. A mozgás meghatározása háromdimenziós koordináták feljegyzésével történt.



40. ábra – OptiTrack Motive alkalmazás[56].

Az infrakamerákhoz tartozik egy Microsoft Windows operációs rendszerre telepíthető alkalmazás ami lehetővé tette a leírt adatok gyűjtését és a mobil alkalmazáshoz hasonlóan csv formátumban történő exportálását. Az infrakamerák elhelyezésénél és a környezet felállításánál minden a kamerákat zavarni képes tényezőt meg kellett szüntetni, ilyen lehet

például a természetes fény, a tükröződő felületek. A kamerákhoz tartozó alkalmazás funkciói közül kiemelten hasznosnak bizonyult például a felvételek kezdetének időzítése, illetve a fennmaradó fényforrások maszkolása, ennek használata mindenképp szükséges egymás látóterébe eső kamerák esetén.

time	p1t0x	p1t0y	p2t0x	p2t0y	p3t0x	p3t0y	p4t0x	p4t0y	p5t0x	p5t0y	p6t0x	p6t0y	p1t1x	p1t1y	p2t1x	p2t1y	p3t1x	p3t1y	p4t1x	p4t1y	p5t1x	p5t1y	p6t1x	p6t1y
12:33:13.942	608	388	570	386	594	431	580	456	595	651	519	262	606	387	568	385	592	429	578	454	592	647	518	262
12:33:14.034	606	387	576	386	592	429	518	262	578	455	599	457	604	386	574	385	590	427	517	262	576	453	597	455
12:33:14.125	604	387	567	385	591	429	576	315	605	317	576	454	603	386	567	384	590	427	576	315	604	316	575	452
12:33:14.218	604	386	575	385	591	428	577	452	577	299	597	454	601	385	573	384	588	426	575	450	575	299	594	452
12:33:14.309	602	385	573	384	575	450	588	426	574	315	603	306	600	384	571	383	573	448	586	424	572	314	601	306
12:33:14.402	600	384	572	383	573	448	587	425	593	450	517	265	597	382	570	381	571	445	584	423	590	447	515	265
12:33:14.493	598	383	570	382	585	423	571	446	599	306	516	266	596	382	568	381	583	422	569	444	597	306	515	267
12:33:14.584	596	383	569	381	570	445	583	422	515	267	567	622	593	382	566	380	567	443	580	420	513	269	564	616
12:33:14.673	593	383	567	381	581	421	568	302	514	269	594	318	589	381	564	379	577	418	564	301	512	269	590	317
12:33:14.765	589	381	556	379	577	418	591	308	583	442	571	243	585	380	553	378	574	416	587	308	580	439	568	245
12:33:14.856	586	380	561	378	562	438	575	416	587	318	561	317	584	379	560	377	561	436	573	414	585	318	560	317
12:33:14.946	586	379	560	378	574	415	579	438	558	501	562	303	582	377	557	376	571	412	575	434	555	496	559	303
12:33:15.036	558	376	582	378	576	435	571	412	565	712	559	304	555	375	579	377	573	432	568	410	562	702	556	304
12:33:15.128	579	377	556	375	568	411	556	318	557	431	443	355	576	375	553	374	565	409	553	318	554	428	443	354
12:33:15.218	576	376	547	374	566	409	532	709	555	429	495	457	574	375	545	373	564	407	531	703	553	427	494	455
12:33:15.309	575	375	552	374	564	408	553	427	443	354	506	286	573	374	550	373	562	406	551	425	443	353	505	287
12:33:15.400	573	374	544	373	563	406	506	278	439	463	552	425	571	372	542	371	561	404	505	278	439	460	550	422
12:33:15.490	571	373	549	372	522	214	561	405	620	32	550	423	568	372	547	371	520	215	558	403	617	35	548	421
12:33:15.580	541	371	569	372	520	215	559	403	498	340	545	478	539	370	567	371	519	217	557	402	497	340	543	475
12:33:15.671	567	372	546	371	557	403	607	91	519	217	545	560	565	371	545	370	555	401	605	95	518	218	544	555
12:33:15.763	566	371	545	370	519	219	556	401	535	575	543	469	563	370	543	369	517	220	553	399	533	570	541	466
12:33:15.854	564	370	543	369	557	50	554	400	517	221	602	99	562	369	541	368	555	53	552	399	515	222	600	101
12:33:15.944	562	370	542	369	403	472	608	54	438	349	556	49	560	369	541	368	404	469	605	59	438	348	554	54

41. ábra – iOS alkalmazásból származó adatok.

Ezután az adatok összesítése, összehangolása következett. A mozgás kezdetének összehangolását a következőképpen oldottam meg: minden felvétel indítása után körülbelül három másodpercen át a készülék egy helyben állt, így mindkét forrásból származó adatok esetében jól észrevehetővé vált a mozgás kezdete. Ezeket az időpontokat minden felvételben kikerestem, és levágtam az előttük lévő szakaszokat. Az OptiTrack kalibrációja során a használt koordináta-rendszert úgy helyeztem el, hogy a mobil készülék mozgatása pontosan egy dimenzió mentén történjen, így a másik két dimenzió adatai elhanyagolhatóak. A mozgás végét, a kezdetéhez hasonlóan körülbelül három másodperces egy helyben állás jelezte. A két kiválasztott időintervallum hosszát összehasonlítva megerősítést kaphatunk az adatok helyességére.

Time	Z
0.000000	0.000000
0.008333	0.000325
0.016666	0.000740
0.025000	0.001131
0.033333	0.001557
0.041666	0.001997
0.050000	0.002414
0.058333	0.002833
0.066666	0.003261
0.075000	0.003687

42. ábra – OptiTrack rendszerből származó adatok.

Az iOS alkalmazásból származó adatok esetén a mentések időpontját átalakítottam a következőképpen: az első elem legyen a nulla időpont és az ezt követő elemek esetében a kimentésükig eltelt idő jelenjen meg. Az OptiTrack-ből származó adatok esetében az időpontok alaphoz az előzőek szerint kerültek kimentésre, mindössze annyi átalakítást kellett végezni, hogy a mozgás kezdeténél legyen a nulla időpont. A mobil készülék elmozdulásának mértékét szintén átalakítottam, hogy nulláról induljon, illetve nem a kiindulástól megtett távolságra van szükségünk, hanem az előző méréstől megtettre, így kivontam minden távolság értékéből az előző mérésnél lévő értéket. Az OptiTrack-ből származó távolság adatok mértékegysége a méter, ezt átváltottam milliméterre, hogy könnyebben átlátható értékeket kapjunk.

A következő lépés a készülék elmozdulásának mértékét hozzáilleszteni a mobil alkalmazásból származó adatokhoz. A két készülék adatmentésének gyakorisága körülbelül egy nagyságrenddel eltér egymástól. A Numbers táblázatkezelő alkalmazásban található vlookup függvénnyel megkerestem minden a mobil alkalmazásból származó időponthoz a hozzá legközelebb eső, nála kisebb vagy vele egyenlő értéket az OptiTrack-ből származó adatok közül. Annak következtében, hogy a két készülékből származó időpontok nem egyeznek tökéletesen, a végső elmozdulást lineáris interpolációval számítottam ki, az előzőekben megkeresett időponthoz tartozó érték és a sorban következő érték között. Mivel az OptiTrack-ből származó két egymást követő mentés adatai közötti különbség ezred másodperc és tized milliméter nagyságrendű, így a lineáris leképezésből adódó hiba elhanyagolható.

3.3.2 Regressziós modell elkészítése

object	p1t0x	p1t0y	p2t0x	p2t0y	p3t0x	p3t0y	p4t0x	p4t0y	p5t0x	p5t0y	p6t0x	p6t0y	p1t1x	p1t1y
chair	95	443	184	161	585	316	631	468	373	213	209	284	93	443
chair	93	444	182	161	631	468	585	316	372	213	208	284	90	444
chair	91	444	189	157	372	212	586	315	632	469	647	216	88	444
chair	88	445	179	159	586	315	633	470	205	283	647	216	85	446
chair	86	446	186	156	634	471	587	316	371	211	677	189	83	446
chair	83	447	184	155	370	211	588	316	635	472	649	215	80	447
chair	80	448	176	157	589	316	370	211	635	473	680	188	76	449
chair	182	154	78	450	637	474	590	316	201	284	654	134	180	153
chair	173	156	76	451	590	316	369	210	638	475	200	284	172	155
chair	73	452	172	155	592	317	370	210	640	476	200	284	69	453

p2t1x	p2t1y	p3t1x	p3t1y	p4t1x	p4t1y	p5t1x	p5t1y	p6t1x	p6t1y	target
182	160	585	315	631	468	372	212	207	283	3.67
180	159	632	468	586	315	371	212	207	283	5.04
186	156	370	211	585	315	632	469	647	215	5.81
178	158	587	315	634	471	204	283	648	215	6.14
184	155	634	471	587	316	370	210	677	188	5.99
183	154	370	210	588	316	635	472	650	214	6.6
174	156	590	316	369	210	636	474	681	187	6.72
75	451	638	475	590	316	200	284	654	133	7.54
73	452	591	316	369	209	640	476	199	283	7.7
169	154	592	317	368	209	640	477	197	283	7.62

43. ábra – Tanításra használt adatok végleges formája.

```
import turicreate as tc

# Load the data
data = tc.SFrame.read_csv('data.csv',
                           header=True)
```

11. forráskód – Adatok beolvasása.

A teljes tanító adathalmaz 5459 esetből áll. Minden eset 26 értéket tartalmaz, a felismert objektum nevét, a hat darab legkönnyebben követhető pont x és y koordinátáját egymást követő két időpillanatban, és a készülék elmozdulásának mértékét. Az elmozdulás itt már milliméterben szerepel, pozitív szám jelzi az előre, az adott tárgyhoz közeledve történő mozgást, negatív a hátra fele történő, távolodó mozgást.

```
# Make a train-test split
train_data, test_data = data.random_split(0.8)
```

12. forráskód – Adatok bontása.

A korábban az objektumok megkülönböztetésére betanított neurális hálózhoz hasonlóan, az adatokat két részre bontottam, 80% került felhasználásra a tanításhoz, és a maradék 20% teszteléshez. Az előzőleg bemutatott tanításhoz hasonlóan az adatok 5%-a került felhasználásra validálás céljából. A teszhalmazon történő kiértékelést a tanítási folyamat lezárulása után végeztem.

```
# Automatically picks the right model
# based on your data.
model = tc.regression.create(train_data,
                             target='target')
```

13. forráskód – Regresszió automatikus modell választással.

Amennyiben nem jelölünk ki konkrét modellt a tanításhoz, a Turi Create automatikusan választ a tanító adathalmaz alapján. A mi esetünkben a Boosted Regression Trees modellt használta. Az első tanítási kísérlet nem vezetett használható eredményre, mindössze néhány százalékos javulás volt felfedezhető a teljesen random generált eredményekhez képest.

A Turi Create-ben elérhető regressziós modellek a következők[44]:

- Lineáris Regresszió (Linear Regression)
- Döntési Fa (Decision Tree)
- Véletlen Erdő (Random Forest)
- Fejlődő/Sokasított Regressziós Fák (Boosted Regression Trees)

```
model = tc.decision_tree_regression.create(train_data,
                                           target='target',
                                           max_depth=24)
```

14. forráskód – Döntési Fa mélység meghatározással.

A Turi Create-ben elérhető egyéb modellek sem vezettek jobb eredményre. A mélység növelésével túltanulás (overfitting) volt megfigyelhető. A tanítóhalmazon feltűnően jól

teljesítés után a validálásból és a tesztelésből kiderült, hogy csak romlott a modell teljesítménye.

Iteration	Elapsed Time	Training-max_error	Validation-max_error	Training-rmse	Validation-rmse
1	0.090990	25.051100	25.196983	6.278406	7.256741
2	0.169707	21.196804	22.394909	5.092135	6.902473
3	0.245684	18.578045	19.981215	4.163632	6.710398
4	0.332687	15.922437	19.716761	3.384690	6.538250
5	0.426131	13.592540	21.859707	2.669532	6.391981
6	0.505570	11.908092	22.430510	2.232780	6.372654
11	0.878876	6.251930	22.653488	1.094305	6.299362
25	1.847756	1.759507	22.196232	0.223939	6.157979
50	3.796201	0.060986	22.071854	0.004132	6.148653
51	3.865525	0.051838	22.072067	0.003526	6.148671
75	4.128311	0.027819	22.071060	0.002095	6.148696
100	4.210749	0.027819	22.071060	0.002095	6.148696

44. ábra – Túltanulás (Overfitting).

A Turi Create-ben megvalósított Gradient Boosted Regression Trees modell több döntési fából tevődik össze (model ensembling), ehhez a Gradient Boosting eljárást használja[44]. A tanítás során a költséget az MSE (Mean Squared Error) alapján számolja ami a regressziós feladatoknál legjellemzőbben használt költség függvény[57].

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \tilde{y}_i)^2$$

5. képlet – MSE (Mean Squared Error).

```
model = tc.boosted_trees_regression.create(train_data,
                                          target='target',
                                          max_iterations=200,
                                          max_depth=4,
                                          step_size=0.2)
```

15. forráskód – Boosted Regression Trees paraméterekkel.

A Fejlődő/Sokasított Regressziós Fák (Boosted Regression Trees) esetében az iterációk számát (és ezzel egyúttal a fák számát) növelve nagyon hamar a túltanulás jelei jelentkeznek. Ezt a fák mélységének, illetve a lépések méretének (step size) a csökkentésével

tudtam ellensúlyozni.

```
# Evaluate the model and save the results into a dictionary
results = model.evaluate(test_data)
print(results)
```

16. forráskód – Regressziós modell tesztelése.

```
{'max_error': 19.342164039611816, 'rmse': 4.870457906498114}
```

45. ábra – Modell tesztelése.

A kézzel paraméterezett, kisebb komplexitású modell tanítása során kapott 3.85 és 6.05 milliméteres négyzetes hiba a tanító illetve validációs halmazon még mindig mutatja túltanulás jelét, viszont a tesztelés során ezzel a modellel sikerült elérnem a legjobb 4.87 milliméteres eredményt, ami több mint 60%-al jobb a szélsőértékeken belül generált random értékekhez képest. A modell az átlagos 6.72 milliméteres elmozdulásnál is jobban teljesít, így az kijelenthető, hogy a felállított tanító adathalmazban felfedezhetőek összefüggések, amelyek alapján lehetséges következtetést levonni a készülék elmozdulásának irányára, mértékére. A mobil alkalmazásba illesztéskor kapott értékekből egyértelműen látszik az előre és hátra irányuló mozgás, illetve a mozgás mértékére is lehet következtetni belőlük.

```
# Save the model
model.save('regression.model')

model.export_coreml('regression.mlmodel')
```

17. forráskód – Regressziós modell mentése.

3.3.3 Megvalósítás ARKit használatával

Az ARKit használatával megvalósított kiterjesztett valóság alkalmazások fő építőeleme az ARSCNView, ami biztosítja a grafikus felhasználói felületet a készülék kamerájának képével, illetve a segítségével helyezhetők virtuális elemek a látottakra. Minden ARSCNView rendelkezik egy ARSession objektummal ami a készülék mozgásának követéséért felel. Az ARSession futtatásához szükség van egy ARConfiguration objektumra ami meghatározza milyen célra szeretnénk használni az alkalmazást.

```

import UIKit
import ARKit

class ViewController: UIViewController, ARSessionDelegate {

    @IBOutlet weak var arscnView: ARSCNView!
    @IBOutlet weak var label: UILabel!

    var previousPosition: simd_float4?
    var resultCount = 0

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        let configuration = ARWorldTrackingConfiguration()
        arscnView.session.run(configuration)

        arscnView.session.delegate = self
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        arscnView.session.pause()
    }

    func session(_ session: ARSession, didUpdate frame: ARFrame) {
        resultCount += 1

        if resultCount >= 30 {
            DispatchQueue.main.async {
                let currentPosition = frame.camera.transform.columns.3
                let distance = distance(previousPosition, currentPosition)

                if let previousPosition = self.previousPosition {
                    self.label.text = "\(Int(distance * 1000)) mm"
                }

                self.resultCount = 0
                self.previousPosition = currentPosition
            }
        }
    }
}

```

18. forráskód – ARKit használata.

ARConfiguration objektumok:

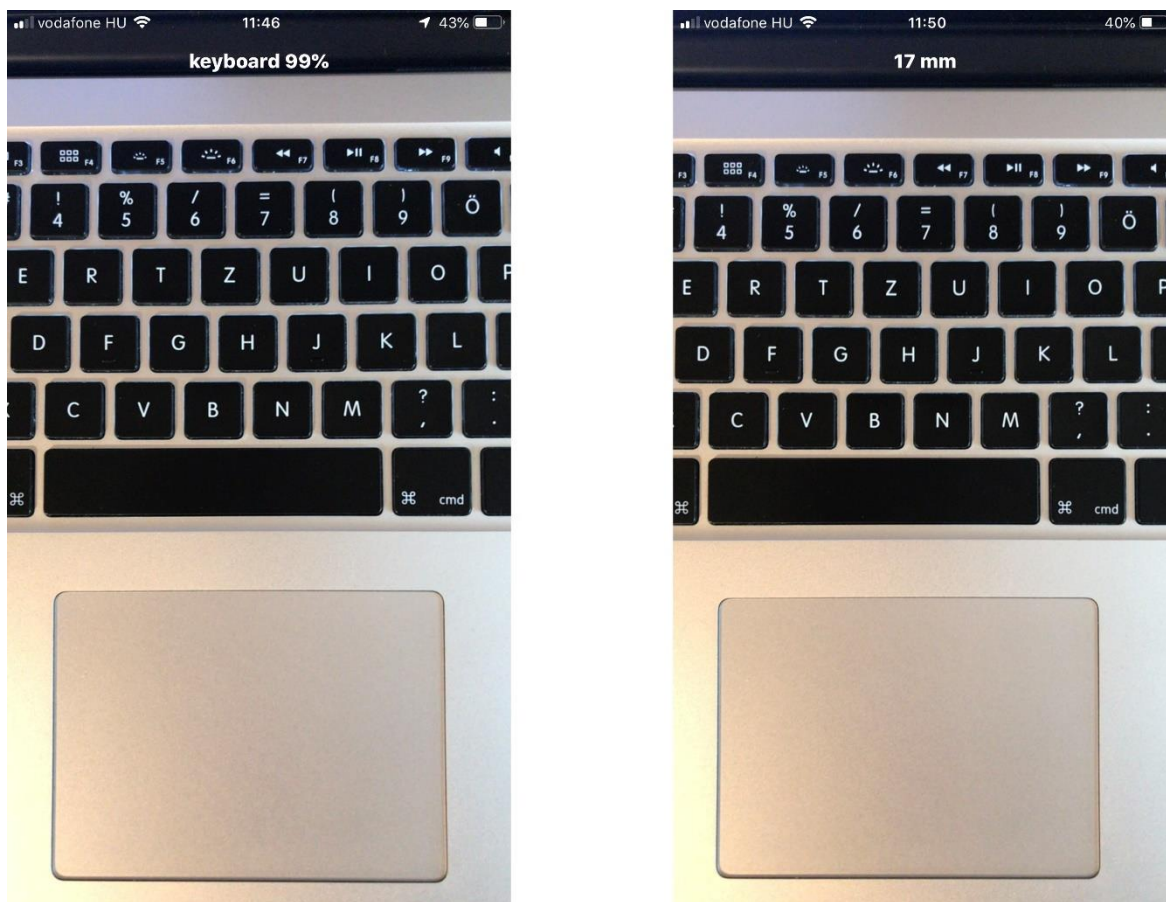
- ARWorldTrackingConfiguration: a hátlapi kamerát használja a készülék pozíciójának és orientációjának követésére, illetve lehetőséget nyújt sík felületek és objektumok felismerésére is.
- AROrientationTrackingConfiguration: a hátlapi kamerát használja a készülék

orientációjának követésére.

- `ARImageTrackingConfiguration`: előre meghatározott képeket ismer fel, így például további virtuális információ helyezhető rájuk.
- `ARFaceTrackingConfiguration`: az előlapi kamerát használja emberi arcok követésére.
- `ARObjectScanningConfiguration`: lehetővé teszi valós tárgyak virtuális objektumokká történő letapogatását.

Az esetemben célként kitűzött, készülék elmozdulásának meghatározásához egy olyan metódust fejtettem ki, amely minden esetben meghívódik amint egy új pozíció rendelkezésre áll. Annak érdekében, hogy az előzőleg ARKit nélkül megvalósított alkalmazáshoz hasonló működést érjek el, minden harmincadik képkockánál kerül kiszámolásra a készülék előző pozíciójától megtett távolság. Ezen alkalmazás megvalósításához elengedhetetlen volt az `ARSCNView` használata, amelynek szüksége van a készülék fizikai kamerájára, viszont képek nem nyerhetők ki belőle, így a dolgozat előző fejezeteiben bemutatott alkalmazás nem futtatható `ARSCNView` használata mellett. A bemutatott két módszer nem ad lehetőséget egy adott készüléken egyszerre történő, azonos környezetbeli adatgyűjtésre.

4 EREDMÉNYEK KIÉRTÉKELÉSE



46. ábra – iOS alkalmazás képernyőképek.

A dolgozat készítése számos részre bontható, melyek mindegyike esetében beszélhetünk elért eredményekről. Ide sorolható például az objektumok felismerését végző modell betanításához szükséges képek gyűjtésének az automatizálása egy Python script megírásával. A képek könyvtárának rendezett formára hozása Automator folyamatok elkészítésével. Mindenképp sikeres eredménynek tekinthető az objektumok felismerésére betanított modell 210 képpel történt tesztelésekor tapasztalt 100%-os teljesítmény. Az iOS operációs rendszerű mobil készüléken végzett sikeres optical flow számítás. Mivel a végső regressziós modell betanításához használt adathalmaz felállítása során két nagyban különböző eszközről származtak az adatok, így ezen folyamatnak több lépése is kiemelhető. Ilyen a felvételek kezdő időpontjainak összehangolása, illetve a két készülékből különböző gyakorisággal érkező adatok egyesítése. Ezen modell 4.87 milliméteres tévedése több mint 60%-al jobb a szélsőértékeken belül generált random értékekhez képest, illetve az átlagos

6.72 milliméteres elmozdulásnál is jobban teljesít, így az mindenképp kiderül belőle, hogy a felállított tanító adathalmaz tartalmaz összefüggéseket, amelyek alapján lehetséges következtetést levonni a készülék elmozdulásának irányára, mértékére. A mobil alkalmazásba illesztéskor kapott értékekből egyértelműen látszik az előre és hátra irányuló mozgás, illetve a mozgás mértékére is lehet következtetni belőlük.

IRODALOMJEGYZÉK

- [1] Kiterjesztett valóság alkalmazások
http://alpha.tmit.bme.hu/meresek/ov_05a.pdf 2018.04.23.
- [2] D. W. F. van Krevelen, R. Poelman: A Survey of Augmented Reality Technologies, Applications and Limitations
Systems Engineering Section, Delft University of Technology Jaffalaan 5, 2628BX Delft, The Netherlands, 2010.
- [3] Paul Milgram, Haruo Takemura, Akira Utsumi, Fumio Kishino: Augmented Reality: A class of displays on the reality-virtuality continuum
ATR Communication Systems Research Laboratories, 2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan, 1994.
- [4] IKEA Place
<https://itunes.apple.com/app/ikea-place/id1279244498?mt=8> 2018.04.23.
- [5] Pokémon GO
<https://itunes.apple.com/us/app/pokémon-go/id1094591345?platform=iphone&preserveScrollPosition=true#platform/iphone>
2018.04.23.
- [6] Complete Anatomy
<https://itunes.apple.com/app/complete-anatomy-2018-courses/id1054948424?mt=8>
2018.04.23.
- [7] Altrichter M., Horváth G., Pataki B., Strausz G., Takács G., Valyon J.: Neurális hálózatok
Panem Könyvkiadó Kft., Budapest, 2006.
- [8] Clustering
<http://www.saedsayad.com/clustering.htm> 2018.04.23.
- [9] A Beginner's Guide To Understanding Convolutional Neural Networks
<https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To->

- [Understanding-Convolutional-Neural-Networks/](#) 2018.04.23.
- [10] Yann Lecun, Léon Bottou, Joshua Bengio, Patrick Haffner: Gradient-based Learning Applied to Document Recognition
Proc. of the IEEE, 1998.
- [11] CS231n Convolutional Neural Networks for Visual Recognition
[http://cs231n.github.io/convolutional-networks/](#) 2018.04.23.
- [12] A Practical Introduction to Deep Learning with Caffe and Python
[http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/](#)
2018.04.23.
- [13] Dropout layer
[https://www.quora.com/In-TensorFlow-what-is-a-dense-and-a-dropout-layer](#)
2018.04.23.
- [14] Deep Convolutional Networks
[http://www.ritchieng.com/machine-learning/deep-learning/convvs/](#) 2018.04.23.
- [15] Min Lin, Qiang Chen, Shuicheng Yan: Network In Network
Graduate School for Integrative Sciences and Engineering, Department of Electronic & Computer Engineering, National University of Singapore, Singapore, arxiv:1312.4400, 2014.
- [16] Fully-connected neural network
[https://math.stackexchange.com/questions/2048722/a-name-for-layered-directed-graph-as-in-a-fully-connected-neural-network](#) 2018.04.23.
- [17] Cross entropy
[http://ml4dummies.blogspot.com/2017/08/cross-entropy-loss-and-maximum.html](#)
2019.04.06.
- [18] Machine Learning
[https://www.coursera.org/learn/machine-learning](#) 2018.04.23.
- [19] Konvolúciós neurális hálózatok (CNN)

<http://home.mit.bme.hu/~engedy/NN/NN-CNN.pdf> 2018.04.23.

- [20] Jason Yosinski, Jeff Clune, Yoshua Bengio, Hod Lipson: How transferable are features in deep neural networks?

Dept. Computer Science, Cornell University, Dept. Computer Science, University of Wyoming, Dept. Computer Science & Operations Research, University of Montreal, Dept. Mechanical & Aerospace Engineering, Cornell University, arxiv:1411.1792v1, 2014.

- [21] iOS 11

<https://www.apple.com/lae/ios/ios-11/> 2018.04.23.

- [22] iOS

<https://en.wikipedia.org/wiki/IOS> 2018.04.23.

- [23] Python

<https://www.python.org> 2018.04.23.

- [24] Python about

<https://www.python.org/about/> 2018.04.23.

- [25] Python (programozási nyelv)

[https://hu.wikipedia.org/wiki/Python_\(programozási nyelv\)](https://hu.wikipedia.org/wiki/Python_(programoz%C3%A1si_nyelv)) 2018.04.23.

- [26] TensorFlow

<https://www.tensorflow.org> 2018.04.23.

- [27] Mély tanulási rendszerek felhasználása robotikai alkalmazásoknál

<https://gael.medialab.bme.hu/mdlb/intraweb/temalista/tema/TMIT2018-076>
2018.04.23.

- [28] TensorFlow

<https://en.wikipedia.org/wiki/TensorFlow> 2018.04.23.

- [29] Swift

<https://developer.apple.com/swift/> 2018.04.23.

- [30] Swift (programming language)

- [https://en.wikipedia.org/wiki/Swift_\(programming_language\)](https://en.wikipedia.org/wiki/Swift_(programming_language)) 2018.04.23.
- [31] ARKit
<https://developer.apple.com/arkit/> 2018.04.23.
- [32] Optical flow ábra
https://docs.opencv.org/3.4/d7/d8b/tutorial_py_lucas_kanade.html 2018.10.29.
- [33] Horváth Péter: Képek szegmentálása szín és mozgás alapján
Szegedi Tudományegyetem, 2004.
- [34] Optical flow
https://en.wikipedia.org/wiki/Optical_flow 2018.10.29.
- [35] K. Prazdny: Motion and Structure from Optical Flow
IJCAI, 1979.
- [36] B. K. P. Horn, B. G. Schunck: Determining Optical Flow
Artificial Intelligence Laboratory, Massachusetts Institute of Technology,
Cambridge, MA 02139, U.S.A, 1981.
- [37] B. D. Lucas, T. Kanade: An Iterative Image Registration Technique with an
Application to Stereo Vision
Computer Science Department, Carnegie-Mellon University, Pittsburgh,
Pennsylvania, 1981.
- [38] Jupyter ábra
<http://jupyter.org> 2018.10.29.
- [39] Project Jupyter
https://en.wikipedia.org/wiki/Project_Jupyter 2018.10.29.
- [40] Digitális képfeldolgozás
<http://www.inf.u-szeged.hu/~tanacs/oktatas/kepfeldtg2018/> 2018.10.29.
- [41] OpenCV
<https://en.wikipedia.org/wiki/OpenCV> 2018.10.29.

- [42] Turi Create ábra
<https://turi.com> 2018.10.29.
- [43] GraphLab
<https://en.wikipedia.org/wiki/GraphLab> 2018.10.29.
- [44] Turi Create
<https://github.com/apple/turicreate> 2018.10.29.
- [45] CocoaPods ábra
<https://cocoapods.org> 2018.10.29.
- [46] CocoaPods
<https://en.wikipedia.org/wiki/CocoaPods> 2018.10.29.
- [47] Saving images from google search using selenium and python
<https://simply-python.com/2015/05/18/saving-images-from-google-search-using-selenium-and-python/> 2018.10.29.
- [48] Google image downloader
https://github.com/atif93/google_image_downloader 2018.10.29.
- [49] TensorFlow to CoreML Converter
<https://github.com/tf-coreml/tf-coreml> 2018.10.29.
- [50] OpenCV 3.2.0 Setup on iOS
<https://blog.kickview.com/opencv-setup-on-ios/> 2018.10.29.
- [51] Mat - The Basic Image Container
https://docs.opencv.org/3.4/d6/d6d/tutorial_mat_the_basic_image_container.html
2018.10.29.
- [52] Sarokpont detektálás
http://mialmanach.mit.bme.hu/erdekesssegek/sarokpont_detektalas 2018.10.29.
- [53] Jianbo Shi, Carlo Tomas: Good Features to Track
IEEE Conference on Computer Vision and Pattern Recognition (CVPR94) Seattle,

1994.

- [54] Chris Harris, Mike Stephens: A combined corner and edge detector

Plessey Research Roke Manor, United Kingdom, 1988.

- [55] OptiTrack Flex 13

<https://optitrack.com/products/flex-13/> 2019.04.06.

- [56] OptiTrack Motive

<https://optitrack.com/products/motive/> 2019.04.06.

- [57] Mean squared error

<https://stackoverflow.com/questions/44910940/mean-squared-error-in-scikit-learn-ridgecv> 2019.04.06.

MELLÉKLETEK

- */image_downloader* – képek gyűjtésének automatizálása
 - */image_downloader.py* – képek gyűjtését végző script
 - */searchlist.txt* – tárgyak nevei, amelyekről képeket kívánunk gyűjteni
 - */rename_to_sequential.app* – képek elnevezéséhez használt Automator folyamat
- */collected_data* – a regressziós modell tanításához gyűjtött adatok
 - */data_from_ios_app* – az iOS alkalmazással gyűjtött adatok
 - */data_from_optitrack* – az OptiTrack rendszerrel gyűjtött adatok
 - */transform_ios_data.numbers* – az iOS alkalmazásból származó adatok átalakítása
 - */transform_optitrack_data.numbers* – az OptiTrack rendszerből származó adatok átalakítása
 - *data.csv* – adatok a regressziós modell tanításához
- */training* – gépi tanulás modellek létrehozása
 - */data_to_sframe.ipynb* – adatok SFrame formátumra alakítása
 - */classifier_training.ipynb* – objektum felismerő modell tanítása
 - */regression_training.ipynb* – regressziós modell tanítása
- */tracking* – az iOS alkalmazás projekt és forrásfájlai
- */arkit_tracking* – az ARKit alkalmazás projekt és forrásfájlai