ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Pocsai Norbert János**
2025	Hallgató törzskönyvi száma:	**T009276/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Pocsai Norbert János**
Törzskönyvi száma: T009276/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Gesztusfelismerés élő videóból
Gesture Recognition from Live Video Feed

Intézményi konzulens: Dr. Tusor Balázs
Külső konzulens:

Beadási határidő: 2024. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Felhő szolgáltatási technológiák és IT biztonság specializáció

A feladat

A feladat egy olyan rendszer kialakítása, mely képes neurális hálózat segítségével különböző kézmozdulatokat és mozdulat sorokat azonosítani, lehetőség szerint valós időben. A programnak képesnek kell lennie különböző kézmozdulatok felismerésére és szükség szerint új gesztusok betanításra is lehetőséget kell adnia. Mindezt neurális háló segítségével valósítja meg. A kialakított rendszer működőképessége egy saját készítésű játék környezetben kerül vizsgálatra, különböző utasítások és események vezérlésére használva.

A dolgozatnak tartalmaznia kell:

- bevezetést a gesztusfelismerés témájában,
- már létező megoldások vizsgálatát és implementálását,
- a neurális háló betanításának folyamatát és részletes leírását,
- Cross-platform opciók felmérését,
- tesztelés folyamatát és eredményeit,
- következtetést.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

Ph.

.....
Dr. habil. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2026. december 15.**
(ÓE HKR 54.§ (10) bekezdés szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....	.....
külső konzulens	intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2024. 12. 14.

Lacsa Norbert
hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: **Pocsai Norbert János** .....

Neptun Kód: [redacted]

Tagozat: **Mérnökinformatikus BSc**

Telefon: [redacted]

Levelezési cím (pl.: lakcím):

Szakdolgozat / Diplomamunka[1] címe magyarul:

Gesztusfelismerés élő videóból .....

Szakdolgozat / Diplomamunka[2] címe angolul:

Gesture Recognition from Live Video Feed .....

Intézményi konzulens: **Dr. Tusor Balázs**.....

Külső konzulens: .....

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2024.04.10	Szakdolgozat szerkezetének megbeszélése	Tusor Balázs
2.	2024.04.17	Gépi tanulás alkalmazási területeinek megbeszélése	Tusor Balázs
3.	2024.04.24	Eddigi haladásról alkotott vélemény	Tusor Balázs
4.	2024.05.08	Jövőre vonatkozó tervek megbeszélése	Tusor Balázs

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat / Szakdolgozat I. / Szakdolgozat II. / Projektlabor 1 / Projektlabor 2 / Projektlabor 3 / Záródolgozati projekt / Diplomamunka I / Diplomamunka II / Diplomamunka III / Diplomamunka IV [3] tantárgy követelményét teljesítette, beszámolóra / védésre [4]bocsátható.

A konzulens által javasolt érdemjegy: .....5.....

Budapest, 2024.05.13.....

Tusor Balázs
.....
Intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Pocsai Norbert János

Neptun Kód: [redacted]

Tagozat: Nappali

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Gesztusfelismerés élő videóból

Szakdolgozat / Diplomamunka[2] címe angolul:

Gesture Recognition from Live Video Feed

Intézményi konzulens: Dr. Tusor Balázs

Külső konzulens: .....

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2024.09.17.	A szakdolgozat formai követelményeinek átbeszélése	Tusor Balázs
2.	2024.10.07.	Továbbfejlesztési lehetőségek megbeszélése	Tusor Balázs
3.	2024.10.28.	Haladás átbeszélése	Tusor Balázs
4.	2024.11.26.	Szakdolgozat véglegesítése	Tusor Balázs

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat / Szakdolgozat I. / Szakdolgozat II. / Projektlabor 1 / Projektlabor 2 / Projektlabor 3 / Záródolgozati projekt / Diplomamunka I / Diplomamunka II / Diplomamunka III / Diplomamunka IV [3] tantárgy követelményét teljesítette, beszámolóra / védésre [4]bocsátható.

A konzulens által javasolt érdemjegy: .....

Budapest, 2024. 11. 29.

Tusor Balázs

Intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

TARTALOMJEGYZÉK

ABSZTRAKT

A szakdolgozat célja, hogy átfogó áttekintést nyújtson az élő videóból történő gesztusfelismerés területéről, különös tekintettel a gépi tanulás és a neurális hálók alkalmazására. Az első rész a neurális hálók, különösen a konvolúciós neurális hálók alapvető fogalmait és szerepét tárgyalja a gesztusok valós idejű felismerésében.

Ezután bemutatom az élő videófeldolgozás specifikus technikáit, ideértve a képkockák elemzését és a gesztusok pontos értelmezéséhez szükséges előfeldolgozási lépéseket. A rendszer gyakorlati megvalósításához a TensorFlow és OpenCV keretrendszereket alkalmazom, és részletezem az adatgyűjtés, modellépítés és optimalizálás folyamatait.

Végül az eredmények elemzése tesztkörnyezetekben igazolja a rendszer pontosságát és használhatóságát, valamint feltárom a technológia kihívásait és jövőbeli alkalmazási lehetőségeit az ember-gép interakció, a virtuális valóság és egyéb területeken.

ABSTRACT

This thesis aims to provide a comprehensive overview of real-time gesture recognition from live video, with a particular focus on the application of machine learning and neural networks. The first section discusses the fundamental concepts of neural networks, especially convolutional neural networks, and their role in real-time gesture recognition.

Next, specific techniques for live video processing are presented, including frame analysis and preprocessing steps necessary for accurate gesture interpretation. The practical implementation of the system leverages TensorFlow and OpenCV frameworks, with detailed coverage of data collection, model development, and optimization processes.

Finally, the results are analyzed within test environments, demonstrating the system's accuracy and robustness. The thesis also explores the challenges and future applications of this technology in fields such as human-computer interaction, virtual reality, and other domains.

1 BEVEZETÉS

Az élő videóból történő gesztusfelismerés területe izgalmas fejlesztéseknek ad otthont napjainkban. A technológiai előrelépések, különösen a gépi tanulás és a neurális hálók alkalmazása, lehetővé teszi számunkra, hogy pontosan és hatékonyan értelmezzük az emberi gesztusokat, kinyitva ezzel az utat számos alkalmazási terület előtt, például az ember-gép-interakcióban vagy a virtuális valóság alkalmazásában.

Ebben a szakdolgozatban részletesen bemutatom, hogyan alkalmazhatók a legújabb technológiai módszerek az élő videóban megjelenő gesztusok felismerésére és értelmezésére. Saját megvalósításomban egy olyan rendszer fejlesztésére törekszem, amely képes valós időben, nagy pontossággal és robusztus módon azonosítani különböző kézmozdulatokat, miközben gépi tanulási technikákra, elsősorban neurális hálókra és konvolúciós neurális hálókra támaszkodik. A célom, hogy a rendszer ne csupán a gesztusok felismerését végezze el hatékonyan, hanem annak gyakorlati alkalmazhatóságát is demonstrálja például az ember-gép interakció vagy a játékfejlesztés területén.

A gépi tanulás és a neurális hálók iránti fokozott érdeklődés nem véletlen, ezek az eszközök lehetővé teszik a hagyományos képfeldolgozási technikákat meghaladó teljesítményt azáltal, hogy a mintafelismerés komplex, nem lineáris összefüggéseit képesek megtanulni. A szakdolgozat során bemutatott rendszer a legkorszerűbb gépi tanulási módszereket alkalmazza, hogy kezelje az élő videóval járó kihívásokat, például a változó fényviszonyokat, a háttérzajt vagy a gyors mozgásokat. Ezzel a megközelítéssel nemcsak a gesztusok pontos felismerésére nyílik lehetőség, hanem olyan megoldások kidolgozására is, amelyek valós alkalmazási környezetekben is helytállnak.

A dolgozat további részeiben a következő témaköröket tárgyalom. A 2. fejezetben irodalomkutatás keretében bemutatom a gesztusfelismerés különböző alkalmazási területeit, majd a gépi tanulás alapjait és alkalmazásait részletezem, különös tekintettel a neurális hálókra és konvolúciós neurális hálókra. A 3. fejezet a gesztusfelismerő rendszer gyakorlati implementációját mutatja be, kiemelve a TensorFlow, PyTorch és OpenCV eszköztárát. A 4. fejezetben az adatgyűjtés, előfeldolgozás és a modell felépítése kerül ismertetésre, beleértve a statikus és dinamikus gesztusok feldolgozását. Az 5. fejezet a modell betanításának és teljesítményének részleteit tartalmazza. A 6. fejezetben a rendszer tesztelését és az elért eredmények kiértékelését mutatom be, végül a 7. fejezetben összegzem a dolgozat főbb megállapításait és jövőbeli fejlesztési irányokat jelölök meg.

2 IRODALOMKUTATÁS

2.1 Gesztusfelismerés alkalmazása különböző területeken

A gesztusfelismerő rendszerek számos előnnyel járnak. Könnyen használhatók, intuitívak, és nem igényelnek semmilyen speciális eszközt. Ezen kívül a gesztusfelismerés lehetővé teszi a felhasználók számára, hogy természetes módon kommunikáljanak a gépekkel.

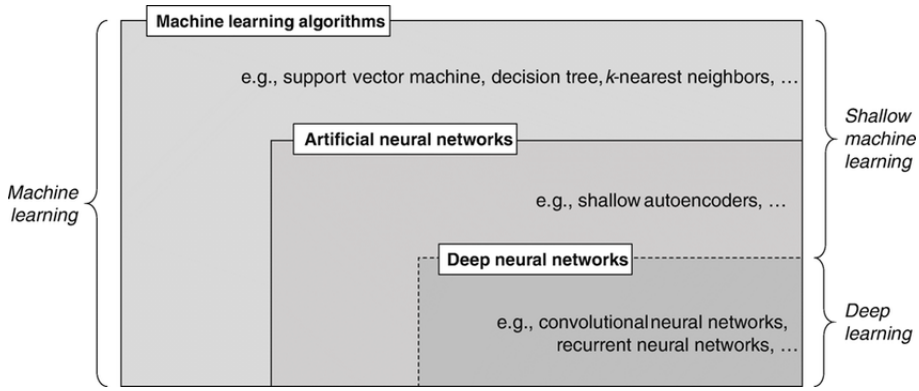
Ezzel együtt jár, hogy számtalan alkalmazási területük van, többek közt:

- A virtuális valóság (VR) [1] egy olyan technológia, amely lehetővé teszi a felhasználók számára, hogy teljesen belemerüljenek és interaktívan részt vegyenek egy számítógép generált környezetben. A gesztusfelismerés nagy jelentőséggel bír a VR területén, mivel lehetővé teszi a felhasználók számára, hogy természetes módon kommunikáljanak és interakciót létesítsenek a virtuális környezettel.
  Az egyik legfontosabb alkalmazási terület a virtuális valóságban a kéz és ujjmozgás nyomon követése. Ez lehetővé teszi a felhasználók számára, hogy a valóságban megszokott módon manipulálják és irányítsák a virtuális tárgyakat. Például, ha valaki egy VR játékban részt vesz, a kézmozgás nyomon követése lehetővé teszi számára, hogy például fegyvert fogjon, objektumokat emeljen vagy dobjon, és közvetlenül reagáljon a környezetében történő eseményekre.
- Játékok: Amikor a számítógépes játékok gesztusait vizsgáljuk. Freeman metódus a játékos kezének vagy testhelyzetének követése az interaktív játéktárgyak, például autók mozgásának és tájolásának vezérléséhez. Konrad és társai [2] gesztusokat használtak avatárok mozgásának irányítására egy virtuális világban, a Play Station 2 pedig bevezette az Eye Toy-t, egy kamerát, amely a kézmozgást követi az interaktív játékokhoz.
- Jelbeszéd: A jelnyelv a kommunikatív gesztusok fontos esete. Mivel a jelnyelvek nagymértékben strukturáltak, nagyon alkalmasak a látási algoritmusok tesztelésére [3]. Ugyanakkor arra is alkalmasak lehetnek, hogy segítsék a fogyatékkal élőket a számítógépekkel való interakcióban. A siketek jelnyelve (pl. az amerikai jelnyelv) olyan példa, amely jelentős figyelmet kapott a gesztusokkal foglalkozó szakirodalomban.[4], [5]

2.2 Gépi tanulás

2.2.1 Áttekintés

A gépi tanulás[6] mélyen gyökeredzik az intelligens technológiai fejlődésben, megnyitva az utat a számítógépek számára, hogy adaptív módon sajátítsák el és fejlesszék teljesítményüket. Ez az a terület, ahol a számítógépek képesek tanulni a múltbeli tapasztalatokból, és ezáltal javítani előrejelző és prediktív képességeiket. A tapasztalat itt széles értelemben fogható fel, legyen az digitalizált adathalmaz vagy interakció a környezettel [7]. Az alábbi 2.1 ábra a gépi tanulás különböző rétegeit és azok közötti kapcsolatokat szemlélteti. Az ábrán látható, hogy a gépi tanulás magában foglalja a hagyományos algoritmusokat (pl. SVM, döntési fák), valamint az artificial neural networks (mesterséges neurális hálózatok) területét, amely tovább osztható sekély és mély tanulásra, ez utóbbit például konvolúciós és rekurrens neurális hálózatok képviselik.



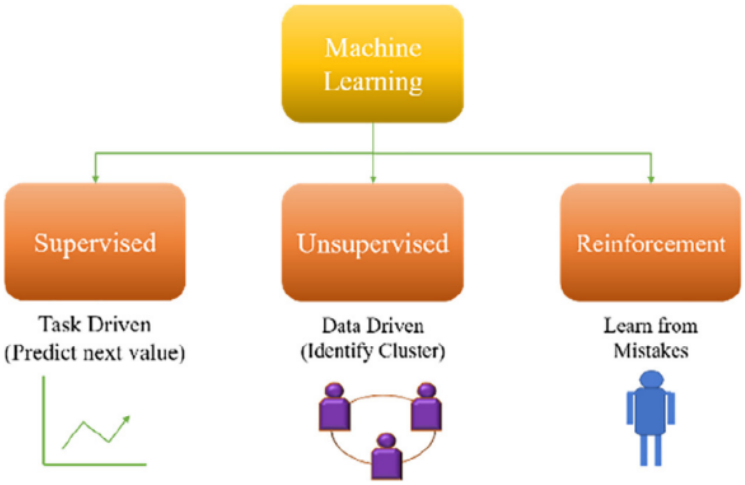
2.1. ábra: A gépi tanulás fogalmainak és osztályainak Venn-diagramja[8]

Gépi tanulás során a számítógépek nem passzívan kapnak utasításokat, hanem aktivitást mutatnak az adatok felfedezése és értelmezése terén, hasonlóan egy felfedezőhöz, aki az ismeretlen földön barangol. Az algoritmusok képesek önállóan felfedezni és tanulni a mintákat az adatokban, de ehhez gyakran szükség van egy "tanárra", ami a felügyelt tanulás fogalmát jelenti. Ebben az esetben minden tanítópéldához tartozik egy megoldás is, amely segíti a gépet a problémák megoldásában.

A gépi tanulás valódi jelentősége abban rejlik, hogy a rendszerek képesek folyamatosan fejlődni és adaptálódni a dinamikusan változó környezeti feltételekhez. A rendszerek nem csupán egy meghatározott feladatra vannak kódolva, hanem képesek saját képességeiket fejleszteni és adaptálódni az új körülményekhez és problémákhoz. Ez a dinamikus és adaptív jelleg teszi a gépi tanulást olyan hatékony és izgalmas területté az intelligens technológiai innovációban.

2.2.2 Megközelítések

Az ML algoritmusok széles körű osztályozása három kategóriában történik, amelyek a felügyelt, a felügyelet nélküli és a megerősítéses tanulás, amint az a 2.2. ábrán látható.



2.2. ábra: Megközelítések a gépi tanulásban [9]

A felügyelt tanulás olyan algoritmust használ, amely külső segítséget igényel. A megadott bemeneti adatbázist képzési és tesztelési adathalmazokra osztják. A kimeneti változót a képzési adatbázisból jósolják vagy osztályozzák. Az algoritmusok megpróbálnak megtanulni néhány alakzatot az adatbázis képzése során, és ezeket a tanult mintákat a tesztelő adatbázisba implementálják, amely eredményeket ad a becslésben.

A felügyelet nélküli tanulás olyan gépi tanulási algoritmus, amely a bemeneti információ néhány jellemzőjét tanulja meg. Új adatbázis biztosítása után a korábban megtanult jellemzőket használja fel az adatok osztályának azonosítására. Leginkább a jellemzők csökkentésére, valamint a klaszterezéshez használják előszeretettel.

A megerősítéses tanulás a döntési koncepció tanulásán alapuló művelet. Ebben a tanulásban a cselekvések a meghozott döntésnek megfelelően alapulnak, hogy az eredmények értékesebbé váljanak a kimeneten vagy a kívánt kedvező állapotban. A tanuló azonban nem rendelkezik előzetes információval az adatokról. Miután megadta a helyzetet, megtanulja eldönteni, hogy melyik cselekvés az adott helyzetnek megfelelően kell végrehajtani. A jelenlegi és a jövőbeli helyzetet a tanuló döntése, azaz a megtett cselekvés befolyásolja. A megerősítéses tanulás kizárólag két feltételre támaszkodik: a késleltetett eredményre és a próba és hiba keresésére.

2.2.3 Alkalmazási területei

A szakirodalom számos alkalmazási területet, részterületet mutat be a gépi tanulással kapcsolatosan. A valós alkalmazásokat az alábbiakban felsorolom és a 2.3. ábrán vizualizálom.

2.3. ábra: Gépi tanuális applikációjának területei [10]

A számítógépes látás a gépi tanulás sokoldalú területe, amely a vizuális adatok feldolgozására, elemzésére és felismerésére képezi a gépeket. A számítógépes látás különböző kulcsfontosságú algoritmusai a KNN, SVM, Naïve Bayes. E terület részterületei az objektumdetektálás, objektumfeldolgozás, felismerés.

A COVID-19 járvány óta az új korszak technológiái, mint például az arcfelismerés és az íriszszkennelés, a legnagyobb keresletet élvezik, mivel az ujjlenyomat-hitelesítés nem felelt meg a távolsági normáknak.

A gépi tanuláson alapuló arcfelismerő technológiát arra használják, hogy a zsúfolt helyeken a kongresszusi központok, repülőterek és különböző más fontos események látogatóiból felismerjék esetleges terrorcsoportok tagjait. A COVID-19 járványos helyzete óta ez a technológia nagyon hasznosnak bizonyul az érintés nélküli kommunikációban és a biztonságban. Így jelenleg számos vállalkozásban használják. A számítógépes látást az arcfelismerésben is használják biztonsági célokra. Ezenkívül a szakmai intézmények automatikus jelenléti rendszerének ellenőrzésére is használják. Ez megkönnyíti a hagyományos módszerek, például a könnyen ellopható kulcsok és személyi igazolványok használatát. Különböző más alkalmazásokat, például a FacePRO, Waymo alkalmazásokat használnak arcfelismerésre, illetve az autó biztonságos vezetésére.

A kézírás-felismerő alkalmazás megkönnyíti a munkát olyan szervezetek számára, ahol a kézzel írott dokumentumok száma hatalmas. Például egyetemek, vizsgaközpontok, rendőrség stb. A dokumentumok beolvasása és digitalizálása néhány perc alatt elvégezhető.

A beszédfelismerés a beszélt szavak szöveggé fordításának folyamata. Előnyökkel jár az egészségügyben, a hadseregben, az autós rendszerekben, illetve a mindennapi életben a

hangalapú interfészek és hangalapú asszisztensek létrehozásában, mivel segít a hozzáférhetőség javításában.

A beszédfelismerés más néven beszédből szöveggé és automatikus beszédfelismerésként is ismert. Erre különböző technológiákat használnak: mesterséges neurális hálózatok, vektor kvantálás, dinamikus időcsomagolás.

Sok kutatással és ráfordított energiával egy speciális szoftver képes lehet pontosan felismerni az emberi testben bármilyen eltérést az egészségügyi osztályon. Egyszerre képes különböző paramétereket észlelni és feldolgozni az orvosi feljegyzésekhez való valós idejű alkalmazásokban. Az orvosi dokumentáció statisztikai elemzése is nagyszerű feladatnak bizonyul.

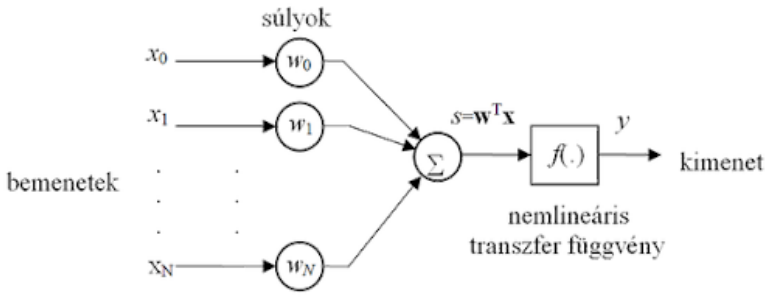
A múltbeli adatokon alapuló előrejelzések gépi tanulással végezhetők el. Különböző alkalmazások, például részvényárfolyam-előrejelzések, tudományos kutatás, marketingkampányok és még sok más eset. Általában mesterséges neurális hálózatokat és véletlen erdő algoritmusokat használnak előrejelzésekhez. Különböző részterületei a szövegosztályozás, képosztályozás, orvosi diagnosztika.

A gépi tanulás egyik hasznos területe a banki és pénzügyi szektor, ahol a csalások felderítésének így nagyobb lehet az esélye, amennyiben a pénzmozgások digitálisan zajlanak. A csalások felderítése és megelőzése az ügyféltranzakciók mintáinak azonosítása, a furcsa viselkedés azonosítása, a hitelpontszámok alapján történik. A gépi tanulás osztályozási és regressziós technikáit, valamint a neurális hálós munkákat használják a csalások felderítésében. A Tensorflow és a Keras segítségével automatikus kódolási technikát hoznak létre a hitelkártyacsalások felderítésére, amely hatalmas összegeket takarít meg a pénzintézetek számára a költségvisszatérítésre és a biztosításra.

2.3 Neurális hálók

A neurális hálózat [11] egy olyan rendszer, amely sok kis részegységből álló elrendezést alkot, és azok hasonló vagy azonos típusú információfeldolgozási feladatokat végeznek. Ez a hálózat képes tanulni, általában példák alapján, és van egy előhívási mechanizmusa, amely lehetővé teszi a megtanult információk használatát.

A neurális hálózat egyik részegysége a neuron, amely az információfeldolgozás alapegysége, és a biológiai neuronok felépítését (2.4. ábra) modellezi. Ahogy a biológiai neuronoknál, a mesterséges neuronoknak is van véges számú bemenete (X = x_1, x_2...x_n), amelyek mindegyikéhez tartozik egy súly (W = w_1, w_2...w_n). Ezeket a bemeneteket megszorozzuk a súlyokkal, majd hozzáadjuk egy eltolás (bias) értékhez, hogy előállítsuk a lineáris kimenetet (Z):



2.4. ábra: Egy neuron felépítése [12]

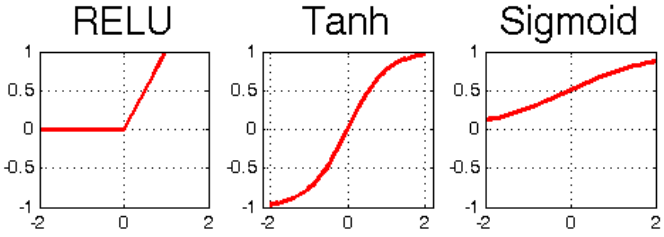
$$Z = \sum_{i=0}^{\infty} X_i W_i + b. \tag{2.1}$$

A mesterséges neuronok, akárcsak biológiai társaik, nem minden bemenetre reagálnak egyformán. Vannak esetek, amikor "tüzelnek", aktiválódnak, míg mások esetén passzívak maradnak. Ezt a viselkedést egy nemlineáris függvény, az aktivációs függvény (f(Z) = Y) segítségével modellezzük.

Az aktivációs függvényekkel szemben két alapvető elvárás fogalmazható meg:

- Nemlinearitás: A függvény ne legyen lineáris. Ez elengedhetetlen ahhoz, hogy a neuron komplex, nem triviális bemenet-kimenet relációkat tanuljon meg.
- Aktivitás: A függvény "aktív" legyen a releváns bemenetekre. Ez azt jelenti, hogy a neuron reagáljon a számára fontos információkra, és ne maradjon passzív, ha aktiválódnia kellene.

A neuron kimenete nem csak az aktivációs függvény értékétől függ, hanem befolyásolja a küszöbérték is. Ez egy eltolás, amely beállítja, hogy a függvény mikor aktiválódjon. Ha a bemenet értéke meghaladja a küszöbértéket, a neuron "tüzel", aktiválódik. Ellenkező esetben passzív marad. Az aktivációs függvények kulcsfontosságú elemei a mesterséges



2.5. ábra: Aktivációs függvény típusok [13]

neuronhálózatoknak. Számos különböző típus létezik, mindegyiknek megvannak a maga

előnyei és hátrányai. A leggyakoribbak közé tartoznak a(z) (2.5. ábrán is látható) szigma (sigmoid), a hiperbolikus tangens (tanh), és a ReLU (Rectified Linear Unit) függvények:

- Szigmoid függvény (o): Alacsony Z értékek esetén a neuron kimenete közel 0-hoz. Magas Z értékek esetén a kimenet 1-hez közelít.
- Tangens hiperbolikusz (tanh): Hasonlít a szigmoid függvényhez, de -1 és 1 közötti értékeket adhat ki. Gyakran használják kimeneti neuronoknál, ahol a szélsőértékeknek jelentése van (pl. 0/1: nem/igen, -1/0/1: nem/nem tudom/igen).
- ReLU (Rectified Linear Unit): f(x) = max(0, x) formulával írható. Gyakran alkalmazzák belső neuronoknál, főleg gépi látási feladatokban. Gyorsabban konvergál, mint a szigmoid vagy tanh függvények, és nem telítődnek a kimeneti értékek a pozitív tartományban. Módosított változata a "leaky ReLU", ami kezeli a negatív értékeket is.
- Softmax: Bizonyos esetekben meg kell követelni, hogy a kimeneti vektor valószínűségi eloszlást kövessen. A softmax aktivációs függvény ezt biztosítja. A legnagyobb bemenetet kiemeli, a kisebbeket pedig csökkenti, így segítve a helyes osztályozást. Sikeres előrejelzés esetén a kimeneti vektor egyik elemének értéke közel 1-hez konvergál, míg a többiek 0-hoz.

A választás a konkrét feladattól és a hálózat architektúrájától függ. Fontos megjegyezni, hogy az aktivációs függvények jelentősen befolyásolhatják a hálózat tanulási képességét és teljesítményét.

2.4 Konvolúciós neurális hálók

2.4.1 2D CNN

A 2D CNN-ek széleskörűen alkalmazhatók a számítógépes látásban.

Képosztályozás: Ez a feladat egy kép kategóriába sorolását jelenti. A CNN-ek forradalmasították ezt a területet. A LeNet-5 tekinthető az első alkalmazásnak, amelyet kézzel írt számjegyek osztályozására használtak. Az AlexNet[14] továbbfejlesztette a CNN-alapú osztályozást. Ezután a kutatók külön figyelmet fordítottak a hálózatok mélységének fontosságára, ami olyan mélyebb hálózati struktúrákhoz vezetett, mint a GoogLeNet [15] és a VGGNets[16], amelyek jelentősen javították az osztályozási pontosságot. Olyan technikákat vezettek be, mint az SPP-Net és a ResNet a kihívások kezelésére és a még mélyebb hálózatok képzésének lehetővé tételére. A ResNet és a DenseNet közötti hasonlóságok és különbségek elemzésére a képosztályozáshoz a Double Path Networks (DPN) hálózatokat javasolták. A DPN nem csak a képjellemzőket osztja meg, hanem rugalmasságot is biztosít a struktúrajellemzők kinyerésében. A Facebook kiadta a ResNeXt-101 forráskódját, amely az ImageNet-en a legkorszerűbb eredményeket érte el. A CNN-ek az

orvosi képosztályozásban, a közlekedési jelenetek osztályozásában és más területeken is alkalmazhatók.

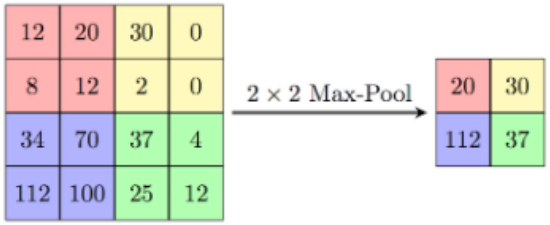
Objektum-felismerés: A tárgyfelismerés a képosztályozásra épül. A rendszereknek nemcsak a képen lévő objektum kategóriáját kell azonosítaniuk, hanem egy határoló kerettel is meg kell jelölniük azt. Két fő megközelítés létezik: egylépcsős és kétlépcsős. Az egylépcsős megközelítések, mint például a YOLO, az SSD és a CornerNet, közvetlenül visszaadják a képen lévő objektumok kategóriájának valószínűségét és pozíciójának koordinátáit. A kétlépcsős megközelítések, mint például az R-CNN, a Fast R-CNN és a Faster R-CNN, először kiválasztják az érdekes régiókat (javaslatokat), majd osztályozzák a javaslatokon belüli objektumokat. A Faster R-CNN a sebesség növelése érdekében bevezetett egy régiójavaslat-hálózatot (RPN). A Faster R-CNN a több skálájú jellemzők fúziójának lehetővé tétele érdekében Feature Pyramid Network (FPN) hálózattal egészült ki. A fent említett megközelítések mindegyike horgonyzódobozokra támaszkodik az objektumok helyének meghatározásához, ami gondos kiválasztást és sok hiperparamétert tesz szükségessé. A CornerNet ezt úgy oldja meg, hogy elhagyja a horgonydobozokat, és közvetlenül a határoló dobozok sarkait jósolja meg.

Képszegmentálás: A képszegmentálás a képet különböző területekre osztja a különböző szemantikai entitások határainak jelölésével. A teljesen konvolúciós hálózatok (FCN) voltak az első CNN-struktúrák, amelyeket képszemiantikai szegmentációra alkalmaztak. Az U-háló, amely több multiskálájú jellemzőt tartalmaz, különösen sikeresnek bizonyult az orvosi képek szegmentálásában. Más hálózatokat, például az ENet-et és a PSPNet-et speciális problémákra fejlesztették ki. A maszk R-CNN olyan példányszegmentálási feladatokat old meg, ahol minden egyes objektumot azonosítani kell, és annak maszkját generálni kell. A RetinaNet-alapú módszereket valós idejű példányszegmentálásra tervezték. A panoptikus szegmentálás a szemantikus és a példányszegmentálást ötvözi.

Arcfelismerés: Az arcfelismerés az arcvonásokon alapuló biometrikus azonosítási technika. A DeepFace és a DeepID kiváló eredményeket ért el, első alkalommal felülmúlva az embereket. A DeepFace magában foglalja a felismerést, az összehangolást, az extrakciót és az osztályozást. A DeepID közvetlenül két arcképet ad be az osztályozáshoz. Az újabb megközelítések a képzés során használt veszteségfüggvény javítására összpontosítanak a jobb felismerési pontosság elérése érdekében. [17]

2.4.2 Max Pooling

A (2.6 ábrán látható) Max Pooling egy olyan pooling művelet, amely kiszámítja a jellemzőtérkép foltjainak maximális értékét, és ezt használja fel egy lemintavételezett (pooling) jellemzőtérkép létrehozásához. Általában egy konvolúciós réteg után használják. A művelet kis mértékben növeli a transzlációs invarianciát - vagyis a kép kis mértékű átfordítása nem befolyásolja jelentősen a legtöbb összevont kimenet értékét.[18]



2.6. ábra: Max pooling vizualizáció [18]

Max Pooling működése

Egy képet kisebb részekre, gyakran rácsnak vagy szűrőnek nevezett részekre osztunk. A Max Pooling ezeken a részeken külön-külön, a következőképpen működik:

- Szkennelés: A Max Pooling egy kis ablakot (általában 2x2 vagy 3x3) mozgat a bemeneti jellemzőtérképen.
- Kiválasztás: A Max Pooling minden egyes ablakban kiválasztja a maximális értéket. Ez olyan, mintha kiválasztaná a legfontosabb információt az adott területen.
- Down-sampling: Miután kiválasztotta a maximális értékeket az összes ilyen ablakból, a Max Pooling csökkenti a jellemzőtérkép méretét. A kép kisebb lesz, de a lényeges részletek megmaradnak.

2.5 Szűrés

2.5.1 Harris sarokdetektor

A Harris Corner Detector egy sarokdetektáló operátor, amelyet a számítógépes látás algoritmusaiban gyakran használnak a sarkok kinyerésére és a kép jellemzőinek kikövetkeztetésére. Először Chris Harris és Mike Stephens mutatta be 1988-ban a Moravec-féle sarokdetektor továbbfejlesztése nyomán. Az előzőhöz képest Harris sarokdetektora közvetlenül figyelembe veszi a sarokpontok differenciálását az irányra vonatkoztatva, ahelyett, hogy 45 fokos szögenként eltolt foltokat használna, és bizonyítottan pontosabbnak bizonyult az élek és a sarkok megkülönböztetésében. Azóta továbbfejlesztették és számos algoritmusban alkalmazták a képek előfeldolgozására a későbbi alkalmazásokhoz.

Sarokérzékelés

Az ötlet lényege, hogy a kép minden egyes p pixele körül egy kis ablakot veszünk figyelembe. Az összes ilyen pixelablakot azonosítani akarjuk, amelyek egyediek. Az egyediséget úgy lehet mérni, hogy minden ablakot egy kis mértékben eltolunk egy adott irányba, és mérjük a pixelértékekben bekövetkező változás mértékét.

Formálisabban fogalmazva, az eltolás előtti és utáni pixelértékek négyzetes különbségének összegét (SSD) vesszük, és azonosítjuk azokat a pixelablakokat, ahol az SSD mind a 8 irányban történő eltolás esetén nagy. Definiáljuk az E(u,v) változásfüggvényt az összes négyzetes különbségösszeg (SSD) összegeként, ahol u,v a 3 x 3 ablakunkban lévő minden pixel x,y koordinátája, I pedig a pixel intenzitásértéke. A kép jellemzői az összes olyan képpont, amelynek E(u,v) értéke nagy, és amelyet egy bizonyos küszöbérték határoz meg.

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \tag{2.2}$$

A sarokfelismeréshez ezt az E(u,v) függvényt kell maximalizálnunk. Ez azt jelenti, hogy a második kifejezést kell maximalizálnunk. A fenti egyenletre a Taylor-féle kiterjesztést és néhány matematikai lépést alkalmazva a következő végső egyenletet kapjuk:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.3}$$

Most nevezzük át az összegzett mátrixot, és nevezzük M-nek:

$$M = \sum w(x,y) \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix} \tag{2.4}$$

Ne feledjük, hogy azt akarjuk, hogy az SSD mind a nyolc irányban nagy legyen az eltolásokban, vagy fordítva, hogy az SSD egyik irányban se legyen kicsi. Az M sajátvektorainak megoldásával megkaphatjuk az SSD legnagyobb és legkisebb növekedésének irányait is. A megfelelő sajátértékek megadják számunkra ezeknek a növekedéseknek a tényleges értékösszegét.

2.5.2 Szuperpixelek és SLIC

A szuperpixel olyan pixelek csoportjaként definiálható, amelyek közös jellemzőkkel rendelkeznek (például a pixel intenzitása). A szuperpixelek számos számítógépes látás- és képfeldolgozási algoritmusban, például a képszegmentálásban, a szemantikus címkézésben, a tárgyak felismerésében és követésében válnak hasznossá a következők miatt.

- Több információt hordoznak, mint a pixelek.
- A szuperpixelek érzékelési jelentéssel bírnak, mivel az adott szuperpixelhez tartozó pixelek hasonló vizuális tulajdonságokkal rendelkeznek.
- A képek kényelmes és kompakt ábrázolását biztosítják, ami nagyon hasznos lehet számításigényes problémák esetén.

SLIC (Simple Linear Iterative Clustering) algoritmus a Superpixel generálásához

Ez az algoritmus szuperpixeleket hoz létre a pixelek színhasonlóságuk és a képsíkon belüli közelségük alapján történő klaszterezésével. Ez egy ötdimenziós [labxy] térben történik, ahol [lab] a pixel színvektora a CIELAB [19] színtérben, *xy* pedig a pixel pozíciója. A térbeli távolságokat normalizálnunk kell ahhoz, hogy az euklideszi távolságot ebben az 5D térben használhassuk, mivel a CIELAB-térben a két szín közötti maximális lehetséges távolság korlátozott, míg az *xy*-síkban a térbeli távolság a kép méretétől függ. Ezért a pixelek klaszterezéséhez ebben az 5D térben egy új távolságmérőt kell bevezetni, amely figyelembe veszi a szuperpixel[20] méretét, és amely a következőkben ismertetve lesz.

N	Number of pixels in the input image
K	Number of Superpixels used to segment the input image
N/K	Approximate size of each superpixel
$\boldsymbol{S = \sqrt{N/K}}$	For roughly equally sized superpixels there would be a superpixel centre at every grid interval S

2.7. ábra: Algoritmus megértéséhez hasznos jelölések

Az algoritmus bemenetként egy kívánt számú, közel azonos méretű szuperpixel *K*-t vesz fel. Az algoritmus kezdetén *K* szuperpixel klaszterközpontot választunk ki

$$C_k = [l_k, a_k, b_k, x_k, y_k]k = [1, K] \tag{2.5}$$

értékkel, rendszeres *S* rácsközökkel. Mivel bármely szuperpixel térbeli kiterjedése megközelítőleg S^2 (egy szuperpixel hozzávetőleges területe), biztonsággal feltételezhető, hogy az ehhez a klaszterközponthoz tartozó pixelek az xy-síkon a szuperpixel központja körüli

2S×2S területen belül helyezkednek el. Az 5D térben használandó normalizált távolságmérték D_s a következőképpen definiált :

$$D = d_{la\beta} + (m/S) * d_{xy} \ldots . (eq1)[20] \tag{2.6}$$

ahol $d_{la\beta}$ = $((l_k\text{-}l_i)^2 + (a_k\text{-}a_i)^2 + (b_k\text{-}b_i)^2)$, d_{xy} = $((x_k\text{-}x_i)^2 + (y_k\text{-}y_i)^2)$ és D_s a labor távolság ($d_{la\beta}$)és az xy-sík távolság (d_{xy}) összege az *S* rácsintervallummal normalizálva. A D_s-be bevezetünk egy m változót, amely lehetővé teszi számunkra, hogy szabályozzuk a szuperpixel tömörségét. Minél nagyobb az m értéke, annál kompaktabb a klaszter. Ez az érték a következő tartományban lehet[1,20] [20].

Algoritmus működése

Ez az algoritmus *K* szabályos távolságban lévő klaszterközpontok mintavételezésével kezdődik, és a 3×3 szomszédságban lévő legalacsonyabb gradiens pozíciónak megfelelő maghelyekre helyezi őket (ez azért történik, hogy elkerüljük a klaszterek szélére helyezését, és hogy csökkentsük a zajos pixel kiválasztásának esélyét).

A kép gradiensek kiszámítása a következőképpen történik:

$$G(x, y) = I(x + 1, y) - I(x - 1, y) + I(x, y + 1) - I(x, y - 1) \tag{2.7}$$

ahol *I(x,y)* az *(x,y)* pozícióban lévő pixelnek megfelelő labvektor. Ez mind a szín-, mind az intenzitásinformációt figyelembe veszi. A kép minden egyes képpontjához hozzárendeljük a legközelebbi klaszterközpontot, amelynek keresési területe átfedésben van az adott képponttal. Miután az összes pixelt hozzárendeltük a legközelebbi klaszterközponthoz, egy új központot számolunk ki a klaszterhez tartozó összes pixel átlagos labxy vektoraként.[20]

2.6 Jupyter

A Jupyter Notebook egy interaktív programozási környezet, amelyet széles körben használnak adatelemzéshez, gépi tanuláshoz és vizualizációhoz. A Jupyter Notebook lehetővé teszi a kód futtatását, szöveg beírását és a kimenetek megjelenítését egyetlen dokumentumban, ami megkönnyíti a kísérletezést és a prototípuskészítést.

A Jupyter telepítése egyszerű, és többféleképpen is elvégezhető. A leggyakoribb módszer a pip csomagkezelő használata:

```
pip install jupyter
```

Ha a Jupyter telepítése sikeres, elindíthatja a következő paranccsal:

```
jupyter notebook
```

Ez megnyit egy böngészőablakot a Jupyter Notebook felülettel, alapértelmezetten a localhost:8888 porton.

2.6.1 A Jupyter kapcsolata a gesztusfelismerő rendszeremmel

A Jupyter Notebookot számos feladatra lehet használni a gesztusfelismerő rendszerek fejlesztése során, többek között:

- Adatok betöltése és előkészítése: A Jupyter Notebookot az érzékelőadatok betöltésére, tisztítására és előkészítésére lehet használni a modellezéshez.
- Modellek betanítása és értékelése: A Jupyter Notebookot gépi tanulási modellek betanítására és értékelésére lehet használni gesztusfelismeréshez.
- Eredmények vizualizálása: A Jupyter Notebookot az eredmények vizualizálására lehet használni, például a felismert gesztusok megjelenítésére képekben vagy videókban.

A Jupyter Notebook egy hatékony eszköz a gesztusfelismerő rendszerek fejlesztéséhez. A kódolás, az adatelemzés és a vizualizáció széles skálájára nyújt lehetőséget, ami megkönnyíti a kutatók és fejlesztők számára a hatékony gesztusfelismerő rendszerek létrehozását.

Példa a Jupyter Notebook használatára gesztusfelismerő rendszerben:

Tegyük fel, hogy a gesztusfelismerő rendszerhez egy olyan funkciót fejlesztünk, amely kézi jelek felismerésére képes. A Jupyter Notebookot a következő feladatokra használhatjuk:

- A kézi jelek adatkészletének betöltése és előkészítése.
- Egy konvolúciós neurális hálózat (CNN) modell betanítása a kézi jelek felismerésére.
- A modell értékelése a tesztkészleten.
- A felismert gesztusok megjelenítése képekben a jegyzetfüzetben.

A Jupyter Notebook lehetővé teszi számunkra, hogy ezeket a feladatokat egyetlen interaktív környezetben végezzük el, ami megkönnyíti a kísérletezést és a prototípuskészítést.

3 GESZTUSFELISMERŐ RENDSZER

3.1 TensorFlow

A Google 2015 végén az Apache 2.0 licensz alatt nyíltan elérhetővé tette a TensorFlow nevű gépi tanulási keretrendszerét. Ezt megelőzően a Google saját tulajdonában volt, és többek között a beszédfelismerő, a kereső, a fényképek és a Gmail alkalmazásaiban használta.

A könyvtár C++ nyelven van implementálva, és rendelkezik egy kényelmes Python API-val, valamint egy kevésbé kedvelt C++ API-val. Az egyszerűbb függőségek miatt a TensorFlow gyorsan telepíthető különböző architektúrákra. A Theanóhoz[21] hasonlóan a számításokat folyamatábrák formájában írják le, elválasztva a tervezést a megvalósítástól.

Ez a kettősség kevés gonddal lehetővé teszi, hogy ugyanazt a tervezést ne csak nagyméretű, több ezer processzorral rendelkező oktatórendszereken, hanem mobil eszközökön is megvalósítsuk. Az egyetlen rendszer a platformok széles skálájára terjed ki. A TensorFlow egyik leghasznosabb tulajdonsága az automatikus differenciálási képessége. Új hálózatokkal lehet kísérletezni anélkül, hogy számos kulcsfontosságú számítást újra kellene definiálni.

Az összes matematika elvonatkoztatva és a motorháztető alatt kibontva. Olyan, mintha a WolframAlpha-t használnánk egy matematikai feladatsorhoz. A könyvtár másik jellemzője a TensorBoard nevű interaktív vizualizációs környezet. Ez az eszköz folyamatábrát mutat az adatok átalakításának módjáról, időbeli összefoglaló naplókat jelenít meg, és nyomon követi a teljesítményt.

A 3.1 ábra egy példát mutat arra, hogyan néz ki a TensorBoard használat közben.

3.1. ábra: TensorBoard bemutatása [22]

A TensorFlow-ban a prototípusok készítése sokkal gyorsabb, mint a Theano-ban (a kód elindítása másodpercek alatt történik, szemben a percekkel), mivel sok művelet előre lefordítva érkezik. A kód hibakeresése egyszerűvé válik az algráfok végrehajtása miatt; egy teljes számítási szegmens újrafelhasználható újraszámítás nélkül.

A TensorFlow nem csak neurális hálózatokkal foglalkozik, hanem tartalmaz beépített eszközöket mátrixszámításokhoz és -manipulációkhoz is. A legtöbb könyvtár, például a Torch és a Caffe kizárólag mély neurális hálózatokhoz készült, míg a TensorFlow rugalmasabb és skálázhatóbb is.

A könyvtár jól dokumentált, és a Google hivatalosan támogatja. A gépi tanulás összetett téma, így megnyugtató, hogy a TensorFlow mögött egy kivételesen jó hírű vállalat áll.

3.1.1 Keras

A Keras [23] kezdeti építőköve egy modell, a legegyszerűbb modellt pedig szekvenciálisnak nevezzük. A szekvenciális Keras modell egy lineáris csővezeték (egy verem) neurális hálózatok rétegeiből. Ez a kódrészlet egyetlen réteget definiál 12 mesterséges neuronnal, és 8 bemeneti változót (más néven feature-t) vár:

```
1 from keras.models import Sequential
2 model = Sequential()
3 model.add(Dense(12, input_dim=8, kernel_initializer="random_
    uniform"))
```

1. Kód: Keras importálás

Minden neuron inicializálható meghatározott súlyokkal. A Keras néhány választási lehetőséget biztosít, amelyek közül a leggyakoribbak a következők:

- random-uniform: A súlyok inicializálása egyenletesen véletlenszerű kis értékekkel történik a (-0.05, 0.05) tartományban. Más szóval, az adott intervallumon belül bármelyik érték egyforma valószínűséggel kirajzolódik.
- random-normal: A súlyok inicializálása egy Gauss-féle, nulla átlaggal és 0,05-ös kis szórással. Azok számára, akik nem ismerik a Gauss-t, gondoljanak egy szimmetrikus haranggörbe alakjára.
- zero: Minden súlyt nullára inicializálunk.

Rétegek [24]

A tf.keras.layers.Layer osztály a Keras alapvető absztrakciója. Egy Layer egy állapotot (súlyok) és néhány számítást (a tf.keras.layers.Layer.call metódusban definiálva) foglal magába.

A rétegek által létrehozott súlyok lehetnek betaníthatóak vagy nem betaníthatóak. A rétegek rekurzívan összeállíthatók: Ha egy rétegpéldányt egy másik réteg attribútumaként rendelünk hozzá, a külső réteg elkezdi követni a belső réteg által létrehozott súlyokat.

A rétegeket olyan adatelőfeldolgozási feladatok kezelésére is használhatja, mint a normalizálás és a szövegvektorizálás. Az előfeldolgozó rétegeket közvetlenül a modellbe lehet beépíteni, akár a képzés során, akár a képzés után, ami hordozhatóvá teszi a modellt.

Modellek [24]

A modell egy olyan objektum, amely rétegeket csoportosít, és amely adatokon képezhető.

A legegyszerűbb modelltípus a szekvenciális modell, amely rétegek lineáris halmaza. Az összetettebb architektúrákhoz vagy a Keras funkcionális API-t használhatja, amellyel tetszőleges réteggráfokat építhet, vagy alosztályozással a semmiből írhat modelleket.

A tf.keras.Model osztály beépített képzési és kiértékelési módszerekkel rendelkezik:

- tf.keras.Model.fit: A modellt egy meghatározott számú epochára képzi.
- tf.keras.Model.predict: Kimeneti előrejelzéseket generál a bemeneti mintákhoz.
- tf.keras.Model.evaluate: Visszaadja a modell veszteség- és metrikaértékeit; a tf.keras.Model.compi metódussal konfigurálható.

Ezek a metódusok hozzáférést biztosítanak a következő beépített képzési funkciókhoz:

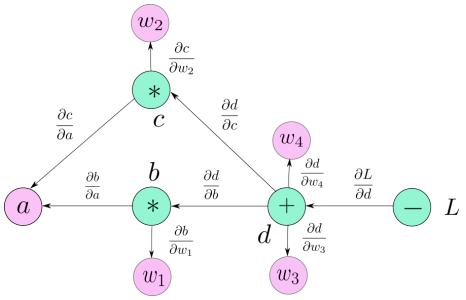
- Visszahívások. A beépített callbackeket kihasználhatja a korai leállításhoz, a modell ellenőrzőpontozásához és a TensorBoard felügyeletéhez. Egyéni visszahívásokat is implementálhat.
- Elosztott képzés. Könnyedén kiterjesztheti a képzést több GPU-ra, TPU-ra vagy eszközre.
- Lépésfúzió. A steps-per-execution argumentummal a tf.keras.Model.compile-ban több tételt is feldolgozhat egyetlen tf.function hívásban, ami jelentősen javítja az eszközkihasználtságot TPU-kon.

3.2 Pytorch

A PyTorch [25] az értékeket Tensorokba szervezi, amelyek általános n-dimenziós tömbök, és amelyek gazdag adatmanipulációs műveletekkel rendelkeznek. Egy modul definiál egy transzformációt a bemeneti értékekből a kimeneti értékekbe, és a viselkedését az előrehaladás során a forward tagfüggvénye határozza meg. Egy Modul paraméterként Tenzorokat tartalmazhat. Például egy Lineáris modul tartalmaz egy súly paramétert és egy torzítás paramétert, amelynek forward függvénye a bemenetet a súllyal szorozva és a torzítást hozzáadva állítja elő a kimenetet.

Egy alkalmazás úgy állítja össze a saját Modulját, hogy a saját Modulokat (pl. lineáris, konvolúció stb.) és Függvényeket (pl. relu, pool stb.) összefűzi az egyéni forward függvényben. Egy tipikus képzési iteráció tartalmaz egy előrehaladást a veszteségek generálására a bemenetek és címkék felhasználásával, egy visszafelé haladást a paraméterek gradienseinek kiszámítására, és egy optimalizáló lépést a paraméterek frissítésére a gradiensek felhasználásával.

Pontosabban, az előremenő lépés során a PyTorch egy autograd gráfot (lsd. 3.2 ábra) [26] épít az elvégzett műveletek rögzítésére. Ezután a visszafelé irányuló lépés során az autograd gráfot használja a grádiensek létrehozásához szükséges visszaterjedés elvégzéséhez. Végül az optimalizáló a gradienseket alkalmazza a paraméterek frissítésére. A képzési folyamat ezt a három lépést addig ismétli, amíg a modell konvergál.



3.2. ábra: Autograd gráf

A Python nyelvvel való zökkenőmentes integrációja lehetővé teszi a könnyű kísérletezést és a gépi tanulási modellek gyors fejlesztését:

Dinamikus számítási grafikon: A PyTorch dinamikus számítási gráfot használ, ami nagyobb rugalmasságot és egyszerűbb használatot tesz lehetővé, mint a más keretrendszerekben használt statikus számítási gráfok. Dinamikus számítási gráfok: A PyTorch dinamikus számítási gráfot használ, ami lehetővé teszi a fejlesztők számára, hogy a program végrehajtása közben menet közben módosítsák a gráfot. Ez a funkció megkönnyíti az összetett architektúrákkal és dinamikus bemenetekkel való munkát, és nagyobb rugalmasságot tesz lehetővé a neurális hálózatok építésében és képzésében.

Neurális hálózatok építése: A PyTorch gazdag eszközkészletet biztosít a neurális hálózatok építéséhez és képzéséhez, beleértve a konvolúciós neurális hálózatok (CNN), a rekurrens neurális hálózatok (RNN) és más architektúrák támogatását. Zökkenőmentes GPU-gyorsítás: A PyTorch könnyen használható API-kat biztosít a GPU-gyorsítás kihasználásához, ami jelentősen javíthatja a mélytanulási modellek képzésének sebességét. A fejlesztők mindössze néhány sornyi kóddal kihasználhatják a GPU-k erejét a gyorsabb és hatékonyabb számítások érdekében.

GPU-gyorsítás: A PyTorch kihasználja a GPU-k erejét, hogy felgyorsítsa a mesterséges

intelligencia modellek képzését és következtetését, ami gyorsabb és hatékonyabb számításokat eredményez.

TorchScript: A PyTorch tartalmazza a TorchScriptet, egy hatékony eszközt a modellek sorozatba rendezéséhez és optimalizálásához a termelési telepítéshez. Ez a funkció lehetővé teszi a fejlesztők számára a zökkenőmentes átmenetet a prototípusok és kísérletek készítéséből a modellek termelési környezetbe történő telepítésébe.

Ez az ökoszisztéma olyan népszerű könyvtárakat tartalmaz, mint a torchvision a számítógépes látási feladatokhoz és a torchaudio az audiofeldolgozási feladatokhoz. [27]

3.3 OpenCV

Az OpenCV könyvtár (a 2.2-es verzió óta) több modulra oszlik, ahol minden modul általában úgy értelmezhető, hogy a számítógépes látás problémáinak egy-egy csoportjára vonatkozik. Az összes osztály és függvény a cv névtérben van definiálva. Ezért a hozzáférésükhöz vagy a fő függvénydefiníció előtt a cv névteret használó deklarációval; vagy az OpenCV osztály- és függvénynevek előtt a cv:: névtérspecifikációval előzhetjük meg az OpenCV osztály- és függvényneveket. A fő objektum a Mat osztályba tartozik. Amint azt az osztálynév is sugallja, ez lényegében egy mátrix, amely valamilyen kép pixelértékeit és ezen kívül számos attribútumot tartalmaz a képről. A legegyszerűbb esetben egy kép létrehozható a cv:: Mat image;, létrehozva egy 0 x 0 méretű képet. Az image objektum talán legfontosabb tagváltozója a data, ahol az image.data tag tulajdonképpen egy mutató a képadatokat tartalmazó allokált memóriablokkra (ebben a triviális esetben image.data=0). Alternatívaként a Mat objektum létrehozása során explicit módon megadhatjuk az egyes mátrixelemek kezdeti méretét és típusát. Ez a típus megadja például az előjeles 1 bájtos pixeles képértékeket (CV-8U), vagy egy színes kép három csatornáját (CV-8UC3), vagy akár 32 bites/64 bites lebegőpontos számokat (CV-32F). [28]

4 ADATGYŰJTÉS, ELŐFELDOLGOZÁS ÉS MODELL FELÉPÍTÉSE

A gesztusfelismerési rendszer fejlesztése során először megfelelő adatokat gyűjtök és dolgozok fel, majd ezek alapján alakítom ki a modell architektúráját. A modell célja, hogy képes legyen az előre meghatározott kézmozdulatok, gesztusok felismerésére, legyenek azok statikusak vagy dinamikusak.

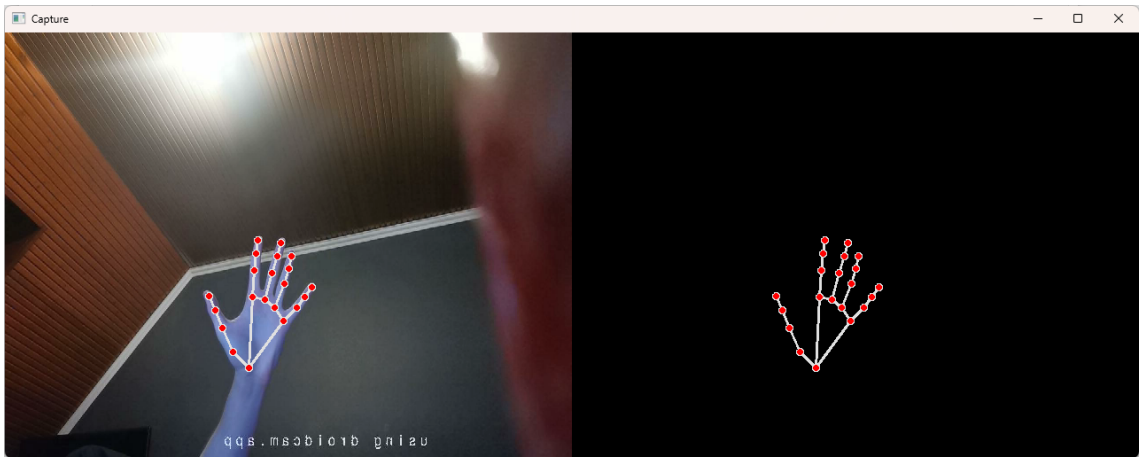
4.1 Adatgyűjtés és Előfeldolgozás

A gesztusok rögzítése kétféleképpen történik: *statikus* és *dinamikus* gesztusokként, amelyek különböző formátumban kerülnek tárolásra a modell betanítása érdekében. Ezek külön tárolása lehetővé teszi a modell számára, hogy specifikus mintákat tanuljanak meg, ami növeli a gesztusfelismerés pontosságát és hatékonyságát.

4.1.1 Statikus Gesztusok Adatgyűjtése

A statikus gesztusokat egyetlen képkockából nyerem ki, ahol a *MediaPipe* könyvtár segítségével azonosítom a kéz karakterisztikus pontjait, az úgynevezett *landmark* pontokat [29]. Ezeket a pontokat háromdimenziós koordináták (x, y, z) formájában `JSON` fájlban tárolom. Minden egyes gesztushoz a rögzített kulcspontokat többször megismétlem, hogy egyetlen statikus képből mozdulatsort hozzak létre. Ez lehetővé teszi, hogy a modell számára időbeli dimenzióval rendelkező, egységes hosszúságú adatokat biztosítsak. Az előállított mozdulatsorok használata segít az LSTM rétegeknek abban, hogy a statikus gesztusokat dinamikus jellegű adatként kezeljék.

A statikus gesztusok adatgyűjtéséhez egy interaktív alkalmazást készítettem, amely a 4.1 ábrán látható. Ez az alkalmazás valós időben képes a MediaPipe könyvtár által detektált kéz kulcspontjait rögzíteni. Balra látható a kamerából nyert szűretlen kép, jobbra pedig a kimaszkolt MediaPipe kulcspontok jelennek meg. Az alábbiakban bemutatom az alkalmazás működését, amelyet az adatgyűjtési folyamat megkönnyítésére fejlesztettem ki.



4.1. ábra: Gesztusok begyűjtésére használt labeler

Könyvtárak Inicializálása és Adatmappák Létrehozása: Az adatokat strukturáltan tárolom, ezért a gesztus nevének megfelelően külön mappákat hozok létre a képek és kulcspontok tárolására. Ez a mappaszerkezet biztosítja a különböző gesztusok egyszerű szétválasztását és későbbi hatékony feldolgozását.

```
import os

gesture_name = 'defend'
images_folder = f'data/images_{gesture_name}/{gesture_name}'
landmarks_folder = f'data/landmarks_{gesture_name}/{gesture_name}'
os.makedirs(images_folder, exist_ok=True)
os.makedirs(landmarks_folder, exist_ok=True)
```

2. Kód: Statikus gesztusok mentése python alkalmazásban

MediaPipe és Webkamera Inicializálása: A *MediaPipe Hands* modult inicializálom a kéz kulcspontjainak detektálására, majd csatlakozom a webkamerához. A program biztosítja, hogy a kamera megnyitása előtt ellenőrzi annak elérhetőségét, így elkerülve a futás közbeni hibákat.

```
import cv2
import mediapipe as mp

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
hands = mp_hands.Hands(static_image_mode=False, max_num_hands=1)
cap = cv2.VideoCapture(0)
```

3. Kód: Inicializálások

Kulcspontok Kinyerése és Mentése: A webkamera által biztosított képkockákból a *MediaPipe* segítségével kinyerem a kéz kulcspontjait, majd a képeket és kulcspontokat elmentem. Ez a folyamat biztosítja az adatokat tartalmazó fájlok megfelelő elnevezését és indexálását a későbbi feldolgozáshoz.

```
while True:
    ret, frame = cap.read()
    if not ret:
        print("Error: Cannot read frame from camera.")
        break

    frame = cv2.flip(frame, 1)
    image_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(image_rgb)

    if results.multi_hand_landmarks:
        hand_landmarks = results.multi_hand_landmarks[0]
        landmarks = [{'x': lm.x, 'y': lm.y, 'z': lm.z} for lm in
            hand_landmarks.landmark]

        with open(f'{landmarks_folder}/gesture_{image_counter}.
           json', 'w') as f:
            json.dump(landmarks, f)
        cv2.imwrite(f'{images_folder}/gesture_{image_counter}.
           jpg', frame)
        image_counter += 1
```

4. Kód: Kulcspontok kinyerése és mentése

4.1.2 Dinamikus Gesztusok Adatgyűjtése

A dinamikus gesztusok esetében egy teljes mozdulatsort rögzítek, amely több egymást követő képkockát foglal magába, mindegyik képkockán a kéz aktuális *landmark* pontjaival. Az ilyen típusú gesztusokat `.npy` fájlokban tárolom, ahol minden fájl egy adott mozdulatsort reprezentál. Az LSTM rétegek segítségével a modell képes időbeli függőségeket felismerni ezekből a mozdulatsorokból, ami elengedhetetlen a dinamikus gesztusok megbízható felismeréséhez [30]. Ez a megközelítés lehetővé teszi az időbeli változások elemzését, amely az egyszerű képkockák elemzésével nem lenne lehetséges.

A dinamikus gesztusok rögzítése során több egymást követő képkockát mentettem el, amelyek a mozdulatsort reprezentálják. Az egyes mozdulatsorokat önálló adatfájlként tárolom, így biztosítva a gesztusok időbeli folytonosságát és az adatfeldolgozás könnyű-

ségét.

Mozdulatsor Paraméterek Beállítása: A rögzítendő mozdulatsor hosszát (`sequence_length`) és nevét előre meghatároztam. Ez a beállítás lehetővé teszi, hogy az alkalmazás minden mozdulatsornál egységes hosszúságú adatokat rögzítsen.

```
gesture_name = 'circle_around'
sequence_length = 60
data_folder = f'data/sequences_{gesture_name}/{gesture_name}'
os.makedirs(data_folder, exist_ok=True)
```

5. Kód: Paraméterek beállítása

Mozdulatsor Rögzítése: A felhasználó által indított rögzítés során minden képkockán kinyerem a kulcspontokat, vagy nullákkal töltöm fel az adatokat, ha a kéz nem detektálható. Ez biztosítja, hogy minden mozdulatsor azonos formátumú legyen, függetlenül a kéz jelenlététől.

```
sequence = []
for _ in range(sequence_length):
    ret, frame = cap.read()
    frame = cv2.flip(frame, 1)
    image_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(image_rgb)

    if results.multi_hand_landmarks:
        hand_landmarks = results.multi_hand_landmarks[0]
        landmarks = [lm.x, lm.y, lm.z for lm in hand_landmarks.
            landmark]
        sequence.append(landmarks)
    else:
        sequence.append([0] * 63)
```

6. Kód: Mozdulatsor rögzítése

Mozdulatsor Mentése: A teljes szekvenciát egy `NumPy` tömbbe konvertálom, és fájlként elmentem. Ez a lépés biztosítja az adatok egységes tárolását és a fájlok gyors betöltését a modell betanítása során.

```
import numpy as np

sequence_path = f'{data_folder}/gesture_{sequence_counter}.npy'
np.save(sequence_path, np.array(sequence))
```

7. Kód: Mozdulatsor mentése

4.2 Adatgyűjtési Protokoll

Az adatgyűjtés során különös figyelmet fordítottam a következőkre:

- **Változatos Kézpozíciók:** A kéz távolságát és pozícióját változatosan tartottam a robusztus adatgyűjtés érdekében.
- **Környezet Stabilitása:** Egységes, jól megvilágított környezetben végeztem a felvételeket.
- **Hibás Adatok Szűrése:** Az adatokat rögzítés után átnéztem, és eltávolítottam a hibás vagy hiányos adatokat.

Ezek a lépések biztosították, hogy a modell számára megbízható, pontos és változatos adatokat gyűjtsek mind statikus, mind dinamikus gesztusokhoz.

4.2.1 Adatok egységesítése és padelés

A gyűjtött mozdulatsorok hosszúsága változó, ezért a modell számára előkészítve minden mozdulatsort az egységes `max_sequence_length` hosszra padellek. Ehhez a `pad_sequences` függvényt használom, amely nullákkal egészíti ki a rövidebb mozdulatsorokat, biztosítva, hogy minden bemenet azonos hosszúságú legyen. Ezáltal a modell fix méretű bemeneteket kap, ami segíti az LSTM rétegeket a hatékony feldolgozásban, és minimalizálja az időbeli eltérésekből adódó hibákat.

```
1 from tensorflow.keras.preprocessing.sequence import pad_
     sequences
2
3 sequences_padded = pad_sequences(sequences, maxlen=max_sequence_
     length, dtype='float32', padding='post', truncating='post')
```

8. Kód: Adatok egyesítése és padelése

4.2.2 Normalizálás és címkekódolás

Az adatok feldolgozásának részeként a kulcspontok háromdimenziós koordinátáit normalizálom a `StandardScaler` segítségével, amely a koordinátákat középértékükre és szórásukra standardizálja [31]. Ez a lépés szükséges ahhoz, hogy a modell számára könnyen feldolgozható adatokat biztosítsak, javítva ezzel a tanulási folyamatot. A gesztusokhoz szöveges címkéket rendeltem, például *attack* és *defend*. A címkéket a `LabelEncoder` segítségével numerikus osztályokká alakítom, hogy a modell számára értelmezhető formában adhassam meg a különböző gesztusokat.

```
#Címkék kódolása
le = LabelEncoder()
y_encoded = le.fit_transform(labels)

#LabelEncoder mentése
joblib.dump(le, 'label_encoder.joblib')

#Adatok normalizálása
num_samples, seq_len, num_features = sequences_padded.shape
X_reshaped = sequences_padded.reshape(-1, num_features)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_reshaped)

X_scaled = X_scaled.reshape(num_samples, seq_len, num_features)

#Skálázó mentése
joblib.dump(scaler, 'scaler.joblib')

#Label map létrehozása és mentése
label_map = {label: index for index, label in enumerate(le.
    classes_)}
joblib.dump(label_map, 'label_map.joblib')
```

9. Kód: Normalizálás és címkekódolás

5 A MODELL BETANÍTÁSA

Ebben a fejezetben részletesen bemutatom a modell betanításának főbb lépéseit, kezdve az adatok betöltésétől és előfeldolgozásától, folytatva a modell architektúrájának kialakításával és fordításával, egészen a tanulási folyamat és az elért eredmények kiértékeléséig. Kiemelten foglalkozom a tanulási folyamat optimalizálásával, valamint a létrehozott modell mentési és későbbi felhasználási lehetőségeivel.

5.1 Adatok Betöltése és Előfeldolgozása

Az adatok előkészítése során a statikus és dinamikus gesztusok különböző módon kerülnek betöltésre, mivel eltérő adatstruktúrákban tárolódnak. Ez a megkülönböztetés szükséges annak érdekében, hogy mindkét típusú adat megfelelően előkészíthető legyen a modell számára.

A **statikus gesztusok**[32] esetében az adatok JSON formátumban tárolódnak, amelyek a *MediaPipe* által generált 21 kulcspont háromdimenziós koordinátáit (`x`, `y`, `z`) tartalmazzák. Ez lehetővé teszi, hogy a statikus képkockákban rögzített gesztusokhoz kapcsolódó adatokat egyszerűen betöltsük és feldolgozzuk. Mivel a statikus gesztusok nem rendelkeznek időbeli dimenzióval, a modell számára egy mesterséges mozdulatsort hozok létre azáltal, hogy a kulcspontok koordinátáit többször megismétlem. Ez biztosítja, hogy a modell időbeli dimenzióval rendelkező bemenetet kapjon, amely kompatibilis az LSTM rétegekkel. Az alábbi kódrészlet szemlélteti a statikus gesztusok betöltését:

```
def load_static_gestures(folder_path, label, sequence_length):
    sequences = []
    labels = []
    for filename in os.listdir(folder_path):
        if filename.endswith('.json'):
            filepath = os.path.join(folder_path, filename)
            with open(filepath, 'r') as f:
                landmarks = json.load(f)
                coords = []
                for lm in landmarks:
                    coords.extend([lm['x'], lm['y'], lm['z']])
                sequence = [coords] * sequence_length  # Ismétlés az időbeli dimenzióhoz
                sequences.append(sequence)
                labels.append(label)
    return sequences, labels
```

10. Kód: Statikus gesztusok betöltése

A **dinamikus gesztusok** ezzel szemben időbeli dimenzióval rendelkező mozdulatsorokat tartalmaznak[33], amelyeket több egymást követő képkockán rögzítettem. Ezeket az adatokat `NumPy` tömbök formájában tárolom a már korábban a 4. fejezetben említett módon. A `.npy` formátum előnye, hogy hatékonyan tárol nagy méretű numerikus adatokat, és a NumPy könyvtár segítségével gyorsan betölthető és manipulálható. Ez különösen hasznos a dinamikus gesztusok esetében, ahol egyetlen gesztus sok képkockából áll, és az adatstruktúra nagy mennyiségű koordinátát tartalmaz. Az alábbi kódrészlet bemutatja a dinamikus gesztusok betöltését:

```
def load_dynamic_gestures(folder_path, label):
    sequences = []
    labels = []
    for filename in os.listdir(folder_path):
        if filename.endswith('.npy'):
            sequence = np.load(os.path.join(folder_path,
               filename))
            sequences.append(sequence.tolist())
            labels.append(label)
    return sequences, labels
```

11. Kód: Dinamikus gesztusok betöltése

A különbség abból adódik, hogy a statikus gesztusok esetében egyetlen képkocka adatait ismétlem meg az időbeli dimenzió[34] biztosítása érdekében, míg a dinamikus gesztusok már eleve időbeli adatokat tartalmaznak. A `.npy` formátum alkalmazása a dinamikus gesztusoknál nemcsak hatékony tárhelykezelést biztosít, hanem a NumPy könyvtár gyors betöltési képességeit is kihasználja. Ez különösen fontos nagyobb adathalmazok esetében, ahol az adatfeldolgozás gyorsasága jelentősen befolyásolja a tanulási folyamat teljesítményét.

5.2 Modell létrehozása, rétegei és fordítása

A modell fő komponensei az LSTM (Long Short-Term Memory) rétegek, amelyek az időbeli függőségek modellezésére alkalmasak. Az LSTM rétegeket azért integrálom a hálóba, hogy hatékonyan felismerjék a mozdulatsorok egymást követő képkockáinak időbeli kapcsolatait. Az LSTM struktúrája lehetővé teszi, hogy a réteg hosszabb ideig fenntartsa a korábbi képkockák információit, ami javítja a pontosságot a hosszabb mozdulatsorok esetén [30]. A modell fordításához az `adam` optimalizálót használom, amely adaptív tanulási rátát biztosít a tanulási folyamat során [35]. Az osztályozási feladathoz sparse-categorical-crossentropy veszteségfüggvényt választok, mivel több osztály közötti megkülönböztetésre van szükség.

A háló fő rétegei (lsd. 12. kódrészlet) a következők:

- **Masking réteg**: Az LSTM réteg előtt egy Masking réteget alkalmazok, amely figyelmen kívül hagyja a padeléssel keletkezett nulla értékeket. Ez biztosítja, hogy a padelés miatt hozzáadott nullák ne befolyásolják a tanulási folyamatot.
- **LSTM rétegek**: Két LSTM réteget építek be a modellbe. Az első LSTM réteg `return_sequences=True` beállítással rendelkezik, hogy minden időlépésnél továbbadja az információt, ezzel segítve a következő réteg tanulási képességeit. A második LSTM réteg már csak a teljes sorozat végén ad vissza egy összegző vektort.
- **Dropout rétegek**: A túltanulás elkerülése érdekében minden LSTM réteg után Dropout rétegeket helyezek el. Ezek a rétegek véletlenszerűen lenullázanak bizonyos neuront az edzés során, ezzel biztosítva, hogy a hálózat ne tanulja meg túlzottan a konkrét edzési mintákat.
- **Kimeneti Dense réteg**: A háló utolsó rétege egy Dense (sűrű) réteg, amely a gesztusok számának megfelelő neuronokból áll, `softmax` aktivációs függvénnyel. Ez a réteg a különböző gesztusok osztályozására szolgál, és a kimenetében minden osztályhoz egy valószínűségi értéket rendel.

A hálózat felépítése az alábbi kódrészletben látható:

```
model = Sequential()
model.add(Masking(mask_value=0.0, input_shape=(max_sequence_
    length, num_features)))
model.add(LSTM(128, return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(128, return_sequences=True))
model.add(Dropout(0.5))
model.add(GlobalAveragePooling1D())
model.add(Dense(num_classes, activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_
    crossentropy', metrics=['accuracy'])
```

12. Kód: Modell felépítése

5.3 Betanítás

A betanítást 32 epoch-on keresztül végeztem, és osztálysúlyokat alkalmaztam a kiegyensúlyozatlan adatok kezelésére. Az alábbi kódrészlet bemutatja a folyamatot:

```
history = model.fit(
    X_train, y_train,
    epochs=32,
    batch_size=16,
    validation_data=(X_test, y_test),
    class_weight=class_weights
)
```

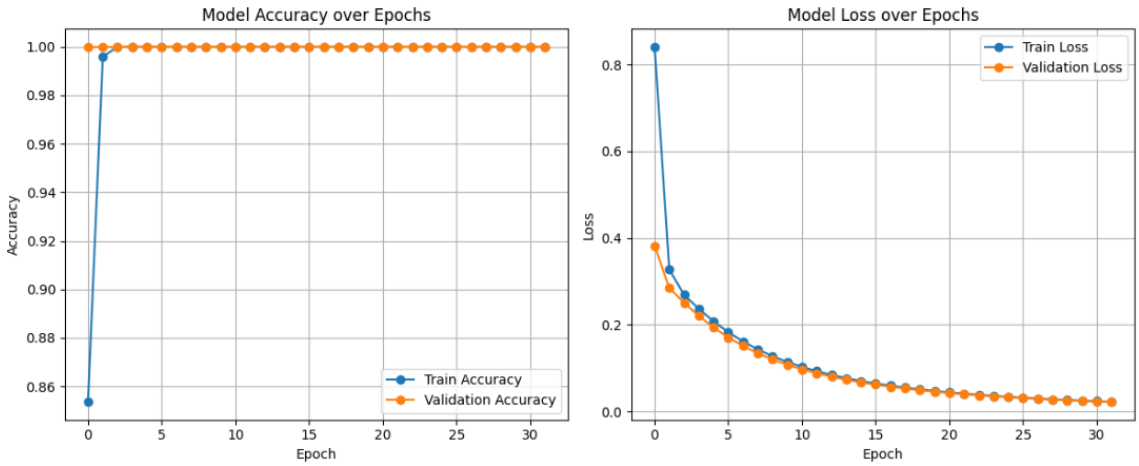
13. Kód: Modell betanítása

5.4 Tanulási eredmények és teljesítmény

A betanítás során a pontosság folyamatosan javult, és a validációs adatokon mért pontosság a 3. epoch végére (5.1 ábra) elérte a 100%-ot. A gépi tanulási modellek betanítása során a 100%-os pontosság elérése ritka, különösen akkor, ha az adathalmaz komplex, vagy a feladat osztályai átfedésben vannak egymással. Azonban ebben az esetben a kiemelkedően magas pontosság több tényezőre vezethető vissza:

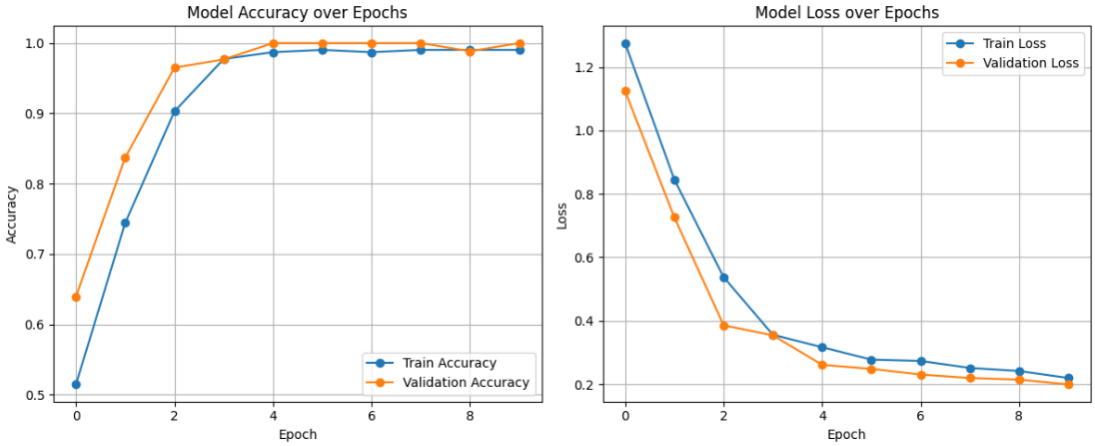
- **Kis adathalmaz:** A viszonylag kevés mintából álló adathalmaz lehetővé teszi, hogy a modell könnyen "megtanulja" a minták jellemzőit, ami gyors tanulást eredményezhet. Ez ugyanakkor megnöveli az overfitting kockázatát is.
- **Kevés osztály:** A felismerendő gesztusok alacsony száma (pl. három különböző gesztus: *attack*, *defend*, *circle_around*) szintén megkönnyíti a modell számára az osztályok közötti különbségek felismerését.
- **Jól elkülönülő gesztusok:** Ha a gesztusok vizuális és mozgási mintázatai jelentősen különböznek egymástól, a modell számára egyszerűbb lehet az osztályok közötti különbségeket felismerni és megfelelően osztályozni azokat.

Ezért a modell jelenlegi pontossága annak tudható be, hogy a gesztusok jól elkülöníthetőek, a feladat egyszerűbb a gépi tanulás számára, és a validációs adatok feldolgozása megfelelően biztosította az adatszivárgás elkerülését.



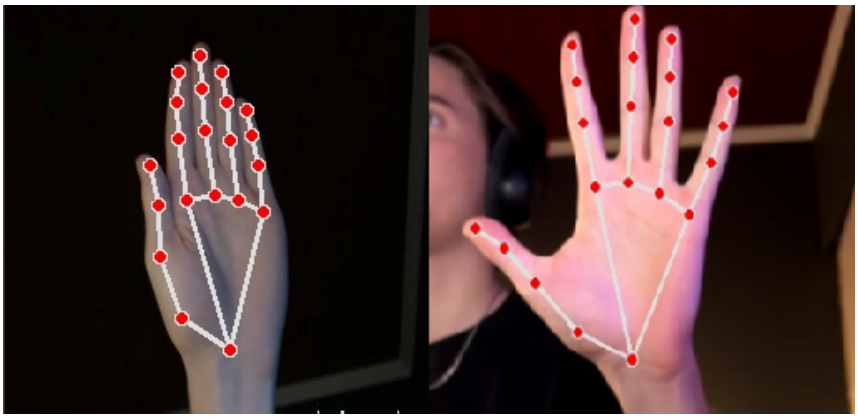
5.1. ábra: Tanítási folyamat kiértékelése a 100%-ot elért modellnél

Mivel a modell a fenti feltételek miatt túl könnyen érte el a 100%-os pontosságot, a következő kísérlet során nehezítettem a feladatot. Ennek érdekében hozzáadtam egy újabb gesztust, amely szinte teljesen megegyezik az *attack* gesztussal, így a modell számára már nem triviális a két mozdulat megkülönböztetése. Az újraindított tanítási folyamatban a modell pontossága így már nem éri el a 100%-ot, hanem körülbelül 99%-nál megáll, ekkor pedig a beépített *stoploss* függvény (egy paraméterezett EarlyStopping callback) leállítja a tanítást. Ez biztosítja, hogy a modell ne futtassa tovább feleslegesen az epokszámot, így időt és erőforrást spórolunk. Az új modell pontossága az 5.2 ábrán látszik.



5.2. ábra: Tanítási folyamat kiértékelése a kísérletben

Ezzel a kísérlettel sikeresen bizonyítom, hogy a 100% nem egy hiba, hanem annyira jól elkülöníthetők a gesztusok ebben az esetben, hogy nem okoz a modellnek problémát hibátlanul betanulni őket. A különbség a két gesztus között az 5.3 ábrán látható.



5.3. ábra: A két egymáshoz hasonló gesztus bemutatása

```
# Plot acc
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy',
    marker='o')
plt.plot(history.history['val_accuracy'], label='Validation
    Accuracy', marker='o')
plt.title('Model Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss', marker='o'
    )
plt.plot(history.history['val_loss'], label='Validation Loss',
    marker='o')
plt.title('Model Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

14. Kód: A kitértékelést megvalósító python kód

5.5 Modell mentése és későbbi felhasználása

A betanított modellt és az előfeldolgozási eszközöket (`LabelEncoder`, `StandardScaler`) elmentettem, hogy a későbbi predikciók során felhasználhassam. Az alábbi kódrészlet mutatja a mentési folyamatot:

```
model.save('gesture_recognition_model_keras.keras')
joblib.dump(le, 'label_encoder.joblib')
joblib.dump(scaler, 'scaler.joblib')
```

15. Kód: Modell mentése

6 TESZTELÉS

A gesztusfelismerő rendszer tesztelését két fázisban végeztem. Az előzetes funkcionalitást egy Python-alapú tesztkörnyezetben ellenőriztem, majd a végső megoldást Unity környezetbe integrálva teszteltem. Ez utóbbi biztosította a rendszer valós idejű működésének pontos kiértékelését.

6.1 Előzetes tesztelés pythonban

Az előzetes tesztelés célja az volt, hogy ellenőrizzem a modell, a MediaPipe alapú kézdetektálás és a predikciós folyamat hibátlan működését. Egy Python-alapú scriptet készítettem, amely a webkamera képkockáit valós időben dolgozza fel. Az alábbi folyamatokat hajtja végre:

- Kéz kulcspontjainak detektálása a MediaPipe segítségével.
- A kulcspontok koordinátáinak skálázása a betanítás során mentett StandardScalerrel[36].
- Az adatok predikciós ablakba (mozgó ablak) helyezése, amely biztosítja a megfelelő hosszúságú mozdulatsorok elemzését.
- A modell predikciójának kiértékelése és a gesztusok valós idejű felismerése.

Az alábbi kódrészlet mutatja a Python script fő részleteit:

```
if results.multi_hand_landmarks:
    hand_landmarks = results.multi_hand_landmarks[0]
    landmarks = []
    for lm in hand_landmarks.landmark:
        landmarks.extend([lm.x, lm.y, lm.z])

    landmarks_scaled = scaler.transform([landmarks])[0]
    window_seq.append(landmarks_scaled)
else:
    window_seq.append(np.zeros(num_features))
if len(window_seq) == sequence_length:
    sequence = np.array(window_seq).reshape(1, sequence_length,
       num_features)
    prediction = model.predict(sequence)
    predicted_class = np.argmax(prediction)
    predicted_label = le.inverse_transform([predicted_class])[0]
    confidence = prediction[0][predicted_class]
```

16. Kód: Előzetes tesztelés Pythonban

Ez a megközelítés megfelelőnek bizonyult az algoritmus pontos működésének ellenőrzésére, de nem teremtett valós idejű, komplex környezetet a teljes rendszer teszteléséhez.

6.2 Valós Tesztkörnyezet Unity-ben

A valós idejű tesztkörnyezet célja a rendszer életszerű szituációkban történő vizsgálata és finomhangolása. Ennek érdekében a gesztusfelismerő rendszert egy interaktív Unity-szimulációba integráltam, amely lehetővé teszi a valós idejű visszacsatolást és a rendszer funkcionalitásának tesztelését különböző környezetekben. Az architektúra két fő komponensből áll: egy Flask-alapú Python szerverből, amely a képfeldolgozást és a gesztusok felismerését végzi, illetve egy Unity mini alkalmazásból, amely valós időben kommunikál a szerverrel.

Ez a megközelítés lehetővé teszi:

- A valós idejű adatfeldolgozás optimalizálását, különös tekintettel a képkockák gyors átvitelére és feldolgozására.
- Az interakciós hibák azonnali azonosítását és kijavítását.
- A gesztusfelismerő modell valós helyzetekben történő tesztelését, például dinamikus háttérrel vagy változó fényviszonyokkal.

A következőkben részletesen ismertetem a Flask alapú szerver működését, valamint a Flask és Unity közötti kommunikáció implementációját.

6.2.1 Flask Alapú Szerver

A Python szerver biztosítja a képek feldolgozását, a gesztusok felismerését és a predikciók visszaküldését Unity-be. A Flask alkalmazás fogadja a Unity-ből érkező képkockákat, és az alábbi folyamatokat hajtja végre:

- A képkockák előfeldolgozása és kulcspontok detektálása MediaPipe segítségével.
- A mozdulatsor ablakba helyezése és a modell predikciójának meghívása.
- A gesztus és a predikciós bizonyosság visszaküldése Unity-be.

Az alábbi kódrészlet mutatja a Flask szerver kulcsfontosságú részleteit:

```
@app.route('/predict', methods=['POST'])
def predict():
    frame = preprocess_image(file.read())
    image_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(image_rgb)

    if results.multi_hand_landmarks:
        hand_landmarks = results.multi_hand_landmarks[0]
        landmarks = [lm.x, lm.y, lm.z for lm in hand_landmarks.
            landmark]
        landmarks_scaled = scaler.transform([landmarks])[0]
        window_seq.append(landmarks_scaled)
    else:
        window_seq.append(np.zeros(num_features))

    if len(window_seq) == sequence_length:
        sequence = np.array(window_seq).reshape(1, sequence_
            length, num_features)
        prediction = model.predict(sequence)
        predicted_label = le.inverse_transform([np.argmax(
            prediction)])[0]
        return jsonify({'gesture': predicted_label, 'confidence'
            : f"{prediction[0][np.argmax(prediction)] * 100:.2f}%
            "})
    else:
        return jsonify({'gesture': 'Collecting sequence...', '
            confidence': None})
```

17. Kód: Flask szerver gesztusfelismeréshez

6.2.2 Unity és Flask integráció

Unity-ben az OBS Studio által biztosított virtuális kamerát alkalmaztam, amely valós idejű kameraképeket biztosított a feldolgozáshoz. Nem muszáj ezt a megoldást használni, nekem azért volt ez fontos, mert nem volt webkamerám itthon és a telefonból érkező képeket a Unity csak ezen az OBS nyújtotta viruális kamerán keresztül tudta valós időben kezelni. A `RenderTexture` egy olyan speciális textúra, amelybe a kamera által látott kép renderelhető, ahelyett, hogy közvetlenül a képernyőre kerülne. Ez nekünk hasznos, mert a képet további feldolgozásra vagy manipulációra kell előkészíteni. A rögzített képeket tömörítettem `.JPG` formátumba, hogy minimalizáljam a hálózati adatátvitelhez szükséges

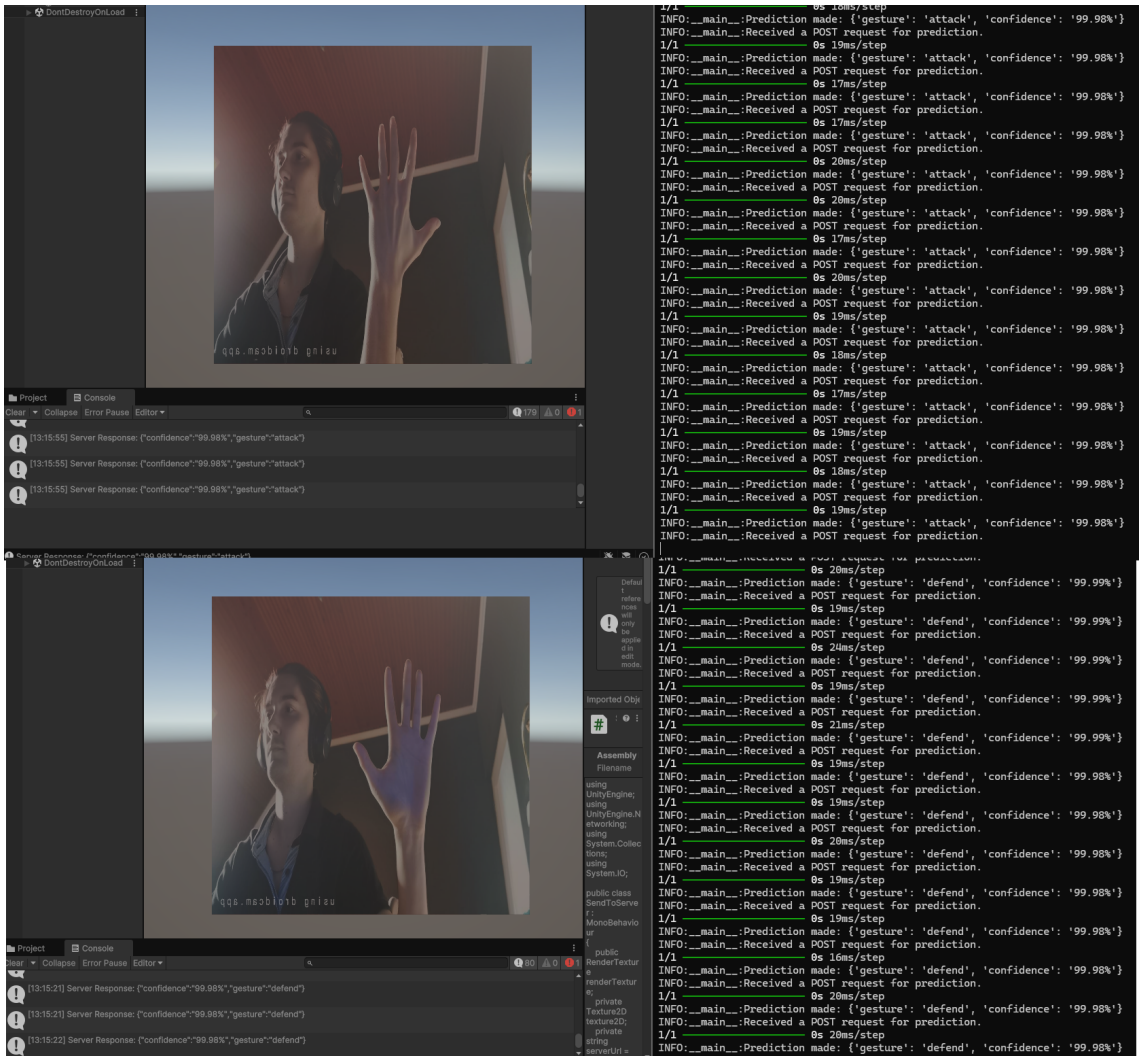
sávszélességet. Az így előkészített képfájlokat egy C# script segítségével HTTP POST kéréssel továbbítottam a Flask szerver felé. A Flask szervertől kapott válaszokat a script feldolgozta, és az eredményeket Unity-ben megjelenítette és valós időben elmentette egy változóban.

Az alábbi kódrészlet bemutatja a Unity-ben használt C# script egy részét:

```
RenderTexture.active = renderTexture;
texture2D.ReadPixels(new Rect(0, 0, renderTexture.width,
    renderTexture.height), 0, 0);
texture2D.Apply();
RenderTexture.active = null;

byte[] imageBytes = texture2D.EncodeToJPG();

WWWForm form = new WWWForm();
form.AddBinaryData("file", imageBytes, "frame.jpg", "image/jpeg"
    );

using (UnityWebRequest www = UnityWebRequest.Post(serverUrl,
    form))
{
     yield return www.SendWebRequest();

     if (www.result == UnityWebRequest.Result.Success)
     {
          string responseText = www.downloadHandler.text;
          Debug.Log("Server Response: " + responseText);
     }
     else
     {
          Debug.LogError(www.error);
     }
}
```

18. Kód: Unity-beli kép küldése a Flask szerverhez

6.3 Eredmények és tanulságok



6.1. ábra: Unity Integráció

Az Unity környezetben végzett valós idejű tesztek (6.1 ábra) bebizonyították, hogy a rendszer képes a gesztusokat pontosan felismerni, és gyorsan reagál a bemenetekre. A 6.1 ábra bal oldalán látható a Unity-ba importált kamera nézet, ami az esetleg lefejlesztett játék esetén el is tüntethető, most csak a jó reprezentatív értéke miatt csináltam így. Az ábra jobb oldalán pedig a Flask szerveren futó modell kiértékeléseit láthatjuk, hogy hogyan kommunikál a Unity kliensünkkel. Az OBS Studio virtuális kamerája és a Flask szerver közötti kapcsolat zökkenőmentesen működött, lehetővé téve a gesztusok valós idejű detektálását és alkalmazását a Unity-beli környezetben.

7 ÖSSZEGZÉS

A dolgozatomban egy olyan rendszert mutattam be, amely képes élő videóból gesztusokat felismerni, különös tekintettel a gépi tanulásra és a neurális hálózatok alkalmazására. A kutatásom során áttekintettem a gesztusfelismerés alapjait, és bemutattam annak jelentőségét különböző területeken, például a virtuális valóságban, a jelnyelv-felismerésben és az ember-gép interakciókban.

A rendszerem megvalósítása során a TensorFlow-t használtam, mivel ez a keretrendszer biztosította a szükséges rugalmasságot és hatékonyságot a gépi tanulási modellek fejlesztéséhez és betanításához. Bár említettem más keretrendszereket, például a PyTorch-ot, ezek a dolgozatomban csak referenciaértékkel szerepeltek. A megoldásom alapját a konvolúciós neurális hálózatok (CNN) képezték, amelyek kulcsszerepet játszottak a videófeldolgozásban és a gesztusok felismerésében.

A rendszert először az adatok gyűjtésével és előfeldolgozásával alapozzam meg. A statikus és dinamikus gesztusok adatgyűjtéséhez pontos protokollokat dolgoztam ki, majd ezeket normalizáltam és címkéztem. A modell felépítése és kompilálása során külön figyelmet fordítottam arra, hogy a rendszer pontosan tudjon alkalmazkodni a változatos gesztusmintákhoz.

A tesztelés Python-alapú környezetben történt, majd a rendszert integráltam egy Unity alapú alkalmazásba is, hogy valós környezetben vizsgálhassam a működését. Az eredmények alapján a rendszer képes volt nagy pontossággal felismerni és értelmezni az emberi gesztusokat, ugyanakkor bizonyos kihívásokra, további fejlesztésekkel kell reagálnom.

Úgy vélem, hogy a dolgozatom egy olyan megbízható alapot nyújt az élő videóból történő gesztusfelismeréshez, amely számos alkalmazási területen felhasználható. A jövőbeli céljaim között szerepel a rendszer robusztusságának növelése, valamint a különböző alkalmazási környezetekhez való még jobb adaptáció biztosítása.

SUMMARY

In my thesis, I presented a system that can recognize gestures from live video, with a focus on machine learning and the use of neural networks. In my research, I reviewed the basics of gesture recognition and demonstrated its importance in various domains, such as virtual reality, sign language recognition and human-machine interaction.

I used TensorFlow to implement my system, as this framework provided the flexibility and efficiency needed to develop and train machine learning models. Although I mentioned other frameworks, such as PyTorch, they were only used as a reference in my thesis. My solution was based on convolutional neural networks (CNN), which played a key role in video processing and gesture recognition.

I first grounded the system by collecting and pre-processing the data. I developed precise protocols to collect data for static and dynamic gestures, then normalized and labeled them. In building and compiling the model, I paid particular attention to the system's ability to adapt accurately to the diverse gesture patterns.

The testing was done in a Python-based environment, and then I integrated the system into a Unity-based application to test its operation in a real environment. The results showed that the system was able to recognize and interpret human gestures with high accuracy, however, I need to address some challenges with further improvements.

I believe that my thesis provides a solid foundation for gesture recognition from live video that can be used in a variety of applications. My future goals include increasing the robustness of the system and ensuring even better adaptability to different application environments.

8 HIVATKOZÁSOK

[1] J.M. Zheng, K.W. Chan, and I. Gibson. Virtual reality. *IEEE Potentials*, 17(2):20–23, 1998.

[2] S.S. Rautaray and A. Agrawal. A novel human computer interface based on hand gesture recognition using computer vision techniques. In *Proceedings of ACM IITM'10*, pages 292–296, 2010.

[3] N. Liu and B. Lovell. Mmx-accelerated realtime hand tracking system. In *Proceedings of IVCNZ*, 2001.

[4] F. Chen, C. Fu, and C. Huang. Hand gesture recognition using a real-time tracking method and hidden markov models. *Image and Vision Computing*, pages 745–758, 2003.

[5] C.S. Lee, S.W. Ghyme, C.J. Park, and K. Wohn. The control of avatar motion using hand gesture. In *Proceedings of Virtual Reality Software and Technology (VRST)*, pages 59–65, 1998.

[6] P.P. Shinde and S. Shah. A review of machine learning and deep learning applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6, 2018.

[7] K. Kavukcuoglu and et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[8] I. Goodfellow et al. Venn diagram of machine learning concepts and classes, 2022. Utoljára megtekintve: 2024-10-04.

[9] S.B. Kotsiantis. Supervised machine learning: a review of classification techniques. *Informatica*, 31:249–268, 2007.

[10] A. Rukhsaar, A.R. Chogule, A.P. Budaragade, V. Pujari, and A. Potdar. A vision - machine learning and deep learning applications. *Zenodo*, 17(10), 2022.

[11] M. Altrichter, G. Horváth, B. Pataki, G. Strausz, G. Takács, and J. Valyon. *Neurális hálózatok*. Panem, Budapest, 2006.

[12] Budapesti Műszaki és Gazdaságtudományi Egyetem. Neurális hálózatok, n.d.

[13] M. Hamdan. Activation functions: Relu, tanh, sigmoid, n.d. Utoljára megtekintve: 2024-10-04.

[14] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, 2012.

[15] P. Ballester and R. Araujo. On the performance of googlenet and alexnet applied to sketches. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016.

[16] S. Feng, L. Zhao, H. Shi, M. Wang, S. Shen, and W. Wang. One-dimensional vggnet for high-dimensional data. *Applied Soft Computing*, 135:110035, 2023.

[17] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, 2022.

[18] Papers with Code. Max pooling. Elérhető itt: `https://paperswithcode.com/method/max-pooling`, Utoljára megtekintve: 2024-05-12.

[19] Bao Ly, Ethan Dyer, Jessica Feig, Anna Chien, and Sandra Bino. Research techniques made simple: Cutaneous colorimetry: A reliable technique for objective skin color measurement. *The Journal of investigative dermatology*, 140:3–12.e1, 01 2020.

[20] Darshita. Superpixels and slic. Elérhető itt: `https://darshita1405.medium.com/superpixels-and-slic-6b2d8a6e4f08`, Utoljára megtekintve: 2024-05-12.

[21] The Theano Development Team. Theano: A python framework for fast computation of mathematical expressions, 2016.

[22] N. Gift. Tensorboard: A visualization suite for tensorflow models, 2017. Elérhető itt: https://towardsdatascience.com/tensorboard-a-visualization-suite-for-tensorflow-models-c484dd0f16cf Utoljára megtekintve: 2024-05-04.

[23] A. Gulli and S. Pal. *Deep Learning with Keras*. Packt, 2017 April. Utoljára megtekintve: 2024-05-06.

[24] TensorFlow. Keras: The python deep learning library, 2024. Elérhető itt: https://www.tensorflow.org/guide/keras Utoljára megtekintve: 2024-05-06.

[25] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020.

[26] Paperspace. Pytorch 101: Understanding graphs and automatic differentiation, n.d. Elérhető itt: https://blog.paperspace.com/pytorch-101-understanding-graphs-and-automatic-differentiation/ Utoljára megtekintve: 2024-05-06.

[27] T.S. Jalolov. Artificial intelligence python (pytorch). *Oriental Journal of Academic and Multidisciplinary Research*, 1(3):123–126, 2023.

[28] H. Adusumalli, D. Kalyani, R.K. Sri, M. Pratapteja, and P.V.R.D.P. Rao. Face mask detection using opencv. In *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, pages 1304–1309, 2021.

[29] Google. Mediapipe: Cross-platform, customizable ml solutions for live and streaming media, 2023.

[30] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python, 2011.

[32] F. Zhang, V. Bazarevsky, A. Vakunov, G. Sung, Y. Chang, and M. Grundmann. Mediapipe hands: On-device real-time hand tracking. *arXiv preprint arXiv:2006.10214*, 2020.

[33] S. Van Der Walt, S.C. Colbert, and G. Varoquaux. *Guide to NumPy*. Trelgol Publishing, 2011.

[34] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[35] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. *StandardScaler — scikit-learn 1.6.0 documentation*, 2024. Hozzáférés: 2024-12-10.

9 ÁBRÁK JEGYZÉKE

KÓDRÉSZLETEK