

TDK-dolgozat

Bagladi Milán Zsolt

Mesterséges Intelligencia felhasználása emberi karjelzések értelmezésében

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

MESTERSÉGES INTELLIGENCIA TANSZÉK



Szerző:

Bagladi Milán Zsolt

programtervező informatikus BSc

1. évfolyam

Témavezető:

Dr. Gulyás László

egyetemi docens

Budapest, 2024

Tartalomjegyzék

1. Bevezetés	1
1.1. Problémafelvetés	1
1.2. Vázlatos tartalom	2
2. Korábbi releváns kutatások	3
2.1. Fogalmi ismertető	3
2.2. RNN rendőri karjelzések felismerésére	4
2.3. Pózbecslés Kinect rendszerrel	5
2.4. Összegzés	5
3. Megközelítés	7
3.1. Felismerési folyamat	7
3.2. OpenPose mint kulcspontbecslő megoldás	8
3.3. Normalizálás	9
4. Statikus karjelzések felismerése	12
4.1. K-means klaszterezés	12
4.1.1. Konfigurálás	12
4.1.2. A K-means algoritmus menete	12
4.1.3. Adatgyűjtés — Konfigurációs adathalmaz elkészítése	13
4.1.4. Kutatásban használt minták	14
4.1.5. Tesztelés menete	14
4.1.6. Konfigurációs adathalmaz mérete	14
4.1.7. Távfolságfüggvény	14
4.1.8. Időtényező	16
4.1.9. Valós idejű statikus felismerés	17
4.1.10. A módszer tanulságai	17
4.2. Megoldás neurális hálóval	21

4.2.1.	Adatgyűjtés	21
4.2.2.	Modell architektúra	22
4.2.3.	Tanítás	23
4.2.4.	Validációs lépés	23
4.2.5.	Időtényező a tanítás kapcsán	24
4.2.6.	Valós idejű felismerés	24
4.2.7.	Eredmény	25
4.3.	Statikus felismerő módszerek összehasonlítása	27
4.3.1.	A tanító adathalmaz mérete, címkézés	27
4.3.2.	Szakmai felkészültség szükségessége	27
4.3.3.	Idő és teljesítmény	27
4.3.4.	Hatékonyság	28
4.3.5.	Összegzés	28
5.	Dinamikus karjelzések felismerése	29
5.1.	Csúszó ablak megközelítés	29
5.2.	LSTM és GRU	30
5.3.	Adatgyűjtés	30
5.4.	Jelzések	32
5.5.	Modell architektúra	32
5.6.	Tanítás	34
5.7.	Validáció	34
5.8.	Időtényező a tanítás kapcsán	34
5.9.	Valós idejű felismerés	35
5.10.	Tesztelés	36
5.11.	Eredmény szabadon választott sebességre	37
5.12.	Kimenet több jóslásból	38
5.12.1.	Naiv megközelítés	38
5.12.2.	Egy helyes megközelítés	39
5.12.3.	Eredmények	40
5.13.	Felismerés megváltoztatott sebességű jelzésekre	41
6.	Interaktív robotvezérlés karjelzésekkel	43
6.1.	Komponensek	44
6.1.1.	Motorok	44

6.1.2. OpenCR Board	44
6.1.3. Raspberry PI	44
6.2. Megközelítés	44
6.3. Használt karjelzések	45
6.4. Megvalósítás	45
6.5. Értékelés	45
7. Összefoglalás	47
Köszönetnyilvánítás	48
Irodalomjegyzék	48
Ábrajegyzék	50
Táblázatjegyzék	51

Kivonat

A karjelzésekkel történő kommunikáció fontos szerepet tölt be a mindennapi életünkben. A dolgozat a mesterséges intelligencia használatával az emberi mozgásformák és pózok automatikus értelmezésére összpontosít, amely napjaink technológiai fejlődésének köszönhetően egyre inkább előtérbe kerül, különösen a közlekedésbiztonsági rendszerek terén, ahol az emberi gesztusok pontos felismerése és helyes értelmezése létfontosságú lehet. Az ilyen típusú megoldások fejlesztésekor elsődleges cél, hogy a rendszer képes legyen az emberi karjelzéseket gyorsan, pontosan és megbízhatóan feldolgozni.

Az emberi karjelzések két típusával foglalkoztam: statikus és mozgást tartalmazó, dinamikus jelzésekkel. A felismerési folyamat két lépésben történik: először az emberi alakról készült képeken kulcspontbecslést hajtok végre, majd a kapott adatok különböző mesterséges intelligencia algoritmusokkal kerülnek elemzésre.

A statikus karjelzések felismerésére a K-means klaszterezési módszert, valamint neurális hálózatot használtam. A dinamikus karjelzések értelmezésére speciális, idősorok kezelésére alkalmas mesterséges neurális hálózatot, GRU-t alkalmaztam. A kutatás során az OpenPose szoftvert használtam kulcspontbecslőként, amely segítségével a képek alapján a test kulcspontjai kerülnek meghatározásra, ezeket követően pedig a megfelelő algoritmusok elvégzik a jelzések osztályozását.

A dolgozat bemutatja az említett módszerek alkalmazhatóságát és új eredményként, egy módszeres összehasonlítás keretében kiemeli az egyes megoldások előnyeit és hátrányait. Ezek arra is rávilágítanak, hogy a legújabb, legmodernebb megoldások nem mindig jelentik a legjobb választást egy adott feladatra. Továbbá, a dolgozat részeként egy konkrét demonstrációs alkalmazás is bemutatásra kerül egy karjelzésekkel vezérelhető robot keretében, amely szemlélteti a vizsgált módszerek gyakorlati alkalmazhatóságát és alátámasztja a kutatási eredményeket.

1. fejezet

Bevezetés

Az élet számtalan területén találkozhatunk ember által adott karjelzésekkel, melyek értelmezése az emberek számára nyilvánvaló, azonban egy számítógép számára komoly kihívást okozhatnak. A számítási kapacitások növekedése miatt az elmúlt években a mesterséges intelligencia népszerű téma lett, és hatalmas fejlődésen ment keresztül. A mesterséges neurális hálók segítségével lehetőség nyílik a képi és videó alapú információk feldolgozására: például megvalósítható az emberi alakok megtalálása és a testrészek számítógépes feltérképezése.

1.1. Problémafelvetés

A különböző emberi mozgásformák és pózok automatikus értelmezése napjaink technológiai fejlődésének köszönhetően egyre inkább előtérbe kerül. Különösen igaz ez a közlekedésbiztonsági rendszerek terén, ahol az emberi gesztusok pontos felismerése és helyes értelmezése létfontosságú lehet. Az ilyen típusú technológiák fejlesztésekor elsődleges cél, hogy a rendszerek képesek legyenek az emberi test különböző jelzéseit, mint például a kar- és kézmozdulatokat azonnal és pontosan feldolgozni.

Felmerülhet az igény az emberi gesztusok, pózok felismerésére, értelmezésére. Az emberi karjelzések felismerésének jelentős alkalmazási területe lehet az önvezető járműveknél a rendőri karjelzések vagy a vasútnál a kézi jelzések értelmezése. A vasúti karjelzések felismerésének megoldása különösen érdekes téma, hiszen ez a kötöttpályás közlekedés emberi karjelzéseket használó területein az automatizációt nagymértékben meggyorsíthatná.

Az emberi karjelzések két típusba sorolhatók: vannak statikus, illetve dinamikus jelzések. Statikus jelzésnek nevezzük azt, amikor egy karjelzés jelentése egyetlen képkockából megállapítható, tehát a póz önmagában hordozza az információt, függetlenül a mozgás előzményeitől vagy következményeitől. Ez azt jelenti, hogy a statikus jelzések általában egyszerű, jól azonosítható gesztusok, mint például egy megállj jelet formázó kéz, vagy egy irányt mutató ujj.

Dinamikus jelzések pedig azok, amelyek értelmezéséhez a szekvenciális jelleget is figyelembe kell venni, továbbá a hasonló állapotokat felvevő dinamikus jelzéseknek is lehet eltérő a jelentése, például balra vagy jobbra körzés során ugyanazok a statikus pózok kerülnek jelzésre, de a jelentés megállapításához az irány értelmezése is szükséges.

1.2. Vázlatos tartalom

A dolgozat első felében két, statikus karjelzések felismerésre alkalmas megoldást mutatok be: elsőnek egy K-means algoritmuson alapuló módszert, majd másodszorra a problémát neurális háló betanításával kísérelem meg megoldani. Ezeket módszeresen összehasonlítom az elvégzett tesztelési eredmények alapján.

A dolgozat második felében a dinamikus karjelzések értelmezésével foglalkozom: egy speciális, mesterséges intelligencia alapú hálót tanítok be az említett probléma megoldására. A felismerés pontosítása céljából bemutatok egy olyan kiegészítő módszert is, amivel a karjelzések értelmezése magasabb szinten, több kimenet egyesítésével valósul meg.

2. fejezet

Korábbi releváns kutatások

Ebben a fejezetben áttekintünk néhány, releváns kutatást a témában azért, hogy az olvasó is megismerhesse a témában már megszületett jelentősebb megközelítéseket, módszereket.

2.1. Fogalmi ismertető

Az ismertetett cikkek megértéséhez szükséges néhány fogalom definiálása, a rövidítések feloldása. A fontosabb, felhasznált megoldásokhoz igyekszem az ezekről szóló cikket is mellékelni.

Az RNN [1], vagy rekurrens neurális hálózat, egy olyan típusú mesterséges neurális hálózat, amely különösen jól alkalmazható időben változó adatok, mint például a szöveg vagy videók feldolgozására. Az RNN legfőbb jellegzetessége, hogy képes "emlékezni" a korábbi információkra a hálózaton belüli visszacsatolások révén, ami lehetővé teszi számára, hogy összefüggéseket ismerjen fel az adatok idősorában. Ez a tulajdonság teszi ideálissá például a nyelvi modellezés és más, időben sorra következő adatok feldolgozására is.

A konvolúciós művelet egy matematikai eszköz, amelyet gyakran használnak kép-feldolgozásban. A konvolúciós neurális hálózatok (CNN-ek) [2] alkalmazásával a gépi tanulás képes hatékonyan jellemző vonásokat meglátni a képeken, például éleket és szín-átmeneteket, mely információk összegezésével elérhetővé válik a kép magasabb szintű értelmezése.

A Microsoft Kinect rendszer, amelyet eredetileg videojátékokhoz fejlesztettek ki, a Microsoft egyik innovációjaként képes a mozgás érzékelésére és a kulcspontok (key-

points) meghatározására az emberi testen. A kulcspontok az emberi test fontos anatómiai pontjait jelentik, mint például a könyök, a térdek, a csípő és a vállak, amelyek megjelenése lehetővé teszi a testtartás és a mozgások pontos analízisét. A Kinect technológia az infravörös érzékelők és kamerák segítségével készít térbeli képet, amelyben az egyes felhasználókat felismerve a szoftver valós időben becsli meg ezeket kulcspontokat. Ezáltal képes követni a testmozgás dinamikáját, és interpretálni az emberi gesztusokat, így forradalmasítva a gépi látás alkalmazásait.

2.2. RNN rendőri karjelzések felismerésére

Az egyik, ide kapcsolódó cikkben [3] rekurrens neurális hálókat és konvolúciót alkalmaznak a szerzők a közúti közlekedés során a rendőri karjelzések felismerésére.

A cikkükben ismertetett módszer roppant költséghatékony, hiszen nem igényel drága, a képi adatok mellett más adatokat is szolgáltató kamerát vagy szenzort. Ehelyett egy egyszerű és olcsó RGB kamera használatával működik, és szerintem ez egy hatalmas előny. A módszer egyedi, mivel nem szükséges az emberi pózbecslés klasszikus formája, hanem a nyers képből közvetlenül előállítható a jóslt eredmény, ami 30 képkocka/másodperc sebességgel valós időben történik.

Mivel a képkockák közvetlenül kerülnek feldolgozásra, ezért a modell tanítása különféle hátterek felhasználásával történt, hogy a kimenet ne legyen érzékeny a környezeti tényezőkre. A rendszer képes megkülönböztetni, hogy kihez szól a karjelzés, és csak a kijelölt járműnek ad utasítást, ha a jelzések értelmesek.

A cikkben négy különböző RNN modellt teszteltek, figyelembe véve a bemeneti szekvencia hosszát és a rétegek számát. A bemeneti szekvencia hossza nagyon fontos tényező dinamikus jelzések felismerésében. A GRU hálózat négy réteggel bizonyult a legjobbnak, a túltanulás elkerülése mellett hatékony működést biztosítva. A Gated Recurrent Unit egy speciális típusú rekurrens neurális hálózat (RNN), amelyet szekvenciális adatok, például szöveg vagy idősoros adatok feldolgozására használnak. A GRU célja, hogy megoldja az RNN-ek tipikus problémáját, az úgynevezett "eltűnő gradiens" problémát [4], amely akkor fordul elő, amikor az RNN tanulás során lassan elfelejti az előző bemenet hatását, így képtelen fenntartani az információkat hosszú időn keresztül.

A kutatás fontos előrelépést jelent a közúti közlekedés során történő karjelzések felismerésében, és megnyitja az utat az egyszerűbb, kisebb számítási igényű megoldások felé. Új karjelzések felvétele azonban nagyon sok munkával járna, hiszen az adatok rögzíté-

sét, majd a modell tanítását a használónak kell elvégeznie. A tanítás emellett hatalmas erőforrásokat igényel.

2.3. Pózbecslés Kinect rendszerrel

A téma megismerése céljából egy másik cikk [5] rövid összefoglalóját is érdemes lehet elolvasni. A szerzők konvolúciós neurális hálózatot alkalmaznak kézi gesztusok felismerésére és értelmezésére.

Bemeneti eszköz gyanánt a Microsoft Kinect kamerájának valós idejű adatait használják. A Kinect olyan kamerára hasonlító hardver, amely pontos kulcspontbecslést biztosít az emberi testen és a rendszer felépít egy 3D-s virtuális csontvázat. A kutatók a Kinect által érzékelt 20 kulcspontból 8-at használtak a jelzések értelmezéséhez, így egyszerűsítve a problémát és csökkentve a zajosságot (tehát felesleges adatok nem jelennek meg a bemenetben, továbbá a zajos adatokat kevésbé lényeges kiszűrni).

Az előre rögzített nyersanyagot videószerkesztő szoftverrel megvágták, és lineáris interpolációval azonos hosszúságúvá tették a mintákat. Ez azt jelenti, hogy minden dinamikus jelzést tartalmazó minta egyenlő hosszú lett.

Készítettek egy adathalmazt, amellyel tanították, illetve mérték a modellt. Kihangsúlyozzák, hogy 5 nő és 5 férfi által kerültek kiadásra a különféle gesztusok, de ők a kutatás semmilyen más fázisában nem vettek részt, így intuitív módon kellett a mozgásokat bemutatniuk. Mindegyiküknek tízszer kellett egy adott gesztust ismételnie, és ezek közül a legjobb öt vet részt a modell tanításában. Tehát kézzel, mesterségesen kiszűrték a kutatók a zajos adatokat a felvételek közül.

Az említett kutatás kiválóan megmutatja, hogy csupán a csontváz pontjaiból is lehet kiváló pózbecslést végezni, viszont a költséges hardver nem biztos, hogy feltétlen szükséges. Jelen dolgozat készítésének pillanatában már több éve nem gyártják a hardvert, így a módszer használatát ez roppant nehézkessé teszi.

2.4. Összegzés

Az imént két, egymástól jelentősen eltérő megközelítést idéztem korábbi kutatásokból. Látható, hogy mindkét megoldás a már felvetett problémára keresi a megoldást, tehát az emberi karjelzéseket próbálja értelmezni, de a két megközelítés jelentősen eltér egymástól.

A RNN-es megoldáshoz csak egy egyszerű RGB kamera szükséges, nem igényel drága szenzorokat. Emellett a valós idejű képfeldolgozás sebessége (30 képkocka/másodperc) és a környezeti tényezőkkel szembeni nagy mértékű tűrés is dicséretes. Azonban hátrányok is akadnak, mint például az új karjelzések integrálásának munkaigényessége és a jelentős számítási erőforrásigény, ami korlátozhatja a szélesebb körű alkalmazást.

A Kinect rendszerrel végzett pózbecslés szintén előnyös megoldás, mivel a 3D csontváz rekonstrukciója és a kulcspontok magas szintű pontossága kiváló alapot nyújt a gesztusok értelmezésére. Hátrányként jelentkezik viszont a Kinect hardver megszűnt gyártása, ami akadályozza a technológia további elterjedését és fenntarthatóságát.

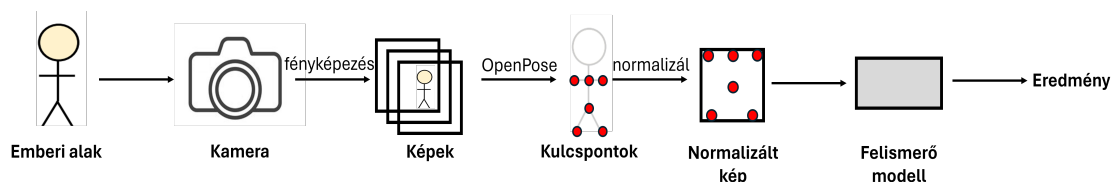
Mindkét cikk innovatív megoldást nyújt a karjelzések értelmezésére, azonban a felsorolt előnyök mellett vannak hátrányok is. Ennek ellenére a technológia fejlődésével és az új kutatási irányok megjelenésével lehetséges, hogy ezeket a limitációkat kiküszöböljük. Ezáltal még gyorsabb, egyszerűbb és hatékonyabb rendszereket hozhatunk létre, amelyek még szélesebb körű alkalmazást tesznek lehetővé a jövőben.

3. fejezet

Megközelítés

3.1. Felismerési folyamat

Ebben a fejezetben a karjelzések felismerésének folyamatát írom le vázlatosan. Először az emberi alakról egy kamerával készítünk képeket, majd ezeken elvégezzük a kulcspontbecslést, tehát egy digitális csontvázat állítunk elő. A csontváz feldolgozása során normalizáljuk a keletkezett kulcspontokból álló vektort, majd az így létrejött vektort betöltjük a különféle megoldási módszereket megvalósító modellekbe. Ezek működését a megfelelő szakaszban részletezem. A modell eredményként megadja a felismert gesztust, majd a bemenetre új adat küldhető.



3.1. ábra. A felismerés bemutatása folyamatábrán.

A 3.1 folyamatábrán az említett lépések és egységek illusztrálják a leírt folyamatot. Az ábra bemutatja az emberi alakról készült képek feldolgozásának lépéseit, a digitális csontváz előállítását, és a felismerési modell működését.

A statikus pózfelismerést megvalósítom két, lényegében különböző módszerrel, illetve a mozgást tartalmazó karjelzések értelmezését is végrehajtom egy rekurrens neurális hálóval. A karjelzések értelmezése után ezek jelentése már parancsszintre emelve alkalmazható a közúti/vasúti közlekedésben.

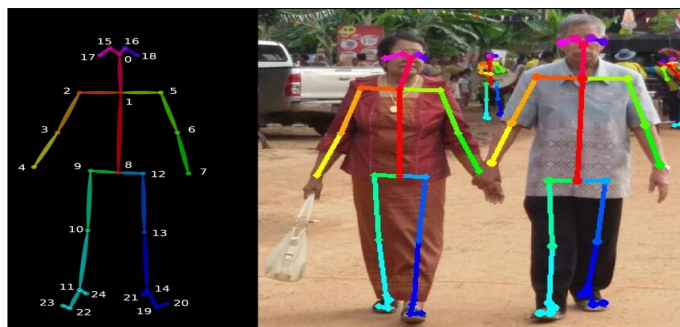
3.2. OpenPose mint kulcspontbecslő megoldás

A kutatás során felmerült egyik probléma az volt, hogy a kamera által létrehozott képből egy olyan adathalmazt nyerjünk ki, amely tartalmazza az adott karjelzés felismeréséhez szükséges információkat (és lehetőleg ne tartalmazzon a felismerés szempontjából irreleváns dolgokat).

A tudományos cikket olvasva eljutottam az OpenPose [6] nevű szoftverhez. Az OpenPose nyílt forráskódú program, amelyhez többféle kulcspontbecslő modell is letölthető. Ahogy már említettem, a pózbecslést kulcspontokból álló csontváz használatával végzem el, ami már az idézett kutatásban megfelelő módszernek bizonyult. Mivel az eféle megközelítés kielégíti a fent említett kritériumokat, továbbá a feladatot számszerű adatok feldolgozására redukálja, így ez egy megfelelő választás.

A kutatáshoz a BODY-25 nevű modellt választottam, ugyanis ezt ajánlották általános alkalmazásokra, illetve a BODY-25 által épített csontváz a felsőtesti kulcspontokat praktikusabban építi fel, mint a COCO modell. Az OpenPose használatának számtalan előnye van a más, kezdetleges pózbecslő algoritmusokhoz képest, hiszen nemcsak gyors és pontos, hanem már előre betanított súlyokkal rendelkezik, így gyakorlatilag egy kész kulcspontbecslő alkalmazás.

Az OpenPose kiemelkedik a többi pózbecslő megoldás közül, hiszen egyszerűen alkalmazható a valós idejű pózfelismerési feladatokban. A rendkívüli sebessége és pontossága révén az OpenPose kiválóan alkalmas akár komplex feladatok megoldására is, miközben minimális futásidővel dolgozik, ami létfontosságú lehet valós idejű adatok feldolgozása során. Az OpenPose már eleve tartalmaz előre betanított modelleket, amik lehetővé teszik azt, hogy azonnal használatba vegyük a rendszert anélkül, hogy időigényes és költséges tanítási folyamaton kellene átesni. Az OpenPose emellett támogatja több ember egyidejű felismerését is, ami tovább bővíti a lehetőségeket.

3.2. ábra. OpenPose felismerés Body-25 modell alapján.¹

Ugyan a fent említett kezdeti kritériumok teljesülnek, hiszen elegendő, releváns adathoz jutunk, azonban az adathalmaz, amelyet az OpenPose készít, még nem eléggé jó ahhoz, hogy a felismerést el lehessen végezni rajta.

A program egy képkockán elvégzi a felismerést, majd 25 darab, rendezett számpárt ad eredményül. A számpárok rendre kulcspontokat jelentenek a képernyőn, tehát megadják azt, hogy a képernyő pontosan melyik pontján volt fellelhető az adott emberi kulcspont. Vegyük észre, hogy az OpenPose kimenetét nem elhanyagolható módon befolyásolja az ember képernyőn elfoglalt helye, továbbá a kamerától vett előre-hátra mozgás is átszabja a pontthalmaz kiterjedését a síkon. A problémákat még fokozza, hogy eltérő testalkatú emberek eltérő arányokkal rendelkező pontthalmazként kerülnek a kimenetre, így az adatok nem lesznek konzisztensek. Ebből következik, hogy ilyen formájukban még alkalmatlannak a felismerések végrehajtására.

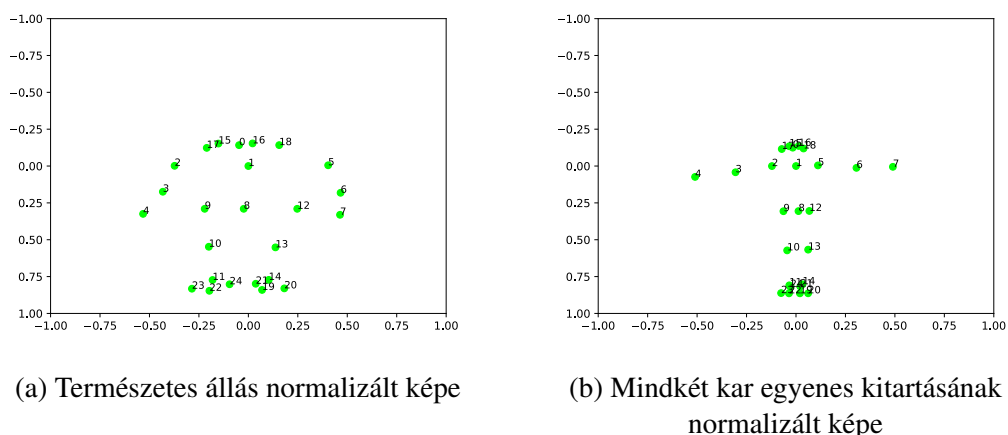
3.3. Normalizálás

Ahhoz, hogy a pózfelismerés általános esetben is működni tudjon, az OpenPose kimeneti adathalmazát oly módon kell transzformálni, hogy az egymáshoz közeli pózok hasonlósága a pontthalmazban is valamilyen módon tükröződjék. Itt nevezzük közelinek azon emberi testtartásokat, amelyeket az átlagos emberi szemlélő azonosnak talál, tehát ezek egymással a lényeges jegyekben az általánosan elvárt mértékig egyeznek.

Először a képernyőn való elhelyezkedés befolyásától szabadulunk meg oly módon, hogy a pontthalmazt egy alternatív, általunk alkotott Descartes-féle koordináta-rendszerbe emeljük át úgy, hogy egy jellemző, lehetőleg középső kulcspontot tekintünk origónak. Minden más pont a kitüntetett, origóban lévő kulcsponthoz képest helyezkedik el rela-

¹Forrás: https://www.researchgate.net/figure/The-left-image-is-the-OpenPose-body-25-keypoints-fig4_364309383

tív módon, így az emberi test képen való elhelyezkedéstől való függés nem jelent már problémát.



3.3. ábra. A kulcspontbecslés és a normalizálás utáni adatok 1x1 méretű diagramja, a Body-25 modell sorszámaival ellátva.

Az ábrán látható, hogy origónak az 1-es sorszámú pontot választottam. Az említett pont a különféle pózoknál egyaránt középen helyezkedik el, a biológia adottságoknak megfelelően helye nem változik.

A különféle testalkatok kezelésére normáljuk az adathalmazt. A normálás során a ponthalmazt úgy transzformáljuk, hogy a keletkezett pontok egy 1x1-es egységnégyzetben elférjenek. Eközben kiemelten fontos arra ügyelni, hogy az adathalmaz a felismerés szempontjából releváns információkat ne veszítse el, tehát a transzformálás a távolságokon kívül mást ne módosítson, hiszen ezesetben a felismeréshez szükséges, esetlegesen fontos információk (relatív távolságok, irányok, pontok helyzete egymáshoz képest) csorbulhatnak. (Itt megjegyzem, hogy a leírt módszer mondjuk a szöveget módosíthatja, azonban ezek egy transzformáló lépés után jellemzően ugyanúgy változnak, tehát ez nem okoz bajt.) A transzformálást egyaránt elvégezzük a vízszintes és a függőleges tengely mentén is úgy, hogy a két legtávolabbi pont megfelelő koordinátájának különbségének reciprokával megszorozzuk az összes pont megfelelő koordinátáját.

Az eljárás végén egy olyan 1x1-es négyzetet kapunk, amely az összes kulcspontot tartalmazza úgy, hogy a felismeréshez szükséges információk megmaradnak.

<i>Eredeti méretek meghatározása</i>	
$\text{maxY} = -\text{INF}$	
$\text{maxX} = -\text{INF}$	
$\text{minX} = \text{INF}$	
$\text{minY} = \text{INF}$	
p in pontok	
	$\text{maxX} = \max(\text{maxX}, p[0])$
	$\text{minX} = \min(\text{minX}, p[0])$
	$\text{maxY} = \max(\text{maxY}, p[1])$
	$\text{minY} = \min(\text{minY}, p[1])$
<i>Normalizálás 1x1-es képpé</i>	
$\text{varX} = 1 / (\text{maxX} - \text{minX})$	
$\text{varY} = 1 / (\text{maxY} - \text{minY})$	
p in pontok:	
	$p[0] = p[0] * \text{varX}$
	$p[1] = p[1] * \text{varY}$

3.4. ábra. Normalizálás algoritmus. A bemenet egy *pontok* nevű kételemű vektorokat tartalmazó vektor úgy, hogy $\text{pontok}[i][0]$ és $\text{pontok}[i][1]$ rendre az x és y koordináták.

4. fejezet

Statikus karjelzések felismerése

Statikus (mozgást figyelembe nem vevő) emberi pózok felismerésére kétféle, lényegében különböző módszert is megvizsgáltam. A következő bekezdésekben bemutatom mindkét módszer működését, majd egy módszeres összehasonlítást végzek.

4.1. K-means klaszterezés

A K-means népszerű klaszterezési technika, mellyel adatokat tudunk az algoritmus által kiszámolt K darab halmazokba besorolni, így különféle testtartások osztályozását is képes elvégezni a módszer. Jellemzője, hogy viszonylag kevés mintával lehetséges a konfigurálása (ezt itt szándékosan nem nevezem tanításnak), továbbá a bemenő adatokat lineáris időben képes a legjellemzőbb csoportba besorolni.

4.1.1. Konfigurálás

A K-means konfigurálásához szükség van egy nemnulla természetes számra, amely a halmazok (és így a centroidok) számát adja meg. Ez a K érték. Szükséges továbbá K-féle, a halmazokat kellően jellemző kezdeti minta. A felismerés konfigurálása során a kezdeti adatokat K darab halmazba rendezi az algoritmus, miközben kiszámol egy-egy, az adott halmazt legjobban leíró „átlagos” elemet. Az átlagos elemeket centroidoknak nevezzük.

4.1.2. A K-means algoritmus menete

A módszer magját a centroidok adják, így ezek kiszámítását ismertetem először. A struktogram a 4.1 ábrán látható. A centroidokat a következőképp számoljuk:

1. Induláskor véletlenszerűen inicializáljuk az összes centroidot.
2. Minden mintához hozzárendeljük a hozzájuk legjobban hasonlító aktuális centroidot.¹ A hozzárendelés után a most egy halmazba tartozó adatokból új centroidokat számolunk. Az eljárást addig folytatjuk, amíg van olyan centroid, amely megváltozott az előző iterációhoz képest.
3. A centroidok elkészültek.

Mivel a centroidokat csak a konfigurációs adathalmaz határozza meg, így azokat elegendő egyszer, a konfigurációs fázisban kiszámolni, majd azokat elmentve a felismeréshez bementi adatként felhasználni.

4.1.3. Adatgyűjtés — Konfigurációs adathalmaz elkészítése

A kezdeti adatgyűjtés során megfelelő képkockákat gyűjtünk azért, hogy a halmaz centroidját átlagolással ki lehessen számolni. A módszer fontos jellemzője, hogy a begyűjtött adatok címkézése nem szükséges, hiszen az algoritmus a K -darab centroidot az összesített, címkézetlen adatokból képes kiszámolni.

Az adatgyűjtés során érdemes arra ügyelni, hogy a rögzített adatok tiszták legyenek, tehát a lehetőségeknek mérten csak olyan adatok kerüljenek a konfiguráló adathalmazba, amelyek a jövőben kiszámítandó centroidhoz kellően hasonlóak. Amennyiben a konfigurációs adathalmazban vannak kiugró értékek, akkor azok az átlagot egy nem kívánt irányba mozdítják el, így a K -means kiszámításakor jelentősen befolyásolják a centroidok alakját. Az ilyen, minden más mintától távol eső adatokat könnyedén ki lehet szűrni a konfiguráció előtt. Vegyük észre, hogy nagy valószínűséggel hibásak azok az adatok, amelyekhez nem rendelhető még néhány másik, közeli adat. Ezek eltávolításával létrehozható olyan adathalmaz, amely eléggé tiszta ahhoz, hogy a K -means konfigurációt elvégezzük és megfelelő centroidokat számoljunk.

A konfigurációs adathalmaz elkészítése során szintén szükséges arra figyelni, hogy az egyes képkockákon minden BODY-25 kulcspont látható legyen. Ha van olyan képkocka, amelynél nem az összes kulcspont került felismerésre (ilyenkor az OpenPose kimeneten nullák szerepelnek), akkor ez az adat nem használható a konfigurációs adathalmazban, ugyanis a konfigurációs fázisban az átlagolás során a fel nem ismert kulcspont a centroidokat jelentősen befolyásolhatja.

¹Egy mintához az a centroid hasonlít legjobban, amely centroidtól vett távolsága a minimális az összes centroid között.

4.1.4. Kutatásban használt minták

A kutatás során háromféle konfigurációt készítettem: 2 pózt, 4 pózt és 9 pózt használót. A 4.2 ábrán látható karjelzések közül a) és b) szerepel a 2 pózt használó teszteken, majd ezeket kiegészítve c) és d) a 4 pózt használókban. A maradék jelzések a 9-es tesztekben szerepelnek. Azért bontottam szét ezeket, hogy a két vizsgált módszer közötti különbség jobban mérhető legyen, továbbá a K-means is máshogy teljesít egyszerű és összetett feladatokban.

Figyeljük meg, hogy a normalizálás valóban egy 1x1-es négyzetbe transzformálja a pontokat. A pontok kirajzolásából látható, hogy az emberi szem számára is megmaradnak a felismeréshez szükséges jellemzők.

4.1.5. Tesztelés menete

Minden teszthez előre elkészítettem a centroidokat, majd lefutattam a felismerési modellt a tesztadatokra. A fejezetben levő tesztek olyan címkézett adatok voltak, amelyeket nem használtam fel a centroidok elkészítéséhez. Egy teszt akkor és csak akkor számít sikeresnek, ha a K-means jóslata megegyezik a címkével.

4.1.6. Konfigurációs adathalmaz mérete

A centroidok meghatározásához szükséges konfigurációs adathalmaz számossága nem kell, hogy túl nagy legyen. Ez az én adathalmazaimban jellemzően K-szor 10-20 adatot jelent. A túl sok adat jellemzően nem segíti a kívánt centroidok kialakítását, ugyanis így sokkal diverzebb lesz a konfigurációs adathalmaz, továbbá az adatgyűjtés és a centroidok számítása is több időbe kerül. Fontosnak tartom megemlíteni, hogy az általam ajánlottnál több konfigurációs adat a K-means felismerés elméleti pontosságát nem fokozza, hiszen ezek csak a centroidok elkészítéséhez szükséges kezdeti adatok, és maga a felismerés a kiszámított centroidokkal történik.

4.1.7. Távolságfüggvény

A klaszterezéshez érdemes az adatok egy eddig még nem említett értelmezését bevezetni. Tudjuk, hogy a Body-25 modell 25 pár koordinátát hoz létre, amelyeket a már említett módszerrel normalizálunk a konzisztens felismerés végett. A továbbiakban minden ilyen adatot az ötvendimenziós tér egy pontjaként érdemes elképzelni.

Az új értelmezésnek megfelelően van K darab pont, amelyek a centroidok. Minden képkocka feldolgozása során lesz egy pont (ami egy 50 hosszú vektor), amelyről el kell dönteni, hogy melyik centroidhoz van a legközelebb. Ehhez azonban a távolság definícióját még szükséges tisztázni.

A bevezetett fogalmakkal legkézenfekvőbb az Euklidészi-távolság kiszámítása, melynek értelmében az aktuális képkockán azon póz látható, amely póz pontjához a kiválasztott centroidnál nincs közelebbi centroid. Ezzel a módszerrel már nagyon pontos statikus pózbecslés valósítható meg (ezt igazolja a 4.1 táblázat), azonban ez a pontosság további megfontolásokkal még tovább javítható. Ezt a következőkben súlyozásnak nevezem és a pontos algoritmust is bemutatom.

K-means teljesítménye a normalizált adatokon súlyozás nélkül/súlyozással			
Jelzések száma	Konfigurációs adatok száma	Tesztadatok száma	Eredmény
2	30	5053	98.81%
2			100%
4	60	19578	99.75%
4			100%
9	135	24379	97.27%
9			99.84%

4.1. táblázat. K-means speciális súlyokkal. Az alsó eredmény a súlyozott.

A 4.1 táblázatban láthatóak azok a mérési eredmények, amelyeket a 4.2 táblázat súlyait használva mértem meg.

A 4.3 ábrán figyeljük meg a *balemelt-balderek* és a *jobbderek-jobbgyenes* cellákat! Az értékek nagyon lecsökkentek, tehát a felismerés pontosabb lett.

Ahogy a mérésekből látszik, általánosan fokozható a pontosság azzal, ha az Euklidészi-távolságnál egy jobb távolságfüggvényt definiálunk.

A távolságfüggvénytől várjuk el azt, hogy teljesítse a távolság axiómáit, továbbá a felismerés szempontjából jelentősebb tulajdonságokat nagyobb mértékben vegye figyelembe, mint az Euklidészi-távolság.

A súlyozás során két pont távolságánál néhány dimenzióban vett távolság nagyobb mértékben kerül a távolság összegébe, mint mások. Az általam használt függvény vár egy $A[0..49]$ és egy $B[0..49]$ megfelelően normalizált, két pontot leíró vektort és egy $W[0..49]$ súlyozó vektort. Ezek alapján A és B súlyozott távolsága a következő:

$$\text{Táv}(A,B) = \sqrt{\sum_{i=0}^{49} ((A[i] - B[i]) * W[i])^2}$$

A nem súlyozott teszteken a K-means viszonylag jó eredményt ér el (4.1-as táblázat): a könnyebb esetekben 2 és 4 jelzés esetén súlyozás nélkül is képes a tesztek jelentős részét helyesen megoldani. A nehezebb esetben, tehát 9 jelzés esetén a súlyozás nélküli megközelítés jelentősen rosszabbul teljesített, mint a felsőtestre szűkített, súlyozott megközelítés.

Pont sorszáma	Súly (W_i)
0 (origó)	1
1	1
2	1
3 (könyök)	10
4 (kézfaj)	10
5	1
6 (könyök)	10
7 (kézfaj)	10

4.2. táblázat. Súlyok táblázata.

Több, mint 2%-os javulást lehetett elérni azzal, ha csak a felsőtesti pontok távolságából képeztem a távolságot. További javulást lehet elérni úgy, ha a könyök és a kézfaj kulcspontjait nagyobb mértékben vettem figyelembe a felismerésnél. Ekkor csak a 0 – 7 pontok számítottak, amelyek súlyait a (4.2-as táblázat) mutatja. A súlyozó vektor minden más értéke 0.

A kutatás során több súlyozó vektort kipróbáltam azért, hogy a legkiválóbbat megtaláljam. A próbálkozásokat a (4.3 táblázat) tartalmazza. Látható, hogy felesleges túl nagy súlyokat beállítani az érdekes dimenziókhoz, továbbá a kevésbé fontos adatok kihagyása (tehát 0 súllyal való figyelembe vétele) is megfelelő eredményre jut.

4.1.8. Időtényező

Ahogy a 4.4 táblázat mutatja, a K-means módszer viszonylag gyors megoldás, tehát minimális számítási kapacitást és processzoridőt igényel a használata. Látható, hogy egy elem besorolása is elhanyagolható időt vesz el.

A következő mérési eredményeket nagyban befolyásolja a futatókörnyezet teljesítménye. Az értékek egy Intel i7 12700KF 3,6 Ghz processzoron kerültek mérésre. A dolgozatban szereplő mérési eredmények egy konkrét számítógépen kerültek mérésre.

Jelzések száma	Helyes felismerés (%)	Súlyvektor (W[0..7])
2	100	[minden 1 súllyal]
2	100	[1, 1, 1, 1, 1, 1, 1, 1]
4	99,77525794	[minden 1 súllyal]
4	100	[1, 1, 1, 1, 1, 1, 1, 1]
9	97,276344	[minden 1 súllyal]
9	99,827721	[1, 1, 1, 1, 1, 1, 1, 1]
9	99,819517	[1, 1, 1, 10, 20, 1, 10, 20]
9	99,815415	[1, 1, 1, 50, 100, 1, 50, 100]
9	99,840026	[1, 1, 1, 100, 50, 1, 100, 50]
9	99,840026	[1, 1, 1, 200, 100, 1, 200, 100]
9	99,844128	[1, 1, 1, 100, 100, 1, 100, 100]
9	99,844128	[1, 1, 1, 200, 200, 1, 200, 200]
9	99,844128	[1, 1, 1, 50, 50, 1, 50, 50]
9	99,844128	[1, 1, 1, 10, 10, 1, 10, 10]
9	99,844128	[0, 0, 0, 10, 10, 0, 10, 10]
9	99,844128	[0, 0, 0, 1, 1, 0, 1, 1]

4.3. táblázat. Súlyozó vektorok tesztje.

4.1.9. Valós idejű statikus felismerés

A valós idejű statikus jelzések felismerését az eddig leírt ismeretekkel már könnyedén meg lehet valósítani. A centroidokat javasolt előre kiszámolni és a felismerés előtt a programba betáplálni. A felismerés során a kamera következő képkockáján el kell végezni az OpenPose kulcspontbecslést, majd a kapott adatokat szabványosan normalizálni kell. Ezután a távolságfüggvény segítségével meg kell határozni a legközelebbi centroidot, majd az aktuális mintát a megtalált halmazba sorolni.

A távolságfüggvény szerint meg lehet adni küszöbtávolságokat. Ezek arra jók, hogy nem minden jelzést sorolnak be, hanem csak azokat, amelyek megfelelően közel esnek egy adott centroidhoz. (Világos, hogy a két legközelebbi centroid távolságának felénél kisebb küszöb már helyes eredményre vezet.)

4.1.10. A módszer tanulságai

A K-means módszer egyszerűsége egyszerre áldás és átok. Az algoritmussal könnyedén létrehozhatóak az adott halmazt kiválóan leíró átlagos elemek, és a lineáris futásidő miatt a felismerés roppant gyorsan végrehajtható.

A konfigurációs adathalmaz elkészítése könnyű és olcsó, a továbbfejlesztett távolságfüggvénnyel a pontosság is kellően nagy. A módszer hátránya az egyszerűségében rejlik. Mivel a besorolás során csak a távolságokra hagyatkozik, figyelmen kívül hagyja a pontok

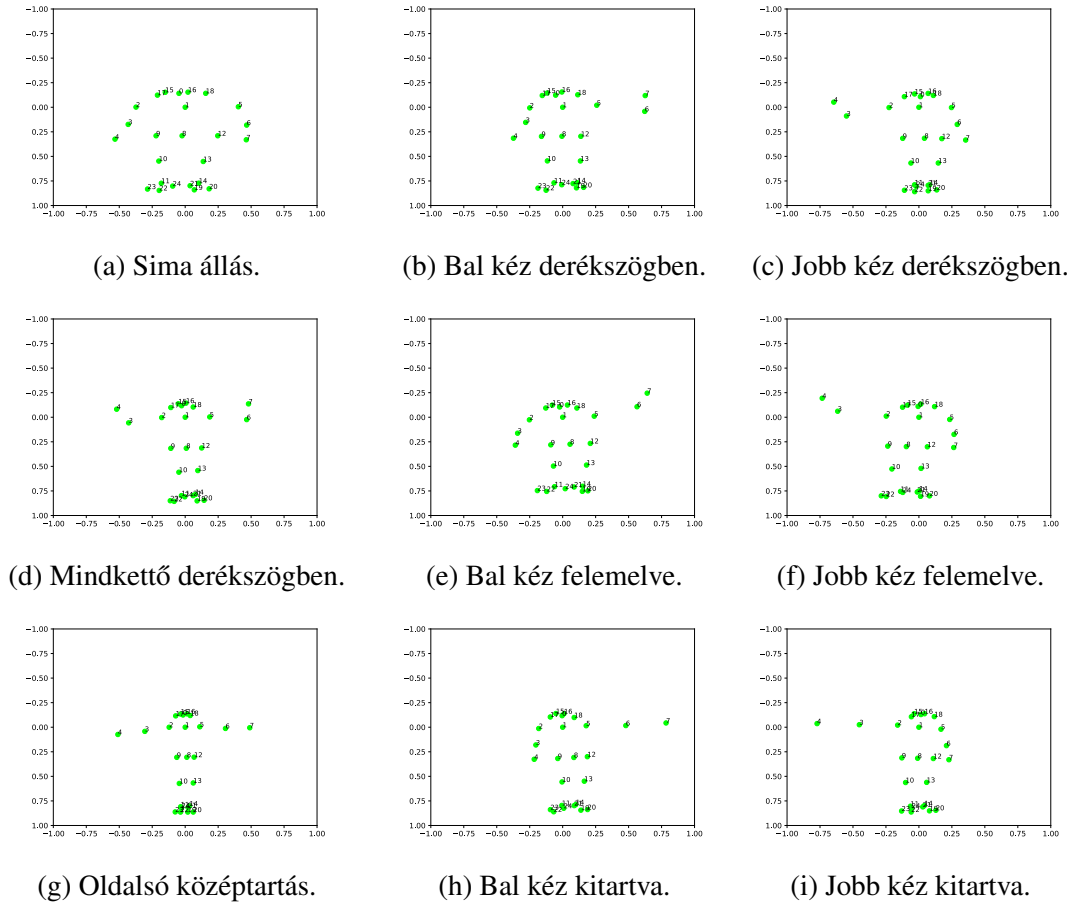
Jelzések száma	Egy mintára jutó átlagos idő (s)
2	0,000077122
2	0,000068838
4	0,000932569
4	0,000091583
9	0,000160322
9	0,00013753
9	0,000132296
9	0,000131315
9	0,00013023
9	0,000133199
9	0,000131133
9	0,000131485
9	0,000130332
9	0,000132198
9	0,000128453
9	0,000128003
Átlag (2)	0,00007298
Átlag (4)	0,000512076
Átlag (9)	0,000005017

4.4. táblázat. Az egy mintára jutó jóslás idejének mérése.

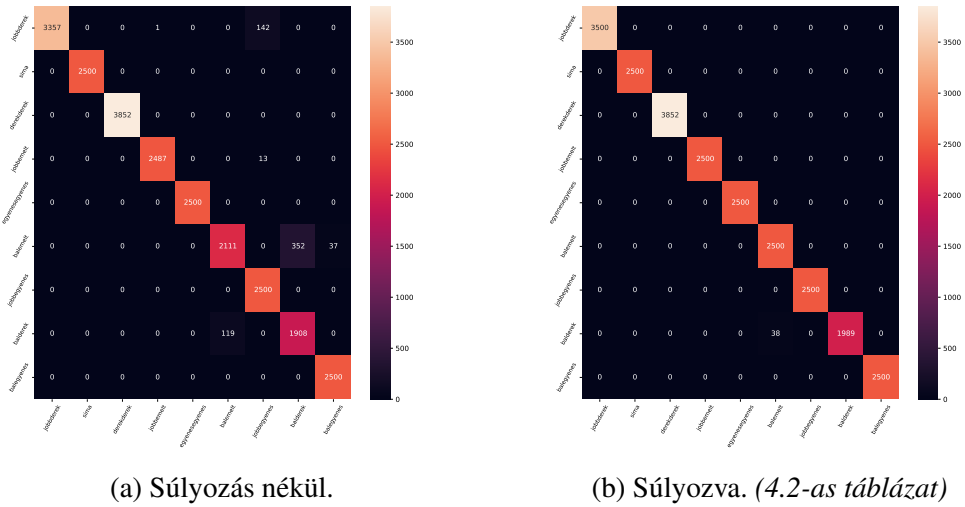
sorszámát (így biológiai helyét). Nagyon sok, alig különböző póz esetén jelentősen romlik a felismerés pontossága, ugyanis a távolságok közötti távolságok különbsége kicsi.

<i>Centroid számítás</i>	
valtozott>P	
besorol()	
ujcentroid()	
<i>A minták az aktuális centroidokhoz való besorolása - Besorol()</i>	
i in range(len(mintak)):	
legjobb = 0	
j in range(len(centroidok))	
if (tavolsag(centroidok[legjobb], mintak[i]) > (tavolsag(centroidok[j], mintak[i])):	
legjobb = j	
besorolas[i] = legjobb	
<i>ujcentroid() - Új centroidok számítása</i>	
i in range(len(centroidok))	
aktujatlag = [0] * 50	
db = 0	
j in range(len(mintak))	
if (besorolas[j] == i):	
k in range(50)	
aktujatlag[k] = aktujatlag[k] + mintak[j][k]	
db = db + 1	
db == 0:	
true	false
db = 1	
k in range(50)	
aktujatlag[k] = aktujatlag[k] / db	
(aktujatlag != centroidok[i])	
true	false
valtozott = True	
centroidok[i] = aktujatlag	

4.1. ábra. A centroidok kiszámításának algoritmus. A *mintak* vektorban tároljuk a konfigurációs adatokat és a *centroidok* vektorban lesznek a kiszámolt centroidok.



4.2. ábra. A 9 különböző karjelzés, amelyet a kutatásban használtam.



4.3. ábra. Teszteredmények 9 jelzésre.

4.2. Megoldás neurális hálóval

A statikus jelzések felismerését egy másik módszerrel is megvalósítottam. A következőkben megmutatom, hogy hogyan lehet egy neurális hálóval elvégezni az adatok klaszterezését.

A K-means módszerhez képest jelen megoldásnál néhány megvalósítási lépés jelentősen eltér, így ezek bemutatására különös figyelmet fordítok.

4.2.1. Adatgyűjtés

Az adatok begyűjtése a K-means adatgyűjtéséhez hasonlóan történhet: vegyünk egy egyszerű kamerát, majd rögzítsünk képkockákat a megfelelő klaszterhez. Az OpenPose végezze el a kulcsponthez tartozó pontbecslést, majd a normalizált képet tároljuk el a tanító adatok közé.

A K-means bemeneti adathalmazához képest annyi különbség azonban adódik, hogy jelen esetben a tanításhoz címkézett adatokra van szükség, tehát minden képkockáról tudni kell, hogy pontosan milyen klaszterbe tartozik a rajta levő karjelzés.

A kutatás során én úgy készítettem el az adathalmazt, hogy egy adott karjelzéshez álltam be, és azt adtam folyamatosan, kisebb változtatásokkal. A kamerával videót rögzítettem, majd az OpenPose és a normalizálás a videóból kinyert képkockákon történt meg. Így a tanító adathalmaz könnyen kezelhető lett, és a címkézést is könnyedén el lehetett végezni az éppen aktuális karjelzésnek megfelelően. Én címke gyanánt a már említett karjelzések neveit használtam.

A minták jelentős része saját jelzés. A kutatás eredményeinek igazolásához azonban még másik három, független szereplőt is bevontam az adathalmaz gyarapítása érdekében. Nekik csak szószavúán elmagyaráztam, hogy nagyjából mit kellene mutatniuk, tehát a minták egymáshoz képest természetes eltéréseket tartalmaznak és így az adathalmaz kellően diverz lett ahhoz, hogy az eredményekből helyes következtetéseket tudjak levonni.

A 4.5 táblázatban látható a minták számossága.

Megjegyzem, hogy a K-meanshez képest azonos körülmények megteremtése céljából a két módszer összehasonlítása során a konfigurációs/tanító adathalmaz elemei megegyeztek, egyedül a számosságuk volt különbség. A pontos értékek a 4.1 és a 4.5 táblázatokból olvashatóak ki. Figyeljük meg, hogy a K-means konfigurációs adathalmaz 135 minta volt, míg itt jelentősen több adattal, minden fajtának a 60%-ával, tehát összesen 26365 darab mintával tanult a modell.

Jelzés neve	Minták száma
balderek	6943
balegyenes	4371
balemelt	4247
derekderek	4380
egyenesegyenes	4481
jobbderek	4349
jobbegyenes	4189
jobbemelt	3059
sima	7929
Összesen	43498

4.5. táblázat. A kutatás során használt minták típusa és száma a 4.2 ábrának megfelelően.

4.2.2. Modell architektúra

A következő lépés, hogy meghatározzuk, milyen modellel lehetséges a feladat megoldása. A dolgozatban egy olyan neurális hálózatot használok, amely három lineáris rétegből és ReLU aktivációs rétegekből áll.

A lineáris réteg, más néven teljesen összekötött (sűrű) réteg, a neurális hálózatok alapvető építőeleme. Egy lineáris réteg az előző réteg minden kimenetét súlyozva összegzi, mindegyikhez hozzáadva egy bias (eltolás) tagot, amivel a modell komplexitása nő. Matematikailag egy lineáris réteg a következőképpen írható fel:

$$y = Wx + b$$

- x : az bemeneti vektor (a korábbi réteg kimenete),
- W : a súlymátrix,
- b : az eltolás vektora,
- y : az aktuális réteg kimenete.

Ez a réteg arra szolgál, hogy lineáris transzformációt hajtson végre, ami alapvetően azt jelenti, hogy az adatokat egy magasabb vagy alacsonyabb dimenziós térbe helyezi át, lehetővé téve a modell számára, hogy összetett összefüggéseket is felismerjen a tanító adathalmazban.

Aktivációs függvénynek ReLU-t használtam. Az aktivációs függvény célja, hogy nemlineáris tulajdonságokat adjanak a neurális hálózatoknak. A ReLU (Rectified Linear Unit) az egyik leggyakrabban használt aktivációs függvény, amely a következőképpen

működik:

$$f(x) = \max(0, x)$$

A ReLU függvény egy nemnegatív értékeket változatlanul hagy, míg a negatív értékeket nullára állítja. Ez az egyszerűsége ellenére nagyon hatékonyan segít a gradiens-eltűnés [4] problémájának kezelésében, és gyorsítja a konvergenciát a tanítás során.

Az így összegyűjtött adathalmazt 60-20-20 arányban osztottam fel a három lépés között: tanítás, validáció és tesztelés. Ezekről a következőben lesz szó.

4.2.3. Tanítás

A modell tanítás közben egy gráf súlyozott éleinek súlyait próbálja meg a legmegfelelőbbben beállítani, így minden egyes iterációval egyre jobban ráilleszti a modell kimenetét az elvárt kimenetre. Ebből következik, hogy a tanításhoz jelentősen több adat szükséges, mint a K-means esetében.

A modell tanítása során a tanító adathalmaz elemeit batch-enként adagoljuk a modellbe, majd ezt a folyamatot néhányszor megismételjük. Egy batch egy adatcsoportot jelent, itt 1024-es batch mérettel tanítottam a modellt. Egy ilyen teljes batch-iterációt epochnak nevezünk, tehát egy epochban minden tanító adatot pontosan egyszer lát a modell. A véletlenszerű keverésnek köszönhetően az adatok sorrendje a tanítást nem befolyásolja.

A tanításhoz az Adam (Adaptive Moment Estimation) optimalizáló algoritmust használtam. Ez az algoritmus ötvözi a lendület alapú (momentum) és a RMSprop (Root Mean Square Propagation) algoritmusok előnyeit, így biztosítva gyors és hatékony konvergenciát.

4.2.4. Validációs lépés

A túltanulás elkerülése érdekében szokás validációs adathalmazt is leválasztani, majd a modell teljesítményét a tanítás közben ezen az adathalmazon mérni. Minden epoch után ellenőriztem, hogy a modell mennyire teljesít jól ismeretlen adatokon.

A tanítás során a kilépési maximum teljesítmény küszöbértéke a validációs adathalmazon 99.95% volt. Az első epoch után, amikor a validáció során a modell ezen küszöbnél jobbat ért el, a tanítás megszakadt.

4.2.5. Időtényező a tanítás kapcsán

Ahogy a 4.6 táblázat mérési eredményeiből látszik, a neurális háló tanítása jelentősen több időt és nagyobb erőforrást igényel, mint a K-means centroidok kiszámítása.

Rejtett rétegek mérete	Megvalósult epoch	Tanítás ideje (s)	Eredmény (%)
[64, 128]	50	171,959584	96,56544979
[32, 64]	100	351,0402982	97,80507222
[16, 32]	200	656,224021	98,51017855
[64, 128]	64	223,6670747	98,79449562
[128, 256]	28	88,4792521	98,8741044
[128, 256]	33	110,5910323	99,09018537
[128, 256]	32	109,4232106	99,2721483
[64, 128]	58	200,2347596	99,45411122
[512, 1024]	12	31,30686116	99,52234732
[1024, 512]	8	20,18117857	99,56783805
[256, 512]	22	72,08290553	99,71568293
[256, 512]	20	64,77841473	99,71568293
[512, 1024]	10	25,69633865	99,90901854

4.6. táblázat. A tanítás mért adatai.

A megvalósult epochok száma az az érték, amennyi a validáció miatt valóban megvalósult.

A mérési eredményeket nagyban befolyásolja a futtatókörnyezet teljesítménye. Az értékek egy Nvidia RTX 3060 típusú kártyán kerültek mérésre. A dolgozatban szereplő mérési eredmények egy konkrét számítógépen kerültek mérésre.

A 4.6-as táblázat adatainak az egy epochra jutó tanítás átlagos ideje 3,344 másodperc. Egy adott rejtett réteg méretből azért mértem többet, mert a kezdeti súlyok véletlenszerűen kerülnek inicializálásra, így némi eltérés mutatkozhat a mért értékekben.

4.2.6. Valós idejű felismerés

A K-meanshez hasonlóan zajlik a valós idejű felismerés itt is, azonban a két módszer hatékonyságában mindenféleképpen mutatkoznak jelentős különbségek.

A mérési eredményeket nagyban befolyásolja a futtatókörnyezet teljesítménye. Az értékek egy Nvidia RTX 3060 típusú kártyán kerültek mérésre.

A 4.7-as táblázat adatai azt mutatják, hogy a háló méretének ilyen kis változásai a futásidőt alig befolyásolják.

A modell kimenetén megjelenik a jósolt eredmény minden képkocka feldolgozása után.

Rejtett rétegek mérete	Egy felismerés ideje (s)
[512, 1024]	0,00025965
[512, 1024]	0,000238599
[512, 1024]	0,000240428
[512, 1024]	0,000227653
[256, 512]	0,000227718
[256, 512]	0,000238035
[256, 512]	0,000230923
[128, 256]	0,000214798
[128, 256]	0,000242621
[16, 32]	0,000249478
[16, 32]	0,000237387

4.7. táblázat. A neurális hálózat felismerési ideje.

4.2.7. Eredmény

Rejtett rétegek mérete	Megvalósult epoch	Jó	Összes	Eredmény (%)
[16, 32]	50	4174	8793	47,470
[16, 32]	200	8662	8793	98,510
[32, 64]	100	8600	8793	97,805
[64, 128]	64	8687	8793	98,794
[64, 128]	50	8491	8793	96,565
[64, 128]	58	8745	8793	99,454
[128, 256]	32	8729	8793	99,272
[128, 256]	28	8694	8793	98,874
[128, 256]	33	8713	8793	99,090
[256, 512]	22	8768	8793	99,716
[256, 512]	20	8768	8793	99,716
[512, 1024]	12	8751	8793	99,522
[512, 1024]	10	8785	8793	99,909

4.8. táblázat. A neurális hálózat felismerés eredményei.

A 4.8 táblázatban láthatóak a tesztelés eredményei. A rejtett rétegek mérete jelenti a már ismertett hálózat belső méretét. A tanítás során megvalósult epochok száma annyi, amennyi szükséges volt a kilépéshez szükséges validációs eredményhez.

A tesztadatok az összes minta (4.5 táblázat) közül kerülnek ki, minden jelzés adathalmazának 20%-a. Egy jóslás akkor és csak akkor jó, ha a címkével egyező kimenetet adott rá a modell.

A táblázatban a neurális háló felismerésének hatékonysága látható. Figyeljük meg, hogy az első néhány kis rejtett méret kihasználta a maximális epoch számokat (50,200,100), tehát a validációs halmazon sem teljesített annyira jól, hogy megálljon a

tanítás. Ezzel szemben elég ideig tanítva meggyőző eredményre volt képes a modell, de ez végtelenül idő- és erőforráspazarló megközelítés.

A nagyobb méret már több neuront jelent, így ez magával vonja azt is, hogy a modellnek több lehetősége van, így kevesebb lépésben is képes megfelelően tanulni.

4.3. Statikus felismerő módszerek összehasonlítása

A statikus karjelzések felismerésének két módszerét már ismertettem. A jelenlegi fejezetben módszeresen összehasonlítom a két megoldást, különösen ügyelve az adott módszer előnyeire és hátrányaira.

4.3.1. A tanító adathalmaz mérete, címkézés

Az egyik legszembetűnőbb különbség az adathalmaz elkészítésében mutatkozik.

A K-means esetében néhány konfigurációs adattal is kiváló eredmények érhetőek el, míg a neurális hálós megközelítéshez jelentősen több adat szükséges. További különbség, hogy a K-means nem igényel címkézett adatokat, így az adatok felvétele sokkal könnyebb, mint a neurális háló esetében.

Egy esetleges változtatásnál, például egy másik karjelzésre való átállás esetén a neurális hálót újra kell tanítani, ami rengeteg erőforrást igényel (amely sok futtatókörnyezetben nem is áll rendelkezésre) és sok időt is emészt fel. A K-means estén egy változtatáshoz csak annyit kell módosítani, hogy a konfigurációs halmazhoz néhány új mintát kell adni, majd a centroidokat újra kell számolni. Mivel a számolás nagyon egyszerű, ezért ez bármilyen környezetben gyorsan végrehajtható.

4.3.2. Szakmai felkészültség szükségessége

Úgy vélem, hogy azt is fontos kiemelni, hogy egy neurális háló újratanítása nem tartozik a triviális feladatok közé. Tehát egy egyszerű felhasználó esetén komoly erőfeszítéseket kell abba fektetni, hogy olyan környezetet hozzon létre a fejlesztő, amivel a felhasználó is képes változtatásokat eszközölni a modellben.

A K-means esetében egyetlen számot kell megváltoztatni (K érték), és néhány mintát felvenni. Ezt jelentősen könnyebb végrehajtani egy átlagos felhasználónak is, továbbá felhasználóbarát környezet is könnyebben fejleszthető köré.

4.3.3. Idő és teljesítmény

A tanítás/konfigurálás ideje szintén kiemelendő különbség a két megoldás között.

A K-means centroidszámítása jelentősen gyorsabban fut le, mint a neurális háló tanítása, hiszen a K-means olyan gyorsan számol centroidot, mint amennyi egy epoch végigfuttatása a neurális háló esetében.

De a legfontosabb, hogy a felismerés sebessége is hatalmas különbség. Ahogy az adatokból látszik, a K-means a 9 pózra számított átlagos ideje (4.4 táblázat) egy mintára jelentősen (százas nagyságrend!) gyorsabb, mint a neurális háló egy jóslása (4.7 táblázat).

Ez egyébként a valós próbák alkalmával is szembeűnő különbség volt, tehát a neurális háló jóslatára érezhetően többet kellett várni, mint a K-means eredményére. Ez számtalan helyzetben nagy hátránya lehet a neurális hálós megoldásnak, főleg akkor, ha rövid idő alatt változhat a szituáció (különösen említve a közlekedést).

4.3.4. Hatékonyság

A felismerés pontosságának tárgyalása szintén érdekes szempont. A mért adatokból látszik, hogy a két módszer hatékonysága között túl nagy különbség nincs: 4.1 táblázatból látszik, hogy a K-means 99.84%-os pontossággal képes megoldani a feladatot. A neurális háló maximális értéke a 4.8 táblázatból 99.91%.

A neurális háló ezekből következően képes jobban megoldani a feladatot, többek között azért, mert képes általánosítani, és nem egy előre jól meghatározott algoritmust használ a különbségek megállapítására.

4.3.5. Összegzés

A statikus pózok felismerését két, lényegében különböző módszerrel valósítottam meg.

A K-means egy egyszerű és meglepően hatásos módszer a statikus karjelzések felismerésére. Könnyen konfigurálható és gyorsan futtatható megoldás. Úgy vélem, hogy általános alkalmazásokra a K-means megfelelő.

A neurális háló egy nehezebb, bonyolultabb megközelítés. Látható, hogy bizonyos esetekben jobb eredményt képes elérni a K-meansel szemben. Hátránya az, hogy nagyon sok adatot igényel, nehéz és drága tanítani, továbbá a felismerés sebessége is jelentősen alulmúlja a K-means teljesítményét.

Erre az egyszerű problémára a K-means egy tökéletesen elégséges megoldást nyújt, és úgy vélem, hogy nagyon speciális esetektől eltekintve ez a javasolt.

5. fejezet

Dinamikus karjelzések felismerése

A dolgozat korábbi fejezeteiben statikus pózfelismerést valósítottunk meg, tehát minden egyes képkockáról el tudtuk dönteni, hogy melyik kiindulási mintára hasonlít leginkább. Eközben nem fordítottunk figyelmet az adatok időbeliségére, a minták egymásutánosságára.

Az emberi karjelzések egy másik csoportját alkotják azok a jelzések, amelyek mozgásokban is hordoznak jelentést. Ezeket dinamikus jelzéseknek nevezzük.

A dinamikus karjelzések értelmezéséhez rekurrens neurális hálót használok. Ez annyi-
val több az egyszerű hálózatokhoz képest, hogy az adatok nemcsak előre felé, hanem bizonyos körülmények között hátrafelé is tudnak haladni, tehát az adatok egymásutánosságára vonatkozó információ tárolódik a neuronokban a hátrafelé mutató éleknek köszönhetően, így a modellnek a tanítás során kialakul az a képessége, hogy emlékezni tud korábban keresztülhaladt információkra.

5.1. Csúszó ablak megközelítés

A dinamikus jelzések felismerése céljából fontos megállapítani, hogy ebben a problémában egy szekvenciáról szeretnénk eldönteni, hogy mit ábrázol.

A dinamikus karjelzések felismerését úgy végzem el, hogy egy előre beállított mennyiségű, időben egymást követő képkockát küldök a modell bemenetére mindig egygel arrébb kezdve. A tanítás során a tanítómintákat egy, az ablak hosszának megfelelő képkockasorozat alkotta. Fontos ügyelni a képkockák időbeliségének meghagyására.

Különös tekintettel kell lennünk arra, hogy a tanítás során az ablakot minden egyes tesztadat után csak egyetlen képkockával csúsztottam el, tehát az addigi második kép-

kocka képezte a következő tanítóadatban az első képkockát. Ha az ablak teljes hosszával odébb csúsztatnám a szekvencia kezdetét, akkor a tanító adathalmaz diverzitása sokkal jelentősebben befolyásolná a modellt, továbbá a bemeneti képkockák ciklikussága miatt a modell nem lenne képes jól általánosítani. Képzeljünk csak el egy szélső esetet: az ablak hossza pontosan akkora, mint amekkora a mozgás egy periódusa. Világos, hogy ekkor determinisztikusan az adott pozíciók képe mindig egy adott bemeneti neuronra kerülne, amely az általánosítást gátolná.

Mindezeknek megfelelően a valós idejű pózfelismerés is a fentiekkel egyező logikájú csúszó ablakos módszerrel működik, így a tesztelést is ezzel a módszerrel végeztem.

A csúszó ablak megfelelő méretének meghatározását a mérések alapján teszem meg.

5.2. LSTM és GRU

Az első ilyen célra készült megoldás az LSTM (Long-Short Term Memory) volt. Néhány évvel ezelőtt megszületett az LSTM egy továbbfejlesztése, a GRU (Gated Recurrent Unit). Az LSTM három kaput használ: bemeneti kaput, kimeneti kaput és felejtő kaput. Ezek a kapuk segítenek a hálózatnak eldönteni, hogy milyen információkat tartson meg a múltból, mit felejtson el, és mit engedjen át a következő rétegbe. Az LSTM jól teljesít hosszú távú függőségek kezelésében, viszont a tanítása időigényes, ugyanis rengeteg paraméterrel rendelkezik. A GRU ezzel szemben csak két kaput használ: frissítési kaput és visszaállítási kaput. A frissítési kapu egyesíti a bemeneti és felejtő kapu funkcióit az LSTM-ből, így egyszerűbbé téve a hálózatot. A GRU ennek megfelelően kevesebb paramétert tartalmaz, mint az LSTM, és a gyorsabb tanítás mellett általános alkalmazásokra sokkal inkább megfelelő. Ezen általános megfontolás nyomán döntöttem úgy, hogy a felismerési modellt a GRU köré építem fel.

5.3. Adatgyűjtés

A neurális háló betanításához viszonylag nagy mennyiségű adat szükséges. Az adatok begyűjtése az eddigiekhez hasonlóan történik: rögzíték egy videót, majd a videó képkockáin elvégzem az OpenPose kulcspontbecslést, és a normalizált képet eltárolom az adatbázisban.

Az adatokat úgy kell felvenni, hogy a képkockák időbelisége ne változzék, tehát az adatok pontosan egymás után következzenek a létrejövő adatsorozatban.

Az adatokat a 4.2.1 fejezethez hasonlóan itt szintén címkézni szükséges.

Érdemes ügyelni arra, hogy a mozgást folyamatosan végezzük a felvétel során, ne legyenek „üres” részek. A felvételen minimális számú vágást végezzünk azért, hogy a mozgás folyamata ne sérüljön.

Jelen kutatásban négy fő bevonásával készült a mozgásokat tartalmazó adathalmaz. Egy adott mozgás felvétele előtt egyszer, egyetlen oda-vissza mozdulattal mutattam meg az adatgyűjtésben részt vevő alanyoknak a kívánt dinamikus karjelzést. Ez azért fontos, mert egyetlen mozdulatból nem lehet pontosan megállapítani a kívánt mozgást, így a saját intuíciójukra is szükség volt a mozgás logikus megalkotásában. Kiemelem, hogy egyetlen oda-vissza mozgásból az ismétlések közötti idő (tehát a mozgás sebessége) szintén nem állapítható meg, így ez is a résztvevők saját belátásán múlt. A modell jelentősen nagyobb mértékben lesz képes általánosítani akkor, ha a felvételen mozgás sebessége változó (itt megjegyezve azt a tényt, hogy a felhasználók is egyéntől függően fogják a mozgás sebességét meghatározni.)

A mozgás sebességével a 5.13 fejezetben foglalkozom részletesebben.

A videóból úgy nyertem ki az adatokat, hogy minden normalizált képkockát eltároltam, és a fájl nevében jelöltem a főbb ismérveket, például: *jobbraKoroz_00000000024* - tehát a karjelzés neve utáni szám jelzi, hogy a videó hányadik képkockája ez. Világos, hogy így a vezető nullák segítségével a lexikografikus sorrend megegyezik az időbeni sorrenddel.

A kutatás során felhasznált adatok számossága a 5.1 táblázatban látható. A jelzések nevei a mentési fájlnevekkel egyezően szerepelnek.

Jelzés neve	Minták száma
balraKoroz	2325
jobbraKoroz	4642
simaAllas	2758
balMeneszt	7084
jobbMeneszt	8058
felhivasAthaladasra	7724
balkezJobbraKoroz	2974
balkezBalraKoroz	3770
balEgyenesenKitartvaAllo	1782
Összesen	41117

5.1. táblázat. A kutatás során használt minták típusa és száma.

A Body-25 alapján csak a 8 felsőtesti pontot veszem figyelembe, hiszen ezt már megmutattuk, hogy elégséges a jelzések felismeréséhez. A használt jelzésekből 9 féle van:

- Jobb kézzel balra körzés (5.1a. ábra)
- Jobb kézzel jobbra körzés (5.1b. ábra)
- Sima állás(5.1c. ábra)
- Bal kéz fel-le (menesztés)(5.1d. ábra)
- Jobb kéz fel-le (menesztés)(5.1e. ábra)
- Bal kéz egyenesen kitartva és jobb kézzel integetés (felhívás az áthaladásra)(5.1f. ábra)
- Bal kéz jobbra körzés(5.1g. ábra)
- Bal kéz balra körzés(5.1h. ábra)
- Bal kéz egyenesen kitartva és jobb kéz nyugalomban(5.1i. ábra)

Vegyük észre, hogy ezen olyan dinamikus jelzések, amelyeknél nem elégséges csak statikus jelzéseket felismerni, hiszen a mozgások irányának is szerepe van, továbbá nem mindegy, hogy egy kéz nyugalomban van vagy mozog.

5.4. Jelzések

A 5.1 ábrán a felhasznált dinamikus jelzések vannak. Minden jelzésből 5 képkockát mutatok be, amelyek a jelzés kezdetétől számított 70 képkocka közül vannak, egyenletesen kiválasztva. A jelzések nevei a képek alatt szerepelnek, összhangban a 5.3 pontban felsoroltakkal.

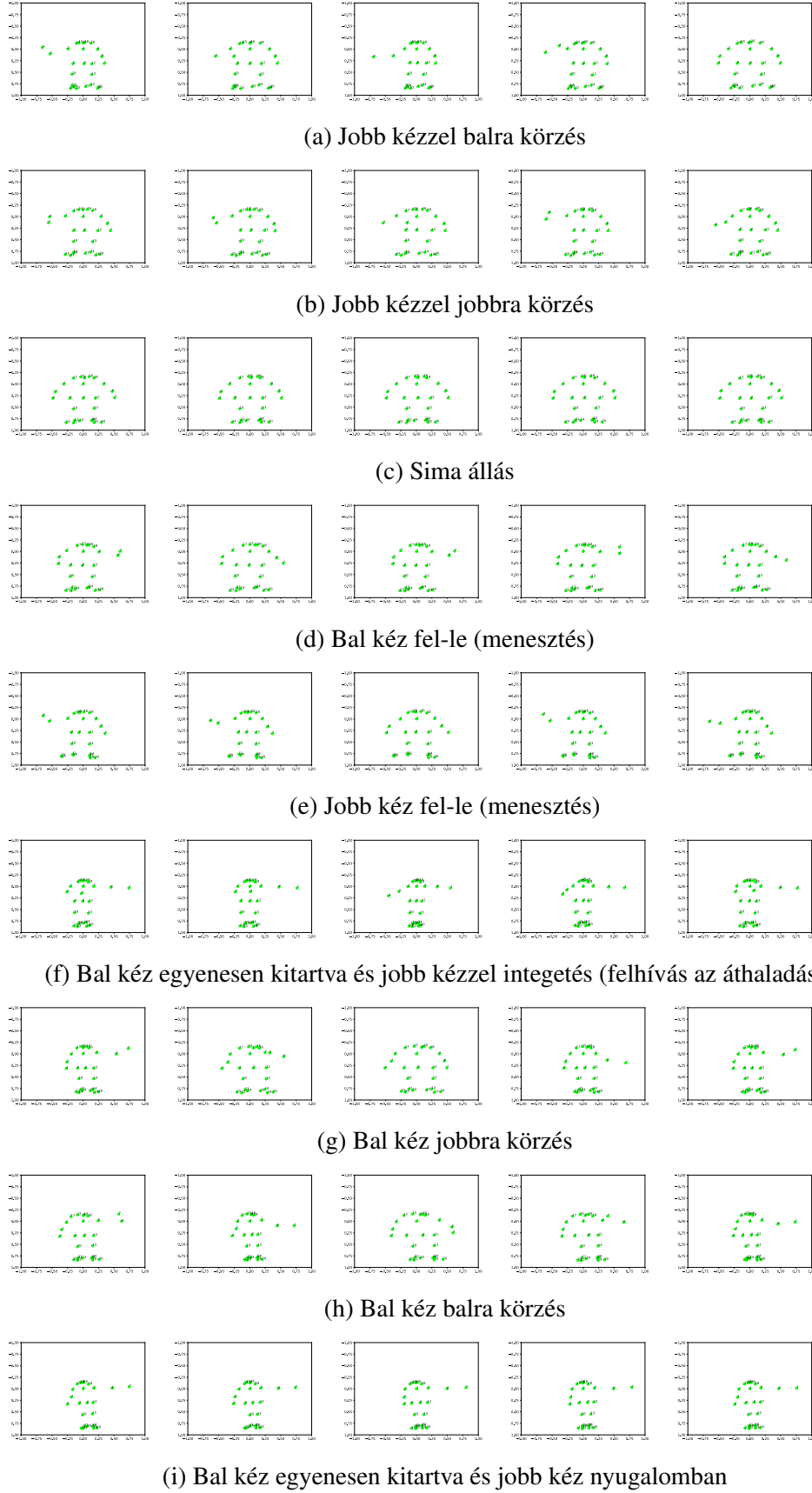
5.5. Modell architektúra

A következőkben elmondom, hogy milyen komponensekből áll fel a modellem.

A használt modell a következő:

1. **Lineáris** - Bemenet: 16 (figyelembe vett pontok száma), Kimenet: 4096
2. **Lineáris** - Bemenet: 4096, Kimenet: 2048
3. **GRU** - Bemenet: 2048, Kimenet: 1024
4. **Lineáris** - Bemenet: 1024, Kimenet: 256
5. **Lineáris** - Bemenet: 256, Kimenet: 9 (jelzések száma)

A 8×2 darab bemeneti számot (amely 8 pont x és y koordinátája, normalizáció után) az első rétegre küldöm. Az első kettő lineáris réteg a klasszifikáció elejét végzi el: gya-



5.1. ábra. A felhasznált dinamikus jelzések. Minden sor egy-egy jelzésből mutat 5 jellemző képkockát. A jelzések nevei a képek alatt szerepelnek.

korlatilag felcímkézi az adatokat (ahogy a statikus felismerés tette, csak sokkal több fiktív halmazra) és ezeket a címkéket küldi tovább a GRU-nak. A GRU megpróbálja felismerni a sorozatosságot a kapott adatokban, majd az utolsó lineáris rétegek osztályozzák a GRU kimenetét.

5.6. Tanítás

A modell tanítását olyan adathalmazon végeztem, amelyet három részre daraboltam a klasszikus 60-20-20 arányban: rendre tanító, validációs és tesztelő adatokra.

A tanítás során 2048-as batch méretet használtam, ugyanis a feladat bonyolultságának megfelelően minél nagyobb batcheket alkalmazunk, annál több adatot lát egyetlen lépésben a modell, így az általánosítási képessége nő és emellett az optimalizáló algoritmus átlagosan jobb irányba lép. További szempont volt, hogy csak ekkora fért rá a GPU-mra.

5.7. Validáció

A validáció a 4.2.4 fejezetben már említett módon működik itt is, tehát egy előre beállított küszöbértéknél a validációs halmazon jobban teljesítő modell tanítása megszakad, elkerülve a túltanulást.

A tanításhoz minden mérésnél 99%-os validációs küszöböt alkalmaztam azzal a megköttéssel, hogy a modellnek az átlagos hibája a tanító halmazon (average loss) legfeljebb 0.01.

5.8. Időtényező a tanítás kapcsán

Figyeljük meg a 5.2 táblázat adatait! Látható, hogy a tanítás ideje jelentősen több a statikus módszernél és érdemes hangsúlyozni, hogy a tanítás kiemelten erőforrásigényes művelet. Ez látható a statikus felismerés adataiból is (4.6 táblázat).¹

¹A mérési eredményeket nagyban befolyásolja a futtatókörnyezet teljesítménye. Az értékek egy Nvidia RTX 3060 típusú kártyán kerültek mérésre.

Megvalósult epoch	Tanítás ideje (s)
42	523
32	395
35	707
36	724
16	452
29	816
33	932
19	617
32	1039
34	1104
27	1200
24	1065
24	1065
25	1329
22	1170
65	3465

5.2. táblázat. Időtényező a tanítás kapcsán. Minden sor egy tanítás időtartamát mutatja a megvalósult epochok függvényében.

5.9. Valós idejű felismerés

Jelen szakaszban bemutatom, hogyan lehet a valós idejű felismerést megvalósítani dinamikus jelzések esetén. Fontos megállapítani, hogy a modell szekvenciákból képes megmondani a dinamikus karjelzés jelentését.

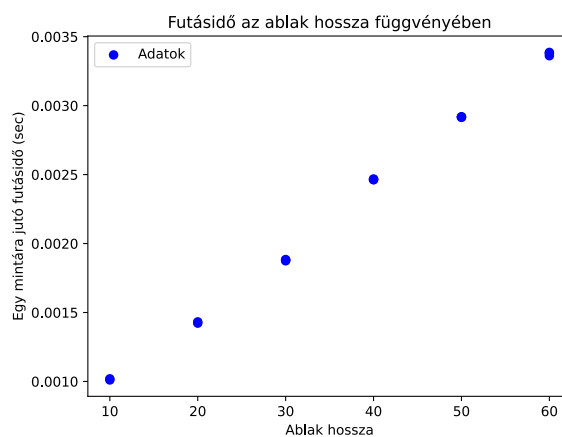
Vegyünk egy csúszó ablak hosszú üres sort! Ezt kezdetben töltjük fel a csúszó ablak méretének megfelelő számú képkockával, hogy az első felismerést le tudjuk futtatni. A felismerés megtörténte után a sor első elemét (tehát a legrégebbi képet) töröljük a sorból, majd a végére szúrjuk be az aktuális képkockát! Majd ezt alkalmazzuk a program futásáig.

A 5.3 táblázatban csúszó ablak mérete szerint vizsgáltam az egy mintára jutó jóslási időt, több mérés alapján. Észre lehet venni, hogy a futásidő egészen kevés, tehát a valós idejű felismerés hatékonyságát nagy mértékben nem befolyásolja. Az adatok lineárisan változnak, ezt az 5.2 ábra mutatja.

A modell minden szekvenciára megjósolja, hogy mi az aktuális dinamikus karjelzés. Világos, hogy így egy hirtelen váltás nem okoz azonnali hatást a kimenetre (hiszen a régi képkockáknak el kell fogyniuk a sorból), továbbá tökéletes modell hiányában előfordulhat téves jóslás is. Az imént említett problémák egy átlagos alkalmazásnál nem biztos, hogy gondot okoznak, azonban biztonságkritikus rendszerekben hatalmas zavart okozhat az, ha a kimenet gyorsan és megbízhatatlanul váltakozik.

Csúszó ablak mérete	Egy kimenet előállításának ideje (s)
10	0,001019
10	0,001011
20	0,001423
20	0,001432
30	0,001880
30	0,001875
30	0,001884
40	0,002464
40	0,002463
40	0,002468
50	0,002918
50	0,002919
50	0,002915
60	0,003380
60	0,003386
60	0,003362
Átlag	0,002300

5.3. táblázat. Egy mintára jutó átlagos jóslási futásidő.



5.2. ábra. Átlagos futásidő az ablak hosszának függvényében.

5.10. Tesztelés

A modell tesztelését a tanítás során alkalmazott sliding window hosszával végeztem. Minden jelzés darabszámának(5.1 táblázat) 20%-a volt tesztadatként felhasználva, tehát a modellt ezeken teszteltem. Egy tesztadatra a kimenet akkor és csak akkor volt helyes, ha a címkével egyező volt a kimenet.

5.11. Eredmény szabadon választott sebességre

Ebben a szakaszban ismertetem a modell eredményeit olyan adatokra, amely dinamikus jelzéseket a felkért alanyok a saját belátásuknak megfelelő sebességgel adtak ki.

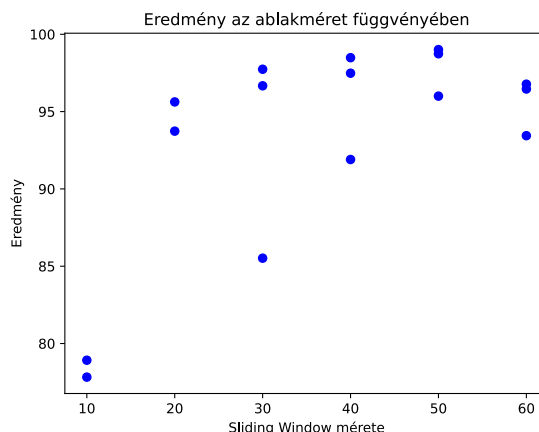
Ablak hossza	Megvalósult epoch	Jó	Összes	Eredmény (%)
10	42	6332	8136	77,827
10	32	6421	8136	78,921
20	35	7542	8046	93,736
20	36	7694	8046	95,625
30	16	6804	7956	85,520
30	29	7691	7956	96,669
30	33	7776	7956	97,738
40	19	7229	7866	91,902
40	32	7668	7866	97,483
40	34	7747	7866	98,487
50	27	7699	7776	99,010
50	24	7465	7776	96,001
50	24	7678	7776	98,740
60	25	7182	7686	93,443
60	22	7414	7686	96,461
60	65	7438	7686	96,773

5.4. táblázat. GRU eredmények szabadon választott sebességre.

A mért eredmények az 5.4 táblázatból látszanak. Minden ablakmérettel több mérést végeztem azért, hogy az eredményekből megbízható következtetéseket lehessen levonni. Az ablakméret jelentősen befolyásolja a tanítás folyamatát és az eredményt is. Ahogy az 5.3 ábrán látszik, a legjobb eredményt az 50 hosszú ablakméret esetén érte el a modell. A 10 hosszú ablak nagyon kevés információt tartalmazott a felismeréshez, a 60-as méret pedig már túl hosszúnak bizonyult.

Az eredmények értelmezése kulcskérdés. Látható az 5.4 táblázatból, hogy a modell 97 – 98%-os eredményre képes megfelelő ablakméret esetén. A tesztelés során azt mértem, hogy a tesztelő minták mekkora részét képes jól felismerni a modell. Tehát ez annyit jelent, hogy ezeket a dinamikus jelzéseket egyetlen, adott hosszú képsorozatból 97%-os eredménnyel felismerte.

A folyamat közben azonban (különösen a valós idejű alkalmazásokat nézve) lehetnek félreértelmezett jelzések is, továbbá hirtelen megváltozhat a kiadott jelzés. (Vegyünk például egy indító jelzést, majd valamiért hirtelen a megállásra felhívó jelzés kerül kiadásra.)



5.3. ábra. Eredmények a csúszó ablak méretének függvényében.

Az imént említett helyzeteket jó volna, ha a felismerő megoldás tudná kezelni. Emiatt a végső kimenet kiszámításához még kifinomultabb módszert lehet keresni: az eredményt több jóslás egyesítéséből számoljuk ki.

5.12. Kimenet több jóslásból

Ahogy láttuk az 5.4 táblázatból, a felismerő modell egyetlen, ablakhossznyi adatból is egész jó arányban tudja felismerni a karjelzést. Jelen szakaszban azt részletezem, hogy ez a felismerés több bemeneti minta felismerésének összekombinálásával hogyan javítható.

A modell kimenete minden esetben egy, a jelzést azonosító nemnegatív egész szám úgy, hogy egy számhoz pontosan egy dinamikus karjelzés tartozik.

5.12.1. Naív megközelítés

Az első, talán természetes megközelítés az lenne, hogy vegyünk N darab jóslást, majd a modell kimenetén megjelenő azonosító számokból vonjunk átlagot. Ezen megközelítés akkor működhet, ha az N darab kimenet közül szinte mindegyik a kívánt kimenet és az eltérők nem húzzák el rossz irányba az átlagot.

Vegyük észre, hogy ez az első megközelítés a legtöbb esetben hibás eredményre jutna. Az átlagolás miatt előfordulhat az, hogy mondjuk 9 jelzés esetén $N/2$ esetben 0, a maradék esetben 8 a kimenet. Ekkor az átlag valahol 4 környékén lenne, de az ennek megfelelő jelzést egyszer sem adta ki a jelzést adó személy.

5.12.2. Egy helyes megközelítés

Második megközelítésként vegyük azt, hogy megszámloljuk egy N hosszú, kezdetben nullával feltöltött D (darabszámokat tartalmazó) tömbben azt, hogy az adott azonosító hányszor került az N vizsgált lépésben a kimenetre. Ezen megközelítés végső kimenete a maximális darabszámú azonosító legyen.

Belátható, hogy ez a megközelítés csak olyan eredményt ad, amelyet valóban kiadott a jelzést adó személy, továbbá a végső kimenet több jóslás alapján még pontosítható. Azonban még nem oldottuk meg azt, hogy a hirtelen váltásokat tudja kezelni a megoldás, hiszen az első $N/2$ esetben 0 (indulás), majd a maradék esetben 8 (megállás) azonosítójú jelzéseknél nem figyeli azt, hogy a vizsgált N eset második felében kiadott jelzések fontosabbak, mint a korábbiak.

Ezt is könnyedén lehet orvosolni: a kiadott jelzések darabszámát tartalmazó D tömböt állítsuk elő úgy, hogy minden felismert jelzés esetén ne 1-et adjunk a darabszámok tömbjének megfelelő helyére, hanem egy $S : N \rightarrow R$ függvény értéket, ahol $S(\text{hányadik})$ megadja az N vizsgált jelzés közül azt, hogy a k -adik milyen súllyal kerüljön figyelembe vételre.

A súlyozó függvény igazából bármi lehet, de fontos, hogy szigorúan monoton növekvő legyen, de ne legyen nagyon meredek, hisz ezesetben az utolsó kimenet lesz mindig a győztes, tehát megint elromlik a kiegyensúlyozott kiválasztás.

Néhány példa jó S súlyozó függvényre:

- $S(h) = \frac{1}{2}h$
- $S(h) = \frac{1}{8}h^2$

ahol h az egyetlen paraméter azt jelzi, hogy az éppen feldolgozásra kerülő N hosszú sorozat hányadik kimenete ez.

A D tömb itt már a súlyozott értékek tömbje. Az ezt elkészítő algoritmus struktogramja a 5.4 ábrán látható. Az algoritmus vár egy legalább K hosszú bemeneti *Kepek* vektort, amelynél minden K -ra a $Kepek[K]$ az egy csúszó ablak méretű, normalizált képkockákat tartalmazó vektor.

Biztonságkritikus kérdés lehet az is, hogy a rendszer nem jósol valótlan, tehát például egy extermális pózt nem sorol be sehova és egy nem egyértelmű jelzésre úgyszintén nem mond semmit. Ezt a kritériumot az elágazás oldja meg 5.4 ábrán látható programban. Ha a darabszám kisebb, mint a tömb összegének fele, akkor -1 értékkel tér vissza a program, ami azt jelenti, hogy a felismerésben nem eléggé biztos és inkább nem mond semmit. Más

<i>Kimenet több jóslásból</i>	
i in range(K):	
pred = model(torch.tensor(Kepek[i]).to(device))	
predInt = torch.argmax(pred).item()	
D[predInt] += S(i)	
veglegesKimenetId = D.index(max(D))	
D[veglegesKimenetId] < sum(D)/2	
<i>true</i>	<i>false</i>
return -1	return veglegesKimenetId

5.4. ábra. Egy bemenetre a D tömb elkészítése, majd a végleges kimenet kiszámolása.

esetben pedig visszaadja a legvalószínűbb jelzés azonosítóját.

Világos, hogy így minden kritériumot teljesít a módszer, tehát nem jelennek meg sosem mutatott értékek a kimeneten, és a legvalószínűbb – valóban kiadott jelzés – azonosítója fog megjelenni úgy, hogy az időben távolabbi jelzések fontossága alacsonyabb, mint az időben közelebbi jelzéseké.

5.12.3. Eredmények

Jelen fejezetben a már bemutatott 9 dinamikus karjelzést fogom használni a modell teljesítményének megméréséhez. A tanítás után lefuttatom a minden tesztre rákérdező egyszerű tesztelő programot, amely működését már a 5.10 részben ismerttettem. Emellett végrehajtok egy új mérést is, amelyben a 5.4 stuktogramon ismerttetett módszerrel összeítem a pontosan K darab eredményt, majd ezekből előállítom a végleges kimenetet.

Eddig a dolgozatban ismerttetett összes tesztelés esetén egy tesztesetre adott kimenet akkor és csak akkor számított helyesnek, ha a címkével egyező volt az eredmény. Most ettől eltérően az is helyes válasznak számít, ha a rendszer a -1 extermális értékkel tér vissza, tehát bizonytalanság miatt inkább nem mond semmit.

A 5.5 és a 5.6 táblázatokban az látható, hogy milyen eredményt ér el egy adott modell

K érték	S(h) függvény	Eredmény (%)
Egyetlen jóslás	-	94,76
3	1	99,91
3	h	99,89
3	$\frac{1}{8}h^2$	98,03
5	1	99,91
5	h	99,92
5	$\frac{1}{8}h^2$	99,9

5.5. táblázat. Több jóslás összesítésének eredménye 40 hosszú csúszó ablak esetén.

K érték	S(h) függvény	Eredmény (%)
Egyetlen jóslás	-	98,34
3	1	99,83
3	h	99,81
3	$\frac{1}{8}h^2$	99,79
5	1	99,82
5	h	99,83
5	$\frac{1}{8}h^2$	99,84

5.6. táblázat. Több jóslás összesítésének eredménye 50 hosszú csúszó ablak esetén.

az egy adatból való jóslás során, majd az is látszik, hogy a különféle K értékekre és S függvényekre mennyivel változik ez az érték.

Ahogy már korábban láttuk 5.4 táblázatból, a 40 hosszú csúszó ablakos felismerés önmagában nem produkált kimagasló eredményt a tesztek során. Azonban most megmutattam, hogy a fejezetben említett módszerrel a kezdeti 94,76%-os eredményt 99%-ra sikerült növelni. Az 50 hosszú csúszó ablakos felismerés esetén a kezdeti 98,34%-os eredményt 99,84%-ra sikerült növelni.

Az eredmények alapján látható, hogy a több jóslás egyesítésével a modell teljesítménye nagy mértékben javítható.

5.13. Felismerés megváltoztatott sebességű jelzésekre

Érdekes felvetés lehet az is, ha a kiadott dinamikus jelzések sebessége eltér a tanító adatok sebességétől.

Ezzel kapcsolatban is végeztem tesztek azert, hogy ilyen helyzetekre is meghatározzam a modell teljesítményét.

A modellek tanítása a már ismertetett módon történt, minden esetben a normál sebességű adathalmazon. Emiatt gyorsított/lassított mintákat nem látott a modell a tanítási fázisban.

A betanított modelleken három, kizárólag egyetlen bemeneti adatból dolgozó tesztet futtattam le: a normál sebességűt (tehát a tanító adatokkal egyező sebességűt), a normál sebességűt kétszeres gyorsítással és a normál sebességűt kétszeres lassítással. A gyorsított tesztadatok úgy készültek, hogy a tesztadatok minden második képkockáját használtam fel. A lassított tesztadatoknál pedig minden képkockát még egyszer megismételtem.

Sliding window	Sebesség	Eredmény (%)
40	Normál	95,08
40	Gyorsított	86,39
40	Lassított	88
50	Normál	98,15
50	Gyorsított	97,4
50	Lassított	91,1

5.7. táblázat. Megváltoztatott sebességű dinamikus jelzések felismerésének eredményei.

A 5.7 táblázatban a mérés eredményei láthatóak. Az eredmények a várakozásoknak megfelelően lecsökkentek a normál sebességű jelzések értékeihez képest, hiszen a modell nem volt ilyen feladatra külön betanítva. Ettől függetlenül látható az, hogy az általánosító képesség nagy mértékben kialakult (tehát a tanítás jó volt), így az eredmények még mindig azt mutatják, hogy ez egy használható modell.

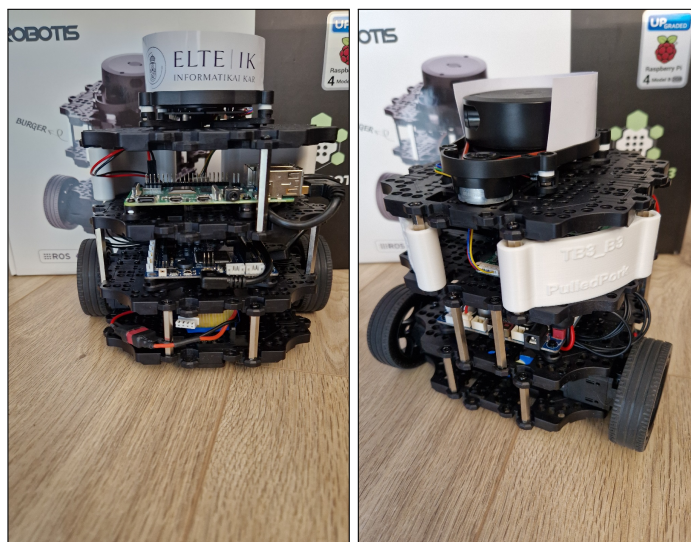
Fontos megjegyezni, hogy a valós világban nagyon ritka az, hogy egy dinamikus karjelzést valaki az átlagos sebesség kétszeresével vagy felével ad ki, hiszen ez emberi léptékkal egy nagy mértékű eltérés. Ennek megfelelően (kisebb mértékű változtatás esetén) a modell teljesítménye valószínűleg a normál eredményekhez közelít, így a leírt megoldás kielégítő a felvetett probléma megoldására.

6. fejezet

Interaktív robotvezérlés karjelzésekkel

Személy szerint szerettem volna mindjárt a vizsgált módszerek alkalmazását egy valós problémára is bemutatni.

Az egyetemi tanszéken rendelkezünk néhány *TurtleBot 3 Burger* típusú, oktatási célra készült robottal. A kutatáshoz kapcsolódóan felmerült az az ötlet, hogy a robotot emberi karjelzések alapján irányítsuk. A jelzés egy egyszerű kézmozdulat, amelyet a vezérlő számítógép kamerája felismer, feldolgoz és a robotnak továbbít. A robot a kapott jelzésnek megfelelően mozog.



6.1. ábra. TurtleBot 3 Burger

A munkám során az imént felvetett ötletet megvalósítottam és a demó néhányszor bemutatásra is került az elmúlt időszakban. Köszönettel tartozom a témavezetőmnek a lehetőségekért.

6.1. Komponensek

6.1.1. Motorok

A roboton két Dynamixel (*XL430-W250-T*) motor található, amelyeket egy erre készült vezérlővel lehet irányítani. A motorok irányítása egy szabványos protokollon keresztül történik, ahol kommunikációt a gyártó által előre megírt könyvtárak biztosítják.

6.1.2. OpenCR Board

A motorokat irányító vezérlő az OpenCR1.0 nevet viseli. Ez egy ARM alapú mikrokontroller, amelyet például az Aruino IDE programmal egyszerű vezetékes kapcsolattal (USB-n) lehet programozni.

6.1.3. Raspberry PI

A roboton található még egy vezérlő számítógép: egy Raspberry PI. Ennek fontos jellemzője, hogy vezeték nélküli kapcsolatot (WiFi) is lehet vele létesíteni, így a robot vezérlése vezeték nélkül, hálózati kommunikációval valósul meg.

A Raspberry számítógép soros porton kommunikál az OpenCR Board-al.

6.2. Megközelítés

A robot vezérléséhez szükséges a karjelzések felismerése. A felismerés a vezérlő számítógépen történik, a kamera képének feldolgozásával.

A karjelzések felismerése K-means segítségével történik. Ez azért jó, mert láttuk, hogy a K-means is képes pontos eredményt adni, de a sebessége jelentősen felülmúlja a neurális hálós megoldást. A robot vezérlését szerettem volna minél precízebben megoldani, így a karjelzések felismerésének gyorsasága fontos szempont.

A felismerő számítógép egyben hálózati kapcsolódási pont is, tehát egy saját vezeték nélküli hálózatot ad, amelyre a robot Raspberry PI-ja felkapcsolódik. A robotra az adatok egy PHP híváson keresztül jutnak el: a vezérlő számítógép egy kiszámolt képkocka után meghívja a robot IP címén elérhető PHP programot, amely a kapott felismerési eredményt egy lokálisan tárolt szöveges állományba írja.

Majd a Raspberry bizonyos időközönként elküldi a fájl tartalmát soros porton keresztül az OpenCR Board-nak, ami a jelnek megfelelően irányítja a motorokat. A vezérlés megvalósult.

6.3. Használt karjelzések

A robot vezérléséhez a már említett 9 karjelzés közül választottam ki 4-et. A 4.2 ábrán látható jelzések közül az a), b), c) és d), g) jelzéseket használtam, amelyek jelentése a következő:

- a) - megállj
- b) - balra fordulj
- c) - jobbra fordulj
- d) - előre
- g) - hátra

6.4. Megvalósítás

A Raspberryre való operációs rendszert telepítettem, majd installáltam a szükséges komponenseket: Apache servert használtam és egy saját PHP programot helyeztem el a publikus könyvtárban. A roboton fut egy "kiíró" program is, amely az OpenCR Board soros portjára írja az aktuális utasítás értékét. Ez a program szintén saját fejlesztés.

Az OpenCR-en futó programot szintén én készítettem. Ez azt a feladatot látja el, hogy a soros porton kapott adatoknak megfelelően irányítja a motorokat. A kommunikációban egész számokat (a soros kommunikáció szempontjából egyedüli karaktereket) használok: 0,1,2,3,4 - a kívánt műveletnek megfelelően.

A projekt működése egy videóban is megtekinthető: https://baginf.hu/TDK_robot.mp4.

6.5. Értékelés

Úgy vélem, hogy az imént bemutatott demó egy tökéletes példája annak, hogy a cikkben vázolt megoldásokat milyen valós feladatokban lehet használni. Azt is megmutattam, hogy a tárgyalt módszerek valóban működőképesek és kielégítik egy átlagos felhasználó

igényeit. Boldog vagyok, hogy számtalan embernek okozhattam örömet ezzel a játék(os) robottal.

7. fejezet

Összefoglalás

A dolgozatban a statikus és dinamikus karjelzések felismerésének két különböző módszerét vizsgáltam meg. A statikus jelzések felismerésének első módszere a K-means algoritmussal végzett klaszterezés volt. Ez egy egyszerű, mégis meglepően hatékony módszer a statikus karjelzések felismerésére. A második módszer egy bonyolultabb, neurális hálózatokon alapuló megközelítés, amely bizonyos esetekben jobb eredményt ér el, de maga a módszer nagyobb adatigényű és nehezebben tanítható. A K-means módszer az, amelyiket a gyorsasága és egyszerűsége miatt átlagos alkalmazásokra javaslok.

A dolgozatban dinamikus karjelzések felismerésére is kísérletet tettem. A dinamikus jelzésekhez neurális hálózatokat (például GRU) használtam. A neurális hálózatokkal elért eredmények kiemelkedően magas pontosságot mutattak, hiszen a modell megfelelően választott ablakméret mellett 98-99% körüli pontosságot ért el, és ezt még további módszerekkel tovább lehetett javítani. A felismerés hatékonyságát a jelzések megváltoztatott sebességű mintáira is megvizsgáltam.

A statikus jelzések felismerésére adott módszer működőképességét egy TurtleBot 3 Burger robottal is demonstráltam, hiszen a robot interaktív vezérlése karjelzésekkel történt.

A jövőbeni kutatásokban érdekes lehetne megvizsgálni azt, hogy a dinamikus jelzések esetén milyen módon lehetne a jelzés sebességét felismerni, tehát a gyorsabb/lassabb jelzések között különbséget tenni. Jelen dolgozatban nem tértem ki rá, de valós alkalmazásokhoz elengedhetetlen az, hogy egy bemeneti képről meg lehessen állapítani, hogy melyik az a személy a képen, akinek a jelzéseit értelmezni szeretnénk, így ez is egy hasznos továbbfejlesztési lehetőség.

Köszönetnyilvánítás



Köszönettel tartozom a témavezetőmnek a jó hangulatú konzultációkért és a szakmai támogatásért. Hálás vagyok a mentorálásért Szalay Gergőnek.

A Kulturális és Innovációs Minisztérium ÚNKP-23-6 kódszámú Új Nemzeti Kiválóság Programjának a Nemzeti Kutatási, Fejlesztési és Innovációs Alapból finanszírozott szakmai támogatásával készült.

Irodalomjegyzék

- [1] Wenpeng Yin és tsai. „Comparative Study of CNN and RNN for Natural Language Processing”. *CoRR* abs/1702.01923 (2017). arXiv: 1702 . 01923. URL: <http://arxiv.org/abs/1702.01923>.
- [2] Keiron O’Shea és Ryan Nash. „An Introduction to Convolutional Neural Networks”. *CoRR* abs/1511.08458 (2015). arXiv: 1511 . 08458. URL: <http://arxiv.org/abs/1511.08458>.
- [3] Jian He és tsai. „Visual Recognition of Traffic Police Gestures with Convolutional Pose Machine and Handcrafted Features”. *Neurocomputing* 390 (2019. okt.). DOI: 10.1016/j.neucom.2019.07.103.
- [4] Seol-Hyun Noh. „Analysis of Gradient Vanishing of RNNs and Performance Comparison”. *Information* 12.11 (2021). ISSN: 2078-2489. DOI: 10 . 3390 / info12110442. URL: <https://www.mdpi.com/2078-2489/12/11/442>.
- [5] Eirini Mathe és tsai. „Arm Gesture Recognition using a Convolutional Neural Network”. (2018. szept.), 37–42. old. DOI: 10.1109/SMAP.2018.8501886.
- [6] Zhe Cao és tsai. *OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*. 2019. arXiv: 1812.08008 [cs.CV].

Ábrák jegyzéke

3.1. A felismerés bemutatása folyamatábrán.	7
3.2. OpenPose felismerés Body-25 modell alapján	9
3.3. A kulcspontbecslés és a normalizálás utáni adatok 1x1 méretű diagramja, a Body-25 modell sorszámaival ellátva.	10
3.4. Normalizálás algoritmusa. A bemenet egy <i>pontok</i> nevű kételemű vektoro- kat tartalmazó vektor úgy, hogy $\text{pontok}[i][0]$ és $\text{pontok}[i][1]$ rendre az x és y koordináták.	11
4.1. A centroidok kiszámításának algoritmusa. A <i>mintak</i> vektorban tároljuk a konfigurációs adatokat és a <i>centroidok</i> vektorban lesznek a kiszámolt centroidok.	19
4.2. A 9 különböző karjelzés, amelyet a kutatásban használtam.	20
4.3. Teszteredmények 9 jelzésre.	20
5.1. A felhasznált dinamikus jelzések. Minden sor egy-egy jelzésből mutat 5 jellemző képkockát. A jelzések nevei a képek alatt szerepelnek.	33
5.2. Átlagos futásidő az ablak hosszának függvényében.	36
5.3. Eredmények a csúszó ablak méretének függvényében.	38
5.4. Egy bemenetre a D tömb elkészítése, majd a végleges kimenet kiszámolása.	40
6.1. TurtleBot 3 Burger	43

Táblázatok jegyzéke

4.1. K-means speciális súlyokkal. Az alsó eredmény a súlyozott.	15
4.2. Súlyok táblázata.	16
4.3. Súlyozó vektorok tesztje.	17
4.4. Az egy mintára jutó jóslás idejének mérése.	18
4.5. A kutatás során használt minták típusa és száma a 4.2 ábrának megfelelően.	22
4.6. A tanítás mért adatai.	24
4.7. A neurális hálózat felismerési ideje.	25
4.8. A neurális hálózat felismerés eredményei.	25
5.1. A kutatás során használt minták típusa és száma.	31
5.2. Időtényező a tanítás kapcsán. Minden sor egy tanítás időtartamát mutatja a megvalósult epochok függvényében.	35
5.3. Egy mintára jutó átlagos jóslási futásidő.	36
5.4. GRU eredmények szabadon választott sebességre.	37
5.5. Több jóslás összesítésének eredménye 40 hosszú csúszó ablak esetén. . .	41
5.6. Több jóslás összesítésének eredménye 50 hosszú csúszó ablak esetén. . .	41
5.7. Megváltoztatott sebességű dinamikus jelzések felismerésének eredményei.	42