NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Dankó Bence**
2019 Hallgató törzskönyvi száma: **T/004595/FI12904/N**

ÓBUDAI EGYETEM

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

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

Budapest, 2018. december 14.

.....

hallgató aláírása

Tartalomjegyzék

1. fejezet

Bevezetés

Szakdolgozatomban a siketek és nagyothallók által használt jelnyelv felismerésére készítetek olyan megoldást, amely amellett, hogy alkalmas valós időben értelmezni a kézjeleket, rugalmasan fejleszthető, egyszerűen bővíthető adatkészlettel rendelkezik.

Ezen nem funkcionális követelmények figyelembe vételével először áttekintem és röviden elemzem a probléma megoldásának lehetséges módszereit (1.2 fejezet), majd részletesen is bemutatom a megoldásként választott konvolúciós neurális hálózatokat (1.2.6 fejezet).

Felismerve, hogy a feladat egy többosztályú osztályozási probléma, olyan megoldást kell tervezni és megvalósítani, amely hatékonysága a lehető legmagasabb kevés számú osztály és tanítóminta esetén is. Ezért a bemeneti adatok előfeldolgozásának tekintetében két módszer kerül kiértékelésre és összehasonlításra (2.1 fejezet). Az első az úgynevezett „end-to-end deep learning", ahol a bemeneti adatok előfeldolgozás nélkül kerülnek a hálózatba, ezzel a gépi tanulásra bízva a szegmentálás részfeladatát. A másik módszerhez mélységérzékelő szenzor kerül alkalmazásra, ahol a távolsági adatok szolgáltatják a bemenetet.

A neurális hálózatok megvalósítása (2 fejezet) után hasonló feltételeket támasztva 8-8 hasonló betű betanítása következik, majd az eredmények kiértékelésével (2.5 fejezet) megállapításra kerül hogy mely módszerrel, és milyen architektúrájú hálózattal célszerű a munkát folytatni.

A kutatás további részében mélységérzékelő szenzor adatai alapján készített adathalmaz segítségével először a magyar jelnyelvi ujjábécé statikus, mozgást nem tartalmazó kézjelek felismerése valósul meg (3 fejezet). A lehető legmagasabb pontosság elérése érdekében több különböző architektúra összehasonlító elemzése szükséges (3.2 fejezet). Ezután a javasolt hálózati architektúra felépítése, betanítása és az eredmények kiértékelése (3.3 fejezet) következik.

Ahhoz, hogy a rendszer képes legyen a statikus kézjelek mellett a dinamikus kézjelek felismerésére is, először bemutatásra kerülnek a rekurrens (4.1 fejezet) és az LSTM (4.2 fejezet) neurális hálózatok. A dinamikus kézjelekkel kibővített tanítómintahalmaz (4.4 fejezet) felhasználásával betanított hálózat bemutatását és a tanítási, tesztelési eredmények analízisét (4.5 fejezet) követően a dolgozat a fejlesztői (5 fejezet) és felhasználói (6 fejezet) dokumentummal zárul. A fejlesztői dokumentumban bemutatásra kerül a fejlesztés lépése, illetve a program kódja, míg a felhasználói dokumentációban a program funkcióról

és működéséről lesz szó.

1.1. A magyar jelnyelvi ujjábécé

A jelnyelv a siketek és nagyothallók által használt nyelv, mely a vizuális kódrendszerek egyik legfejlettebb verziója. A jelnyelv fogalmának egy közismert változata [1] "A jelnyelv a legtöbb siket ember elsődleges nyelve. A jelnyelv lehetőséget ad arra, hogy a siket emberek ki tudják fejezni magukat és fejleszteni tudják a lehetőségeiket (...)". A jelnyelv a siket közösségben, a siket emberek közötti kölcsönhatás által született és fejlődött ki.

A jelnyelv nem nemzetközi nyelv, minden országnak saját nemzeti nyelve van. A Finn Jelnyelvi Központ kutatási eredménye [1] alapján elmondható, hogy bár a különböző országok jelnyelvei szerkezetileg ugyan hasonlítanak egymáshoz, de a jelnyelvi szókészletüket vizsgálva olyan méretű különbségek fedezhetők fel, hogy különböző jelnyelvekről beszélhetünk. Önálló nyelvként azonban az 1960-as években kezdték el alkalmazni, Magyarországon a 2009. évi CXXV. törvényben emelték a jelnyelvet a hivatalos nyelv szintjére.

Egy jelnyelvi egység nem betűt vagy hangot idéz fel, hanem komplett jelentése van. A jelnyelv kiegészítésére szolgál az ujjábécé. Ennek használata azonban kevésbé elterjedt, ugyanis a jelnyelv használata sokkal gyorsabb. A magyar jelnyelvben kétféle ujjábécét különböztetünk meg: a fonomimikai, mely a siketet között a legnépszerűbb ujjábécé kivitelezése, és a daktil. A legnagyobb különbség a kettő között, hogy a fonomimikában a kéztartás valamely testrészen, de általában arcon történik, illetve a jelentés pontosítása érdekében arckifejezés is járul hozzá. A dolgozatban az ujjábécé, azon belül is a daktil felismerése és szöveggé alakítása a kitűzött cél, ezért az kerül bemutatásra.

A daktil jelrendszer alapja, hogy egy jel egy betűnek felel meg. A daktil nem különbözteti meg a nagy és kisbetűt. Az alapértelmezett ábécéje az angol nyelv betűkészletével egyezik meg, innen adaptálták át magyar változatra.

Egyik legnagyobb előnye a daktil ujjábécének, hogy pontos és gyakran egyértelműbb mintha fonomimikával fejeznénk ki magunkat, ugyanis itt csupán az egyik kezünket használjuk jelelés során. Emellett rendkívül gyors, ugyanis egy perc alatt akár 300 betű jelelését is teszi lehetővé a daktil. Azonban ez még mindig lassabb mintha jelnyelvi gesztusokat alkalmaznánk. Hátrány továbbá, hogy a jelkészlet nyelvenként itt is eltér, viszont létezik egy nemzetközi ujjábécé, melyet a jelelők többsége ismer.

Ujjábécét elsősorban nevek, betűhalmazok, esetleg ismeretlen szakkifejezések visszaadása során alkalmazzák. Ugyanakkor, mivel minden nemzetnek saját jelnyelve van, a külföldiekkel való kommunikációban kifejezetten hasznos tud lenni. Továbbá ujjábécé segítségével jelelhetők azok a kifejezések is, melyekre nincs a jelnyelvben megfelelő jel, ugyanis ilyenkor lehetőségünk van a szavakat betűnként eljelelni.

1.2. Módszertan

A jelnyelvi kézjelek felismerésének problémáját többféle megközelítéssel oldották már meg [2]. A kézjel felismerésének egy általános, elterjedt megközelítési formája a gépi látás alapú azonosítás. Ennek egy lehetséges változata lehet, a kamera képének feldolgozása, a kéz éleinek (kontúrjának) detektálása , majd az így kapott ritka mátrix feldolgozása,

1.1. ábra. A magyar daktil ujjábécé [1]

esetleg előre leprogramozott feltételrendszer segítségével. Ugyanígy az élek detektálásán alapuló módszer lehet a mintaillesztés is, amely esetben az egyes felismerendő mintákról készített egy-egy sablonnal összehasonlításra kerül a feldolgozott kép.

Starner [3] 1998-ban publikált közleményében egy olyan rendszer készítését javasolta az amerikai jelrendszer felismerésére, amelyben a jelelő két, különböző színű kesztyűt visel, ezzel segítve a vizuális szegmentációt.

További gépi látás alapú megoldás lehet a Haar jellemzők [4] használata. A jellemzők a képen szomszédos régiók pixelösszegének differenciáját írják le. A Haar-jellemzőket kaszkádokba gyűjtő, majd ezt azonosításra használó módszer nagy hatékonyságú, robusztus módszernek bizonyult arcok felismerésére.

A jelnyelv felismerés további módszerei lehetnek a szabályalapon nyugvó felismerés [5]. Egy szenzorokkal felszerelt kesztyű viselése során vagy a kamera által készített képen látható kéz szkeletonizációja után keletkező adatok alapján lehetővé válik egy szabályrendszer kialakítása, ami alapján megtörténhet az egyes kézjelek azonosítása.

Ezen az elven többféle, szenzorokkal ellátott kesztyű is készült [6] [7] [8], amelyek bár hatékony technológiai megoldást szolgáltatnak, de erős ergonómiai ellenérv velük szemben, hogy viselésük kényelmetlen lehet hosszútávon.

A fentebbi módszerek sajátossága, hogy minden esetben a bemeneti információk előfeldolgozása szükséges. Mivel a jelnyelvi ábécé több jelből áll, és a felismerendő kézjelek későbbi bővítése nincs kizárva, a szabályok manuális fejlesztése rossz iránynak tekinthető.

Mintaillesztés használata esetén komoly dilemmát okozna annak megoldása, hogy a ke-

resett általános mintától kissé eltérő, de azonos jelet mutatva kissé bizonytalan eredmény, netán gyakran tévesen negatív eredmények jelennének meg. Haar-jellemzők használata esetén ez kizárható, azonban használat előtt a rendszert tanítani kell, azaz elő kell állítani a kaszkádokat tartalmazó leírót. Bár a Haar-jellemzőket alkalmazó, Adaboost alapú Viola-Jones detektor robusztusnak tekinthető, a bemeneti képek méretével arányosan növekszik a szükséges feldolgozási idő [9] más tanuláson alapuló módszereknél – például konvolúciót alkalmazó neurális hálózatoknál – a bemenet méretének hatása redukálható.

A gépi tanulás és kifejezetten a mély tanulás (deep learning) [10] módszere lehetővé teszi azt, hogy a jelek felismerésére előfeldolgozás nélkül, könnyen és gyorsan tanítható, magas precizitású, gyors kiértékelésű rendszer készüljön. A szegmentálás problémájára két lehetséges megoldást is megvizsgálok, és a dolgozatomban kiértékelek.

Az automatikus jelnyelvi feldolgozás ma is népszerű kutatási terület, mi sem bizonyítja ezt jobban, mint olyan innovatív megoldások megjelenése a piacon, mint a SignAll [11] nevű, magyar gyökerekkel rendelkező startup 2016-os megjelenése. A SignAll által fejlesztett megoldás az amerikai jelnyelvről angol nyelvre és fordítva is, angolról jelnyelvre fordít, mindezt kamerák, mikrofonok és kijelzők segítségével. A startup által alkalmazott megoldás sajátossága, hogy a kéz mellett a testtartást és az arckifejezéseket is figyelembe veszi a döntéshozatal során.

Hasonló kétirányú fordítóeszközön dolgozik a KinTrans [12] is, ahol az alkalmazott szenzorok szintén kamerák, eseténként mélységi szenzorokkal együtt.

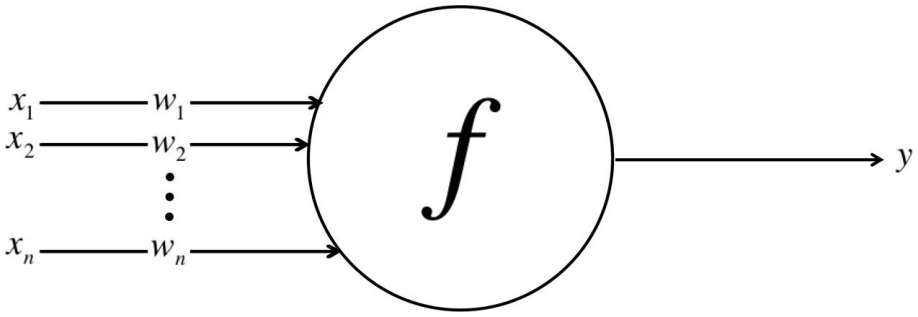
1.2.1. Gépi tanulás

A gépi tanulás [13] [14] széles körben definiálható olyan számítási módszerként, mely tapasztalatot használ a teljesítmény javítása érdekében, vagy, hogy pontos előrejelzéseket, predikciókat tegyen a felhasználó számára. A tapasztalat olyan múltbéli információ, mely elérhető volt a tanuló számára. Az információ lehet digitalizált, ember által jelölt adathalmaz, vagy akár környezettel való interakció során szerzett adat is.

A gépi tanulás lehetővé teszi az olyan feladatok kezelését, melyet túl komplikált lenne megoldani ember által írt hagyományos programmal. Ehelyett, a gépi tanulási feladatok leírják, hogy a rendszernek miként kellene feldolgozni egy példát. A gépi tanuló algoritmusokat két részre lehet osztani az alapján, hogy milyen tapasztalattal rendelkeznek, mennyire informáltak a tanulási folyamat során. Ez alapján megkülönböztetünk felügyelt és felügyelet nélküli tanulást. Felügyelt tanítás esetén a tanítópéldákhoz a feladat megoldásához szükséges információk, válaszok is rendelkezésre állnak, míg a másik esetben a nem áll rendelkezésre információ a megoldásra vonatkozóan.

A gépi tanulást alkalmazó rendszerek egyik legfontosabb és legérdekesebb tulajdonsága, hogy képesek megismerkedni a problémával a tanulási képességüknek köszönhetően, majd elegendő és eredményes tanítást követően képesek a hasonló, ismeretlen problémák megoldására. A tanulási képesség lehetővé teszi, hogy a rendszer a változó körülményekhez való alkalmazkodás céljából módosítani tudja a viselkedését. Ezért ezeket a rendszereket nem egy feladat állandó ellátására tanítják be, hanem arra, hogy képességeik továbbfejlesztésével képesek legyenek alkalmazkodni a változó környezethez, körülményekhez.

Nem felügyelt tanulásnál [15] csupán a bemeneti adatok érhetőek el. A rendszer feladata, hogy a bemenetek és kimenetek alapján a megfelelő viselkedést kialakítsa, ugyanis



1.2. ábra. Neuron felépítése [16]

ilyenkor a környezetből nincs visszajelzés, ami a viselkedés helyességére utal. Nem ellenőrzött tanulás során azt kell felderíteni, hogy a bemeneti információk között felfedezhető e hasonlóság, van e köztük korreláció, léteznek-e kategóriák a bemeneti adatok között. Az ilyen fajta tanulás sokkal jobban hasonlít az emberi tanuláshoz.

Felügyelt tanulás során a tanulási adathalmaz magába foglalja a bemeneti mintákat a hozzá tartozó kimeneti eredménnyel. Durván megfogalmazva a tanítás során az algoritmus megtanulja társítani a bemenet valamilyen kimenettel. Sok esetben a kimeneti adatok összegyűjtése nehezen megvalósítható automatikusan a rendszer által és ezt manuálisan az „oktatónak", azaz a felügyelőnek kell biztosítania. Mivel a be- és kimeneti adatok ismertek, a rendszer válaszát a tanítás során minden esetben össze lehet hasonlítani az elvárt válasszal.

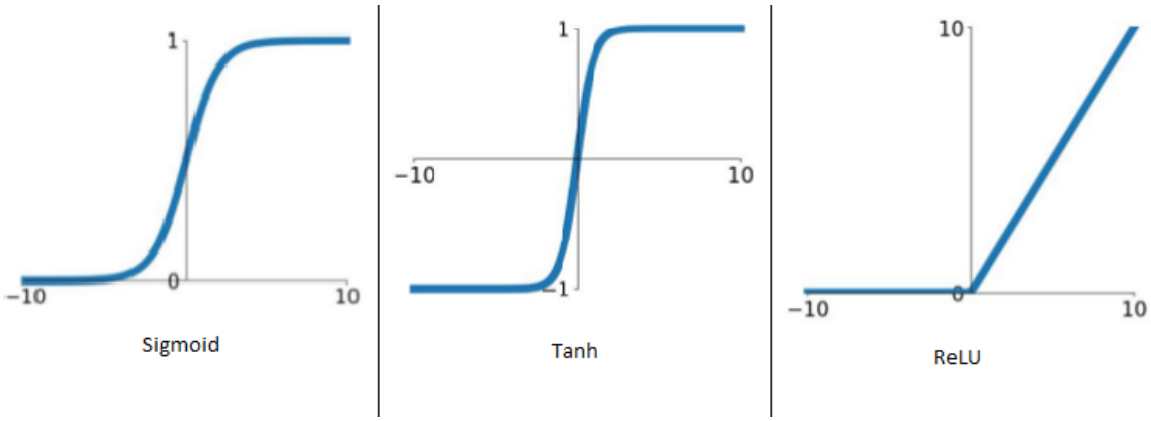
1.2.2. Neurális hálózat

Neurális hálózat [15] egy információfeldolgozó eszköz, mely azonos vagy hasonló típusú lokális feldolgozást végző elemek rendezett topológiája. A neurális hálózatok rendelkeznek tanulási algoritmussal, mely legtöbbször minta alapján való felügyelt tanulást jelent, és amely az információfeldolgozás módját határozza meg. Emellett rendelkeznek egy előhívási algoritmussal is, mely a megtanult információ felhasználását teszi lehetővé.

A neurális hálózat eleme a neuron [16] az információ feldolgozásának egysége, felépítését a biológiai neuron ihlette. Mint a biológia neuronoknál, a mesterséges neuronok esetében is véges számú bemenetet definiálhatunk ($X = \{x_1, x_2 ... x_n\}$), melyek mindegyike egy súlyértékkel ($W = \{w_1, w_2 ... w_n\}$) van megszorozva. A bemenetek súlyozott összege egy eltolás (bias) értékkel együttesen előállítja a lineáris kimenetet (Z):

$$Z = \sum_{i=0}^{\infty} X_i W_i + b. \tag{1.1}$$

A mesterséges neuronnak a biológiai neuronhoz hasonlóan bizonyos bemenetekre „tüzelnie" kell, mások esetén pedig passzívnak maradni. Ennek a működésnek az eléréséhez egy nemlineáris függvényt kell alkalmazni, amelyet aktivációs függvénynek ($f(Z) = Y$) szokás nevezni. Aktivációs függvényekkel szemben két elvárás van. Fontos, hogy az aktivációs függvény nem lineáris függvény legyen, illetve, hogy helyes bemenet esetén a függvény legyen „aktív", különben „inaktív". Egy neuron kimenete továbbítható más neuronokhoz. Az eltolás [17] használatával beállítható az aktivációs függvény küszöbpontja, vagyis, hogy



1.3. ábra. Aktivációs függvények

mikor aktiválódjon a függvény.

Három nemlineáris aktiválási függvény típus a legelterjedtebb a gyakorlatban, [16] ezek a szigmoid (σ), a tangens hiperbolikusz (tanh) és a ReLU függvények. A 1.3. ábrán látható a három függvény görbéje. Szigmoid függvénynek amennyiben Z értéke rendkívül alacsony, a neuron kimeneti értéke közel lesz a 0-hoz. Ellenkező esetben, mikor Z nagyon magas, a kimenet 1-hez közelít. A tanh() függvény görbéje hasonlít a szigmoid görbéjéhez, azonban nullától egyig terjedő értékek helyett a függvény kimeneti értékei -1 és 1 közötti értéktartományon lehetnek. Ezeket a függvényeket jellemzően kimeneti neuronoknál szokás alkalmazni, ahol a szélsőértékek jelentéssel bírnak (0/1 esetén nem/igen, -1/0/1 esetén nem/nem tudni/igen).

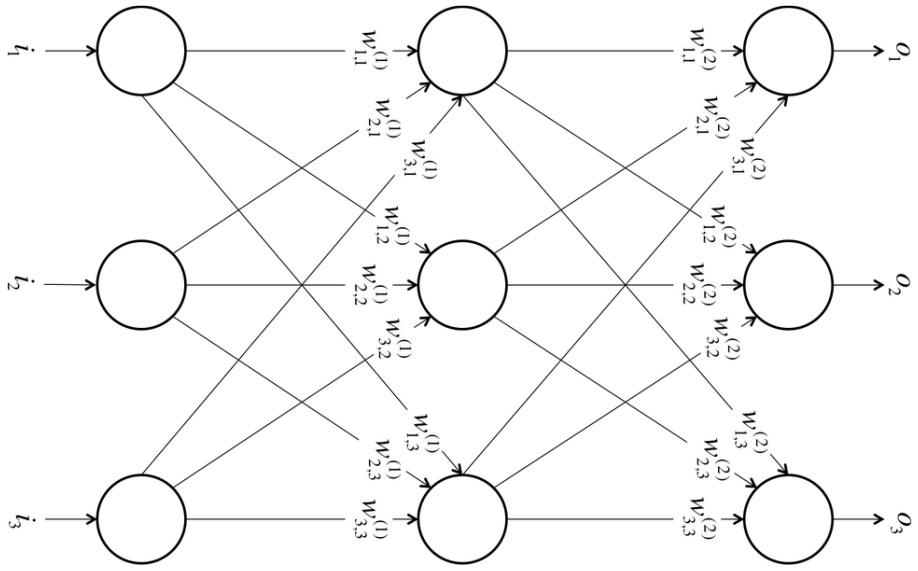
A ReLU (Rectified Linear Unit) egy az előzőektől eltérő nemlineáris aktivációs függvény. A ReLU felírható

$$f\left(x\right) = max(0, x) \tag{1.2}$$

formában. Ez az egyik legtöbbször választott aktivációs függvény a belső rétegek esetén, főleg a gépi látás témakörében [15], ugyanis sokkal gyorsabban konvergál, mint a $\sigma()$ vagy tanh() függvény, és a kimeneti értékek nem telítődnek a pozitív tartományban. Ennek oka a differenciálhatóságnál keresendő: ReLU esetén pozitív tartományban 1 a meredekség, szigmoid, tanh esetén $-\infty, +\infty$ felé közeledve egyre kisebb ez a felszívódó gradeiensek problémája. A ReLU egy módosított változata a „leaky ReLU", melynek számítása költségesebb lehet, mint a ReLU esetében, ugyanis a negatív értékek egy alacsony súllyal kerülnek figyelembe vételre.

Bizonyos esetek megkívánják, hogy a kimeneti vektor egy valószínűségi eloszlás legyen egymást kizáró osztályok felett. Például egy klasszifikáció során a modell nem valószínű, hogy képes 100% magabiztossággal felismerni az osztályozó egy elemét. A valószínűségeloszlás használatával jobban megérthető, hogy a neuron mennyire biztos az előrejelzésében. Ez elérhető egy speciális kimeneti réteg használatával, amit „softmax" aktivációs függvénynek [16] neveznek. A softmax aktivációs függvény megadható, mint

$$f(x_i) = \frac{e^{xi}}{\sum_N^{i=0} e^{x_i}} \tag{1.3}$$



1.4. ábra. Előrecsatolt neurális hálózat 3 réteggel

ahol k az elérhető osztályok száma, és x_i jelenti az egyes bemeneteket. A softmax aktivációs függvény hatására a legnagyobb bemenet kiemelésre, a kisebbek pedig csökkentésre kerülnek, így segítve a helyes klasszifikációt. Egy előrejelzés akkor sikeres, ha a kimeneti vektor egyik elemének értéke közel van 1-hez, míg a vektor többi elemeinek értéke a 0-hoz konvergál.

1.2.3. Felépítése

A neuronok a mesterséges neurális hálózatban rétegekbe vannak rendezve. Az egyes rétegekben különböző számú neuron helyezkedhet el. Az egyes rétegek bemenetei az előző rétegekben található neuronok kimenetei. Az információ a bemeneti rétegen keresztül további feldolgozó rétegekbe kerül, és a kimeneti réteg felé áramlik. Ezt a felépítést előrecsatolt hálónak [14] nevezzük.

A 1.4. ábra egy a többrétegű előrecsatolt neurális hálózatot ábrázol, melyet McCulloch és Pitt [18] dolgozott ki 1943-ban. Előrecsatolt neurális hálózatoknál nincs visszacsatolási kapcsolat, vagyis a modell kimenete önmagába nem térhet vissza. Amennyiben a háló visszacsatolást tartalmaz, akkor visszacsatolt, rekurrens neurális hálóról van szó.

Az előre csatolt neurális hálózatnak minimum két réteggel kell rendelkeznie: egy be- és kimeneti réteggel. Az egymást követő rétegek minden neuronja az azt követő réteg minden neuronjával összeköttetésben áll, azaz a réteg kimenetei a következő réteg neuronjainak bemenetei. A két kiemelt réteg között szabadon választott számú rejtett réteg helyezkedhet el. A rejtett réteg esetén a kimeneti értékek nem érhetőek el a külvilág számára.

Az egyes rétegben található neuronok számának nem kell megegyeznie. Gyakran a rejtett rétegek kevesebb neuronnal rendelkeznek, mint a bemeneti réteg. Szintén nem kötelező, hogy minden neuron kimenete kapcsolódjon a következő réteg összes bemenetére.

Előrecsatolt neurális hálózatok betanításakor leggyakrabban alkalmazott eljárás a hibavisszaterjesztéses tanítás, másnéven backpropagation [14]. Az algoritmus egy optimalizációs eljárás, amely célja a hibafüggvény globális minimumának megtalálása. A hálózat

hibája kiszámítható a kimenet és az elvárt kimenet ismeretében: $L(\widehat{y}, y)$ reprezentálja a költséget (loss), ahol $\widehat{y}$ és y jelenti a kimenetként elvárt és kapott értékeket. A hibavisszaterjesztéshez szükséges kiszámolni a gradienst, amely a hiba szerinti deriváltat jelenti. Ez a kimeneti aktivációs függvény ismeretében megadható.

A backpropagation algoritmus 4 részből épül fel: forward pass, veszteség függvény (loss function), backward pass és a súlyok frissítése. A betanítás elején a hálózat súly értékei véletlenszerű értékek, jellemzően 0 és 1 között. A forward-pass lépés során kerül a háló bemenetére a felcímkézett bemenet, majd végigáramoltatva az információt a hálózaton, kiszámolódik a kimenet. Kezdetben mivel véletlenszerű értékek a háló súlyai, vélhetően nem az elvárt kimenet lesz. Ennek az eltérésnek a mértékét határozza meg a veszteségfüggvény. Többféle veszteségfüggvény létezik. Az egyik legelterjedtebb az MSE (mean squared error), az átlagos négyzetes hiba, amely megadható

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y_i})^2 \tag{1.4}$$

formában.

A tanítás során az a cél, hogy arra a pontra jusson el a háló amikor a bemeneti értékekre adott kimenet eredménye valóshoz közeli eredményt ad, azaz a veszteséget minimalizálni kell. A backward pass célja, hogy megtaláljuk azokat a súly értékeket, amelyek a legjobban növelték a veszteséget, és ezen értékeket módosítjuk a megfelelő módon.

A neurális hálózatok tanításánál fontos szempont, hogy a háló milyen kezdeti súlyértékekből indul ki, ugyanis ezek az értékek jelentősen befolyásolják a tanulást alakulását. A neurális hálók kezdeti súlyértékeinek megválasztására jelen pillanatban nincs explicit megfogalmazható összefüggés, de több ajánlás is olvasható a kapcsolódó szakirodalomban [19].

A hálózat tanulási sebességét a tanulási ráta (learning rate) segítségével lehet befolyásolni. Túl kicsi érték esetén a tanulás sokáig fog tartani, túl nagy érték megválasztása esetén pedig könnyen egy nem optimális eredmény érhető el. A tanulási ráta a hiba csökkenésével arányos.

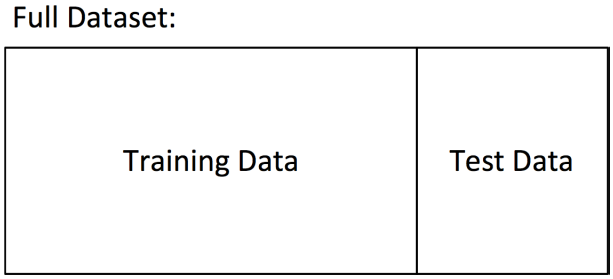
1.2.4. Felügyelt tanítás: klasszifikáció és regresszió

Az ellenőrzött tanuláson alapuló feladatokon belül több feladatcsoportot különböztetünk meg. Ezek közül a legfontosabb a klasszifikáció és a regresszió.

A klasszifikáció célja, az X bemenetekről az Y kimenetekre való leképezés megtanulása. Y egy N elemből álló vektor, ahol az N az osztályok számát jelenti. Amennyiben a Y pontosan 2 elemből áll, bináris klasszifikációról beszélünk. Ha viszont a kimeneti vektor kettőnél több elemet tartalmaz akkor többosztályos klasszifikációról van szó. Bizonyos eseteknél nem lehet egyértelműen eldönteni mi a bemenetre adott kimeneti érték. Ilyenkor a legjobb, ha az egyes elemek valószínűségét jelöli a rendszer, a bemeneti vektor és a tanítási adathalmaz felhasználásával

$$P(y \mid x, D) \tag{1.5}$$

ahol az x a bemeneti vektor, a D a tanítási halmaz. Ezzel a valószínűségi kimenettel már meg lehet határozni a „legjobb találatot”.



1.5. ábra. Példa az adathalmaz felépítésére [14]

A regresszió [20] a felügyelt gépi tanulás egy típusa, amely a folytonos értékű kimenet előrejelzését jelenti. A regresszió azt vizsgálja, hogy miként illeszkedik legjobban egy görbe az adatok összességéhez. A legjobban illeszkedő görbe megad egy olyan modellt annak magyarázatára, hogy az adatkészlet hogyan lett előállítva. A regresszió megpróbálja megérteni az adatpontokat azáltal, hogy felfedjük azt a görbét, amely esetleg generálta őket. Két regressziós modellt [21] különböztetünk meg: A lineáris és nem lineáris. Lineáris regressziónál a regresszorokra adott válasz függését egy lineáris függvény határozza meg, amely matematikailag kiszámíthatóvá teszi a statisztikai elemzésüket. Ezzel szemben a nemlineáris regressziós modellnél ez a függőség egy nemlineáris függvény által van definiálva, ezzel megnehezítve az elemzésüket.

Az osztályozástól eltérően a regressziónál nem elvárt, hogy a kimenet valószínűségi változó legyen, illetve, hogy zárt intervallumon értelmezett legyen a kapott érték.

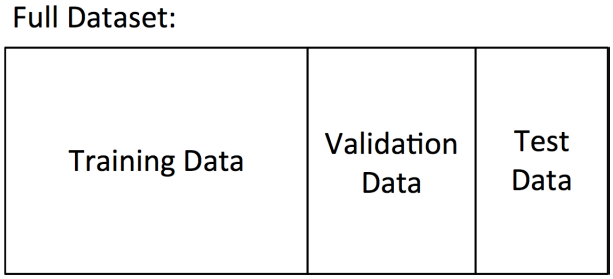
1.2.5. A tanítóminták felbontása

A neurális hálózatok tanításakor az elérhető tanítóminták egészét nem célszerű felhasználni tanításkor bemenetként, legtöbbször szétválasztják őket. Többféle megközelítést is szoktak ilyenkor alkalmazni. Az egyik lehetőség, hogy a tanítómintákat két részre, tanítási- és tesztmintákra osztják.

A tanítási minták felhasználásával történik meg a neurális hálók betanítása, míg a tesztmintákkal az tanult modellt értékelik. Az általános megoszlási arány az ilyen felépítésű tanítási mintáknál [22] , hogy 70% a tanítási a maradék 30% pedig a tesztelési célokra lesz alkalmazva, mely minták véletlenszerűen lesznek kiválasztva. Azonban ez az arány gyakran változik a minták számának függvényében. Nagy mennyiségű tanítóminta esetén kisebb teszthalmaz-arány is megfelelő lehet

A tanítóminták ilyen jellegű felbontása azért szükséges, mert a modell betanításakor a tanítóminták túlzott használatával a modell általánosítóképessége elveszhet, ezt nevezzük túlillesztésnek (overfitting). A túlillesztés jelenségét megfigyelhetjük úgy, hogy a tanításra használt mintákra a rendszer egyre pontosabb kimeneteket produkál, míg a tesztelésre használt adatok esetén egyre rosszabbakat.

Az overfitting megelőzésére több módszer is létezik. Az egyik lehetőség, hogy egyenletesen növeljük a tanítási minták számát, ezzel elősegítve, hogy a háló könnyebben vegye észre a releváns / szignifikáns jellemzőket. Az általánosítás elősegítésére további módszerek is elérhetőek, mint például a „dropout” [23] használata, ahol a tanulás során előre definiált valószínűséggel kimaradnak az adott réteg egyes neuronjai.



1.6. ábra. Adathalmaz 3 részre való osztása [14]

Az overfitting elkerülhető úgy is, ha a tanítás előbb leáll, mint az overfitting folyamata elkezdődne. Erre az egyik lehetséges módszer minden tanítási ciklus végén megmérni, hogy a tanított modell mennyire jól általánosít. Tanítási ciklusok meghatározhatóak úgy, hogy a teljes tanítókészleten való egyszeri iterációkat („epoch") különböztetjük meg.

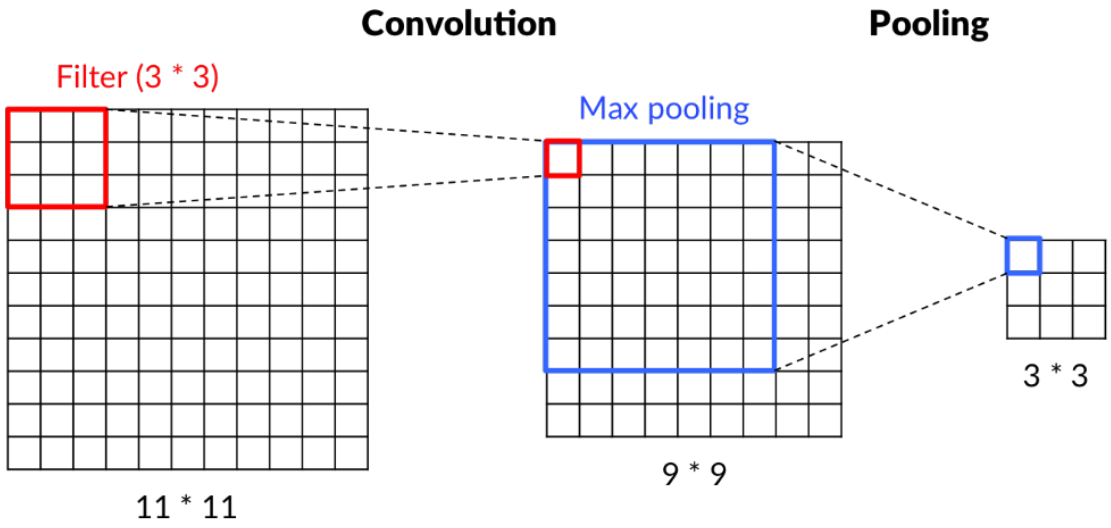
Ahhoz, hogy az egyes epochok végén megmérjük a pontosságot, szükséges egy további adathalmaz, mely validációs mintákat tartalmaz. Az egyes epochok végen a validációs minták fogják megmondani, hogy az epoch végén kapott modell miként viselkedik a validációs mintákon. A betanítása után pedig a teszt halmazban lévő mintákkal értékelik ki a teljes modell pontosságát. A 3 halmaz megoszlási aránya az egész adatkészlet nagyságától függ. 10000 elemből álló halmaz esetén 8:1:1 arány a jellemző, míg milliós nagyságoknál nem ritka a 98:1:1 felbontás, ahol a legtöbb mintát a tanítás során használt halmaz tartalmazza.

Megemlítendő, hogy az egyes epochok során a tanítási folyamatban több bemeneti minta összevonható, és együttesen is kezelhető. Ezeket az összekapcsolt bemeneteket kötegeknek (batch) nevezzük, és a backpropagation algoritmusa együttesen kezeli őket. Erre a processzorok felépítése, és a mátrixműveletek hatékony végrehajtása miatt van lehetőség. Ez indokolja a mély neurális hálók tanításakor a grafikus processzorok (GPU) alkalmazását is.

A nem megfelelő tanulás egy másik megjelenése az alulillesztés, vagy „underfitting". Alulillesztés jelenség akkor tapasztalható, ha a modell nem képes megtanulni a különbségeket és jellemzőket az egyes a tanítási mintákon, ellenben a validációs minták jó pontosságot jeleznek. Maga a jelenség általában akkor szokott előfordulni, mikor arányaiban kevés mintával valósul meg a tanulás egy nagy neuron-és rétegszámú (mély) neurális hálózatban.

1.2.6. Konvolúciós Neuráls Hálózat

A konvolúciós hálózatok (Convolutional Neural Network - CNN) [14] olyan speciális neurális hálózatok, melyek többdimenziós felépítésű adatok feldolgozására alkalmaznak. Ilyen adat lehet például az idősoros adatok, melyet úgy lehet elképzelni, mint egy 1 dimenziós rács, mely rendszeres időközönként veszi fel a mintákat, és a képek, melyekre 2 dimenziós rácsként kell tekinteni. Az elnevezés azt jelzi, hogy a hálózat egy matematikai lineáris műveletet, konvolúciót alkalmaz. A konvolúciós hálók egyszerű neurális hálózatok, melyek legalább az egyik rétegükben konvolúciót alkalmaznak az általános mátrixszorzás helyett.



1.7. ábra. Konvolúciós és pooling réteg

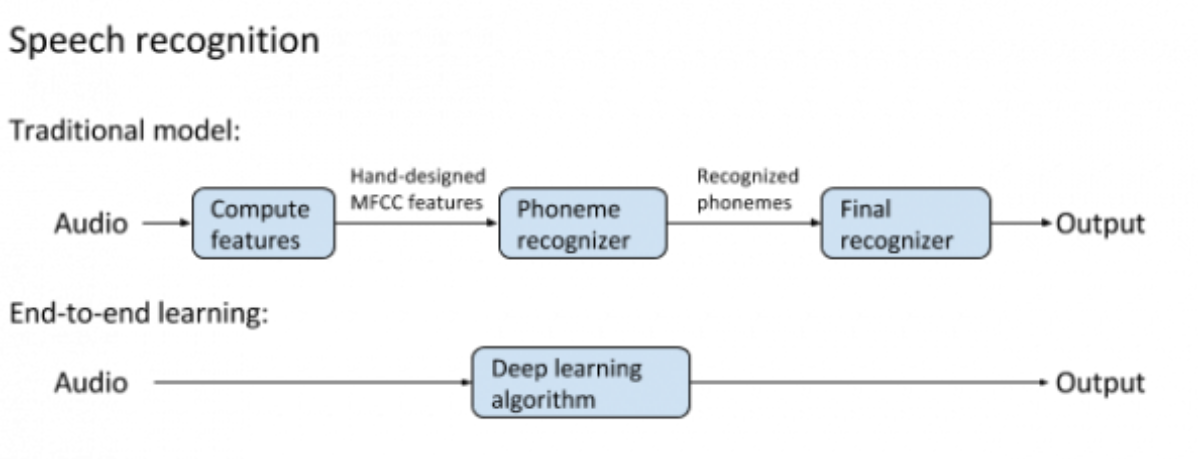
A hagyományos neurális hálózatok nem skálázzák a teljes képet. Például a MNIST [24] adathalmaz képei 28x28 pixel nagyságú fekete fehér képek, ennek eredményeként egy neuron egy teljesen összekapcsolt rétegben összesen 784 paraméterrel kell, hogy rendelkezzen. Ez a mennyiség növekedni fog a kép méretének megnövelésével, illetve, ha több neuron található a rétegekben. Egyértelmű, hogy a teljesen összekötött réteg használata nemcsak pazarló, hanem növeli a túlillesztés esélyét a tanítás során.

Ehelyett a konvolúciós rétegben lévő neuronok az előző réteg egy kisebb régiójával vannak összekötve, ezáltal csökken a paraméterek száma. Így a tanítási idő csökken, illetve a modell tanításhoz szükséges adatmennyiség is kevesebb.

Egy konvolúciós hálózat háromféle rétegtípusból épül fel. A legfontosabb építőelemek a hálózatban a konvolúciós rétegek. A konvolúciós rétegek paraméterei meghatározott számú tanítható szűrőkből (filter, kernel) épül fel. Minden szűrő a képhez képest kisebb méretű, például $2x2$. Ezek a filterek a kapott képből egy úgynevezett aktivációs térképeket hoznak létre. A hálózat a betanulás során a szűrőket oly módon súlyozzá, hogy akkor aktiválódjanak, ha valamilyen éleket, foltokat vagy alakzatokat, esetleg mintákat fedez fel a képen. Egy hálóban több konvolúciós réteg van és mindegyik réteg több szűrőt tartalmaz, melyek egyenként külön-külön két-dimenziós aktivációs térképet állítanak elő.

Egymást követő konvolúciós rétegek közé összevonó, tömörítő (pooling) rétegeket szokás elhelyezni. A pooling rétegek csökkentik a reprezentáció méretét, lecsökkentve ezzel a paraméterek mennyiségét és a számítások mennyiségét a konvolúciós háló architektúrában, kontrollálva ezzel az overfittinget. Két elterjedt pooling függvény a max pooling és az average pooling. Max pooling réteg alkalmazása során az aktivációs térképen meghatározott méretű csúszóablakokon belüli értékek összevonásra kerülnek úgy, hogy egyenlő méretű területre osztja, majd ezután „összevonja” azt, úgy, hogy a felosztott cellákból kiválasztja a legnagyobb értékűeket. Ezek lesznek a kimenet elemei.

A konvolúciós és pooling rétegek száma nincs meghatározva, azonban magas számú alkalmazás esetén a kapott hálózat túlságosan komplex lesz, aminek hatására a betanítás



1.8. ábra. Beszédfelismerés hagyományos és E2E környezetben [26]

több erőforrás igényel, emellett több bemeneti adatra is szükség lesz.

A korábban bemutatott teljesen összekötött (fully connected) rétegeket a konvolúciós rétegek után szokás helyezni. Amennyiben osztályozásra alkalmazzák a hálózatot, akkor a teljesen összekötött réteg kimenete egy N méretű vektor, ahol N az osztályok száma. A kimenet vektorának minden eleme egy-egy osztály valószínűségét reprezentálják, amennyiben softmax függvényt használunk.

1.2.7. End-to-end deep learning

Az úgynevezett „end-to-end deep learning" vagyis a végponttól végpontig terjedő „mély" tanulás [25] jól illeszkedik a gépi tanulás általános megközelítésébe, hogy az emberi beavatkozást kiveszi tanítási ciklusból és tisztán adatvezérelt módon oldja meg a problémát. Az end-to-end tanítás (E2E) egy mély és komplex rendszer betanítását jelenti a gradiens alapú tanulásnak a rendszer egészére történő alkalmazásával. Az ilyen tanulási rendszerek kifejezetten úgy lettek kialakítva, hogy a rendszer minden modulja differenciálható legyen. Ezt az elegáns, mégis egyértelmű és néha a brute force-ra emlékeztető technikát a deep learning összefüggésében népszerűsítették.

A deep learning architektúrák egy természetes következménye a gépi tanulás és más feldolgozó komponens közti klasszikus határok elhomályosítása, egy esetlegesen összetett feldolgozóvezeték bevonásával. A 1.8. ábrán látható egy példa klasszikus és az E2E megvalósításra. A végponttól végpontig tanulás magas szintű automatizáláson alapszik, azonban mégsem teljesen autonóm. A tanuló rendszer felépítése és megtervezése, kialakítása bizonyos szintű tapasztalatot igényel. A deep learning rendszerek modulokból, rétegekből épülnek fel. A tervezési fázis után, a teljes automatizálás átveszi a tanítási feladatot: a paraméterek inicializálva és véletlenszerűen vannak kiválasztva, és ezt követően tanítja őket az E2E rendszer.

Hagyományos neurális hálóknál az összetettebb probléma megoldása úgy történik meg, hogy azokat több részre bontja. Ezt az "oszd meg és uralkodj" megközelítést az E2E kifejezetten figyelmen kívül hagyja. Helyette, a tanítás abban a reményben történik, hogy a strukturális előkészítés elég erős ahhoz, hogy irányítson egy folyamatot a véletlenszerűen kiválasztott állapotól a nem triviális megoldásig [26]. Az ilyen fajta megközelítés veszélyes

lehet, ugyanis a tanítás akkor hatékony, ha a rendszer feladata „szinte triviális”. Az adatok hatékonysága a másik probléma. A modulok közötti nem módosított interakciók szükségessé tehetik egy olyan tanítási minta mennyiséget, amely exponenciálisan nő a modulok számával, hogy a problémát jól minta vételezzek.

1.2.8. A feladat megoldására alkalmazott módszerek

A jelnyelvi ábécé betűinek felismerésére felügyelt tanítású konvolúciós neurális hálózatot terveztem és valósítottam meg. Kutatásom során megvizsgáltam az end-to-end deep learning alkalmazásának lehetőségét, valamint az előfeldolgozott, szegmentált kézjelekkel történő betanítás és felismerés módszerét is.

E2E architektúra esetén a bemenetek a kézjelekről készített színes képek, míg a szegmentálást mélységérzékelővel ellátott kamera távolsági adatai szolgáltatják.

Ahhoz, hogy a későbbiekben a megfelelő típusú tanítóminta legyen alkalmazva a jelnyelv kézjeleinek felismerésére a két különböző módszerrel előállított tanítómintával betanított neurális hálózat lett elkészítve és kiértékelve 8-8 tanítómintára.

A magyar ujjábécé kézjeleire nem létezik használható dataset, ezért a megvalósítás előtt egy saját dataset készítése szükséges. A kézjelek rögzésében több személy is részt vett. Emiatt a tanítóminták ugyanolyan kézjel esetén is változatos lett, mivel minden ember kézfelépítése és mérete eltérő.

2. fejezet

Mélységi adatok alkalmazhatóságának analízise

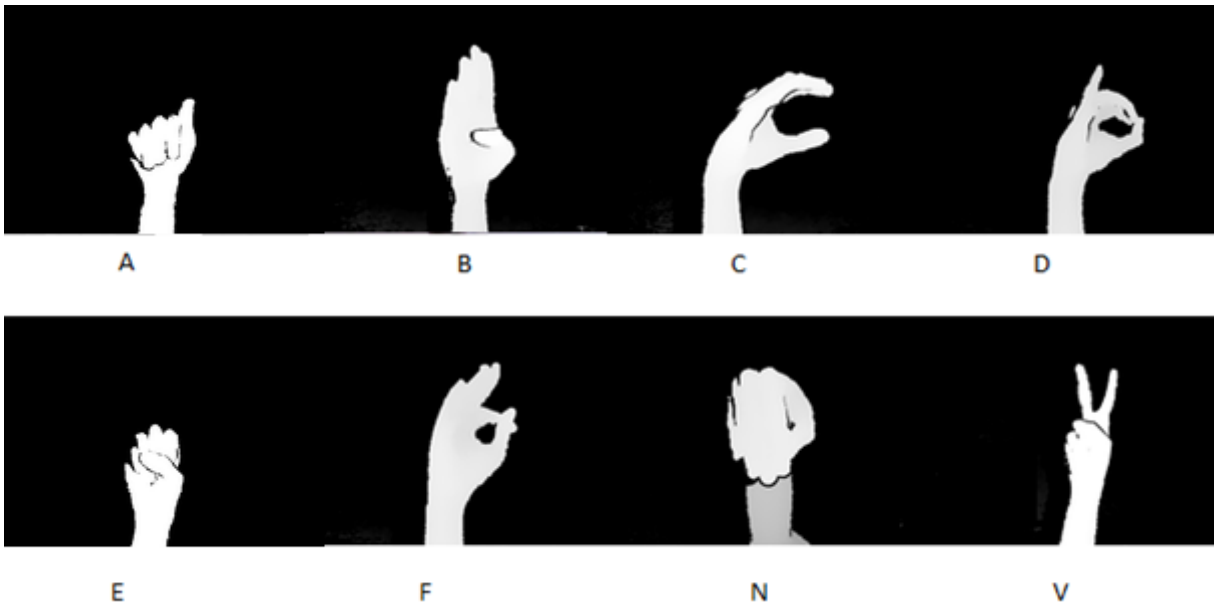
Jelnyelv kézjeleinek felismerésénél fontos szempont, hogy a tanítási mintákon a jelelő kézen kívül ne legyen semmi a képen. Mélységi adatoknál a kéz szegmentálása és háttér kivonás megtörténik amennyiben távolság a kamera és a kéz között megfelelő és küszöbértékek is helyesen vannak megválasztva. Hagyományos módszer során képek előfeldolgozásával érhető el a kéz szegmentálása és a háttér kivonása és utána megtörténhet a jelfelismerése. Ez end-to-end környezetben valósul meg.

A két módszer kipróbálása, több különböző szempont figyelembe vétele miatt volt szükséges. Jelelés során a kézjelek térbeliségének gyakran meghatározó szerepe van. Sok esetben nem mindegy, hogy a jelelő kéz milyen térbeli pozícióban helyezkedik el. A 8 betanított betű közül az „A" és „N" betűhöz tartozó kézjel esetén figyelhető meg legjobban a térbeli megjelenés. A 2.1. és 2.2. ábrán is észre lehet venni, hogy az „A" betű esetén az ujjak teljesen a tenyéren helyezkednek el, míg „N" betű esetén az ujjak inkább a tenyér előtt vannak. A színes képek készítése egy általános módszer, mely nem alkalmas a képek térbeliségét visszaadni úgy, mint a mélységi szenzorból származó adat. Ezért az olyan területeken, ahol a térbeli megjelenés fontos, a színes képpel készült tanítási minták összezavarhatják a neurális hálót, rontják annak pontosságát.

Meg kell említeni azt is, hogy az egyes kézjelek felismerésénél nem csak a kéz a kontúrja játszik meghatározó szerepet. A 2.1. és 2.2. ábrán észre lehet venni, hogy előfordulnak olyan jelek, ahol a kéz közel összezárt állapotban vannak. Ilyenkor a kezek körvonalai hasonlítanak egymáshoz. A 8 betűből az „A", „E" és „N" betűknél figyelhető meg az összezárt kézpozíció. Ebben az esetben az ujjak elhelyezkedése határozza meg, hogy mi is a mutatott kézjel. Színes képek használata esetén az ujjak pozíciója könnyen megkülönböztető. Mélységi képek eseténél ennek megkülönböztetése már bonyolultabb, hiszen ebben az esetben a képek kevésbé részletgazdák, mint a színes képek esetében. Ezért ennek megvizsgálata is szükséges a megfelelő módszer kiválasztásához.

2.1. Tanítóminták

Az első módszer során a neurális hálózat betanítása egyszerű színes, míg a második módszer során a mélységi szenzorból származó adatokkal történik.



2.1. ábra. Tanító minták mélységi képek esetén

A betanításra használt tanítási mintákat a magyar jelnyelv ujjábécé kézjelei adják. A színes képekben az információt a piros, zöld és a kék csatornák (RGB) adatai határozzák meg. A mélységi szenzor kimeneti adatai egyszerű egész értékekként értelmezhetőek, melyek pixelenként 8 biten tárolhatóak.

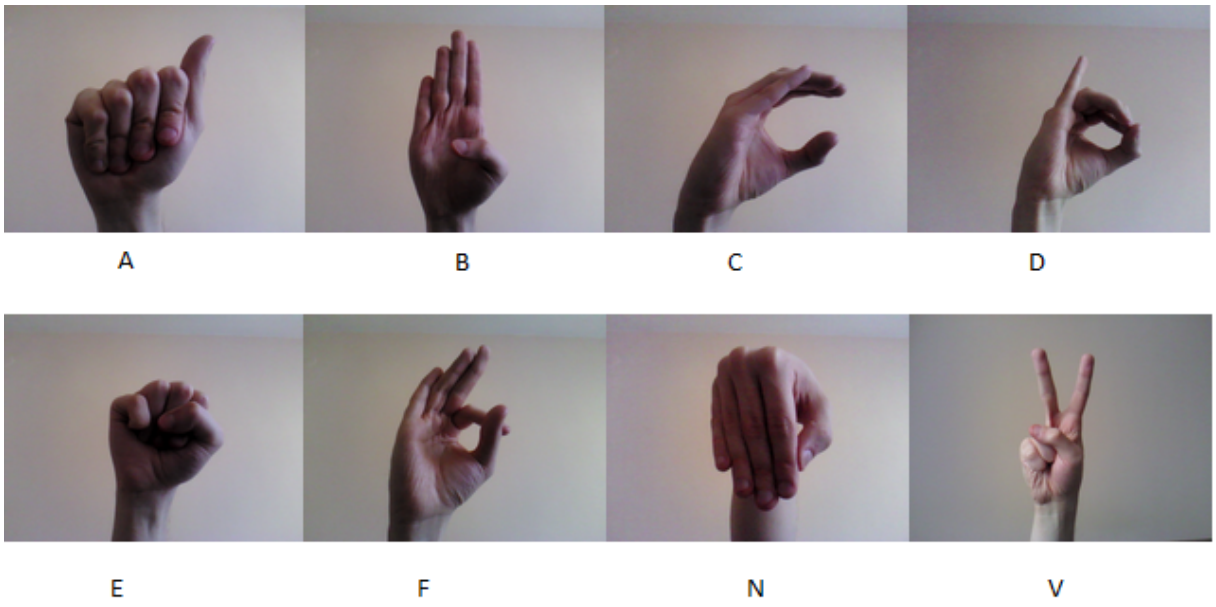
A többrétegű neurális hálózat tervezésekor az egyik megoldandó kérdés, hogy a hálózat bemenetére szánt kép milyen méretű legyen. Elsősorban a bemenet és a rétegek neuronszáma határozza meg, hogy a hálózatban hány paraméter lesz. Ez nemcsak a tanítás idejét és a modell bonyolultságát határozza meg, hanem a rendszer pontosságát is befolyásolja.

A két háló a jelnyelvi ujjábécé 8 kézjelének, az A, B, C, D, E, F, N, V betű, felismerésére képes. A kézjelek kiválasztásánál a cél az volt, hogy minél inkább hasonlítsanak egymásra. Ezáltal könnyebben derül ki, hogy melyik módszer a hatékonyabb. Az értékelés során kiválasztott tanítási mintával fog megtörténni a 36 betűből álló jelnyelvi ujjábécé felismerésére képes neurális hálózat betanítása.

A két módszer tanítómintáinak előállítása egyszerre történt, ezáltal a két hálózat tanítómintái teljes mértékben megegyeznek, csupán a képek előállításának módszere különbözik. Az egyes képek 640x480 pixel felbontással lettek rögzítve. Azonban ez a méret szükségtelenül nagy, ezért a kevesebb bemeneti paraméter elérése érdekében, az egyes képek mérete 160x120 pixelre lett csökkentve. Színes képek esetén 32 bites RGB, míg mélységi képek esetén 8 bites szürkeárnyalatos képek készültek.

2.2. Rétegek

A 8 darab kézjel felismerésére képes neurális hálózat felügyelt tanuláson alapú, többosztályú klasszifikátor. A konvolúciós hálózat felépítése Yann LeCun által megvalósított LeNet [27] konvolúciós neurális hálózat architektúráján alapszik, melyet egyszerű, viszonylag kevés osztállyal rendelkező klasszífikációs probléma esetén szokták alkalmazni. Egy

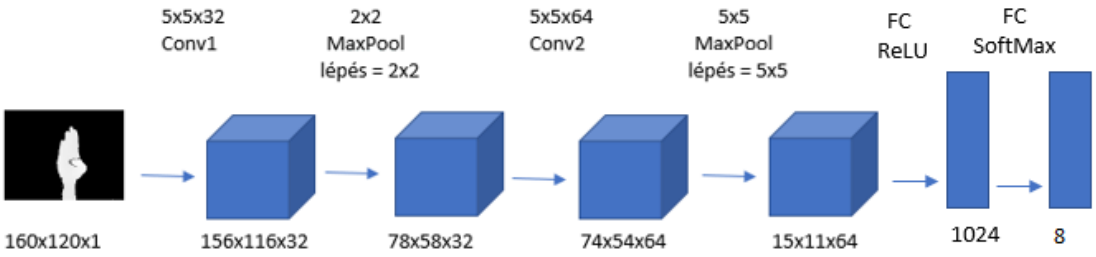


2.2. ábra. Tanító minták színes képek esetén

LeNet-hez hasonló konvolúciós háló a következő módon szokott felépülni: Egy konvolúciós réteg és egy pooling réteg következik, melyet még egy konvolúciós és pooling réteg követ. Végül az architektúra két egymást követő darab teljesen összekötött réteggel zárul, ahol az utolsó réteg aktivációs függvénye softmax. Mindkét hálózat esetén az említett architektúra van alkalmazva, azonban az egyes rétegek mérete meg lett változtatva a bemeneti mátrixhoz igazodva.

A neurális hálózat bemenetként egy 160x120 pixel nagyságú képet vár. Az első rejtett réteg egy konvolúciós réteg, mely 32 darab, 5x5 méretű szűrőből épül fel. A réteg bemenete egy 160x120 darab paraméterből, míg a kimenete 32x156x116 paraméterből áll. Ezt követően egy max pooling réteg következik, aminek mérete 2x2. A rétegen 2x2 lépésköz van alkalmazva, ami miatt a szűrő kettesével fog lépni a réteg bemenetén. A réteghez nincs padding hozzáadva. Padding hozzáadásával a kimenetek térbeli mérete nem fog csökkeni. Ezáltal lecsökken a paraméterek száma. A padding miatt a következő konvolúciós réteg bemeneti paramétereinek száma lecsökken 32x78x58 darabra. Ez a réteg 64 darab, 5x5 méretű szűrővel rendelkezik, melyet egy 5x5 méretű max pooling réteg követ, 5x5 lépésköz használatával. A pooling réteg után a paraméterek száma 15x11x64 lesz. Még a következő réteg előtt a rendszer a 2D adatot átalakítja egy vektorrá. Ez lehetővé teszi, hogy a kimenet feldolgozható legyen egy teljesen összekötött réteggel, aminek a kimenete 1024 neuronból épül fel és az aktivációs függvénye pedig ReLU. Ezen a rétegen egy dropout [23] regularizáció van alkalmazva, mely során a neuronok 40%-t figyelmen kívül hagyja, annak érdekében, hogy az overfitting esélyét csökkentse. A neurális háló utolsó rétege 8 neuront tartalmaz, az aktivációs függvénye softmax. Ezáltal a kimenet értékeivel megvizsgálhatóvá válik az egyes osztályok valószínűsége. A 2.3. ábrán látható a neurális hálózat vizuális megjelenítése, amin jól látható az egyes rétegek közti paraméter változás.

A színes képeket felismerő neurális hálózat esetén az architektúra közel megegyezik a mélységi adatokkal dolgozó hálózattal. Ebben az esetben a hálózat bemenetére 160x120x3



2.3. ábra. Neurális hálózat felépítése mélységi képek esetén

pixel nagyságú képet kap. Ennek hatására a hálózat paramétereinek száma sokkal több lesz, ami a tanítás hosszát is befolyásolja.

2.3. Dataset számossága

Mivel a cél a két különböző módszerrel előállított tanítási mintákkal betanított neurális háló vizsgálata és összehasonlítása, ezért fontos, hogy a tanítási halmaz mérete mindkettő hálózat esetén megegyezzen. Egy kézjelről 500 tanítási minta készül, vagyis a neurális háló betanítása 4000 mintával fog megtörténni mindkét neurális hálózat esetén. A tanítás hatékonyságának növelése érdekében a tanító minták tanító és validációs adathalmazokba lettek szétválogatva. A folyamat során a mintákat a rendszer összekeveri, majd a két halmazba szeparálja őket. Ennek eredményeképp a 4000 darab mintából 3333 tanító adatként szolgál, míg 667 pedig validációs mintaként. Emellett a betanítást követően a rendszer 1000, kézjelenként 125 darab a tanítási és validációs mintáktól független teszt mintával lesz tesztelve.

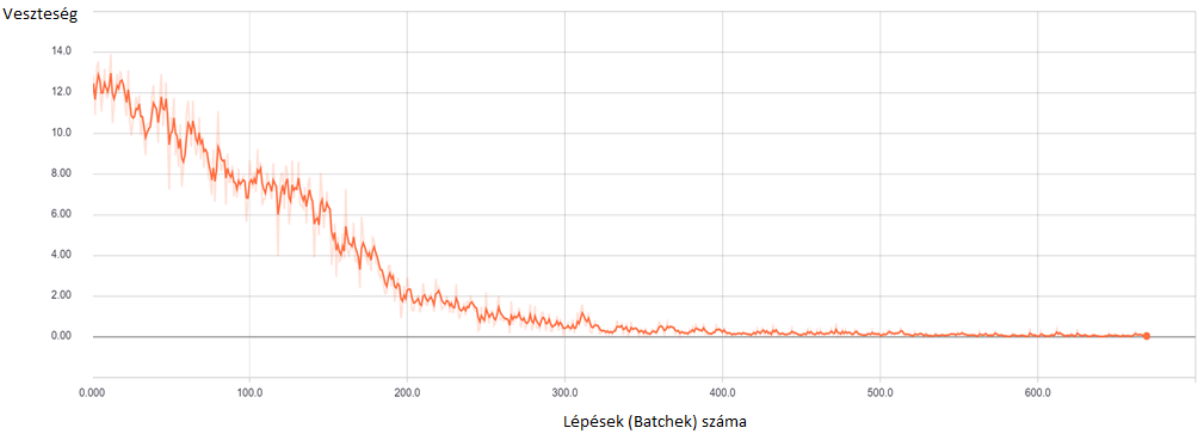
2.4. Konkrét betanítás

A tanítóminták előállítása és a két neurális háló rétegeinek felépítése után a következő lépés a hálók betanítása. Mindkettő neurális háló 0,001 tanulásai rátával lett betanítva. A tanulás során sztochasztikus gradiens csökkentés [28] volt alkalmazva. A batch mérete 50 lett, vagyis a neurális háló bemenetére egyszerre 50 darab véletlenszerűen kiválasztott tanító adat kerül, a teljes tanító halmaz helyett.

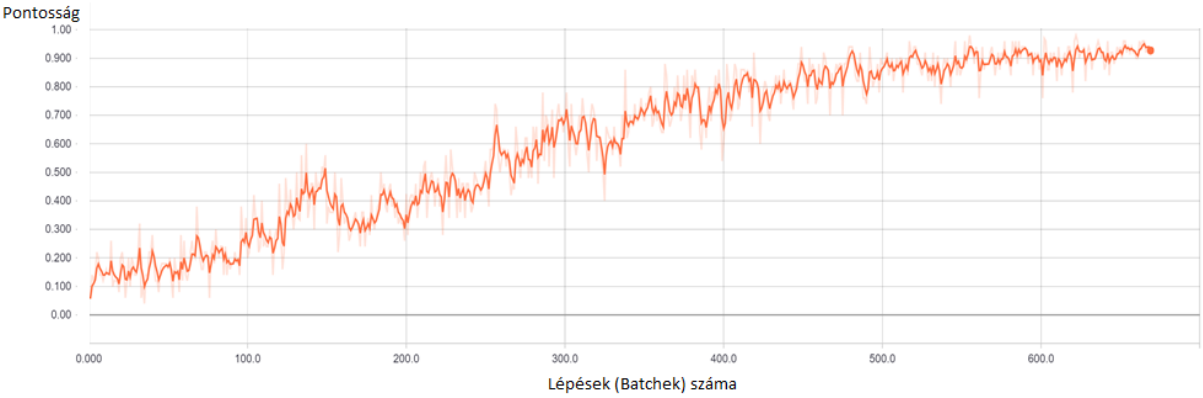
Minden tanítási ciklus (epoch) után a háló a validációs mintákon keresztül ellenőrizte, hogy a háló pontossága mennyit növekedett. Amennyiben 2 epochon keresztül nem nőtt ez az érték a tanítást a rendszer leállította. Ez alapján neurális hálózat 10 epochon keresztül tanul, ugyanis e feletti érték között a rendszer pontossága gyakran már nem, vagy számottevően nem növekedett.

Az 2.4. és 2.5. ábrán látható, hogy miként változik a mélységi adatokkal betanított neurális hálózat pontossága és a veszteség értéke a tanítás során, valamint a 2.6. és 2.7. ábra mutatja be ugyanezt a színes képekkel betanított hálózat esetén.

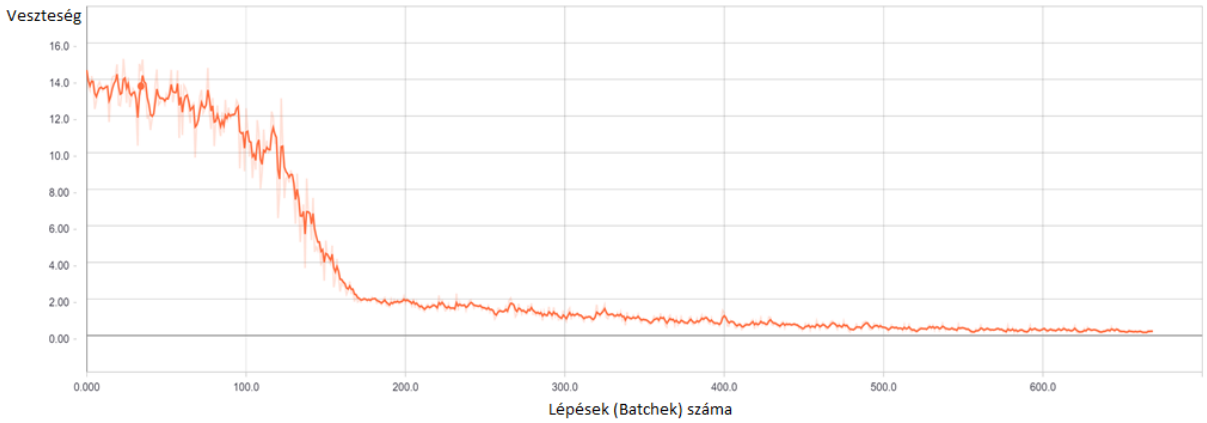
2.4. ábra. Pontosság változása mélységi módszer tanításakor



2.5. ábra. Veszteség változása mélységi módszer tanításakor



2.6. ábra. Pontosság változása RGB módszer tanításakor



2.7. ábra. Veszteség változása RGB módszer tanításakor

2.5. Eredmények értékelése

A két neurális hálózat megfelelő működésének ellenőrzésére 1000, kézjelenként 125-125 darab teszt minta készült. Ezeknek a mintáknak elkészült az RGB és mélységi változata is. Az így készült minták felhasználásával történt meg a háló tesztelése.

Klasszifikációt megvalósító rendszer teljesítményét többféle módszerrel lehet értékelni. Az első ilyen módszer a teljes klasszifikáció pontosságának (accuracy) meghatározása, ami a helyes becslések számának és a teljes elemszám hányadosa:

$$\text{Pontosság} = \frac{\text{Sikeres tesztminta felismerések száma}}{\text{Tanítóminták száma}} \tag{2.1}$$

A tanítás végén a pontosság értéke mélységi képekkel betanított hálózat esetén 0,959, míg színes képekkel betanított hálózat esetén 0,913. A mért veszteség ugyanekkor mélységi háló esetén 0,081, míg a másik hálózat esetén 0,12.

A második módszer során a rendszer pontosságának mérése történik meg osztályonként, tehát esetünkben kézjelenként. Az így készült táblázatot konfúziós mátrixnak hívják.

Teljes klasszifikáció pontosságának meghatározása esetén szükség van a teszt minták kiértékelése során kapott döntések eredményeire, hogy a sikeres-e adott kézjel felismerése, vagy sem. A színes kamera képekkel betanított neurális háló pontossága a teszt minták felhasználásával 89,4%-s eredményt ért el, míg a mélységi szenzor adataival betanított háló esetén a rendszer pontossága 93,1%-os. A két hálózat pontossága közötti különbség jól látható, és ez alapján elmondható, hogy a mélységi szenzor képeivel betanított neurális hálózat hatékonyabb, mint a színes képekkel betanított háló.

A konfúziós mátrix használatának több előnye is van. Egyrészt ezzel egy részletesebb képet lehet kapni a neurális hálózat előrejelzéseinek pontosságáról, illetve lehetőség van megvizsgálni, hogy az egyes sikertelen felismerések esetén, a háló milyen osztályt vélt felismerni a várt helyett. Ez olyan esetben lehet hasznos, ahol az egyes osztályok reprezentáció között valamilyen hasonlóság van. Jelnyelv esetén vannak olyan kézjelek, ahol a mutatott jelek hasonítanak egymásra. A betanított hálózatnál hasonlóság fedezhető fel az A-B-E-N és a C-D-E-F betűkhöz köthető kézjeleknél (2.2. ábra). Az 2.1. táblázatból leolvasható a színes, míg a 2.2. táblázatból a mélységi szenzori képekkel betanított neurális háló osztályonkénti pontosság mérésének eredményei.

RGB		Becsült kézjel							
		A	B	C	D	E	F	N	V
Valódi kézjel	A	101	8	0	0	0	0	4	0
	B	4	99	3	0	0	0	0	0
	C	1	11	104	0	1	2	1	0
	D	0	0	16	116	0	7	0	0
	E	7	5	0	6	113	0	0	0
	F	0	0	2	0	0	116	0	0
	N	12	2	0	3	11	0	120	0
	V	0	0	0	0	0	0	0	125
	Pontosság (%)	80,8	79,2	83.2	92,8	90,4	0,928	96	1

2.1. táblázat. Osztályonkénti pontosság mérésének eredményei RGB módszernél (osztályonként 125 darab teszmintával)

DEPTH		Becsült kézjel							
		A	B	C	D	E	F	N	V
Valódi kézjel	A	108	4	0	0	0	0	0	0
	B	0	113	0	0	0	0	0	0
	C	0	1	106	0	4	1	1	0
	D	0	0	12	118	0	0	0	0
	E	7	6	2	7	114	0	1	0
	F	1	0	5	0	0	124	0	0
	N	9	1	0	0	7	0	123	0
	V	0	0	0	0	0	0	0	125
	Pontosság	0,864	0,904	0,848	0,944	0,912	0,992	0,984	1

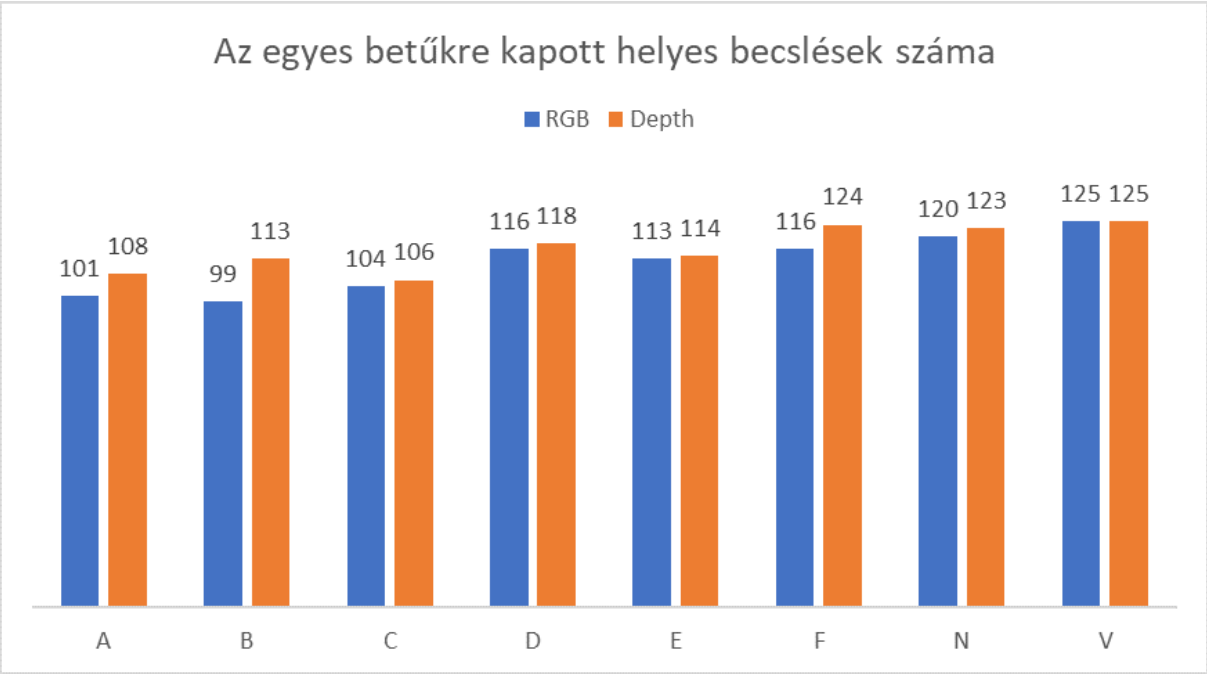
2.2. táblázat. Osztályonkénti pontosság mérésének eredményei mélységi módszernél (osztályonként 125 darab teszmintával)

A táblázat oszlopaiból kiolvasható, hogy a osztályonkénti 125 teszt mintából a neurális hálózat milyen betűt hányszor becsült. Az „Pontosság” nevű sorból lehet kiolvasni az egyes osztályhoz tartozó pontosság értékét. Az n-edik osztályhoz tartozó pontosság értéke a következő képlet alapján történik:

$$\text{Pontosság}_n = \frac{\text{Sikeres teszt minta felismerés száma az n-edik betűnél}}{\text{N. betű teszt mintáinak száma}} \tag{2.2}$$

Az értekből jól kiolvasható, hogy a mélységi képekkel betanított hálózat minden kézjel esetén nagyobb pontosságú, mint a másik háló.

Ahogy az a fejezet elején meg volt említve, érdemes megvizsgálni és összehasonlítani, hogy a két neurális háló az egyes osztályoknál mennyi helytelen becslést adott és olyankor mely osztályokat jelölt meg helyesként. Az egyik érdekes észrevétel, hogy a „V” betű felismerése során mindkét háló 100%-s pontossággal ismerte fel a kézjelet. Ennek az oka ott keresendő, hogy ez az egyetlen olyan kézjel a 8-ból, ahol nem fedezhető fel hasonlóság a többi jellel. A betű használatának oka az volt, hogy meg lehessen vizsgálni, hogy a két háló mennyire pontosan ismeri fel az ilyen típusú eseteket. Ez alapján elmondható,



2.8. ábra. Az egyes betűkre kapott helyes becslések száma

hogy minkét hálózat esetén pontos a felismerés. Érdemes megfigyelni a hasonló jelek esetén kapott eredmények közti összefüggést. Észrevehető, hogy a hibás becslések során a két háló nagyjából ugyanazokat a rossz betűket becsüli. Azonban a mélységi képekkel betanított háló esetén, a hibás becslések száma kevesebb.

Ez legjobban az „B" betűre kapott becslések során vehető észre. Mindkettő háló esetén előfordult, hogy a kapott teszt mintára a rendszer az „B" betű helyett „A", „C", „E" vagy „N" betűt ismert fel. Azonban a mélységi képekkel betanított háló esetén a téves becsülések száma 17,5%-kal kevesebb. Ez az eredmény megközelíthetőleg azonos minden betű esetében. Azonban minél magasabb a színes képekkel betanított háló pontossága, annál kevesebb az eltérés a hibás becslések számában a két háló között. Összeségében elmondható, hogy osztályonként történő pontosság vizsgálata során is a mélységi képekkel betanított hálózat pontosabb, mint a színes képeket tartalmazó neurális háló. Ezt az állítást támasztja alá a 2.8. ábrán látható diagram.

Lehetőség van megvizsgálni az egyes betűkhöz tartozó bizonyossági átlagot (confidence average) is. Az n-edik betűhöz tartozó confidence average érték kiszámolható a

$$\text{ConfAvg}_n = \frac{\sum_{j=0}^{m} \widehat{Y}_n}{m} \tag{2.3}$$

képlet segítségével, ahol az n a n-edik betűt, az m a tesztminták számát és az $\widehat{Y}_n$ pedig a becsült kimeneti vektor n-edik elemét jelenti. A 2.3. és 2.4. táblázat egyes oszlopai azt mutatják be, hogy az egy betűhöz tartozó 125-125 teszt mintából a rendszer átlagosan milyen magabiztos a becslésben. Észrevehető, hogy a mélységi képekkel betanított rendszer esetén a neurális háló a becslés során magabiztosabb, mint színes képeket használó hálózat. A két hálózat közti különbség átlagosan 4,4%. Ez elsősorban a későbbiekben fog fontos szerepet játszani, mikor a hálózat sokkal több kézjel felismerésére lesz képes. Össze-

RGB		Becsült kézjel							
		A	B	C	D	E	F	N	V
Valódi kézjel	A	0,609	0,053	0,006	0,001	0,023	0,0117	0,065	0
	B	0,0303	0,721	0,045	0,002	0,01	0,001	0,002	0
	C	0,029	0,104	0,7691	0,003	0,002	0,026	0,0004	0,0022
	D	0,0112	0,005	0,144	0,8892	0,0104	0,091	0,001	0,033
	E	0,0712	0,044	0,006	0,049	0,83	0,06	0,05	0
	F	0,11	0,009	0,023	0,007	0,042	0,861	0,0007	0,0048
	N	0,141	0,06	0,002	0,055	0,084	0,007	0,89	0
	V	0,005	0,003	0,006	0,0013	0,004	0,004	0,001	0,96

2.3. táblázat. Átlagos confidence értékek RGB módszernél

DEPTH		Becsült kézjel							
		A	B	C	D	E	F	N	V
Valódi kézjel	A	0,806	0,003	0,002	0,007	0,0019	0,006	0,03587	0,01
	B	0,001	0,74	0,001	0,0023	0,006	0,003	0,0024	0,011
	C	0,004	0,088	0,77	0,013	0,003	0,0368	0,006	0,001
	D	0,003	0,022	0,054	0,9101	0,04	0,012	0,001	0,003
	E	0,13	0,0197	0,009	0,041	0,87	0,05	0,009	0,004
	F	0,0049	0,09	0,161	0,035	0,004	0,9	0,0505	0,001
	N	0,058	0,024	0,002	0,004	0,08	0,001	0,91	0,004
	V	0,007	0,015	0,005	0,002	0,001	0	0	0,98

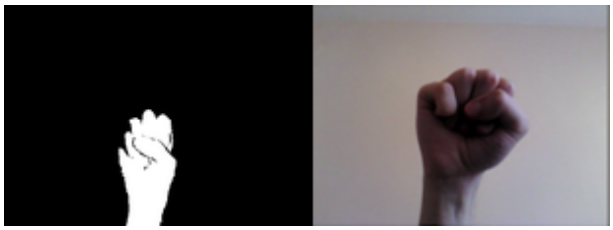
2.4. táblázat. Átlagos confidence értékek mélységi módszernél

hasonlítva a 2.1. és 2.3. táblázat eredményeit, általánosságban elmondható, hogy minél több a téves becslés egy betű esetén, annál kevesebb lesz a megbízhatósági átlag az adott betűnél. Érdemes megfigyelni, hogy a „V" betű esetén is észre lehet venni 2%-os változást a magabiztosságban a hálózatok között, miközben minkét esetben a 100%-os pontossággal találták el a helyes kézjelet.

Mivel jelnyelv esetén a térbeliség is számít, a két neurális hálózat e szempont alapján is meg kell vizsgálni. Ez az „A" és „N" betű képeinek tesztelésével kapott eredményekkel ellenőrizhető, ugyanis „A" betű esetén az ujjak teljesen a tenyéren helyezkednek el, míg „N" betű esetén az ujjak inkább a tenyér előtt vannak. A 2 betű közti térbeli különbséget a 2.9. ábra jól érzékelteti. A két háló közti különbség leginkább az „A" minták tesztelése során jelentős. Itt vehető észre leginkább, hogy a színes képek esetén a térbeli megjelenés összezavarhatja a neurális hálót. Ezt a mélységi szenzor képeivel betanított háló valamennyivel hatásosabban dolgozza fel. Vagyis elmondható, hogy a térbeliség vizsgálata alapján a mélységi képek jobb eredményt értek el, mint a színes képek.

Az „E" betű (2.10. ábra) összehasonlításával történt meg a kézjelek részletességének vizsgálata, ugyanis ennél a betűnél vehető észre leginkább a két különböző módszerrel előállított képek részletessége közti különbség. Mélységi képek esetén a kép kontúrja mellett a kéz belső részleteinek csupán néhány részlete fedezhető fel. Az „E" betűhöz tartozó minták tesztelése során kapott értékek vizsgálatával könnyen ellenőrizhető, hogy az egyes neurális hálók mennyire képesek felismerni a kézjelek részletességét. A táblázatoknál az

2.9. ábra. Az "N" és "A" betű térbeliségének vizsgálata



2.10. ábra. Az "E" betű részletességének vizsgálata

„E” oszlopának értékeinek leolvasásával észrevehető, hogy mind a két háló esetén a helyesen felismert kézjelek száma szinte megegyezik, viszont mélységi képek használata esetén elmondható, hogy a pontosság értéke egy kicsivel nagyobb. Ezért elmondható, hogy mindkettő módszer alkalmas a kézjelek részleteinek megkülönböztetésére.

2.6. Konklúzió

A két neurális hálózat tesztelése során azt a következtetést lehetett levonni, hogy a jelnyelvi ujjábécé felismerése mind a színes és mélységi képekkel betanított hálózat esetén hatékony és pontos. Ezt jól mutatja, hogy mindkettő hálózat 85% feletti pontossággal ismerte fel a kézjeleket. Azonban a tesztelés során megfigyelhető volt, hogy a mélységi képekkel betanított hálózat közel 4%-al jobb eredményt ért el. Mélységi képek használata esetén emiatt a rossz becslések száma alacsonyabb volt és esetleges helytelen becslés során is a helyes jelhez hasonló megjelenésű jelet ismert fel.

A két módszer bizonyossági értékeit tekintve szintén elmondható, hogy a mélységi képek esetén a neurális háló magasabb értéket ért el, mint a színes képekkel betanított hálózat. Ebben az esetben a két hálózat közti különbség átlagosan 4,4%. Ez elsősorban majd később fog fontos szerepet játszani, mikor a hálózat sokkal több kézjel felismerésére lesz képes.

Emellett kijelenthető, hogy a jelnyelvi felismerés során bemutatott szempontok alapján a mélységi képekkel betanított hálózat jobban teljesített, mint az RGB képekkel betanított rendszer. A térbeliség vizsgálata az „A” és az „N” betű összehasonlításával történt meg. A hibás becslések száma színes képek 7%-kal volt több, mint mélységi esetén.

Az egyes kézjelek részletességének vizsgálatanál a mélységi képek hiába voltak kevésbé részletgazdagok, mint a színes képek, a neurális hálók mindkét esetben képes volt megfelelően felismerni a mutatott kézjelet, ami a „E” betű vizsgálatával történt meg. Említést

lehet tenni arra is, hogy a mélységi képekkel betanított rendszer ennél a vizsgált betűnél is pontosabb eredményt el, mint a színes képeknél, azonban a különbség jelentéktelen.

Meg kell említeni azt is, hogy a hálózat betanításakor a bemeneti paraméterek száma színes képek esetén 160x120x3, míg a mélységi képek csupán 160x120x1 darab bemeneti paraméterrel rendelkeznek. Ez jelentősen befolyásolja a hálózat tanításának hosszát.

3. fejezet

Statikus kézjelek felismerése konvolúciós neurális hálózat alkalmazásával

A kutatás további részében mélységi szenzor adatai alapján készített adathalmazzal történik a neurális hálózatok tanítása, a fenti okokból.

A magyar jelnyelvi ujjábécé 42 kézjelből áll, melyből 27 mozgást nem tartalmazó (statikus) kézjel és 15 kézmozgással rendelkező kézjel. Az első feladat a statikus kézjelek felismerése. Meg kell lehet említeni az "I" és "J" jel ebben az részben statikus formában lett értelmezve, ami valójában dinamikus mozgást tartalmazó kézjel.

3.1. Tanítóminták számossága

A tanítás és a tesztelés során alkalmazott minták mélységi szenzorból származó adatokból épül fel. A rögzített minták 160x120 pixel felbontású, 8 bites szürkeárnyalatos képek. Ezáltal a hálózat tanításakor a bemeneti paraméterek száma 160x120x1, ami jelentős mértékben lecsökkenti a tanítás hosszát.

A robusztus és effektív jelfelismerés érdekében szükség van arra, hogy a tanítóminták minél változatosabbak legyenek. Minden ember kezének felépítése és mérete eltérő, emellett előfordulhat az is, hogy egy adott jelet 2 ember, különböző módon mutatja, miközben ugyanazt a kézjelet mutatják. Sok esetben a mutatott kézjel dőlésszöge is megváltozik jelelés során. Fontos, hogy ilyen esetekben is képes legyen a rendszer a megfelelő jelfelismerésre. Az 3.1. ábra jól érzékelteti, hogy egy adott kézjelet ("C" betű) mennyi féle képpen lehet jelelni, anélkül, hogy ezzel megváltozna a kézjel jelentése.

Ahhoz, hogy az előbb említett problémák ne forduljanak elő, a tanítás és tesztelés során használt mintákat 20 alany által rögzített minták adják. Ezáltal biztosítva van a minták sokszínűsége, változatos felépítése, hiszen minden embernek sajátos jelelési módja van. 150 minta lett rögzítve egy betűről minden alany esetén. A rögzítés során fontos cél volt a kéz pozíciójának és dőlésszögének folytonos változtatása. Így a 26 statikus kézjel felismerésére szolgáló tanítóminták száma összesen 78000 minta lett.

Az overfitting elkerülése érdekében a 1.2 fejezetben bemutatott módszer alapján 3 részre lett bontva: tanító, validációs és teszt halmazra 8:1:1-s arányban [14]. A tanító

3.1. ábra. A "C" betű jelelésének különböző előfordulása

és validációs halmaz 18 alany által biztosított kézjelekből épül fel, míg a tesztelés során alkalmazott mintákat további 2 alany biztosította. Ez a 2 alany kézjelei a tanító/validációs halmaz számára ismeretlenek, teljesen elkülönülnek tőlük. A tanító és validációs minták esetében az egy kézjelről készült 150 minta egy alany esetén fel lett bontva 2 részre: 133 darab tanító és 17 darab validációs mintára. A teszt minták esetén az egy betű esetén mind a 150 minta fel lett használva a tesztelés során.

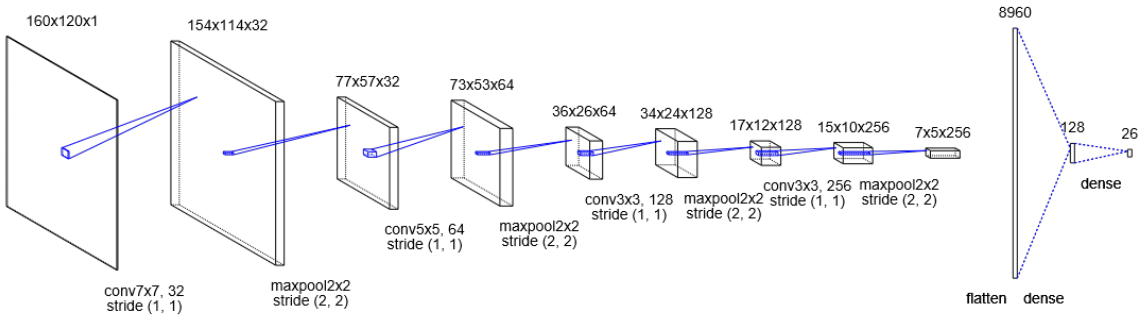
Így összességében a 62244 minta lett felhasználva tanításra, 7956 darab a validációra, míg további 7800 minta a tesztelésre.

3.2. Megvizsgált architektúrák

Ahhoz, hogy a lehető legmagasabb pontosságot érje el a rendszer, több különböző konvolúciós neurális hálózat architektúra lett megvizsgálva. Minden modell esetén többosztályú klasszifikációs módszer volt alkalmazva a jelnyelv kézjeleinek felismerésére. Emellett minden architektúra során ugyanaz a tanító-validációs-teszt halmaz volt használva. Minden architektúra esetén a neurális hálózat egy 8 bites 160x120 felbontású képet kap a bemenetére, míg a kimenete a hálózat becslése lesz. Az egyes architektúrák közti különbség a rétegek számában, típusában volt felfedezhető. Ugyanakkor a hálózat teljesítményének optimalizálására használt regularizációs módszereknél is fordulhatnak elő változtatások az egyes modellek között.

A saját architektúra felépítése előtt több olyan modell is megvizsgálásra került, mint például a LeNet [29] vagy AlexNet [30]. Ezek az architektúrák korábban hatékonynak bizonyultak hasonló klasszifikációs problémák megoldására. Emellett mások által készített, kevésbé ismert modellek is elemzésre kerültek, melyeket kifejezetten a jelnyelv kézjeleinek felismerésére alkották meg. Minden esetben az alapértelmezett, de-facto értékek és paraméterek voltak alkalmazva, ahogy azokat definiálták a publikációikban.

A LeNet [29] architektúra, melyet írott számok felismerésére alkalmaztak, 2 egymást követő konvolúciós és pooling réteget tartalmaz, melyet 2 teljesen összekötött réteg zár. A Bheda [31] által átdolgozott AlexNet több réteget és magasabb konvolúciós kernel méretet alkalmaz mint a LeNet. Az architektúra 5 konvolúciós, 3 pooling és 3 teljesen összekötött rétegekből épül fel. A harmadik vizsgált architektúra struktúráját tekintve hasonlít az AlexNet-ére, azonban itt 2 egymást követő konvolúciós réteg után következik a pooling réteg, mindezt 3-szor megismételve. Az architektúra két teljesen összekötött réteggel zárul.



3.2. ábra. Az ajánlott konvolúciós neurális hálózat strukturális felépítése. A bement egy 160x120 méretű szürkeárnyalatos kép, míg a kimenet egy 26 dimenziós vektor az adott osztályhoz becsült értékkel.

Architektúra név	Tanító halmaz Pontosság	Validációs halmaz Pontosság	Teszt halmaz Pontosság	Teszt halmaz confidence Average	Paraméterek száma
LeNet [29]	93.12	81.23	55.23	51.78	1068742
AlexNet [30]	99.83	97.80	83.92	83.37	16284786
Bheda et al. [31]	95.20	91.34	79.84	77.55	10906106
Ajánlott módszer	98.56	98.90	93.97	93.45	1572634

3.1. táblázat. Az eredmények kiértékelése az egyes architektúráknál és az ajánlott módszer esetén.

Az egyes architektúrák paramétereinek száma leolvasható a 3.1 táblázatból.

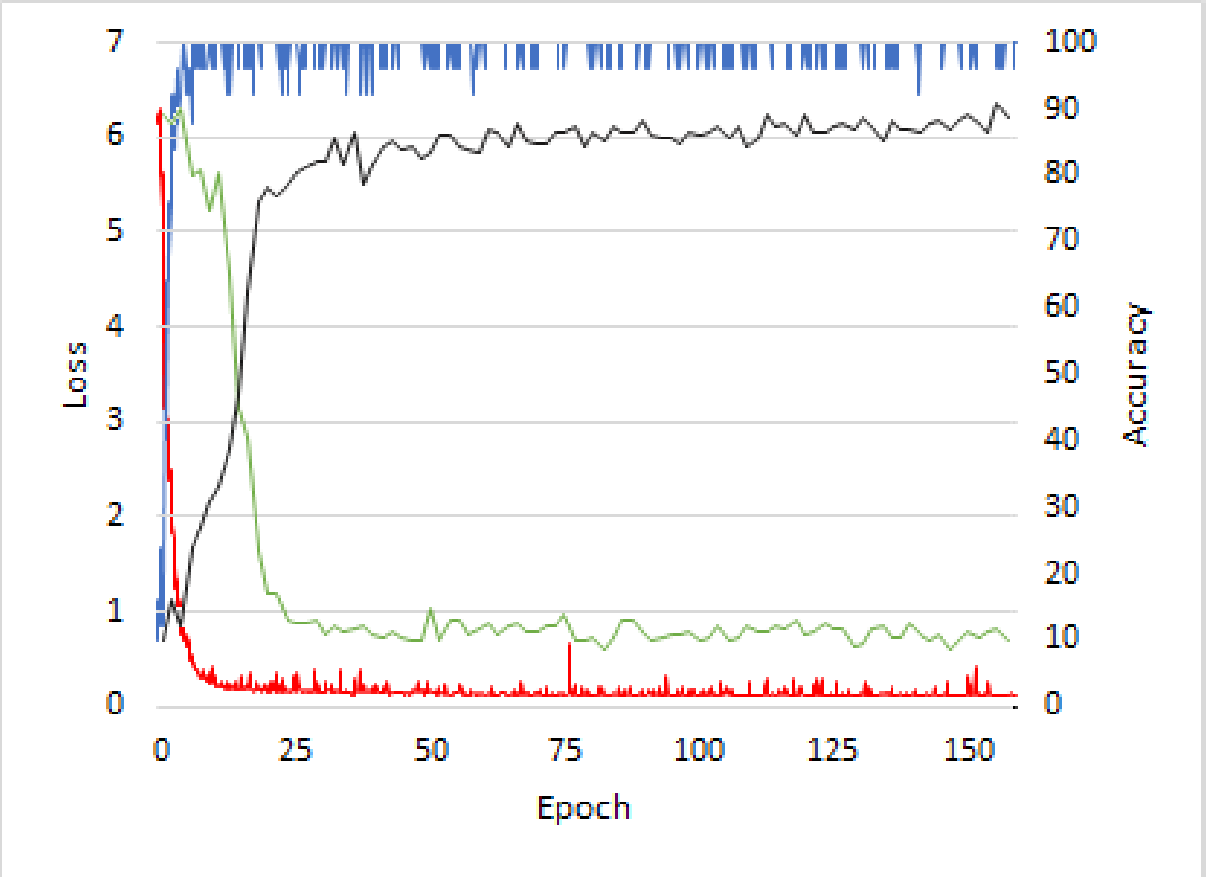
A kifejezetten erre a problémára készült saját neurális hálózat architektúra a következő rétegekből épül fel: A bemeneti kép 4 konvolúciós-pooling rétegpáron megy végig Az egyes konvolúciós rétegek 7x7, 5x5, 3x3 és 3x3 aktivációs térképet használnak, míg a filterek száma 32, 64, 128 és 256. A pooling rétegek 2x2 szűrőkből épül fel 2x2 lépésköz (stride) értékkel, aminek köszönhetően a reprezentáció mértékének csökkentése gyorsabb lesz. A hálózat 2 teljesen összekötött réteggel zárul 128 és 26 neuronnal. Az egyes rétegek aktivációs függvényénél ReLU volt alkalmazva, az utolsó réteg kivételénél, ahol az aktivációs függvény softmax. A hálózat felépítése vizuálisan a 3.2. ábrán látható.

Az overfitting elkerülése érdekeben droupout volt alkalmazva az első konvolúciós réteg és a teljelesen összekötött rétegek után. Hinton [32] 0,5-s dropout értéket ajánl a teljesen összekötött rétegek után, azonban újabb kutatások eredményei [33] a Hinton által preferált 0,5 dropout értéknél alacsonyabb (0,2) érték használatát ajánlja a konvolúcicós rétegek után. Emellett batch normailizáció és L2 típusú regularizáció volt alkalmazva a rétegeket után.

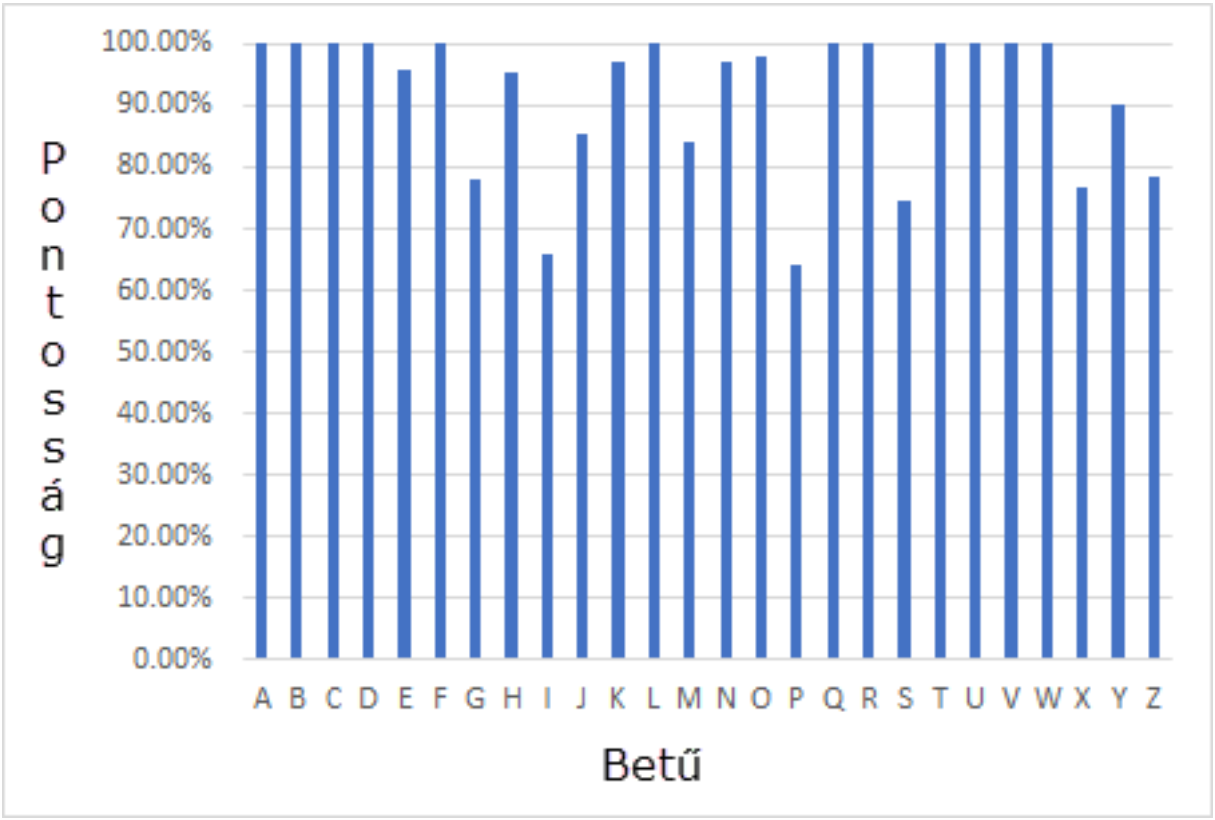
A neurális hálózat tanítása 150 epochon keresztül történt. A tanítás Adam optimalizáció volt használva eredeti értékek mellett. A veszteség mértéke categorical cross entropy függvény segítségével volt meghatározva. A 3.3 ábrán látható a pontosság és a veszteség értékeinek változását a tanítás során. A pontosság a teljes tanító és a validáció mintákra történő predikció eredményét reprezentálja.

3.3. Eredmények

A tanítás és a tesztelés eredményeit a 3.1. táblázatból lehet kiolvasni. Ezek alapján elmondható, hogy bár a bemutatott hálózatok hasonló klasszifikációs probléma megoldására alkalmasak, jelnyelv kézjeleinek a felismerése során az említett hálózatok nem mutatnak



3.3. ábra. A veszteség (vörös görbe (tanító), zöld görbe (validáció), az értékek a bal oldalon) és a pontosság (kék görbe (tanító), fekete görbe (validáció, az értékek a jobb oldalon) értékeinek változása a tanítás során a saját módszer esetén



3.4. ábra. Az egyes betűkre kapott helyes becslések százalékos eloszlása

kiemelkedő eredményeket. A tanítás és validálás során használt minták kiértékelése során mindhárom hálózat magas pontosságot ért el. Ugyanakkor meg kell említeni, hogy a LeNet architektúra esetén a tanítás és validáció során kapott eredmények közti különbség több mint 10%, míg a többi esetében a különbség néhány százalék. Azonban a hálózat számára ismeretlen tesztelési minták során az első 3 modell esetén alacsonyabb pontosság vehető észre, vagyis a neurális hálózatok nem képesek rendesen általánosítani, overfitting fedezhető fel.

Az ajánlott, saját architektúrával rendelkező, modell esetén a tesztelés során pontosásság értéke 93,97% lett, vagyis a vizsgált modellek közül ez érte el a legmagasabb pontosságot. A confidence average esetén jelentős különbségek vehető észre a vizsgált és saját modellek esetén. Az egyes architektúrák paramétereinek a számát megvizsgálva szintén elmondható, hogy a saját megvalósítás során a paraméter szám alacsonyabb mint a többi magasabb pontosságot elérő architektúráknál, ezzel lecsökkentve a tanítás és a becslések idejét.

Az eredmények tanulmányozásával lehetőség van a neurális hálózat működésének részletesebb vizsgálatára is. A 3.4. ábrán látható az egyes betűkre kapott helyes becslések száma a tesztminták használata esetén. Ez alapján elmondható, hogy az "I" és a "P" betű kivételével az összes osztály legalább 75% feletti pontossággal rendelkezik. 17 betű esetén a modell 95% feletti pontosságot ér el, sokszor hibátlan becslésekkel. Azoknál az betűknél, ahol a pontosság mértéke alacsonyabb, a jelenség megmagyarázható azzal, hogy előfordul olyan kézjel ami az említett betűhöz tartozó kézjelhez nagyon hasonlít. Az "I" betű esetén

ilyen például a "J", míg a "P" esetében a "Q" betűhöz tartozó kézjel. Konfúziós mátrix használatával egyszerűen megvizsgálható, hogy ez okozza e a több tévesztést a becslések során.

A 3.3. táblazatban látható konfúziós mátrixból kiolvasható a tesztelés során kapott összes becslést táblázatos formában, a helyes és téves becslésekkel együtt. Téves becslés során az is tanulmányozható, hogy a modell melyik betűt becsülte meg helytelenül. Az alacsonyabb pontosságot elért betűk esetén érdemes megvizsgálni, hogy milyen betűket becsült meg tévesen. A 3.3. táblázat és a 1.1. ábra segítségével megállapítható, hogy a tévesztések során sokszor olyan betűt becsült a modell, ami a valóságban hasonlítanak egymásra, csupán kisebb különbségek fedezhetőek fel a kézjelek között. Ilyen különbség lehet az "M" és "N" betű esetén például, hogy "M" betűhöz tartozó kézjel között egy "extra ujj" a különbség. De meg lehet említeni az "I" és "J" (ami valójában dinamikus mozgást tartalmazó kézjel, de itt statikus formában lett értelmezve a jel) betű esetén a különbség egy kicsivel ferdébb ujj, ezért fordulhat elő, hogy bizonyos esetekben a modell téved.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	100.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
B	-	100.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
C	-	-	100.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
D	-	-	-	100.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
E	-	1.3	-	-	96.0	-	-	-	-	-	-	-	-	-	-	-	-	-	2.7	-	-	-	-	-	-	-
F	-	-	-	-	-	100.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
G	-	-	-	-	-	-	78.0	17.3	-	-	-	-	-	1.3	-	-	-	-	-	3.3	-	-	-	-	-	-
H	-	-	-	-	-	-	4.7	95.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
I	-	-	-	-	-	-	-	-	66.0	34.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
J	-	-	-	-	-	-	-	-	12.0	85.3	-	-	-	1.3	-	-	-	-	-	-	-	-	-	-	-	1.3
K	-	-	-	-	-	-	-	-	-	-	97.3	-	-	-	-	-	-	1.3	-	-	-	-	-	-	-	1.3
L	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-
M	-	-	-	-	-	-	-	-	-	-	-	-	84.0	16.0	-	-	-	-	-	-	-	-	-	-	-	-
N	-	-	-	-	-	-	-	-	-	-	-	-	0.7	97.3	-	-	2.0	-	-	-	-	-	-	-	-	-
O	-	-	1.3	-	-	-	-	-	-	-	-	-	0.7	-	98.0	-	-	-	-	-	-	-	-	-	-	-
P	-	-	-	-	-	-	-	-	-	-	-	-	-	3.3	-	64.0	32.7	-	-	-	-	-	-	-	-	-
Q	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-	-	-	-	-	-
R	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-	-	-	-	-
S	-	-	-	-	12.7	-	-	12.7	-	-	-	-	-	-	-	-	-	-	74.7	-	-	-	-	-	-	-
T	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-	-	-
U	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.0	100.0	-	-	-	-	-
V	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-
W	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-
X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2.7	-	0.7	-	-	-	-	-	76.7	-	20.0
Y	-	-	-	-	-	-	-	-	-	10.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	90.0	-
Z	-	-	0.7	-	-	-	17.3	-	-	-	-	-	-	-	-	-	-	-	3.3	-	-	-	-	-	-	78.7

3.2. táblázat. Osztályonkénti pontosság mérésének eredményei konfúziós mátrixban százelék formában.

3.4. Konklúzió

A statikus kézjelek pontos felismerése érdekében megvizsgáltam több architektúrát és összehasonlítottam őket a saját architektúrámmal. Az eredmények alapján azt a következtetést vontam le, hogy az általam létrehozott architektúra esetén jobb eredményeket értem el. Ezt jól mutatja, hogy a saját modellem a tesztmintákat 93,7% pontossággal becsülte meg helyesen, míg a többi architektúra esetén ez az eredmény alacsonyabb volt.

Érdemes megemlíteni, hogy saját modell a LeNet kivételével, az összes bemutatott architektúra paramétereinek száma magasabb mint az ajánlott architektúra alkalmazása esetén. Ezáltal a modell mérete is sokkal nagyobb lesz amely közvetve a tanítás hosszát is befolyásolja.

4. fejezet

Dinamikus kézjelfelismerés rekurrens neurális hálózattal

A magyar ujjábécé nem csak statikus, mozgást nem tartalmazó kézjelekből épül fel. Bizonyos jelek dinamikusak, vagyis mozgást is tartalmaznak. A magyar ujjábécében 15 dinamikus kézjel található. A statikus kézjelek felismerésére létrehozott neurális hálózat nem képes rendesen felismerni ezeket a kézjeleket.

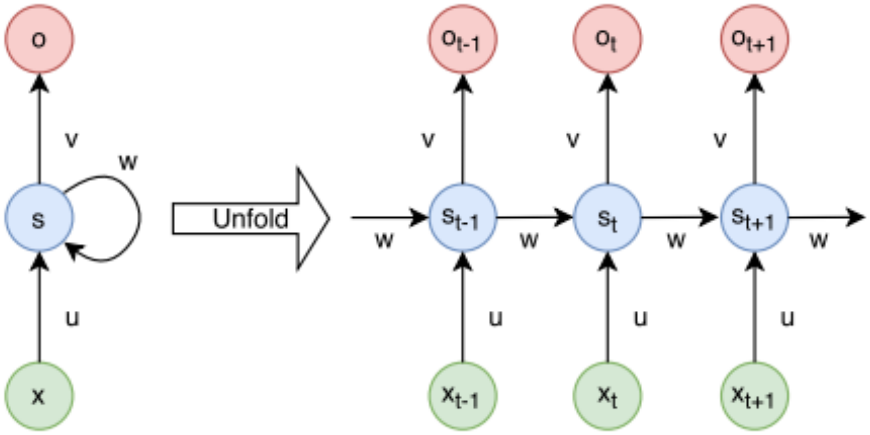
4.1. Rekurrens neurális hálózat

A rekurrens neurális hálózatok (RNN) [34] szekvenciális adatok feldolgozására alkalmasak. Ilyen szekvenciális adat lehet például a beszéd, kézírás vagy videó. Az egymástól függő bemeneti adatokat a hagyományos előrecsatolt neurális hálózat nem tudja feldolgozni. A rekurrens hálózatok alkalmasak ilyen problémák megoldására a struktúrájuk miatt.

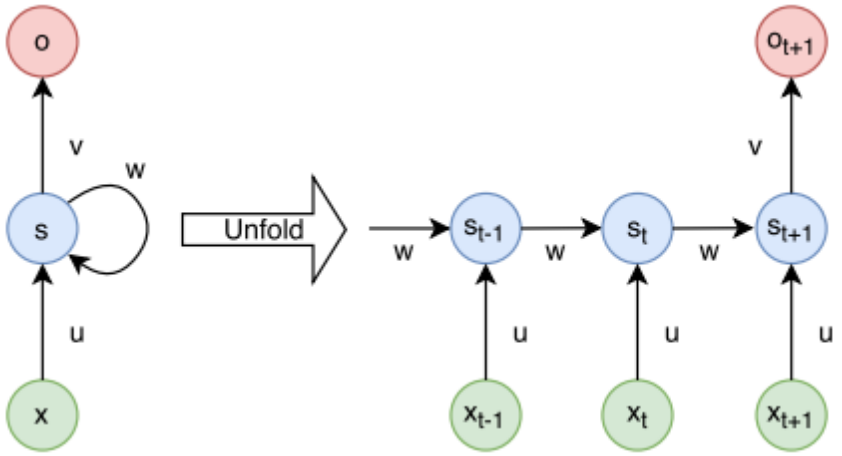
A rekurrens hálózat végrehajtja ugyanazt a feladatot a szekvencia összes elemén és a kimenet az előző számítások eredményétől függ. RNN esetén elmondható, hogy rendelkeznek "memóriával", ami eltárolja az addig kiszámolt értékeket. Elméletben az RNN hosszú szekvenciák esetén is képes felhasználni az eltárolt információt, de a gyakorlatban csak néhány lépéshosszan tud visszatekintheti.

A 4.1. ábrán egy zárt és egy kibontott rekurrens neurális hálózat látható. Az ábra bal oldalán a hálózat struktúrája van. Az x csomópont a bemenetet, az s csomópont a rejtett állapotot, míg az o a hálózat kimenetét jelenti. Az s a rejtett állapotra vonatkozó információ, amit a hálózat memóriájának is hívják, visszacsatolásra kerül a rejtett állapotba. Hasonlóan az előrecsatolt neurális hálózatnál, a u, v és w súlyok, illetve a b bias a tanítási folyamat során beállított paraméterek. A rekurrens hálózatok egyik előnye, hogy a súlyértékek ugyanazok maradnak a a teljes szekvenciális bemenet számára. Ez jelentősen lecsökkenti a paraméterek számát az előrecsatolt hálózatokkal ellentétben, ahol a súlyok minden bemenettel megváltoznak. Az ábra jobb oldala a hálózat kiértékelésének három időbeli lépését mutatja. A t időbeli lépésben, az s_t rejtett állapot kiszámítható az x_t bemeneti és az előző s_{t-1} rejtett állapot felhasználásával

$$s_t = f(u \times x_t + w \times s_{t-1} + b) \tag{4.1}$$



4.1. ábra. Rekurrens neurális hálózat három időbeli lépésre bontva szekvenciális kimenettel.



4.2. ábra. Rekurrens neurális hálózat három időbeli lépésre bontva nem szekvenciális kimenettel.

képlettel, ahol a f az aktivációs függvény. A s_t rejtett állapot kimenete továbbításra kerül az s_{t+1} rejtett állapot bemenetére. Ez lehetővé teszi a hálózat számára az előző információ feldolgozását.

Rekurrens hálózatok egy megvalósítási formája lehet, hogy mind a bemenet és kimenet szekvenciális. Ilyen felépítés látható a 4.1. ábrán is, mely használható szöveg generálásra [35], gépi fordításra [36]. Azonban vannak olyan esetek ahol nem szekvenciális kimenetre van szükség. Ilyen eset lehet a videó alapú klasszifikáció. Az 4.2 egy ilyen megvalósítást ábrázol. Az s rejtett állapot az o kimenet csupán az utolsó x bemenet esetén állítja elő, ahogy az az ábra bal oldalán is látható. Ez a folyamat van ábrázolva az ábra jobb oldalán is, ahol az s_{t-1} és s_t rejtett állapot nem állít elő semmilyen kimenetet. Csupán az s_{t+1} rejtett állapot adja meg az o_{t+1} kimenetet.

A rekurrens neurális hálózatok tanítása hasonlít a hagyományos neurális hálózatokéhoz. Két ismert algoritmust dolgoztak ki a hatékony súlyértékek kiszámítására: valós idejű rekurzív tanítás (real time recurrent learning, RTRL) [37] és az időbeli hibavisszaterjesztés (backpropagation through time, BPTT) [38] [39]. Azonban a BPTT megvalósítást és számítási időt tekintve hatékonyabb. Itt is backpropagation algoritmust használnak, egy

kis változtatással.

Mivel a paramétereket a hálózat összes időbeli lépései osztják meg, az egyes kimeneteknél a gradiens nemcsak az aktuális időbeli lépés számításai alapján van kiszámítva, hanem az előző lépésesek is figyelembe vannak véve. Például, a $t = 6$ gradiens kiszámításához az az őt megelőző 5 lépés esetén szükség a backpropagation során kapott értékre, majd utána a gradiens értékekek összeadása szükséges.

A hibafüggvény megadható a

$$E = \sum_{t=0}^{l} E_t \tag{4.2}$$

formában, ahol az l a kimeneti szekvencia hosszát jelenti és az

$$E_t = \frac{1}{2} \left\| o_t - y_t \right\|^2, \tag{4.3}$$

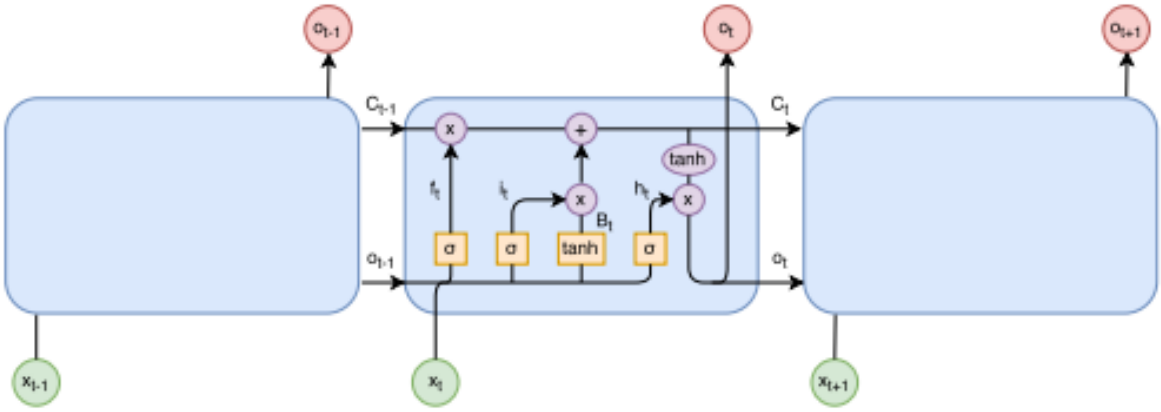
az adott időegységre vonatkozó hiba. Az $(y_1, y_2, ..., y_l)$ a valódi kimenet és az $(o_1, o_2, ..., o_l)$ pedig a hálózat által generált kimenet.

A BPTT módszernek azonban van egy limitációja, amit eltűnő grandiens (vanishing gradient) [40] [41] problémának hívnak. Észrevehető, hogy a sigmoid függvény derivált határa a nullához közelít mindkét végénél. Ilyen alacsony értékek szorzása láncszabályban a gradiens eltűnését okozhatja egy későbbi időbeli lépéskor. Ez azt jelenti, hogy a rekurrens neurális hálózatok képtelenek hosszú távú függőségeket megtanulni szekvenciális bemenetek esetében. A gradiens információ eltűnése vagy késleltetése exponenciálisan megnő ha a bemenet több mint 5-10 idő lemaradást tartalmaz. Erre egy megoldás lehet a ReLU aktivációs függvény használata, vagy a Long Short-Term Memory [42] alkalmazása.

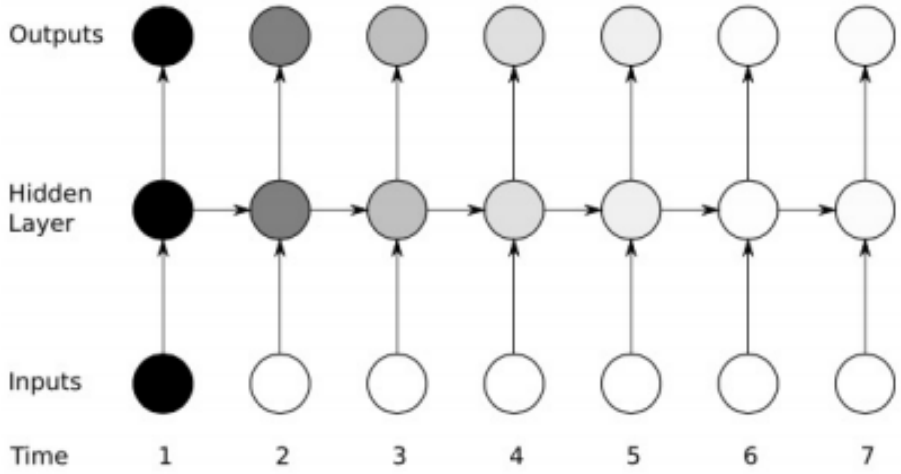
4.2. Long Short-Term Memory

Ahhoz, hogy a neurális hálózat képes legyen hosszú távú szekvenciák betanítására Hochreiter és Schmidhuber bemutatta a Long Short-Term Memory-t (LSTM) [42]. Az RNN-nel ellentétben, az LSTM-ben lévő memória blokknak van egy rejtett egysége (memory cell), amely egy visszatérő (rekurrens) kapcsolattal és két kapuzó egységgel (input és output gate) rendelkezik, amelyek az előző kontextusnak megfelelően szabályozzák az adatokhoz való hozzáférést a memória cellához. Később Gers [43] módosította ezt a kezdeti architektúrát egy felejtő kapu hozzáadásával. Ez megtanulja a memória visszaállításának (forget gate) viselkedését. Az LSTM hálózatokat sikeresen alkalmazták kézírás felismerésre [44] vagy beszéd felismerésre [45], ahol a hálózat bemenetére hosszú szekvenciális adatok kerülnek.

Az LSTM egy rejtett egysége, mely a 4.3. ábrán látható, 4 részből épül fel: egy cella állapot (cell state) c_t, egy felejtő kapu (forget gate) f_t, egy bemeneti input kaput i_t és egy kimeneti output kaput h_t. A cella állapot egy memória egységként működik, mely tartalmazza a szekvenciális bemeneti vektorban rendelkezésre álló összes fontos információt. Ezt az információt a három kapu választja ki és szűri. Mindhárom kapu rendelkezik súly értékekkel és bias-sal. Az u_f, w_f súlyok és a bf bias a forget kapuhoz tartozik. Az u_i, w_f u_b és w_b súlyokat és a b_i, b_b bias értékeket az input kapu használja. Az output kapu az u_h és w_h súly és b_h biast tartalmazza. A rekurrens hálózattal összehasonlítva,



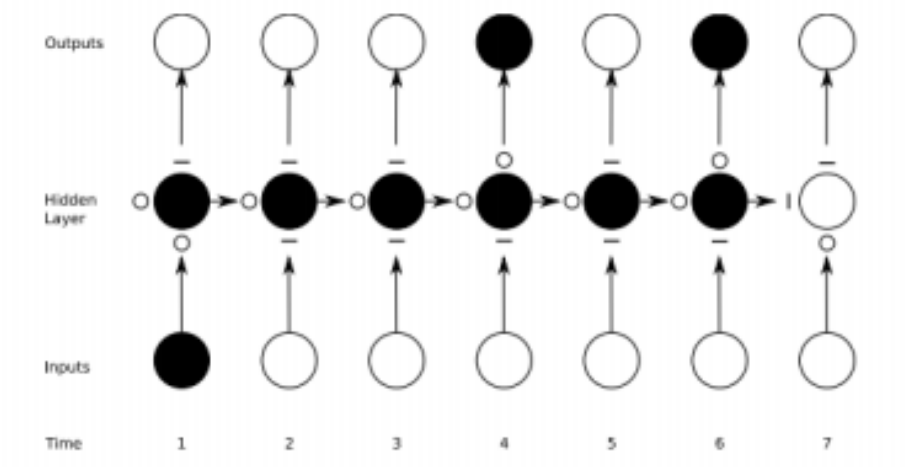
4.3. ábra. Az LSTM rejtett egysége.



4.4. ábra. A gradiens információ eltűnése RNN esetén. A hálózat csomópontjainak árnyékolása az érzékenységet mutatja a bemenetekre az első idő alapján. Minél erősebb az árnyalat az érzékenység annál nagyobb. Az érzékenység idővel csökken, mivel az új bemenetek felülírják a rejtett réteg aktivációit, és a hálózat "elfelejti az első bemenetet [46].

a nagyobb mennyiségű súly és bias értékek megnövelik az LSTM hálózatok tanulási folyamatának számítási komplexitását. Viszont ez teszi lehetővé az LSTM számára, hogy megtanulják az adatok hosszú távú összefüggéseit. Az input kapu lehetővé teszi, hogy a bejövő jel megváltoztassa a memória cella állapotát vagy blokkolja azt. Továbbá az output kapu lehetővé teszi, hogy a memória cella állapota megváltoztasson más neuronokat, vagy megakadályozza azt. A forget kapu hagyja, hogy a cella emlékezzen vagy elfelejtse korábbi állapotát.

Az LSTM megóvja a gradiens információt, ahogy az a 4.5. ábrán is észrevehető. A 4.4. ábrán a csomópontok árnyékolása az érzékenységet jelzi a bemenetekre az első időben. LSTM esetében a fekete csomópontok a maximális érzékenységet, míg a fehér csomópontok a teljesen érzéktelenek. Az input, a forget és az output kapuk alul, a bal és a rejtett réteg fölött vannak ábrázolva. Minden kapu vagy teljesen nyitva ("O") vagy zárva ("-") vannak. A memória cella addig "emlékszik" az első bemenetre, amíg a forget kapu kapu es az input



4.5. ábra. A gradiens információ megőrzése LSTM esetén [46].

kapu zárva van. Az kimeneti réteg érzékenysége be és kikapcsolható az output kapuval, anélkül hogy befolyásolná a cellát [46].

Az LSTM 3 kapuja védi és irányítja a cella állapot. Ahogy egy új input érkezik az input kapu aktiválódik, az új információ felhalmozódik a cellába. Emellett, ha a forget kapu be volt kapcsolva, a múltbéli c_{t-1} cella státusz "elfelejthető" lehet. A forget kapu a következő módon definiálható

$$f_t = \sigma(u_f \times x_t + w_f \times o_{t-1} + b_f), \tag{4.4}$$

ami a bemeneti x_t-t és az előző rejtett egység o_{t-1} kimenetét használja. A szigmoid függvény kimenete 0 és 1 között lehet, ahol a nulla érték azt jelenti, hogy a cella állapot teljesen törölve lesz, míg az egyes érték, hogy az információ a cella állapotban marad.

Később az input kapu tárolja az új információt a cella állapot számára:

$$i_t = \sigma(u_i \times x_t + w_i \times o_{t-1} + b_i) \tag{4.5}$$

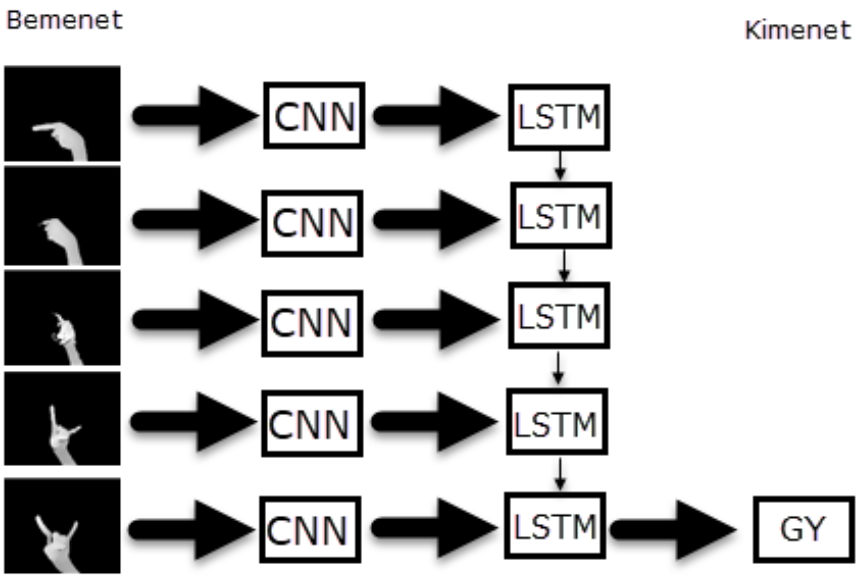
$$B_t = \tanh(u_b \times x_t + w_b \times o_t + b_b). \tag{4.6}$$

A szigmoid függvény az i_t input kapuban megadja, hogy a cella állapot mely része fog frissülni, és a tangens hiperbolikusz aktiváció a B_t-ben specifikálja, hogy milyen értékek lesznek eltárolva a kiválasztott részekben. A cella állapot az alábbiak szerint frissül, az f_t kapu és az i_t bemeneti kapu és a B_t szorzása segítségével:

$$C_t = i_t \times B_t + f_t \times C_{t-1}. \tag{4.7}$$

A rejtett egység utolsó része az o_t output kapu, ami kiválasztja, hogy a cella állapot mely részeit fogja a következő rejtett egységbe továbbítani:

$$h_t = \sigma(u_h \times x_t + w_h \times o_{t-1} + b_h). \tag{4.8}$$



4.6. ábra. CNN és LSTM architektúra alkalmazása GY kézjel felismerésére.

A kiválasztás után a h_t kimeneti kaput megszorozzák a cella állapotával, hogy elérhetővé váljanak, hogy mely tényleges információkat kell továbbítani a következő rejtett egység felé:

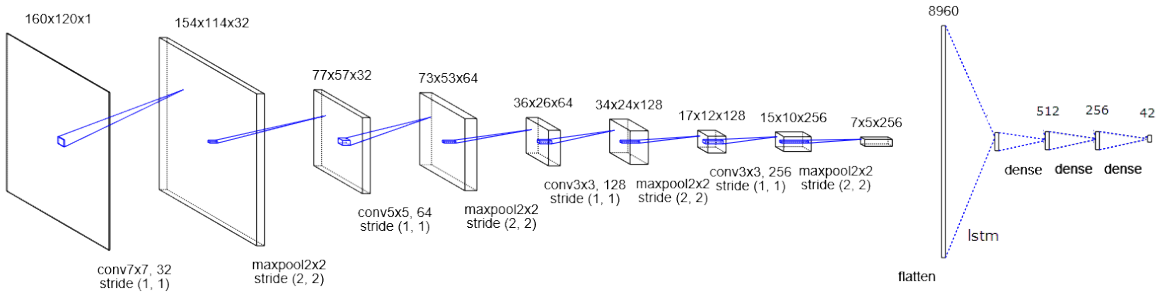
$$o_t = \sigma(h_t \times \tanh(C_t)). \tag{4.9}$$

4.3. Konvolúciós LSTM architektúra

A konvolúciós LSTM egy LSTM architektúra mely szekvenciális rácsszerű bemenettel rendelkezik, mint például a videófolyam. A CNN LSTM magába foglalja a konvolúciós neurális hálózat rétegeinek használatát, a bemeneti adatok jellemzőinek kiemelésére, kombinálva az LSTM rétegekkel, a szekvencia predikciójának támogatására. Ilyen megvalósítás fedezhető fel videó és kép leírás során, illetve videóban lévő tevékenység felismerésekor. A dinamikus kézjelek felismerése, ahol a bemenetet videók adják, leginkább az utóbbi példához hasonló problémaként írható le.

Egy CNN LSTM hálózat definiálható úgy, hogy a hálózat elején konvolúciós rétegek állnak, melyet LSTM és egyszerű teljesen összekötött rétegek követik. Érdemes két modellre bontani az architektúrát: egy CNN modellre, ahol a jellemzők kiemelése történik, illetve egy LSTM modellre, a jellemzők időbeli lépéseinek elemzésére. Egy ilyen felépítés látható a 4.6. ábrán.

Az ajánlott hálózat CNN része megegyezik a statikus kézjelek felismerése során használt ajánlott architektúrával, kisebb változtatásokkal. Az architektúra rétegein változás nem történt, tehát a hálózat CNN modellje 4 konvolúciós-pooling rétegpárból épül fel. A konvolúciós rétegek aktivációs térképet használnak míg a filterek száma 32, 64, 128 és 256 lett. Ezt a műveletet többször is meg kell ismételni a bemenet többi képére is, hogy az LSTM képes legyen a belső rejtett állapotot felépíteni, hogy frissíteni tudja a súlyokat



4.7. ábra. CNN és LSTM architektúra.

a BPTT algoritmus használatával. Ez megvalósítható úgy, hogy a CNN modell minden rétege egy úgynevezett Time Distributed rétegbe lesz csomagolva. Ez a réteg a kívánt eredményt ugyanazon réteg vagy rétegek alkalmazásával többször is elérheti. Ebben az esetben a réteg ezt a többszörös beadási idő lépésekre történő alkalmazásával, és ezzel egyidejűleg a LSTM-modell a kép jellemzőinek sorozatát biztosítja.

A konvolúciós rétegek után következik a hálózat LSTM modell része, ami egy LSTM rétegből és 3 egymást követő teljesen összekötött rétegből áll, ahol a neuronok száma sorban 512, 256, 42. Az első két teljesen összekötött réteg aktivációs függvénye ReLU, míg az utolsó réteg softmax függvény. A hatékony általánosítás és az overfitting elkerülése érdekében dropout és batch normalizálció volt alkalmazva mind a konvolúciós és LSTM rétegek után [47]. Az LSTM rétegek után Hinton [32] közleménye alapján 0,5-s dropout lett használva, míg konvolúciós rétegek után 0,2-s dropout volt alkalmazva [33]. Emelett a teljesen összekötött rétegek után L2 típusú regularizáció történt. A teljes architektúra a 4.7. ábrán látható.

4.4. Tanítóminták

A hálózat bemenete egy 10 frameből álló képsorozat, ahol a képek egy körülbelül 2 másodpercből álló felvételből lettek kivágva. A 4.8. ábrán a "gy" kézjelről készült tanítóminta látható. Az egyes képek felbontása továbbra is 160x120 felbontűsú, 8 bites szürkeárnyalatos formátumban. Mivel egyszerre a hálózat bemenetére 10 ilyen kép kerül, a paraméterek száma 10x160x120x1 lesz.

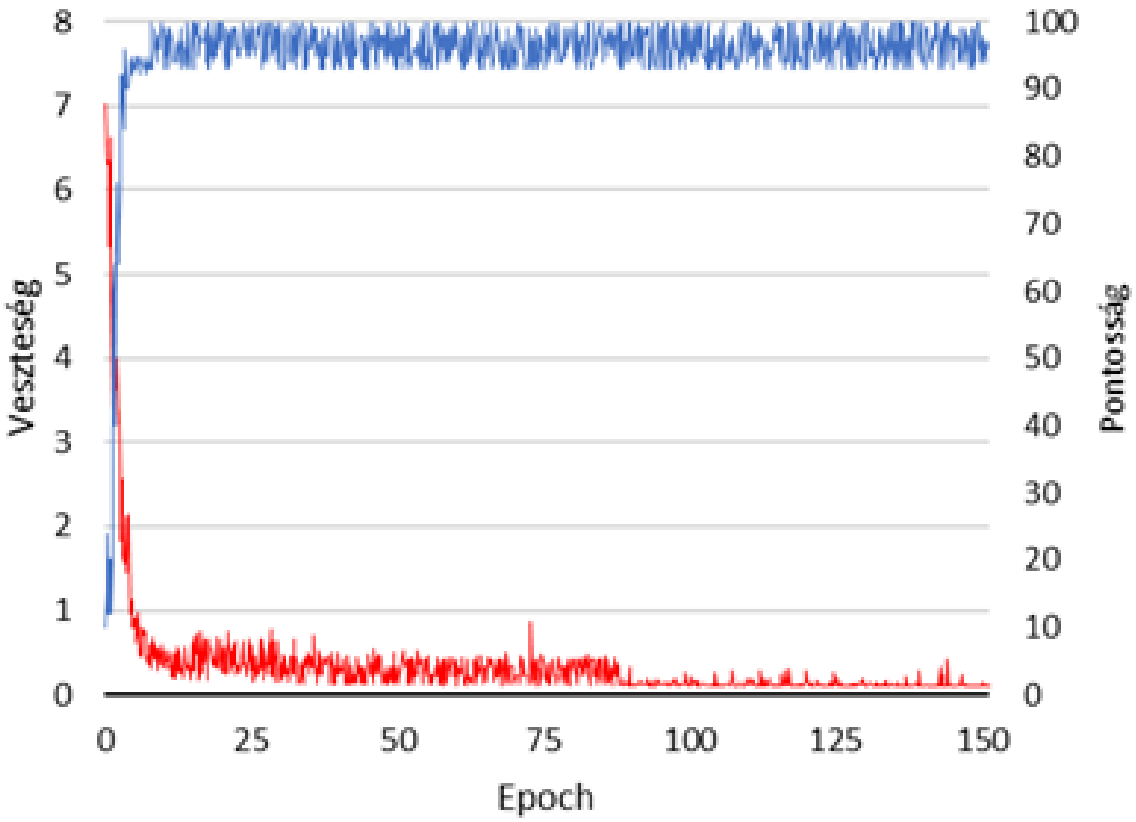
Ahhoz, hogy a hálózat hatékonyan tudja felismerni mind a statikus mozgást nem tartalmazó és a mozgást tartalmazó dinamikus kézjeleket, a tanító és teszthalmaz összesen 20 ember által rögzített kézjelekből épül. Minden minta 10 frameből áll. Egy ember egy kézjelről 15 mintát biztosított, változatos kézpozícióval. Ezáltal egy kézjelről összesen 300 készült. A tanító, validációs és teszthalmaz a statikus kézjelek felismerése során bemutatott módszer alapján lett felbontva.

A magyar ujjábécé kézjelei esetén az ékezetes és nem ékezetes magánhangzók közti különbség annyi, hogy ékezetes betű esetén a jelelő kezet jobbra kell tolni, miközben a nem ékezetes megfelelőjét jeleli az ember. A 4.9. ábrán jól látható az "O" és az "Ó" jel közti különbség. Emellett a jelnyelvben a kettős betűket (például "gy") úgy kell jelelni, hogy a két betű (jelen esetben a "G" és "Y") betűt kell egymás után gyorsan jelelni.

Fontos, hogy a hálózat képes legyen ezeket az előbb említett nehézségeket felismerni és

4.8. ábra. A "GY" betűhöz tartozó kézjel.

4.9. ábra. Az "O" és "Ó" kézjel közti különbség.



4.10. ábra. A veszteség (vörös görbe, az értékek a bal oldalon) és a pontosság (kék görbe, az értékek a jobb oldalon) értékeinek változása a tanítás során.

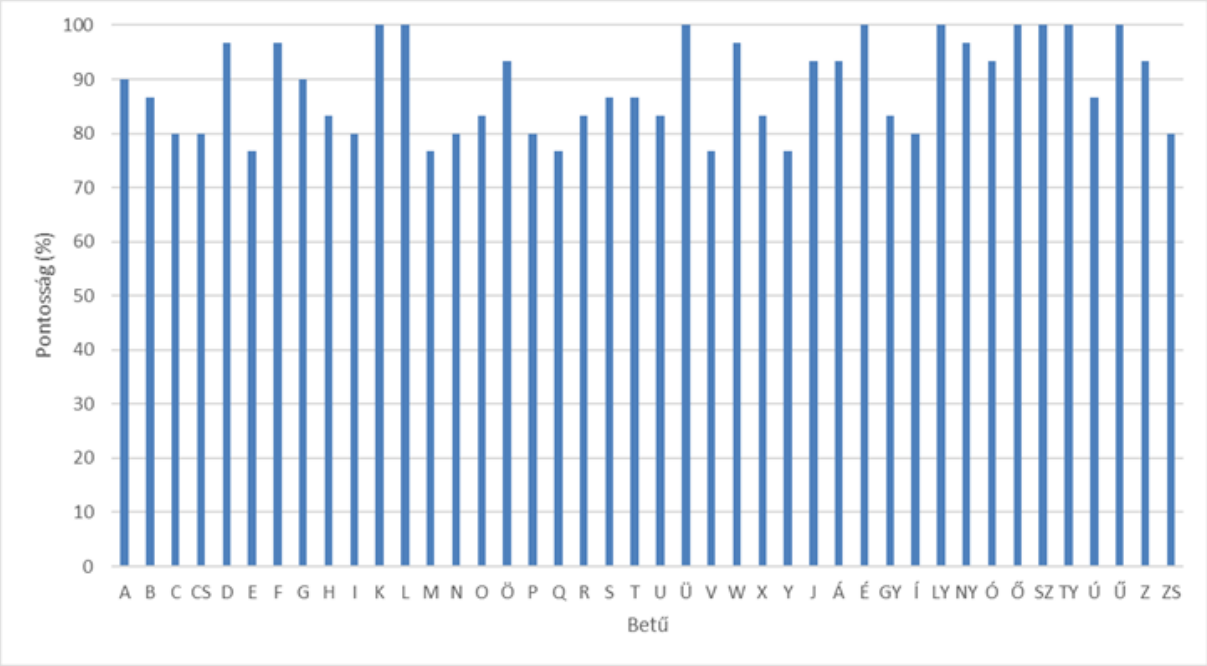
megkülönböztetni. A 4.3. ábrán észre lehet venni, hogy "Gy" betű esetén a jelelő először egy "G" betűt jelel, majd a jelelő kezét "Y" betűhöz tartozó kézjelre módosítja. Ezek az okok is indokolják a tanítóminták magas számát az egyes betűk esetén.

4.5. Eredmények

A neurális hálózat tanítása 150 epochon keresztül történt, Adam optimalizáció használatával, az eredeti értékek megtartása mellett. Cross entropy függvény segítségével lett meghatározva a veszteség mértéke. A tanítás és a veszteség értéke a 4.10. ábrán látható módon változott. A pontosság a teljes tanító és a validáció mintákra történő predikció eredményét reprezentálja.

A neurális hálózat pontossága 2, a hálózat számára ismeretlen személy által biztosított tesztmintákkal lett megvizsgálva. Egy személy esetében egy kézjelről 15 minta készült. A hálózat pontossága a teljes tesztmintákat figyelembe véve 88.6% lett. Ez az eredmény több mint a hasonló megvalósításon alapú jelnyelvfelismerést [48] [49] megvalósító munkák.

Az egyes betűkre kapott helyes becslések alapján könnyen megvizsgálható, hogy mely betűk esetén ért a hálózat alacsonyabb pontosságot. Az eredmények alapján ki lehet jelenteni, hogy a hálózat összes betű esetén képes volt legalább 75% arányban felismerni a helyes osztályt, és sok esetben néhány minta kivételel az adott betűhöz tartozó összes



4.11. ábra. Az egyes betűkre kapott helyes becslések százalékos eloszlása.

minta során helyes becslést adott. Alacsonyabb pontosság azoknál a betűknél fordult elő, ahol a hozzá tartozó kézjel hasonlóságot mutat valamilyen másik kézjellel. Ilyen történt a "C" - "Cs", a "G" - "H" és az "M" - "N" betűk esetében.

Egy másik előfordulása az alacsonyabb pontosságnak, mikor a vizsgált betűnek van statikus vagy dinamikus megfelelője (például ékezet nélküli és ékezetes magánhangzóknál). Ez a probléma legjobban a "E" - "É" és az "I" - "Í" betűknél látszódik. Ezek konfúziós mátrix használatával jobban megvizsgálhatóak, amit a 4.2. táblázat tartalmaz.

Az LSTM hálózat a csak statikus, mozgást nem tartalmazó kézjelek felismerését átlagosan 86.04%-s pontossággal volt képes előrejelezni a tesztminták vizsgálata során. Ez az eredmény a dinamikus, mozgást tartalmazó kézjelek esetén 93.33 %. Ezen két érték alapján el lehet mondani, hogy a hálózat mozgást tartalmazó kézjeleket többször képes helyesen felismerni mint a statikus kézjeleket. Emellett ha egy statikus kézjelnek létezik egy mozgást tartalmazó dinamikus változata (pl. "A"-"Á" jelek), akkor nagy valószínűséggel fordul elő az, hogy a statikus kézjelet a modell a dinamikus megfelelőjével téveszt össze.

A 4.1. táblázaton észrevehető, hogy ékezetes, dinamikus kézjelek esetén a tévesztések száma kisebb mint az ékezet nélküli, statikus kézjeleknél. Mivel az ilyen típusú betűknél a különbség annyi, hogy ékezetes betűk során a jelelő kéz jobbra tolódik, elfogadható ha tévesztés során az ékezetes, vagy épp a nem ékezetes megfelelőjével téveszti össze a hálózat a jelet. A táblázat értékei alapján elmondható, hogy ékezetes karakterek esetén amennyiben tévesztés történt, az majdnem minden esetben a ékezet nélküli párral történt.

Kivétel az "Í" betű esetében figyelhető meg: az 4.2. konfúziós táblázat adatai alapján az látszik, hogy a tévesztések esetén az esetek 66.5%-ában a statikus párral ("I" betűvel), a fennmaradó esetekben a "J" betűvel tévesztett a modell.

Fordított esetben is magas a párosított magánhangzók esetében a tévesztések száma.

Jel	Teszt Hiba %	Pár	Előfordulás
A	10%	Á	100%
Á	6.7%	A	100%
E	23.3%	É	85%
É	0%	E	-
I	20%	Í	100%
Í	20%	I	66.5%
O	16.7%	Ó	19.7%
Ó	6.7%	O	100%
Ö	6.7%	Ő	100%
Ő	0%	Ö	-
U	16.7%	Ú	100%
Ú	13.3%	U	100%
Ü	0%	Ű	-
Ű	0%	Ü	-

4.1. táblázat. Tévesztések megoszlása ékezetes és nem ékezetes magánhangzók esetén.

Megjegyzendő, hogy hasonló értékek figyelhetőek meg a mássalhangzók és kétjegyű párjaik között, mint például "T" ÉS "TY" esetén.

Jellemző adat, hogy amennyiben ezeket a hasonló jelelésű és értelmű betűket egyben kezeljük, akkor a teljes teszthalmazra mért pontosság miként változik: ebben az esetben a mért érték 91.421%, amely 2.71%-al jobb eredmény, mint eredetileg.

Észre lehet venni, hogy jelen eredmények alacsonyabbak, mint a korábban bemutatott csak statikus kézjelek felismerésére készített neurális hálózat. Ennek több lehetséges oka lehet. Egy lehetséges magyarázat lehet a jelenségre, hogy a felismerendő betűk száma jelentősen megnőtt, ezzel együtt a tévesztés esélye is magasabb lett. Ez jól látható az ékezetes, nem ékezetes magánhangzók felismerésekor. Az ékezetes karakterek megjelenésével megnőtt a tévesztések száma a nem ékezetes megfelelőjénél. Hasonló magyarázat lehet a kettős mássalhangzók esetében is, ugyanakkor abban az esetben a tévesztések száma kevesebb, mint a magánhangzók esetén. Meg kell említeni azt is, hogy a sima konvolúciós hálózat esetén a bemenet egyetlen képből, míg CNN LSTM modell esetén a bemenet 10 képből áll. Minél több képből áll a bemenet annál nagyobb a valószínűsége annak, hogy a bemenet egyik képe nem megfelelő minőségű lesz, esetleg zaj jelenik meg a képen, amely zajok felerősödhetnek a képek folyamán.

	A	B	C	CS	D	E	F	G	H	I	K	L	M	N	O	Ö	P	Q	R	S	T	U	Ü	V	W	X	Y	Á	É	GY	Í	J	LY	NY	Ó	Ő	SZ	TY	Ú	Ű	Z	ZS
A	90.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10.0	-	-	-	-	-	-	-	-	-	-	-	-	-
B	-	86.7	-	-	-	-	-	-	-	-	-	-	3.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6.7	-	-	-	-	-	-	3.3	-	-	-	-	-
C	-	-	80.0	10.0	-	-	-	6.7	-	-	-	-	-	-	-	-	-	-	-	-	3.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CS	-	-	10.0	80.0	-	-	-	10.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
D	-	-	-	-	96.7	-	-	-	-	3.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
E	-	-	-	-	-	76.7	-	-	-	-	-	-	-	-	-	-	-	-	-	3.3	-	-	-	-	-	-	-	-	20.0	-	-	-	-	-	-	-	-	-	-	-	-
F	-	-	-	-	3.3	-	96.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
G	-	-	-	-	-	-	-	90.0	10.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
H	-	-	-	-	-	-	-	6.7	83.3	-	-	-	-	-	-	-	6.7	-	-	-	3.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
I	-	-	-	-	-	-	-	-	-	80.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	20.0	-	-	-	-	-	-	-	-	-	-
K	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
L	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
M	-	-	-	-	-	-	-	-	-	-	-	-	76.7	16.7	-	-	-	6.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
N	-	-	-	-	-	-	-	-	-	-	-	-	20.0	80.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
O	-	-	-	-	-	-	-	-	-	-	-	-	-	-	83.3	-	3.3	-	-	-	-	-	-	-	-	-	10.0	-	-	-	-	-	-	-	3.3	-	-	-	-	-	-
Ö	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	93.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6.7	-	-	-	-	-
P	-	-	-	-	-	-	-	-	-	-	-	-	-	6.7	-	-	80.0	13.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Q	-	-	-	-	-	-	-	-	-	-	-	-	-	20.0	-	-	3.3	76.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	83.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	13.3	-	3.3	-
S	-	-	-	-	-	-	-	3.3	-	-	-	-	6.7	-	-	-	-	-	-	86.7	-	-	-	-	-	-	-	3.3	-	-	-	-	-	-	-	-	-	-	-	-	-
T	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	86.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	13.3	-	-	-
U	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	83.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	16.7	-	-	-
Ü	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
V	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	16.7	76.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6.7	-	-
W	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3.3	96.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	13.3	-	-	-	-	-	-	-	-	83.3	-	-	-	-	-	-	-	-	-	-	-	-	3.3	-	-
Y	-	-	-	-	-	-	-	-	-	23.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	76.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Á	6.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	93.3	-	-	-	-	-	-	-	-	-	-	-	-	-
É	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-	-	-	-	-	-	-	-	-
GY	-	-	-	-	-	-	-	-	13.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3.3	-	-	83.3	-	-	-	-	-	-	-	-	-	-	-
Í	-	-	-	-	-	-	-	-	-	13.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	80.0	6.7	-	-	-	-	-	-	-	-	-
J	-	-	-	-	-	-	-	-	-	3.3	-	-	-	-	-	-	3.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	93.3	-	-	-	-	-	-	-	-	-
LY	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-	-	-	-	-
NY	-	-	-	-	-	-	-	-	-	-	-	-	-	3.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	96.7	-	-	-	-	-	-	-
Ó	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	93.3	-	-	-	-	-	-
Ő	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-	-
SZ	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-	-
TY	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-	-	-
Ú	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	13.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	86.7	-	-
Ű	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100.0	-
Z	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	93.3	-
ZS	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	16.7	80.0

4.2. táblázat. Osztályonkénti pontosság mérésének eredményei konfúziós mátrixban százelék formában.

4.6. Konklúzió

A rekurrens neurális hálózat alkalmazásának segítségével valósítottam meg a dinamikus és statikus kézjelek felismerését. A konkrét megvalósítás előtt megvizsgáltam az RNN és LSTM hálózatokat.

A statikus kézjelek felismerése során létrehozott architektúra módosításával, és LSTM rétegek előtti alkalmazásával alkottam meg a rekurrens architektúrát, melynek betanításával a modell a magyar ujjábécé mind a 42 kézjelét képes felismerni.

A hálózat becslésének pontossága a teljes tesztminta-halmazt figyelembe véve 88.6% lett. Az eredmények kiértékelésekor analizáltam a dinamikus jelek eredményeit, összevetettem a tévesztéseket a konfúziós mátrix alapján, és arra a megállapításra jutottam, hogy a statikus jelek és a dinamikus párok közötti tévesztések gyakoriak, amely hibák viszont enyhébb súlyúnak tekinthetőek, mint teljesen eltérő jelentésű jelek esetében.

5. fejezet

Fejlesztői dokumentáció

Az alkalmazás fejlesztése 3 lépésre bontható. A fejlesztés első részében a neurális hálózat tanításához szükséges tanítóminták előállítása és rögzítése történt meg Intel RealSense kamera segítségével. A rögzítés megkönnyítése miatt egy egyszerű rögzítő program lett fejlesztve. A fejlesztés következő fázisában a neurális hálózat betanítása történt meg Tensorflow épülő Keras könyvtár használatával. A betanítást követően az újonnan létrehozott modell használatával a konkrét kézjel felismerést végző alkalmazás létrehozásával zárult a fejlesztés.

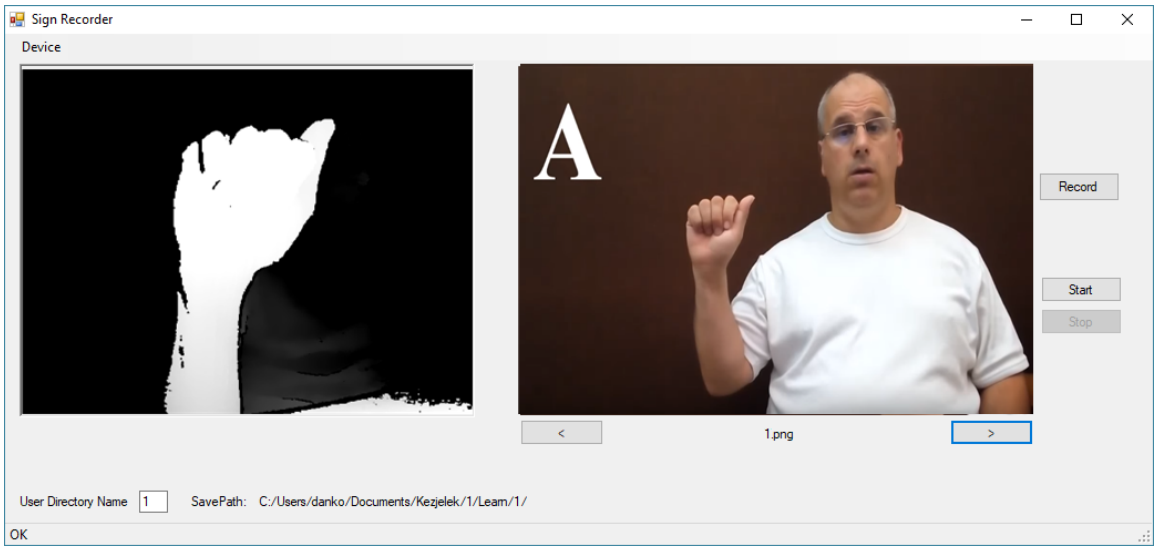
5.1. Tanítóminták rögzítése

A neurális hálózat betanítása rengeteg tanítómintát igényel és ezeket először elő kell állítani. A feladat megkönnyebbítése érdében egy rögzítést végző alkalmazás készült. A program feladata a mélységérzékelő kamera használatával rögzített kézjelek tárolása. Mivel a tanítómintákat jelnyelvet nem ismerő alanyok szolgáltatták, szükség van az éppen rögzített kézjelről egy mintakép, ami alapján képes lesz a kézjel elmutatására a kamera számára az alany. Ugyanakkor, hogy a rögzítés folyamatos és rugalmas legyen, elvárás az is, hogy a program újraindítás nélkül legyen képes különböző kézjel rögzítésére.

5.1.1. A program kinézete és működése

A program kinézete az 5.1. ábrán látható működés közben. A jelelést segítő minta a a program jobb oldalán lévő kép mutatja, míg a kamera képét a program bal oldalán látható kép ábrázolja. A mintakép alatt lévő két ("<" és ">") gomb segítségével lehet változtatni, hogy melyik kézjelről készül a tanítóminta. Mivel a tanítómintákat több alany szolgáltatta, és minden alany esetében külön mappában lett eltárolva a tanítóminták, szükség volt arra is, hogy meg lehessen határozni az alany számát. Ez, illetve a rögzített kézjel száma határozza meg, hogy a tanítóminták hova kerülnek eltárolásra.

A "Start" gomb lenyomásával bekapcsolódik a kamera és a képe megjelenik a kijelzőn, míg a "Stop" gombbal lehet ezt leállítani. A "Record" gomb lenyomása esetén elkezdődik a tanítóminták rögzítése, amennyiben előtte a "Start" gomb le lett nyomva. Ekkor a program 15 darab tanítómintát rögzít és tárol.



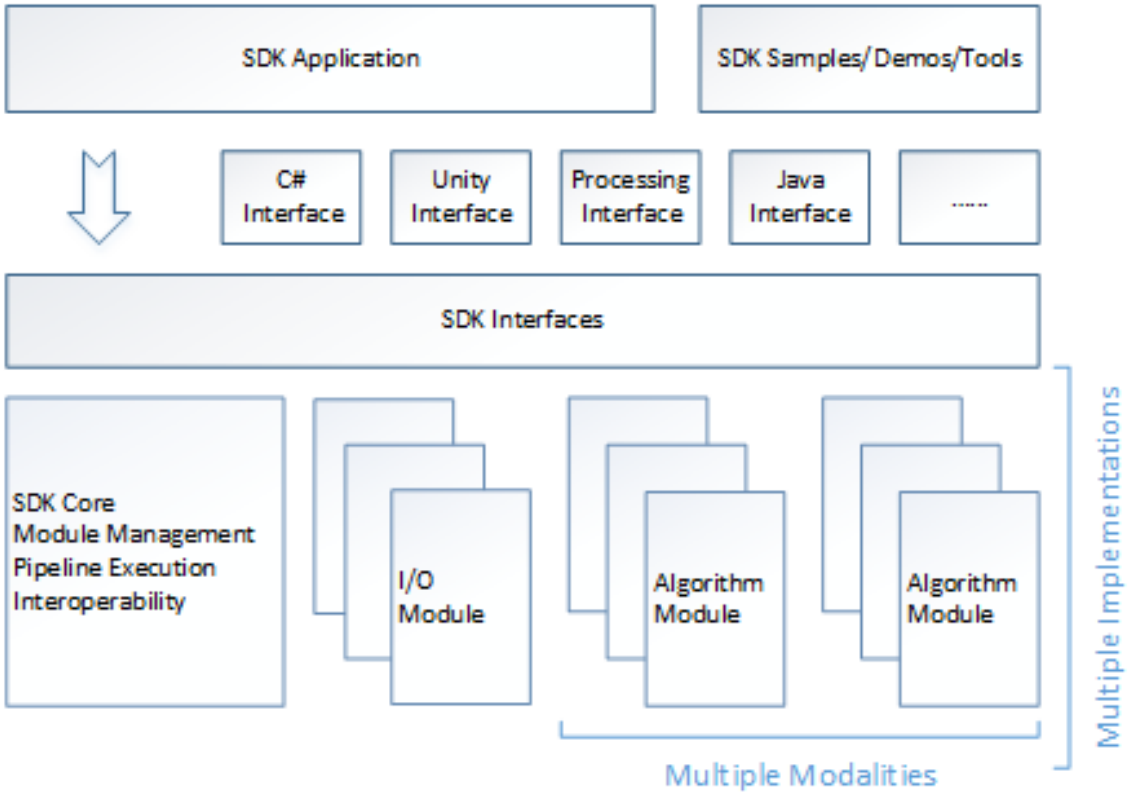
5.1. ábra. Az "A" betűhöz tartozó tanítóminta rögzítése az alkalmazás futása közben

5.1.2. A kamera és a szoftver összehangolása

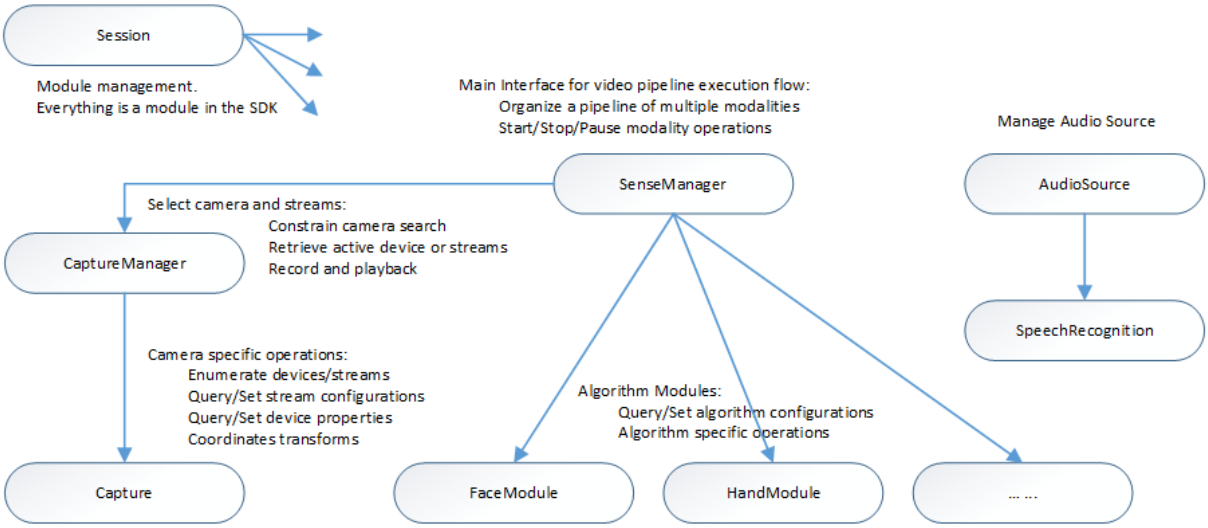
Az alkalmazás C# nyelven íródott és Intel RealSense SDK könyvtárat használ. Az SDK felelős a RealSense kamerához a program hozzáférhessen, illetve használhassa. Az SDK, ahogy az 5.2. ábrán is látható, több komponens rétegből épül fel. A funkciók nagy része a "I/O" és az "algoritmus" modulban találhatók. Az I/O modul visszaadja a kamera képét és megjeleníti azt a kijelzőn, míg az algoritmus modul detektálást és felismerést végző algoritmusokat tartalmazza. Az SDK-ban az "SDK Interfaces" modul feladata, hogy általánosítja az "I/O" és "Algorithm" modulban interfészeit.

Az "SDK Interfaces" több több interfészből épül fel, melynek hierarchiája az 5.3. ábrán látható. A "Session" interfész kezeli a modulokat. Először ennek példányosítását kell elvégezni az alkalmazásban. A kamera képének rögzítése a "SenseManager" interfész használatát igényli, ami az "CaptureManager" interfészt használja, hogy kiválassza a kamerát, illetve mélységi

A "Session"-t megvalósító "PXCMSession" osztály implementációja után a rendszer először feltölti a "Device" menüpont elemeit a számítógéphez kapcsolt kamerák hozzáadásával a "PopulateDeviceMenu" metódus meghívásával. Amennyiben több mint egy kamera van csatlakoztatva az alkalmazás automatikus kiválasztja az menüben lévő első kamerát és azt fogja használni alapértelmezetten.



5.2. ábra. Az SDK egyszerűsített arhcitektúrája



5.3. ábra. Az interfészek hierarchiája

```
var desc = new PXCMSession.ImplDesc();
desc.group = PXCMSession.ImplGroup.IMPL_GROUP_SENSOR;
desc.subgroup = PXCMSession.ImplSubgroup.IMPL_SUBGROUP_VIDEO_CAPTURE;

DeviceMenu.DropDownItems.Clear();

for (var i = 0;; i++)
{
    PXCMSession.ImplDesc desc1;
    if (session.QueryImpl(desc, i, out desc1) < pxcmStatus.
       PXCM_STATUS_NO_ERROR) break;
    PXCMCapture capture;
    if (session.CreateImpl(desc1, out capture) < pxcmStatus.
       PXCM_STATUS_NO_ERROR) continue;
    for (var j = 0;; j++)
    {
        PXCMCapture.DeviceInfo dinfo;
        if (capture.QueryDeviceInfo(j, out dinfo) < pxcmStatus.
           PXCM_STATUS_NO_ERROR) break;
        if ((dinfo.streams & PXCMCapture.StreamType.STREAM_TYPE_DEPTH) ==
            PXCMCapture.StreamType.STREAM_TYPE_DEPTH
            && dinfo.orientation == PXCMCapture.DeviceOrientation.
                DEVICE_ORIENTATION_FRONT_FACING)
        {
            var sm1 = new ToolStripMenuItem(dinfo.name, null,
                MenuExclusiveSelection);
            devices[sm1] = dinfo;
            devices_iuid[sm1] = desc1.iuid;
            sm1.CheckOnClick = true;
            DeviceMenu.DropDownItems.Add(sm1);
        }
    }
}
```

5.1. Forrás. A "PopulateDeviceMenu" metódus egy része

A metódus az "PXCMSession.ImplGroup.IMPL_GROUP_SENSOR" segítségével lehet megkapja a számítógépre kapcsolt kamerák listáját. Ezután ellenőrzi, hogy az első kamera esetén hibaüzenetet kap e a rendszer. Ha igen, akkor a metódus leáll. Majd a többi felismert kamera esetén is megcsinálja ezt az ellenőrzést. Ezután megvizsgálja, hogy a kamera képes e mélységi képek rögzítésére. Amennyiben igen, eltárolja a kamera adatait és hozzáadja a "Device" menübe a kamerát mint lehetséges eszköz. Végül az első kamerát kiválasztja, mint alapértelmezett készülék.

```
PXCMVideoModule.DataDesc inputs;
var resDict = new Dictionary<int, Resolution>();
var tempRes = new Resolution();
var i = 0;
while (m.QueryCaptureProfile(i++, out inputs) >= pxcmStatus.
    PXCM_STATUS_NO_ERROR)
     if (!resDict.ContainsKey(inputs.streams.color.sizeMax.height))
     {
          tempRes.color = inputs.streams.color.sizeMax;
          tempRes.color = inputs.streams.depth.sizeMax;
          resDict.Add(inputs.streams.color.sizeMax.height, tempRes);

          var color = ProfileToString(inputs.streams.color).PadLeft(9);
          var depth = ProfileToString(inputs.streams.depth);
          var sm1 = new ToolStripMenuItem("Color: " + color + " Depth: " +
              depth, null,
               MenuExclusiveSelection);
          profiles[sm1]  = inputs.streams;
          sm1.CheckOnClick = true;
     }
```

5.2. Forrás. A "PopulateProfileMenu" metódus egy része

Ezután a kiválasztott kamera rögzítési profiljait gyűjti ki és jeleníti meg a "Profile" menüpontban a "PopulateProfileMenu()" függvénnyel. Itt eltárolja, hogy a kamera maximum milyen felbontásban képes rögzíteni RGB és mélységi képek esetén. Ezeket a "Profile" menüpontban eltárolja. Ezzel a metódussal befejeződik az UI felépítésével kapcsolatos metódusok.

A "Start" gomb megnyomásával fog elindulni a kamera képének megjelenítése a kijelzőn a "DoRendering()" metódus meghívásával. A "Profile" menüből kiválasztott menüpont adatait a program eltárolja. Ez fogja meghatározni a rögzített kép felbontását. A "StreamColorDepth" metódus végén fogja elkezdeni a program a rögzített képek megjelenítését. A "PXCSenseManager" példányosítása után kezelni kell, hogy milyen típusú képet akar a kamera rögzíteni, ugyanis mélységi kép mellett lehetséges RGB, illetve szegmentált kép rögzítése is. Ezután már folyamatosan meg fog jelenni a kamera képe, egészen addig míg a felhasználó nem nyomja meg a "Stop" gombot.

```
switch (form.primary)
{
    case 1: // depth
        if (sample.depth != null)
        {
            bitmap_data = GetRGB32Pixels(sample.depth, out bitmap_width,
                out bitmap_height);
            form.SetBitmap(0, bitmap_width, bitmap_height, bitmap_data);
            timer.Tick(sample.depth.info.format.ToString().Substring(13) +
                " " +
                        sample.depth.info.width + "x" + sample.depth.info.
                            height);
        }

        break;
}
```

5.3. Forrás. Mélységi kép kiválasztásának kódrészlete

A kamera képének megjelenése és a "Record" gomb lenyomásával megkezdődik a tanítóminták rögzítése a "SetBitmap()" metódus meghívása során. Ez a metódus felelős a képernyőn megjelenő kép előállításáért, képernyőn való megjelenítésért. Paraméterként egy byte tömböt, a kép magasságát, szélességét kapja meg. Ebből a byte tömbből előállít egy bitmapet. Ezt a bitmapet fogja felhasználni a tanítóminták elkészítése során. Mivel egy tanítóminta 10 képből áll, 10 ilyen bitmapre lesz szükség. Ezeket egy listába fogja tárolni a program. Ha a lista eleminek száma elérte a szükséges mennyiséget, egyesével menti el őket a megadott mappába.

Egy rögzítés során egy betűről egymás után 15 darab tanítómintát rögzít a program. Az egyes tanítóminták létrejötte után egy számláló ellenőrzésével nézi meg, hogy a rögzítés során elérte e program a 15 darab mintát. Amennyiben igen, akkor erről a felhasználót egy felugró ablakkal tájékoztatja.

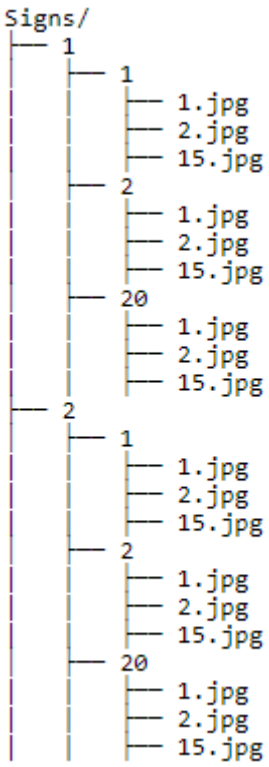
```
if (bitmap != null && mehet && imageSaved<hatar)
{

    if (j == 0)
    {
        bitmap.Save(saveDirectory + imageSaved+ ".jpeg", ImageFormat.Jpeg);
        imageSaved++;

    }
    j++;
    if (j == 4)
    {
        j = 0;
    }
}
else if (imageSaved == hatar)
{
    if (mehet)
    {
        MessageBox.Show("Vege");
    }
    mehet = false;
}
```

5.4. Forrás. Kép rögzítét végző metódus kódrészlete

5.1.3. Tanítóminták struktúrája és átalakítása

A tanítóminták megfelelő struktúrába való rendezése fontos a neurális hálózat betanítása szempontjából. Minden betűhöz tartozó tanítómintákat egy saját könyvtárban kell tárolni. Azonban a minták rögzítése során minden alany esetén egy saját ilyen struktúra formájában lettek elmentve. Ennek a struktúrája látható a 5.4. ábrán, ahol az első szint a alany azonosítóját, a második a rögzített betű számbeli megfelelőjét, míg a harmadik szint a betűhöz tartozó tanítómintát jelenti. Ennek átalakítását meg kell csinálni a hálózat betanítása előtt. Emelett a rögzítés során a tanítómintákat a rögzító program 480x640 pixel felbontásban tárolja, míg a hálózat bemenetére 160x120 pixel felbontású képet vár, ezért a kép felbontásának csökkentését is el kell végezni. Ez legegyszerűbben egy Python nyelven írt scriptekkel lehet megvalósítani.

A képfeldolgozással kapcsolatos feladatok végrehajtása során OpenCV könyvtár lett alkalmazva. A scricp végigmegy minden alany által rögzített összes tanítómintán. A kép beolvasása után először 8 bites szürkeárnyalatos képpé konvertálja, majd utána a felbontást megváltoztatja 160x120 pixelre. A módosítások után az új képet elmenti a megadott helyre.

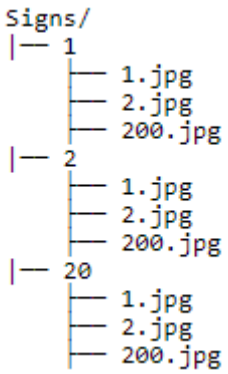


5.4. ábra. Tanítóminták struktúrája az átalakítás előtt

```
def imageProcessing():
    for item in characterDirs:
        pathCharacterDir = pathLearn + item + "/"
        characters = os.listdir(pathCharacterDir)
        for characterImage in characters:
            if os.path.isfile(pathCharacterDir + characterImage):
                im = cv2.imread(pathCharacterDir + characterImage)
                im = cv2.cvtColor(im, cv2.COLOR_RGB2GRAY)
                im = cv2.resize(im, (160, 120))
                cv2.imwrite(pathCharacterDir + characterImage, im)
    print("IMAGE PROCESSING IS DONE")
```

5.5. Forrás. Képfeldolgozást végző metódus kódja

Az egy betűhöz tartozó összes tanítóminta egy mappába való áthelyezése két ciklus segítségével könnyen megvalósítható. A képek átmásolása pedig a "shutil.copy()" metódus meghívásával valósul meg, ahol a fájl előző elérési útját, illetve a másolandó hely elérési útvonalát kell megadni. A scriptek lefutása után a 5.5. ábrán látható struktúra jön létre.



5.5. ábra. Tanítóminták struktúrája az átalakítás után

```
def moveLearnData():
    for item in characterDirs:
        pathCharacterDirLearn = pathLearn + item + "/"
        pathCharacterDirAllLearn = pathLearnAllDirectory + item + "/"
        dir = os.listdir(pathCharacterDirLearn)
        for j in dir:
            shutil.copy(pathCharacterDirLearn + j, pathCharacterDirAllLearn
                + allCharacter + j)
    print("MOVE LEARN DATA IS DONE")
```

5.6. Forrás. A tanítóminták struktúrális átalakítását végző metódus kódja

5.1.4. A tanításhoz szükséges pickle fájl generálása

Egy tanítás elindítása során a hálózatnak szükség van az összes tanítómintára. A tanítóminták memóriába való betöltése hosszú és művelet igényes folyamat. A megfelelő modell létrehozása pedig több tanítást igényel. Azért, hogy az összes tanítóminták betöltését ne kelljen állandóan betölteni a memóriába előlről, pickle létrehozása a javasolt.

A pickle modul bináris protokollokat valósít meg a Python objektumszerkezet szerializálásához. Szerializálás alatt az objektum memóriába konvertálásának folyamatát érthető egy byte stream-be. Ez később vissza szerializálható Python objektummá.

A tanítás során használt pickle fájlok folyamata először az összes tanítóminta elérési útvonalának tárolásával. Emelett eltárolásra kerül a tanítóminta kimenetele is, ami jelen esetben egy betű lesz.

```
def split_images_labels(images_labels):
    images = []
    labels = []
    for (image, label) in images_labels:
        images.append(image)
        labels.append(label)
    return images, labels
```

5.7. Forrás. Pickle előkészítésének egyik metódusa

Ezután következik a tanítóminták a picklebe történő betöltése. Egy ciklussal végig kell menni a tanítóminták elérési útvonalát tartalmazó listán. Itt először az OpenCV-vel megnyitott képet 160x120 tömbbé kell alakítani. Majd "o" változóban tárolt pickle fájlba fel kell tölteni a tanítómintákat a "dumb" metódus meghívásával. Eközben egy másik pickle-ben ("p") a program eltárolja az adott tanítóminta kimenetét.

```
for k, v in enumerate(images):
    img = cv2.imread(v, 0)
    sequence.append(np.array(img, dtype=np.float32))
    indexer += 1
    if indexer == 10:
        o[a_train_seq] = sequence
        a_train_seq += 1

        p[a_train_lab] = gesture
        a_train_lab += 1

        sequence = []

        indexer = 0

        if a_train_seq % 100 == 0:
            print(a_train_seq)
            o.dump()
            o.clear()
            p.dump()
            p.clear()
```

5.8. Forrás. A pickle készítő metódus kódrészlete

A pickle fájlok létrehozásával már ténylegesen megkezdhető a neurális hálózat tanítása, ha a tanítás előtt a pickle betöltése után megfelelő formátummá van szeriálálva.

5.2. Neurális hálózat betanítása

A tanítóminták előállítása után a fejlesztés következő lépése a neurális hálózat létrehozása és betanítása. Az implementáció Python nyelvben Tensorflow és Keras nyílt forráskódú könyvtárak segítségével valósult meg.

5.2.1. Tensorflow és Keras

A TensorFlow egy Google által fejlesztett nyílt forráskódú könyvtár, mely gépi tanulásos algoritmusok megvalósítását teszi lehetővé. A Google rendszerét felhasználták már gépi tanulásos rendszerek létrehozásához különböző területeken, például beszéd felismerés, gépi látás, robotika, információ gyűjtés, természetes nyelv feldolgozás és földrajzi adatok gyűjtésére.

A Keras egy gépi tanulást elősegítő könyvtár, mely a TensorFlow könyvtárra épül és egy magasabb szintű API-t biztosít a használatához. Előnye, hogy átláthatóbb kódot eredményez és lecsökkenti a megírt kód méretét, így jelentősen felgyorsítja a fejlesztést.

A Keras által használt adatszerkezet a modell, amit rétegekbe kell szervezni. A legegyszerűbb model típus a Sequential modell. A modellnek szüksége van a hálózat bemenetének méretére, ezért az első rétegnek rendelkeznie kell ezzel az információval. Későbbi rétegek esetén ez az információ autómatikus megkapja a réteg. A rétegek egymásra pakolása a .add() metódussal lehetséges. Itt kell megadni, hogy a hálózat milyen rétegekkel fog rendelkezni. Az egyes rétegek esetén meg kell adni paraméterben, hogy réteg jellemzőit: Teljesen összekötött (Dense) réteg esetén meg kell adni, hogy a réteg hány neuront tartalmaz, illetve mi a réteg aktivációs függvénye, konvolúciós réteg esetén a szűrők számának, kernel méretének és az aktivációs függvény megadása szükséges. Pooling réteg esetén pedig a lépésköz nagyságát és a pooling méretét kell megadni. Köztes rétegekek között az aktivációs függvény relu, míg az utolsó réteg esetén softmax. Emellett lehetőség van optimalizálni a hálózatot regularizáció és dropout alkalmazásával. A hálózat utolsó rétege lesz a hálózat kimenete.

```
def cnn_model():
    model = Sequential()
    model.add(TimeDistributed(Conv2D(64, (9, 9), activation="relu",
        kernel_regularizer=regularizers.l2()), input_shape=(10, image_x,
        image_y, 1)))
    model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)
        )))

    model.add(TimeDistributed(Conv2D(128, (5, 5), activation="relu",
        kernel_regularizer=regularizers.l2())))
    model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)
        )))

    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(128, (7, 7), activation="relu",
        kernel_regularizer=regularizers.l2())))
    model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)
        )))

    model.add(TimeDistributed(Conv2D(256, (5, 5), activation="relu",
        kernel_regularizer=regularizers.l2())))
    model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)
        )))

    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Flatten()))
    model.add(LSTM(1024, return_sequences=False, dropout=0.5))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(128, activation="relu", kernel_regularizer=regularizers
        .l2()))
    model.add(Dense(43, activation='softmax'))
```

5.9. Forrás. A neurális hálózat rétegeinek kódja

A tényleges betanítás előtt a tanítási folyamatot konfigurálni kell. Ekkor a metódusnak

3 paramétert kell megadni. Az első paraméter az optimizer, ami lehet már létező vagy saját megoldás ami az Optimizer osztály leszármazottja. Utána a veszteség függvényt kell megadni. Az utolsó paraméter során a mérési módját (metric) kell határozni, vagyis, hogy a tanítás eredményessége mi alapján jelenjen meg. Klasszifikáció esetén az érték "accuracy" lesz, vagyis ilyenkor a modell pontsságga lesz megvizsgálva.

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

5.10. Forrás. A modell konfigációját végző kódrészlet

A tanítás a .fit() metódus meghivásával kezdődik meg. Itt kell megadni, hogy mi lesz a hálózat számára a tanító halmaz, a batch méretét, az epochok számát. A tanítás végével az ujonnan keletkezett modellet a .save() függvény meghivásával lehet elmenteni, ahol meg kell adni. Ilyenkor egy HDF5 formátumban menti el a program a modellt.

```
history = model.fit(train_images, train_labels, validation_data=(
    valid_images, valid_labels), epochs=50,
                        batch_size=3,
                        callbacks=callbacks_list, shuffle=True)

    filepath = "Models/rnntest7.h5"
    history.save(filepath)
```

5.11. Forrás. A tanítást elindító kódrészlet

A betanítást követően a modell tesztelése következett. Egy tesztminta becslése a .predict() metódus meghívásával történik. A függvény kimenete egy 42 elemű tömb lesz. A legnagyobb értéke lesz a modell által becsült kézjel, míg a tömb elemei az adott kézjelhez tartozó magabiztossági (confidence) értéket adják vissza.

```
Confidence = model.predict_proba(i)[0]
RealLabel = valid_labels[id]
PredictedLabel = list(Confidence).index(max(Confidence))
```

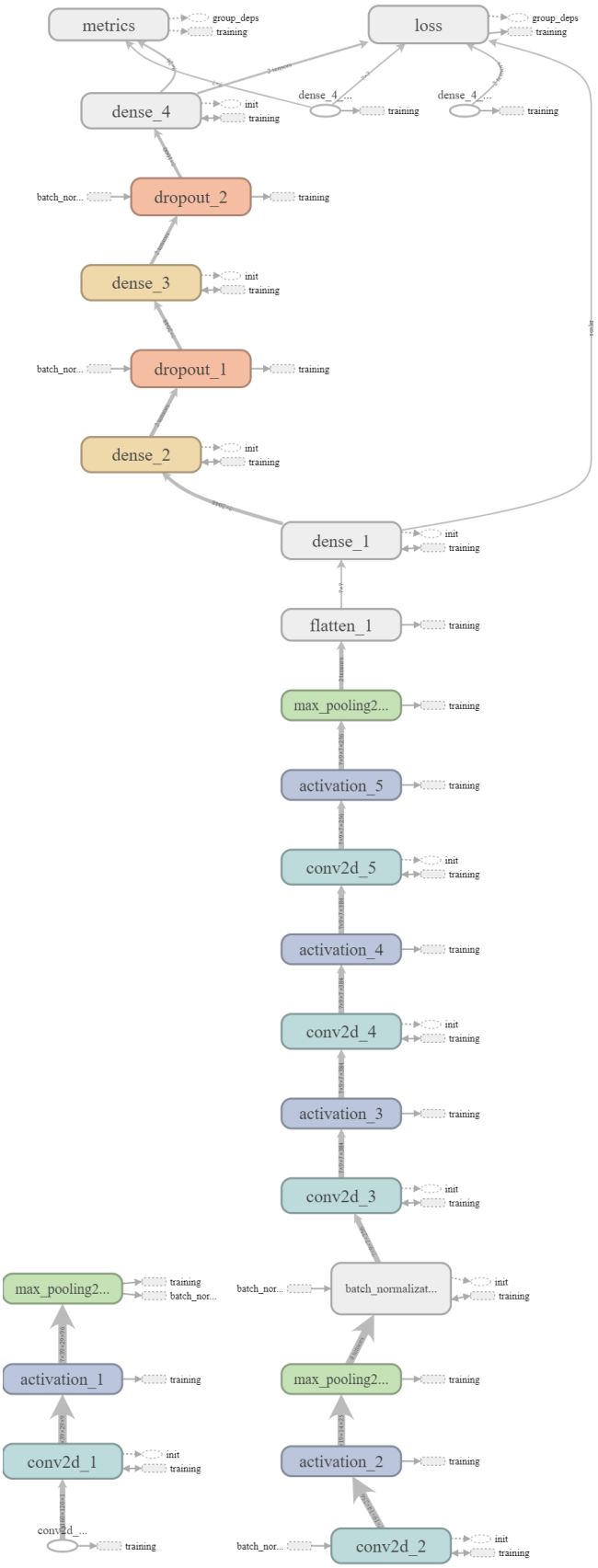
5.12. Forrás. Becslést végző kódrészlet

5.2.2. TensorBoard

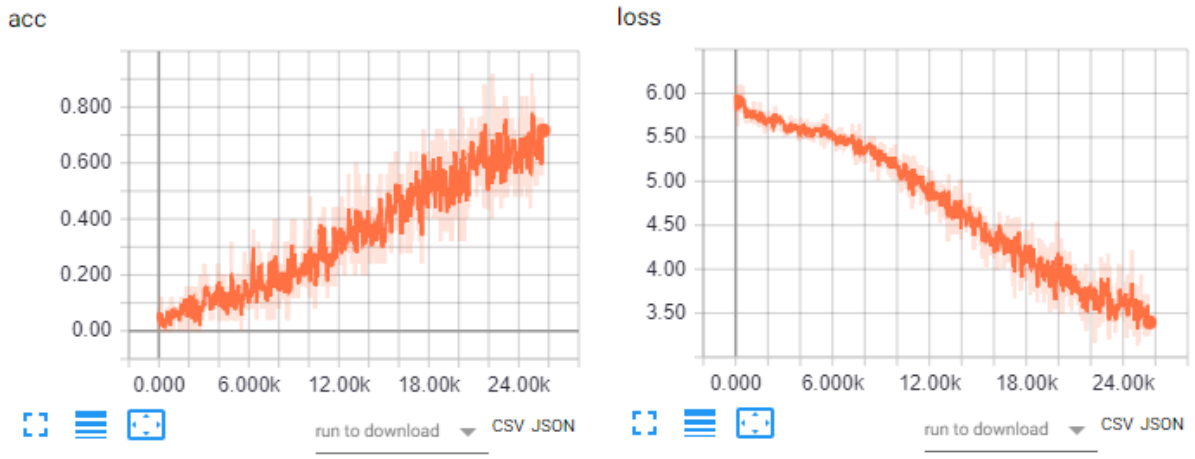
Tensorflow és Keras használata mély neurális hálózatok tanítása esetén komplex folyamat. A TensorBoard, mely egy vizualizációs kiegészítő, segít abban, hogy megértse és debugolja a hálózatot a felhasználó. TensorBoard segítségével könnyedén megjeleníthető a TensorFlow által generált gráf (5.6. ábra), tanítás során keletkező adatokból különféle diagramokat (5.7. ábra) készít. A TensorBoard elindítása a parancssorból történik:

tensorboard –logdir utvonal

A böngészőbe a megfelelő címet megadva látható a tensorboard kezelőfelülete.



5.6. ábra. A TensorBoard által készített gráf



5.7. ábra. A TensorBoard által készített diagramok

5.3. Kézjelfelismerő program

A tanítást követően kapott modell segítségével a ténylegleges alkalmazás fejlesztése következik. A program csakúgy mint a tanítóminták rögzítésénél, C nyelven íródott és az Intel RealSense SDK-t használja. Emellett TensorFlowSharp könyvtár lett alkalmazva a fejlesztés során, hogy az alkalmazás a kamera képeit képes legyen átadni az újonnan létrejött neurális hálózatnak és az alapján becsléseket hozni.

A TensorFlowSharp a TensorFlow és a .Net keretrendszer összekötését valósítja meg. A könyvtár magába foglalja a TensorFlow alacsony szintű használatát, emelett a Pythonban írt Keras vagy TensofFlow modellek megnyitását, futtatását, módosítását.

A kamera képének megjelenítésére a rögzítés során használt alkalmazás kódrészlete lett újra használva, kisebb módosításokkal. A program mostantól nem fogja eltárolni a kamera képét, hanem csak átadja a neurális hálózatnak. Emellett a rögzítés során használt segéd képek is kikerült a programból. Helyette a modell által becsült kézjel szöveges megfelelője jelenik meg az alkalmazásban.

```
public static void SetupTensor()
{
    Start();
    FirstPredict("C:\\Users\\danko\\Documents\\test");
    setupIsDone = true;
    MessageBox.Show("Process for Sign Recognition is done!");
}
```

5.13. Forrás. Első becslést végző kódrészlet

A program indítást követően meghívja a "Tensorflow" statikus osztály SetupTensor() nevű metódusát A metódus először a "Start()" függvény meghívásával betölti a modellt, ami egy ".pb" kiterjesztésű fájlt. A modell betöltése után egy becslés csinál előre megadott bemenettel a "FirstPredict()" metódussal. Erre azért van szükség, mert első becslés időtartama körülbelül egy perc is lehet, utána azonban egy becslés kevesebb mint 1 másodperc alatt is elvégez. A függvény paramétere a tesztmintákat tartalmazó mappa elérési

útvonala. A meghívást követően a mappában lévő összes ".jpeg" formátumú fájl elérési útját eltárolja egy string tömbbe, majd ezeket "Emgu" könyvtár segítségével először szürkeárnyalatos formában megnyitja, majd a metódus átkonvertál minden képet 160x120x1 formátumú byte mátrixxá, amit majd tensorrá alakítani.

```
public static void FirstPredict(string file)
{
    string[] filesTesting = System.IO.Directory.GetFiles(file, "*.jpeg",
        System.IO.SearchOption.AllDirectories);

    var matrix = new List<byte[,,]>();
    for (int i = 0; i < filesTesting.Length; i++)
    {
        Mat img = CvInvoke.Imread(filesTesting[i], ImreadModes.Grayscale);
        Image<Gray, Byte> im = img.ToImage<Gray, Byte>();
        matrix.Add(im.Data);
    }

    byte[][,,] matr = matrix.ToArray();
    float[,,,] asd = new float[10, 120, 160, 1];

    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 120; j++)
        {
            for (int k = 0; k < 160; k++)
            {
                asd[i, j, k, 0] = matr[i][j, k, 0];
            }
        }
    }

    TensorFlow.TFTensor tensor = asd;
    Predict(tensor);
}
```

5.14. Forrás. A FirstPredict metódus kódja

Ezt a formátumot a betöltött modell már képes a bemenetként fogadni és véghez lehet vinni a becslést a "Predict()" metódussal. "Predict()" során a modell első rétegére adjuk a bemenetet, ami végigmegy a hálózat összes rétegén. Az utolsó, output, réteg lesz a hálózat becslése. Ez lesz majd a metódus visszatérési értéke.

```
public static float[,] Predict(TFTensor tensor)
{
    runner = session.GetRunner();
    runner.AddInput(graph["time_distributed_1_input"][0], tensor);
    runner.Fetch(graph["dense_3/Softmax"][0]);
    output = runner.Run();

    var vecResults = output[0].GetValue();
    float[,] results =(float[,]) vecResults;

    return results;
}
```

5.15. Forrás. Predict metódus kódja

A tényleges jelfelismerés akkor indul el miután a kamera be lett kapcsolva, illetve a legelső becslés megtörtént. Ameddig ezek a feltételek nem igazak a program addig nem fog belemenni az elágazásban. Az elégazás elején először a kamera által rögzített képet az Emgu könyvtárral szürkeárnyalatos képpé alakítja, majd 160x120 pixel felbontásra csökkenti azt. Majd egy sor adatszerkezetbe tárolja el ezeket a képeket. A kézjel felismerés akkor fog elkezdődni ha a sor tartalma 10 elemű. Ezután a sor tartalmát betölti egy 10x120x160x1 felbontású byte mátrixszá. Ebből a program egy tensort csinál amit már a hálózat bemenetére lehet adni. Ezután meghívja a Predict() metódus, az újonnan létrehozott tensor paraméterrel.

```
if (j == 0 && tensorflow.setupIsDone && mehet)
{
    var im = new Image<Gray, byte>(bitmap);
    im = im.Resize(160, 120, Inter.Linear);

    photos.Enqueue(im.Data);
    if (photos.Count > 10)
    {
        photos.Dequeue();
        var matr = photos.ToArray();
        var asd = new float[10, 120, 160, 1];

        for (var i = 0; i < 10; i++)
        for (var j = 0; j < 120; j++)
        for (var k = 0; k < 160; k++)
            asd[i, j, k, 0] = matr[i][j, k, 0];

        lock (photoLock)
        {
            TFTensor tensor = asd;
            var task = new Task(() =>
            {
                var res = tensorflow.Predict(tensor);
                var maxId = 0;
                for (var i = 0; i < res.GetLength(1); i++)
                    if (res[0, maxId] <= res[0, i])
                        maxId = i;

                updateLabels(maxId.ToString());
            });
            task.Start();
        }
    }
```

5.16. Forrás. Becslés előkészítésének kódja valós kézjelfelismerés során

A metódus kimenete egy 43 elemű mátrix lesz, ahol egy elem a modell az adott betűhöz tartozó magabiztosságát jelenti. Ezután ki kell választani a legmagasabb becslési értékkel rendelkező betűt. Majd az UpdateLabels() meghívásával megjeleníti a betűt a kijelzőn.

5.3.1. Hibaüzenetek

A szoftver a felhasználó számára több hibaüzenetet is küld. Amennyiben a program futása közben a kamera nincs elindítva, vagyis a kijelzőn nem jelenik meg a kamera képe, illetve ha a program elindítása során a háttérban futó első becslés még nem ért véget, akkor a program egy felugró üzenetet küld a felhasználónak.

```
private void CameraError(object sender, EventArgs e)
{
    if (!tensorflow.setupIsDone && !stop)
        MessageBox.Show("Camera is not started or process for Sign
            Recognition is not finished.", "ERROR",
             MessageBoxButtons.OK, MessageBoxIcon.Error);
    else
    {
        mehet = true;
    }
}
```

5.17. Forrás. Lehetséges hibaüzenet kódja

A program elindítás után először megvizsgálja, hogy a számítógéphez van e csatlakoztatva kamera. Ha a státusz tartalma, ami a vizsgálat után frissül a vizsgálat eredményétől függően, hibát tartalmaz, akkor egy felugró ablak formájában figyelmezteti a felhasználót erről.

```
public void UpdateStatus(string status)
{
    Status2.Invoke((Action)(() => { StatusLabel.Text = status; }));
    if (StatusLabel.Text.Contains("Failed"))
    {
        MessageBox.Show("Camera is not connected or found.", "ERROR",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

5.18. Forrás. Lehetséges hibaüzenet kódja

6. fejezet

Felhasználói dokumentáció

A program a magyar jelnyelvi ujjábécé kézjeleinek felismerésére képes és a kézjel szöveges megfelelőjét. A felhasználó a kamera felé kell mutatnia a kézjelet a jobb kezével. A kéz pozicionálása során igyekezni kell, hogy lehetőleg csak a jelelő kéz legyen benne a kamera képében. Ebben segít az alkalmazás úgy, hogy megjeleníti a kamera képét a kijelzőn. Ezután a program feladata, hogy a kamera által rögzített kézjelet felismerje és megjelenítse a kijelzőn szöveges formában.

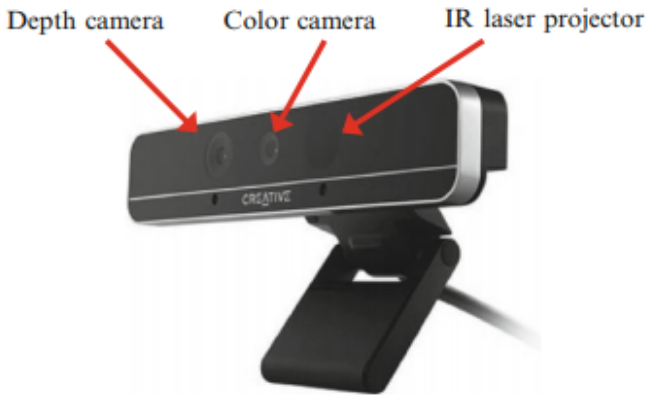
6.1. Futási környezet leírása

Az alkalmazás futtatása Windows operációs rendszer alatt lehetséges. Ahhoz, hogy a program rendeltetésszerűen tudjon működni, szükség van egy Intel RealSense mélységérzékelő kamerára, illetve egy USB 3.0 porttal rendelkező számítógépre. Enélkül a program nem képes a kézjelek felismerésére, viszont a program futtatása továbbra is lehetséges.

6.1.1. Intel Realsense

Az Intel 2014-ben mutatta be az Intel RealSense elnevezésű mélységérzékelő kamerát. A cég célja egy olyan készülék elkészítése volt, mely képes megérteni és feldolgozni a következő kommunikációs módokat: kéz, arc, hang és környezet. Az eszközben található szenzorok lehetővé teszik az arc és gesztus felismerést, a kéz és ujjak követését, a hangfeldolgozást, illetve kiterjesztett valóság megvalósítását.

A kamera, ahogy a 6.1. ábrán is látható, egy színes, illetve egy mélységi kamerából (melyet egy IR kamera és IR projektor alkot), és több mikrofonból épül fel. A mélységi kamera VGA (640x480) felbontásban, 20-120 cm mélység tartományban képes rögzíteni, akár 120 Hz-es időbeli felbontással. Ezek a jellemzők lehetővé teszik a fej és kéz követést, gesztusfelismerést. A IR projektor változó szélességű függőleges fénycsíkokat vetít ki három különböző fényerő- vagy tartomány szinten. A színes kamera segítségével az eszköz képes akár Full-HD minőségben rögzíteni képet 30 fps sebességgel.



6.1. ábra. Az Intel RealSense kamera felépítése

6.2. Az alkalmazás felépítése

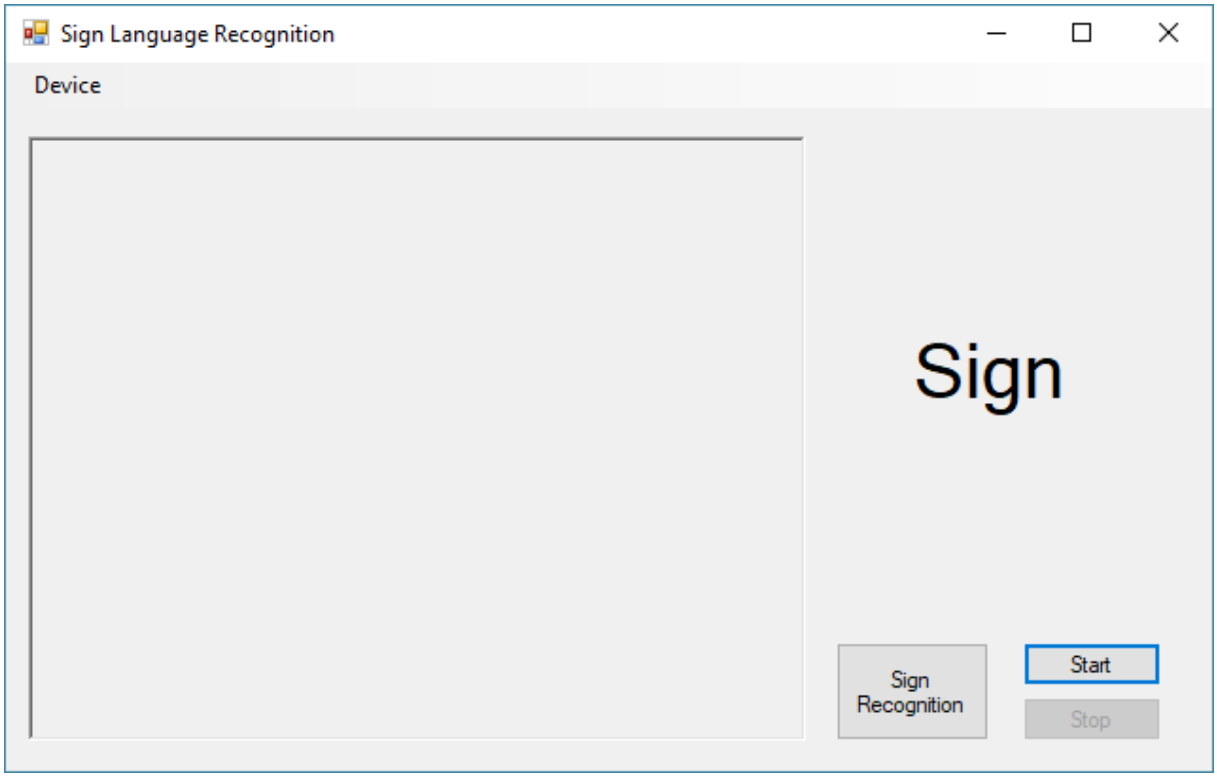
A program egyetlen ablakból épül fel, mely az indítás után rögtön megjelenik. A 6.2. ábrán látható ennek a felépítése.

A "Device" menüpont megnyomásával egy lista jelenik meg, ami a számítógégre kapcsolt RealSense eszközöket listázza ki. Itt kell kiválasztani a használni kívánt eszközt. Ha csak egy kamera van a számítógépre csatlakoztatva, az eszköz manuális kiválasztása nem szükséges, az alkalmazás automatikus kiválasztja azt. Amennyiben egy ilyen eszköz sincs csatlakoztatva a lista üres lesz és az alkalmazás funkcióját tekintve nem lesz használható.

A program 3 gombbal rendelkezik. A "Start" gomb felel a kamera elindításáért. A lenyomása esetén a mélységérzékelő kamera képe megjelenik a bal oldalt látható dobozban, amennyiben a csatlakoztatva van a kamera. A kamera képének megjelenése után a gombra nem lehet majd kattintani. Ez egészen addig lesz igaz amíg a kamera működésben van. A "Stop" gomb felel a kamera leállításáért. Innentől kezdve a kamera képe nem jelenik meg a kijelzőn, míg a felhasználó nem nyomja meg újra a "Start" gombot. A gombot addig nem lehet lenyomni, míg a kamera nem kezdett el működni. A program harmadik gomb a "Sign Recognition" felel a kézjelek felismerését végző folyamat elkezdéséért. Ez csak akkor indul el ténylegesen ha a kamera képe megjelenik a bal oldali dobozban. Megnyomást követően az alkalmazás elkezdi felismerni a jelnyelv kézjelit.

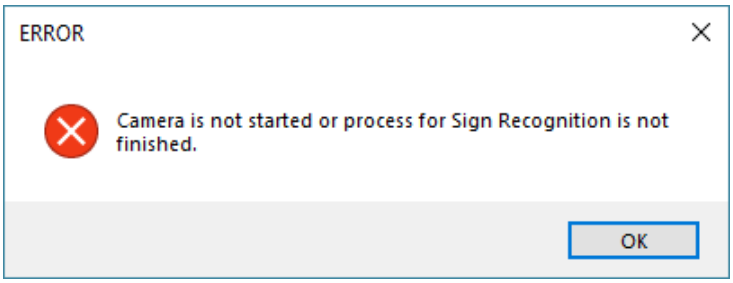
A "Sign" felirat szövege a "Sign Recognition" gomb lenyomása után folyamatosan változik. A szöveg tartalmát a neurális hálózat által becsült kézjel szöveges megfelelője határozza meg. Minden esetben a magyar ábécé egyik betűje fog megjelenni. A 6.3. ábrán látható, hogy a rendszer az "A" betűhöz tartozó kézjelet ismerte fel helyesen.

A program bal oldalán látható dobozban fog megjelenni a mélységérzékelő kamera által rögzített kép. A rögzített kép szürkeárnyalatos formában jelenik meg a felhasználó számára, ahol már csak a jelelő kéz jelenik meg, ugyanis azt a kamera már elvégezte a kéz szegmentálását és a háttér eltüntetését. A kamera képének megjelenítésével lehetőség van folyamatosan megfigyelni a jelelő kezet, illetve pontosítani a kéz pozícióját a hatékonyabb jelfelismerés érdekében. A 6.3. ábra ábrázolja, hogy a kamera képét miként jeleníti meg a program.

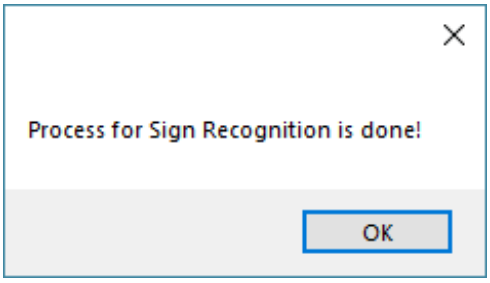


6.2. ábra. A program felépítése az indítást követően

6.3. ábra. A program kézjel feismerés közben



6.4. ábra. Hibaüzenet a "Sign Recognition" gomb nem megfelelő időben történő lenyomásakor



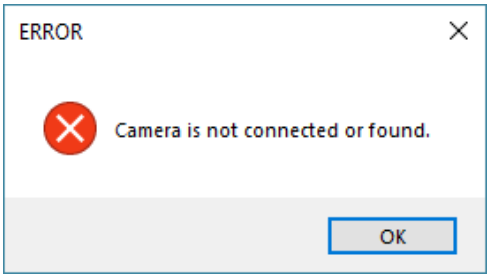
6.5. ábra. Üzenet miután a szükséges a kézjelfeismeréshez szükséges háttérfolyamat befejeződött

6.3. A program használata

A program elindítása előtt szükség van a kamera csatlakoztatására egy USB 3.0 portra. Ez az előfeltétele a megfelelő működéshez, enélkül nem lehetséges a kézjelek felismerésre. A program elindítását követően a 6.2. ábrán látható ablak jelenik meg. Ezt követően az alkalmazás a háttérben elindít egy folyamatot ami a kézjelek felismerését végzi majd el. Ez egy körülbelül egy percet vesz igénybe. Amennyiben a folyamat vége előtt a "Sign Recognition" gombra kattint a felhasználó, az alkalmazás az 6.4. ábrán látható üzenetet fogja megjeleníteni egy felugró ablakban. Amint ez a háttérfolyamat véget ér, a rendszer egy felugró ablakban tájékoztatja erről a felhasználót (6.5. ábra). Ugyanakkor a kamera képe megjeleníthető már ilyenkor is a "Start" gomb megnyomásával. Ha nincs csatlakoztatva kamera a számítógéphez a 6.6. ábrán látható szöveggel egy ablak fog megjelenni.

Miután a háttérfolyamat befelyeződött, egy kamera van csatlakoztatva a számítógéphez, illetve a felhasználó megnyomta a "Start" gombot, a "Start Recognition" gombra kattintva a 6.3. ábra tartalma fog megjelenni a felhasználó számára. Ekkor kezdődik el a kézjelek felismerése. A felhasználó a kamerának mutatott kézjelei az ablakban megjelennek, ezt pedig a rendszer megpróbálja helyesen felismerni. Az általa helyesnek vélt kézjel szöveges megfelelőjét pedig megjeleníti. A 6.3. ábrán a felhasználó az "A" betű kézjelét jeleli és annak képe illetve szöveges megfelelője jelenik az ablakban.

Amennyiben a felhasználó le akarja állítani a programot, abban az esetben a "Stop" gombot kell megnyomnia. Ilyenkor a kamera a rögzítést leállítja és az utolsó rögzített kép (frame) képe jelenik meg. Ekkor a kézjelek felismerését végző folyamat is leáll és a legutoljára felismert kézjel szöveges megfelelőjét írja ki a kijelzőre. Ha újra el akarja indítani a kézjel felismerést meg kell ismét meg kell nyomni a "Start" gombot, illetve utána a "Sign Recognition" gombot.



6.6. ábra. Hibaüzenet a "Start" gomb nem lenyomásakor, ha nincs kamera csatlakoztatva vagy felismerve

7. fejezet

Összefoglalás

Szakdolgozatomban a magyar jelnyelvi ábécé betűinek felismerésére készítettem megoldást. Első lépésként megvizsgáltam az elérhető módszereket, majd azt a megállapítást tettem, hogy mesterséges intelligencián alapuló, felügyelt gépi tanulási megoldást kell terveznem.

Így került kiválasztásra a konvolúciós neurális hálózat architektúrája, amely bemeneteként strukturált adatok, például képkockák megadhatóak. A kéz szegmentálására kétféle megoldást vizsgáltam meg: az end-to-end deep learning módszerét, amikor az alkalmazott neurális hálózatra, a tanulásra bízzuk a szegmentációs lépést, valamint alkalmas eszközből kinyert mélységi információkkal is elvégeztem a tanítást.

A tanítás elvégzése után megállapítottam, hogy a mélységi információk alkalmazása pontosabban és nagyobb magabiztossággal rendelte a validációra alkalmazott mintákat a megfelelő osztályokba, így a későbbiekben is ez a módszer került használatra.

Ezután a statikus, mozgást nem tartalmazó kézjelek felismerésére elemeztem több konvolúciós neurális háló architektúrát, majd összehasonlítottam őket egy saját, kifejezetten erre a problémára készített saját felépítéssel. Az eredmények alapján arra a következtetésre jutottam, hogy az általam tervezett architektúra használatával lehet a legmagasabb pontosságot elérni.

Mivel az ujjábécé mozgást tartalmazó kézjelekkel is rendelkezik, megvizsgáltam a célra a szakirodalomban általánosan javasolt rekurrens hálózati felépítés alkalmazhatóságát. A rekurrens hálózatok egyik speciális típusával, LSTM (hosszú-rövid távú memória) elemeket tartalmazó neurális hálózat segítségével valósítottam meg a statikus és a dinamikus kézjelek osztályozását.

Miután a neurális hálózat betanítása befejeződött, egy asztali alkalmazás fejlesztését kezdtem el, ami a tanítás után létrehozott modell és egy Intel RealSense kamera segítségével megvalósítja a magyar ujjábécé kézjeleinek felismerését. A fejlesztés lépéseit ledokumentáltam, majd egy felhasználói dokumentációt is készítettem a programról azt követően.

7.1. További kutatási célok

A dolgozat megvalósítása során csupán a magyar ujjábécé 42 darab kézjelének felismerése történt meg. Triviális kutatási célkitűzés lehet az általános magyar jelnyelv általános

célú felismerése, amely azonban sokkal több kézjelből épül fel. Ezek bonyolultsága és sokszerűsége a probléma komplexitását nagyban növelik. Jól jellemzi ezt, hogy a probléma megoldásán dolgozó startupok komplett kutatócsoportokkal dolgoznak együtt, és több szenzorral vizsgálják az alanyok kézfejei mellett a testtartásukat és arckifejezésüket is.

Jelen eredmények relatíve kis méretű tanítóminta halmaz alkalmazásával születtek. Fontos cél, hogy további alanyok is bevonhatóak legyenek a kutatásba, hiszen a szélesebb adathalmaz jobb általánosítóképességet, és ennek köszönhetően nagyobb pontosságot is eredményezne. Nehézség, hogy a bemeneti adatok a magyar daktil ujjábécé kézjeleiről készült mélységi felvételekből állnak, amelyekről nem létezik nemzetközi dataset, és az otthoni elkészítés is nehézkes, eszköz hiányában nem lehetséges.

A területen nagy médiafigyelmet élvező startup vállalkozások [11] [12] mintájára célszerű lehet több szenzor egyidejű alkalmazása, így a hasonló jelelésű betűk közti határvonalak a bemenetek nagyobb dimenzionalitása hatására eltávolodnának, így a tévesztések száma általánosságban csökkenthető lenne.

A deep learning kutatásának olyan izgalmas, új eredményekkel rendelkező területeivel is szeretnék foglalkozni, mint a transfer learning [50], amely lényege hogy egy korábban már betanított neurális hálózat tudására építkezve történik az új képességek elsajátítása. Érdemes lenne annak vizsgálata, hogy például az amerikai jelnyelv (american sign language - ASL) felismerésére betanított hálózat mily módon alakítható át egy a magyar jelnyelvet osztályozni képes modellé.

A mélytanulásos technika alapja a nagy bemeneti adat, amennyiben ez nem áll rendelkezésre, az a hatékonyságban jelentkezik. A one-shot learning [51] technika lényege, hogy a modell a bemenetek (esetünkben kézjelek) osztályozása helyett egy alacsonyabb dimenzionalitású reprezentációra képez le. A modell ezen reprezentációkat képes összehasonlítani egymással, felismerve az azonos kézjeleket. Ilyen jellegű gépi tanulási megoldás készítésének eredményeként egy adatbázis alapú, könnyen bővíthető alkalmazás készíthető, amely esetén nem szükséges kézjelenként sok tanítóminta.

7.2. Köszönetnyilvánítás

A szakdolgozat alapjául az Óbudai Egyetem XLVIII. Tudományos Diákköri Konferenciájára készített "A magyar jelnyelvi ujjábécé kézjeleinek felismerése rekurrens konvolúciós neurális hálózat segítségével" című dolgozat eredményei szolgálnak, amely elkészítése során a témavezetői tevékenységet Kertész Gábor látta el.

Köszönettel tartozom az Óbudai Egyetem GPGPU programozás kutatócsoportjának, hogy a tanításhoz szükséges eszközöket rendelkezésemre bocsátották, valamint hogy értékes szakmai tanácsaikkal segítették a kutatásom.

Ki szeretném emelni a hallgatótársak és kollégáim segítségét, akik a kutatásban részt vettek, a tanítóminták alanyai voltak. Munkájukért köszönettel tartozom.

A dolgozathoz az Óbudai Egyetem XLVI. Tudományos Diákköri Konferenciájára Szvatkó Árpáddal közösen készített „Daktil ujjábécé kézjeleinek feldolgozása kamera segítségével” című dolgozatunk szolgáltatta az alapötletet, amely dolgozat elkészítése során a témavezetői tevékenységet Cserfalvi Annamária és Dr. habil. Molnár András látták el.

8. fejezet

Conclusion

In my thesis, I made a solution for recognizing the Hungarian fingerspelling sign language. As first step, I analysed the available methods, then decided that a supervised machine learning solution based on artificial intelligence had to be implemented.

Based on that Convolution Neural Network was selected, where sturctured data, like image frames, can be added as input source. Two ways of hand segmentation were examined: the end-to-end deep learning method, where segmentation is done by the neural network, as well as depth information, which was obtained from a suitable device.

After the training I stated, that the use of depth information has assigned the validation samples to the appropiate classes more accurately and more confidently. so this method was used later.

After that I analyzed several Convolutional Neural Network architecture to recognize the static signs, where there is no movement during marking, and then compared them to my own architecture, specifically created for this problem. Based on the results, I came to the conclusion that using the architecture I created, can achieve the highest accuracy.

Because the fingerspelling sign language contains signs with movement, I examined the applicability of the recurrent network structure, recommended in the literature. I implemented the classification of static and dynamic sign recognition with a network containing LSTM (Long-Short-Term-Memory) layers, which is a a special type of recurrent networks.

After the training was completed, I started to develop a desktop application, with the help of the model created after the training and an Intel RealSense camera to recognize the Hungarian sign language. I documented the steps of the development, then I made a user documentation about the program after that.

Irodalomjegyzék

[1] V. Iván, *A jelnyelv elmélete és módszertana.* Siketek és Nagyothallók Országos Szövetsége, 1995.

[2] B. S. Parton, „Sign language recognition and translation: A multidisciplined approach from the field of artificial intelligence," *JOURNAL OF DEAF STUDIES AND DEAF EDUCATION*, vol. 11, 2005.

[3] T. Starner, J. Weaver, and A. Pentland, „Real-time american sign language recognition using desk and wearable computer based video," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 12, pp. 1371–1375, 1998.

[4] P. Viola and M. Jones, „Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–I.

[5] B. C. Bedregal, A. C. Costa, and G. P. Dimuro, „Fuzzy rule-based hand gesture recognition," in *IFIP International Conference on Artificial Intelligence in Theory and Practice*. Springer, 2006, pp. 285–294.

[6] G. D. Kessler, L. F. Hodges, and N. Walker, „Evaluation of the cyberglove as a whole hand input device," 1995.

[7] K. Thomas, „Glove lends the deaf a hand," *USA Today. Retrieved October*, vol. 10, no. 2002, pp. 2002–0, 2002.

[8] J. L. Hernandez-Rebollar, N. Kyriakopoulos, and R. W. Lindeman, „A new instrumented approach for translating american sign language into sound and text," in *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*. IEEE, 2004, pp. 547–552.

[9] P. C. U. Murillo, R. J. Moreno, and J. O. P. Arenas, „Comparison between CNN and Haar Classifiers for Surgical Instrumentation Classification," *Contemporary Engineering Sciences*, vol. 10, no. 28, pp. 1351–1363, 2017.

[10] Y. LeCun, Y. Bengio, and G. Hinton, „Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[11] „SignAll," http://www.signall.us, accessed: 2018-11-07.

[12] „KinTrans," http://www.kintrans.com, accessed: 2018-11-07.

[13] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning.*, ser. Adaptive computation and machine learning. MIT Press, 2012.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ser. Adaptive computation and machine learning. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

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

[16] N. Buduma and N. Locascio, *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms.* " O'Reilly Media, Inc.", 2017.

[17] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.).* Pearson Education, 2010.

[18] W. S. McCulloch and W. Pitts, „A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[19] X. Glorot and Y. Bengio, „Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[20] N. Shukla and K. Fricklas, *Machine learning with TensorFlow.* Manning, 2018.

[21] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural networks and learning machines.* Pearson Upper Saddle River, NJ, USA:, 2009, vol. 3.

[22] D. Kriesel, *A Brief Introduction to Neural Networks*, 2007. [Online]. Available: http://www.dkriesel.com

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, „Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, „Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] T. Glasmachers, „Limits of end-to-end learning." *CoRR*, vol. abs/1704.08305, 2017. [Online]. Available: http://dblp.uni−trier.de/db/journals/corr/corr1704.html#Glasmachers17

[26] „E2E speech recognition," (http://blog.easysol.net/building−ai−applications, accessed: 2018-09-16.

[27] Y. LeCun *et al.*, „LeNet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, p. 20, 2015.

[28] L. Bottou, „Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[29] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, „Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, „Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[31] V. Bheda and D. Radpour, „Using deep convolutional networks for gesture recognition in american sign language." *CoRR*, vol. abs/1710.06836, 2017. [Online]. Available: http://dblp.uni−trier.de/db/journals/corr/corr1710.html#abs−−1710−06836

[32] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, „Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: http://dblp.uni−trier.de/db/journals/corr/corr1207.html#abs−1207−0580

[33] S. Park and N. Kwak, „Analysis on the dropout effect in convolutional neural networks," in *Asian Conference on Computer Vision*. Springer, 2016, pp. 189–204.

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Wilson, „Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[35] I. Sutskever, J. Martens, and G. E. Hinton, „Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017–1024.

[36] K. Cho, B. van Merrienboer, C. Gülcehre, F. Bougares, H. Schwenk, and Y. Bengio, „Learning phrase representations using rnn encoder-decoder for statistical machine translation." *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://dblp.uni−−trier.de/db/journals/corr/corr1406.html#ChoMGBSB14

[37] A. Robinson and F. Fallside, *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering, 1987.

[38] R. J. Williams and D. Zipser, „Gradient-based learning algorithms for recurrent networks and their computational complexity," *Backpropagation: Theory, architectures, and applications*, vol. 1, pp. 433–486, 1995.

[39] P. J. Werbos, „Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[40] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, „Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Neural Networks*, Kremer and Kolen, Eds. IEEE Press, 2001.

[41] Y. Bengio, P. Simard, and P. Frasconi, „Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[42] S. Hochreiter and J. Schmidhuber, „Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[43] F. A. Gers, J. Schmidhuber, and F. Cummins, „Learning to forget: Continual prediction with LSTM," *Neural Computation*, vol. 12, pp. 2451–2471, 2000.

[44] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, „A novel connectionist system for unconstrained handwriting recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 5, pp. 855–868, 2009.

[45] H. Sak, A. Senior, and F. Beaufays, „Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth annual conference of the international speech communication association*, 2014.

[46] K. Kawakami, „Supervised sequence labelling with recurrent neural networks," Ph.D. dissertation, PhD thesis. Ph. D. thesis, Technical University of Munich, 2008.

[47] W. Zaremba, I. Sutskever, and O. Vinyals, „Recurrent neural network regularization." *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://dblp.uni−trier.de/db/journals/corr/corr1409.html#ZarembaSV14

[48] T. Liu, W. Zhou, and H. Li, „Sign language recognition with long short-term memory," in *Image Processing (ICIP), 2016 IEEE International Conference on.* IEEE, 2016, pp. 2871–2875.

[49] L. Pigou, A. Van Den Oord, S. Dieleman, M. Van Herreweghe, and J. Dambre, „Beyond temporal pooling: Recurrence and temporal convolutions for gesture recognition in video," *International Journal of Computer Vision*, vol. 126, no. 2-4, pp. 430–439, 2018.

[50] Y. Bengio, „Deep learning of representations for unsupervised and transfer learning," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, 2012, pp. 17–36.

[51] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, „Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 3630–3638.

Ábrák jegyzéke