Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedésmérnöki és Járműmérnöki Kar

**Közlekedés- és Járműirányítási Tanszék**

# GÉPI TANULÁS ALAPÚ SÁVTARTÓ FUNKCIÓ MEGVALÓSÍTÁSA HIL KÖRNYEZETBEN

TDK dolgozat

Készítette: Fehér Árpád, járműmérnöki (MSc)

Konzulens: Dr. Aradi Szilárd, egyetemi adjunktus

2017. november

# Tartalomjegyzék

# 1. BEVEZETÉS

A dolgozat egy gépi tanulás alapú sávtartó funkció megvalósítását tűzte ki célul, amely HiL (Hardware-in-the-Loop) felépítésű szimulációs környezetben kerül kifejlesztésre.

A HiL során a szimulációs körben található egy hardverelem, ami érzékeli a szoftver kimenetét, és bemeneti jelet szolgáltat annak. A feladatom során a hardver egy kamera, ami egy járműdinamikai szimulációs szoftver által megjelenített utat detektál, és annak kimeneti jele alapján a kifejlesztett szoftvernek meg kell határozni az úton haladó jármű pályán tartásához szükséges paramétereket. A kifejlesztett szoftver neurális hálózat alapokon nyugszik.

## 1.1. Motiváció

A dolgozat elkészítését megelőző munkáim során, először egy gokarton, utána pedig egy személygépjárművön autonóm járműfunkciók megvalósításával és tesztelésével foglalkoztam. Számos témába illő cikket, tudományos munkát olvastam, amik közül kiemelten érdekeltek a neurális hálózat alapú rendszerek.

1989 januárjában megjelent cikkben leírtak szerint ALVINN (Autonomous Land Vehicle In a Neural Network) néven, egy katonai projekt keretében megvalósítottak egy sávtartó funkciót. Egy 3 rejtett rétegből álló neurális hálózat bemeneti jelét egy 30x32 képkockából álló kék színsávú kép és egy 8x32 csatornás lézeres távolságmérő szolgáltatta. A hálózat kimenete a jármű irányítására szolgáló kormányszög volt. A cikkben azt írják, hogy azért alkalmaztak kék színsávot, mert ott különült el a legjobban az út a környezettől. A szenzorokat és a feldolgozó számítógépet egy katonai mentőautóra telepítették. A telepített számítógép hűtőszekrény méretű volt, táplálásához egy 5 kW-os generátort használtak. Számítási kapacitása egy tizede egy mai okosóráénak. A jármű képes volt elérni akár a 112 km/h sebességet.

Az utóbbi években újra felkapott téma lett a kamerán alapuló neurális hálózatok a jármű és IT iparban. Az NVIDIA nagy erőkkel bekacsolódott az autonóm járműkutatásba. Készített Drive PX néven egy hardvert, amelynek nemrég adta ki a harmadik generációs változatát, ami elvileg támogatja az ötös szintű automatizáltsági szintet, ami azt jelenti, hogy a jármű emberi beavatkozás nélkül tud közlekedni. Természetesen el kell készíteni a hozzá tartozó szoftvert.

Az NVIDIA 2016 augusztusában publikált egy cikket, amiben konvolúciós neurális hálózatot alkalmazva egy három kamerás Drive PX alapú rendszerrel megvalósítottak egy sávtartó funkciót. A hálózat betanításához használt adatkészlet 72 órányi anyagot tartalmazott többféle látási és útviszonyok mellett. Kiemelkedően jó eredményeket értek el. A megoldásukat nagy részletességgel publikálták.

A fent leírtak motiváltak arra, hogy kipróbáljak egy ilyen rendszert és megvizsgáljam az alkalmazási lehetőségeit.

## 1.2. Felépítés

A kutatómunkát angol és magyar nyelvű szakirodalmak, valamint gépi tanulás alapú példák elemzésével kezdtem, aminek során elsajátítottam a tudományterület alapvető módszereit, és áttanulmányoztam az aktuális trendeket és eljárásokat, aminek két fejezetet szenteltem. A második fejezetben bemutatásra kerül a munkám során elsajátított, a dolgozat megértéséhez elengedhetetlen neurális hálózatok alapjait kifejtő ismeretanyag.

A feladatot felügyelt tanulás módszerrel végeztem el, ahol a CarSim szimulációs szoftver adott volt, melyben az irányítást egy Matlab/Simulink beépülő környezetben készített algoritmus végzi. Ugyanazt a feladatot két különböző típusú neurális hálózattal is megoldottam. Az első típus egy többrétegű, regresszió típusú neurális hálózat, a második pedig egy konvolúciós neurális hálózat.

Mindkét hálózat esetén, első lépésként kifejlesztésre került egy laterális és egy longitudinális szabályzó, amely végig vezeti az ismert geometriájú pályán a járművet. A szoftver eközben elmenti a neurális hálózat tanításához szükséges összetartozó be- és kimeneti értékeket, a tanító mintapont párokat. A bemeneti értékek a kamera képkockái, a kimeneti pedig a jármű irányító parancsok. A második lépésben meg kellett határozni a hálózat méretét, felépítését és egyéb paramétereit (hiperparaméterek), valamint a mintapárokat tanításra alkalmassá tenni. Ezek után megtörtént a hálózat betanítása. A betanított hálózat teljesítmény mutatóinak értékelése alapján a hálózat felépítése és paraméterei többszöri finomhangoláson estek keresztül. A harmadik fázis a betanított hálózat beépítése a szimulációs körbe, amikor a jármű irányítását a kamera képe alapján a hálózat végzi.

# 2. NEURÁLIS HÁLÓZATOK ALAPJAI

A fejezet célja egy szakirodalmi áttekintés, amely magában foglalja a neurális hálózatok felépítésével kapcsolatos legfontosabb fogalmakat. Továbbá feladata a dolgozat témáit lefedő elméleti háttéranyag ismertetése, amely hozzásegíti az olvasót a tudományos tartalmak értelmezéséhez.

Az elméleti rész kezdetén a dolgozat címében megjelenő gépi tanuláshoz szorosan kapcsolódó mesterséges intelligencia fogalmát szeretném értelmezni. Az MI fogalmára a terület alapvető könyvének számító Stuart Russel, Peter Norvig: Mesterséges Intelligencia [2] könyvéből emelnék át két részletet. Ezen szemléletmódot követve készült a dolgozat.

23. oldal, 5. Összefoglalás: „Ebben a könyvben azt a nézetet fogadjuk el, hogy az intelligencia lényegében a racionális cselekvéssel kapcsolatos. Egy intelligens ágens, ideális esetben, az adott szituációban a legjobb cselekvéshez folyamodik. Az ilyen értelemben vett intelligens ágensek építési problémáit fogjuk tanulmányozni.”

Korábban 4. oldal 1.4 fejezet: Racionálisan cselekedni: a racionális ágens:

„Egy ágens (agent) nem más, mint valami, ami cselekszik (az ágens szó forrása a latin agere – cselekedni). Számítógépes ágensektől azonban elvárjuk, hogy legyenek más jellemzői is, amelyekben különböznek a „mezei” programoktól. Ilyen jellemzők például az autonóm vezérlés felügyelte cselekvés, a környezet észlelése, a hosszabb idejű tartós létezés, a változásokhoz történő adaptáció és mások céljainak az átvétele. Egy racionális ágens (rational agent) a legjobb kimenetel érdekében vagy – bizonytalanság jelenlétében – a legjobb várható kimenetel érdekében cselekszik.”

Az utolsó mondat definiálja a racionális ágenst, amelyet a szerzők intelligensnek fogadnak el.

Az olyan rendszereket, amelyek egy bizonyos feladathoz tartozó probléma néhány megoldásából tudtak általánosításokat végrehajtani, és azokból következtetni az ismeretlen részekre, neurális hálóknak nevezték.

A következőkben bemutatásra kerülnek a neurális hálózatok alapfogalmai, alapvető definíciók. A fejezet alapjául a 2006-ban kiadott Neurális hálózatok [3] című könyv első és második fejezete szolgált.

## 2.1. Definíció [3]

Neurális hálózatnak nevezzük azt a hardver vagy szoftver megvalósítású párhuzamos, elosztott működésre képes információfeldolgozó eszközt, amelyre igaz, hogy:

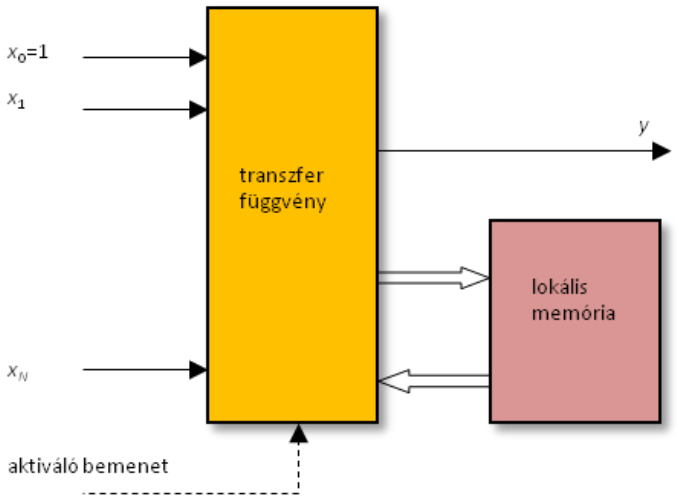
- Azonos vagy hasonló típusú – általában nagyszámú – lokális feldolgozást végző műveleti elem, neuron (processing element) többnyire rendezett topológiájú, nagymértékben összekapcsolt rendszeréből áll.
- Rendelkezik tanulási algoritmussal (learning algorithm), mely általában minta alapján való tanulást jelent, és amely az információfeldolgozás módját határozza meg.
- Rendelkezik a megtanult információ felhasználását lehetővé tevő információ előhívási, vagy röviden előhívási algoritmussal (recall algorithm).

A fenti definíció kulcsfogalmainak tisztázás érdekében a következő fejezetekben kifejtésre kerül az alkalmazott műveleti elemek (neuronok) felépítése, az összeköttetések, tipikus topológiák és a tanulási, előhívási algoritmusok.

A neurális hálózatok alkalmazásánál definíció szerint alapvetően két szakaszt különböztetünk meg. Az első szakaszban történik a hálózat betanítása, amikor a többnyire tapasztalati úton meghatározott struktúrájú és méretű háló változói egy tanulási algoritmus alkalmazásával értékeket kapnak. Ezzel a hálózat felveszi a betanított mintában lévő tulajdonságokat, annak információ tartalmát. Ezt hívjuk tanulási szakasznak, ami sok esetben egy számítás- és időigényes folyamat. A második szakaszban a cél a hálózatban tárolt információ kinyerése, többnyire a betanított mintától különböző adatokon. Ezt hívjuk előhívási szakasznak. A két szakasz nem minden esetben különül el egymástól, mert vannak olyan módszerek, amik az előhívási szakaszban is tanulnak, vagyis módosítják a hálózat tulajdonságait. Jellemzően ezeknél a módszereknél is van egy hosszabb tanulási fázis.

## 2.2. Felépítés, topológia

A neuron egy MISO eszköz, vagyis több bemenettel és egy kimenettel rendelkezik. Az elvi felépítése az 1. ábrán látható.




1. ábra: A neuron általános, elvi felépítése
*Forrás: http://mialmanach.mit.bme.hu/neuralis/ch01s02 [4]*

A be és kimenetek között rendszerint valamilyen nemlineáris leképezés valósul meg. Léteznek lokális memóriával rendelkező neuronok, de a memória nélküli neuronok alkalmazása a legelterjedtebb. A memória be- és kimeneti értékeket vagy a neuron előéletére vonatkozó információkat tárolhat. A neuront más néven említik feldolgozó elemnek és csomópontnak is. A bemenetek (és, ha vannak tárolt értékek, akkor azok) alapján a kimenetet egy nagyobbrészt nemlineáris függvény hozza étre. Ezt nevezzük aktiváló vagy aktivációs függvénynek.

Egy neuronnak lehetnek olyan bemenetei, amik működés közben változnak (ilyenek lehetnek a hálózat bemeneti jelei, vagy az előző réteg neuronjainak kimeneti jelei) és lehetnek konstans értékű bemenetei. Az 1. ábrán látható aktiváló bemenet a neuron ütemezését végzi diszkrét esetben.

A neuron működését leíró általános matematikai forma diszkrét idejű hálózatra a következő.

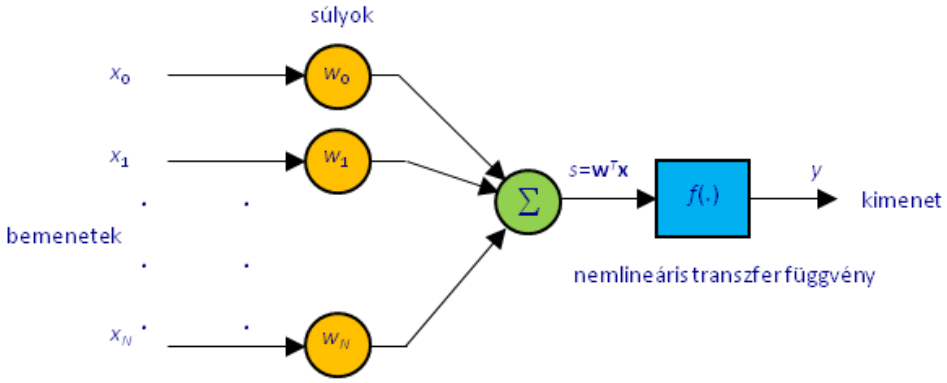
$$y(k) = f\big(\boldsymbol{x}(k), \boldsymbol{x}(k-1), \dots, \boldsymbol{x}(k-M), y(k-1), \dots, y(k-L)\big) \quad (1)$$

A képlet a neuron válaszát adja meg k időlépésnél **x** bementek esetén, ahol

$$\boldsymbol{x}(k) = [x_0(k), x_1(k), \dots, x_N(k)\,]^T \quad (2)$$

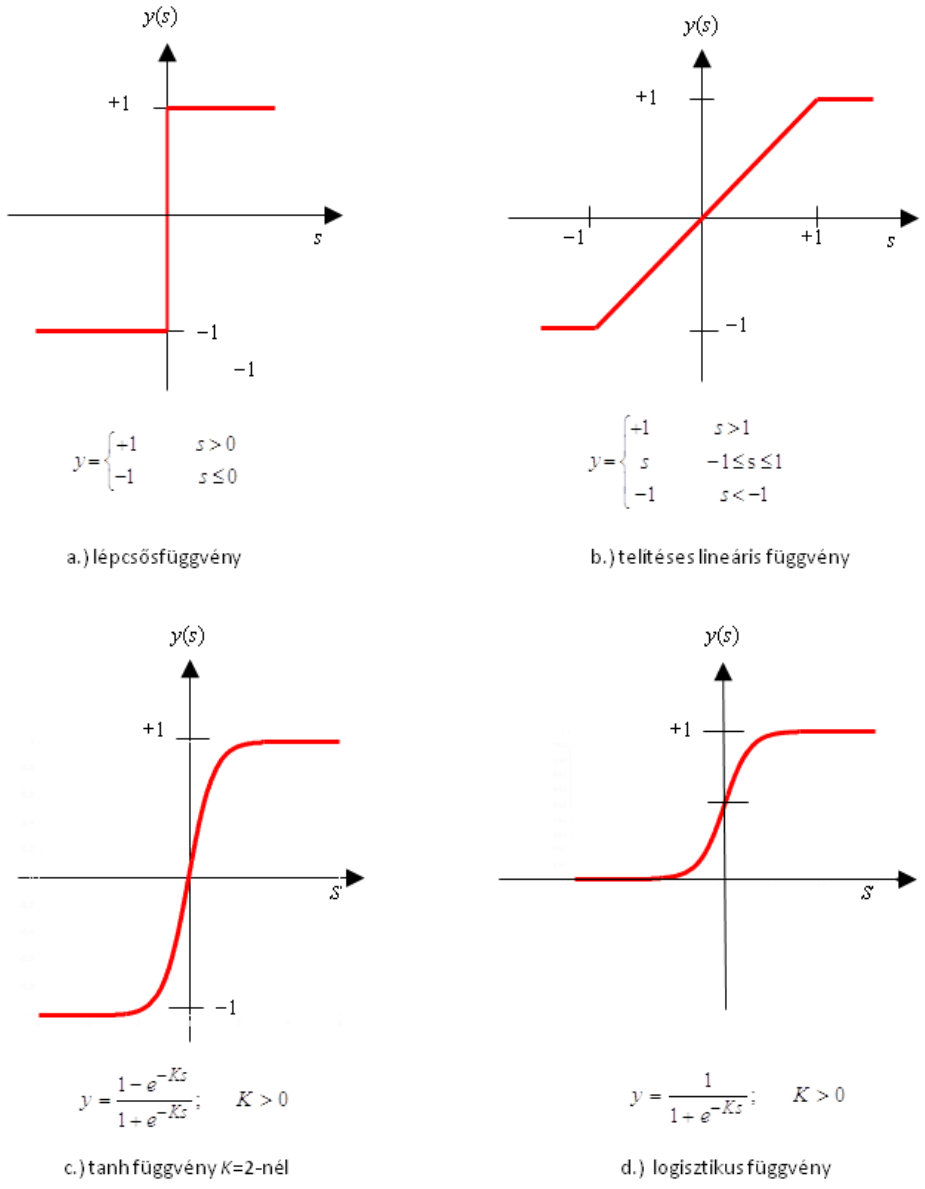
Az **x** vektor tartalmazza az adott k időlépés szerinti bemeneteket. A képlet szerint a neuron egy f függvény szerinti leképezést valósít meg.

A következő ábra szemlélteti a legelterjedtebb és legegyszerűbb változatát egy műveleti elemnek.




2. ábra: Egyenrangú bemenetekkel rendelkező memória nélküli neuron
*Forrás: http://mialmanach.mit.bme.hu/neuralis/ch01s02 [4]*

A képen látható neuron bemenetei $x_i$ bemenetek, amelyek $w_i$ súlyokat kapnak. A súlyozott bemenetek összegzésre kerülnek, vagyis az összegzés után a bemenetek lineáris kombinációját kapjuk. A neuron kimenete úgy áll elő, hogy az összeget egy nemlineáris aktivációs függvénybe vezetjük. A legelterjedtebb ilyen nemlinearitások a következők:




3. ábra: Aktivációs függvények
*Forrás: http://mialmanach.mit.bme.hu/neuralis/ch01s02 [4]*

Az (a) az ugrásfüggvény, a (b) a telítéses lineáris függvény, a (c) és a (d) folytonos, monoton növekvő, telítődő jellegű függvényeket szigmoid függvényeknek nevezzük, és két leggyakoribb formájuk látható az ábrán. Vannak olyan esetek, amikor az aktivációs függvény elmarad. Ilyen neuronok lehetnek a hálózat kimenetén, vagy speciális hálózatoknál, ahol a speciális tulajdonságot a súlyok kialakításánál, egyedi tanulási eljárással kapja meg a hálózat. Vannak olyan memória nélküli neuronok, amik esetében

az összegzés elmarad, mert a bementei közvetlenül a több bemenetű aktivációs függvénybe mennek.

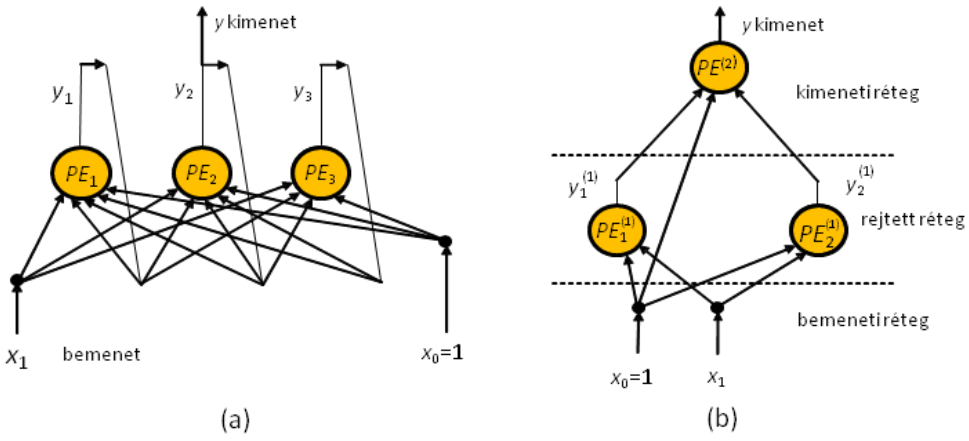
A következőkben a neurális hálózatok topológiájáról lesz szó, amin az egyes neuronok összeköttetési rendszerét, ki és bemeneti helyeit értjük. A hálózati topológiát irányított gráffal szemléltetjük.

Elhelyezkedés, funkció és a többi neuronnal való kapcsolat szerint a hálózatban szereplő neuronokat három diszjunkt halmazba oszthatjuk:

- **bementi neuronok**, amiknek közvetlen bemenete a hálózat bemenete, egybemenetűek, jellemzően egy kimenettel rendelkeznek, más neuronok meghajtására szolgálnak
- **kimeneti neuronok**, amik a hálózat kimenetére továbbítják az információt
- **rejtett neuronok**, amik be és kimeneteikkel közvetlenül más neuronokhoz kapcsolódnak

Ezek alapján az egyes típusú neuronokat bemeneti, rejtett és kimeneti rétegbe csoportosítva említjük. Egy neurális hálózat minimum két réteggel kell, hogy rendelkezzen, a be és kimeneti réteggel. A két réteg között szabadon választott számú rejtett réteg helyezkedhet el. A legtöbb esetben egy adott feladathoz alkalmazandó hálózat topológiájának megválasztására nincs egzakt meghatározás, hanem tapasztalati úton történik.

Összeköttetési rendszer szempontjából megkülönböztetünk előrecsatolt és visszacsatolt hálózatokat. Visszacsatolt hálózatról csak akkor beszélünk, ha az irányított gráf tartalmaz hurkot, egyébként a típusa előrecsatolt. A visszacsatolásnak több módja is ismert, amiket terjedelmi okokból nem részletezek.




4. ábra: Visszacsatolt (a) és előrecsatolt (b) hálózati topológia
*Forrás: http://mialmanach.mit.bme.hu/neuralis/ch01s02 [4]*

## 2.3. Tanulási fázis

Az évek során - az egyre nagyobb számítási képességű számítógépek megjelenésével - folyamatosan teret nyernek a neurális hálózatok.

A tanulási fázis igen számításigényes folyamat. Egy neurális hálózat tanítása jellemzően nagy mennyiségű adatból kinyert minták alapján történik. Ahogy a rendszer rendszerint felveszi a mintákban lévő ismereteket és a viselkedése módosul.

Több eljárást dolgoztak ki a neurális hálózatok tanítására. Jellemző módszer, amikor a hálózat tanítását egy nagy adathalmaz készítése előzi meg. Ilyenkor a rendszer kívánt működéséhez tartozó összetartozó mintapárokat, - amik a rendszer bemeneteit tartalmazzák - a kívánt válaszok ismeretében elő kell állítani. A tanítás során az előállított adathalmaz mintáiban lévő ismereteket akarjuk kinyerni, és azokkal a rendszer viselkedését módosítani. A feladatom során ezt a módszert alkalmaztam, ami a negyedik fejezetben kerül majd kifejtésre.

A tanulás nem csak összetartozó ki- és bemeneti mintapárok tanításával lehetséges. Létezik olyan módszer, amikor csak a bemeneteket ismerjük, és a tanulás során valamilyen szabályosságakat, hasonlóságokat vagy különbözőségeket kell felismerni. Van olyan eset is, amikor ismerjük a bemeneteket, és el tudjuk dönteni, hogy azokra a rendszer jó vagy rossz választ adott-e, és ennek alapján lehet módosítani a hálózat tulajdonságait. Az egyes módszerekre több elnevezés is használatos. A neurális hálózatokra jellemző három fő tanulási forma a következő:

- felügyelt vagy ellenőrzött tanulás
- felügyelet nélküli vagy nemellenőrzött tanulás
- analitikus tanulás

A tanulási formáknál meg kell említeni a félig ellenőrzött tanulást, amikor a felhasználható adathalmaznak csak egy része biztosítja az ellenőrzött tanulás lehetőségét, viszont a rendszer további tanításához más módszert alkalmaznak. Ilyenkor a már rendelkezésre álló mintapárok egy alapot biztosítanak a további fejlődéshez.

A három tanulási forma közül a dolgozatban is többször alkalmazott felügyelt tanulás módszeréről írok bővebben.

### 2.3.1. Felügyelt tanulás

A felügyelt - vagy más néven ellenőrzött - tanulás esetében az összetartozó ki- és bemeneti mintapárok rendelkezésre állnak, a tanítás ezen adathalmaz alapján történik. A tanító algoritmus feladata, hogy megtanítsa a mintapárok által reprezentált leképezést. A kívánt válaszok ismertek, tehát a hálózat közvetlen válasza összehasonlítható azokkal, és a különbségből kiderül, hogy szükséges-e további módosítás. A tanító eljárás lényege a kívánt és tényleges kimenetek közti hiba minimalizálása.

Nem minden esetben áll rendelkezésre a hálózat bementére adott pontos, részletes válasz. Jó néhány esetben csak annyi visszajelzést kapunk egy adott bemenetre történő válaszra, hogy az helyes vagy helytelen, vagyis csak bitnyi információt. Ezen információ birtokában kell eldöntenie a tanító algoritmusnak, hogy szükséges-e és, ha igen, milyen irányba kell módosítani a hálózat paramétereit. Ezt az eljárást megerősítéses tanulásnak hívjuk. A környezet kiszámíthatatlan viselkedése miatt elképzelhető olyan eset is, amikor nem tudjuk pontosan megmondani, hogy a válasz helyes vagy helytelen, hanem csak a megerősítés pontosságát tudjuk megbecsülni.

## 2.4. Előhívási fázis

A betanított hálózat, a neki szánt funkció betöltése során az előhívási szakaszban van. Ez alapvetően egy gyors folyamat, ami során a hálózat bementére adott bemenő jel alapján, a tanítás során kalkulált súlyokat felhasználva a kimeneten megjelenik a válasz. Előfordul, hogy ez a szakasz összeolvad a tanulási fázissal. Ilyen lehet például a

megerősítéses tanulás, ami során a rendszer bemenetére adott válasz jósága alapján kerülnek módosításra a paraméterek.

## 2.5. Konvolúciós neurális hálózat

A konvolúciós neurális hálózat hagyományos neurális hálózat alapokon nyugszik. Annak egy speciális fajtája, amit képeken található minták feltárására fejlesztettek. Neuronokból épülnek fel és hasonlóképpen rendelkeznek tanítható súlyokkal. A bemeneteiken kapott értékek skaláris szorzatát egy nem lineáris leképezés követheti. Az egész hálózat reprezentálhat egy egyszerű osztályozó eljárást, aminek a bemenetei a nyers képpontok, és a kimenete lehet egy adott kép osztályba tartozás valószínűsége vagy, ha regresszió típusú hálózatokban gondolkodunk, akkor akár a képen megjelenő jármű úton tartásához szükséges kormányszög.

A hagyományos neurális hálózatok nem skálázzák a teljes képet. Ha példaként vesszük egy 32 pixel széles, 32 pixel magas színes képet és ezt bevezetnénk egy teljesen összekapcsolt rejtett rétegbe, akkor az neurononként 32x32x3 = 3072 súly paramétert jelentene. Ez még kezelhetőnek tűnik, de tisztán látszik, hogy a teljesen összekapcsolt struktúra nem kezeli jól a képeket. Ha példaként megnézünk egy 200x200 méretű színes képet, akkor neurononként 120 000 súly paramétert kapunk. Ha végig gondoljuk, hogy az hány paramétert jelent még egy kis neuronszámú kevés rejtett rétegből álló hálózat esetében, akkor rájöhetünk, hogy nagyobb méretű képen fellelhető minták felismeréséhez a tisztán hagyományos neurális hálózatok alkalmazása nem célravezető, emiatt a kép elő feldolgozására, szűrésére konvolúciós rétegeket vezetnek be.

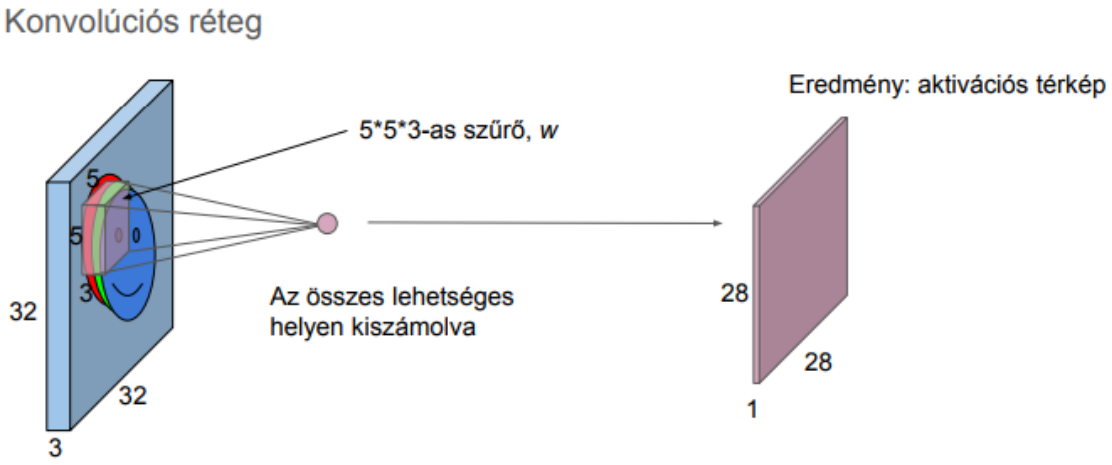
### 2.5.1. Felépítés

A konvolúciós neurális hálózatokban szereplő rétegeket funkció alapján alapvetően két csoportra lehet bontani. Az elsőbe sorolhatók a konvolúciós rétegek, amik - mint paraméterezhető szűrők - előfeldolgozást végeznek a képen. Ezáltal a kép mérete lecsökken, a hordozott információtartalom kiemeltté válik, és a hagyományos neurális hálózat (második csoport) számára feldolgozható lesz, az osztályozást el tudja végezni. Egy konvolúciós neurális hálózat tehát konvolúciós és hagyományos rejtett rétegekből épül fel.

A konvolúciós rétegek alapjában véve színes képek feldolgozására lettek kifejlesztve, ezért a neuronjaik három dimenzióban (szélesség, magasság, színcsatornák) vannak elrendezve.

A következő felsorolásban - a tipikus, de nem kizárólagos sorrendre ügyelve - bemutatnám a konvolúciós hálózat rétegtípusait:

- **bemeneti réteg**: Tartalmazza a kép nyers pixel értékeit, jellemzően R, G, B csatorna szerint rendezve.
- **konvolúciós réteg**: Adott képpont csoportokra konvolúció matematikai műveletet alkalmazunk. A konvolúció eredménye egy skalár (a skaláris szorzata a kép egy adott részének és a szűrőnek). Ha minden képpont csoportra elvégezzük a konvolúciót, akkor egy aktivációs térképet kapunk. Általában több szűrő paraméterrel is elvégezzük a konvolúciót és aktivációs térképeket egymásra rakjuk. Az aktivációs térkép, a művelet tulajdonsága miatt, kisebb méretű lesz.

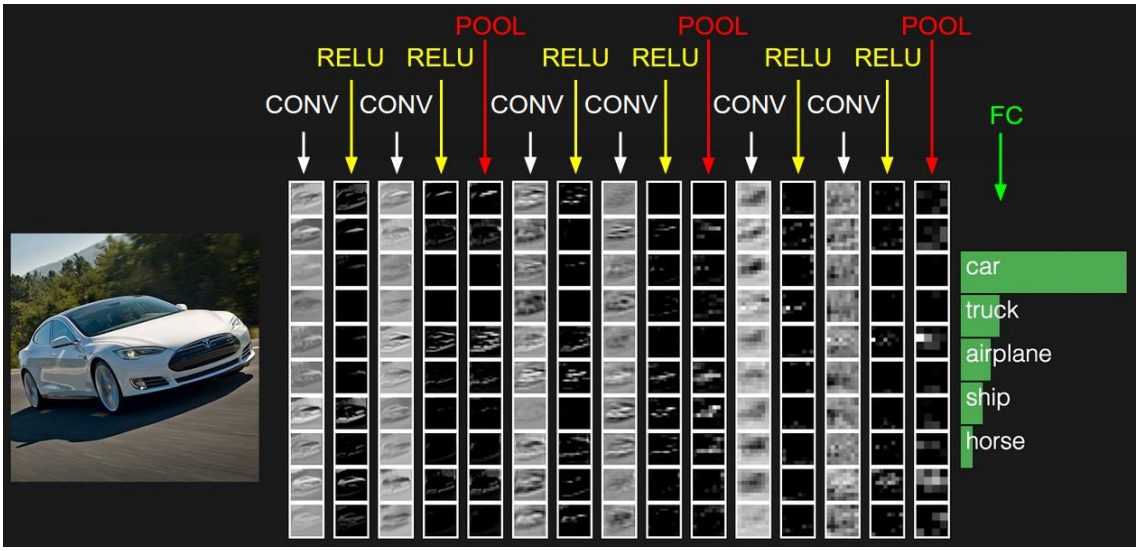



5. ábra: Konvolúció eredménye.
*Forrás: http://home.mit.bme.hu/~engedy/NN/NN-CNN.pdf [5]*

- **RELU réteg**: A konvolúciós réteg aktivációs függvénye, ami a következő leképezést valósítja meg: $f(x) = max(x, 0)$. Vagyis, ha a bemenet kisebb, mint nulla, akkor a kimenet nulla lesz, ha nagyobb, mint nulla, akkor a kimenet a bemenet értékét veszi fel.

- **összevonó réteg (pooling layer)**: Az aktivációs térképekre alkalmazzuk. Csökkenti a reprezentáció méretét, így kezelhetőbbé téve azt. Leskálázza a képet.
- **teljesen összekötött réteg (fully connected, FC)**: Utolsó rétegekként szokták használni, hagyományos réteg.

A következő ábra egy konvolúciós feldolgozást mutat be, ahol szemléltetve vannak az egyes rétegek utáni állapotok. A képen látható, hogy ez egyes konvolúció utáni állapotok tulajdonság érzékelőként viselkednek.




6. ábra: Konvolúció működése.
*Forrás: http://cs231n.github.io/convolutional-networks/ [6]*

A konvolúciós hálózatok betanításához nagyszámú mintára és nagyteljesítményű számítógépekre van szükség. Amennyiben nem áll rendelkezésünkre nagyjából egymillió képkockából álló tanító adathalmaz, akkor kisszámú mintáról beszélünk. Ha nincs lehetőségünk vagy erőforrásunk nagy adathalmazzal tanítani, akkor is megvalósíthatjuk a kívánt leképezést. A *transfer learning* (tanulás átadása) módszer segítségével egy előre betanított hálózatot veszünk alapul, és annak utolsó pár rétegét lecserélve végezzük a tanítást. Ilyenkor a megmaradt rétegek betanított tulajdonság érzékelő funkciója segítségével a lecserélt rétegek által megvalósítandó leképezés könnyebbé vállhat. A konvolúciós neurális hálózatok tanítása esetében ez egy gyakori eljárás. A weben számos előre betanított hálózat található.

# 3. FELÜGYELT TANULÓ RENDSZER KÍSÉRLETI FEJLESZTÉSE

A TDK munkám célja egy sávtartó funkció megvalósítása (HiL) Hardware-in-the-Loop környezetben, amit az elmúlt években újra fellendült és népszerű gépi tanulás eszköztárának segítségével valósítottam meg. A rendszer kifejlesztését megelőző kutatómunka összefoglalása után rátérek a tényleges megvalósítási kérdésekre. A fejezet a fejlesztési és tesztkörnyezetet bemutatása után kitér a munka során felmerülő feladatokra, és azok megoldásaira, valamint értékeli az elért eredményeket.

A sávtartó funkciót hagyományos többrétegű neurális hálózatra és konvolúciós neurális hálózatra alapozva is kifejlesztettem. Az egyes megoldásokat az egységes HiL környezet ismertetése után külön fejezetben mutatom be.

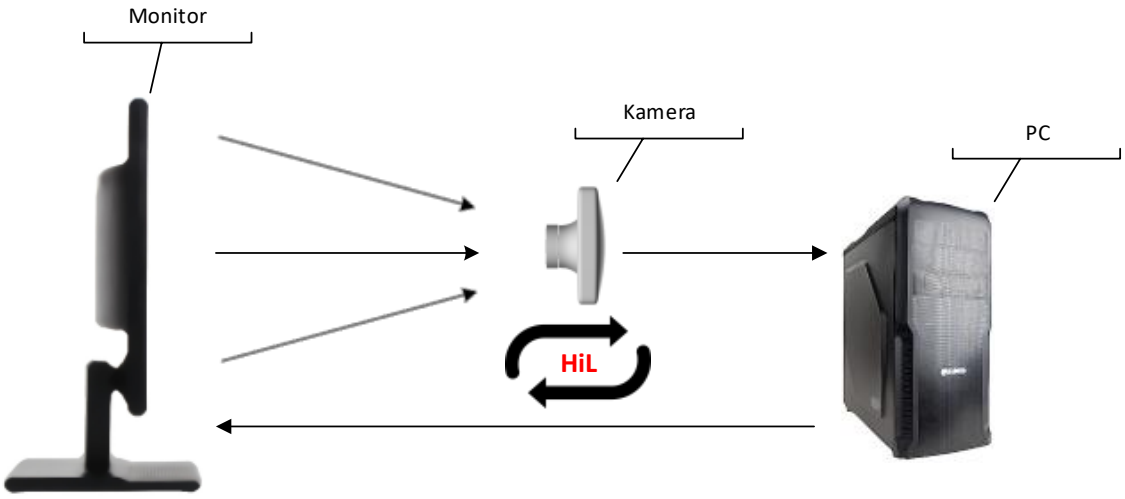
## 3.1. A HiL környezet bemutatása

A járműipari szereplők egy-egy termék fejlesztése során és jellemzően a piacra dobás előtt, azt hosszadalmas, átfogó és költséges teszt procedúrának vetik alá. A laboratóriumi fejlesztések után valós körülmények között is fontos a megfelelő működés. Gyakran egy-egy eszköz vagy funkció fejlesztése során a tesztautók nem ritkán több százezer kilométert futnak, de vannak olyan funkciók, amelyeknél ennyi sem elég. Ha példaként veszünk egy hagyományos kamera szenzort, aminek a feladata a sáv, objektum és tábla felismerés, és figyelembe vesszük azt, hogy egy autót hány országban használhatják, akkor könnyen belátható, hogy csak egy ország határain belül is jónéhány kilométer megtétele szükséges.

Az összetett, valósidejű rendszerek során gyakran alkalmazott eljárás az úgynevezett Hardware-in-the-Loop (HiL) tesztelési technika. A „hardver a hurokban” tesztek effektívebb hibaszűrést tesznek lehetővé, amivel csökkennek a fejlesztési költségek és a termék piacra dobásának ideje. A módszer lényege, hogy a teszt során a vizsgált rendszerelem irányítási szempontból a valós körülményekhez hasonló környezetben legyen. Vagyis esetünkben az érzékelőket hasonló fizikai hatás érje, mint a tényleges környezetükben, amely hatást (az irányítandó rendszer dinamikájával együtt) egy szimulációs környezet állít elő. Az eljárás előnye, hogy a rendszer teszteléséhez olyan virtuális környezet létrehozása válik lehetővé, ami vizsgálhatóvá teszi a rendszer többi elemével való kölcsönhatást, ha azok fizikailag nem állnak rendelkezésre. Ezeknél a

teszteknél egy pontos és valós idejű modellje kell a környezetnek, aminek szolgáltatnia kell a bemeneteket az eszköz felé, és reagálnia kell annak kimeneteire.

Munkám során az alábbi ábrán látható HiL rendszert valósítottam meg.



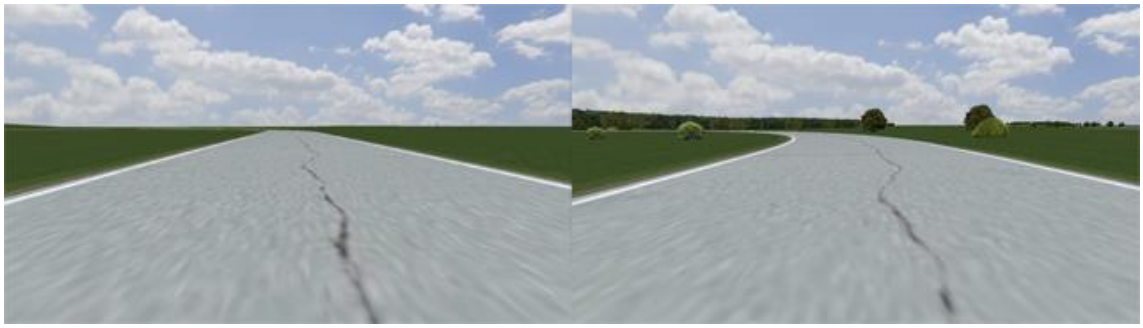

7. ábra: HiL hurok.

A hurokban a hardverlem szerepét egy kamera tölti be, a környezet és a jármű dinamikai modelljét pedig egy CarSim elnevezésű PC szoftver hozza létre, és jeleníti meg. A monitoron megjelenő képet a kamera detektálja, és küldi a számítógép felé. A detektált kép alapján a szoftver valós időben módosítja a környezetet.

### 3.1.1. CarSim szoftver

A fejlesztés során CarSim járműdinamikai szimulációs szoftverrel dolgoztam, ami a HiL hurkon belül jelentős szerepet tölt be. Valós időben futva, egy személygépjármű szemszögéből jelenít meg egy körpályát, amin egy járművet irányító parancsok küldésével körbe lehet vezetni. A pálya vonalvezetése állandó, de a körülötte lévő környezet alakítható. Ki és be lehet kapcsolni a fák, bokrok, egyéb növények megjelenítését, valamint az aszfalt, a fű és a vonalak színe is megadható.

Az alábbi ábra mutatja a program által létrehozott és megjelenített pályát két különböző összeállításban. Bal oldalt látható egy tereptárgyak nélküli, jobb oldalon pedig egy fákkal, bokrokkal kiegészített pálya.

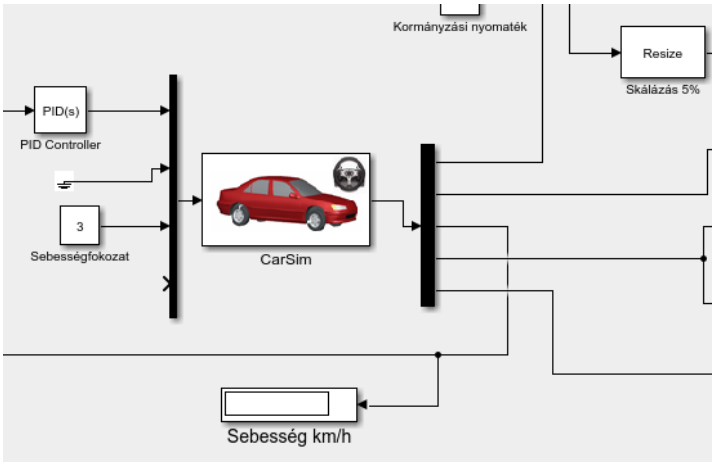


8. ábra: CarSim pálya.

Jelen munkám során a szoftvert három fő részre osztanám. A program indulásakor a projekt betöltése után kapunk egy mindenre kiterjedő több almenüből és fülből álló beállítások ablakot, ahol többek közt meg lehet adni a pálya tulajdonságait, járművet lehet paraméterezni és be lehet állítani, hogy a szoftver milyen nézőpontból mutassa a monitoron megjelenített képet. Itt lehet elindítani a szimulációt és ki lehet választani, hogy a jármű irányítása miként történjen. A CarSim lehetőséget biztosít beágyazott program létrehozására, amivel a jármű, vagy adott esetben több jármű irányítása történhet. Támogatja a Matlab/SimuLink alapú algoritmusok futtatását. Biztosít egy olyan Simulink környezetet, amiben valós időben futtatva rendelkezésünkre áll a pályáról és a járműről szinte minden információ, és lehetőséget biztosít a jármű irányítására.

A Simulink egy Matlab szoftverbe integrált blokk diagram környezet, ami támogatja a modell alapú tervezést, szimulációk futtatását, automatikus kód generálást és a valós idejű futtatást.

A Simulink programban kapunk egy blokkot, ami egy CarSim interfészként működik. A baloldalon irányító parancsokat küldhetünk a szoftverbe, a jobb oldalon pedig a járműről, vagy a jármű és a környezet viszonyáról kaphatunk valós idejű adatokat. A be- és kimeneti jelek szabadon konfigurálhatók.




9. ábra: CarSim-ba ágyazott Simulink algoritmus.

A program harmadik fő része egy megjelenítő ablak, ami - kommunikálva a Simulink beépülő programmal - felelős a jármű és a környezet megjelenítéséért.

A pályán vezethető jármű dinamikai viselkedését egy a CarSim szoftverben megvalósított valós járműmodell írja le. A jármű vezérlő jelei a nyomaték- és kormányszög-parancsok.

A TDK munkám számottevő részét a CarSim szoftverbe beépülő Simulink programban valósítottam meg a következő fejezetekben leírtak szerint.

## 3.2. Többrétegű neurális hálózat alapú rendszer

A CarSim szoftver biztosít egy környezet, ahol egy pályán haladó járművet egy Simulink program segítségével lehet irányítani. A jármű előtt lévő pályát megjeleníti egy monitoron, amit egy kamera figyel. A kamera képi adatai valós időben beérkeznek a Simulink programba.

A feladat tehát a következő: egy olyan módszer, algoritmus készítése Simulink környezetben, ami valós időben a kamera képi adatait felhasználva meghatározza a jármű irányításához szükséges parancsokat. Ezt két külön módszerrel oldottam meg, az első egy többrétegű neurális hálózaton alapul, ami ebben a fejezetben kerül bemutatásra.

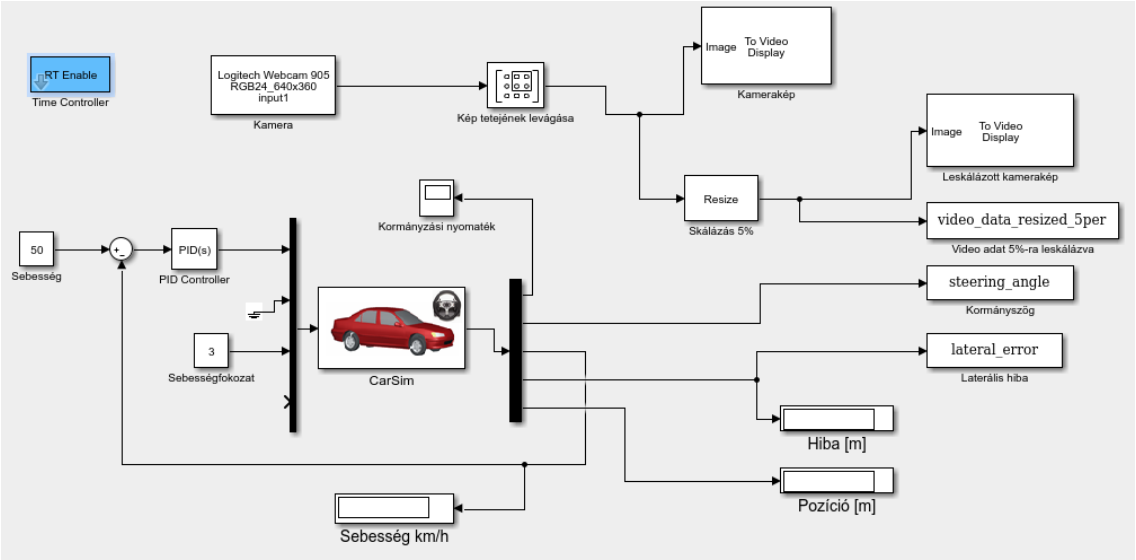
A munka a neurális hálózatok jellegét ismerve több részből állt. Az első fázisban elő kellett állítani a hálózat tanításához szükséges mintapont párokat. Ezután következett a hálózat paramétereinek megválasztása, amit a hálózat betanítása követett. Végül a betanított hálózat előhívása következett.

### 3.2.1. Tanító mintapontok előállítása

Az irányító feladatot megvalósító neurális hálózat betanítása felügyelt tanulás módszerrel történik. A minták alapján történő tanulás lényege, hogy az eljárás során a be és kimeneti mintapárokból igyekszünk megfelelő ismereteket kinyerni és ezzel a rendszer viselkedését módosítani. A hálózat feladata, hogy megtanulja a rendelkezésre álló mintapont párok által reprezentált bemenet-kimenet leképezést. Ehhez elő kell állítani a megfelelő adathalmazt.

A hálózat bemeneti adatai a kamera által szolgáltatott képpontok, a válasz pedig egy kormányszög érték. A tanító adatstruktúra létrehozásához a járművet minél pontosabban végig kell vezetni a pályán, és közben el kell tárolni az adott kormányszög értékekhez tartozó képeket.

A tanító adathalmaz elkészítéséhez az alábbi ábrán látható Simulink algoritmust készítettem, ami a CarSim belső longitudinális szabályzóját felhasználva vezeti a pályán a járművet, miközben egy általam készített laterális szabályzó algoritmus fix, 50 km/h sebességgel hajtja azt. A képek és a hozzájuk tartozó kormányszög értékek mellett a pálya középvonalához viszonyított hiba, tized másodpercenként eltárolásra kerül.




10. ábra: Tanító adathalmaz generálása

A többrétegű neurális hálózaton alapuló rendszer egyik fő kérdésköre, hogy a hálózat bemenetére szánt kép milyen formátumú és mekkora méretű legyen. Főként ez és az

első rejtett réteg neuronszáma határozza meg, hogy a bemeneti réteg és az első rejtett réteg között hány paraméter lesz.

A kamera 640 x 360 pixeles színes képet szolgáltat, ami 16:9-es méretaránynak felel meg. Azért ezt a felbontást választottam, mert a megjelenítőnek beállított monitor méretaránya szintén 16:9-es, így a monitor elé helyezett kamera képe tökéletesen illeszkedik a monitorra. Ahogy a 8. ábrán jól látszik, hogy az ég a kép felső részét elfoglalja, ami a jármű irányítása szempontjából lényegtelen információ. A kép felső 100 képpontja levágásra került, így kaptam egy 640 x 260 pixel méretű képet, ami 230400 képpont. Egy hagyományos neurális hálózat bemenetére ez túl nagy méret, főként, ha figyelembe vesszük, hogy az egy színes kép, tehát 691200 bemeneti paramétert jelentene. A kép szürkeárnyalatossá konvertálása és leskálázása mellett döntöttem. A leskálázás mértékét tapasztalati úton 5 %-nak választottam, így a hálózat bemenetére egy 13 x 32 pixel méretű szürkeárnyalatos kép került, ami egyszerre 416 bemeneti paramétert jelent.

A tanító mintapontok előállítására készült Simulink szoftverrel négyféle adatkészletet hoztam létre. Az elsőt a tereptárgyak nélküli, a másodikat pedig a fákkal, bokrokkal teli pályán végighaladva, és ezt a kettőt megismételve a másik irányban. Egy ötödik adatkészlet pedig a négy felvett adatsorozat egymásba fűzésével és összekeverésével jött létre, ez lesz a tanító adatkészlet.

### 3.2.2. Hálózat betanítása

A tanító mintapontok előállítása CarSim, Simulink környezetben, a 4.2.1 fejezetben leírtaknak megfelelően történt. A tanító adathalmaz generáló szimuláció végeztével az adatok egy mentés művelet után Matlab környezetből elérhetőek lettek. A tanítás megkezdéséhez azokat, a hálózatnak megfelelő adatformátumba kellett alakítani.

Egy kétdimenziós tömbben (kormányszög x időlépés) szerepelnek a szimuláció kezdetétől tízed másodpercenként eltárolt kormányszög értékek, és egy három dimenziós tömbben (magasság x szélesség x időlépés) a hozzájuk tartozó képkockák.

A tanító algoritmus számára a képkocákat tartalmazó tömböt is kétdimenzióssá kellett alakítani. A feladat a mátrix elrendezésű képpontok tömb elrendezésűvé alakítása volt. A képpontok soronkénti egymáshoz fűzésével előállításra kerültek az adott időlépéshez tartozó képpontokat tartalmazó tömbök (képpontok x időlépés). Egy egyszerű méret-

összehasonlítási és -hibaellenőrzési algoritmus után előálltak a hálózat ki- és bemeneti adatai.

A kiválasztott méretű háló tanításának megkezdéséhez több beállítás megadása szükséges. Az első és legfontosabb a tanító algoritmus kiválasztása. Nagyon nehéz előre megjósolni, hogy melyik algoritmus lesz a leggyorsabb, és melyik adja a legáltalánosabb eredményt egy adott problémára. Amelyik algoritmus előbb eléri a kitűzött hibacélt, az jobban teljesít. Ez számos tényezőtől függhet: a tanító adathalmaz méretétől, a súlyok és konstansok méretétől, a hiba céltól, és hogy a hálózatot mintafelismerésre vagy függvény illesztésre akarjuk-e használni.

A Matlab az alábbi tanító algoritmusokat kínálja fel. Zárójelben a módszer rövidítése és kötőjellel elválasztva a magyar neve.

- Levenberg-Marquardt (LM)
- BFGS Quasi-Newton (BFG)
- Resilient Backpropagation (RP) – Rugalmas hiba-visszaterjesztés
- Scaled Conjugate Gradient (SCG) - Skálázott konjugált grandiens
- Conjugate Gradient with Powell/Beale Restarts (CBG) – Konjugált gradiens Powell/Beale újraindításokkal
- Fletcher-Powell Conjugate Gradient (CGF) - Fletcher-Powel konjugált gradiens
- Polak-Ribiére Conjugate Gradient (CGP) - Polak-Ribiére konjugált gradiens
- One Step Secant (OSS) - Egy lépéses metsző
- Variable Learning Rate Backpropagation (GDX) - Változtatható tanulási arányú hiba-visszaterjesztés

A Matlab tanító algoritmusokhoz tartozó dokumentációja, különböző adathalmazokon alkalmazva összeszedi azok tanítási idejét, ami alapján el lehet dönteni, hogy egy adott feladatra melyiket érdemes választani. Vannak regresszió és osztályozás típusú adatkészletek. Az egyik leginkább hasonlító adatkészlet, amiből ki lehet indulni, a 21 emberi szervezetben mért jellemzőhöz tartozó koleszterin értékeket tartalmazza. Választásom azért esett erre, mert ennél a hálózatnál viszonylag nagyszámú (21) paraméter van a bemeneten, és regresszió típusú az adatkészlet. Egy három rejtett rétegből álló; 21,15, 3 neuront tartalmazó hálózatot betanították az összes módszerrel. A

tanítás minden esetben addig tartott, amíg a négyzetes hiba kisebb nem lett, mint 0.027. Módszerenként 20 tanítást végeztek el.

A következő tanítási időket kapták:

| Algoritmus | Átlag [s] | Szorzó | Min. idő [s] | Max idő [s] | Eltérés s] |
|---|---|---|---|---|---|
| SCG | 99.73 | 1.00 | 83.10 | 113.40 | 9.93 |
| CGP | 121.54 | 1.22 | 101.76 | 162.49 | 16.34 |
| CGB | 124.06 | 1.2 | 107.64 | 146.90 | 14.62 |
| CGF | 136.04 | 1.36 | 106.46 | 167.28 | 17.67 |
| LM | 261.50 | 2.62 | 103.52 | 398.45 | 102.06 |
| OSS | 268.55 | 2.69 | 197.84 | 372.99 | 56.79 |
| BFG | 550.92 | 5.52 | 471.61 | 676.39 | 46.59 |
| RP | 1519.00 | 15.23 | 581.17 | 2256.10 | 557.34 |
| GDX | 3169.50 | 31.78 | 2514.90 | 4168.20 | 610.52 |

1. táblázat: Koleszterin-adatkészlet tanítási idők.
*Forrás: https://www.mathworks.com/help/nnet/ug/choose-a-multilayer-neural-network-training-function.html [7]*

A táblázat megmutatja, hogy a skálázott konjugált gradiens módszer a leghatékonyabb, átlagosan 99.73 másodperc alatt érte el a kitűzött hiba célt.

A leghatékonyabb algoritmus keresését a saját adatkészleten is elvégeztem. A tanítást minden esetre lefuttattam egy 5 rejtett rétegből álló hálózatra, amikbe sorban 50, 30, 20, 10, és 5 neuron került. A tanítást CPU-n hajtottam végre. A tanító adatkészleten 0.2 négyzetes hiba elérése volt a cél, és a tereptárgyakkal ellátott és a tereptárgyak nélküli adatkészlet összeolvasztásával végeztem a tanítást. Az alábbi eredményeket kaptam:

| Algoritmus | Futási idő [s] | Validációs négyzetes hiba | Jóság |
|---|---|---|---|
| SCG | 25 | 0,9268 | 21,5 |
| CGB | 30 | 1,3826 | 57,3 |
| CGP | 54 | 1,3519 | 98,7 |
| CGF | 80 | 1,1994 | 115,1 |
| RP | 5 | 9,4728 | 448,7 |
| OSS | 273 | 1,8914 | 976,6 |
| LM | 1284 | 1,6982 | 3702,9 |
| BFG | 1021 | 6,7337 | 46294,9 |
| GDX | 3600+ | - | - |

2. táblázat: Tanító algoritmusok teljesítése

Az egyes algoritmusokkal való futtatás eredményeképpen feljegyeztem a futási időt és a tanító adatkészleten 0.2 négyzetes hiba után elért validációs adatkészleten lévő hibát.

A futási idő azt mutatja, hogy az adott adatkészletből kinyert információtartalmat az adott méretű hálózatra milyen gyorsan sikerül alkalmazni, a validációs négyzetes hiba nagysága pedig, a betanított hálózat általánosító-képességéről ad információt.

Bevezettem egy jósági számot, amely összeszorozza, négyzetre emelt validációs négyzetes hibát a futási idővel. A négyzetre emeléssel elkerülhető, hogy az irreálisan gyorsan, de viszonylag kis hibát adó eredmény jobb értéket kapjon. Ez alapján felállítható egy sorrend a módszerek jóságára vonatkozóan, ami alapján kijelenthető, hogy a legjobb eredményeket a skálázott konjugált gradiens módszerrel értem el. A további tanításokhoz ezt a módszert alkalmaztam.

A tanító módszeren kívül a tanítás megkezdéséhez még két további beállítást kell megadni. Az első az adat szétosztási algoritmus, a második a teljesítménymutató kalkulációs módszer.

A neurális hálózatok tanításánál alapvetően három adatkészletet különböztetünk meg: a tanító-, validációs- és tesztkészletet. A tanító adatokkal történik a hálózat betanítása, a teszt- és validációskészlet ellenőrzésre szolgál. Jellemző elosztás a 70% tanító, 15% validáció, 15% teszt. Vannak olyan tanító szoftverek, ahol az adatokat a felhasználónak kell készletekre bontani, de a Matlab egy algoritmus segítségével ezt elvégzi helyettünk. Négy féle adatszétbontó algoritmus közül választhatunk:

- dividerand: Random helyekről feltölti a készleteket.
- divideblock: Sorban menve az adatokon feltölti a készleteket.
- divideint: Az előzőt kiegészíti egy funkcióval. Meg lehet adni, hogy az első hány mintapontot vegye figyelembe.
- divideind: Indexekkel lehet megadni a készleteket.

A tanítás során a random adatfeltöltést választottam.

A teljesítménymutató kalkulációnál a négyzeteshiba-számolás általánosan elfogadott módszer, ezért ezt alkalmaztam.

Számos kiegészítő beállítás megadására van még lehetőség. Ezek közül az egyik, a tanító hardver megadása. Alapbeállításként a tanító algoritmusok a CPU-n futnak, de lehetőségünk van GPU-n, vagy parallel módon tanítani.

### 3.2.3. Hálózat méretének megválasztása

A hálózat méretének megválasztása tapasztalati úton történt. Meghatározásra került a rejtett rétegek és az egyes rétegekben lévő neuronok száma.

Az egy rejtett réteg kevés neuronszámától haladva fölfelé kielemeztem, hogy melyik struktúra adja a legjobb illeszkedést és a legáltalánosabb eredményt. Fontos, hogy az illeszkedésen felül a hálózat általánosító-képessége is jó legyen, hogy a változó körülményekre is megfelelően tudjon reagálni. Minden méretválasztási lépést egy tanítás, és egy eredmény-kiértékelés követett.

Mivel nagyszámú (416) bemeneti jel van és egy kimenet, ezért logikusnak tűnik, hogy a több rejtett réteg közül az első sok neuront tartalmazzon, és a rétegszám növekedésével, piramis-szerűen a neuronok száma is csökkenjen.

Az alábbi táblázatba foglalt struktúrákkal próbálkoztam, ahol a [x y z…] jelölés a rejtett rétegek számát, és az abban elhelyezkedő neuronokat jelenti. Például [1 3] jelentése: két rejtett réteg, az első egy neuronból, a második háromból áll.

| Hálózati struktúra | Futási idő [mm:ss] | Validációs négyzetes hiba | Tanulás négyzetes hibája (cél) | Iterációk száma |
|---|---|---|---|---|
| [30 20 5] | 03:37 | 1.40631 | 0.02 | 12632 |
| [50 20 5] | 03:39 | 0.88868 | 0.02 | 9370 |
| [50 30 20 5] | 02:12 | 0.98182 | 0.02 | 5181 |
| [80 50 20 5] | 08:35 | 0.82197 | 0.006 | 11130 |
| [100 50 5] | 09:18 | 0.64459 | 0.006 | 11168 |
| [100 50 30 20 5] | 05:59 | 0.76948 | 0.006 | 6584 |
| [100 50 20 5] | 08:20 | 0.70402 | 0.006 | 8514 |
| [80 40 20 5] | 05:00 | 0.70368 | 0.006 | 5742 |
| [80 30 20 5] | 21:43 | 0.88932 | 0.006 | 27101 |
| [120 50 5] | 07:28 | 0.81488 | 0.006 | 6926 |
| [100 40 20 5] | 07:34 | 0.77396 | 0.006 | 9232 |

3. táblázat: Hálózat méretválasztása.

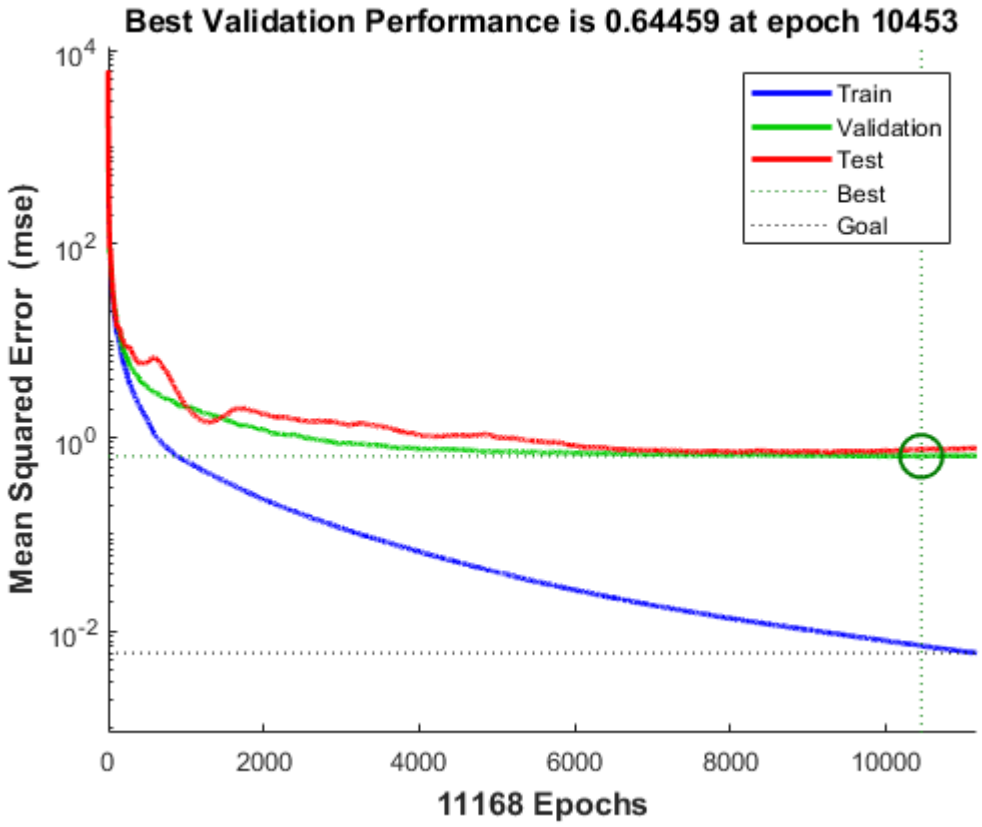
Az első három futtatás teljesítmény diagramjából az látszott, hogy a validációs és teszt adatkészleteken jelentősen különböző hiba értékek jelentkeztek, ami arra enged következtetni, hogy az általánosító képesség gyenge. A negyedik méretválasztásnál már nem jelentkezett ez a probléma, onnantól kezdve a legjobb illeszkedés elérése volt a cél.

A futtatások eredményéből látszik, hogy mind a futási idő, mind a négyzetes hiba tekintetében, a három rejtett rétegből álló hálózat szerepelt a legjobban, így ezt választom a sávtartó funkció megvalósításához.

### 3.2.4. Eredmények kiértékelése

A [100 50 5] struktúrájú hálózat betanítása után a következő diagrammokat készítettem.

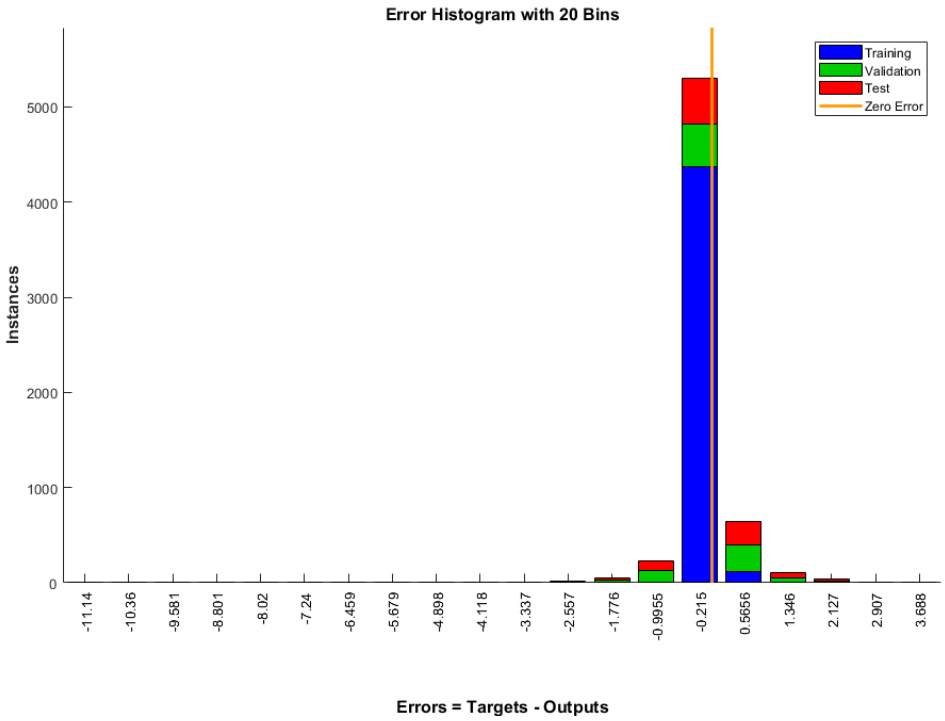
A 11. ábra a tanítás folyamatát mutatja. A vízszintes tengelyen az iteráció, a függőleges tengelyen a négyzetes hiba jelenik meg. A diagramm a tanító, validációs és teszt adatokon vett négyzetes hibát ábrázolja az iterációk függvényében. A legkisebb validációs hiba a diagram tetején szövegesen van feltüntetve. Látható, hogy a tanító adatkészleten vett hiba (kék) jóval alacsonyabb, mint a validációs- és tesztadatokon. Az, hogy a validációs- és tesztadatok egymáshoz simulnak, azt jelzi, hogy a hálózat általánosítóképessége megfelelő.




11. ábra: Teljesítmény görbék.

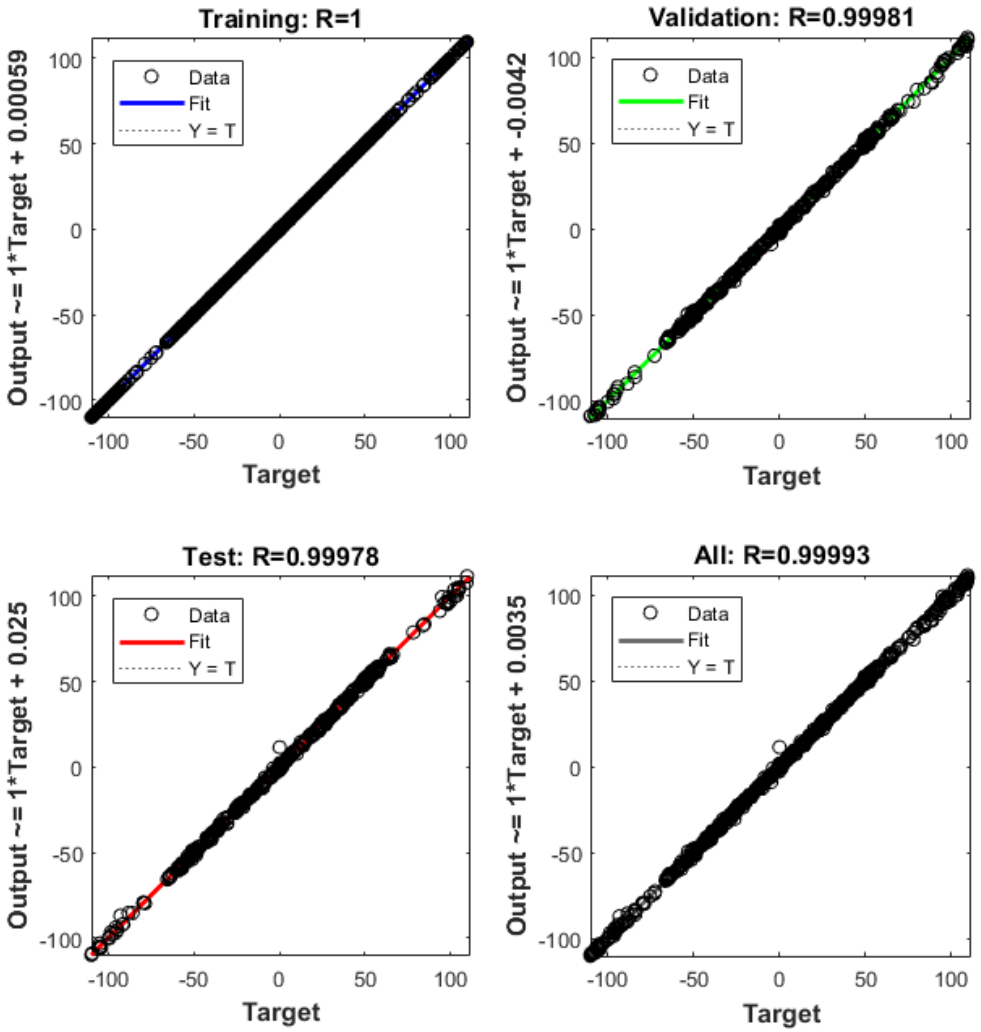
Az 12. ábra a hibák alakulását mutatja. A három adatkészletre vonatkozóan jelenít meg információkat. Megmutatja, hogy az egyes adatkészleten a hibák milyen tartományba

esnek. A függőleges narancssárga vonal jelzi a nulla értéket. Látható, hogy a hibák nagyjából 93%- a 0.1753 tartományba esnek, míg a maradék 7% eloszlik a többi érték között. A diagrammról leolvasható legnagyobb hiba -2.557 fok, ami nagyon kis érték, és nagyon ritkán fordul elő.




12. ábra: Hibák eloszlása

Az 13. ábra a három adatkészlet kormányszög értékét ábrázolja a hálózat válaszának függvényében. Látható, hogy a tanító adathalmaz illeszkedik a legjobban az átlóra, míg a teszt és validációs ponthalmaznál vannak kiugró értékek, amik megjelennek a hibák eloszlásánál is. Leolvasható, hogy van egy nagyobb hiba az 5 fok körül, és 80 fokos kormányszög értéknél a hibák sűrűsödnek. Megfigyelhető, hogy -60 fok körüli kormányszög értékek nincsenek az adatkészletben, annak ellenére, hogy a pályán mindkét irányban körbement a jármű. Ez a tesztpálya karakterisztikájából adódik.



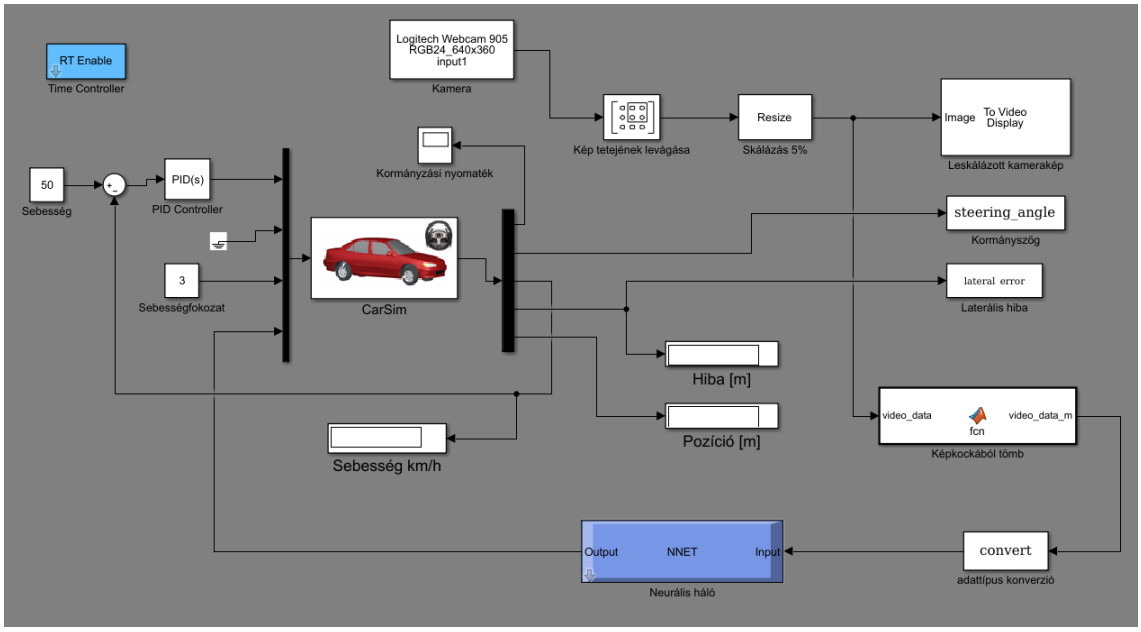

13. ábra: Kormányszög értékek a kimenet függvényében

### 3.2.5. Hálózat előhívása

A neurális hálózat tanítása Matlab függvényekkel történik, a hálózat előhívása ott rögtön megoldható. A feladat, hogy a CarSim alatt futó Simulink kódba átemeljük a hálózatot. Ezt a *gensim* függvénnyel lehet elvégezni. A betanított hálózat paraméterrel történő meghívása után generál a hálózatnak megfelelő Simulink blokkot, aminek a bemenete a képpontokból álló tömb, kimenete pedig a kormányszög lesz.

A hálózat felhasználásához és előhívásához létrehozásra került egy külön Simulink program (14. ábra), ami megkapja a kamera képét, levágja a tetejét, leskálázza és a tanítópontok létrehozása algoritmussal ellentétben nem tárolja el azokat, hanem a

hálózat (kék blokk) bemenetére küldi. A hálózat kimenete, pedig a CarSim blokk kormányszög bementére kapcsolódik. A jármű sebességszabályozásáért szintén egy PID szabályzó gondoskodik. A későbbi kiértékeléshez a laterális hiba és a kormányszög értékek eltárolásra kerülnek.

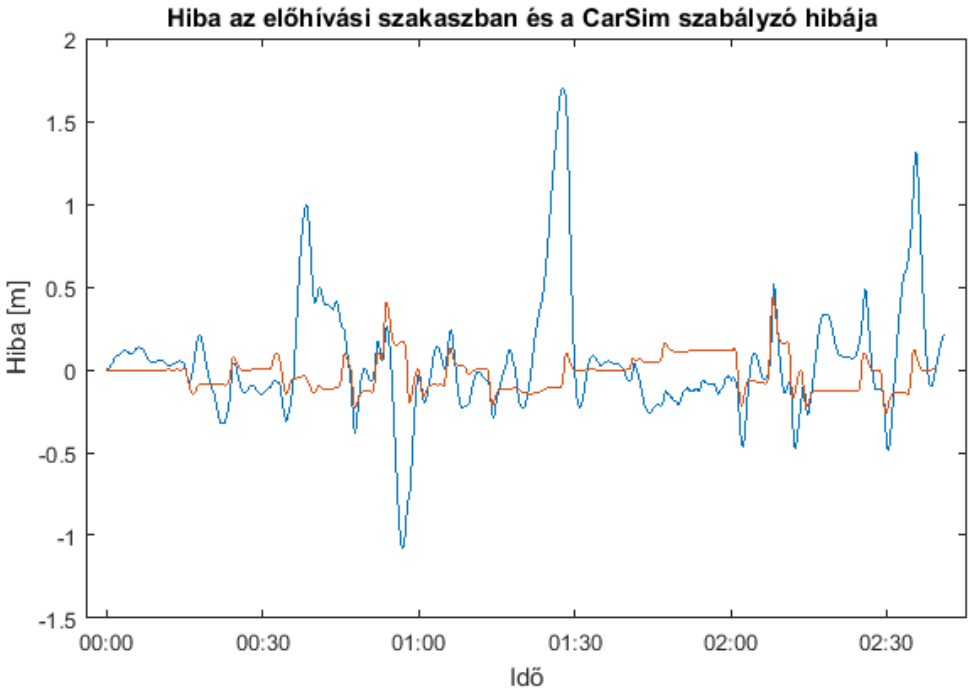



14. ábra: Előhíváshoz tartozó Simulink algoritmus.

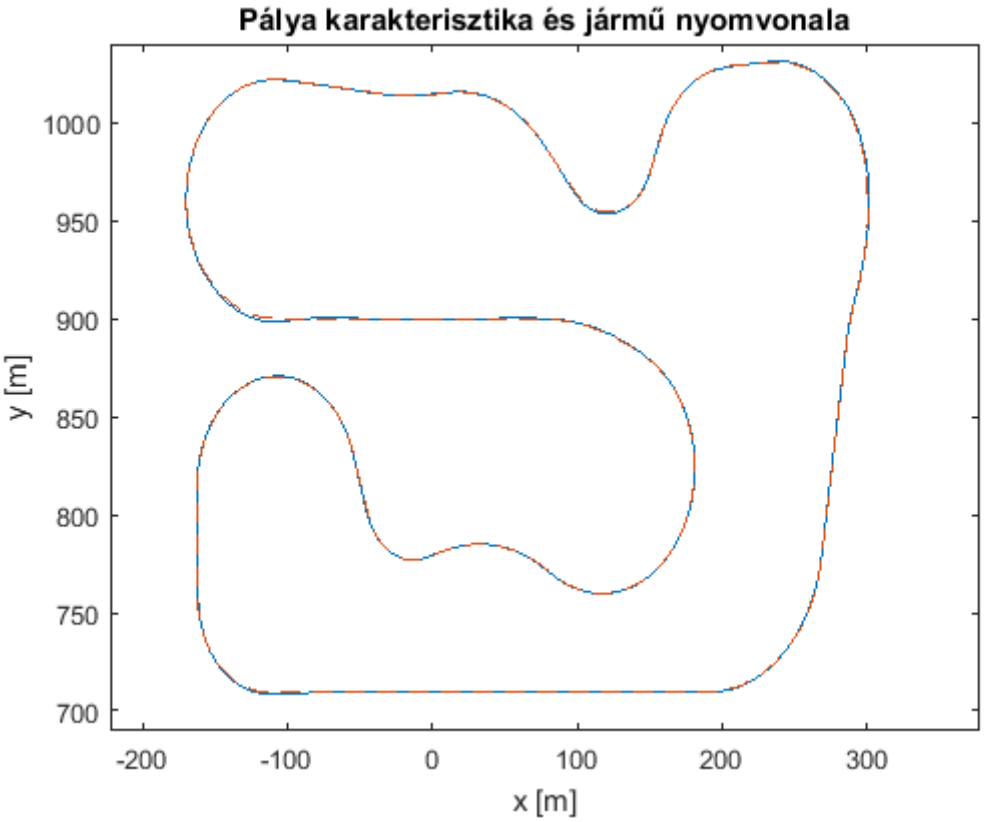
A szimuláció során a neurális hálózat előhívási szakaszában a jármű kormányzása a neurális hálózattól kapott értékek alapján történik. Egy kör megtétele közben rögzítésre került a jármű első tengelyének közepe és a pálya középvonalától mért távolság. Ez látható a 15. ábrán kék vonallal ábrázolva, míg a narancssárgával kirajzolt hiba a CarSim belső laterális szabályozójából származik.

Egy kör megtétele 50 km/h sebességgel két perc negyven másodpercig tartott. A legnagyobb hibák +1.7 és -1.1 méter körül alakultak, amik hatására a rendszer stabil maradt. A diagramon két perc környékén megfigyelhető, hogy a CarSim szabályzójában lévő hiba leképezésre került a neurális hálózatban.

Az 16. ábrán szaggatott narancsszínű vonallal a pálya nyomvonala, kék vonallal a neurális háló által körbevezetett jármű nyomvonala került kirajzolásra. Látható, hogy a legnagyobb hibák főként a nagyobb kanyarok kijáratánál jelentkeznek. Ez annak tudható be, hogy ekkor esik ki a kamera látószögéből leginkább a pálya.




15. ábra: Hiba az előhívási szakaszban és a CarSim szabályozó hibája




16. ábra: Pálya karakterisztika és a jármű nyomvonala

## 3.3. Konvolúciós neurális hálózat alapú rendszer

A második alfejezetben ismertetett módszerrel megvalósítottam szimulációs környezetben egy sávtartó funkciót többrétegű neurális hálózat alapokon. Ezt – a bemenetek számának csökkentése érdekében - a kamera által továbbított bemeneti kép jelentős leskálázásával és szürkeárnyalatossá konvertálásával oldottam meg.

A konvolúciós neurális hálózatok alapvetően képekben lévő információk kinyerésére szolgálnak, azok lehetnek színesek és akár nagyméretűek is. A neuronok száma nem fog olyan jelentős mértékben megnőni, így a bemeneti képen nem kell veszteséggel járó konverziókat végezni. A képek két dimenziós konvolúciója során, a konvolúciós ablakok szűrő paraméterei jelentik a súlyokat, a tanítás során ezeket kell meghatározni. A szűrők tulajdonság-érzékelőként viselkednek.

A feladatom második felében bemutatom a sávtartó funkció megvalósításának kérdéseit konvolúciós neurális hálózat alapokon. A téma aktualitása és jellege miatt számos jellemzőt tapasztalati úton határoztam meg. A következő fejezetekben többek közt megosztom a témával kapcsolatos kutatási eredményeimet, tapasztalataimat.

### 3.3.1. Tanító mintapontok előállítása

A tanító mintapontok előállítása, a már bemutatott megoldáshoz hasonlóan, szintén CarSim szoftverben írt Simulink program futtatásával történt.

A jármű pályán való négyszeri körbevezetése során, tizedmásodperces mintavételezéssel, eltárolásra kerül a 180 x 320 képpontból álló kamera színes képe, és a hozzá tartozó kormányszög. Ez egy 10 perc 48 másodperces anyagnak felel meg és 6464 összetartozó kép és kormány szög értéket foglal magában. Az adathalmaz eltárolása 2013R2 Matlab verzióval történt. A további munkám jelentős részét Matlab szoftverben végeztem, de mivel ez még nem támogatja a konvolúciós hálózatokat, ezért a lementett adatokat megnyitottam a jelenleg legújabb 2017b verziójú változatban.

A következő feladat a nyers adatok tanításra való előkészítése volt. A lementett képek egymás utáni sorozatának lejátszásával megkaphatjuk a négy pálya videóváltozatát. A tanítás hatékonyságát növeli, ha a mintapontok egymásutánisága nem követ törvényszerűséget, ezért írtam egy algoritmust, ami a mintapontokat véletlenszerűen összekeveri, és az összekevert párokat validációs és tanító adathalmazokba szeparálja. A 6460 darab mintából 2000 validációs adatként, míg 4460 tanító adatkét szolgál. Ez

követi az általános alkalmazott 30% -70% struktúrát. Az összekeverést és a szeparálást a következő algoritmussal valósítottam meg:

```
rand_perm=randperm(length(video_data),2000);
train_cnt=0;
valid_cnt=0;
rand_perm_for_loop=randperm(length(video_data));
for c= 1:length(video_data)
    i=rand_perm_for_loop(c);
    if(find(rand_perm==i))
        valid_cnt = valid_cnt + 1;
        steering_angle_validation(valid_cnt)=steering_angle(i);
        video_data_validation(:,:,:,valid_cnt)=video_data(:,:,:,i);
    else
        train_cnt = train_cnt + 1;
        steering_angle_train(train_cnt)=steering_angle(i);
        video_data_train(:,:,:,train_cnt)=video_data(:,:,:,i);
    end
end
```

Az előkészített adatstruktúrába rendezett képek 180 x 320 képpont méretűek. Egy mintakép a 17. ábra bal oldalán látható.

17. ábra: A tanító adathalmaz egy képe.

Megfigyelhető, hogy a kép felső részén látható felhők nem szolgálnak érdemi információval, ezért a kép tetejének levágása mellett döntöttem (17. ábra jobb oldala).

Előállt a tanító és a validációs adathalmaz, amik 70%-30% eloszlásban, összesen 6464 darab, 130 x 320 képpont méretű színes képet, és a hozzájuk tartozó kormányszöget tartalmazzák véletlenszerű sorrendben.

### 3.3.2. Hálózat betanítása

A konvolúciós neurális hálózat Matlab szoftverkörnyezetben való betanításához, a többrétegű „hagyományos” hálózatokkal ellentétben, a tanító mintapontokat véletlenszerűen szét kell választani tanító és validációs adathalmazokká. Miután ez a

művelet a 3.3.1 (Tanító mintapontok előállítása) fejezetben megtörtént, a képeket tartalmazó négydimenziós tömbök (szélesség x magasság x csatornaszám x elemszám), és a hozzájuk tartozó kormányszög értékeket tartalmazó tömbök (kormányszög x elemszám) további konverzió szükségessége nélkül felhasználhatók tanító és validációs adathalmazoknak.

A tanítás elkezdéséhez, a tanító adathalmazok (tanító és validációs) és a hálózati struktúrán kívül további paraméterek megadása szükséges. Meg lehet adni a kezdeti tanulási rátát, amit én 0.0001-re választottam. Ha ez túl kicsi érték, akkor a tanulás sokáig fog tartani, viszont ha túl nagy, akkor könnyen elérhetünk egy nem optimális eredményt. Ha nem adjuk meg a rátát, akkor 0.01-re veszi a rendszer. A tanulási ráta a hiba csökkenésével arányosan csökken. Meg kell adni, hogy mi legyen a tanulás végének feltétele. A maximális iterációs számot 100-ra vettem, és készítettem egy függvényt, ami a tanuló adatkészlet hibáit figyeli iterációnként. Ha a tanulás eléri a maximális iterációk számát, vagy ha a hiba állandósul, a tanításnak vége. További paraméterként meg lehet adni a futtatás hardverét. A tanításokat GPU-n végeztem. Számos beállítás megadására van még lehetőség, főként a tanulási fázis megjelenítése céljából. Ezeket alapértelmezett értéken hagytam.

### 3.3.3. Hálózat tulajdonságainak megválasztása

A konvolúciós neurális hálózatok tulajdonságainak megválasztásakor a rétegek számát, azok típusát, és a típusonként változó rétegparamétereket kell megadnunk. A bemeneti rétegnél a bemeneti képek méretét és csatornaszámát, a konvolúciós rétegnél pedig, a szűrő méretét, lépésközét és a keret méretét kell előre definiálni.

Az áttanulmányozott példákból kiindulva a bemeneti réteg után következő konvolúciós rétegekre a következő törvényszerűségeket fedeztem fel:

- Az első réteg(ek) nagyobb szűrőablakkal és kevesebb aktivációs térképpel rendelkeznek, míg a konvolúciós rétegek vége felé közeledve a szűrőablak mérete csökken, és az aktivációs térképek száma nő.
- A konvolúciós rétegek utáni teljesen összekötött első réteg jellemzően sok neuronból áll és a kimeneti réteghez közeledve ez a szám csökken. Többnyire kettő hatványai számú neuronokat találtam a példákban.

- Öt vagy több konvolúciós réteget alkalmaznak.
- A hálózatok végén 2-3 teljesen összekötött réteg szerepel.
- Az utolsó teljesen összekötött réteg neuron számának meg kell egyeznie a kimenetek számával.

A bevezetésben is bemutatott egyik kiinduló példa, az NVIDIA által készített, konvolúciós hálózaton alapuló önvezető jármű. A fejlesztők egy 9 rétegből álló konvolúciós neurális hálózatot hoztak létre, ami színes 66 x 200 képpont méretű képeket dolgoz fel. Az öt konvolúciós réteg közül az első három 5 x 5-ös, míg az utolsó kettő 3 x 3-as szűrőablakot kapott. Az aktivációs térképek száma sorban a következő: 24, 36, 48, 64, 64. Három teljesen összekötött réteg neuronszámai: 100, 50, 10. Ebből látszik, hogy a hálózat 10 kimeneti adattal rendelkezik. Ezek valósítják meg a jármű irányítását, és adnak információkat a környezetről.

A megfigyelt törvényszerűségeket és kiinduló mintapéldákat figyelembe véve elkészítettem hasonló struktúrájú hálózatokat, de a tanulás során rendre hibákba ütköztem. Ez leginkább annak tudható be, hogy az NVIDIA, 72 órányi adattal, vagyis több mint kétmillió mintapárral tanította be a hálózatát, ellentétben az általam létrehozott 6464 mintából álló adathalmazzal.

További kutatásokat végezve kijelenthető, hogy ekkora méretű konvolúciós hálózat tanításához legalább egymillió mintapont párral kell rendelkezni. Kevés mintaszám esetén két további lehetőség van a hálózat betanítására.

Az első megoldást nagyon gyakran alkalmazzák konvolúciós neurális hálózatokon alapuló rendszereknél. A *transfer learning* (tanulás átadása) módszer során egy előre, adott célra betanított hálózatot felhasználva végzik a tanítást. Jellemzően annak utolsó pár teljesen összekötött rétegét lecserélik a saját célra meghatározott paraméterű rétegekre. Ez azért vezethet jó eredményre, mert a konvolúciós rétegek a nagy mintaszámmal történő tanítás során felvették a tulajdonság érzékelő funkciójukat, és az utolsó pár rétegnek már „magasabb szintű” információkat szolgáltatnak. A Matlab fel van készítve előre betanított hálózatok beolvasására, és azok újbóli tanítására. Az interneten fellelhető számos kifejezetten erre a célra felhasználható hálózat, amik egy adott feladatra készültek. Ezek nagy többsége képek osztályokba sorolására lett betanítva, de az utolsó pár rétegük kicserélésével, akár regresszió típusú hálózat is

megvalósítható velük. A legnagyobb interneten fellelhető gyűjtemény a Model Zoo. Az ott található hálózatok caffe keretrendszerben készültek.

A legújabb verziójú Matlab-ot felkészítették a caffe hálózatok beolvasására, de sajnos ennek vannak korlátai. Két hálózat tűnt a legalkalmasabbnak a feladatom megoldására. Az egyik egy éldetektálásra, a másik pedig egy kép szegmentálásra betanított. A hálózatok Matlab szoftverbe importálása közben sajnos különböző hibákba ütköztem, ezért egy másik megoldás keresése mellett döntöttem.

A kevés mintaszámhoz, a minták jellegéhez és a hálózat kimenetéhez alkalmazkodva összeállítottam több kisebb hálózatot, a bemutatott törvényszerűségek lehetőség szerinti figyelembe vételével. A hálózatok struktúráját a következő felsorolás tartalmazza, a 4. táblázat pedig, a tanítás eredményeit.

1. hálózat:
   (1) 130 x 320 méretű, három csatornás bemeneti réteg
   (2) 5 x 5 méretű, 15 darab, 3 csatornás szűrőből álló konvolúciós réteg
   (3) ReLU aktivációs függvény
   (4) 3 x 3 méretű, 20 darab, 3 csatornás szűrőből álló konvolúciós réteg
   (5) ReLU aktivációs függvény
   (6) 15 neuronból álló teljesen összekapcsolt réteg
   (7) ReLU aktivációs függvény
   (8) 1 neuronból álló teljesen összekapcsolt réteg
   (9) Regressziós kimeneti réteg
2. hálózat:
   (1) 130 x 320 méretű, három csatornás bemeneti réteg
   (2) 13 x 32 méretű, 15 darab, 3 csatornás szűrőből álló konvolúciós réteg
   (3) ReLU aktivációs függvény
   (4) 6 x 15 méretű, 20 darab, 3 csatornás szűrőből álló konvolúciós réteg
   (5) ReLU aktivációs függvény
   (6) 15 neuronból álló teljesen összekapcsolt réteg
   (7) ReLU aktivációs függvény
   (8) 1 neuronból álló teljesen összekapcsolt réteg
   (9) Regressziós kimeneti réteg
3. hálózat:
   (1) 130 x 320 méretű, három csatornás bemeneti réteg
   (2) 13 x 32 méretű, 15 darab, 3 csatornás szűrőből álló konvolúciós réteg
   (3) ReLU aktivációs függvény
   (4) 15 neuronból álló teljesen összekapcsolt réteg
   (5) ReLU aktivációs függvény
   (6) 1 neuronból álló teljesen összekapcsolt réteg
   (7) Regressziós kimeneti réteg

4. hálózat:

   (1) 130 x 320 méretű, három csatornás bemeneti réteg
   (2) 13 x 32 méretű, 6 darab, 3 csatornás szűrőből álló konvolúciós réteg
   (3) ReLU aktivációs függvény
   (4) 6 x 15 méretű, 25 darab, 3 csatornás szűrőből álló konvolúciós réteg
   (5) ReLU aktivációs függvény
   (6) 15 neuronból álló teljesen összekapcsolt réteg
   (7) ReLU aktivációs függvény
   (8) 1 neuronból álló teljesen összekapcsolt réteg
   (9) Regressziós kimeneti réteg

5. hálózat:

   (1) 130 x 320 méretű, három csatornás bemeneti réteg
   (2) 13 x 32 méretű, 20 darab, 3 csatornás szűrőből álló konvolúciós réteg
   (3) ReLU aktivációs függvény
   (4) 5 neuronból álló teljesen összekapcsolt réteg
   (5) ReLU aktivációs függvény
   (6) 1 neuronból álló teljesen összekapcsolt réteg
   (7) Regressziós kimeneti réteg

6. hálózat:

   (1) 130 x 320 méretű, három csatornás bemeneti réteg
   (2) 13 x 32 méretű, 15 darab, 3 csatornás szűrőből álló konvolúciós réteg
   (3) ReLU aktivációs függvény
   (4) 20 neuronból álló teljesen összekapcsolt réteg
   (5) ReLU aktivációs függvény
   (6) 1 neuronból álló teljesen összekapcsolt réteg
   (7) Regressziós kimeneti réteg

| Hálózat száma | Futási idő [mm:ss] | Validációs négyzetes hiba | Tanulás négyzetes hibája | Iterációk száma |
|---|---|---|---|---|
| 1 | 50:31 | 2.1899 | 2.0166 | 2790 |
| 2 | 70:18 | 1.5351 | 1.2596 | 4400 |
| 3 | 124:17 | 0.7633 | 0.3363 | 8310 |
| 4 | 54:43 | 6.3384 | 4.2313 | 5400 |
| 5 | 46:54 | 2.671 | 2.48 | 6160 |
| 6 | 78:09 | 0.9171 | 0.6178 | 3150 |

4. táblázat: Hálózat struktúrájának választása.

A futtatások eredményéből látszik, hogy a 3. hálózati struktúrával értem el a legjobb eredményt. A 8310 iterációt a validációs hiba konvergálásával érte el, több mint két óra

alatt. A többi hálózat hamarabb bekonvergált egy nagyobb hibaértékre. Az eredmények kiértékelése fejezetben, a 3. tanítás eredményét mutatom be.

### 3.3.4. Eredmények kiértékelése

A „hagyományos” neurális hálózaton alapuló rendszer, betanítás után egy függvény meghívásával, könnyen előhívható volt Simulink környezetben. A CarSim szoftverben futó Simulink környezet csak 2013R2 Matlab verzióval kompatibilis, amiben sajnos még nincsenek a konvolúciós neurális hálózatok tanításához és kezeléséhez szükséges eljárások. A betanított konvolúciós neurális hálózatok szimulációs körben való tesztelésére nem volt lehetőségem, de az adatok kiértékelése rendre megtörtént.

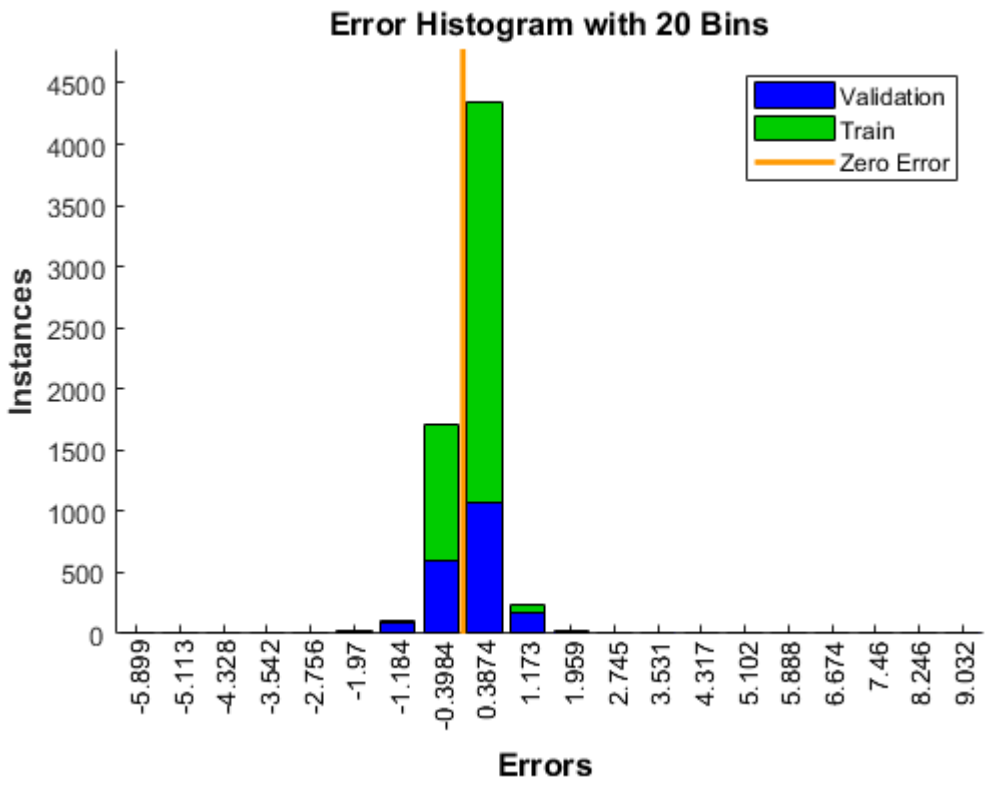
Az egy konvolúciós és egy teljesen összekötött rétegből álló, 3. tanítás a következő diagramokon bemutatandó eredménnyel zárult.

A 18. ábrán látható hibák eloszlásából látszik, hogy mind a tanító és a validációs mintapontok több, mint 90 %-a 0.3874 fokon belül van. A legnagyobb hiba, ami nagyon ritkán fordul elő 1.97, ami egy jármű kormányszög tartományát nézve igen kicsi.

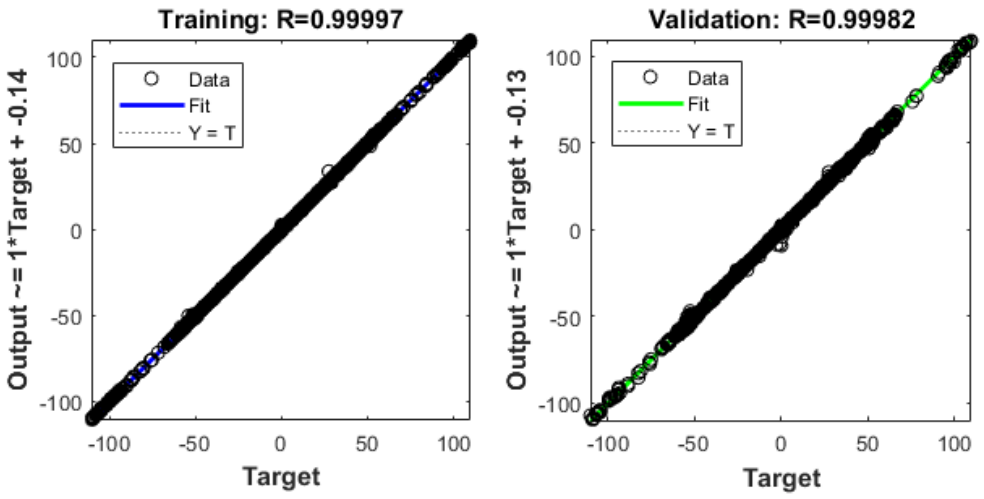
A diagramból megállapítható, hogy a tanító és a validációs tanítópontok eloszlása egymással arányos, amiből a hálózat megfelelő általánosító-képességére lehet következtetni.

A 19. ábrán, baloldalon látható a tanító adatkészleten vett hálózat kimenete a kormányszög függvényében, míg a jobboldalon a validációs készletre meghatározva. Megfigyelhető, hogy a validációs adatok nem illeszkednek annyira jól a regressziós egyenesre. Van néhány érték, főleg az 5 fok körüli tartományban, ahol megjelennek apróbb eltérések.

A diagramok alapján kijelenthető, hogy a hálózat kellő pontossággal reprezentálja a mintákban fellelt törvényszerűségeket.




18. ábra: Hibák eloszlása




19. ábra: Kormányszög értékek a kimenet függvényében

# 4. Összefoglalás, további célkitűzések

A kutatás eredményeképpen elmondható, hogy mind regresszió típusú, mind konvolúciós típusú neurális hálózatokkal komoly eredmények érhetők el a járműirányítás területén.

Az elért eredmények mellett a dolgozat elkészítése, és az azt megelőző kutatómunka számos témakört mélyen érintett, ami hozzájárult egy aktuális és értékes tudásanyag elsajátításához.

A témában további lehetőségek rejlenek. Jelenleg a betanított hálózatok egy kimeneti paraméterrel rendelkeznek, a kormányszög értékkel. A mintapontok előállítása során és a betanított hálózat előhívásakor fix, 50 km/h sebességgel halad a jármű. Megvalósítható feladat lehet a feladat változó sebességre való kiterjesztése, amikor a cél a leggyorsabb kör megtétele. A gépi tanulási megoldások közül nagyon ígéretesek a különböző megerősítéses tanulási módszerek irányítási alkalmazásai. Kutatásaim során erről a területről is sokat olvastam, azonban idő- és helyhiány miatt az ezzel szerzett tapasztalataimat nem tudtam a dolgozatban bemutatni. A jövőben azonban az itt alkalmazott felügyelt „end-to-end” tanulás helyett a megerősítéses tanulásra kívánok fókuszálni.

További lehetőségként megemlíteném, hogy a fejlesztői környezet módosítása komoly lehetőségeket rejt magában. A CarSim szoftver újabb verzióra cserélése vagy kiváltása nagyobb és változatosabb mintapont készlet előállítását tenné lehetővé, és bővülnének a tesztelési lehetőségek. Ezen kívül összetettebb szituációk szimulációjához alkalmasabbak lehetnek a CarMaker vagy PreScan szoftverek. Ezeken túl ingyenes lehetőségek is rendelkezésre állnak. A téma kutatásának kezdeti szakaszában lehetőségem volt megismerni a Unity játékfejlesztő szoftvert, amelyben teljesen az alapoktól elkészíthető egy CarSim programhoz hasonló szimulációs környezet. További lehetőség a The Open Rasing Simulator (TORCS), amely kész járműmodellekkel álla kutatók rendelkezésére, azonban a környeztek a különböző versenypályákra korlátozódnak. A befektetendő többletmunka ellenére ezek azért lehetnek előnyösek, mert könnyen megoldható a kommunikáció a jelenleg legelterjedtebb és legeredményesebben alkalmazható – Python alapú – gépi tanulás környezetekkel.

# 5. KÖSZÖNETNYILVÁNÍTÁS

EFOP-3.6.3-VEKOP-16-2017-00001: Tehetséggondozás és kutatói utánpótlás fejlesztése autonóm járműirányítási technológiák területén - A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

AZ EMBERI ERŐFORRÁSOK MINISZTÉRIUMA ÚNKP-17-2-I KÓDSZÁMÚ ÚJ NEMZETI KIVÁLÓSÁG PROGRAMJÁNAK TÁMOGATÁSÁVAL KÉSZÜLT

# 6. IRODALOMJEGYZÉK


[1] D. A. Pomerleau, „ALVINN, an autonomous land vehicle in a neural," 1989.

[2] S. Russel and P. Norvig, Mesterséges Intelligencia Modern megközelítésban, Panem Kft., 2005.

[3] M. Altrichter, G. Horváth, B. Pataki, G. Strausz, G. Takács és J. Valyon, Neurális hálózatok, Panem Kft., 2007.

[4] „Mesterséges Intelligencia Almanach," Budapesti Műszaki és Gazdaságtudományi Egyetem, Semmelweis Egyetem, Óbudai Egyetem, [Online]. Available: http://mialmanach.mit.bme.hu/. [Hozzáférés dátuma: 28 október 2017].

[5] I. Egendy, „Engedy István honlapja, Konvolúciós neurális hálózatok (CNN)," Budapesti Műszaki és Gazdaságtudományi Egyetem, Méréstechnika és Információs Rendszerek Tanszék, [Online]. Available: http://home.mit.bme.hu/~engedy/NN/NN-CNN.pdf. [Hozzáférés dátuma: 28 október 2107].

[6] „CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Hozzáférés dátuma: 28 október 2017].

[7] „Choose a Multilayer Neural Network Training Function," MathWorks, [Online]. Available: https://www.mathworks.com/help/nnet/ug/choose-a-multilayer-neural-network-training-function.html. [Hozzáférés dátuma: 28 október 2017].

[8] „Simulation and Model-Based Design," MathWorks, [Online]. Available: https://www.mathworks.com/products/simulink.html. [Hozzáférés dátuma: 28 október 2018].

[9] „Mechanical Simulation, CarSim," Mechanical Simulation Corporation, [Online]. Available: https://www.carsim.com/products/carsim/index.php. [Hozzáférés dátuma: 28 október 2017].

[10] A. Nehemiah, „Deep Learning: Transfer Learning in 10 lines of MATLAB Code," MathWorks, 24 február 2017. [Online]. Available: https://blogs.mathworks.com/pick/2017/02/24/deep-learning-transfer-learning-in-10-lines-of-matlab-code/. [Hozzáférés dátuma: 28 október 2018].

[11] A. Nehemiah, „Deep Learning for Automated Driving with MATLAB," NVIDIA, 20 július 2017. [Online]. Available: https://devblogs.nvidia.com/parallelforall/deep-learning-automated-driving-matlab/. [Hozzáférés dátuma: 28 október 2017].

[12] M. Bojarski, B. Firner, B. Flepp, L. Jackel, U. Muller and K. Zieba, "End-to-End Deep Learning for Self-Driving Cars," NVIDIA, 17 augusztus 2016. [Online]. Available: https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/. [Accessed 28 október 2017].

[13] Y. Jia és E. Shelhamer, „Caffe," [Online]. Available: http://caffe.berkeleyvision.org/model_zoo.html. [Hozzáférés dátuma: 28 október 2017].

[14] „Model Zoo," [Online]. Available: https://github.com/BVLC/caffe/wiki/Model-Zoo. [Hozzáférés dátuma: 28 október 2017].

[15] „Unity - Game Engine," [Online]. Available: https://unity3d.com/. [Hozzáférés dátuma: 28 október 2018].

## 7. ÁBRAJEGYZÉK



## 8. TÁBLÁZATJEGYZÉK