EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR
NUMERIKUS ANALÍZIS TANSZÉK

EKG jelek osztályozása neurális hálózatokkal

Témavezető:
Bognár Gergő
egyetemi adjunktus

Készítette:
Milovszky Bence
programtervező informatikus MSc

Budapest, 2022.

Tartalomjegyzék

Ábrák jegyzéke

Táblázatok jegyzéke

1. Bevezetés

Az elektrokardiogram (EKG) jelek számítógépvezérelt feldolgozása évtizedek óta intenzív kutatási területe a jelfeldolgozásnak [5][6]. Az EKG-elemzések terén elért kutatási eredmények pozitívan járultak hozzá a különböző szívbetegségek időben történő felismeréséhez, a pontosabb diagnózis felállításához és az eltérő betegségek jobb kezeléséhez.

A szívritmuszavarok a szívet érintő rendellenes tevékenységre vagy viselkedésre utalnak [7]. Az aritmia bizonyos típusai életvesélyesek, amelyek szívmegállást vagy hirtelen halált okozhatnak. A dolgozatom során nem közvetlen halált okozó aritmiákat vizsgáltam, hanem olyanokat, amik hosszútávon komplikációkhoz vezethetnek. Az EKG jelek feldolgozása komoly humánerőforrást igényel a kardiológusok részéről. A szívütések részben automatikus értelmezése nagyban segíti az orvosi diagnózist.

A dolgozatomban három neurális hálózatot szeretnék bemutatni, melyek segítségével a különböző szívütéseket szeretném osztályozni. A feladat végén kiértékelem a modelleket a kapott eredmények alapján. Az első modellem tradícionális osztályozást hajt végre (azaz először jellemzőkinyerés történik, utána tanítás), a második modellem egy általános, míg a harmadik egy modell-alapú konvolúciós neurális hálózat [8]. Ezeket a modelleket a 4. fejezetben fogom bemutatni.

Az osztályozási folyamat során két fontos témakört érintek: a jelfeldolgozást és a mesterséges intelligenciát. Az orvosi jelfeldolgozás területén a releváns probléma jelentős szakirodalmi előzményekkel rendelkezik, amely alapján hatékonyan kezelhetőek a frekvencia- és időtérbeli dekompozíciók, példalául wavelet-alapú jellemzőkinyerés segítségével, majd gépi tanulás alkalmazásával.

A dolgozatom szerkezete a következő: a bevezetés utáni 2. fejezetben bemutatom a két nagy témakör azon fogalmait, amik kulcsfontosságúak a dolgozat megértéséhez. A 3. fejezetben bemutatom azt az adatbázist és annak feldolgozását, amely azokat a szívütéseket tartalmazza, amelyekkel a dolgozatomban foglalkoztam. Ezt a feldolgozási folyamatot implementáltam MATLAB-ban, amelyet az 5. fejezetben dokumentáltam. Ezek után a feldolgozott jelek osztályozását mutatom be (6. fejezet). Ezt követően pedig a tanítás során kapott eredményeket elemzem ki (7. fejezet).

2. Szakmai bevezetés

A diplomamunkámban digitális orvosi jelfeldolgozás és mesterséges intelligencia algoritmusokat kombinálok. Ebben a fejezetben rövid áttekintést adok a dolgozatom által érintett két nagyobb témakörről: a jelfeldolgozásról és a mesterséges neurális hálózatokról.

2.1. Jelfeldolgozás

Ebben az alfejezetben a jelfeldolgozás azon alkalmazásai kerülnek bemutatásra, amikkel én is foglalkoztam a diplomamunkám során.

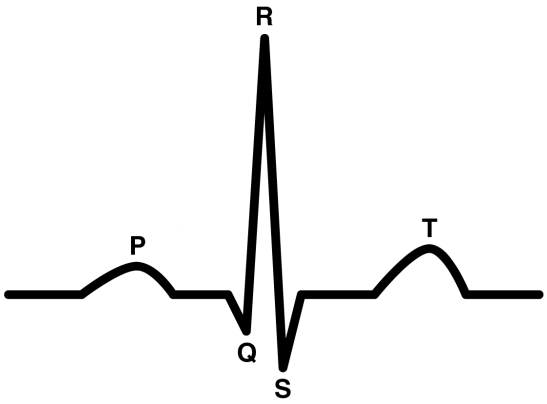
2.1.1. A szívütések morfológiája

A normális szívütéseket, mint digitális jeleket, morfológiai szempontból általánosan három fő részre tudjuk bontani [9] (1. ábra):

- A P-hullám (pitvari hullám) egy általában pozitív amplitúdójú hullám, az ingerület pitvari terjedésének felel meg
- A QRS-komplexum (kamrai hullám) a kamrák depolarizációját jelöli. Általában negatív Q-hullámból, magas pozitív R-hullámból és szintén negatív S-hullámból áll.
- A T-hullám egy közepes amplitúdójú hullám, amely a kamrák teljes repolarizációját jelzi.

A hullámformák morfológiailag eltérő viselkedéséből számos szívbetegségre lehet következtetni: például a magas QRS-komplexum segítségével ki lehet mutatni a szív izomzatának megvastagodását (bal kamra hipertrófiát), amely szövődmény megnöveli a szívinfarktus és a szívelégtelenség kockázatát. [10]

További információforrás lehet még a hullámok jellemzőinek szívütések közötti változása és az egyes szívütések távolsága is, de például a különböző hullámok közötti időintervallumok hosszából is különböző diagnózist lehet felállítani. Ezek, amiket megemlítettem, dinamikus jellemzők.

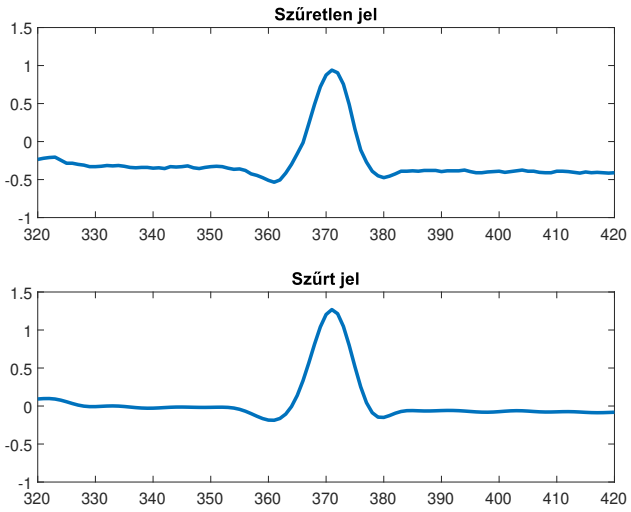


1. ábra. Egy normális szívütés morfológiája [1].

2.1.2. Szűrők használata

A jelfeldolgozás területén egy szűrőre tekinthetünk egy olyan eszközként, amelynek feladata az, hogy eltávolítsa a jelekről a nemkívánatos komponenseket vagy jellemzőket (például a felesleges zajokat). A szűrők legmeghatározóbb jellemzője a jel bizonyos jellegének vagy viselkedésének teljes vagy részleges elnyomása. Ez leggyakrabban bizonyos frekvenciák vagy frekvenciasávok eltávolítását jelenti és nem kizárólag csak a frekvenciatartományban működnek. A képfeldolgozás területén más célt szolgáló szűrőkkel is találkozhatunk. A szűrők széles körben elterjedtek, használják például az elektronikában, rádióban, televízióban, de alkalmazzák még radarokhoz és számítógépes grafikához is egyaránt. A 2. ábrán egy szűretlen és egy szűrt jel látható egymás alatt.

A szűrőket többféleképpen tudjuk kategorizálni. Az egyik ilyen kategóriába kerülnek azok a szűrők, amik típusuknak megfelelően, bizonyos frekvenciájú jeleket áteresztenek, míg bizonyos frekvenciájú jelet módosítanak, finomítanak. A diplomamunkám során lineáris, időben invariáns szűrőkkel dolgoztam, ezen belül is a jelfeldolgozásban szokásos alul- és felüláteresztő, valamint sávszűrőkkel [11].

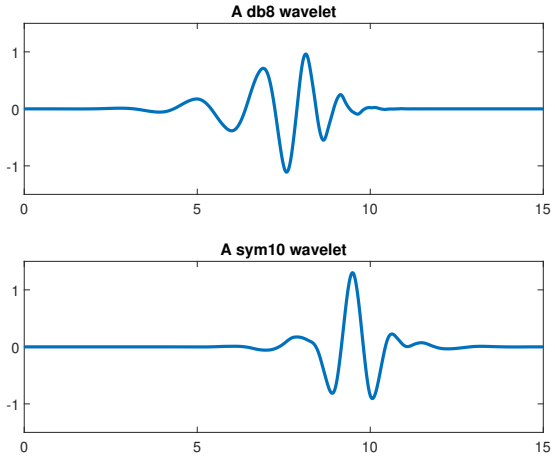


2. ábra. Egy szűretlen (felül) és egy szűrt (alul) jel.

2.1.3. Waveletek

A waveletek bemutatásához a [12] jegyzetet és a [13] könyvet használtam fel.

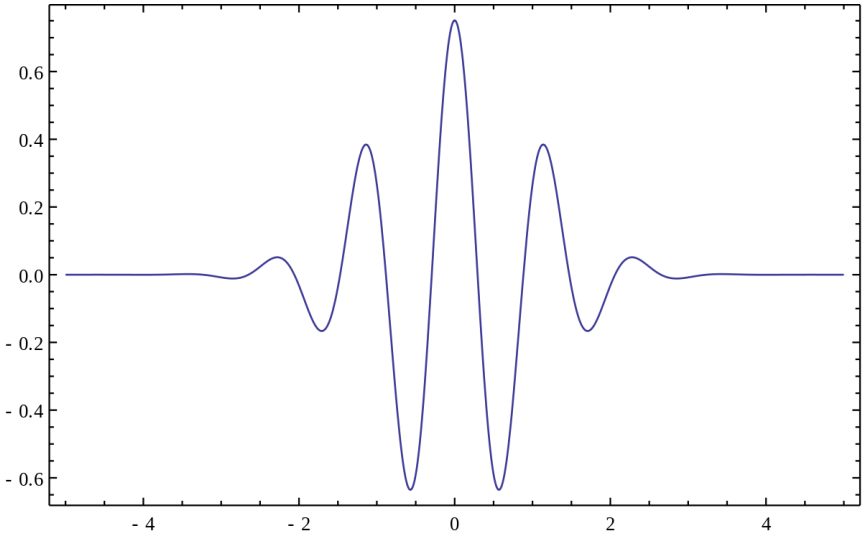
A wavelet transzformáció célja a jelek idő- és frekvenciatérbeli felbontása adott wavelet függvény segítségével. Lehetséges alkalmazásai a jel szűrése, feldolgozása és jellemzőinek kinyerése. Dolgozatomban az EKG jelek alapvonalvándorlásának szűrésére, szívütések jellemzőinek származtatására, valamint dimenziócsökkentésre alkalmaztam.



3. ábra. A diplomamunkámban felhasznált két wavelet: a db8 (felül) és a sym10 (alul).

A gyakorlatban a wavelet transzformációnak két változata ismert, a folytonos- és a diszkrét wavelet dekompozíció, továbbá számos wavelet és wavelet-család. A dolgozatomban a folytonos Morlet-waveletet (4. ábra), valamint a wavelet dekompozícióhoz a Daubechies 8 (továbbakban db8) és a Symlet 10 (továbbiakban sym10) wavelet függvényeket alkalmaztam. Utóbbiak wavelet függvényeit szemlélteti a a 3. ábra.

Folytonos wavelet A wavelet transzformáció folytonos változata a Fourier-transzformációhoz hasonló integráltranszformáció. A transzformációt a wavelet függvény dilatációi és transzlációi alapján végezzük. A gyakorlati alkalmazása során diszkrét konvolúciós szűrőkkel közelítjük.

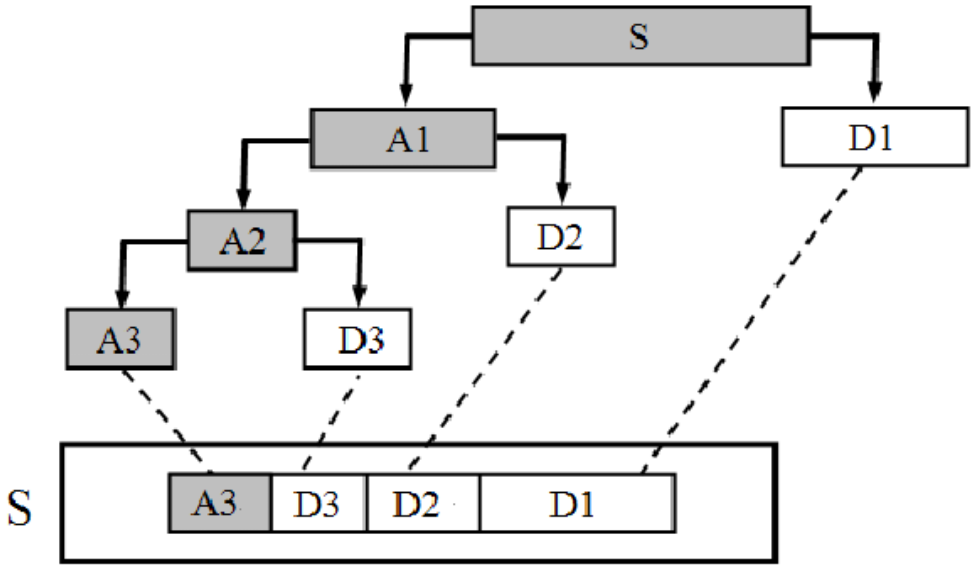


4. ábra. A Morlet-wavelet [2].

Wavelet dekompozíció A dolgozatomban csak a wavelet dekompozíció folyamatát szeretném szemlélteni és bemutatni, mivel a diplomamunka szempontjából fontos megérteni, miképp zajlik le egy ilyen folyamat.

A wavelet transzformáció diszkrét változata speciális tulajdonságú, ortonormált waveletek esetén alkalmazható. A dekompozíció ilyen esetekben elvégezhető egy alul- és felüláteresztő függvény alkalmazásával. Az aluláteresztős szűrő után approximációs együtthatókat (A_1), míg a felüláteresztős szűrő után részletegyütthatókat (D_1) kapunk. Ez a folyamat iteratív, az approximációs együtthatóra alkalmazzuk a szűrőket és kapunk magasabb szintű approximációs- és részletegyütthatókat, melyek

segítségével újabb szinteken tudjuk értelmezni a jeleket. Az 5. ábra a wavelet dekompozíció folyamatát szemlélteti.



5. ábra. A wavelet dekompozíció folyamata [3].

2.1.4. Főkomponens analízis

A főkomponens analízis (angolul: Principal Component Analysis, továbbiakban helyenként: PCA) a változók kombinációinak varianciáját vizsgálja, azaz megmutatja, milyen szorosan függ egymástól két változó.

Egy mérési adathalmazban korreláció, illetve lineáris kapcsolatok állhatnak fenn az egyes változók között. A PCA célja a változók helyettesítése lineáris független komponensekkel (főkomponensekkel). A transzformáció felhasználható az adathalmaz hatékonyabb reprezentációjára, illetve dimenziócsökkentésére, a nem független, illetve az alacsony varianciájú komponensek elhagyásával. Így sokszor jelentősen csökkenthető a változók száma, miközben az adathalmaz varianciáját a lehető legjobban megőriztük [14].

Matematikai szempontból a PCA egy lineáris transzformáció, melynek hátterében a szinguláris értékek felbontásán alapuló approximációs modell áll. A diplomamunkám során az általam elkészített feldolgozási folyamathoz a MATLAB beépített PCA függvényét használtam fel.

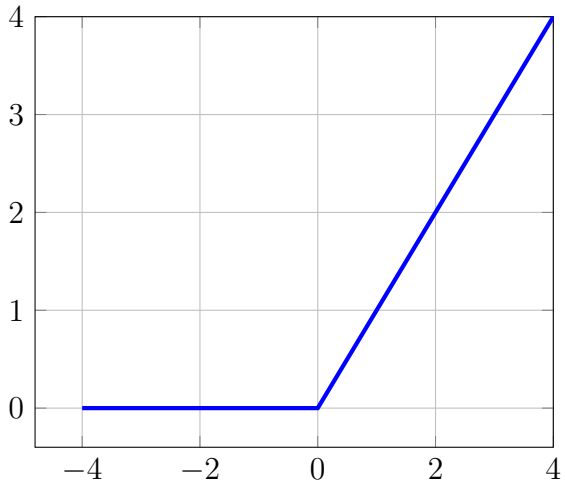
2.2. Mesterséges Neurális Hálózat

Az alfejezet megírásához a [15] és a [16] könyveket, a [8] cikket, valamint a [17] és a [18] honlapokat használtam fel. A diplomamunkámban jelentős szerepet játszik a mesterséges neurális hálózat, mivel ennek segítségével osztályoztam az EKG jeleket.

2.2.1. A mesterséges neuronok és rétegbe szerveződésük

A mesterséges neuronok a mesterséges neurális hálózat elemei. Szerkezetük nagyon hasonlít az emberi agyban található neuronokra, felépítésüket is ez inspirálja. Minden mesterséges neuronra tekinthetünk úgy is, mint matematikai függvényekre, amelyek a bemenet adott súlyok szerinti lineáris transzformációját végzik, majd egy nemlineáris aktivációs függvényt alkalmaznak.

Napjaink egyik legnépszerűbb aktivációs függvénye a ReLU, azért, mert pozitív bemenetekre a függvény lineáris, szigorúan monoton növekvő lesz, ezáltal nem csökken a tanulás hatékonysága. Egyrészt jobban kezeli az eltűnő gradiensek problémáját, mint egy tradícionális szigmoid függvény, másrészt nehezebb túltanítani, harmadrészt nagyon egyszerű implementálni és alkalmazni.



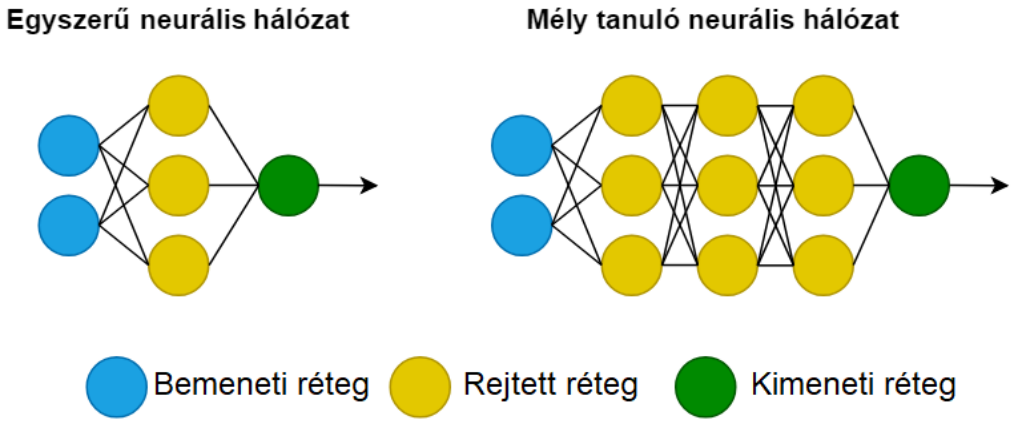
6. ábra. A ReLU aktivációs függvény grafikonja.

A mesterséges neuronok ritkán állnak egyedül, legtöbbször rétegekbe (7. ábra) szerveződve találkozunk velük. Három különböző típusú réteget különböztetünk meg egymástól:

- **Bemeneti réteg**: A bemeneti rétegben található neuronok megkapják a bemenetként átadott adatot, majd ezt módosítatlanul továbbküldik a következő rétegnek. Egy neurális hálózatnak több bemeneti rétege is lehet, ha több elágazást tartalmaz. A bemeneti rétegben található neuronok száma függ a bemeneti adat dimenziójától.

- **Rejtett rétegek**: A bemeneti és kimeneti réteg között elhelyezkedő rétegek. Ezek feladata az adatok feldolgozása és továbbküldése a következő rétegnek. Számuk és egymáshoz való kapcsolatuk különböző paraméterek megadásával változtatható.

- **Kimeneti réteg**: A kimeneti neuronok számát és a réteghez tartozó kimeneti függvényt jellemzően az adott probléma határozza meg. Egy osztályozási feladatnál például jellemzően annyi neuron található a kimeneti rétegben, ahány lehetséges kategória van. A réteghez tartozó kimeneti függvény azt határozza meg, hogy a bemeneti adat mekkora valószínűséggel tartozik egy adott kategóriához.



7. ábra. A mesterséges neuronok rétegekbe szerveződnek [4].

A mesterséges neuronokat gráfként is tudjuk értelmezni. Ha ez a gráfunk aciklikus, azaz egyik későbbi rétegben található neuron sem vezet vissza egy korábbi rétegben található neuronra, akkor előrecsatolt mesterséges neurális hálózatról beszélünk.

2.2.2. A mesterséges hálózatok tanítása

A mesterséges neurális háló tanítását jellemzően numerikus optimalizációs problémaként lehet megfogalmazni: a hálózatot úgy szeretnénk betanítani, hogy a neurális hálózat által meghatározott kimenet és az elvárt kimenet közötti eltérés minél kisebb legyen, azaz minél pontosabb legyen a tanítás. Ehhez veszünk egy függvényt, amely az eltérést vizsgálja. Ezt a függvényt nevezzük veszteségfüggvénynek. A tanítási folyamat során a célunk az, hogy a veszteségfüggvényt minimalizáljuk.

A tanítás általában sztochasztikus gradiens ereszkedő módszerrel történik. A teljes adathalmazra nehezen tudjuk kiszámolni a veszteségfüggvényt, mert a legtöbbször nagyon nagy bemenetet adunk át a mesterséges neurális hálózatnak, így ennek a memóriaigénye elég nagy. Éppen ezért kell valamilyen sztochasztikus módszert alkalmaznunk. Egyik lehetséges módszer az, hogy véletlenszerűen egyesével vesszük az adathalmaz pontjait, viszont a jelenlegi technológia miatt hatékonyabb, ha nem egy, hanem több adatpontot választunk ki véletlenszerűen egyszerre az adathalmazunkból (ezt jelöljük a batch size-zal), ezt a módszert nevezzük kötegelt sztochasztikus gradiens módszernek.

A sztochasztikus gradiens ereszkedő módszer a veszteségfüggvény súlyok szerinti gradiensét vizsgálja, majd módosítja a neurális hálózat súlyait. Ez a módszer nem feltétlenül garantálja, hogy globális minimumba jutunk, mert a veszteségfüggvény nem feltétlenül konvex, viszont a tapasztalat azt mutatja, hogyha eljutunk egy lokális minimumba, akkor már hatékonyan tudjuk alkalmazni a tanult függvényt.

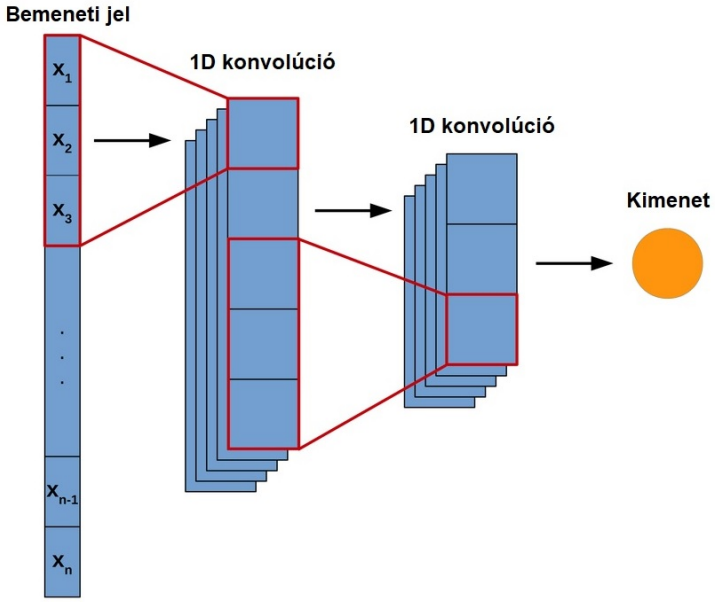
2.2.3. Konvolúciós neurális hálózat

A konvolúciós neurális hálózatok (angolul: Convolutional Neural Networks, továbbiakban helyenként: CNN) jelentős eredményeket a gépi látás területén érnek el.

A hálózatban konvolúciós, összevonó és teljesen összekötött rétegek találhatóak.

A hálózat első rétegei a **konvolúciós rétegek**, amelyek a bemenet jellemzőit emelik ki (8. ábra). Ezeket a rétegeket több szűrővel vizsgáljuk. Mivel dolgozatom során csak egydimenziós konvolúciós rétegekkel (a későbbiekben Conv1D) foglalkoztam

(amelyek jelek osztályozására tökéletesen alkalmazhatóak), így ebben az esetben a szűrők olyan F hosszú kernelek, melyek egy bizonyos S lépésköz mentén letapogatják a bemenetként szolgáló jelet és közben aktivációs függvényt alkalmaznak.



8. ábra. A konvolúciós rétegek működése.

Az eredmény egy aktivációs térkép, amely a szűrők által kinyert eredményeket tárolja. Ezeket az aktivációs térképeket további szűrőkkel vizsgálhatjuk addig, amíg meg nem kapjuk a kimenetet. Ez a kimenet lehet egy szám vagy egy vektor. Az osztályozási problémák esetén a kimeneti vektor speciális, mert a kimeneti valószínűségeket tartalmazza. Az értékei 0 és 1 közöttiek, és a vektor i-edik eleme a vizsgált bemenet i-edik osztályhoz tartozásának valószínűségét fejezi ki.

Mivel az aktivációs térkép mérete egyre kisebb lesz, ezért ha ezt el szeretnénk kerülni, akkor a bemenetünket ki kell párnázni, vagyis paddinget kell hozzáadni. Ebben az esetben a jel két végére pontosan annyi 0 kerül, amekkora paddinget választottunk.

De nem csak konvolúciós réteggel találkozhatunk egy konvolúciós neurális hálózatban, hanem összevonó (vagy angol nevükön pooling) rétegek és teljesen összekötött rétegek is lehetnek a hálózatban.

Az **összevonó rétegek** csökkentik a bemenet dimenzióját, amely így könnyebben kezelhető lesz. Két fajtája elterjedt: az, amelyik maximumot számol (angol nevén

max pooling) és az, amelyik átlagol (angol nevén average pooling). A modelleimben legtöbbször olyan összevonó réteget lehet találni, amelyek bizonyos méretű szűrő és egy adott lépésszám mellett átlagolja az adatokat. Például, ha a szűrő mérete 2 és a lépésköz is 2, akkor az összevonó réteg kettesével átlagol és kettesével is lépked, tehát a jel dimenzióját a felére csökkenti le, ahogy ez a 9. ábrán is látható.

9. ábra. Példa mindkét összevonó rétegre.

A dolgozatomban a modelljeim esetén alkalmaztam adaptív összevonó réteget, amelynek az a lényege, hogy a kernel méret és a lépésköz helyett a kimenet méretét adjuk meg.

A **teljesen összekötött rétegek** (2.2.1. rész) esetében a bemenet minden eleme össze van kötve a következő réteg összes neuronjával. Leginkább a konvolúciós neurális hálózatok utolsó rétegeiként alkalmazzák őket. Ezekből a rétegekből akármennyi lehet és a kimenetelt készítik elő.

2.2.4. WaveletKernelNet

Bár a konvolúciós neurális hálózatok bizonyították hatékonyságukat az orvosi diagnosztika területén, a hálózat szerkezetének fizikai jelentését ritkán tanulmányozták részletesen. Éppen ennek megmagyarázhatósága miatt hozták létre a WaveletKernelNetet [8], egy olyan modell-alapú neurális hálózatot, ami nagyon hasonlít a konvolúciós neurális hálózatra azzal a különbséggel, hogy a hálózat első rétege nem konvolúciós rétegből, hanem folytonos wavelet konvolúciós rétegből áll. Ez egy nagyon hatékony módszer különböző jelekbe ágyazott, hibás komponensek eltávolítására.

A konvolúciós neurális hálózat első rétege nagyon fontos, hiszen az ebből kinyert aktivációs térkép az egész modellre befolyással lehet. Normális esetben a konvolúciós neurális hálózat első rétege idő-alapú konvolúciókat hajt végre, viszont a jelben lévő komponensek nem nyerhetőek ki pontosan konvolúciós műveletekkel, nincs fizikai

jelentésük és nem is olvashatóak ki. Éppen ennek a problémának a kiküszöbölésére alkalmas a folytonos wavelet konvolúciós réteg.

A folytonos wavelet konvolúciós réteget több különböző, tetszőleges folytonosan deriválható wavelet segítségével is létre lehet hozni, ezek közül én a Morlet waveletes [19] változatát fogom alkalmazni az egyik modellem esetében, mivel a Morlet-waveletet már korábban is alkalmazták sikeresen EKG jelek osztályozására [20].

Különböző kutatási eredmények azt bizonyítják, hogy a WaveletKernelNet pontossága több, mint 10%-kal jobb, mint a konvolúciós neurális hálózatoké. Ezt a nagyobb különbséget az első, waveletes réteg használatával lehet indokolni. Továbbá elmondható az is, hogy a WaveletKernelNeteket sokkal könnyebb értelmezni, kevesebb paraméterrel rendelkeznek és képesek gyorsabban konvergálni ugyanazon tanulási cikluson (azaz epochon) belül.

3. Adatbázis feldolgozása

A dolgozatom során összesen három különböző neurális hálózatot szeretnék bemutatni és összehasonlítani a kapott eredmények alapján.

Az első modell működéséhez szükség van egy hosszabb feldolgozási folyamatra, amelyet ebben a fejezetben mutatok be lépésről-lépésre. A kezdetben hatalmas méretű adatbázisban található 44 felvétel a feldolgozás végére jelentősen kisebb méretű lesz. A másik két modellhez egy rövidebb feldolgozási folyamat tartozik (3.5. alfejezetig bezárólag).

Az adatbázis feldolgozásában nagy segítségemre volt a [21] és a [22] cikk, amely tökéletes iránymutatás volt számomra.

3.1. Adatbázis

A diplomamunkámhoz az *MIT-BIH Arrhythmia Database* [23][24] adatbázist használtam fel, amely az egyik legnépszerűbb és leggyakoribb aritmia-adatbázis, amelyet kutatásokhoz használnak fel. A Massachusetts Institute of Technology (MIT) és a bostoni Beth Israel Hospital (BIH) 1975 és 1979 között 47 páciensen összesen 48 különböző EKG felvételt készített. Ezek a felvételek egyenként körülbelül fél óra hosszúak, így összesen nagyjából 24 órányi felvétel áll rendelkezésre.

Az adatbázis körülbelül 110 ezer egyedi szívütést tartalmaz, minden szívütést elláttak 16 különböző annotáció egyikével, valamint szívritmus zavarra utaló jelzésekkel. Az annotációkat több különböző kardiológus egymástól függetlenül végezte el, így alakult az MIT és BIH közös, ma is használatos aritmia adatbázisa.

A diplomamunkám célja az, hogy különböző megközelítésű neurális hálókkal végzett osztályozásokat összehasonlítok. A 48 felvétel közül néggyel nem foglalkoztam: ezek olyan betegek felvételei, akiknek pacemakerük volt, amik olyan szívütéseket generáltak, amik rontják a tanulás hatékonyságát.

Megjegyzés. Az adatbázis összes felvételéről további információkat a [25] honlapon lehet találni, ahol többek között a felvételeken található szívütések különböző annotációinak darabszámáról, valamint a mérések körülményeiről is informálódhatunk.

3.2. A teljes adatbázis szétválasztása

Mivel a mesterséges neurális hálózat tanításához szükségünk van tanító- és teszthalmazra, éppen ezért a feldolgozás elején több korábbi tudományos cikkben leírtakhoz hasonlóan ([21][26]) én is kétfelé, egyenlő arányban osztottam el a felvételeket az alábbiak szerint:

- A 101, 106, 108, 109, 112, 114, 115, 116, 118, 119, 122, 124, 201, 203, 205, 207, 208, 209, 215, 220, 223, 230 felvételeken lévő szívütések alkotják a tanítóhalmazt.
- A 100, 103, 105, 111, 113, 117, 121, 123, 200, 202, 210, 212, 213, 214, 219, 221, 222, 228, 231, 232, 233, 234 felvételeken található szívütések pedig a teszthalmazt alkotják.

Megjegyzés. Ahogy a 3.1. részben is megemlítettem, a 102-es, a 104-es, a 107-es és a 217-es felvételek szívütéseit nem vizsgálom a diplomamunkámban, mivel ezen felvételek pacemakeres betegektől származnak.

A diplomamunkámban az egyszerűségre és az újrafelhasználhatóságra törekedve a két halmazt külön-külön fogom feldolgozni.

Lehetőség van arra is, hogy esetleg máshogy rendeljük hozzá a felvételeket a tanító- és a teszthalmazhoz. A diplomamunkámhoz készített MATLAB kódhoz készítettem két szöveges dokumentumot, amik a különböző halmazokhoz rendelt felvételek sorszámait tartalmazza, amiket tetszőlegesen lehet módosítani. Ezeket később, az 5. fejezetben, a MATLAB programok bemutatásakor még megemlítem.

3.3. Annotációk számának csökkentése

Több korábbi kutatás során két különböző kiértékelési módszert használtak [26].

Az egyik az osztály-orientált kiértékelés, melynek a lényege az, hogy a tanító- és teszthalmazok csak a szívütések osztályai alapján hozzuk létre, nem vesszük figyelembe, hogy az egyes szívütések melyik páciensehez tartoznak. Az irodalmi gyakorlat

alapján ez esetben mind a 16 annotációra osztályozunk. Ez a módszer sokkal optimistább, pontosabb osztályozást eredményez.

A másik kiértékelési módszer esetén a tizenhat helyett csak tizenöt annotációt veszünk figyelembe, mivel az egyik annotációhoz azokat a szívütéseket rendeljük hozzá, amelyek a pacemakeres betegekhez tartoznak (és mivel azokkal a felvételekkel nem foglalkozunk, amelyben ilyen szívütések találhatóak, így a továbbiakban elhagyhatjuk). Ez a módszer páciens-orientált, amelynek az a lényege, hogy a tanító- és a teszthalmazt páciensenként alakítjuk ki, vagyis adott páciens összes szívütése az egyik halmazba kerül. Mivel ez jóval nehezebb probléma, így öt nagyobb osztályra végezzük csak az osztályozást [21]. Ez a kiértékelési módszer kevésbé pontos, mint az osztály-orientált kiértékelés, ellenben sokkal realisztikusabb becsléseket kapunk a különböző annotációk általánosítására.

A diplomamunkámban én is az utóbbi, azaz a páciens-orientált kiértékelési módszert fogom vizsgálni. Az 1. táblázat azt mutatja meg, hogy a tizenöt annotáció összevonása után az öt különböző osztályhoz összesen hány (a tanító- és a teszthalmazt külön figyelembe véve) szívütés tartozik.

Osztály	**Tanító**	**Teszt**	**Össz.**
1	45.842	44.243	90.095
2	1.000	1.972	2.972
3	4.260	3.220	7.480
4	414	388	802
5	8	7	15
Összesen:	51.524	49.830	101.364

1. táblázat. A különböző osztályokhoz tartozó szívütések darabszámai.

3.4. Szűrők használata

Az EKG jelek kiértékelések egyik gyakori problémája az, hogy ha a QRS komplexek, az R és a P hullámok nem azonos szintről indulnak. Ezt úgy nevezzük, hogy az EKG alapvonala a mérés közben vándorol. Ez viszonylag lényeges probléma, hiszen

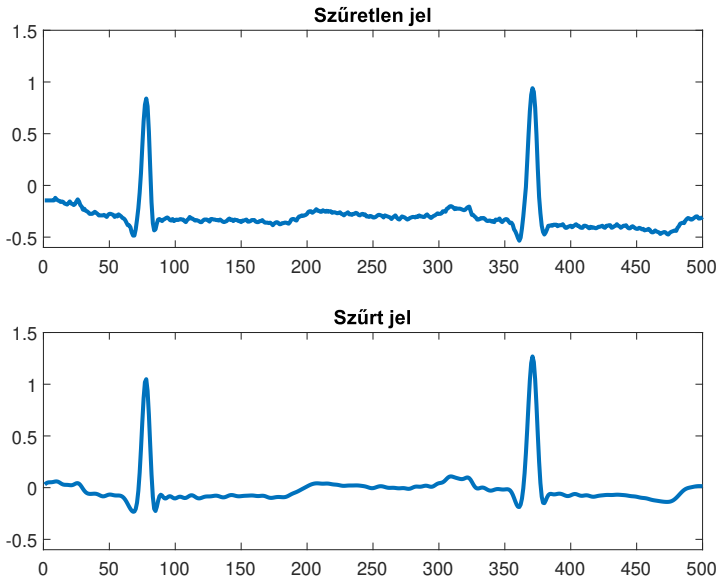
ez megnehezíti a csúcsok egymáshoz képesti értékelését, valamit a szívütések morfológiai értelmezését is. Éppen ezért minden EKG mérés lényeges eleme az alapvonal korrekció, amit a dolgozatom során én is alkalmaztam [27].

A [21] cikkhez hasonlóan én is alkalmaztam egy waveletes alapvonal korrekciót, méghozzá a sym10 nevű wavelet függvény felhasználásával.

Ami viszont az említett cikkhez képest eltérés, hogy az én feldolgozási módszerem során nem sáváteresztős szűrőt alkalmaztam, hanem aluláteresztőset, ahol a határt 35 Hz-en határoztam meg. Ennek oka az, hogy a [22] cikk alapján a szívütések releváns információi ezen frekvenciatartományban találhatóak meg.

A 10. ábrán a 100-as felvétel egy részlete látható szűrés előtt és után.

Megjegyzés. A [21] cikkben található 5-15 Hz-es sáváteresztős szűrőt inkább a QRS komponens detektálására fejlesztették ki, amely főként a QRS-t tartja meg, de a többi hullám releváns részeit eltávolíthatja.

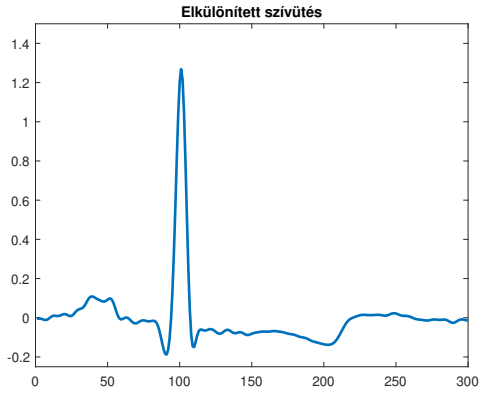


10. ábra. A 100-as felvétel egy részlete szűrés előtt és után.

3.5. Szívütések elkülönítése

Mivel szívütésenként szeretnék osztályozni, ezért el kell őket különíteni. A [21] alapján fix ablakmérettel szeretnék dolgozni, mivel ezek illeszkednek a neurális há-

lókhoz is. Az adatbázis összes szívütését egy 300 széles „ablakba” különítettem el az alábbiak szerint: megnéztem, hogy hol található a szívütések R csúcsai és az R csúcs előtti 100, illetve az R csúcs utáni 200 mérési pont alkotja együtt az adott szívütéshez tartozó szegmenst. Egy ilyen elkülönített szívütés látható a 11. ábrán.



11. ábra. A 100-as felvétel egyik szívütése „elkülönítve”.

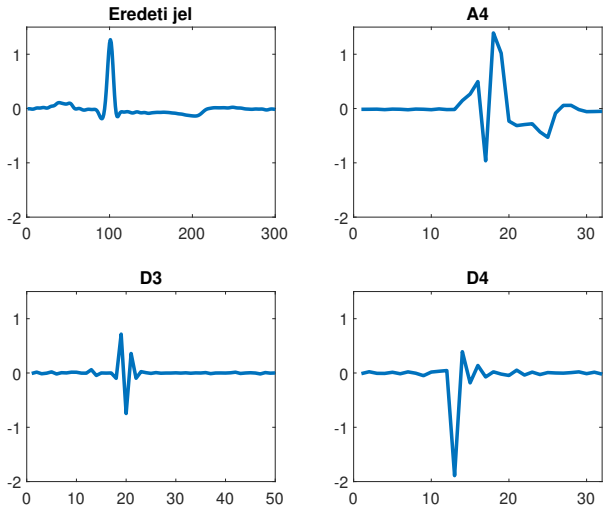
A 300 mintavételi pontból álló szegmens kimérése egy általános módszernek tekinthető, amelyet származtathatunk a P-R távolság és a Q-T távolság arányából is. Bizonyos esetekben, amikor valakinek gyorsabban ver a szíve, a két szívütés közötti távolság rövidebb, így ha megnöveljük a szegmens méretét, akkor előfordulhat az, hogy egy ilyen szegmensben megtalálhatóak a szomszédos szívütésekhez tartozó információk is, amelyek a hamis riasztások megnövekedéséhez vezethetnek.

3.6. Dimenziócsökkentés wavelet dekompozícióval

A wavelet dekompozíció során a db8 nevű waveletet alkalmaztam, mivel ez elég jól hasonlít a legtöbb QRS hullámhoz [21]. Egy korábbi kutatás [28] során bebizonyították azt, hogy az EKG jelek a 0.5-40 Hz-es frekvenciasávban mozognak. Ha alkalmazunk egy négyszintes wavelet dekompozíciót, akkor azt tapasztaljuk, hogy ebben a frekvenciatartományban a 3. (D3) és a 4. (D4) szintű részletegyüttható és a 4. (A4) szintű approximációs együttható található meg. Mivel a D4 és az A4 együtthatók 32 hosszúak, a D3 együttható pedig 50 hosszú, így ha ezt a hármat összerakjuk, akkor 114 wavelet jellemzőt nyertünk ki egy szívütésből.

Megjegyzés. Azokkal a szívütésekkel, amelyeknél nem lehetséges a megfelelő szeg-

mens kimérése, a későbbiekben nem fogok foglalkozni, mert a Wavelet dekompozíció során rengeteg 0-ás érték keletkezik, amelyek befolyásolhatják a tanítás hatékonyságát. Ezek a szívütések nem találhatóak meg az 1. táblázatban!



12. ábra. A 11. ábrán látható szívütés dekompozíciója.

3.7. Dimenziócsökkentés főkomponens-analízissel

Jelen pillanatban 114 wavelet jellemző tartozik minden egyes szívütéshez, amire hivatkozhatunk morfológiai jellemzőkként is, mivel ezek a szívütések hullámhosszainak alakját jellemzik. Erre a 114 morfológiai jellemzőre alkalmazzuk a főkomponens-analízist. A [21] cikkben részletezett eredményeket felhasználva én is 18 főkomponenst választok ki, amelyek az alapján lettek kiválasztva, hogy ez a 18 főkomponens a tanítóhalmaz varianciájának körülbelül 90%-ának felel meg. A feldolgozási folyamat végére minden szívütés dimenzióját jelentősen, 300-ról 18-ra le tudtam csökkenteni.

3.8. Összefoglalás

Röviden összefoglalva a feldolgozási folyamatot:

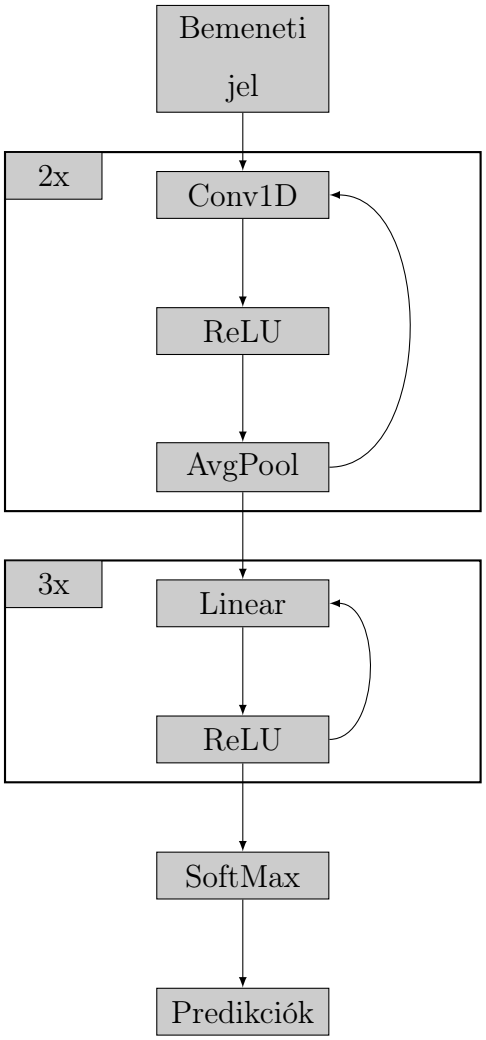
- Az *MIT-BIH Arrhytmia Database* nevű adatbázis 44 felvételével foglalkoztam a dolgozatom készítése során, melyeket két halmazba (tanító- és teszthalmaz), egyenlő arányban osztottam szét.

- A páciens-orientált kiértékelési módszert követve a 15, számunkra lényeges annotációt 5 nagyobb osztályhoz rendeltem hozzá.
- Két különböző szűrés segítségével megszűrtem az összes EKG jelet.
- A felvételeken található szívütéseket 300 mintavételi pontból álló szegmensekre bontottam fel.
- Wavelet dekompozíció alkalmazásával 114 wavelet jellemzőt származtattam.
- Majd a főkomponens analízis segítségével, felhasználva azt, hogy minden szívütéshez 114 wavelet jellemző tartozik, a dimenziót lecsökkentettem 18-ra.

4. Neurális hálózat modellek

Ebben a fejezetben az osztályozáshoz általam készített három neurális hálózat architektúrát mutatom be: a CNN18-t, a CNN300-t és a WKN300-t. Ezekre a későbbiekben különböző modellként fogok hivatkozni.

4.1. CNN18



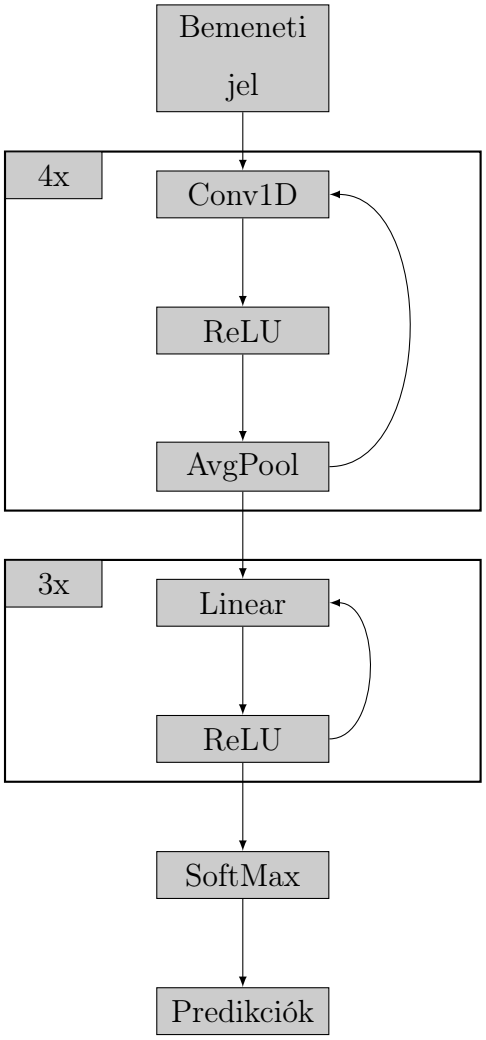
13. ábra. A CNN18 modell.

Az első modellem a CNN18 (13. ábra) nevet kapta, ugyanis a konvolúciós neurális hálózat a teljesen feldolgozott adatbázist kapja meg bemenetként. A CNN18 modell első konvolúciós rétege 1 bemeneti csatornából 4-et készít, a kernel mérete 3 és az alacsonyabb dimenziószám miatt alkalmaztam itt egy 5-ös paddinget. Ezután

alkalmaztam a ReLU aktivációs függvényt, majd egy average poolingot is, aminek a mérete és a lépésköze is 2. A következő konvolúciós réteg 4 bemeneti csatornából csinál 16-ot, a kernel mérete szintén 3. Erre a rétegre is alkalmaztam a ReLU-t és egy adaptív average pooling réteget, aminek segítségével lecsökkentettem a jelek dimenzióját 4-re.

Ezt követően három teljesen összekötött réteget alkalmaztam: az első réteg 64, a második réteg 32, a harmadik réteg pedig 16 neuronból áll. A harmadik réteg össze van kötve a kimeneti réteggel, ahol 5 neuron található, mivel 5 osztályunk van. A rétegek között is alkalmaztam a ReLU-t.

4.2. CNN300



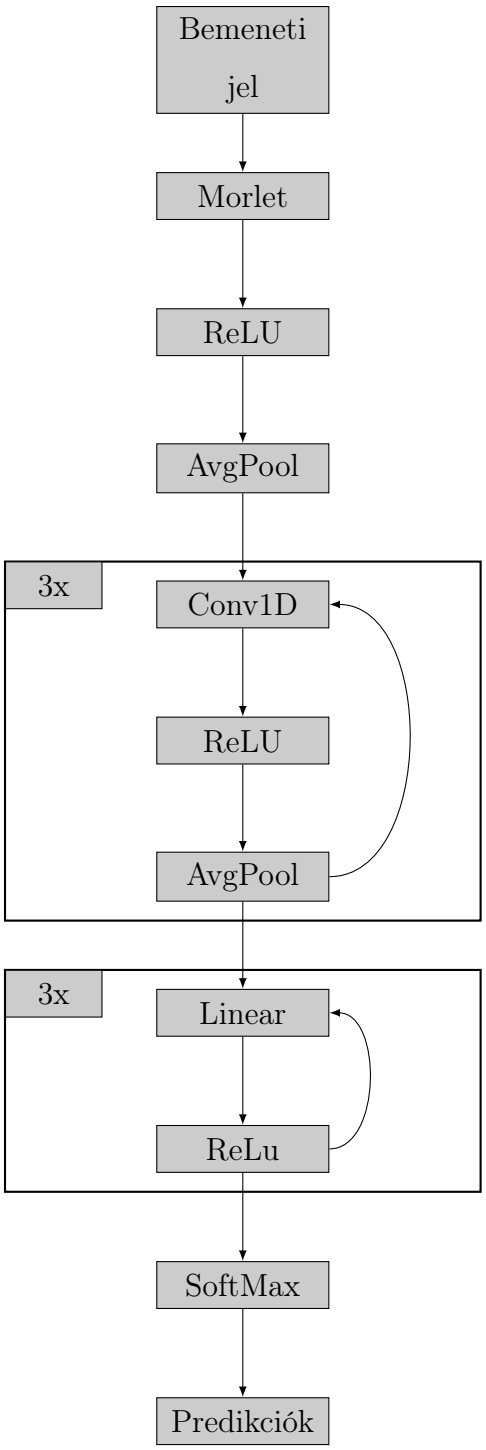
14. ábra. A CNN300 modell.

A második modellem a CNN300 (14. ábra) nevet kapta, mivel a konvolúciós neurális háló a 300-as méretű szívütés szegmenseket kapja meg bemenetként. Ez a modell összesen négy konvolúciós rétegből áll: az első réteg 1 bemeneti és 4 kimeneti csatornából áll és a kernel mérete 5, a második réteg 4 bemeneti és 16 kimeneti csatornából áll és a kernel mérete szintén 5, a harmadik réteg 16 bemeneti és 64 kimeneti csatornából áll és a kernel mérete 3, valamit a negyedik réteg 64 bemeneti és 32 kimeneti csatornából áll és a kernel mérete szintén 3. Minden réteg után alkalmaztam a ReLU aktivációs függvényt, valamint az első és a második réteg után 3-as méretű és 2-es lépésközű average pooling-ot, valamint a negyedik réteg után adaptív average pooling réteget használtam, hogy a dimenziót lecsökkentsem 18-ra.

Ezután, hasonlóan az előző modellhez, három teljesen összekötött réteget használtam fel a neurális hálózathoz. Az első rétegben 576, a második rétegben 256, valamint a harmadik rétegben 16 neuron található. Az utolsó réteg teljesen össze van kötve a kimeneti réteggel, ami 5 neuronból áll. A rétegek között alkalmaztam a ReLU aktivációs függvényt.

4.3. WKN300

A harmadik modellem a WKN300 (15. ábra) nevet viseli, mivel a neurális hálózat első rétege egy folytonos wavelet konvolúciós rétegből áll. Ez a modell szinte megegyezik a CNN300 felépítéséhez annyi különbséggel, hogy az első konvolúciós réteg ki van cserélve egy Morlet-waveletet tartalmazó réteggel, amely 4 kimeneti csatornát készít és a kernel mérete 5. Ezután alkalmaztam a ReLU-t és innentől kezdve a modell felépítése teljesen megegyzik a CNN300-ével.



15. ábra. A WKN300 modell.

5. Jelfeldolgozás MATLAB-ban

Ebben a fejezetben részletesen bemutatom mindhárom MATLAB programnyelvben íródott függvényemet, aminek segítségével a 3. fejezetben részletesen kifejtett módon lecsökkentettem az adatbázisban talált szívütések dimenzióit.

5.1. Előkészületek

A MATLAB programkódok futtatásához szükségünk van egy fordítóprogramra. Az egyik legnépszerűbb fordítóprogram a MathWorks amerikai cég MATLAB nevű programja. A diplomamunkám írásakor az R2021a verziószámú programot használtam.

A MATLAB programon kívül szükségünk van négy különböző toolboxra, amiből hármat fel kell telepítenünk ahhoz, hogy tudjuk futtatni a programokat:

- A *felvételek beolvasásához* szükségünk van a **WFDB** nevű toolboxra. Ezt a toolboxot külön le kell tölteni [29]. A diplomamunkámhoz mellékelt MATLAB nevű mappában külön megtalálható a teljes toolbox a WFDB mappában, így külön nem szükséges letölteni ahhoz, hogy futtatni tudjuk az általam megírt programkódokat.
- A **Wavelet** nevű toolboxra a *Wavelet transzformációkhoz és dekompozícióhoz* van szükség.
- A **Statistics and Machine Learning** nevű toolboxra a *főkomponens analízis* miatt van szükség.
- A **Signal Processing** nevű toolboxra pedig a *szűrők használatára* van szükség.

Mielőtt részletesen bemutatnám a programjaimat, be szeretném mutatni az egyes mappákat, amik a MATLAB könyvtáron belül találhatóak, olyan sorrendben, ahogy a következő alfejezetben megemlítem őket.

A *felvételek* mappa két szöveges dokumentumot tartalmaz. Mindkét szöveges dokumentum azon felvételek sorszámait tartalmazza, amelynek feldolgozott szívütései

vagy a tanító- vagy a teszthalmazba kerülnek a 3.2. alfejezetben tárgyalt módszer szerint. Ezeket a szöveges dokumentumokat tetszőlegesen lehet módosítani: el is hagyhatunk felvételeket, vagy áthelyezhetünk bizonyos felvételeket az egyik fájlból a másikba - mindkét esetben tökéletesen, fennakadás nélkül végrehajtódnak a programok.

A *mitdb* mappa az MIT-BIH aritmia adatbázis (3.1. alfejezet) felvételeit tartalmazza. A mappában megtalálható a négy, általam fel nem használt felvétel is. Minden felvételhez három különböző fájl tartozik - a diplomamunkámban én csak az .atr kiterjesztésű fájlokkal dolgoztam, mert ezek a fájlok tartalmazzák a szívütések pontos helyeit és az annotációkat.

Az *adatok* mappa a tanító- és teszthalmaz teljesen feldolgozott adathalmazát tartalmazza, különböző szöveges dokumentumokban tárolva. Jelenleg található benne négy szöveges dokumentum, amelyek egy újbóli futás során felülíródnak, így tetszőlegesen tudjuk futtatni a függvényeket anélkül, hogy ezzel kapcsolatos előkészületeket kellene eszközölnünk.

5.2. MATLAB programok

5.2.1. feldolgoz.m

A MATLAB-ban történő jelfeldolgozás első részét a **feldolgoz** nevű függvény végzi el. Ez az eljárás két paramétert vár:

- txt - egy szöveges dokumentum, amely azon felvételek számait tartalmazza, amiket külön szeretnénk vizsgálni
- to - egy olyan szöveges dokumentum, amelybe kiírjuk a feldolgozás utáni (nyers) adathalmazt

WFDB Toolbox A kód elején található **addpath('WFDB')** függvény szükséges ahhoz, hogy tudjuk használni a későbbiekben a WFDB toolboxot.

Fájl megnyitása és felülírása A **fopen** függvény gondoskodik arról, hogy minden egyes futásnál felülírjuk (vagy ha még nem létezett, akkor létrehozzuk) a *to* paraméterben megadott szöveges dokumentumot. Erre azért van szükség, hogy többszöri futás esetén ne fordulhasson elő az, hogy egy felvétel feldolgozott szívütései többször szerepeljenek. A **dlmread(txt)** függvény segítségével be tudjuk olvasni a felvételek számait tartalmazó dokumentumokat, amiket egy **records** nevű változóban mentettem el, amit a MATLAB oszlopvektorként fog a továbbiakban értelmezni.

Adatok beolvasása A következő for ciklus pontosan annyiszor fog lefutni, ahány felvételt tartalmaz a **records**. A for ciklus első része az adatok beolvasását szolgálja. **Dir** változó néven el van mentve annak a mappának az elérési útvonala, ahol megtalálhatóak a felvételek. A program először átkonvertálja az adott felvétel sorszámát stringgé, majd a mappával összefűzve felhasználja az elérési utat a felvételek beolvasásához.

Még két függvény található meg ebben a részben, ezek futtatásához van szükségünk a WFDB toolboxra:

- Az **rdsamp()** segítségével az aktuális felvételt tudjuk beolvasni. A diplomamunkám szempontjából két fontos visszatérési értéke van:
  - A **signal** visszaad egy NxM-es mátrixot, ahol az M a felvételek száma, N pedig a különböző jelek hossza. Ebben a mátrixban tároljuk a felvételek különböző mintavételezési pontjaiban lévő értékeit.
  - Az **fs** egy M hosszú sorvektort ad vissza, ahol M a felvételek száma. Ez azt mutatja meg, hogy hány Hz-en történt a felvételen található összes jel mintavételezése.
  - Mivel egyszerre csak egy felvétel beolvasása történik, ezért az M értéke 1 lesz.
- Az **rdann()** az adott felvételhez tartozó annotációkat olvassa be. A diplomamunkámban két visszatérési értékével foglalkozom:
  - Az **ann** egy N hosszú oszlopvektort ad vissza, amely az összes szívütés pontos helyét mutatja meg.

– Az **anntype** szintén egy N hosszú oszlopvektort ad vissza, amely az összes szívütéshez tartozó annotációt tartalmazza.

Annotációk számának csökkentése Az annotációk helyei és a típusai egy $N \times 2$-es mátrixban találhatóak meg (ahol N az annotációk számait jelöli). Az annotációk típusai karakterekkel vannak jelölve az .atr kiterjesztésű fájlokban, viszont ha az annotáció típusokat egy másik mátrixba másoljuk át, akkor a karakterek helyett a nekik megfelelő ASCII-kódok kerülnek be a mátrixba (például az 'N' helyett 78), éppen ezért az egyenlőségvizsgálatot csak az ASCII-kódok alapján tudjuk elvégezni. Az annotációkat és a hozzájuk tartozó ASCII-kódokat a 2. táblázat tartalmazza [30].

Osztály	**Annotáció**	**ASCII-kód**	**Osztály**	**Annotáció**	**ASCII-kód**
1	N	78	3	V	86
	L	76		E	69
	R	82		!	33
	e	101	4	F	70
	j	106	5	f	102
2	A	65		Q	81
	a	97			
	x	120			
	J	74			

2. táblázat. A különböző annotációk és ASCII-kódjuk.

Megjegyzés. Mind a tizenhat annotáció jelölése és az öt nagyobb osztályba történő besorolása megtalálható a [21] cikkben. Az annotációk pontos jelentései megtalálhatóak a [31] honlapon.

Mivel az .atr fájl tartalmaz olyan annotációkat, amik nem szívütéseket, hanem szívritmusváltozásokat annotálnak [31], illetve a korábban többször említett / annotációval sem foglalkoztam a diplomamunkámban, így ezeket az annotációkat ki kellett szűrnöm. Erre szolgál a **datas(i,:) = 0** kód, amivel az ilyen annotációnak a helyét és típusát is nullára írtam át. Néhány sorral lejjebb ezeket a nullás sorokat egyszerűen elhagyom, így a továbbiakban tényleg csak azokkal a szívütésekkel

foglalkoztam, amiket valamelyik osztályhoz hozzárendeltem.

Szűrők használata Az alapvonal korrekcióhoz szükségünk van wavelet dekompozícióra és egy olyan rekonstrukcióra, amely az approximációs együtthatók segítségével megbecsüli az alapvonalat.

Az **ekg_jel** nevű változóba kimentettem a **signal** mátrix első oszlopát. A **level** változóban tároljuk azt, hogy hányas szintű wavelet dekompozíciót szeretnénk végrehajtani. A legegyszerűbben úgy tudjuk ezt az értéket meghatározni, hogyha vesszük az **fs** kettes alapú logaritmusát, majd ennek a felső egészrészét és kivonunk belőle egyet. A **wavedec()** beépített függvényt alkalmaztam a wavelet dekompozícióhoz, amely három paramétert vár:

- a bemeneti jelet
- a dekompozíció szintjét
- egy wavelet függvényt

A függvény kimenete pedig két vektor (**C** és **L**) lesz. A **wrcoef()** beépített függvény segítségével az approximációs vagy részletegyütthatókat tudjuk rekonstruálni. Ez a függvény 5 paramétert vár:

- 'a'-t vagy 'd'-t, attól függően, hogy melyik együtthatót szeretnénk rekonstruálni
- a **wavedec()** függvény alkalmazásával létrehozott **C** és **L** vektorokat
- egy wavelet függvényt
- a dekompozíció szintjét

A programomban a **wrcoef()** eredményét a **baseline** változó tárolja, mivel a függvény az approximációs együtthatók segítségével megbecsülte az alapvonalat. Ezután egyszerűen kivontam az **ekg_jel**-ből az alapvonalat és eltároltam az **ekg_jel2** nevű változóban.

A 3.4. részben már említést tettem arról, hogy az alapvonal vándorlás korrekció mellett alkalmaztam egy aluláteresztős szűrőt, méghozzá 35 Hz-en. Ehhez a **lowpass()** függvényt hívtam meg, amely három paramétert vár bemenetként:

- a jelet, amit meg szeretnénk szűrni
- a határértéket, ameddig áteresztjük a jelet
- a frekvenciát, ahol mintavételeztük a jelet

A szűrt jelet eltároltam az **ekg_jel3** nevű változóban. A következő részben ezzel a (szűrt) jellel dolgozom tovább.

Wavelet dekompozíció A **feldolgoz** függvényem utolsó része egy for ciklusból áll: annyiszor hajtódik végre a ciklusmag, ahány olyan szívütésünk van, amelyet az öt osztály valamelyikébe besoroltam.

A feldolgozott jeleket egy **fjel** nevű oszlopvektorba tárolom, amely 300 darab nullát tartalmaz és amelyet minden iteráció elején lenulláz a program. Ezután következik egy elágazás, amely három feltételből áll, mivel a szegmensek kimérésekor háromféle eset állhat fel:

- A szívütés közelebb van a jel kezdetéhez, mint 100 mintavételi pont
- A szívütés közelebb van a jel végéhez, mint 200 mintavételi pont
- A szívütés kellő távolságban van a jel kezdetétől és végétől

Az elágazás mindhárom ága ugyanazokat a műveleteket tartalmazza, így csak egyszer részletezem.

Minden ág tartalmaz egy for ciklust, amelynek segítségével az **ekg_jel3** szűrt jelet tartalmazó vektor megfelelő mintavételezési pontjait eltárolja a program az **fjel** vektorban. Ezután ezen az **fjel**-en alkalmaztam egy négyszintes wavelet dekompozíciót, amire a db8 nevű wavelet függvényt alkalmaztam.

A következő két függvény segítségével a szükséges együtthatókat nyeri ki a program a szegmensből. A 3.6. alfejezetben már említett két részletegyüttható kinyeréséhez a **detcoef()** nevű függvényt használtam fel. Ez a függvény paraméterként a **C**

és az **L** vektorokat várja, amiket az előző **wavedec()** függvény alkalmazásával kaptam meg, továbbá paraméterként meg kell adnunk egy vektorként az összes szintet, amelynek részletegyütthatóira szükség van.

Az approximációs együtthatót az **appcoef()** nevű függvény alkalmazásával kapja meg a program. Ennek a függvénynek is meg kell adni paraméterként a **C** és az **L** vektorokat, viszont annyi különbség van az előző függvényhez képest, hogy meg kell adni egy wavelet függvényt és azt a szintet, amelyik szintű approximációs együtthatóra szükség van.

Ha az EKG jel valamelyik végéhez közelebb található a szívütés, akkor a megfelelő szegmens kimérésének hiánya miatt előfordul, hogy a wavelet dekompozíció során számos 0 érték keletkezik, amely nagyban befolyásolni tudja a tanítás hatékonyságát. Éppen ezeket a szívütéseket hivatott kiszűrni az utolsó elágazásom. Végezetül azokat a szívütéseket, amelyek a wavelet dekompozíció után legfeljebb 2 darab 0-s értéket tartalmaznak, a program bemásolja egy **x** nevű vektorba, amit a futás végén a *to* paraméterként megadott szöveges dokumentumba ment el.

5.2.2. fokomponens.m

A jelfeldolgozás folyamat második részét a **fokomponens** nevű függvény végzi el. Ebben az eljárásban a **feldolgoz** függvény végén kapott két nyers szöveges dokumentum kerül formázásra, majd a főkomponens analízis alkalmazása után megkapjuk a végleges tanító- és teszthalmazunkat, amiket a 6. fejezetben részletezett tanulási folyamathoz fel tudunk használni.

Ez a függvény egy paramétert vár:

- txt - kezdetben a a **feldolgoz** függvény futása végén kapott, az adatokat ömlesztve tároló, szöveges dokumentum, amelybe kiírjuk a feldolgozás utáni, már rendezett adathalmazt (ez a dokumentum szintén a *adatok* mappába kerül)

Adatok formázása Mivel a **feldolgoz** függvény futása során a különböző értékek rendezetlenül, egy darab oszlopvektorként kerültek be a szöveges dokumentumba, így első lépésként mindenképp mátrixba kell rendezni az adatokat, máskülönben nem

lehet alkalmazni a főkomponens analízist.

Először a **dlmread()** nevű függvény segítségével a program beolvassa a „nyers" adathalmazt. Mivel minden szívütés dimenzióját lecsökkentettük az előző függvény segítségével 114-re, továbbá eltároltuk a szívütésekhez tartozó annotációkat is, így jelenleg minden szívütéshez 115 adat tartozik. A MATLAB az adatokat nem sorfolytonosan, hanem oszlopfolytonosan tölti fel az újraméretezett mátrixba, így egy $115 \times n$-ös mátrixot kellett készítenem, ahol n a szívütések darabszáma. Mivel a vektor hossza osztható 115-tel, így létre is tudja hozni a függvény azt a mátrixot, amiben el tudjuk tárolni az értékeket.

Jelen pillanatban hiába tároljuk egy olyan mátrixban a részben feldolgozott adatbázist, ahol minden oszlop egy szívütéshez tartozik, nem tudjuk alkalmazni úgy a beépített **pca()** függvényt, ahogy szükségünk lenne rá, mivel jelenleg úgy érzékeli a függvény, hogy n különböző változónk van (mivel minden oszlopot egy külön változónak feleltet meg). Éppen ezért a mátrixot transzponálni kell. A transzponálás után kapjuk meg a végleges, $n \times 115$-ös mátrixot, amire már lehet úgy alkalmazni a főkomponens analízist, ahogy ténylegesen szükség van.

PCA A transzponált mátrix utolsó, 115. oszlopát kimentettem egy külön oszlopvektorba, mivel az utolsó oszlopban a szívütések osztályai találhatóak és a főkomponens analízishez nincs ezekre szükség.

Az **X** mátrixot odaadjuk paraméterként a **pca()** függvénynek. Ennek a függvénynek szintén több visszatérési értéke is lehet, amelyeket például statisztikai elemzésekhez is fel tudunk használni, viszont a diplomamunkám során csak az egyik visszatérési értékére, a **scorera** lesz szükség. Ez egy $n \times 114$-es mátrix és ebben a mátrixban található meg az összes főkomponenshez tartozó érték. Mivel a **score** mátrix oszlopai a főkomponensek szórásai alapján vannak rendezve, így a 3.7 rész alapján csak az első 18 oszlopot másoljuk át egy **S** nevű mátrixba. Ehhez a mátrixhoz hozzáillesztettem a PCA függvény alkalmazása előtt kimentett annotációkat tartalmazó oszlopvektort és az így kapott $n \times 19$-es mátrixot elneveztem **tenyleges_adathalmaznak**, mivel a CNN18 nevű modellem ezzel a mátrixszal fog dolgozni.

Adatok kiíratása Hasonlóan a **feldolgoz** függvényhez, itt is szerepel az **fopen** függvény, amely ugyanazt a feladatot látja el: amennyiben létezik a paraméterként megadott szöveges dokumentum, abban az esetben egy üres dokumentummal felülírjuk az aktuálisat, így elkerülhetjük a felesleges adatok generálását.

A program végén, szintén az előző programban látottakhoz hasonlóan, a **dlmwrite()** függvény segítségével kiírjuk a mátrix teljes tartalmát a **to** paraméterként megadott dokumentumba.

5.2.3. feldolgoz_wkn.m

Ez a függvény hasonlít a **feldolgoz** nevű függvényemhez, viszont néhány apróbb változást kellett eszközölni, hogy a megfelelő adatokat tudjam továbbadni a CNN300 és a WKN300 nevű modelljeimnek. A feldolgozási folyamat csak a szegmensek kiméréséig tart és, szemben az előző feldolgozási módszerrel, a függvény végén található meg a mátrixszá alakítás, amely a bemenő adatokból készít egy $n \times 301$-es mátrixot (ahol n továbbra is a szívütések száma). A 301-es méret onnan adódik, hogy minden jelhez tartozik egy 300 mintavételezési pontból álló szegmens és egy osztály. A függvény lefutása után a mátrixokat alapértelmezetten a **training_wkn** és a **testing_wkn** nevű szöveges dokumentumban tárolja a program. Később ezeket használtam fel a CNN300 és a WKN300 tanításához is.

Megjegyzés. Fontos megjegyezni, hogy mivel itt nem történik wavelet dekompozíció, így nincs értelme azokat a jeleket „eltüntetni”, amelyek a felvétel valamelyik végéhez vannak közel. Éppen ezért a **training_wkn** és a **testing_wkn** szöveges dokumentumban több jel található meg, mint a másik két dokumentumban.

5.2.4. diplomamunka.m

A **diplomamunka** nevű MATLAB fájlban találhatóak azok a függvény meghívások, amiket a diplomamunkám elkészítése során felhasználtam. Természetesen a paramétereket át lehet nevezni, viszont arra kell figyelni, hogy a **feldolgoz** függvény **2.** és a **fokomponens** függvény egyetlen paraméterének **meg kell egyeznie** a tökéletes működés érdekében.

6. Osztályozás Pythonban

Ebben a fejezetben kerülnek bemutatásra azok a Python programok, amelyeket végrehajtva lehet a jeleket osztályozni különböző modellek segítségével. Bár három programot készítettem, minden modellhez egyet, de a programok felépítése kis eltéréssel hasonlít egymásra, így először általánosságban mutatom be a programok szerkezetét.

6.1. A programok ismertetése

Előkészületek A programkódok megtalálhatóak a feltöltött könyvtár *Python* nevű mappájában. A programokat a Google Colaboratory (vagy röviden Colab) [32] nevű szolgáltatásán keresztül lehet futtatni. A Colab egy notebook alapú szerkesztő, amely Ipythont használ - éppen ezért ezeknek a notebookoknak a kiterjesztése a Python programoknál megszokott .py helyett .ipynb. A google ezen szolgáltatása segítségével különböző, gépi tanulást érintő problémákat lehet implementálni, a hosszabb futási időt igénybe vevő programokat pedig a Google ingyenesen használható GPU-in is tudjuk futtatni. A szolgáltatás használhatához Google fiókra van szükség.

Mindhárom program úgy van megírva, hogy CPU-val és GPU-val is lehessen futtatni. A CNN18 és a CNN300 esetében mindegy, melyiket választjuk, mert a tanulási folyamat gyors, viszont a WKN300 esetében erősen ajánlott GPU-t használni. A Google Colabon belül a futtatókörnyezetet a „Futtatókörnyezet → Futtatókörnyezet módosítása” opció segítségével lehet beállítani. Az alapértelmezett beállítás a **None**, ekkor a program CPU-t fog használni futtatáskor.

Bár nincs szükség semmit se előtelepíteni, a programok futtatásához szükség van két szöveges dokumentum betöltésére. A CNN18 modell esetében a **training** és a **testing**, a CNN300 és a WKN300 modellek esetében pedig a **training_wkn** és a **testing_wkn** nevű szöveges dokumentumokra van szükség.

Szükséges könyvtárak, packagek A programok helyes működéséhez szükség van néhány könyvtár és package importálására:

- **numpy** - A NumPy egy nyílt forráskódú könyvtára a Python nyelvnek, melynek segítségével nagy, illetve többdimenziós tömbökön és mátrixokon lehet különböző műveleteket végrehajtani. A könyvtárat szokás az importálás után np-ként elmenteni.

- **torch** - Több különböző nyílt forráskódú gépi tanulás keretrendszer létezik (például TensorFlow, Keras), de ezek közül a dolgozatom során én a PyTorchot használtam, amelynek használatához szükség van a Torch nevű könyvtár importálására.

- **torch.optim** - A Torch könyvtár egyik package-e, aminek segítségével optimalizációs függvényeket lehet implementálni. A packaget az importálás után optim néven mentettem el és így is fogom használni.

- **torch.nn** - A Torch könyvtár egyik package, elengedhetetlen, ha PyTorch-csal dolgozik valaki. Ennek a packagenek segítségével lehet neurális hálót létrehozni. Ezt a packaget nn néven mentettem el.

- **torch.nn.functional** - Ennek a packagenek a segítségével a neurális háló egyes rétegeire lehet különböző függvényeket (például aktivációs függvényeket) alkalmazni. Ezt a packaget egyszerűen F-ként mentettem el.

- **torch.utils.data** - Ebből a packageből a Datasetet és a DataLoadert kell importálni, amelyeknek az adatok betöltésében van nagy szerepük.

Adatok beolvasása Az adatok beolvasását a NumPy könyvtár **loadtxt()** nevű függvényének segítségével végeztem el. Ez a függvény két paramétert vár: egy .txt kiterjesztésű szöveges dokumentumot és az elválasztó karaktert.

Gépi tanulás esetén szokás a bemeneti adathalmazt X-szel, a hozzájuk tartozó címkéket Y-nal jelölni. Ezt a logikát követik a változók elnevezései is: az **x_train** és az **x_test** változók a tanító- és teszthalmaz azon adatait tartalmazzák, amiket a neurális hálónak szeretnénk továbbadni - egyszerűen elhagyjuk az utolsó oszlopot, mert ezekben a jelekhez tartozó osztálycímkék vannak, és amelyeket az **y_train** és **y_test** változókban tároltam el.

Az adatok beolvasása, valamint az X és Y halmaz szétválogatása közben néhány adatot megmutatok, ehhez implementáltam egy nagyon egyszerű **kiirat** nevű függvényt. Az CNN18 python notebook esetében 5, a másik kettő esetében pedig 3-3 sort jelenítek meg.

Tanításhoz használt eszköz A tanításhoz szükséges eszközt tárolja a **device** nevű változó. Ennek az értéke 'cuda', ha elérhető GPU-t használunk és 'cpu', ha CPU-t.

Adatok továbbítása a neurális hálózatnak Ahhoz, hogy a neurális hálózat tudjon tanulni, először mind a négy változót, amiben jelenleg a tanító- és teszthalmaz adatai találhatóak, tömbbé kell alakítani. Ehhez alkalmaztam a NumPy array nevű függvényét. Ezután ezeket a tömböket **tenzorrá** kell alakítani.

A tenzor olyan többdimenziós mátrix, amelynek segítségével az adatainkat tudjuk tárolni. Számos PyTorch Tensor [33] létezik, ezek közül az **x_train**-hez és az **x_test**-hez Float Tensort, az **y_train**-hez és az **y_test**-hez pedig Long Tensort használtam. Észrevehető, hogy a tenzorrá alakításnál az **y_train** és az **y_test** értékeiből elvettem egyet. Ennek oka egyszerű: a modell a bemenő jelhez öt osztálycímkét rendelhet, amelyet 0-tól 4-ig indexel. Ha a modell megkapja az 5-ös címkéjű jelet, akkor nem tud vele mit kezdeni és IndexError-ral leáll a tanítás.

A következő lépésben normalizáltam egy kicsit a tenzorokat: a két, bemeneti adatokat tartalmazó tenzor összes elemét leosztottam azzal az értékkel, amelyiknek az abszolút értéke a legnagyobb volt. Így a tenzorok -1 és 1 közötti értékeket fognak tartalmazni. Ezzel a módszerrel lehet korrigálni a nagyon kilengő adatokon.

A következő osztályra, aminek a **MyDataSet** nevet adtam, azért van szükségünk, mert a következő blokkban található DataLoadernek tovább kell adnunk az X és Y halmazokat és ezeknek a halmazoknak indexelhetőnek kell lenniük és ezeket célszerű egy objektumba összerakni.

Ezután az adatokat a **DataLoader()** segítségével betöltjük. Ennek a DataLoader() nevű metódusnak több paramétert is megadtam:

- Az előbb említett objektumot a tenzorokkal.
- A batch sizet, azaz, hogy hányasával küldje el a Data Loader a neurális hálózatnak az adatokat (2.2.2. alfejezet) - mindhárom program esetében 8-as batch sizet választottam.
- Azt, hogy megkeverje-e a Data Loader az adatokat vagy sem (a tanítóhalmazt keverje össze, de a teszthalmazt ne).
- Azt, hogy hány szálon futtassuk az adatok beolvasását (**num_workers**). Ennek 0 értéket adtam, mert felesleges és problémás több szálon futtatni a beolvasást.

A tanításhoz és a teszteléshez is van külön Data Loader és ezeket a beolvasásokat a **trainloader** és a **testloader** változók tárolják.

Tanítás A tanításhoz első lépésként meg kell adnunk egy modellt. Amennyiben van elérhető GPU, úgy a modellt át kell rakni a GPU-ra, mert ha nem tesszük, akkor a tanítás során hibát kapunk.

A következő lépésként meg kell adnunk a veszteségfüggvényt (amit a **criterion** változó tárol) és a tanításhoz használt módszert (amit az **optimizer** változóban tárolunk). A tanításhoz a 2.2.2. részben bemutatott sztochasztikus gradiens ereszkedő módszert alkalmaztam. Ehhez a módszerhez három paramétert kell megadni: a modell paramétereit, a tanulási rátát, ami azt határozza meg, mekkora lépésekben halad a modell a súlyok változtatásánál (nehéz megtalálni, mert ha túl nagy, akkor divergál, ha túl kicsi, akkor pedig sokáig tart a tanítás), illetve egy momentum értéket, aminek gyakran 0.9-es értéket adnak meg.

A korai megállás fontos a neurális hálózatok tanításánál, hiszen enélkül a hálózat túltanulna, azaz a modell az általános tulajdonságok mellett az aktuális véges mintahalmaz egyedi furcsaságait is megtanulja, ami ahhoz vezet, hogy a modell a tanítóhalmazon jó eredményt ér el, de a teszthalmazon kevésbé jót. Éppen ezért a modellt ki kell értékelni epochonként a teszthalmazon: amennyiben a veszteségfüggvény értéke nem csökken, úgy a tanítást is abbahagyjuk. Amennyiben a modellnek

sikerül alacsonyabb veszteséget elérni a korábbi legalacsonyabbhoz képest, úgy a modell súlyait elmentjük a best_net.pth nevű fájlba, amit később, a kiértékelés során fel fog használni a program.

A korai megállást egy **EarlyStopping** nevű osztály reprezentálja, ami két paraméterrel dolgozik. A **patience** érték azt határozza meg, hogy hányadik olyan epoch után maradjon félbe a tanítás, amikor nem sikerült alacsonyabb veszteséget elérni, a **min_delta** pedig azt, hogy mekkora legyen a különbség a jelenlegi legalacsonyabb és az adott epoch végén kiszámolt veszteség között, hogy azt jobb eredményként könyvelhessük el.

A következő három függvény a tanítás kiértékelésében játszik fontos szerepet. Az **evaluate** nevű függvény azt csinálja, hogy végigmegy a teszthalmaz összes elemén és alkalmazza rá a veszteségfüggvényt, hogy kiszámolja rá a veszteséget. Az **eval_acc** és az **eval_stat** függvények ugyanazt csinálják: kiszámolják a modell pontosságát a teszthalmazon. A különbség annyi, hogy az **eval_stat** nevű függvényt a végső kiértékelésnél alkalmaztam, míg a másikat a tanítás közben.

Végezetül a neurális hálózatot a **train** függvény meghívásával lehet tanítani. A tanítás - korai megállás nélkül - pontosan annyi epochon keresztül fog zajlani, amennyi epochot megadunk a **MAX_EPOCH** nevű konstansnak. Ez az érték nálam eltérő notebookonként, de mivel az általam megadott tanulási ráták mellett amúgy is bekövetkezik mindig a korai megállás, így lényegtelen, mennyire van beállítva. A tanítás két lépés ismétléséből áll:

- Előre propagálásnak nevezzük azt, amikor a bemenetből és a pillanatnyi súlyokból kiszámolja a neurális hálózat a kimenetet.
- Visszapropagálásnak hívjuk azt, amikor a hiba alapján kiszámoljuk, hogy hogyan változzanak meg a súlyok és az alapján meg is változtatjuk azokat.

A függvény tartalmaz még egy olyan részt is, ami statisztikákat írat ki: 1000 iterációnként kiíratja a veszteséget, valamint az epochok végén láthatjuk a teszthalmazon mért veszteséget, a neurális hálózat pontosságát és, ha a veszteség nem csökken, akkor azt is, hogy hány epochnál nem csökkent a veszteség a legalacsonyabbhoz képest.

Eredmények kiértékelése Az eredmények kiértékeléshez a legjobban súlyozott modellt töltöttem be (amit a best_net.pth tárol) és arra alkalmaztam az **eval_stat** függvényt, hogy jól lehessen látni, a modell az egyes osztályokhoz tartozó jeleket mely osztályokhoz becsülte meg. A különböző modellek által kapott eredményeket a 7. fejezetben mutatom be és elemzem ki.

7. Eredmények

Ebben a fejezetben bemutatásra kerülnek a különböző modellek által produkált eredmények. Mivel minden futás más és más eredményt hoz, éppen ezért a modellek legjobb eredményeit elemzem ki és hasonlítom össze.

A tanító- és teszthalmaz kiegyenlítetlensége (1. táblázat) nagyban megnehezítette mind a tanítást, mind az eredmények kiértékelését, ugyanis könnyen félre- vagy túltanult a modell, illetve alacsony veszteségfüggvény (és magas pontosság) mellett sem biztos, hogy használható eredményeket produkálnak a különböző modellek.

A modellek pontosságán lehetne fejleszteni, mert jelenleg nem teljesen pontosak, ami meg is látszik a kapott eredményeken. Az architektúrák és a hiperparaméterek további optimalizálásával növelhető lenne a modellek pontossága, de ez túlmutat a diplomamunka keretein.

Nagyon fontos azt is megemlíteni, hogy mivel morfológiailag az 1. és a 3. osztály szívütései között lehetséges különbséget tenni (a 2., a 4. és az 5. osztály szívütései között inkább dinamikus különbségek vannak, amikkel nem foglalkoztam a dolgozat készítése során) és a waveletek csak morfológiai jellemzőket tudnak kinyerni, így a CNN18 és a WKN300 modellek esetében az elvárás az, hogy az 1. és a 3. osztály szívütéseit minél pontosabban ismerje fel az adott neurális hálózat.

A modellek eredményeit vizsgálgatva észrevehető, hogy az 5-ös osztálybeli jelből egyetlen egyet se talált meg egyik modell sem. Ez nem véletlen, mivel a tanító- és teszthalmazban összesen 15 darab ilyen szívütés található. Ez nem tekinthető kritikus hibának, éppen ezért más kutatásoknál előfordult az is, hogy egyszerűen elhagyják ezt az osztályt [22].

7.1. Alkalmazott módszerek

A modellek kiértékeléséhez egyszerű statisztikai mutatókat használtam fel [34]. Ezeknek a statisztikai mutatóknak alapja négy különböző érték:

1. **Valódi pozitívnak** (a későbbiekben röviden TP az angol True Positive kife-

jezésből) tekintünk egy adatot, ha az adat pozitív és a modellünk is pozitívnak becsüli meg.

2. **Hamis pozitívnak** (a későbbiekben röviden FP az angol False Positive kifejezésből) tekintünk egy adatot, ha az adat negatív, de a modell pozitívnak becsüli meg.

3. **Hamis negatívnak** (a későbbiekben röviden FN az angol False Negative kifejezésből) tekintünk egy adatot, ha az adat pozitív, de a modell negatívnak becsüli meg.

4. **Valódi negatívnak** (a későbbiekben röviden TN az angol True Negative kifejezésből) tekintünk egy adatot, ha az adat negatív és a modellünk is negatívnak becsüli meg.

Megjegyzés. Statisztikában a hamis pozitív elsőfajú, míg a hamis negatív másodfajú hibának számít.

A négy érték segítségével számos különböző mérőszámot ki lehet számolni, amelyek alapján a modell hatékonyságát tudjuk vizsgálni.

A három legfontosabb metrika, amelyet a következő alfejezetben vizsgáltam, általában együtt szokott állni:

- **Pontosság** (vagy angolul accuracy), amely a jól osztályozott objektumok számának arányát adja meg az összes objektum számához viszonyítva. Képlettel: $\frac{TP+TN}{TP+TN+FP+FN}$. Önmagából a pontosságból nehéz bármire is következtetni. Vegyük például az általam használt bármelyik halmazt, ahol a szívütések 88%-a 1-es osztályba tartozik. Ha a modell az összes szívütést ehhez az osztályhoz becsüli, akkor a modell pontossága eléri a 88%-ot, viszont semmivel se vagyunk előrébb.

- A **precizitást** (angolul precision) az alábbi hányados adja: $\frac{TP}{TP+FP}$, azaz a pozitívként megtalált objektumok mekkora része volt ténylegesen pozitív.

- A **megbízhatóságot** (angolul recall) a következőképp számolhatjuk ki: $\frac{TP}{TP+FN}$,

azaz azt mutatja meg, hogy az adott osztályhoz tartozó objektumok mekkora részét találta meg az adott modell [35].

A modell eredményességét pedig konfúziós mátrix [36] segítségével tudjuk elemezni. A konfúziós mátrix legáltalánosabb formája egy 2×2-es táblázat, ahol a pozitív és negatív ismert és becsült osztályokat tudjuk vizsgálgatni a fentebb leírt módon. De nem csak a 2×2-es változata ismert, a modell becsült eredményeit egy $n \times n$-es táblázatba is rendezhetjük (ahol n az osztályok számát jelöli) - az összehasonlítás során én is ezt a változatot fogom használni többször is, ugyanis egy ilyen táblázatban sokkal jobban kirajzolódnak a modell becslései és sokkal jobban látszódik, hogy egy adott osztályt mennyire pontosan becsült meg a modell. Ebben az esetben egy kitüntetett osztályt tekintünk pozitívnak, a többi osztályt összevonjuk és negatívként tekintünk rá - ezek után ugyanúgy tudjuk értelmezni a fentebb bemutatott négy értéket.

Egy egyszerű példán keresztül szeretném bemutatni ezt a négy értéket és a konfúziós mátrixot. Tekintsük a 3. táblázatot és tegyük fel azt, hogy ezeket az értékeket az 1-es osztályra szeretnénk megvizsgálni, tehát az 1-es osztály pozitív lesz, a többi osztály pedig negatív.

		Becsült osztály				
		1	2	3	4	5
Ismert osztály	1	TP	FN	FN	FN	FN
	2	FP	TN	TN	TN	TN
	3	FP	TN	TN	TN	TN
	4	FP	TN	TN	TN	TN
	5	FP	TN	TN	TN	TN

3. táblázat. A négy statisztikai értéket szemléltető táblázat (1-es osztály alapján).

- **Valódi pozitívnak** (a táblázatban zöld) tekinthető az összes olyan adat, amely 1-es osztályú és erre az osztályra becsüli meg a jelet a modell.
- **Hamis pozitív** (a táblázatban kék) minden adat, amelyet a modell 1-es osztálybelinek becsül meg, de nem is ahhoz az osztályhoz tartozik, valamint

- **Hamis negatív** (a táblázatban sárga) egy adat, ha a modell az 1-es osztálybeli adatot másik osztályúnak becsüli meg.
- **Valódi negatív** (a táblázatban piros) minden olyan adat, ahol a modell bármelyik negatív osztálybeli adatot jól becsül meg.

7.2. Kiértékelés

Ebben az alfejezetben hasonlítom össze a modellek által kapott eredményeket.

Az első három részben a három modell teljesítményét mutatom meg két táblázatban. Az egyik táblázat egy konfúziós mátrix lesz és a modell legjobb eredményét ábrázolja. A másik táblázat pedig tíz, egymástól teljesen független futtatás különböző statisztikai adatait mutatja meg.

7.2.1. CNN18

A három modell közül ennél a legnehezebb minél nagyobb pontosságot elérni. Ennek oka az, hogy a konvolúciós neurális hálózat viszonylag alacsony dimenziószámú jeleket kap meg bemenetként és ilyen esetekben nagyon nehéz a neurális hálózatnak bármilyen következtetésre jutni. Éppen ezért (és azért is, mert wavelet dekompozíció történt korábban) ennél a modellnél elfogadjuk azt, hogy az 1. és a 3. osztályú jeleket valamennyire jól megtalálja, a többit viszont nem.

		Becsült osztály				
		1	2	3	4	5
Ismert osztály	1	43.916	0	327	0	0
	2	1.966	0	6	0	0
	3	1.835	0	1.385	0	0
	4	381	0	7	0	0
	5	6	0	1	0	0

4. táblázat. A teszthalmazhoz tartozó konfúziós mátrix (CNN18).

Ahogy a 4. táblázatban is látható, ez a modell az 1. osztály jeleinek nagyob-

bik részét (kb. 98%) felismeri, viszont a 3. osztályú jelek közül körülbelül kétszer annyit becsül 1. osztályúnak, mint 3. osztályúnak, amit az alacsony dimenziószámú bemenettel lehet indokolni.

Az 5. táblázatban látható statisztikai adatokból az látszik, hogy a tíz futás során a modell mindig legalább közel 42 ezer 1-es osztályú jelet ismert fel pontosan. A szórásból az is látszik, hogy nem nagyon voltak eltérések a pontos találatok számai között.

Oszt.	**Maximum**	**Minimum**	**Átlag**	**Szórás**
1	43.916	41.967	43.253,4	651,62
2	0	0	0	0
3	1.633	110	984,5	561,82
4	0	0	0	0
5	0	0	0	0

5. táblázat. A CNN18 modell által eltalált jelek statisztikája 10 futás alapján.

A 3-as osztálybeli jelekkel kapcsolatban viszont nem volt ennyire tökéletes a modell. Néhány futásnál alacsonyabb találati arányt ért el (110, 269, 402 találat), néhány futásnál viszont sikerült ezer feletti találatot elérnie. Ezt a kilengést bizonyítja a magasnak tekinthető szórási érték is.

7.2.2. CNN300

A CNN300 modell, mivel a szívütés szegmensekre csak konvolúciós rétegeket és teljesen összecsatolt rétegeket alkalmaz, tehát nem történik semmilyen wavelet transzformáció, valamennyire képes megtalálni a 2. és 4. osztálybeli jeleket is.

A 6. táblázatban látható a legpontosabb eredménye a CNN300 modellnek. Ahogy a konfúziós mátrixon is látható, amikor a modell a legpontosabb eredményt érte el, nem talált 4. osztálybeli jelet, viszont a 3. osztálybeli jelek 86.4%-át jól becsülte meg. Bár eltalált a modell néhány 2. osztálybeli jelet, de ennek az aránya viszonylag elenyésző (7,6%), így emiatt ebben a részben nem lehet teljes bizonyossággal megállapítani, hogy ez a modell jobb, mint az előző.

		Becsült osztály				
		1	2	3	4	5
Ismert osztály	1	42.296	346	1.489	128	0
	2	1.324	149	490	9	0
	3	419	7	2.783	12	0
	4	321	6	61	0	0
	5	3	0	4	0	0

6. táblázat. A teszthalmazhoz tartozó konfúziós mátrix (CNN300).

A 7. táblázatban található statisztikai adatokból az látszik, hogy a CNN300 modell az előzőhöz képest kevesebb 1. osztálybeli szívütést becsült meg pontosan és a szórása is nagy. A 2. osztálybeli jelek esetén is látszik, hogy bár egy esetet leszámítva mindig jól becsült meg a modell néhány, ebbe az osztályba tartozó szívütést, viszont legalább 5%-os pontosságot (azaz megtalált legalább 98 jelet) a tíz futásból csak négynél ért el.

A 3. osztálybeli jelek esetében elmondható, hogy az átlagosan pozitívként megbecsült jelek aránya jónak tekinthető, hiszen átlagosan 79%-os találati arányt ért el a modell, amit a szórás alacsonyabb értéke is megerősít. Átlagosan a 4. osztálybeli szívütések 0,6%-át becsülte meg jól a modell. Ezt az átlagot az egyik futásnak köszönheti a modell, ugyanis egyszer 14 ilyen osztálybeli jelet talált meg a neurális hálózat, míg a többi kilenc esetben legfeljebb 3 pontos találata volt (és többször is előfordult, hogy egy, ebbe az osztályba tartozó szívütést sem talált meg).

Oszt.	**Maximum**	**Minimum**	**Átlag**	**Szórás**
1	43.002	39.149	41.530	1.260,12
2	185	0	77,50	64,59
3	2.816	2.091	2.548,7	217,85
4	14	0	2,30	4,22
5	0	0	0	0

7. táblázat. A CNN300 modell által eltalált jelek statisztikája 10 futás alapján.

7.2.3. WKN300

A WKN300 modell kicsit a CNN18 és a CNN300 keveréke, hiszen van benne wavelet feldolgozás is, mint az előbbi esetében, de a bemenetet azok a 300-as méretű szívütés szegmensek adják, amit az utóbbi modell tanításához is felhasználtam. Éppen ez utóbbiból következik az, hogy jobb és pontosabb eredményt vártam ettől a modelltől, mint a CNN18-tól, hiszen a wavelet kernel után (ami a konvolúciós neurális hálózat első rétege) a neurális hálózat egy nagyobb dimenziószámú jelet tud vizsgálni.

		Becsült osztály				
		1	2	3	4	5
Ismert osztály	1	43.912	0	347	0	0
	2	1.885	0	87	0	0
	3	868	0	2.353	0	0
	4	360	0	28	0	0
	5	1	0	6	0	0

8. táblázat. A teszthalmazhoz tartozó konfúziós mátrix (WKN300).

Ahogy a 8. táblázatban is látható, a modell a legpontosabb futásakor nem talált 1-es és 3-as osztályú szívütéseken kívül más osztályba tartozó jelet, viszont viszonylag nagy arányban találta meg e két osztályhoz tartozó jeleket (99.2% és 73.1%).

Oszt.	**Maximum**	**Minimum**	**Átlag**	**Szórás**
1	43.912	41.928	43.042,4	696,14
2	2	0	0,2	0,63
3	2.598	2.221	2.394,1	104,92
4	2	0	0,8	0,92
5	0	0	0	0

9. táblázat. A WKN300 által eltalált jelek statisztikája 10 futás alapján.

A 9. táblázatban látható statisztikai adatokból az látszódik, hogy a modell az 1-es és a 3-as osztály szívütéseit nagyobb arányban és kisebb szórással találta el. Egy kis

érdekesség, hogy egy illetve öt futás során talált a modell 2. és 4. osztálybeli jeleket is. Mivel az átlagos találati arány ennél a két osztálynál nagyon alacsony (0.2% alatt mindkét esetben), így ezek a találatok a három modell összehasonlítását nem fogják befolyásolni.

7.2.4. Összehasonlítás

A 10. táblázatban látható, hogy az egyes modellek által az adott osztályra becsült jelek közül hány volt valódi pozitív, hamis pozitív és hamis negatív.

	CNN18			**CNN300**			**WKN300**		
Oszt.	TP	FP	FN	TP	FP	FN	TP	FP	FN
1	43916	4188	327	42296	2067	1963	43912	3114	347
2	0	0	1972	149	359	1823	0	0	1972
3	1385	341	1835	2783	2044	438	2353	468	868
4	0	0	388	0	149	388	0	0	388
5	0	0	7	0	0	7	0	0	7

10. táblázat. A modellenként megtalált jelek osztályokra bontva.

Ahogy az a táblázatból is jól kiolvasható, hiába talált a CNN18 modell több 1-es osztálybeli jelet el, illetve hiába becsülte meg jobban a CNN300 modell a 3-as osztálybeli jeleket, a WKN300 modell kevesebb jelet becsült meg tévesen pozitívnak erre a két osztályra, mint a másik két modell, így már ebből a táblázatból is látszik, hogy valószínűleg a legutolsó neurális hálózat tanult a legpontosabban.

Ezt a sejtést csak megerősítik a 11. táblázatban látható értékek, ahol az látható, hogy az egyes modellek mennyire voltak pontosak, precízek és megbízhatóak az osztályokra és összességében nézve.

Ennek a táblázatnak a tanulsága szerint a WKN300, azaz a folytonos wavelet konvolúciós réteget használó modell volt a legpontosabb a három közül, amely szintén megerősíti azt a feltételezést, amit a 2.2.4. részben állítottam és amit a [8] cikk írói is kaptak, miszerint azon konvolúciós neurális hálózatok, amelynek az első rétege wavelet kernel, pontosabb eredményekre képes, mint egy általános konvolúciós

neurális hálózat.

	CNN18			**CNN300**			**WKN300**		
Oszt.	Pont.	Prec.	Megb.	Pont.	Prec.	Megb.	Pont.	Prec.	Megb.
1	90,9%	91,3%	99,3%	91,9%	95,3%	95,6%	93,1%	93,4%	99,2%
2	96,0%	-	0,0%	95,6%	29,3%	7,6%	96,0%	-	0,0%
3	95,6%	80,2%	43,0%	95,0%	57,7%	86,4%	97,3%	83,4%	73,1%
4	99,2%	-	0,0%	98,9%	0,0%	0,0%	99,2%	-	0,0%
5	99,9%	-	0,0%	99,9%	-	0,0%	99,9%	-	0,0%
Össz	**90.9%**			**90.8%**			**92.9%**		

11. táblázat. A modellek pontossága, precizitása és megbízhatósága osztályokra bontva.

Megjegyzés. A 11. táblázat néhány cellájában található **-**. Ennek oka az, hogy azokra az osztályokra 0 valódi pozitív és 0 hamis pozitív jelet talált a modell és mivel e kettő összege alkotja a precizitás számításához használt képlet nevezőjét, így nullával kellene osztani.

Megjegyzés. A 11. táblázatnál is látszik, hogy összességében a modellek ugyanannyira pontosak, precízek és megbízhatóak, éppen ezért ezt a három metrikát a különböző osztályokra célszerű alkalmazni, hogy látszódjanak a különbségek.

8. Összegzés

A diplomamunkám célja az volt, hogy három, szerkezetében eltérő konvolúciós neurális hálózatot szívütések osztályozására betanítsam és a kapott eredményeket összehasonlítsam.

A dolgozat 2. fejezetében bemutattam néhány, a jelfeldolgozás területéhez erősen kapcsolódó fogalmat, valamint ismertettem a mesterséges neurális hálózatot és a tanítási folyamatot.

A 3. fejezetben bemutattam az adatbázis feldolgozásának folyamatát lépésről lépésre, ahol alkalmaztam a 2. fejezetben bemutatott lépéseket. Az adatbázis teljes feldolgozása csak a CNN18 modell szempontjából volt lényeges, míg a CNN300 és a WNK300 modellek esetében csak a szívütések elkülönítésig zajlott az adatbázis feldolgozása.

A 4. fejezetben bemutatásra került mindhárom modellem: a CNN18 tradícionális osztályozási folyamatot hajt végre, a CNN300 modellem egy általános, míg a WKN300 egy modell-alapú konvolúciós neurális hálózat. Ezt a három modellt tanítottam be szívütések osztályozására.

Az 5. és a 6. fejezetekben a programok felépítésének dokumentációi olvashatóak.

A 7. fejezetben az osztályozási folyamat során kapott eredményeket hasonlítottam össze külön-külön és egyben is.

A diplomamunkám végén megállapíthatjuk, hogy a modellek a tanítás során hozták a várt eredményt, valamint azt is, hogy a várakozásnak megfelelően a WKN300 modell lényegesen hatékonyabb volt, mint a másik két modell.

Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőmnek, Bognár Gergőnek, aki szakértelmével és javaslataival segítette a diplomamunkám létrejöttét. Köszönöm a konzultációkat, mely során betekintést nyerhettem a jelfeldolgozás és a mesterséges intelligencia világába.

Szeretném külön megköszönni Herczeg Istvánnak és Kiss Bencének az összes tanulással töltött órát. A csapatmunkának és az összefogásnak fontos szerepe volt abban, hogy az összes akadályt sikerrel vettük.

Külön köszönöm Gáspár Dávidnak, hogy a dolgozatban található nyelvtani hibákra felhívta a figyelmemet és számíthattam a segítségére, ha át kellett olvasni a dolgozatot.

Végezetül köszönöm a családomnak és barátaimnak a támogatást, amit az egyetemi éveim alatt kaptam tőlük.

Hivatkozások

[1] A felhasznált kép forrása: https://bit.ly/ekgjel (Hozzáférés ideje: 2022. május 21.).

[2] Wikipédia. *Morlet-wavelet*. https://en.wikipedia.org/wiki/Morlet_wavelet (Hozzáférés ideje: 2022. május 21.).

[3] Jianping He, zhi Zhou, Genda Chen, and Jinping Ou. Measurement accuracy improvement of brillouin signal using wavelet denoising method. *Proceedings of SPIE - The International Society for Optical Engineering*, 7293, 03 2009.

[4] Mesterin. *Neurális hálózatok*. https://bit.ly/ann-layers (Hozzáférés ideje: 2022. május 21.).

[5] Senén Barro, Ramon Ruíz, Diego Cabello, and José Mira. Algorithmic sequential decision-making in the frequency domain for life threatening ventricular arrhythmias and imitative artefacts: a diagnostic system. *Journal of biomedical engineering*, 11 4:320–8, 1989.

[6] Keiichiro Minami, Hiroshi Nakajima, and Takeshi Toyoshima. Real-time discrimination of ventricular tachyarrhythmia with fourier-transform neural network. *IEEE transactions on bio-medical engineering*, 46:179–85, 03 1999.

[7] Robert E Kass and Colleen E Clancy. *Basis and treatment of cardiac arrhythmias*, volume 171. Springer Science & Business Media, 2005.

[8] Tianfu Li, Zhibin Zhao, Chuang Sun, Li Cheng, Xuefeng Chen, Ruqiang Yan, and Robert Gao. WaveletKernelNet: An interpretable deep neural network for industrial intelligent diagnosis, 11 2019.

[9] Ary Louis Goldberger and Emanuel Goldberger. *Clinical Electrocardiography, A Simplified Approach*, volume 7. Critical Care Medicine, 1981.

[10] June Edhouse, R K Thakur, and Jihad M Khalil. Conditions affecting the left side of the heart. *BMJ*, 324(7348):1264–1267, 2002.

[11] SW Smith. The scientist and engineer's guide to digital signal processing, california tech (14. fejezet). *Publishing, San Diego, CA, USA*, 1997.

[12] Weisz András. *Wavelet-analízis*. Egyetemi jegyzet. 2015.

[13] Karlheinz Gröchenig. *Foundations of time-frequency analysis*. Springer Science & Business Media, 2001.

[14] Zempléni András. *Zempléni András nyilvános Idősorok és többdimenziós statisztika jegyzete (2019, 11. előadás)*. https://zempleni.elte.hu/idos19_11.pdf (Hozzáférés ideje: 2022. május 21.).

[15] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[17] Stanford University. *Neural Network 1.* http://cs231n.github.io/neural-networks-1/ (Hozzáférés ideje: 2022. május 21.).

[18] Engedy István. *Konvolúciós neurális hálózatok (CNN)*. http://home.mit.bme.hu/~engedy/NN/NN-CNN.pdf, (Hozzáférés ideje: 2019. december 31.).

[19] Wenyi Liu and Baoping Tang. A hybrid time-frequency method based on improved morlet wavelet and auto terms window. *Expert Systems with Applications*, 38(6):7575–7581, 2011.

[20] P.S. Addison. Wavelet transforms and the ecg: A review. *Physiological measurement*, 26:R155–99, 11 2005.

[21] Can Ye, B. V. K. Vijaya Kumar, and Miguel Tavares Coimbra. Heartbeat classification using morphological and dynamic features of ecg signals. *IEEE Transactions on Biomedical Engineering*, 59(10):2930–2941, 2012.

[22] Eduardo José da S. Luz, William Robson Schwartz, Guillermo Cámara-Chávez, and David Menotti. Ecg-based heartbeat classification for arrhythmia detection: A survey. *Computer Methods and Programs in Biomedicine*, 127:144–164, 2016.

[23] Moody George and Mark Roger. *MIT-BIH Arrhytmia Database*. https://physionet.org/content/mitdb/1.0.0/ (Hozzáférés ideje: 2022. május 21.).

[24] G.B. Moody and R.G. Mark. The impact of the mit-bih arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, 2001.

[25] MIT-BIH. *MIT-BIH Arrhythmia Database Directory*. https://archive.physionet.org/physiobank/database/html/mitdbdir/mitdbdir.htm (Hozzáférés ideje: 2022. május 21.).

[26] Philip Chazal, Maria O'Dwyer, and Richard Reilly. Automatic classification of heartbeats using ECG morphology and heartbeat interval features. *IEEE transactions on bio-medical engineering*, 51:1196–206, 08 2004.

[27] Donghui Zhang. Wavelet approach for ecg baseline wander correction and noise reduction. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 2:1212–5, 02 2005.

[28] Nitish V. Thakor, John G. Webster, and Willis J. Tompkins. Estimation of qrs complex power spectra for design of a qrs filter. *IEEE Transactions on Biomedical Engineering*, BME-31(11):702–706, 1984.

[29] PhysioNet. *WFDB Toolbox for MATLAB and Octave*. https://archive.physionet.org/physiotools/matlab/wfdb-app-matlab/ (Hozzáférés ideje: 2022. május 21.).

[30] ascii.cl. *ASCII Codes*. https://ascii.cl/ (Hozzáférés ideje: 2022. május 21.).

[31] PhysioNet. *PhysioBank Annotations*. https://archive.physionet.org/physiobank/annotations.shtml (Hozzáférés ideje: 2022. május 21.).

[32] Google Research. *Google Colab*. https://colab.research.google.com/ (Hozzáférés ideje: 2022. május 21.).

[33] PyTorch. *PyTorch Tensors*. https://pytorch.org/docs/stable/tensors.html (Hozzáférés ideje: 2022. május 21.).

[34] Google Developers. *Classification: True vs. False and Positive vs. Negative*. https://developers.google.com/machine-learning/crash-course/

classification/true-false-positive-negative (Hozzáférés ideje: 2022. május 21.).

[35] Google Developers. *Machine Learning Glossary*. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (Hozzáférés ideje: 2022. május 21.).

[36] Google Developers. *Machine Learning Glossary*. https://developers.google.com/machine-learning/glossary#confusion_matrix (Hozzáférés ideje: 2022. május 21.).