



Nemzeti Közsolgálati Egyetem  
Hadtudományi és Honvédtisztképző Kar  
Intézményi Tudományos Diákköri Konferencia



**Biztonságtechnikai célú arcfelismerő rendszer  
fejlesztése**

Készítette: Fazekas Gábor

Intézmény: Nemzeti Közsolgálati Egyetem

Kar: Hadtudományi és Honvédtisztképző kar

Szak: Védelmi infokommunikációs  
rendszertervező mesterképzési képzési szak

Konzulens neve, beosztása: Dr. Németh András, egyetemi docens

Kovács Sándor, fejlesztőmérnök

Neptun kód: KHIYMO



NEMZETI KUTATÁSI, FEJLESZTÉSI  
ÉS INNOVÁCIÓS HIVATAL

AZ NKFI ALAPBÓL  
MEGVALÓSULÓ  
PROJEKT

# Tartalom

|                                                              |           |
|--------------------------------------------------------------|-----------|
| <b>BEVEZETÉS .....</b>                                       | <b>5</b>  |
| <b>1 BIOMETRIKUS ALAPÚ BELÉPTETŐ RENDSZEREK .....</b>        | <b>7</b>  |
| 1.1 BIOMETRIKUS AZONOSÍTÁS ALAPJAI .....                     | 8         |
| 1.2 MESTERSÉGES INTELLIGENCIA ALAPJAI .....                  | 10        |
| 1.2.1 Gépi tanulás .....                                     | 10        |
| 1.2.2 Neurális hálózat alapú mélytanulás .....               | 11        |
| 1.2.3 Mesterséges intelligencia katonai alkalmazása .....    | 12        |
| <b>2 A FEJLESZTÉS SZOFTVERKÖRNYEZETE .....</b>               | <b>14</b> |
| 2.1 DEEP LEARNING KERETRENDSZEREK .....                      | 15        |
| 2.2 MATLAB ÉS A MODELL ALAPÚ FEJLESZTÉS .....                | 17        |
| <b>3 ARCFELISMERŐ RENDSZER FEJLESZTÉSE .....</b>             | <b>25</b> |
| 3.1 KERAS DCGAN ARCGENERÁTOR .....                           | 25        |
| 3.1.1 Fejlesztői környezet kialakítása Docker-ben .....      | 27        |
| 3.1.2 DCGAN futtatása docker-ben .....                       | 32        |
| 3.1.3 DCGAN-nal kapcsolatos eredmények .....                 | 33        |
| 3.2 MATLAB – CNN .....                                       | 36        |
| <b>4 HARDVER IMPLEMENTÁCIÓ .....</b>                         | <b>38</b> |
| 4.1 MIKROVEZÉRLŐ MI KÉPESSÉGEI .....                         | 38        |
| 4.1.1 A vizsgálat célja .....                                | 38        |
| 4.1.2 Állapotgép .....                                       | 39        |
| 4.1.3 Gépi tanulás mikrovezérlőn .....                       | 39        |
| 4.1.4 Feladat meghatározása, neurális hálózat tanítása ..... | 40        |
| 4.1.5 Neurális hálózat felépítése .....                      | 43        |
| 4.1.6 Tensorflow Lite modell generálása .....                | 46        |
| 4.1.7 Tensorflow lite main.c .....                           | 48        |
| 4.1.8 Ellenőrzés .....                                       | 50        |
| 4.1.9 Tesztelés 2000 és 3000 mintával .....                  | 51        |
| 4.1.10 A beágyazott MI konklúziója .....                     | 53        |
| 4.1.11 Nvidia Jetson Xavier .....                            | 53        |
| 4.1.12 FPGA implementáció .....                              | 54        |
| <b>ÖSSZEGZÉS .....</b>                                       | <b>57</b> |
| <b>ÁBRAJEGYZÉK .....</b>                                     | <b>59</b> |

|                      |    |
|----------------------|----|
| IRODALOMJEGYZÉK..... | 61 |
|----------------------|----|

# Bevezetés

A különböző szempontból kiemelt jelentőségű objektumok, területek biztonságának szavatolása évezredes kérdés, melynek elvei kisebb mértékben és lassabban, technikai eszközrendszere viszont annál nagyobb léptékben fejlődött a történelem, különösen a 20. majd a 21. század folyamán. A cél alapvetően a személy-, illetve vagyonbiztonság feltételeinek megteremtése, melynek szintje a személyek, közösségek társadalomban, politikában, gazdaságban betöltött szerepétől, az anyagi javak mértékétől (eszmei és vagyoni értékétől), valamint az elhelyezkedés, elhelyezés körülményitől függött, beszéljünk akár egyéni, akár közösségi érdekekről.

Ennek egyik legfontosabb eleme, hogy a védendő területekre csak azok léphetnek be, akik arra jogosultsággal rendelkeznek, csak akkor, amikor az szükséges, vagy számukra megengedett, és csak olyan eszközöket vihetnek magukkal, amelyek semmilyen módon nem csökkentik az adott hely, személyek, illetve tevékenységek biztonsági szintjét. Ennek felügyeletére különböző megoldások, egyszerűbb vagy komplexebb rendszerek léteznek, melyeknek lehetnek humán és technikai elemi egyaránt. A legegyszerűbb egy biztonsági őr, aki ellenőrzi valamilyen hitelesített lista segítségével a belépési jogosultságot a személyeknél található azonosító okmányok alapján, míg a legbonyolultabbak esetén többszintű, biometrikus azonosítási eljárásokat is alkalmazó rendszerekről beszélhetünk, melyek működtetését, felügyeletét mesterséges intelligenciával (MI) támogatott megoldásokkal végzik.

Terület- illetve objektumvédelem esetén a biztonság megteremtésének egyik fő kritériuma az egyes személyekhez tartozó jogosultságok hiteles ellenőrzése. Az egyén azonosítása alapvetően lehet tudásalapú, birtokalapú, illetve biometrikus [1]. Utóbbi módszer jellemzően a személy testi jellemzői alapján végzi az azonosítást. A biometrikus beléptetőrendszerek már régóta megtalálhatók a piacon, mint például az Idemia cég által gyártott Visionpass (1. ábra), mely jelszavas, kártyás és arcfelismerésen alapuló biometrikus autentikációt is lehetővé tesz.

A biometrikus azonosítás legnagyobb kihívása, hogy szélsőséges üzemi körülmények között is képes legyen minimalizálni a hibás elfogadási arányt ( $FAR^1$ ) és a hibás elutasítási arányt ( $FRR^2$ ) a beléptetések során [1]. A működésükhöz kapcsolódó jel-, adat- és információ feldolgozó technológiák a mesterséges intelligencia témakörébe tartoznak, amely iránti érdeklődés az elmúlt években robbanásszerű növekedésnek indult. Ma már szinte minden

---

<sup>1</sup> FAR: False Accept Rate – Hibás elfogadási arány

<sup>2</sup> FRR: False Reject Rate – Hibás elutasítási arány

digitális eszköz tartalmaz valamilyen szintű mesterséges intelligenciát, amely jelen dolgozatnak is tárgyát képezi. A gépi tanulás, illetve a neurális hálózatokon alapuló mélytanulás jelenleg messze ez a leghatékonyabb módszer a különböző képfeldolgozási feladatok végrehajtására [2].



**1. ábra: Idemia VisionPass (forrás: [3])**

Az eljárás egyik jellemző alkalmazási területét képezik a gépjárművek önvezető rendszerei, amelyek fejlesztése lehetőségeket és buktatókat egyaránt tartalmaz, különös tekintettel a katonai felhasználást követelményeire [4]. Ez utóbbi esetben a személyzet nélküli szárazföldi-, vízi- és légi járművek akár fegyverrendszereket is hordozhatnak, ami tovább szélesíti a figyelembe veendő szempontok körét [5]. Meg kell említeni, hogy ezek a járművek távirányítással vezérelhetők, ugyanakkor alkalmazhatóság szempontjából az autonómia foka is lényeges paraméter, különösen akkor, ha figyelembe vesszük a rádiócsatornát veszélyeztető tényezőket, a nem szándékos zavarokat és a szándékos zavarást.

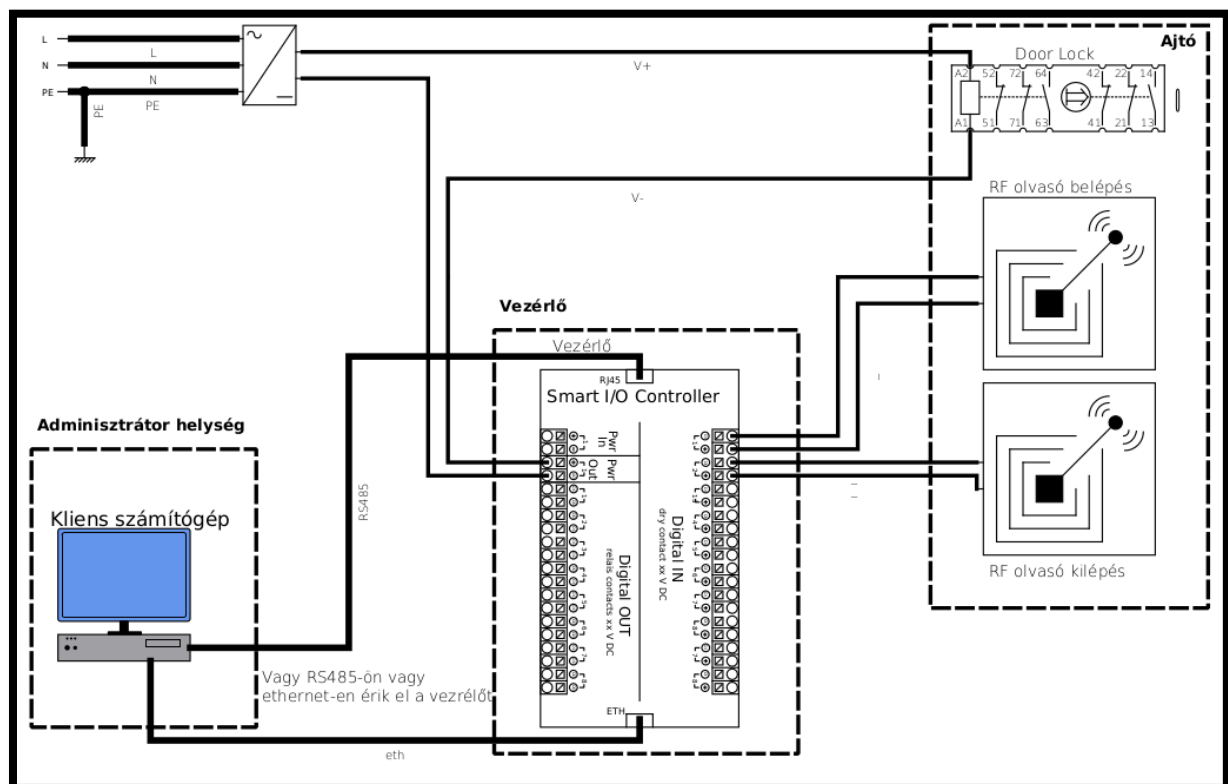
Az MI fejlesztések hardver-implementációjának lehetősége szintén fontos területté vált. Ma már kisteljesítményű mikrovezérlők is képesek neurális hálózatok futtatására, melyet dolgozatomban külön vizsgálok. Végző célom, hogy modellalapú fejlesztés keretében egy arcfelismerésre előtanított neurális hálózat FPGA<sup>3</sup>-ra implementálását készítsem elő.

<sup>3</sup> FPGA: Field Programmable Gate Array – Programozható logikai kapumátrixot tartalmazó integrált áramkör

# 1 Biometrikus alapú beléptető rendszerek

A tervezés alatt álló rendszer feladatai közé tartozik egy adott körlet védelme, az illetéktelen belépések jelzése, illetve a személyi forgalom monitorozása, nyomon követése. A mechanikai akadály kivitelezése során a telepítési utasításban foglaltakhoz, illetve az adott helyszín adottságaihoz kell alkalmazkodni. Ezek függvényében az lehet például személyi sorompó, forgóvilla, vagy ajtó. Jelen dolgozat tárgyát ugyanakkor az automatikus személyazonosítás képezni, amire alapvetően háromféle technológia létezik [1] [6]:

1. Tudásalapú: Ilyenek a számítógépes jelszavak, PIN kódok. Előnye a könnyű cserélhetőség, egyszerű felhasználhatóság, illetve a járulékos költségek hiánya. Hátránya a könnyű megfigyelhetőség, illetve, hogy használata nincsen személyhez kötve, így a kompromittálódás veszélye nagy.
2. Birtokalapú: Amikor az adott felhasználóhoz egy tárgy van rendelve, ami szükséges az azonosításhoz, például egy RFID<sup>4</sup> tag (2. ábra). Előnyei az egyszerű használat, könnyű elfogadtatás a felhasználókkal, egyéb felhasználási lehetőségek (hálózati bejelentkezés, nyomtató vezérlés, étel-ital automaták vezérlése stb).



2. ábra: RFID alapú beléptető rendszer (szerk. a szerző)

<sup>4</sup> RFID: Radio Frequency IDentification – Rádiófrekvenciás azonosító

3. Biometrikus azonosítás: A személyek valamely egyedi testi jellemzőinek, jegyeinek felismerése automatizált rendszerekkel, mint például az ujjlenyomat, írisz, érhálózat, vagy arc.

## 1.1 Biometrikus azonosítás alapjai

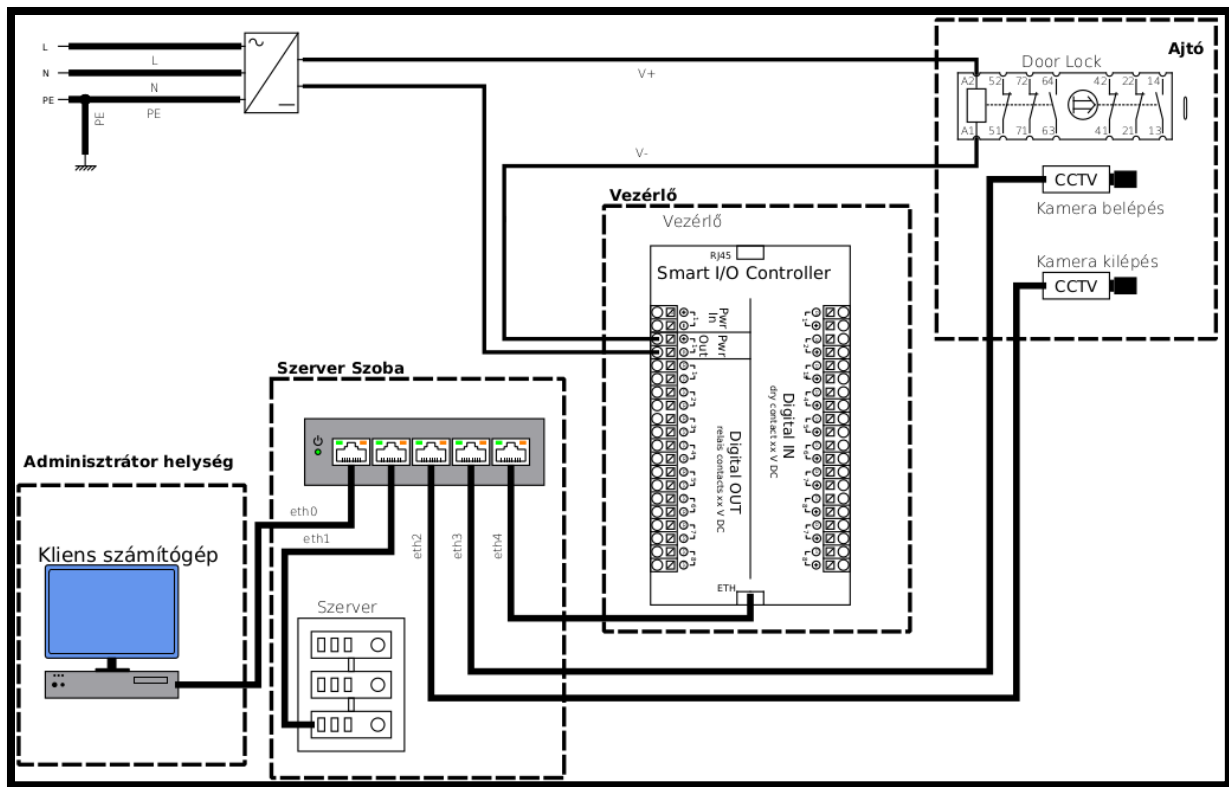
A biometrikus azonosítással kapcsolatos általános szempontok [7]:

- A minta megfelelő levétele és kódolása;
- A magas információtartalom megőrzése a „brute force” támadások ellen úgy, hogy a minta reprodukálható legyen;
- A személyes adatok védelme.

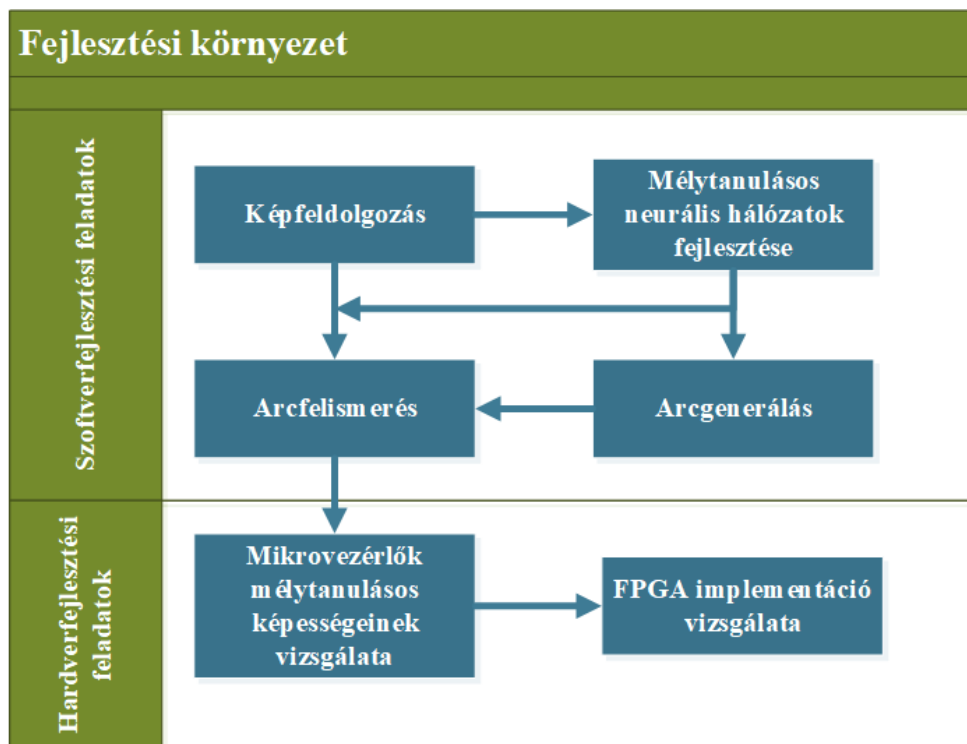
Bár a biometrikus azonosítás egy kézenfekvő megoldás lehet, gyakorlati alkalmazása és alkalmazhatósága több szempont függvénye [1]:

- Az adott jellemzőkkel minden személynek rendelkeznie kell;
- Ezeknek a jellemzőknek egyedinek kell lenniük;
- Ezek a jellemzők adott időn belül ne változhassanak (egészségügyi okok, öregedés);
- A jellemző beolvasható legyen;
- Beolvasás pontossága, sebessége;
- A biometrikus azonosítási módszer elfogadása a felhasználók által;
- Megtéveszthetőség figyelembe vétele.

Jelen dolgozat tárgyát képező biometrikus azonosítás az arc jellemzőit veszi alapul (3. ábra), az azonosításhoz felhasznált érzékelők kamerák, az egyedi, biometrikus azonosító pedig az emberi arcvonások halmaza. Az azonosítást képfeldolgozási eljárásokkal és egyéb, a mesterséges intelligencia témakörébe tartozó mélytanulós neurális hálózatokkal végezzük. A betanított arcfelismerő rendszer első prototípusa szerveren fut, amely egyéb perifériális elemek segítségével az autentikáción túl az engedélyezést, ajtók, kapuk nyitását is végzi. A végleges koncepcióban a szerver feldolgozási feladatait felváltja egy FPGA alapú kártya, mely IoT eszközként funkcionál. Mivel a teljes adatfeldolgozás helyben történik, ezért nincs szükség szélessávú kommunikációra, a rendszer telepítése rugalmasan megoldható, minimalizált infrastrukturális követelmények mellett. Jelen dolgozat a koncepciókat tárgyalja, illetve a megvalósítás menetét, amely a szoftveres keretrendszer kialakításán túl a hardverimplementáció korlátait is vizsgálja (4. ábra). A távlati cél egy megvalósítható fejlesztési munkamenet kidolgozása és végrehajtása.



3. ábra: Gépi látás alapú beléptető rendszer (szerk. a szerző)



4. ábra: Fejlesztési feladatok (szerk. a szerző)

## 1.2 Mesterséges intelligencia alapjai

Dolgozatom célja egy arcfelismerés alapú biometrikus azonosító rendszer kialakítása, mesterséges intelligencia felhasználásával, ezért célszerű áttekinteni annak elméleti alapjait.

Az MI felhasználása ma már rendkívül széleskörű, fejlődése az idők során különböző szakaszokra osztható. Az első lényeges szakasz az 1980-as évekig tartott, ez a szimbolikus mesterséges intelligencia szakasza [8] [9]. Ekkoriban a mesterséges intelligencia csúcsát a kézzel megírt állapotgépek jelentették, amelynek bemenetei a szabályrendszer és az adatok. Ezek alapján az algoritmus döntéseket hoz, így az alkalmas egyszerűbb problémák, szabályozási feladatok és táblajátékok feladványainak megoldására, ugyanakkor képek osztályozására, vagy beszédfelismerésre már nem [10].

### 1.2.1 Gépi tanulás

A klasszikus programozási eljárások mellett a 90-es évek elején gépi tanulás (ML<sup>5</sup>) népszerűsége folyamatos növekedésnek indult a mesterséges intelligencia területén [10]. Ennek lényege, hogy a klasszikus programozási eljárásokkal szemben az algoritmusunkat jellemzően inkább tanítjuk, mint programozzuk. A tanítás mértékét, illetve a tanításra és validálásra szétcsoportosított adatmennyiséget meghatározva az algoritmus bemeneti paraméterei a bemeneti adat és az elvárt kimenet lesznek. A tanítási folyamat során a kimeneten megkapjuk a szabályrendszert, például egy átviteli függvényt, melyet a tanítási folyamat után fel tudunk használni.

A gépi tanulást négy további alcsoportra szokás osztani [11] [12]:

- Felügyelt tanulás (supervised learning): A tanítást az ember végzi felcímkézett adatok segítségével. Alkalmas karakterfelismerésre, beszédfelismerésre, képek osztályozására és fordításra is.
- Felügyelet nélküli tanulás (unsupervised learning): Legfőképpen adat analízishez használják, adatok transzformálásához és vizualizálásához.
- Önfelügyelő tanulás (Self-supervised learning): közel áll a felügyelt tanuláshoz, de önálló kategória. Lényege, hogy nem kell ember az adatok címkézéséhez, hanem azt egy heurisztikus algoritmus oldja meg. Olyan esetekben alkalmazható, mikor a korábbi események alapján kell becsléseket végezni jövőbeli adatsorokra vonatkozóan.

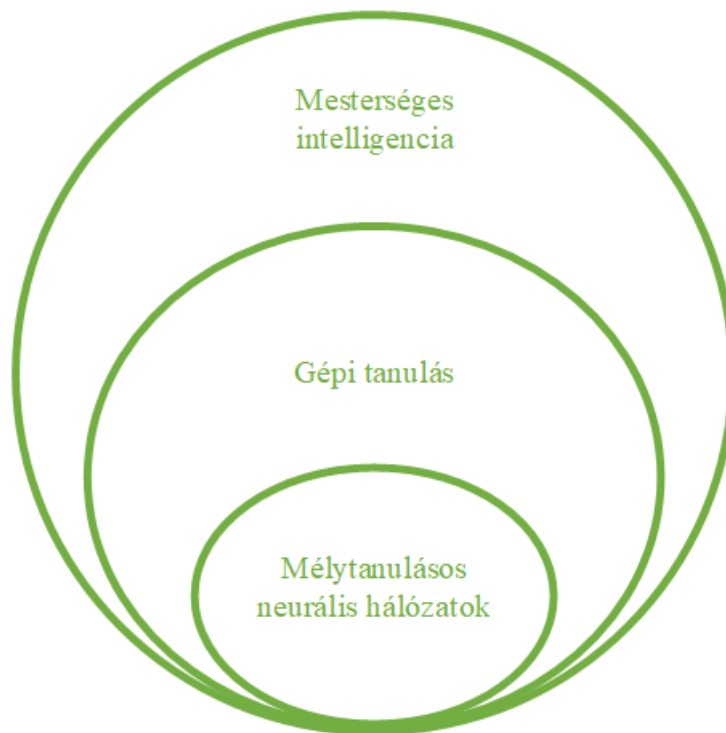
---

<sup>5</sup> ML: Machine learning – Gépi tanulás

- Megerősítéses tanulás (Reinforced learning): Lényege, hogy egy ágens, ami információt kap a környezetéről, döntéseket hoz, amiért jutalmat kap. Célja, hogy a jutalmat maximalizálja, ez alapján önmagát tanítja.

### 1.2.2 Neurális hálózat alapú mélytanulás

A neurális hálózatok egyrétegű „perceptron” formájában már az 1950-as évek óta kutatások tárgyát képezték. Ezek a hálózatok egyetlen réteget tartalmaztak és egy aktivációs függvényt, a bemeneti súlyok beállítása még nem volt megoldott. A nagy áttörés 1989-ben következett be, mikor Yann LeCun megalkotta a LeNet hálózatot, ami alkalmas volt kézzel írott karakterek csoportosítására. A módszer lényege, hogy egy többrétegű perceptron hálózat bemeneten lévő súlyait a kimenet hibája alapján hangoljuk (tanítjuk), a módszer neve „backpropagation”. A LeNet-et 1990-től az Egyesült Államok Postaszolgálatára irányítószámok csoportosítására használta.



**5. ábra: Mesterséges intelligencia csoportosítása (szerk. a szerző)**

Bár 2010 környékén a neurális hálózatok témakörét még többnyire kerülte a tudományos közösség, a területen dolgozó néhány kutatócsoport mégis fontos áttörést ért el. 2011-ben az ImaNet kihíváson 1.4 millió képen végrehajtott tanítást követően nagyfelbontású képeket kellett 1000 osztályba sorolni. Ekkor az első 5 helyezett még klasszikus képfeldolgozási módszereket használt, a legnagyobb pontosság pedig 74.3% volt. 2012-ben Alex Krizhevsky,

Geoffrey Hinton tanácsainak köszönhetően (Torontói Egyetem) már 83.6%-os eredményt ért el konvolúciós neurális hálózattal (CNN<sup>6</sup>). Ezt követően már a CNN vált a kihíváson használt domináns technikává, amivel 2015-ben 96.4%-os eredményt is sikerült elérni [2], így az ImaNet problémát megoldottnak nyilvánították. A mélytanulós neurális hálózatok (DNN<sup>7</sup>) kutatása ezt követően kiszélesedett. A jobb aktivációs függvényeknek, a súly inicializálási módszerek finomításának, a hatékonyabb optimalizálási eljárások megjelenésének (Adam), illetve ezek videokártyán történő futtatása lehetőségének köszönhetően vált lehetővé a 10 és afölötti rétegszámú DNN-ek megalkotása. Manapság már lehetőségünk van már akár több ezer réteggel rendelkező DNN-ek tanítására is.

### 1.2.3 Mesterséges intelligencia katonai alkalmazása

A társadalom erőteljes érintettsége okán [13] Európai Unió 2018-2019 során dolgozta ki a mesterséges intelligenciáról szóló összehangolt tervet, amely felvázolja, hogy az MI értéklánc egyes elemei szempontjából az EU milyen módon kíván hozzájárulni a tagállamok MI fejlesztéseihez. Ebben arra ösztönzi a tagállamokat, hogy 2019 közepére alakítsák ki saját nemzeti MI-stratégiáikat, az európai szinten elvégzett munkára támaszkodva [14]. Magyarországon az Innovációs és Technológiai Minisztérium a Mesterséges Intelligencia Koalícióval a Digitális Jólét program keretein belül 2020-ban megalkotta Magyarország mesterséges intelligencia stratégiáját, mely 2030-ig vázolja a rövid-, közép- és hosszútávú terveket a mesterséges intelligencia polgári célú felhasználását illetően [15].

A Magyar Honvédség a Zrínyi Honvédelmi és Haderőfejlesztési Program (korábban Zrínyi 2026) keretében azt a célt tűzte ki, hogy 2026-ra a térség legmodernebb hadereje legyen. A katonai képességek fejlesztése és az eszközbeszerzés folyamatosan történik és bár ezek a rendszerek valamilyen szintű mesterséges intelligenciát tartalmaznak, az MI, mint eszközkészlet vagy képesség jelenleg nem szerepel a fejlesztési célok között [16]. Fontos megemlíteni ugyanakkor, hogy a Mesterséges Intelligencia Koalíció egyik alapító tagja a Magyar Honvédség [17], tehát nem zárható ki, hogy akár a közeljövőben is induljanak mesterséges intelligenciát érintő projektek.

Napjainkban már nehéz olyan digitális alapú eszközt találni, melyben nem került implementálásra valamilyen szintű MI, gondoljunk akár szélessávú telekommunikációs rendszerek alapsávi jelfeldolgozására, vagy a jelformálásra. Ugyanez elmondható minden olyan

---

<sup>6</sup> CNN: Convolutional Neural Network – Konvolúciós neurális hálózat

<sup>7</sup> DNN: Deep Learning Neural Network – Mélytanulós neurális hálózat

berendezésről is, amelyek szenzorrendszerek segítségével végeznek szabályozási feladatokat vagy transzformációkat adathalmazokon. Ebből következik, hogy pusztán az MI jelenléte polgári alkalmazások tekintetében már nem teszi lehetővé messzemenő következtetések levonását a különböző funkciókra vonatkozóan. Más a helyzet azonban, ha a technológia jelenlegi állását vesszük górcső alá, azaz a mélytanulósos neurális hálózatok alkalmazását, kifejezetten az autonóm járművek vonatkozásában. A polgári célú gépjárműveknél jól megfigyelhető az a tendencia, ahogy a vezetést támogató rendszerek egyre nagyobb teret nyernek a biztonságkritikus alkalmazásokban. Az önvezetés funkció ugyanakkor önmagában egy olyan kérdéskört jelent, ami az autópárhban jelenleg érvényes funkcionális biztonsági szabványba, az ISO 26262<sup>8</sup>-ba nehezen integrálható.

Az autonóm katonai járművekkel szembeni alapvető elvárás, hogy a katonai- és a kapcsolódó támogató műveletek során csökkentse az élőerő veszteségeit [5] [18]. Az alkalmazás hatékonyságát komplex szenzorrendszerek növelik, ugyanakkor fontos megjegyezni, hogy napjainkban ezek a járművek alapvetően távirányítással működnek. Ezen járművek esetében az egyik legfontosabb osztályozási szempont a járművek autonómiájának foka, mely nemcsak hatékonysági, de etikai kérdéseket is felvet [4]. Hasonló aggályok merülhetnek fel egyéb felhasználási módok esetén is, ahol fenn áll nagymennyiségű adat feldolgozásának igénye.

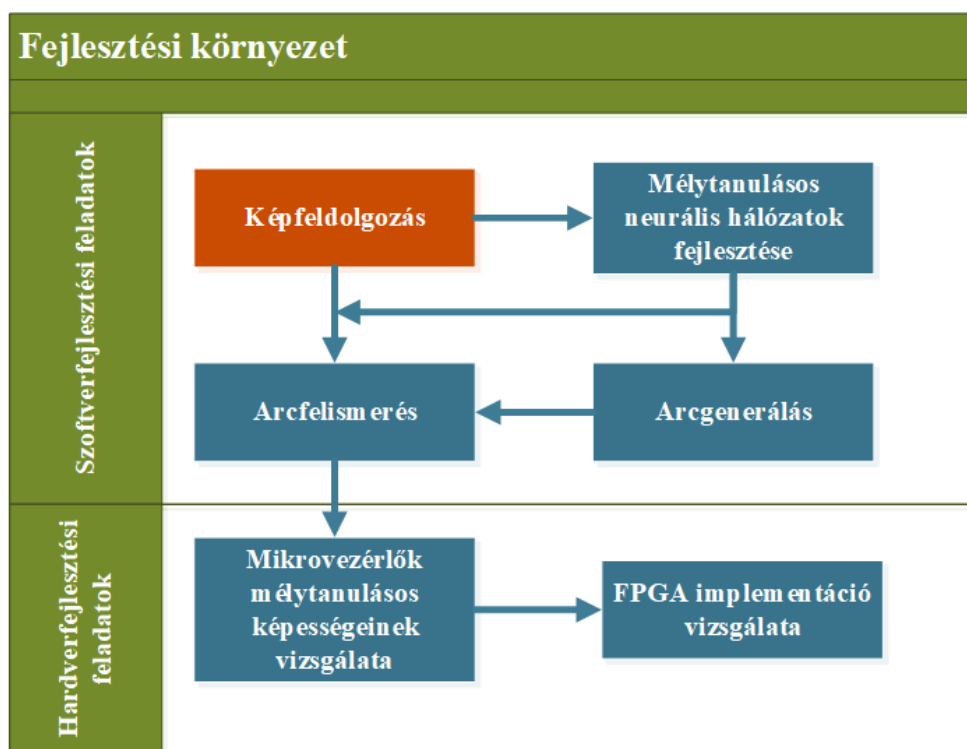
A mesterséges intelligencia széleskörű felhasználása több szempontból is megköveteli a technológiát érintő átfogó ismeretek meglétét. A célom egy olyan védelmi ipari rendszer prototípusának és fejlesztő környezetének kialakítása, mely alapot biztosít a mesterséges intelligenciát használó eszközök jellemző tulajdonságainak gyakorlati kutatására.

---

<sup>8</sup> ISO 26262: „Road vehicles – Functional safety” – Az International Organization for Standardization (ISO) által először 2011-ben meghatározott, forgalomban résztvevő gépjárművek elektronikus rendszereire vonatkozó biztonsági szabvány

## 2 A fejlesztés szoftverkönyezete

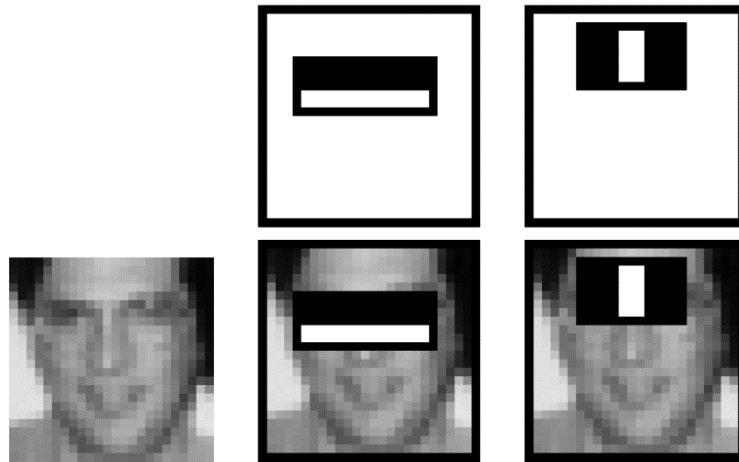
Az objektumdetektálás egyaránt kapcsolódik a képfeldolgozás (6. ábra), illetve a gépi látás (CV<sup>9</sup>) témaköréhez. Az arcfelismerés fő célja, hogy meghatározza, emberi arc látható-e egy adott képen. Napjainkban a telefonjaink kamerája képes az arcokat felismerni, ugyanakkor ebben a szegmensben az első nagy áttörésnek a 2001-ben publikált Viola-Jones algoritmust tekinthetjük. Az algoritmus az őt megalkotó két kutatóról kapta a nevét: Paul Viola-ról és Michael Jones-ról. Első értekezésük az eljárásról a „Rapid Object Detection using a Boosted Cascade of Simple Features” című cikkükben volt olvasható [19].



**6. ábra: Fejlesztési környezet, képfeldolgozás (szerk. a szerző)**

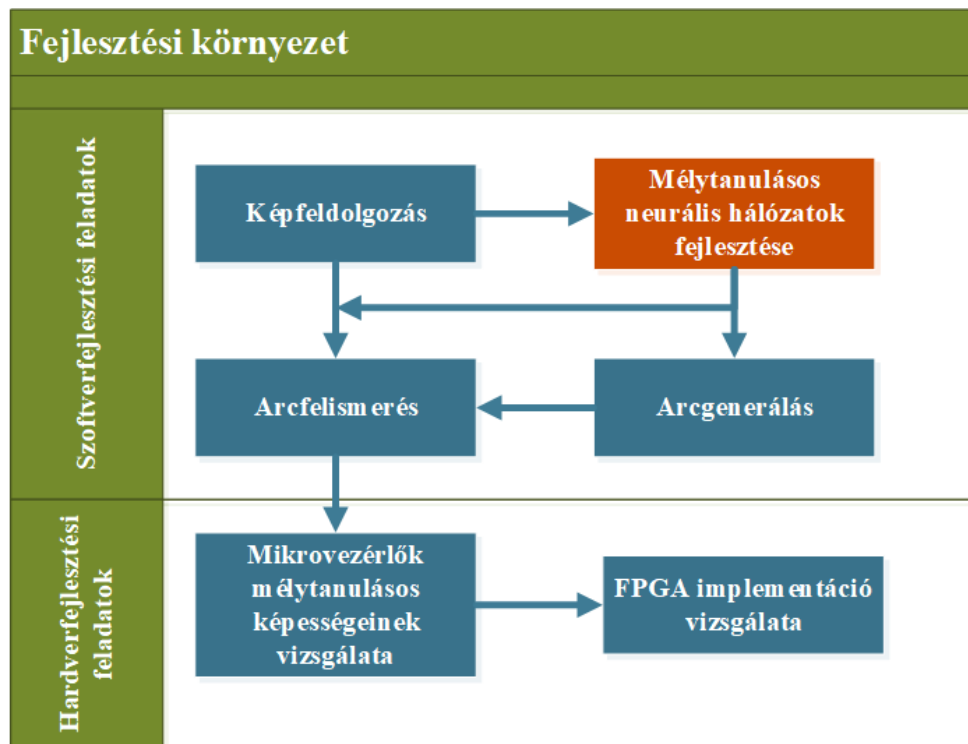
Bár egy igen régi eljárásról beszélünk és a betanítása is rendkívül lassú, a valós idejű arcdetektálási képessége még ma is gyorsnak mondható, mivel a képpontok integrálásával gyorsan kiszűri a nem arcnak minősített részeket [20]. Működésének lényege, hogy egy adott szürkeárnyaltos képet az algoritmus több kisebb szegmensre oszt és konvolúcióval mintákat keres (7. ábra). A jelenleg fejlesztés alatt álló rendszer tekintetében egyenlőre nincs tervben a Viola-Jones algoritmus használata, ugyanakkor a jövőben elképzelhető, hogy előfeldolgozási eljárásként még implementálásra kerül.

<sup>9</sup> CV: Computer Vision – Gépi látás



7. ábra: Viola Jones [19]

## 2.1 Deep learning keretrendszerek



8. ábra: Fejlesztési környezet, mélytanulási neurális hálózatok (szerk. a szerző)

A mélytanulási keretrendszerek teszik lehetővé, hogy komolyabb matematikai ismeretek nélkül, a projekt céljára fókuszálva fejlesszünk és tanítsunk mesterséges neurális hálózatokat.

Erre a célra több eszköz is rendelkezésre áll, melyek más-más előnyökkel és hátrányokkal rendelkeznek.

A Distbelief [21] esetén a keretrendszer kezelése Python szkriptekkel történt, előre definiált rétegek felhasználásával. Bár az így készült hálók több Google termékben is megjelentek, a keretrendszer korlátai miatt módosításokat kellett rajta végrehajtani. Ezen módosítások főbb területei:

- Új rétegek definiálásának lehetősége: A hatékonyság növelése miatt a DistBelief rétegek C++ osztályokba lettek szervezve, azonban az MI kutatók számára a réteg architektúrák kutatását jelentősen megnehezítette a C++ nyelv használata.
- Tanító algoritmusok hangolása: Nehézkes és korlátozott lehetőségek a tanítási eljárások optimalizálásában, továbbá a keretrendszer felépítése kizárja egyes tanító algoritmusok alkalmazását.
- Újabb tanító algoritmusok alkalmazása: A DistBelief felépítése miatt csak egyszerűbb feed-forward hálózatok tanítására alkalmas, komplexebb modelleknél, mint az RNN (Recurrent Neural Network), a GAN (Generative Adversarial Network) vagy a megerősítéses tanulás már nem működik.

A TensorFlow a DistBelief utódja [22], ingyenes, nyílt forráskódú gépi tanulási és mesterséges intelligencia szoftverkönyvtár. A Tensorflow beágyazott rendszerekre implementálható verziója a TensorFlow Lite, melyről még szó lesz a későbbi fejezetekben.

A Caffe-t [23] a Berkeley Vision and Learning Center és az aktív Github-os közössége fejlesztette ki. Programozási modellje hasonló a DistBelief-hez, azaz viszonylag egyszerű a meglévő rétegekből modellt alkotni, de körülményes az újabb rétegek hozzáadása.

Az itt tárgyalt keretrendszerek közül felépítését tekintve a Theano áll a legközelebb a Tensorflow-hoz [24]. A Theano lényegében egy Python könyvtár, ami többdimenziós tömbök gyors és hatékony számítását teszi lehetővé.

A Pytorch [25] egy nyílt forráskódú gépi tanulós keretrendszer. A Tensorflow-val, Caffe-val és a Theano-val ellentétben lehetséges finomhangolni a végrehajtási sorrendet és a memória felhasználást, ami akár jelentősebb optimalizációk végrehajtására is lehetőséget teremt. Ez kutatási szempontból előny, ugyanakkor az üzemszerű használatot és a tanított modell telepítését nehezíti.

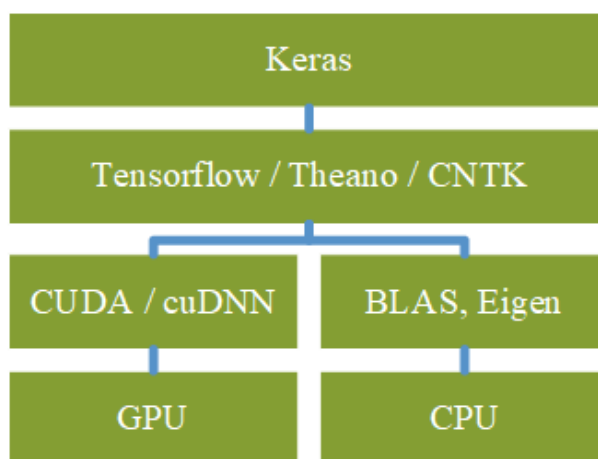
A Microsoft Cognitive Toolkit (korábban CNTK), a Microsoft nyílt forráskódú keretrendszere [26]. Előnye a számítási teljesítményben rejlik, mivel a Microsoft kapcsolatban áll az Nvidia-val, ezért a GPU<sup>10</sup> támogatásnál néha érzékelhető előnyökkel bír. A CNTK volt

---

<sup>10</sup> GPU: Graphics Processing Unit – Videókártya

az első mélytanulási keretrendszer, ami az azóta egységes ONNX (Open Neural Network Exchange) formátumot használta. Ezzel lehetségessé vált a különböző keretrendszerekben betanított neurális hálók interoperabilitásának biztosítása [27].

A Keras egy modell szintű könyvtár, ami magasabb szintű építőelemeket tartalmaz, deep learning modellek készítéséhez [28]. Alacsonyabb, tenzor szintű műveletekkel nem foglalkozik, ehhez más, optimalizált backend-eket használ (9. ábra). Háromféle ilyen backendet lehet a Keras-hoz használni: Tensorflow (alapértelmezett), Theano, CNTK.



9. ábra: Keras felépítése (szerk. a szerző)

A MATLAB Deep Learning Toolbox az előzőekben felsorolt keretrendszerektől lényegesen eltérő, nem ingyenes, ugyanakkor teljeskörű fejlesztőkörnyezetet biztosít. A modell alapú fejlesztési eljárás segítségével képességeit az algoritmus tervezésétől a célhardveres kódgenerálásig fel lehet használni.

## 2.2 MATLAB és a modell alapú fejlesztés

A többi keretrendszerrel tehát némiképp eltér a Mathworks cég által fejlesztett MATLAB, amely tulajdonképpen egy programozási nyelv és fejlesztő környezet egyben, elnevezése pedig a MATrix LABoratories-ból ered. Gyengén típusos nyelv, szintaxisát és felhasználási területét illetően a Python-hoz lehet hasonlítani, viszont lényegesen több eszközt, jobb használhatóságot és támogatást biztosít.

Nem ingyenes és nem nyílt forráskódú, nagy előnye viszont, hogy támogatja a modell alapú fejlesztést. A program egyes kiegészítői toolbox-okba vannak szervezve, melyek egymásra épülhetnek, és külön-külön kell megvásárolni azokat.

A dolgozat tárgyát képező kutatás célja, hogy transfer learning-et használva egy beágyazott rendszeren autonóm módon futó beléptető rendszert hozzunk létre. A projekt igen szerteágazó, egyes részei is igen elmélyült szaktudást igényelnek, így a modell alapú fejlesztési munkafolyamat nem lehetőség, hanem alapvető követelménye a projekt sikeres végrehajtásának.

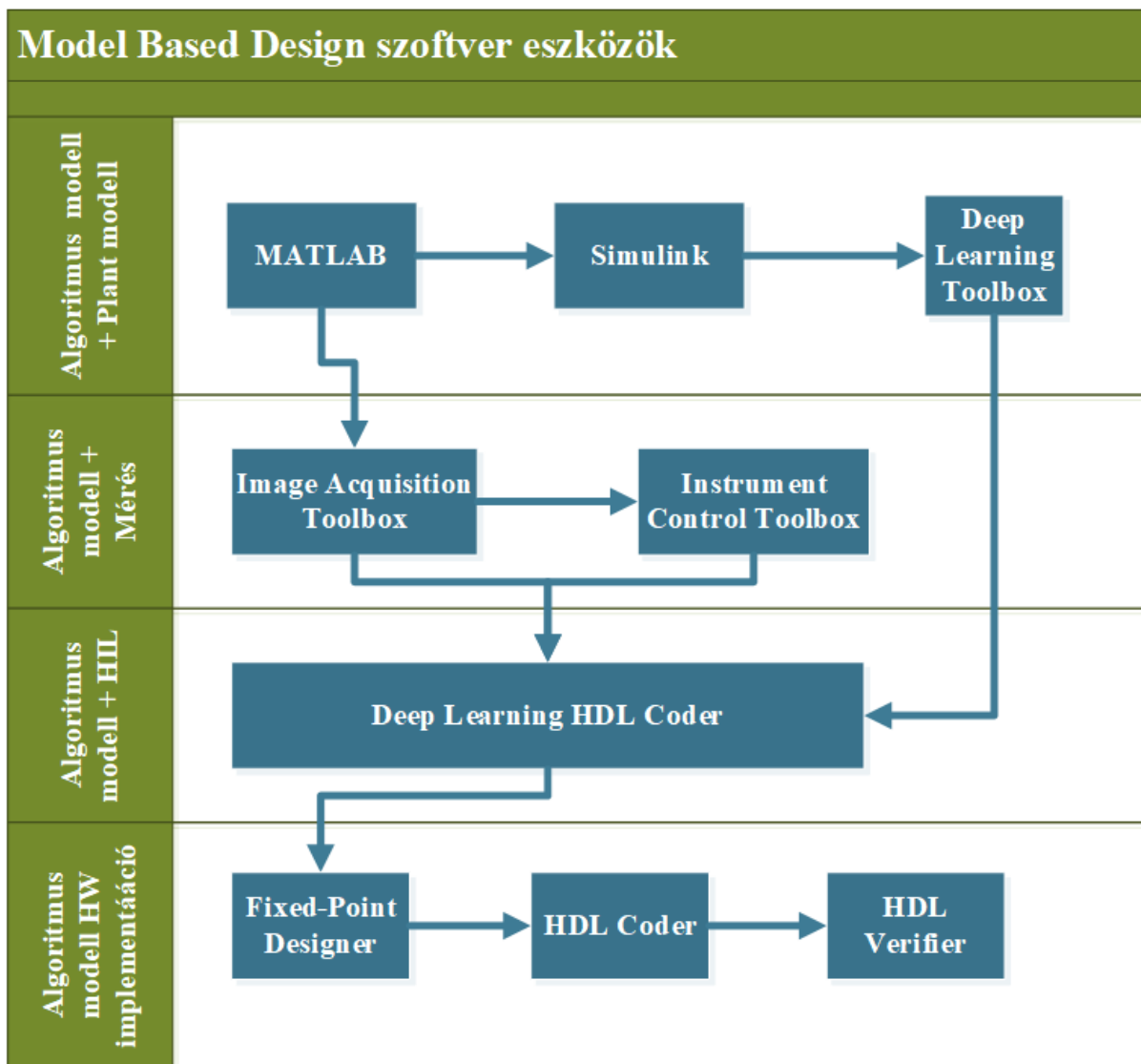
A modell alapú fejlesztés (MBD<sup>11</sup>) értelmezése jelen projekt keretein belül a szimuláció alapú fejlesztés, a részeredmények folyamatos validálása és az automatizált munkamenet kialakítása köré épül. A projekt indulásakor szimulált mérési környezetet (plant model) alakítunk ki a rendszermagunk köré (algorithm model). Jelen esetben a plant model egy meglévő arckép adatbázis, a fizikai perifériák pedig változók. Az algoritmus modellünk a transfer learning-gel újratanított mélytanulós konvolúciós neurális hálózat, a futtatási környezet pedig MATLAB/Simulink. Hasonló, elméleti és hardverimplementációs feladatokat átfogó kutatási-fejlesztési projektekre igen elterjedt ennek a környezetnek a használata [29]. Az előfeldolgozási feladatokat a jel-, kép- és videó feldolgozást segítő eszköztárakkal végezzük. A tanítás kivitelezésére a Deep Learning toolbox adja a keretrendszert. Amennyiben olyan korlátokba ütköznénk, melyet a keretrendszer szab meg, ONNX formátumba exportálással lehetőség van az egyes munkafázisokat egyéb keretrendszeren elvégezni. Jelen pillanatban a GPU-n történő tanítási folyamat jelent akadályt, amit a beszerzésre váró fordítók és toolbox-ok orvosolni fognak.

A fejlesztés első mérföldkövénél valós mérési adatokkal, kameraképekkel folytatjuk a munkát az Image Acquisition és Instrument Control toolbox-ok segítségével. Ezen a ponton a plant model kilép a szimulációs térből. Amennyiben az eredmények megfelelőek, további lépésként megkezdjük a hardver implementációt a Deep Learning HDL toolbox-szal. Ebben a fázisban párhuzamosan fut az algoritmus modell a PC-n és az FPGA-n, így lehetőség nyílik a hardverimplementáció okozta pontosságvesztés kiküszöbölésére. A végső cél az algoritmus teljes implementálása FPGA-ra, melyhez nélkülözhetetlen a fixpontosítás, illetve a kódgenerálásnak a nyomon követése, analizálása a szintézertől a routing-ig.

A felhasznált szoftvereszközök feladatai egymásra épülnek és automatizálhatók. Ennek segítségével, amennyiben egy korábbi fejlesztési fázisban módosításra van szükség, az egyszer már beállított paraméterekkel lehetőség van automatizált kódgenerálással a részfeladatokat újrafordítani. Ehhez, illetve a fejlesztés felgyorsításához több alárendelt toolbox-ra is szükség van.

---

<sup>11</sup> MBD: Model Based Design – Modell alapú fejlesztés



10. ábra: Modell alapú fejlesztéshez felhasznált szoftverek (szerk. a szerző)

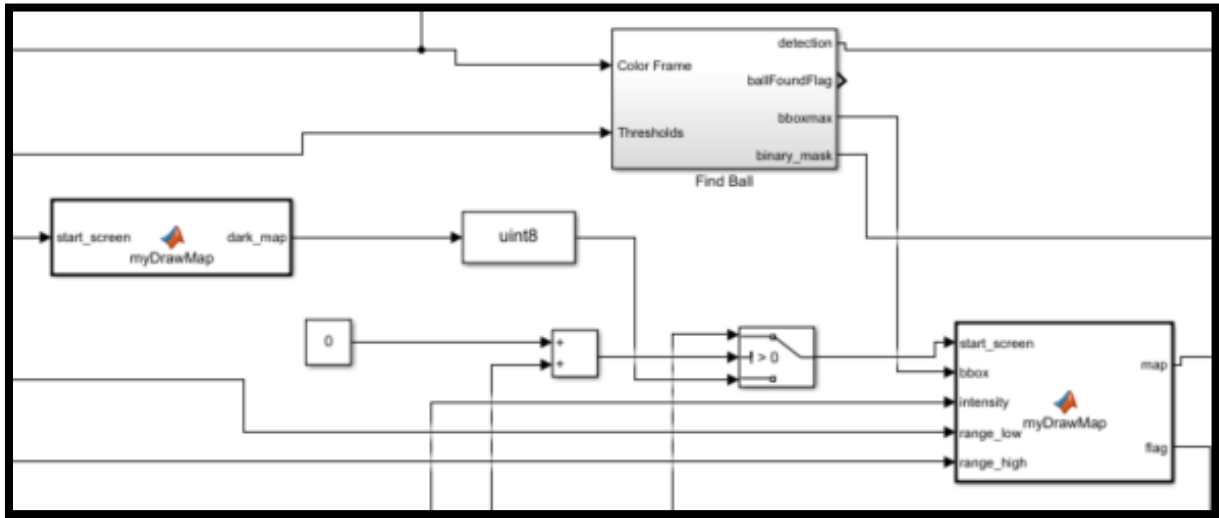
### MATLAB Coder

C, C++ kódgenerálást végez többféle hardveres platformra [30], személyi számítógéptől a beágyazott rendszerekig. Több toolbox függőséggel is rendelkezik, így a Deep learning, GPU Coder, illetve Hardware-in-the-loop (HIL<sup>12</sup>) tesztelésekhez is szükséges. A generált kód meglévő projektben felhasználható forráskódként, statikus-, illetve dinamikus könyvtárként.

<sup>12</sup> HIL: Hardware In the Loop – Hardvereszköz szimulációba importálása

## Simulink

A MATLAB-ra épülő grafikus fejlesztői környezet [31] kifejezetten modell alapú fejlesztéshez használható. A Simulink MBD lényege, hogy komplex rendszerek tervezésekor a munkafolyamat során végig modelleket használunk (11. ábra).



**11. ábra: Simulink modell (szerk. a szerző)**

Előnyei:

- Virtuális modellek használata a szimulációhoz és teszteléshez a fejlesztés korai szakaszától;
- Fejlesztés validálása fizikai modellek segítségével és HIL teszteléssel;
- A termék szintű kódgenerálás beágyazott rendszerekre: C, C++, CUDA<sup>13</sup>, PLC<sup>14</sup>, Verilog<sup>15</sup> és VHDL<sup>16</sup>;
- Modellek kiterjesztése rendszerekké hiba analízishez és megelőzőes karbantartáshoz.

## Simulink Coder

A Simulink Coder feladata a modellből történő C és C++ kódgenerálás [31], amely PC-n és beágyazott rendszereken egyaránt futtatható. Valós és nem valós idejű futtatásra, gyors prototipizálásra és hardware-in-the loop tesztelésre egyaránt használható támogatott beágyazott rendszereken.

<sup>13</sup> CUDA: Compute Unified Device Architecture – Általános célú számításokat tesz lehetővé a támogatott Nvidia videokártyákon

<sup>14</sup> PLC: Programmable logic controller – Programozható logikai vezérlő

<sup>15</sup> Verilog: Hardverleíró programozási nyelv

<sup>16</sup> VHDL: VHSIC Hardware Description Language – Nagyon nagysebességű integrált áramköri hardverleíró programozási nyelv

## Parallel Computing Toolbox

Intenzív számítási és adatfeldolgozási feladatok megoldására használható [32], többmagos processzorok, GPU-k illetve klaszterek megléte esetén. A háttérben a feladatok delegálását automatikusan végzi a CPU<sup>17</sup>-k között, Nvidia videokártya megléte esetén pedig oda allokálja a számításokat anélkül, hogy CUDA-ban kellene megírni az ehhez szükséges függvényeket. Deep learning esetén a tanítás is delegálható GPU-ra, ami gyakorlatilag nélkülözhetetlen mélytanulós rendszereknél.

## GPU Coder

Optimalizált CUDA kód generálása MATLAB-ból és Simulinkből [33]. A generált kód optimalizált CUDA függvényhívásokat végez TensorRT<sup>18</sup>-ből, cuDNN<sup>19</sup>-ből, cuFFT<sup>20</sup>-ből, cuSolver<sup>21</sup>-ből illetve cuBLAS<sup>22</sup>-ből. A generált kód meglévő projektekbe integrálható forráskódként, statikus-, illetve dinamikus könyvtárakként. A futtatási környezet lehet Nvidia videokártya, beágyazott Nvidia Jetson, Nvidia Drive vagy egyéb platform.

## Signal Processing Toolbox

Mintavételezett jelek analizálásához, előfeldolgozásához és tulajdonságok kinyeréséhez használható függvényeket és programokat tartalmaz [34]. Lehetőséget biztosít szűrők tervezéséhez, analizálásához, újra mintavételezéshez, simításhoz, referencia átrendezéshez és teljesítmény spektrum becsléshez. A Signal Analyzer programmal lehetőség nyílik több jel egyidejű vizsgálatára idő, frekvencia és idő-frekvencia tartományban, illetve jelszakaszok kinyerésére. a Filter Designerrel lehetőség nyílik digitális szűrők tervezésére és vizsgálatára.

## Image Processing Toolbox

Átfogó képfeldolgozási függvénygyűjteménnyel rendelkezik, illetve több, képfeldolgozási munkafolyamatot támogató applikációt tartalmaz képfeldolgozással, analízissal, vizualizációval és algoritmus fejlesztéssel kapcsolatban [35]. Segítségével gyors és kézenfekvő lehetőség nyílik a képszegmentálásra, feljavításra, zajcsökkentésre, geometriai

---

<sup>17</sup> CPU: Central Processor Unit

<sup>18</sup> TensorRT – CUDA-ra épülő szoftverfejlesztői környezet

<sup>19</sup> cuDNN: CUDA Deep Neural Network – CUDA-ra épülő mélytanulós szoftverkönyvtár

<sup>20</sup> cuFFT: CUDA Fast Fourier Transform – CUDA-ra épülő gyors Fourier-transzformáció

<sup>21</sup> cuSolver – CUDA optimalizált egyenletrendszer megoldó

<sup>22</sup> cuBLAS: CUDA Basic Linear Algebra Subroutine – CUDA-ra épülő lineáris algebrai műveleteket tartalmazó könyvtár

transzformációkra címkézésre és 3D képfeldolgozásra. Segítségével lehetőség van automatizálni a munkafolyamatokat, interaktív képmanipulációt végezni és az eredményeket összehasonlítani. Támogatja a GPU felhasználást, illetve a C/C++ kódgenerálást.

### Computer Vision Toolbox

Ez az eszköz 2D/3D gépi látás és videó feldolgozási eljárások fejlesztéséhez és teszteléséhez szükséges algoritmusokat, függvényeket és applikációkat tartalmaz [36]. Főbb felhasználási területek az objektum detektálás, követés, jellemzők detektálása, kinyerése, illetve összehasonlítása. Lehetőséget biztosít automatizált kalibrációs munkafolyamatok inicializálására hagyományos, sztereo és fisheye kamerákra. Segítségével a képfeldolgozási rendszerbe integrálhatók egyes saját tanítású DNN architektúrák. Támogatja a GPU felhasználást, illetve a C/C++ kódgenerálást.

### Statistics and Machine Learning Toolbox

Adat analízis, valamint felügyelt, részlegesen felügyelt és felügyelet nélküli gépi tanulással kapcsolatos függvényeket és adat analízis megjelenítőket tartalmaz [37]. Jelen tanulmány fókuszában a regressziós és osztályozó algoritmusok mellett a toolbox olyan egyéb, optimalizált gépi tanulási módszereket tartalmaz, melyeket Simulink-ben is lehet alkalmazni.

### Deep Learning Toolbox

Mélytanulási keretrendszer, amely rendelkezik grafikus szerkesztővel (Deep Network Designer), illetve adatanalízis funkciókkal [38]. További jelentős előnye a modell alapú fejlesztési lehetőség, egyszerűen alkalmazható megjelenítő és analízis funkciók, kódgenerálási képesség, hardver implementálási lehetőségek, HIL tesztek. A toolbox tartalmazza még az Experiment Manager-t, mely egyfajta dokumentációs rendszerként segít nyomon követni több, párhuzamosan fejlesztett mélytanulási kutatás tanítási paramétereinek alakulását, analízisét, illetve vizualizálja a tanítási folyamatot és az egyes rétegek működését.

### Image Acquisition Toolbox

MATLAB-hoz és Simulink-hez biztosít kamera csatlakoztatási lehetőséget [39]. Lehetőséget biztosít továbbá a hardverek automatizált vezérlésére, mérőrendszer kialakítására. Támogatja

az elterjedt szabványokat, mint például az USB3 Vision<sup>23</sup>, GigE Vision<sup>24</sup>, GenICam<sup>25</sup> és GenTL<sup>26</sup>.

### Instrument Control Toolbox

Külső eszközök, műszerek, szenzorok, számítógépek és egyéb beágyazott rendszerek csatlakoztatását közvetlenül teszi lehetővé [40], USB<sup>27</sup>, RS232<sup>28</sup>, TCP/IP<sup>29</sup>, GPIB<sup>30</sup>, Modbus<sup>31</sup>, I2C<sup>32</sup>, SPI<sup>33</sup> interfészen keresztül. Segítségével a modell alapú fejlesztési munkafolyamat szimulációs fázisából egyszerűen át lehet állni a mérés alapú fázisra. Amennyiben a mérőrendszer egyes elemei kicserélődnek más típusokra, az átállás nem jelent különösebb időbeli veszteséget.

### Deep Learning HDL Toolbox

Eszközöket és függvényeket biztosít mélytanulós hálózatok Xilinx és Intel FPGA-ra és SoC<sup>34</sup>-re történő implementálására [41]. Funkciója, hogy a betanított hálózatokat HIL futtatás keretein belül, megadott FPGA kártyán tesztelni tudjuk. A kódgenerálás a háttérben történik, a kimenet egy „.bit” fájl. Ez egy gyors és hatékony módszer neurális hálók hardver prototipizálására.

### Fixed-Point Designer

Adattípusokat és eszközöket biztosít beágyazott rendszerekre szánt algoritmusok változóinak lebegőpontosról fixpontossá konvertálásában [42]. Lehetőség van valós idejű összehasonlításra

---

<sup>23</sup> USB Vision – European Machine Vision Association (EMVA) által szabványosított USB-s csatolófelület ipari kamerákhoz, 2013-ban jelent meg

<sup>24</sup> GigE Vision - European Machine Vision Association (EMVA) által szabványosított s ethernet alapú csatolófelület ipari kamerákhoz, 2006-ban jelent meg

<sup>25</sup> GenICam: Generic Interface for Cameras – European Machine Vision Association (EMVA) által szabványosított programozási felület ipari kamerákhoz

<sup>26</sup> GenTL: European Machine Vision Association (EMVA) által szabványosított GenICam átviteli rétege

<sup>27</sup> USB: Universal Serial Bus – USB Implementers Forum (USB-IF) által először 1996-ban szabványosított soros adatátviteli csatorna

<sup>28</sup> RS232 - Electronic Industries Association (EIA) által először 1960-ban bemutatott szabványos soros adatátviteli csatorna

<sup>29</sup> TCP/IP – Az Amerikai Védelmi Minisztérium által először 1982-ben szabványosított számítógép hálózati protokollcsomag

<sup>30</sup> GPIB: General Purpose Interface Bus – Az Institute of Electrical and Electronics Engineers (IEEE) által először 1960-ban IEEE-488 néven szabványosított párhuzamos adatkommunikációs busz

<sup>31</sup> Modbus –Az iparban de facto szabvánnyá vált, 1979-ben kifejlesztett adatkommunikációs protokoll

<sup>32</sup> I2C: Inter-Integrated Circuit – A Philips által kifejlesztett, 1992-ben de facto szabvánnyá vált soros kommunikációs protokoll

<sup>33</sup> SPI: Serial Peripheral Interface – A Motorola által 1979-ben kifejlesztett, de facto szabvánnyá vált soros kommunikációs protokoll

<sup>34</sup> SoC: System on a Chip – Nincsen konkrét definíció, olyan integrált áramkör, mely képes egy asztali számítógép feladatainak ellátására

a kvantálási hatások, túlcordulások és pontosság tekintetében. Olyan programokat biztosít, melyekkel duplapontosságú lebegőpontos számítást lehet csökkentett pontosságú fixpontossá konvertálni a kívánt optimalizáltság és hibatűrés automatizált vizsgálatával. Fixpontosított algoritmusok C és HDL kód generálása közvetlenül lehetséges.

### HDL Coder

Szintetizálható VHDL és Verilog kódot generál MATLAB függvényekből, Simulink modellekből, illetve Stateflow<sup>35</sup> gráfokból [43]. A generált kód FPGA fejlesztéshez vagy ASIC<sup>36</sup> prototipizáláshoz is használható. A HDL Coder tartalmazza a Workflow Advisor-t, ami Xilinx, Intel és Microsemi FPGA-k automatizált programozását segíti elő. Az implementálási folyamat kontrollálható, a kritikus utak vizsgálhatók, de a szoftver erőforrás becslést is végez. Az eszköz lehetőséget ad a DO-254<sup>37</sup>-es szabvány szerinti fejlesztéshez.

### HDL Verifier

Debug<sup>38</sup> és RTL<sup>39</sup> testbench<sup>40</sup> szimulációs lehetőséget biztosít a generált Verilog és HDL kódon [44]. Koszimuláció esetén az FPGA fejlesztőrendszerhez is használt szimulátort használja, ezen felül lehetőség van logikai analízátor szondákat és trigger jeleket inicializálni az FPGA-n belül, és ezek működését a belső memóriaregisztereken keresztül grafikus felületen monitorozni.

A fent ismertetett szoftverkörnyezet biztosítja azt a hátteret, melynek segítségével egy jól skálázható, rugalmasan fejleszthető rendszer kialakítására lehetővé válik.

---

<sup>35</sup> Stateflow: A Simulink egyik eszközcsomagja, állapotgépek, gráfok grafikus tervezésére szolgál

<sup>36</sup> ASIC: Application-Specific Integrated Circuit – Specifikusan, adott célra fejlesztett integrált áramkör

<sup>37</sup> DO-254: A Federal Aviation Administration (FAA) által először 2005-ben meghatározott kritériumrendszer, mely a légiközlekedésben résztvevő elektronikai berendezésekre vonatkozik

<sup>38</sup> FPGA debug: Az FPGA-ba tesztelési célból implementált logikai analízátor használata

<sup>39</sup> RTL: Register-Transfer Level – Programozható logikai integrált áramkörök alapáramköri programozási módja

<sup>40</sup> RTL testbench – HDL nyelvről RTL-re lefordított áramköri hálózat szimulációja

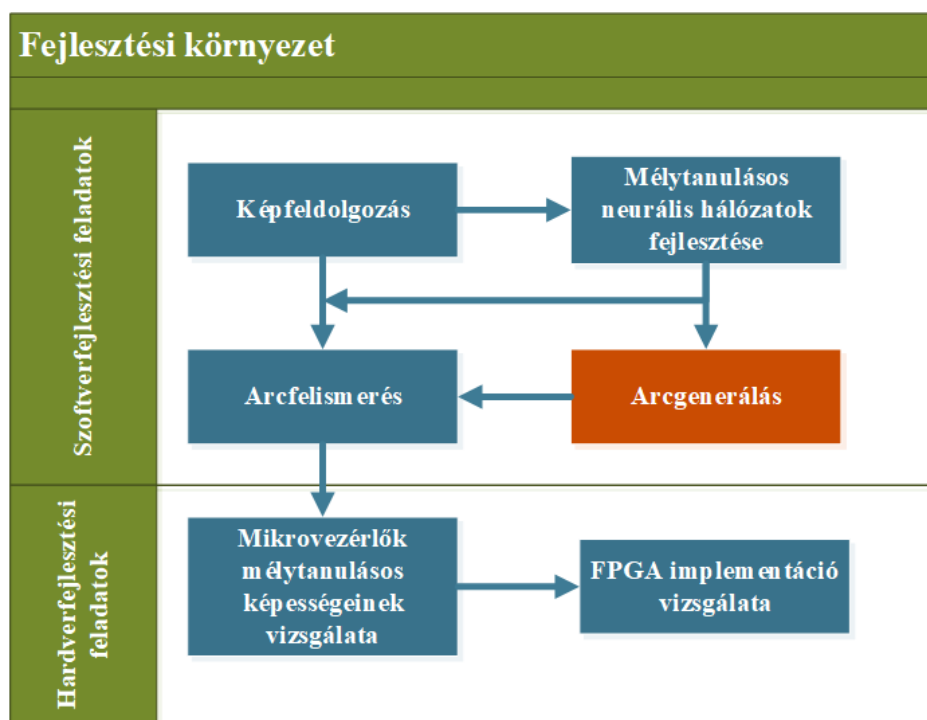
### 3 Arcfelismerő rendszer fejlesztése

Jelen dolgozatban tárgyalt kutatás célja, hogy alternatívát kínáljon az objektumvédelemre széles körben elterjedt autentikálási megoldásokkal szemben egy nagy megbízhatóságú arcfelismerő rendszer formájában, amely integrálásra kerülhet majd egy valós beléptető rendszerbe. Maga a fejlesztés több lépcsőben zajlik a rendelkezésre álló erőforrások függvényében, melynek köszönhetően mostanáig az alábbi eredmények születtek:

- Futtatási környezet kialakítása;
- Mintaadatok generálása;
- Transfer learning megvalósítása.

Az EU-ban hatályos GDPR<sup>41</sup> szabályozás miatt, ami szigorúan szankcionálja a hozzájárulás nélküli személyes adatfelhasználást, a fejlesztési folyamat egyik fontos kezdeti részfeladata a Tensorflow szoftverkönyvtár segítségével végrehajtott arcgenerálás. A véletlenszerű emberi arcok generálását Generative Adversarial Network-kel (GAN) végeztem (12. ábra) a CelebA nyilvánosan elérhető adatkészlet segítségével.

#### 3.1 Keras DCGAN arcgenerátor



12. ábra: Fejlesztési környezet, arcgenerálás (szerk. a szerző)

<sup>41</sup> GDPR: General Data Protection Regulation – Az Európai Unió által 2016-ban megfogalmazott általános adatvédelmi szabályozás

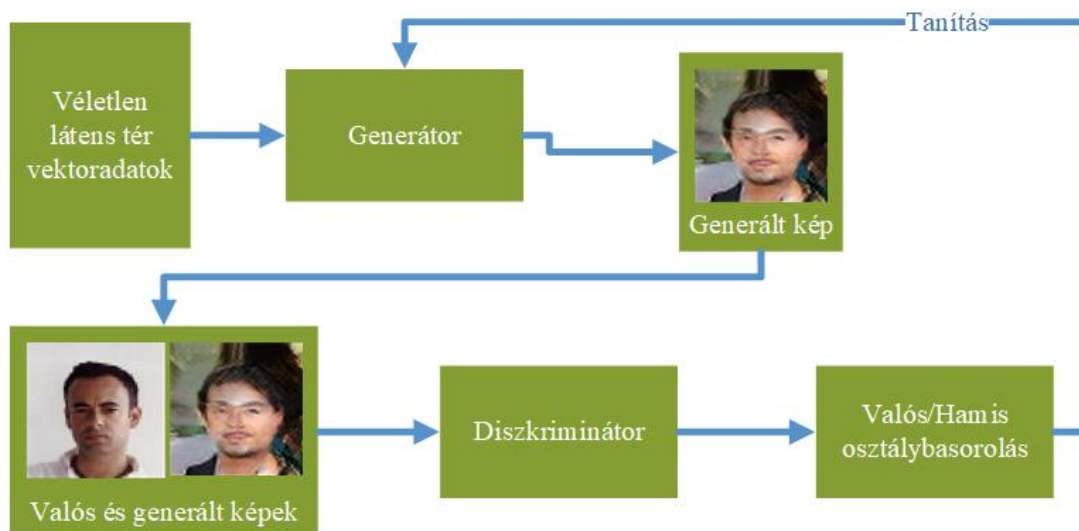
A képgeneráláshoz felhasznált GAN (Generative Adversarial Network) módszert 2014-ben Goodfellow és társai fejlesztették ki [45]. A dolgozatban felhasznált DCGAN (Deep Convolutional Generative Adversarial Network) mélytanulásos neurális hálózatot Francois Chollet készítette [46], aki egyszemélyben a Keras megalkotója is. A python kód a Google Drive-ról, a CelebA adatkészletből 202599 darab, színészek arcát tartalmazó gyűjteményből dolgozik. Ez szolgálja a tanítási adathalmazt a DCGAN hálózathoz, mely felépítését tekintve két részből áll: a generátorból és a diszkriminátorból (14. ábra). Működésének a lényege, hogy a két hálózat egymással szemben dolgozik, a generátor addig finomítja az arcvonásokat, míg a diszkriminátor azokat el nem fogadja valódi képnek [47].

A generátor egy 8192 méretű tömböt fogad, melynek összetétele egy kép, illetve a látens térből származó zaj, melyekből 128 darab feature map-et képez 16x16-os méretben. A bemenet így egy teljesen bekötött perceptron réteg, míg a kimenet egy konvolúciós réteg, ami 3db 64x64-es képet generál meghívásonként (13. ábra).

| Model: "generator"                   |                     |         |
|--------------------------------------|---------------------|---------|
| Layer (type)                         | Output Shape        | Param # |
| dense_1 (Dense)                      | (None, 8192)        | 1056768 |
| reshape (Reshape)                    | (None, 8, 8, 128)   | 0       |
| conv2d_transpose (Conv2DTranspose)   | (None, 16, 16, 128) | 262272  |
| leaky_re_lu_3 (LeakyReLU)            | (None, 16, 16, 128) | 0       |
| conv2d_transpose_1 (Conv2DTranspose) | (None, 32, 32, 256) | 524544  |
| leaky_re_lu_4 (LeakyReLU)            | (None, 32, 32, 256) | 0       |
| conv2d_transpose_2 (Conv2DTranspose) | (None, 64, 64, 512) | 2097664 |
| leaky_re_lu_5 (LeakyReLU)            | (None, 64, 64, 512) | 0       |
| conv2d_3 (Conv2D)                    | (None, 64, 64, 3)   | 38403   |
| =====                                |                     |         |
| Total params: 3,979,651              |                     |         |
| Trainable params: 3,979,651          |                     |         |
| Non-trainable params: 0              |                     |         |

**13. ábra: A DCGAN Generátor hálózatának felépítése (szerk. a szerző)**

A diszkriminátor ezt a generált kimenetet olvassa be az eredeti képekkel együtt, a kimenetet pedig a bemenetre kapott kép osztályba sorolása (14. ábra).



**14. ábra: A DCGAN működése (szerk. a szerző)**

Ahhoz, hogy a DCGAN-al arcképeket generálhassunk, igen nagy erőforrások szükségesek. A Google Collaboratories néhány órányi ingyenes számolási időt biztosít, processzoron, GPU-n és TPU-n egyaránt, azonban ezek rendelkezésre állása egyrészt erősen napszakfüggő, másrészt időben sem elégséges, mivel több napot igénybe vevő folyamatról van szó. Emiatt szükségessé vált a saját eszközön történő számolás megvalósítása.

### 3.1.1 Fejlesztői környezet kialakítása Docker-ben

Mivel a Tensorflow használata különböző szoftver beállításokat igényel, ezért célszerű magát a python kódot Docker-ben futtatni. Erre azért van szükség, mert így a számítógépünk operációs rendszerében nem végzünk el kritikus beállításokat, hanem egy látszólagos virtuális környezetben futtatjuk kódunkat, így elkerülhetjük a számítógépünk operációs rendszerének, illetve egyéb összetevőinek sérülését.

Maga a Docker egy számítógépes program, amely operációs rendszer szintű virtualizációt végez. Image-eket használ, ami egy lecsupaszított operációs rendszer magot tartalmaz egy minimális függvénykönyvtárral. Ebből konténereket generál, amiket a Docker szoftver futtat. A konténerek egymástól elkülönülnek, és saját alkalmazást, eszközöket, könyvtárakat és konfigurációs fájlokat kötnek össze, jól meghatározott csatornákon keresztül kommunikálva egymással.

#### Futtatási módok

Mivel a véletlenszerű emberi arcok generálása során, 1 epoch számítása is órákat vesz igénybe, ezért különböző futtatási módokat kell használnunk. Ez abban nyilvánul meg, hogy a kódunkat

CPU-ban futtatjuk, vagy a számítógép videokártyájában lévő GPU-ban. A számítógép CPU-ja X (2, 4, 8, stb) magot tartalmaz, így a python kódunkat ebben az X CPU magban tudjuk csak futtatni. A számítógépünk videokártyájában nagyságrendekkel több mag (core) található.

Mivel a Tensorflow függvénykönyvtár nagyrészt mátrix számításokat végez, hasonlóan egy videó, vagy egy grafikus animáció megjelenítéséhez, ezért mátrix kalkulációhoz kiválóan használható a számítógép videokártyájának GPU-ja.

### Számítógép leírása

- OPERÁCIÓS Rendszer: Linux Lite 5.8. Ez egy Ubuntu Linux alapú operációs rendszer;
- CPU: Intel XEON E-2200;
- Videókártya: Nvidia GT730.

### Felhasznált szoftverek

- Docker: Docker version 20.10.12, build e91ed57;
- Docker compose: v2.0.1;
- nvidia-docker2:  $\geq 2.8.0$ ;
- NVIDIA driver: Latest Production Branch Version: 510.54.

### Docker telepítése

A futtatási környezet kialakítása (**Hiba! A hivatkozási forrás nem található.**) egyaránt ámaszt hardveres és szoftveres követelményeket. Mivel a telepítés nem magától értetődő, a következőkben az elvégzendő feladatok kerülnek részletezésre.

### A szoftverütközések elkerülése miatt fontos a korábbi verziók törlése:

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

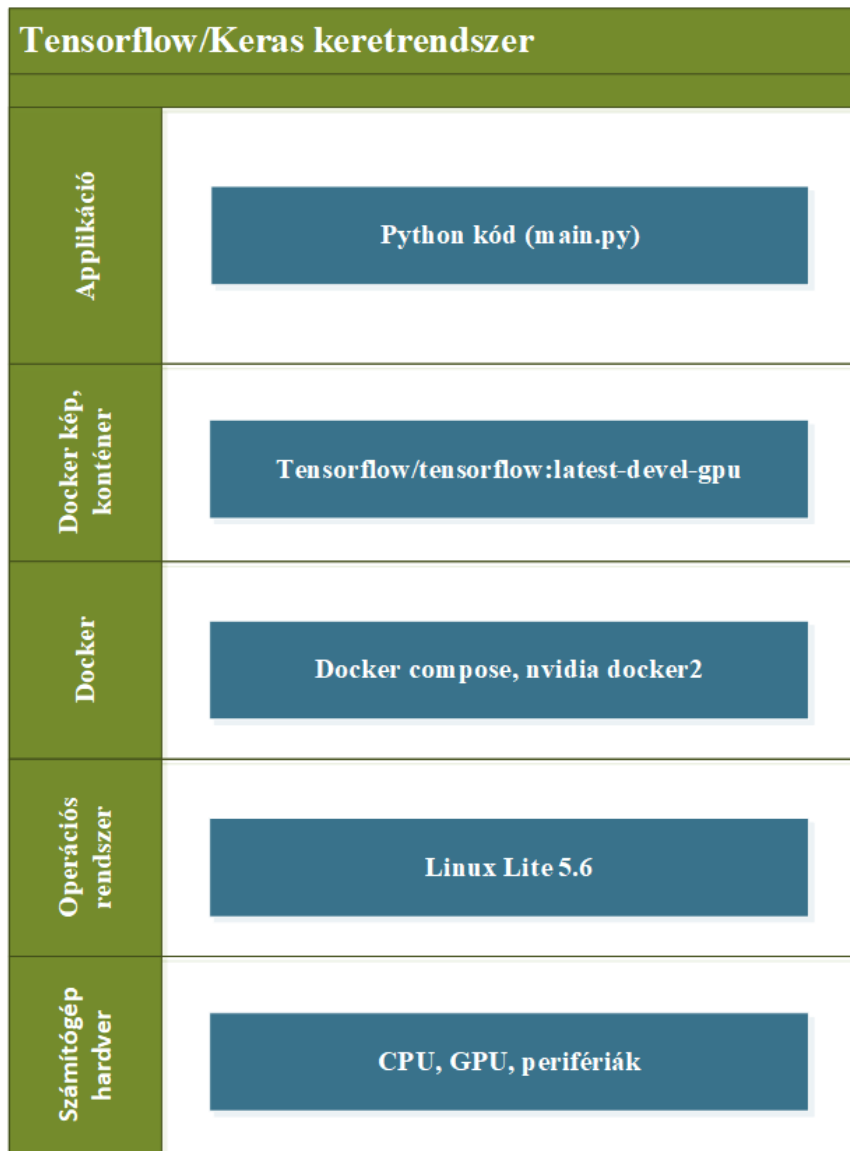
### Repository beállítása:

```
sudo apt-get update

sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

### Hivatalos Docker GPG key hozzáadása a linux csomagkezelőhöz:

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --
dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```



**15. ábra: Tensorflow/Keras keretrendszer kialakítása (szerk. a szerző)**

Stabil, végleges szoftver repository csatorna beállítása:

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/debian \
$(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
```

Docker engine telepítése:

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Telepítés tesztelése:

```
docker --version
Docker version 20.10.12, build e91ed57
sudo docker run hello-world
```

## Docker Compose telepítése

A Docker Compose szoftver kiegészítő csomagra azért van szükség, mert maga a docker képek, konténerek futtatása nagyon bonyolult kapcsolókon keresztül történik. A docker compose egy egyszerűsítést hajt végre, amit egy „.yaml” fájlban ír le (docker-compose.yml). Ennek a fájlnek a kezeléséhez szükséges a docker compose V2 szoftver.

Docker Compose telepítése, könyvtár beállítása és Docker Compose letöltése:

```
DOCKER_CONFIG=${DOCKER_CONFIG:-/usr/local/lib/docker/cli-plugins}  
mkdir -p $DOCKER_CONFIG/cli-plugins  
curl -SL  
https://github.com/docker/compose/releases/download/v2.2.3/docker-  
compose-linux-x86_64 -o $DOCKER_CONFIG/cli-plugins/docker-compose
```

A DOCKER\_CONFIG változó beállítása különösképpen fontos, alapértelmezetten a ~/.docker/cli-plugins van beállítva. Ez nem megfelelő, mert így a compose szoftver csomagot csak egy felhasználó használhatná, így a root felhasználó sem. Mivel a docker az operációs rendszerünk olyan funkcióit is használni fogja, ami csak root jogosultsággal érhető el, ezért fontos, hogy a docker compose szoftver csomag globális hozzáférést kapjon az operációs rendszerhez és a docker szoftverhez is. Ezért a compose csomagot a következő könyvtárba kell letölteni:

```
/usr/local/lib/docker/cli-plugins
```

Futtatási „job” hozzáadása docker compose-hoz:

```
chmod +x $DOCKER_CONFIG/cli-plugins/docker-compose
```

Docker compose tesztelése:

```
docker compose version  
Docker Compose version v2.0.1
```

## Nvidia-docker2 telepítése

Mivel a python kódot GPU-n kell futtatni, ezért fontos, hogy a docker hozzáférést kapjon a videokártyához. Ehhez az nvidia-docker2 kiegészítő szoftver csomagot lehet felhasználni. Ha nem docker-ben fut a python kód, akkor fel kell telepíteni a számítógépünkre a CUDA és a cuDNN szoftvereket. Ezeknek a szoftvereknek telepítése, beállítása és az NVIDIA videokártyához történő finomhangolása azt eredményezheti, hogy valami hibás beállítás hajtódik végre a számítógépen, ütközést okozva egyes szoftverkomponensekben.

Az NVIDIA, a Docker és a Google egy előre beállított docker képfilet készített, ami kötöttségek nélkül felhasználható. Ez az image hozzáférhet a számítógép videokártyájához, ezért fontos az nvidia-docker2 szoftver telepítése.

#### Telepített docker beállítása:

```
curl https://get.docker.com | sh \
  && sudo systemctl --now enable docker
```

#### NVIDIA konténer toolkit beállítása:

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
  && curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo
apt-key add - \
  && curl -s -L https://nvidia.github.io/nvidia-
docker/$distribution/nvidia-docker.list | sudo tee
/etc/apt/sources.list.d/nvidia-docker.list
```

#### Rendszer tároló frissítése:

```
sudo apt-get update
```

#### nvidia-docker2 telepítése:

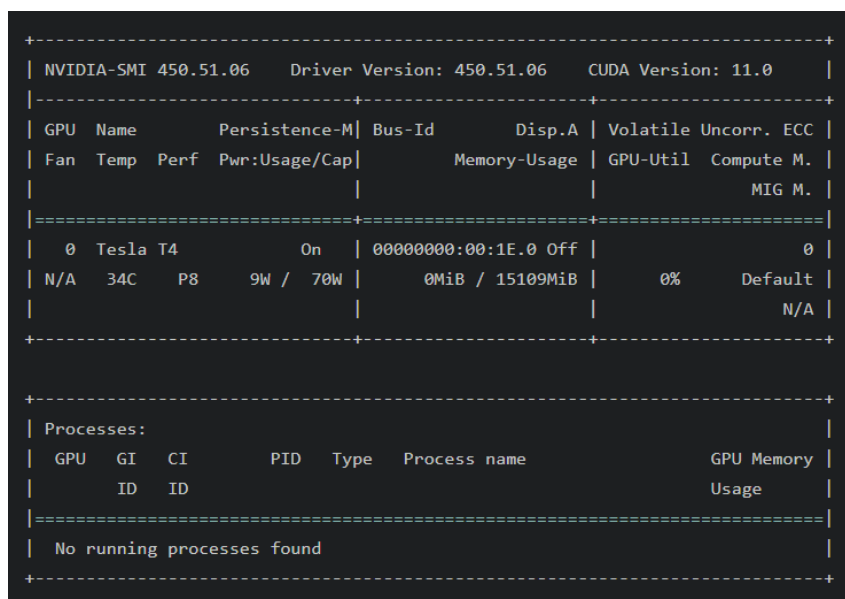
```
sudo apt-get install -y nvidia-docker2
```

#### Docker systemctl újraindítása:

```
sudo systemctl restart docker
```

#### nvidia-docker2 tesztelése (16. ábra):

```
sudo docker run --rm --gpus all nvidia/cuda:11.0-base nvidia-smi
```



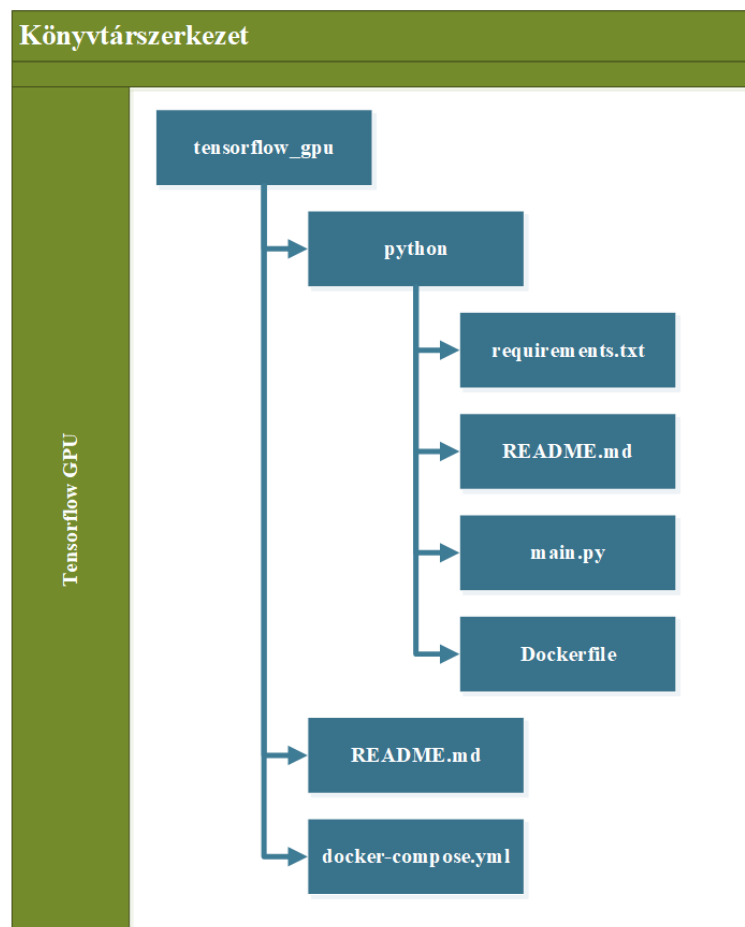
```
+-----+
| NVIDIA-SMI 450.51.06      Driver Version: 450.51.06      CUDA Version: 11.0      |
+-----+-----+-----+-----+-----+
| GPU  Name                Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                       |                    |                    |
|-----+-----+-----+-----+-----+
|   0   Tesla T4              On         | 00000000:00:1E:0  Off  |            0         |
| N/A   34C    P8      9W /  70W |  0MiB / 15109MiB |      0%    Default   |
|                                       |                    |                    |
+-----+-----+-----+-----+-----+

+-----+
| Processes:                                     |
|  GPU   GI    CI          PID    Type   Process name          GPU Memory |
|        ID    ID                                   Usage          |
+-----+-----+-----+-----+-----+
| No running processes found                  |
+-----+
```

**16. ábra: Teszt kimenete (szerk. a szerző)**

### 3.1.2 DCGAN futtatása docker-ben

A rendszer beállítása után DCGAN háló tanításához be kell konfigurálni a környezetet.



17. ábra: Könyvtárszerkezet (szerk. a szerző)

A docker-compose.yml a konténer felépítést írja le:

```
version: "3.4"

services:
  tensorflow_gpu:
    build: ./python
    environment:
      - PYTHONUNBUFFERED=1
      - PYTHONIOENCODING=UTF-8
    volumes:
      - ./python:/app
    ports:
      - 8888:8888
    deploy:
      resources:
        reservations:
          devices:
            - capabilities: [gpu]
```

Ahhoz, hogy a konténer valóban elérje a számítógép GPU-ját, tartalmaznia kell a „.yaml” leíró fájlban a „capabilities: [gpu]” kapcsolót.

### Dockerfile

A Dockerfile a konténer leíró fájl. A docker compose indításakor erre a fájlra hivatkozik, mikor szeretné letölteni a docker képet és beállítani a konténert.

```
FROM tensorflow/tensorflow:latest-devel-gpu
WORKDIR /app
COPY . /app
RUN pip install --upgrade pip
RUN pip install -r requirements.txt
ENTRYPOINT ["python"]
CMD ["main.py"]
```

### A szoftver futtatásához szükséges docker image:

```
tensorflow/tensorflow:latest-devel-gpu
```

### requirements.txt

### A python kód futtatásához szükséges python modulok listája, amit a konténeren belül telepíteni kell:

```
##### Requirements without Version Specifiers #####
tensorflow
numpy
matplotlib
gdown
##### Requirements with Version Specifiers #####
```

### Szoftver futtatása

A python szoftver futtatása docker compose szoftver segítségével történik a következő parancsokkal:

### docker compose build command a tensorflow\_gpu könyvtárban:

```
sudo docker-compose up
```

### docker run command tensorflow\_gpu könyvtárban:

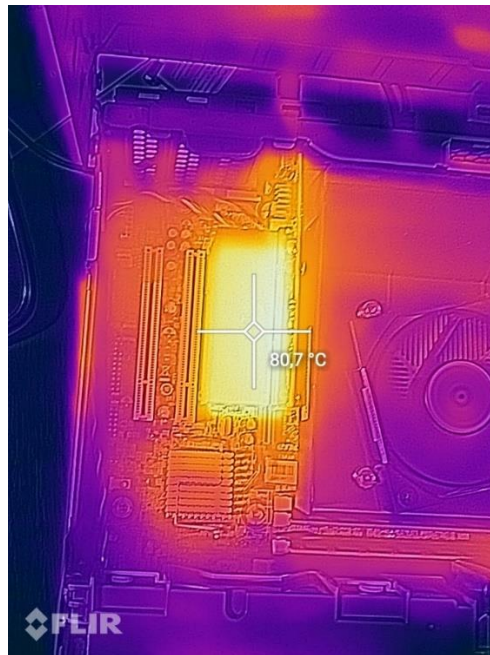
```
sudo docker-compose start
```

## **3.1.3 DCGAN-nal kapcsolatos eredmények**

A következő teszteredmények jöttek létre a python kód Xeon Silver 4208 processzoron és Nvidia Geforce GT730 GPU-n történő futtatásakor:

- Amikor a python szoftver a CPU-ban fut, akkor maximálisan kihasználja a rendelkezésre álló CPU magokat.

Mikor a GPU-ban fut, a számolás teljes mértékig a GPU-ban történik, 100%-os kihasználtság mellett, ezért alig használ CPU erőforrást. A GPU melegevése miatt, később egy jobb hűtéssel rendelkező gépházba került.



**18. ábra: Passzív hűtésű videokártya hűtőbordájának melegevése 100% terhelés mellett 80,7°C (szerk. a szerző)**

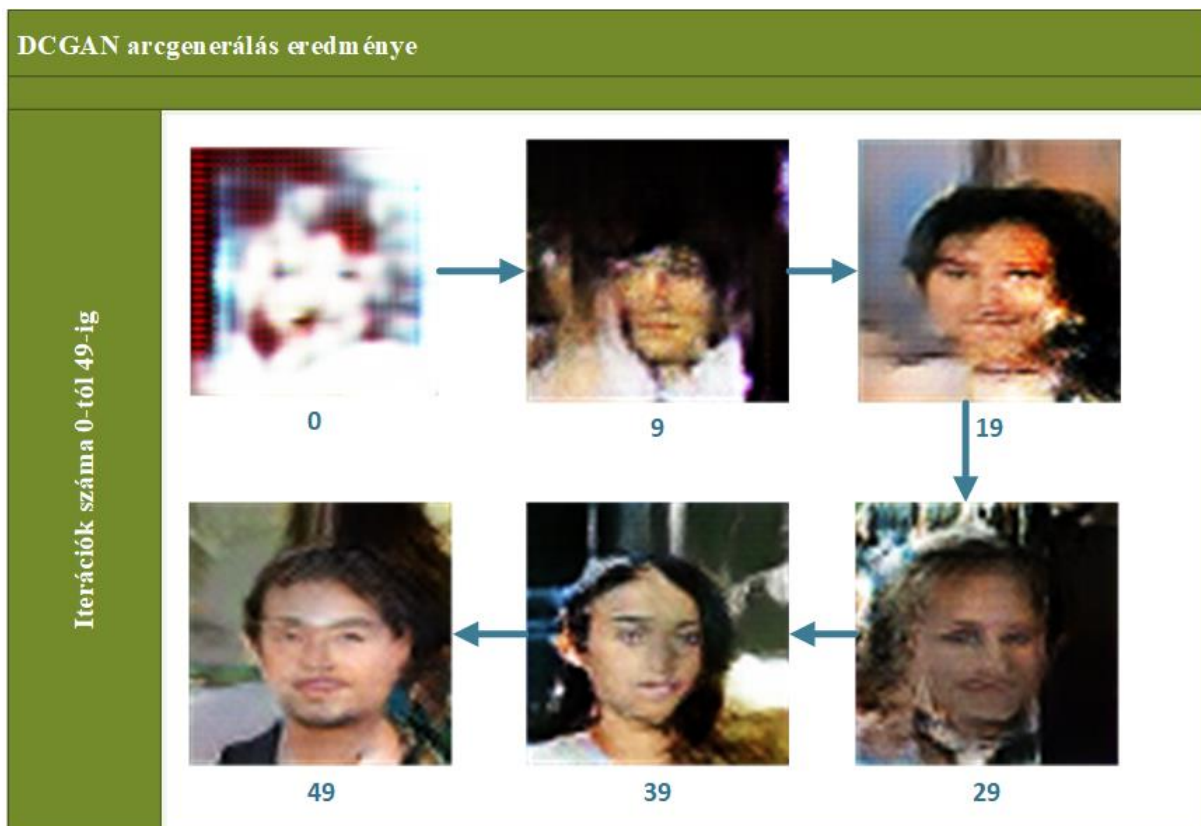
A legnagyobb különbség a számolás sebességében figyelhető meg.



**19. ábra: Xeon vs. GT730 (szerk. a szerző)**

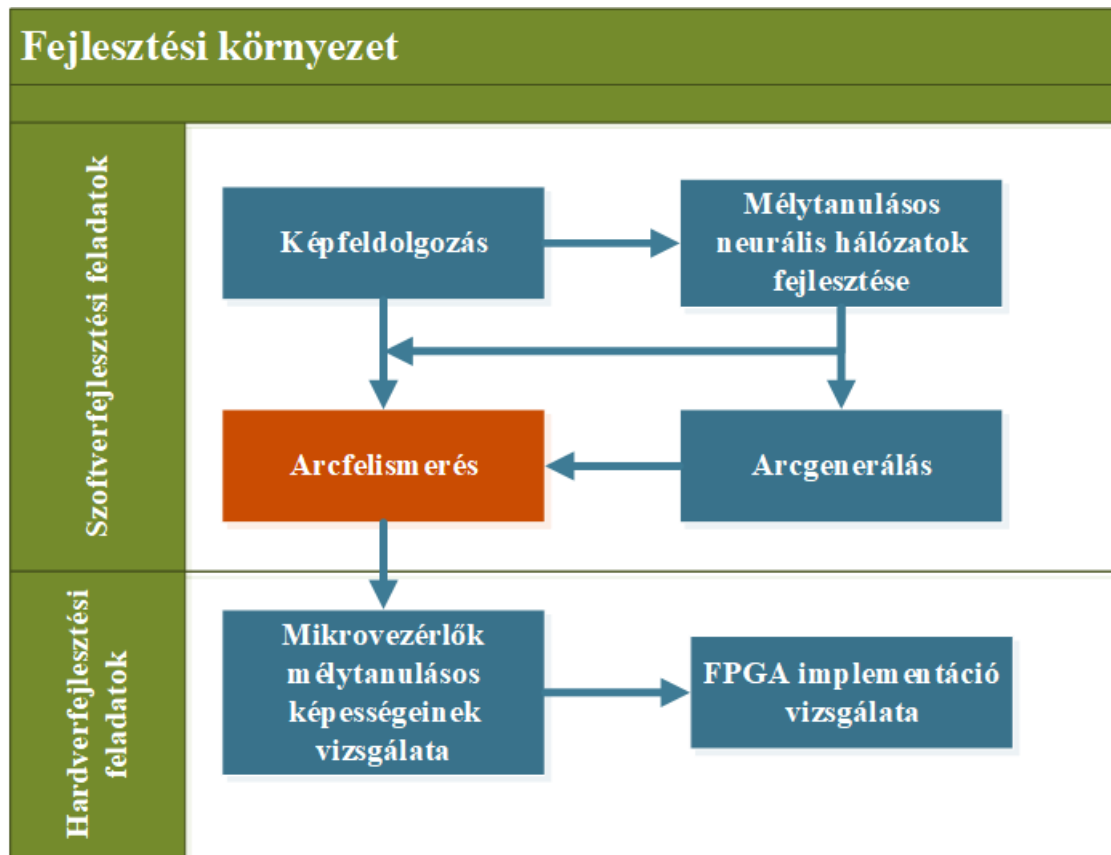
- 19. ábra bal oldali terminál ablakban a 8 magos Xeon Silver 4208 processzor, ~ 8.9 óra/epoch futással (32.203s)
- 19. ábra jobb oldali terminál ablakban a GT730 ~8.1 óra/epoch futási idővel (29.250s)

Bár nem mondható jelentősnek a két futási idő közötti különbség, fontos megjegyezni, hogy a GT730 egy passzív hűtéses, olcsó videokártya. Sajnos technikai okokból egyelőre nem sikerült végig futtatni a 100 epoch-os arcgenerálást, a legjobb eredmény 49 epoch volt, melynek eredménye a 20. ábraán látható.



20. ábra: Arcgenerálás 49 epoch alatt (szerk. a szerző)

## 3.2 MATLAB – CNN



21. ábra: Fejlesztési környezet, konvolúciós neurális hálózat (szerk. a szerző)

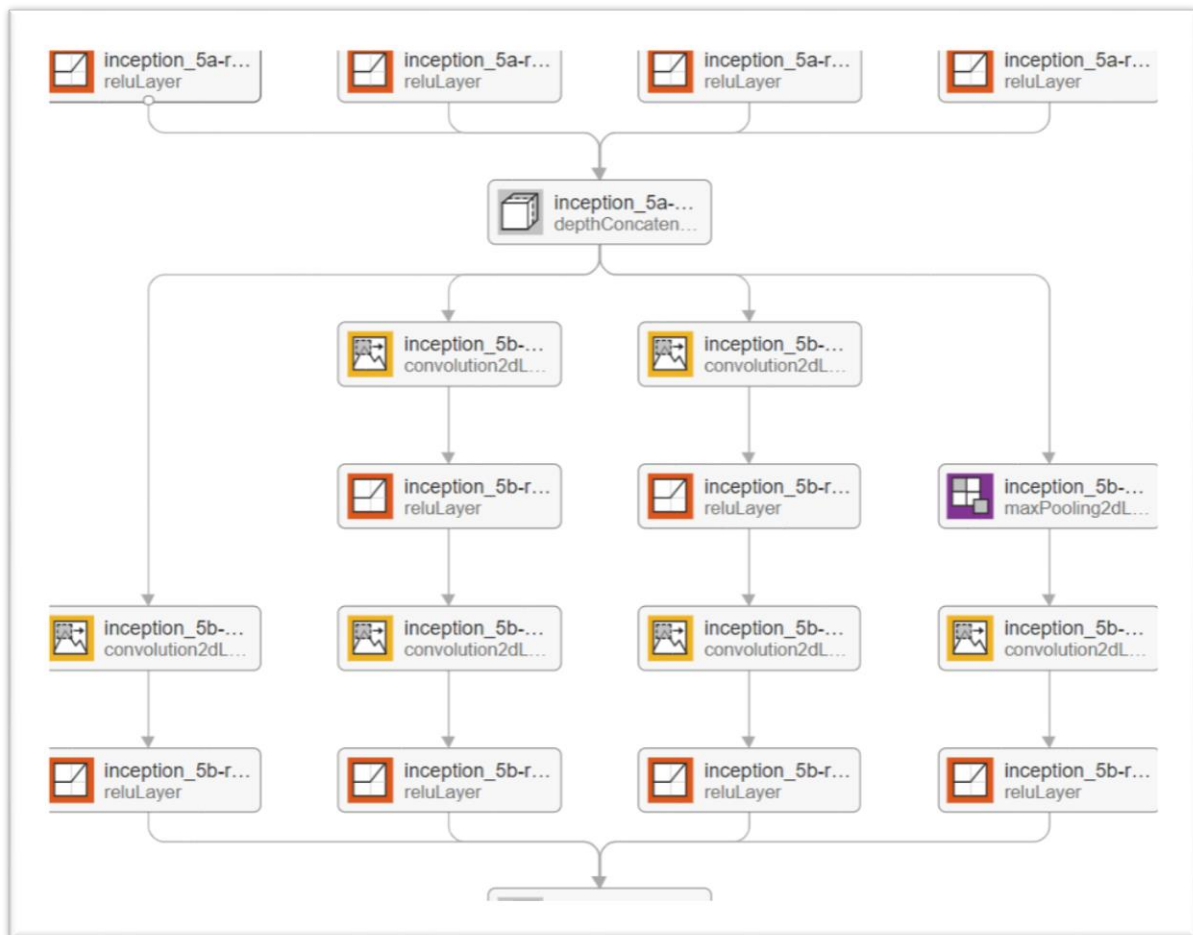
A projekt során az egyik fő feladat az előtanított arcfelismerő algoritmus meghatározása és áttanítása. Ehhez a Deep Learning Toolbox került meghatározásra keretrendszerként (22. ábra).

A szolgáltatások mellett lényeges elvárás, hogy a GPU támogatás ne okozzon nehézségeket, mivel a futtatási környezet fontos alkotóeleme az Nvidia videokártya. A korábban bemutatott machine learning framework-ök támogatják a GPU-n történő program végrehajtást, mivel a mátrix műveletek párhuzamosított futtatása gyors tanítást tesz lehetővé. Az Nvidia-ra az Nvidia CUDA Deep Neural Network library (cuDNN) miatt esett a választás, mivel ez az eszköztár lehetőséget biztosít mélytanulási rendszerek implementálására.

Az optimalizált eszköztárba tartozik:

- Forward, backward konvolúció és keresztkorreláció;
- Mátrix szorzás;
- Forward, backward pooling;
- Forward, backward softmax;
- Aktivációs függvények: relu, tanh, sigmoid, elu, gelu, softplus, swish;

- Tenzor transzformációk;
- Batch normalizálás.



**22. ábra: GoogleNet módosítása Deep Learning Toolbox segítségével (szerk. a szerző)**

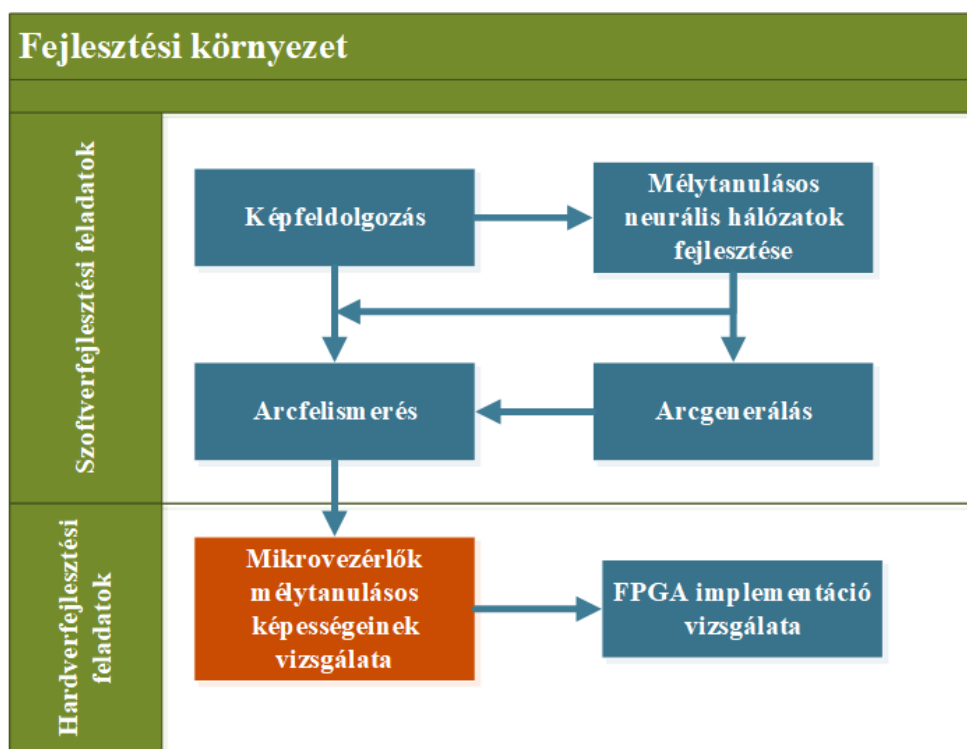
Lényeges szempont, hogy a kiválasztott keretrendszer támogassa a GPU architektúrát. Mivel MATLAB R2021b vagy frissebb verziót használunk, ezért az Ampere, Turing, Volta illetve Pascal architektúrák jöhetnek számításba.

Jelenleg ezen a téren nem történt előrelépés, mivel még nem áll rendelkezésre a megfelelő hardverkörnyezet. Bár a Deep Network Designer-ben tesztadatokon módosítva lettek a SqueezeNet és a GoogleNet előtanított hálózatok, potens GPU, illetve a Parallel Toolbox hiányában a tanítási idő irreálisan hosszúnak bizonyult.

## 4 Hardver implementáció

Számos esetben felmerül az igény, hogy algoritmusunkat ne személyi számítógépen, hanem egy kisebb teljesítmény- és helyigényű környezetben futtassuk, ami jelen projektben egy erőforrásokkal rendelkező beágyazott rendszer. Egy ilyen rendszer fontos tulajdonsága, hogy a szoftverfejlesztésnél a végrehajtási sebesség mellett a memóriaigény figyelembe vétele nem opcionális, hanem elengedhetetlen része a fejlesztésnek.

### 4.1 Mikrovezérlő MI képességei



23. ábra: Fejlesztési környezet, STM32 implementálás

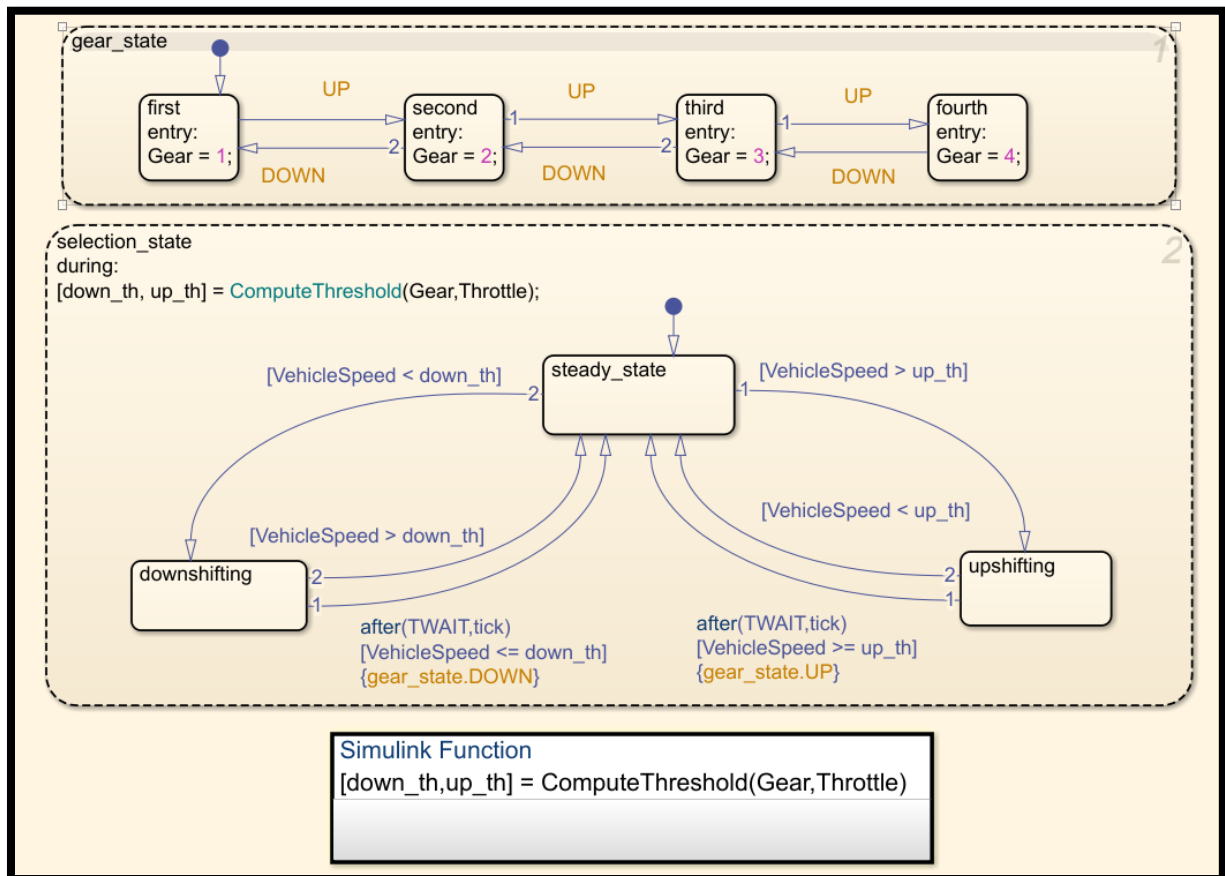
A fejlesztés kezdeti szakaszánál bár még számos eszköz nem áll rendelkezésre, a lehetőségekhez mérten törekedtem arra, hogy minél mélyrehatóbban elemezzem a már rendelkezésre álló beágyazott rendszerek képességeit (23. ábra). A vizsgálat középpontjában a Tensorflow Lite áll, ami a Tensorflow beágyazott rendszerekre implementálható verziója. A kiválasztott célhardver az ST Microelectronic SMT32F746 mikrovezérlője.

#### 4.1.1 A vizsgálat célja

A Tensorflow Lite jellemzően kis teljesítményű mikroprocesszorokon és mikrovezérlőkön futtatható kódot generál. Jelen feladat célja, hogy megvizsgáljam, milyen potenciállal bír egy

ML eljárás a mikrovezérlők világában elterjedt állapotgépes vezérlésekkel és szabályozásokkal szemben. Beágyazott rendszerek esetén általában szenzor információk alapján hoznak döntéseket a mikrovezérlőkre implementált algoritmusok. A szenzor értékeket állapotgépek dolgozzák fel, és ezek alapján hoznak döntést.

### 4.1.2 Állapotgép



24. ábra: Állapotgép (Simulink Stateflow) (szerk. a szerző)

Egy állapotgépnek (24. ábra) az elkészítése (lekódolása) sok időt vesz igénybe. Ha az agilis szoftverfejlesztési módszert vesszük alapul, akkor előfordulhat, hogy az állapotgép elkészülte előtt, annak főbb feladatai, céljai megváltoznak. Ilyenkor az állapotgép részleges vagy teljes újraírása is felmerülhet.

### 4.1.3 Gépi tanulás mikrovezérlőn

A gépi tanulás az állapotgépek alternatívájaként már a mikrovezérlők világában is megjelent. Az ML során, megfelelő mennyiségű mérési adat rendelkezésre állása esetén (amik általában szenzoroktól érkeznek), elvégezhető a tanítás és az ellenőrzés. Megfelelő háló architektúra és

betanítás esetén ezek a modellek képesek lehetnek arra, hogy előre jelezzék egy adott rendszer meghatározott állapotától való eltérést. Ennek fő felhasználási területe az esetleges karbantartások előrejelzése (predictive maintenance), mellyel megelőzhető az adott eszköz idő előtti, váratlan meghibásodása.

Jelen fejezet célja, hogy a Tensorflow Lite által generált C kódot egy STM32F746-os mikrovezérlőn vizsgáljam. A folyamat három részből áll:

- Tanítás, kimenete egy modell;
- Modell befordítása a mikrovezérlő fejlesztő környezetébe;
- Mérés, validálás.

A validálási folyamat után érdemes nem csak az eredményeket figyelembe venni, hanem az eljárást teljes folyamatában vizsgálni piacra kerülési idő (TTM<sup>42</sup> és megbízhatóság) tekintetében is.

#### 4.1.4 Feladat meghatározása, neurális hálózat tanítása

A feladat egy szinusz függvény működésének betanítása egy négyrétegű perceptron hálózatba. Ennek megvalósítása Tensorflow Lite-tal történik a Google Collaboratories (Colabs) felületen. Innen történik a beágyazott C kód generálása, amit az STM Cube Ide fejlesztőkörnyezetben kialakított projektbe importálás után végrehajtásra kerül az STM32F746 mikrovezérlőn.

A tanításhoz ideális lehet a Google Collaboratories (Colabs) fejlesztő környezete. Ez a környezet felhőben fut, amely Gmail-es regisztrációval ingyenesen igénybe vehető, mint ahogy a futtatáshoz felhasználható CPU, GPU és TPU is, igaz csak korlátozott ideig és ingadozó teljesítménnyel. A projektben egy négyrétegű perceptron hálózat tanítását végzem el, melynek a forráskódját a TensorFlow github oldaláról le lehet tölteni:

[https://www.tensorflow.org/lite/microcontrollers/get\\_started\\_low\\_level](https://www.tensorflow.org/lite/microcontrollers/get_started_low_level)

A kód python nyelven íródott. A Colabs környezet képes python kódot futtatni Jupiter környezetben, ami egyfajta livescript-es kódban végrehajtott dokumentálási lehetőséget is biztosít.

##### Kezdeti lépések

```
%tensorflow_version 2.3  
%tensorflow_version` only switches the major version: 1.x or 2.x.
```

---

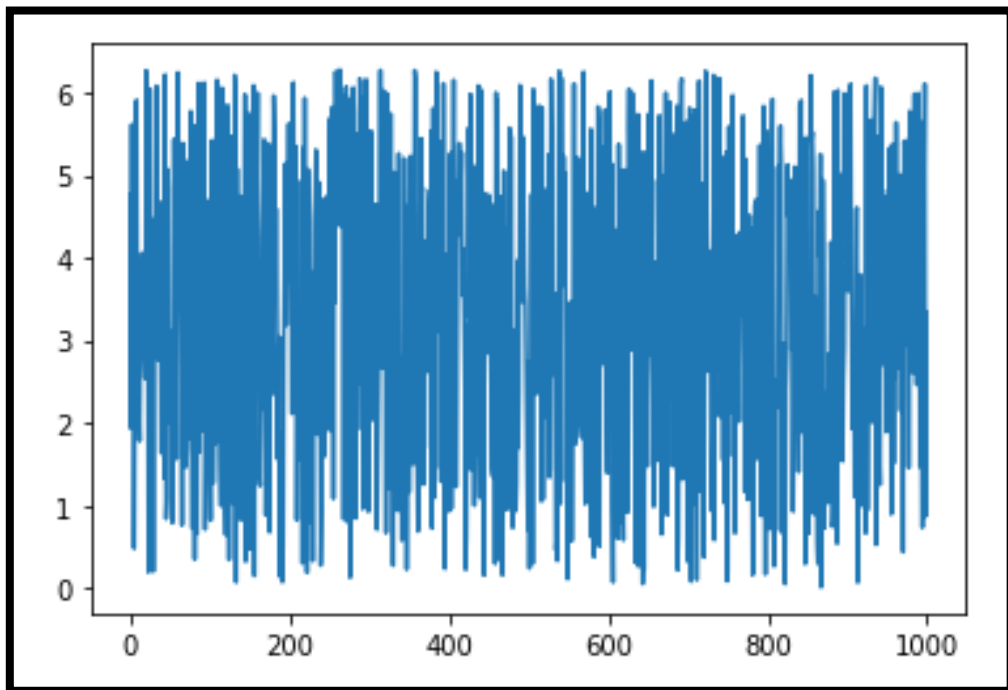
<sup>42</sup> TTM: Time To Market – Piacra kerülési idő, a fejlesztés kezdetétől számítva

```
You set: `2.3`. This will be interpreted as: `2.x`.  
TensorFlow 2.x selected.
```

Lényeges a Tensorflow szoftverkörnyezet verziójának a megválasztása. Sajnos a folyamatos és lényeges elemeket érintő fejlesztések miatt a jól megszokott metódusok egy idő után már nem hozzák az elvárt eredményt.

### Modell működésének leírása

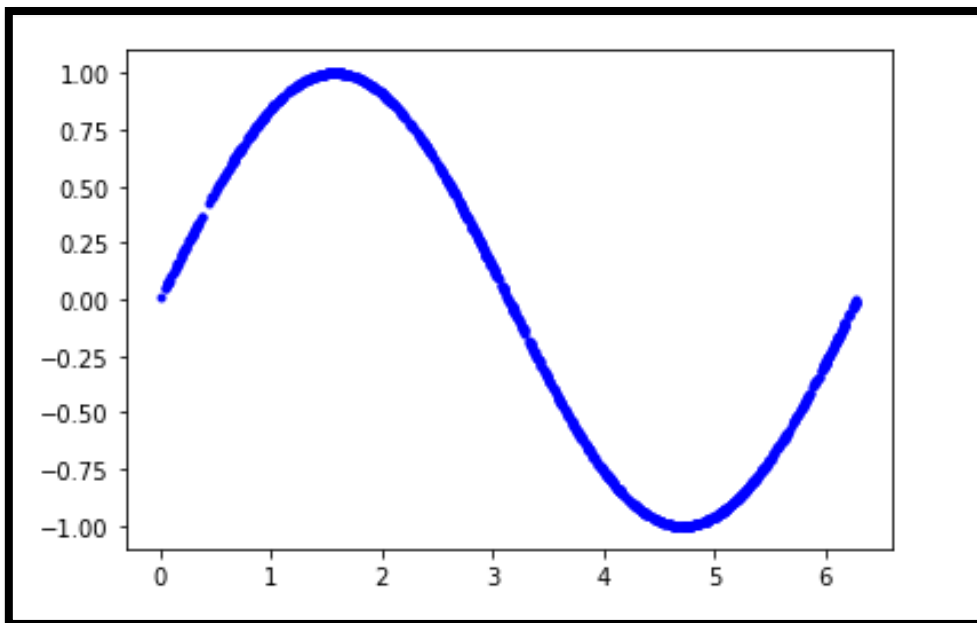
```
SAMPLES = 1500  
np.random.seed(786)  
tf.random.set_seed(786)  
  
x_values = np.random.uniform(low=0, high=2*math.pi, size=SAMPLES)  
plt.plot(x_values)  
plt.show()
```



**25. ábra: Véletlenszerű adat generálása (szerk. a szerző)**

A modell működését le kell írni. A bemeneti adatsor egy 1500 mintás adattömb 0 és  $2\pi$  közötti értékekkel véletlenszerűen feltöltve (25. ábra).

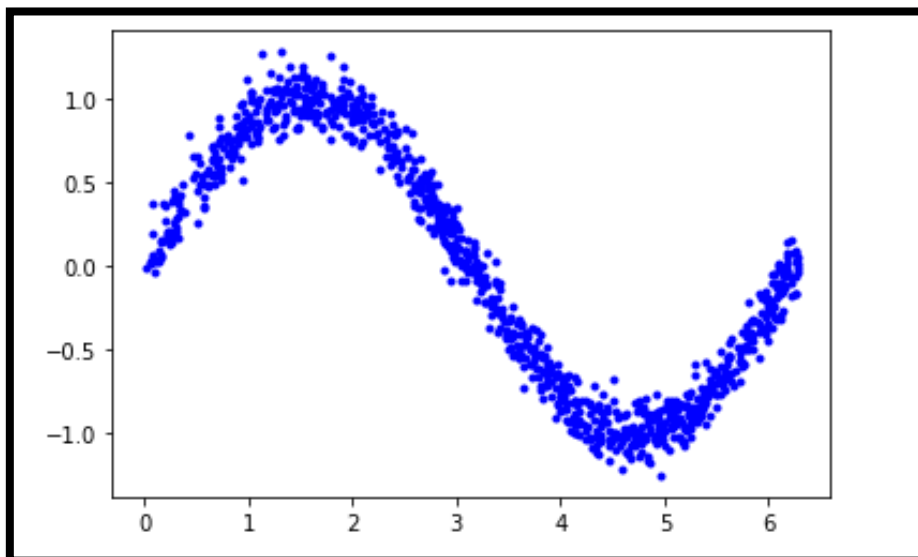
```
np.random.shuffle(x_values)  
y_values = np.sin(x_values)  
plt.plot(x_values, y_values, 'b.')  
plt.show()
```



**26. ábra: Szinusz függvény diszkrét adatpontjainak generálása (szerk. a szerző)**

A modell tanításához meg kell határozni, hogy a különböző bemeneti értékekhez milyen kimeneti értékek tartoznak. Mivel a cél a szinusz függvény betanítása, ezért a különböző véletlenszerű  $0-2\pi$  értékeknek szinuszát jelenítjük meg és vizsgáljuk (26. ábra).

```
y_values += 0.1 * np.random.randn(*y_values.shape)
plt.plot(x_values, y_values, 'b.')
plt.show()
```

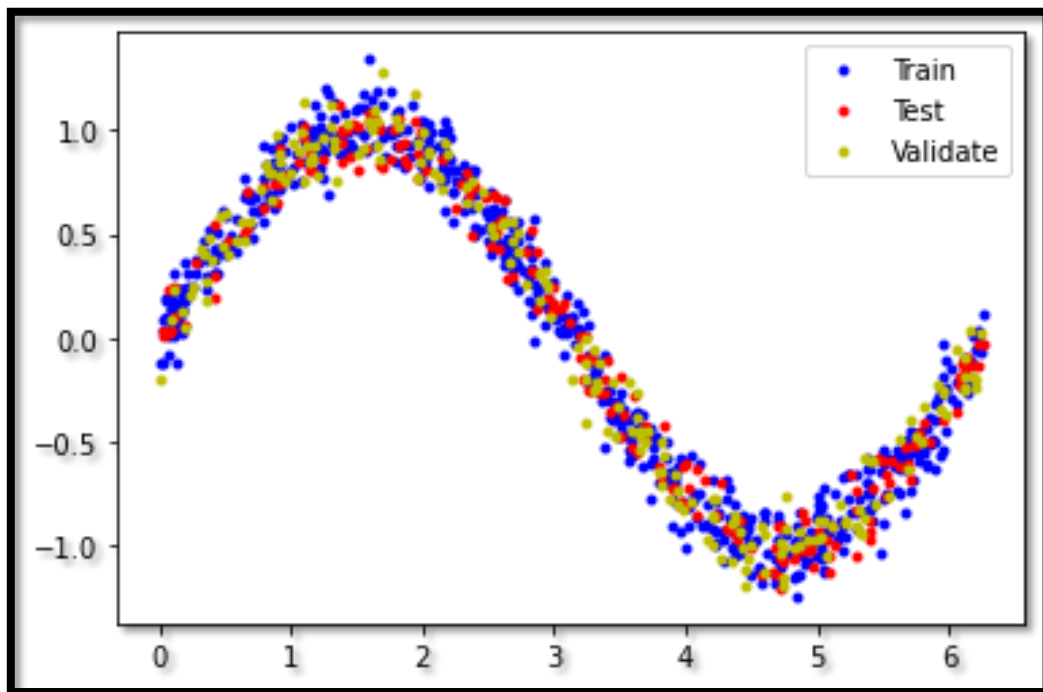


**27. ábra: Véletlenszerű adatpontok a szinusz függvényhez rendelve (szerk. a szerző)**

Egy szenzoros hálózat esetén a bemenet zajt is tartalmaz, ezért a kimeneti adatsort torzítani kell, hogy a valós életből nyerhető adatokhoz hasonlítson (27. ábra).

Ezután három részre osztjuk az 1500 elemű adathalmazunkat (28. ábra):

- Tanítás: 60%
- Validálás: 20%
- Tesztelés 20%

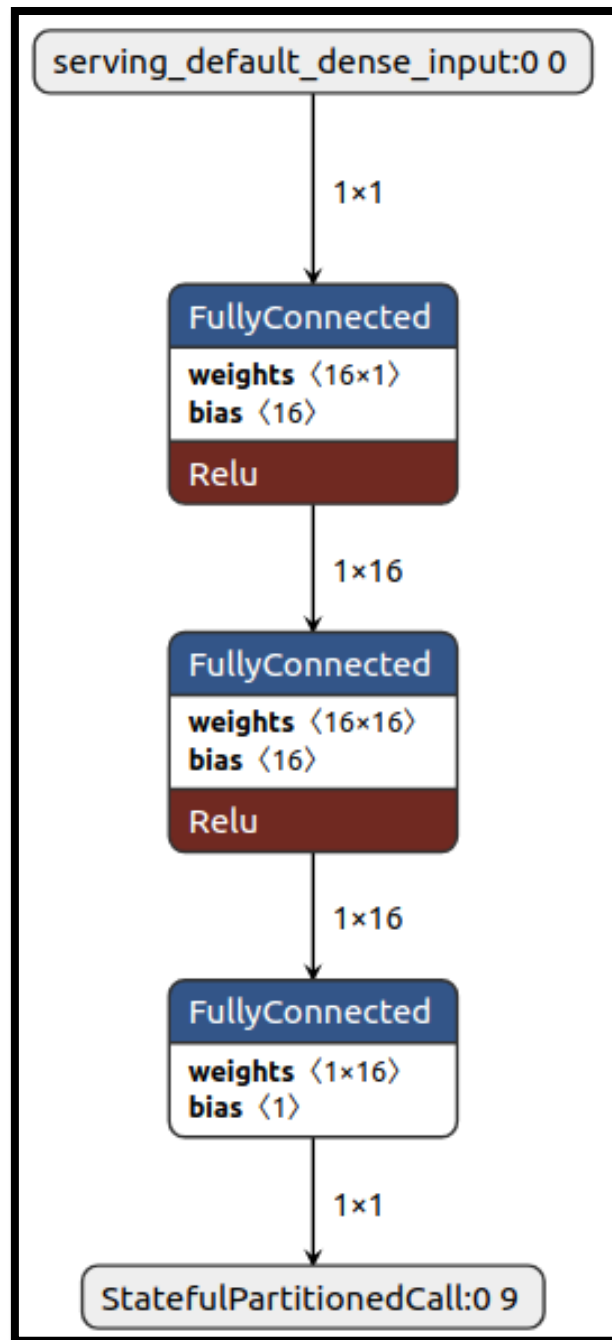


28. ábra: Adatok szegmentálása (szerk. a szerző)

#### 4.1.5 Neurális hálózat felépítése

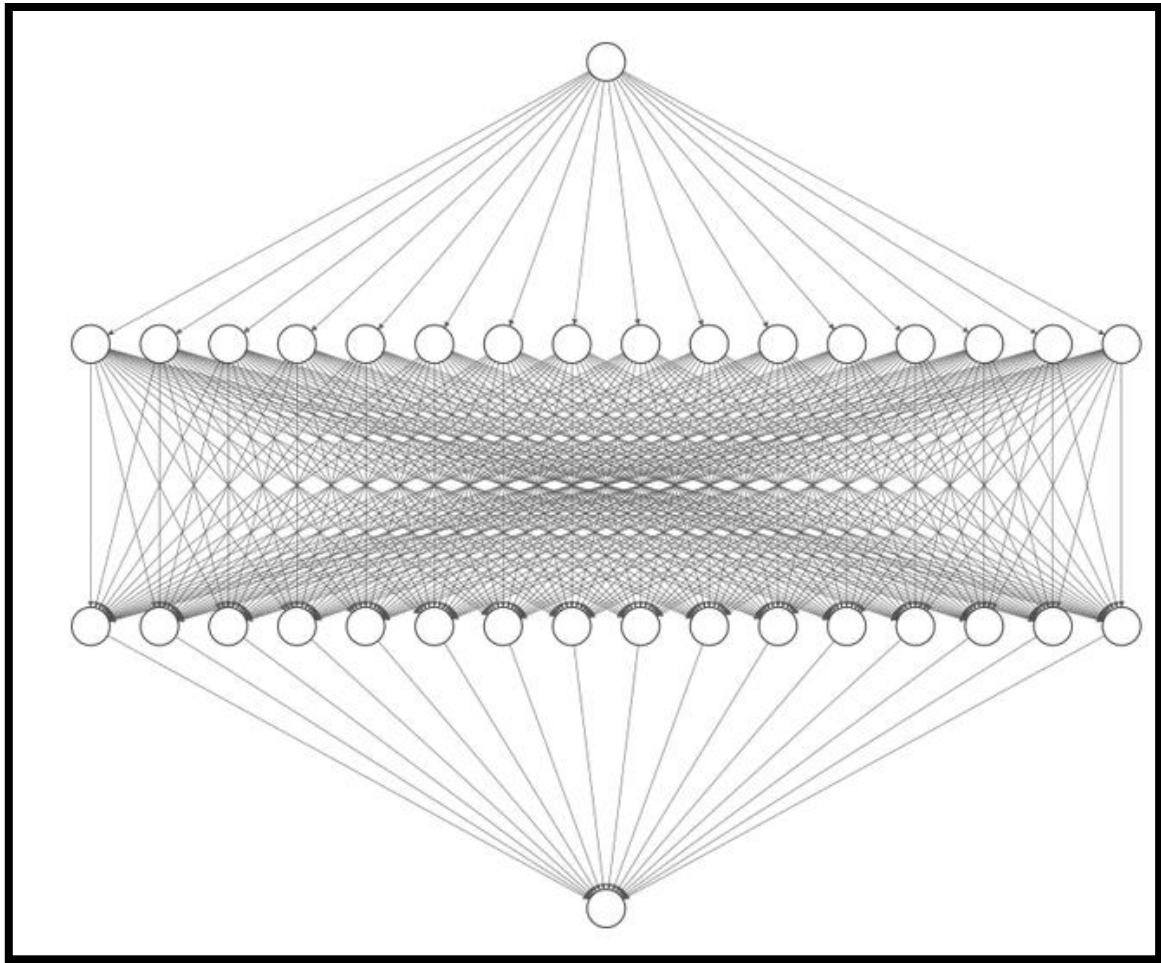
```
model = tf.keras.Sequential()  
model.add(keras.layers.Dense(16, activation='relu', input_shape=(1,)))  
model.add(keras.layers.Dense(16, activation='relu'))  
model.add(keras.layers.Dense(1))  
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

A modell egy négyrétegű perceptron hálózatot tartalmaz, melynek rétegei előre csatoltak és teljesen be vannak kötve (29. ábra).



**29. ábra: Négyrétegű perceptron hálózat felépítése** (forrás: [https://www.tensorflow.org/lite/microcontrollers/get\\_started\\_low\\_level](https://www.tensorflow.org/lite/microcontrollers/get_started_low_level); 2022.02.16)

Az első réteg egy bemenetet tartalmaz, egy darab neuronnal. A második réteg egy 16 neuronból álló teljesen bekötött (dense) hálózat, aminek az aktivizációs függvénye az ReLu (Rectified Linear Unit) függvény. A harmadik réteg szintén 16 neuronból álló dense hálózat, a negyedik rétegünk egy neuront tartalmazó kimenet (30. ábra).



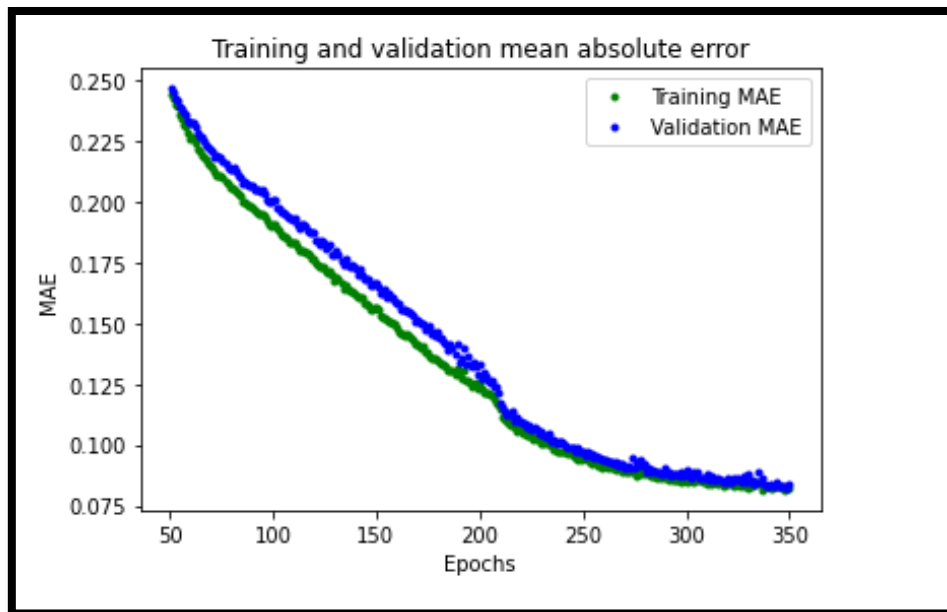
**30. ábra - Négyrétegű perceptron hálózat felépítése** (forrás: [https://www.tensorflow.org/lite/microcontrollers/get\\_started\\_low\\_level](https://www.tensorflow.org/lite/microcontrollers/get_started_low_level); 2022.02.16)

### Hálózat betanítása

```
training_info = model.fit(x_train, y_train, epochs=350,
batch_size=64, validation_data=(x_validate, y_validate))
...
Epoch 350/350
10/10 [=====] - 0s 5ms/step - loss: 0.0105 -
mae: 0.0822 - val_loss: 0.0110 - val_mae: 0.0840
INFO:tensorflow:Assets written to: saved_model_keras_dir/assets
```

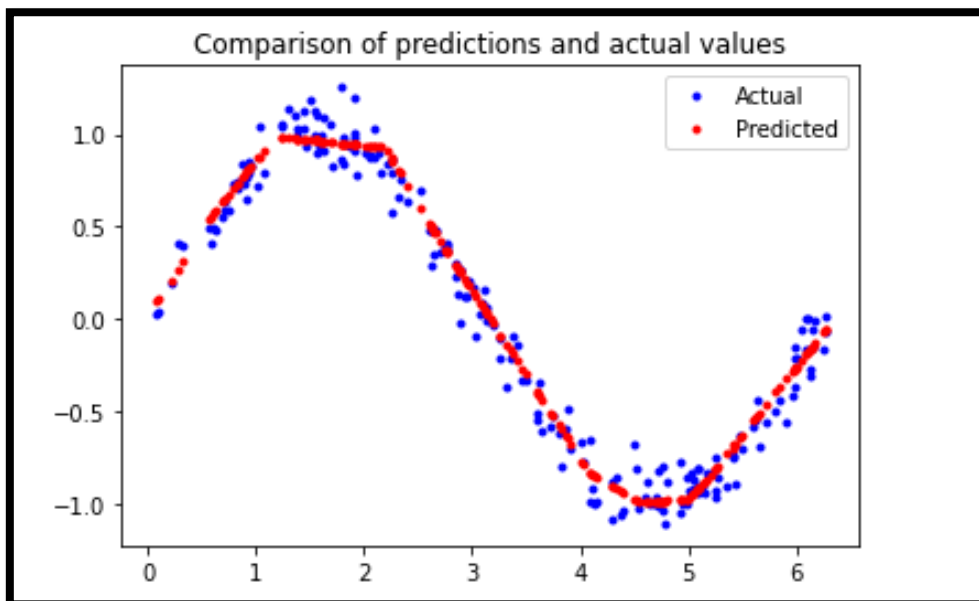
A tanítás 350 epoch alatt lett elvégezve, 64-es batch-ekkel. Ennyi iteráció a meglévő beállítások mellett azt eredményezte, hogy a tanító és validáló adatok feldolgozásakor közel azonos eredményt kapunk, tehát a hálózatunk nincsen túltanítva (31. ábra). Mindemellett az átlagos abszolút hiba (MAE<sup>43</sup>) 8.4%, ami jelen felhasználásnál elégségesnek minősíthető.

<sup>43</sup> MAE: Mean Absolute Error – Átlagos abszolút hiba



31. ábra: Átlagos abszolút hiba (szerk. a szerző)

Amennyiben a pontosság növelése szükséges (32. ábra), a hálózat architektúráján érdemes fejleszteni.



32. ábra: Betanított szinusz függvény pontossága (szerk. a szerző)

#### 4.1.6 Tensorflow Lite modell generálása

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
open("sinewave_model.tflite", "wb").write(tflite_model)
```

A TensorFlow Lite modell generálása az egyik legkritikusabb része a feladatnak. Mivel mikrovezérlős rendszeren lesz implementálva, a hardver erőforrás limitált, és a buszrendszer is 32 bites, tehát körültekintően kell bánni a különböző optimalizációkkal, konverziókkal. Több fajta megoldás létezik, a fejlesztés egyes szakaszaiban többször trial and error<sup>44</sup> módszerre kellett hagyatkozni. Az STM32F746-os mikrovezérlőnél a jó megoldás az volt, hogy Keras segítségével legyen konvertálva a modell az optimailizációk kikapcsolásával.

#### A generált modell neve:

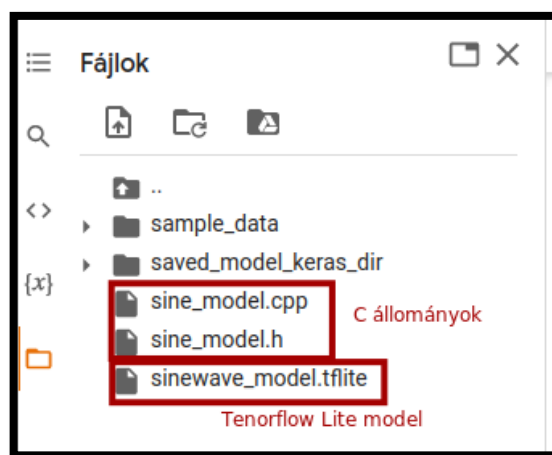
sinewave\_model.tflite

#### C állományok generálása

```
from tensorflow.lite.python.util import convert_bytes_to_c_source
source_text, header_text =
convert_bytes_to_c_source(tflite_model, "sine_model",
include_path="sine_model.h")
with open('sine_model.h', 'w') as file:
file.write(header_text)
with open('sine_model.cpp', 'w') as file:
file.write(source_text)
```

Miután elkészült a modell, ezt olyan forrásállományként kell elmenteni, melyet képesek a különböző fejlesztő környezetek befordítani (33. ábra). A két lényeges file:

sine\_model.h  
sine\_model.cpp

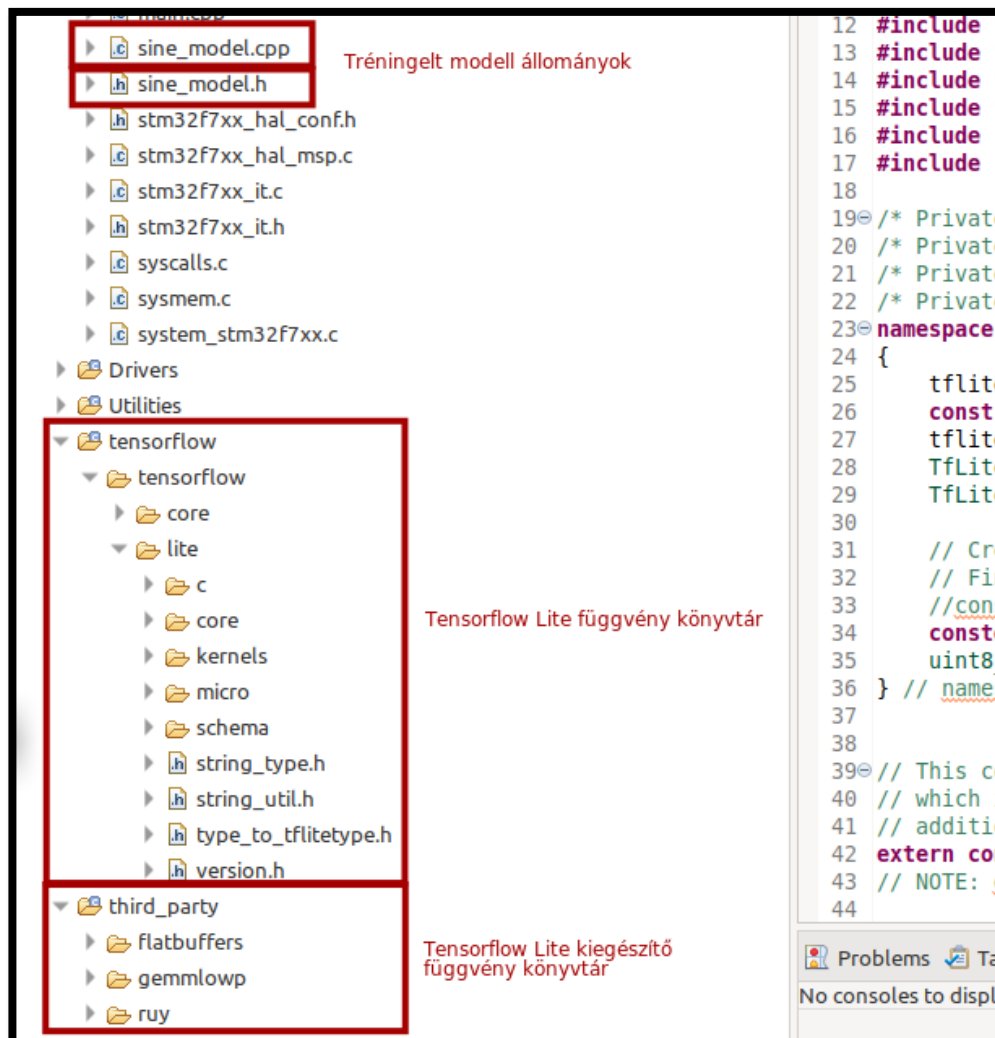


33. ábra: A legenerált modellek (szerk. a szerző)

<sup>44</sup> Trial and error – Alapvető problémamegoldási módszer, lényege hogy folyamatos változtatásokat addig véghezvük, amíg jó eredményt nem kapunk

## Tensoflow Lite STM Cube IDE-ben

Az STM32 Cube IDE projekt neve: STM32F746G\_TFLite\_Sine. Ahhoz hogy a generált modellek felhasználhatók legyenek, szükséges a TensorFlow Lite függvénykönyvtár projektbe illesztése.



**34. ábra: Perceptron hálózatunk importálása STM32 Cube IDE-be (szerk. a szerző)**

A TensorFlow és a third\_party mappába kell helyezni a tensorflow lite függvénykönyvtárat (34. ábra).

### 4.1.7 Tensorflow lite main.c

A modell futtatásához a main.cpp állományba kell elhelyezni a tensorflow lite függvénykönyvtárat behívó programkódokat.

```
#include "sine_model.h"
#include "tensorflow/lite/micro/kernels/all_ops_resolver.h"
```

```
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"
```

- `all_ops_resolver.h` – biztosítja az értelmező által a modell futtatásához használt műveleteket;
- `micro_error_reporter.h` – kimeneti debug függvények;
- `micro_interpreter.h` – a modell betöltéséhez és futtatásához tartalmaz függvényeket;
- `schema_generated.h` – a modell ún. TensorFlow Lite FlatBuffer model file formátumához tartalmaz sémát;
- `version.h` – TensorFlow Lite verzióját kaphatjuk meg.

Külön kiemelendő:

```
#include "sine_model.h"
```

Tensorflow lite namespace:

```
namespace
{
    tflite::ErrorReporter* error_reporter = nullptr;
    const tflite::Model* model = nullptr;
    tflite::MicroInterpreter* interpreter = nullptr;
    TfLiteTensor* model_input = nullptr;
    TfLiteTensor* model_output = nullptr;
    constexpr uint32_t kTensorArenaSize = 2 * 1024;
    uint8_t tensor_arena[kTensorArenaSize];
}
```

A namespace-ben nagyon fontos a memória allokáció. Ha a deklarált memória mérete nem elégséges, akkor könnyen hibára futhat a modell:

```
const int tensor_arena_size = 2 * 1024;
uint8_t tensor_arena[tensor_arena_size];
```

Modell betöltése:

```
model = tflite::GetModel(sine_model);
```

Tensorok allokálása:

```
TfLiteStatus allocate_status = interpreter->AllocateTensors();
```

### Bemenet szolgáltatása a modell-nek:

```
float x_val = 2.0;
// Place our calculated x value in the model's input tensor
model_input->data.f[0] = x_val;
```

A teszt egyszerűsítése érdekében egy konstans float 2-es szám a bemenet, ez később lehet egy analóg-digitális konverter kimenete is.

### Modell futtatása:

```
TfLiteStatus invoke_status = interpreter->Invoke();
if (invoke_status != kTfLiteOk)
{
    TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed on
x_val: %f\n", static_cast<float>(x_val));
    return 0;
```

### Modell kimenet olvasása:

```
float y_val = model_output->data.f[0];
```

## **4.1.8 Ellenőrzés**

A modell működésének visszaellenőrzéséhez a referenciához való viszonyításra van szükség, amivel kiszámolható a hibaszázalék:

$$y = \sin(2)$$

$$y = 0.9092974268256816954$$

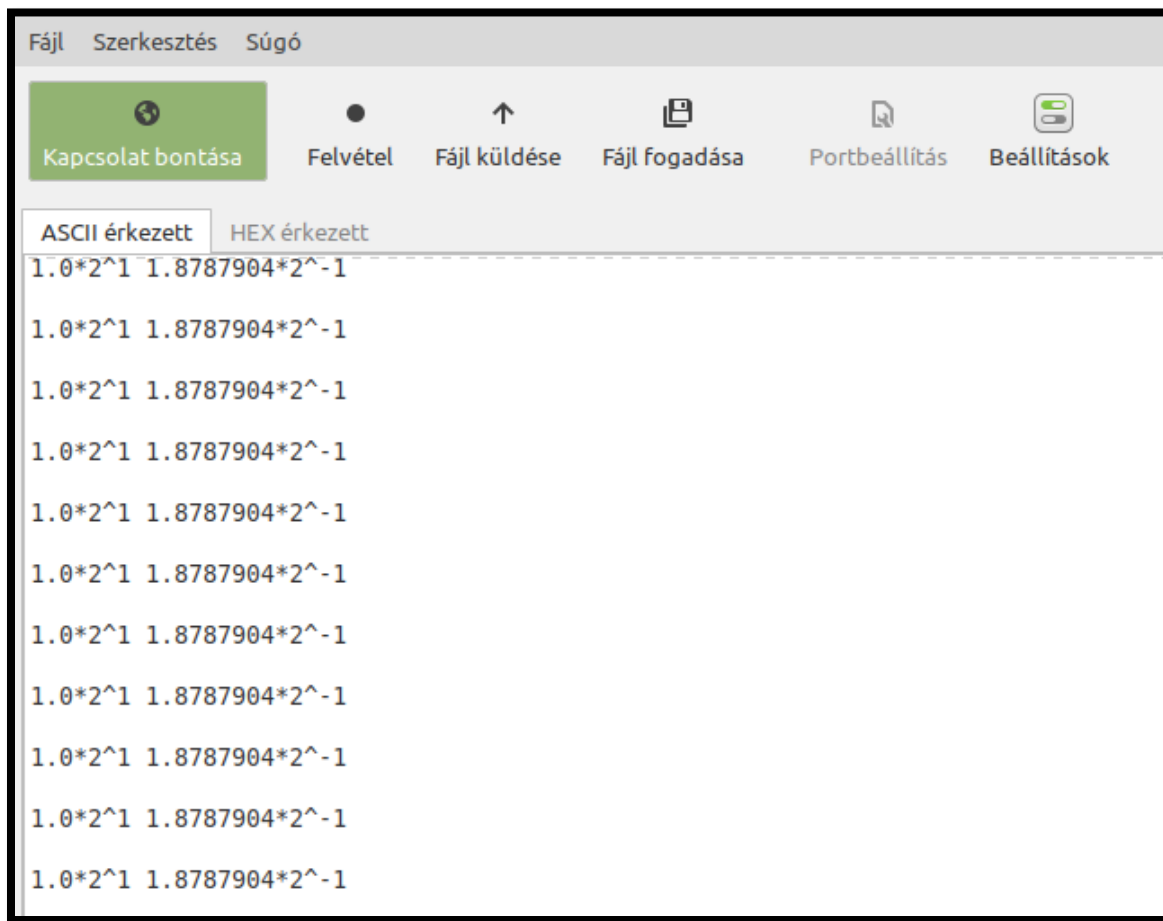
A  $\sin(2)$  értéke körülbelül 0.9092974268256816954, amit a példaként <1%-ban meghatározott pontossággal el kell érnie a betanított neurális hálózatnak. Ez a

```
TF_LITE_REPORT_ERROR(error_reporter, "%f\t%f\n", x_value, y_value);
```

függvénnyel jeleníthető meg.

A bemenet (35. ábra)  $10 \cdot 2^1$ , azaz 2, míg a kimenet  $1.87879042^{-1}$ , azaz 0.9393952. Az elvárt érték körülbelül 0.909297, az eltérés 3.2%-os, amit a hangolás kedvéért toleranciahatáron kívülnek tekintünk. A finomhangoláshoz a python kód paraméterein kell állítanunk. A tanítás

epoch számának növelése ezen a ponton nem hoz jelentős javulást, mivel a modellünk már 250-300 epoch-nál bekonvergált (31. ábra).



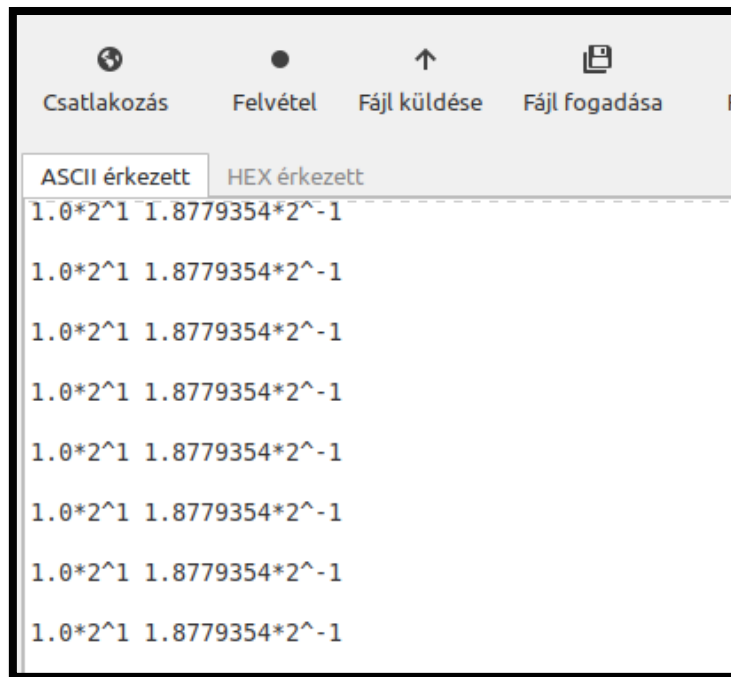
35. ábra: Terminál kimenet (szerk. a szerző)

#### 4.1.9 Tesztelés 2000 és 3000 mintával

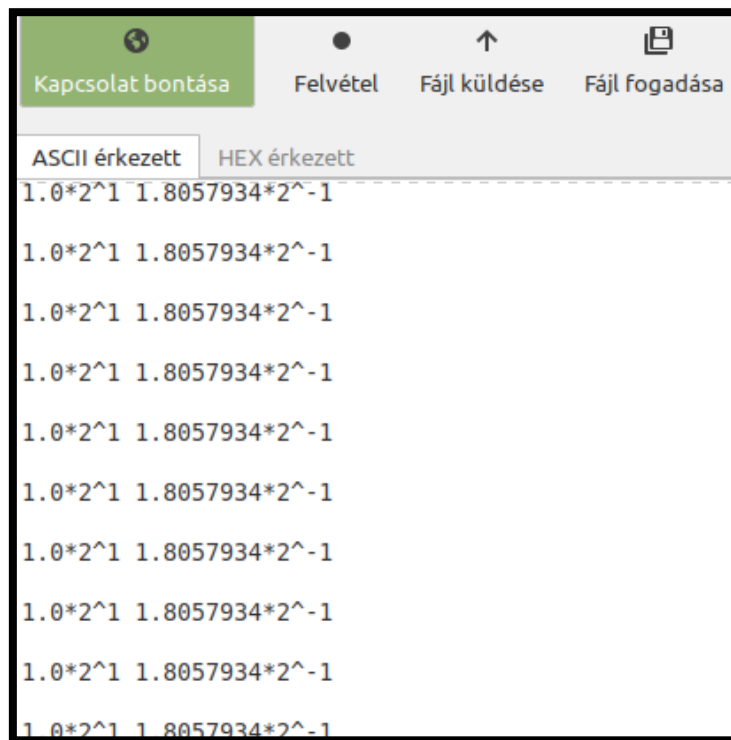
```
# Number of sample datapoints  
SAMPLES = 2000
```

```
# Number of sample datapoints  
SAMPLES = 3000
```

Az első esetben (36. ábra) a kimenet  $1.8779354 \cdot 2^{-1}$  azaz 0.9389677. Ez 3.16%-os eltérést jelent, nem tekinthető jelentős pontosság növekedésnek. A második esetben (37. ábra) a kimenet  $1.8057934 \cdot 2^{-1}$  azaz 0.9028967. Az eltérés 0.7%, ezt viszont már elfogadható eredménynek tekinthető.



36. ábra: Teszt eredménye 2000 mintával (szerk. a szerző)



37. ábra: Teszt eredménye 3000 mintával (szerk. a szerző)

#### **4.1.10 A beágyazott MI konklúziója**

Az  $y=\sin(2)$  műveletet beágyazott rendszeren futtatható neurális hálózatba tanítani és behangolni jelenleg nem egyszerű feladat. Látszik, hogy a jelenleg kész megoldások, a TensorFlow, a TensorFlow Lite és az STM32 Cube IDE folyamatos változásának köszönhetően nem frissülnek megfelelő ütemben, egymással nincsenek összhangban.

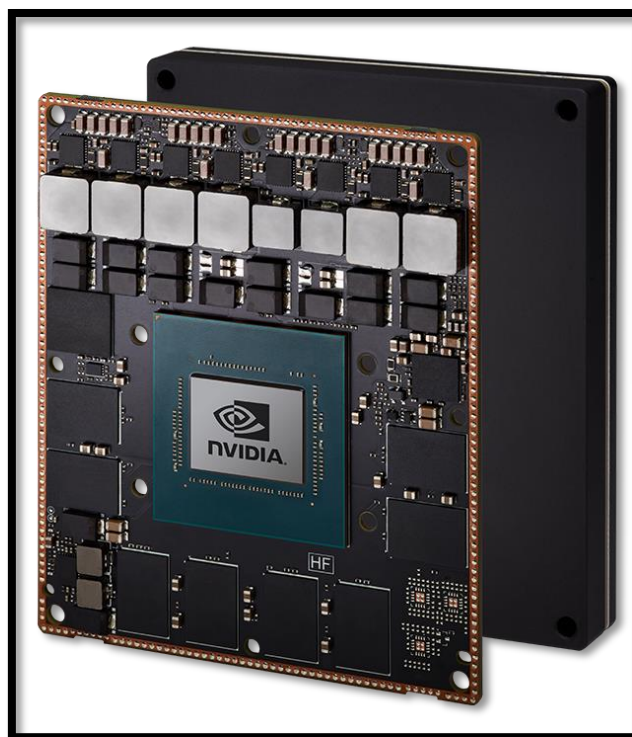
A dolgozat egyik célja megvizsgálni az ML jelentette potenciált a jelenleg még elterjedt állapotgép alapú vezérlési és szabályozási rendszerek leváltására mikrokontrolleres rendszerekben. Jelen eredmények tükrében ez még várat magára, egyelőre mikrovezérlős környezetben az állapotgépes megvalósítást nem veszélyeztetik a gépi tanulási eljárások. Ennek legfőbb oka nem az ML technológia teljesítménye, hanem a fejlesztés kivitelezéséhez szükséges szerteágazó függőségi rendszer, azok nem harmonizált frissítése, inkompatibilitása.

#### **4.1.11 NVidia Jetson Xavier**

Bár az eszköz (38. ábra) jelenleg nem áll rendelkezésre, mégis a beágyazott rendszerek között érdemes megemlíteni, mivel nem általános célú eszközről van szó. Az Nvidia Jetson egy kis fogyasztású beágyazott rendszer, melyet kifejezetten gépi tanulási számítások megoldására fejlesztettek [48]. Ezekben a rendszerekben Nvidia Tegra processzor található, mely SoC-k (System on Chip) ARM magot is tartalmaznak.

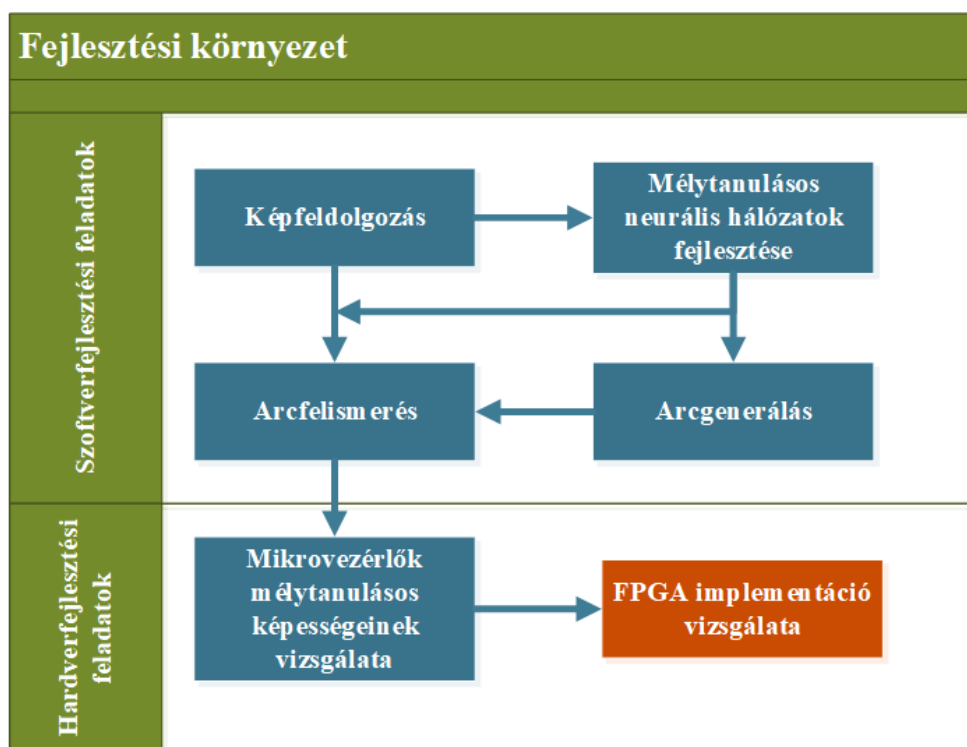
A rendszerhez szoftveres oldalról tartozik a Jetpack software development kit (SDK), amely tartalmazza a Linux for Tegra (L4T) operációs rendszert is. A Jetson Xavier sorozat 384 magos, 48 Tensor Core-os Nvidia Volta architektúrát, illetve 512 magos, 64 Tensor Core-os szintén Volta architektúrát tartalmaz [49].

Az eszköz felhasználása a Deep Learning HDL coder használatával azonos időintervallumban van tervezve, amennyiben addigra az beszerzésre kerül. A GPU Coder segítségével a kódgenerálás hasonlóképpen lehetséges erre a rendszerre is, mint a kiválasztott FPGA kártyára. Ez a fejlesztőkörnyezet szavatol egyfajta stabilitást, ami így kiküszöbölheti a TensorFlow Lite-tal kapcsolatos problémákat.



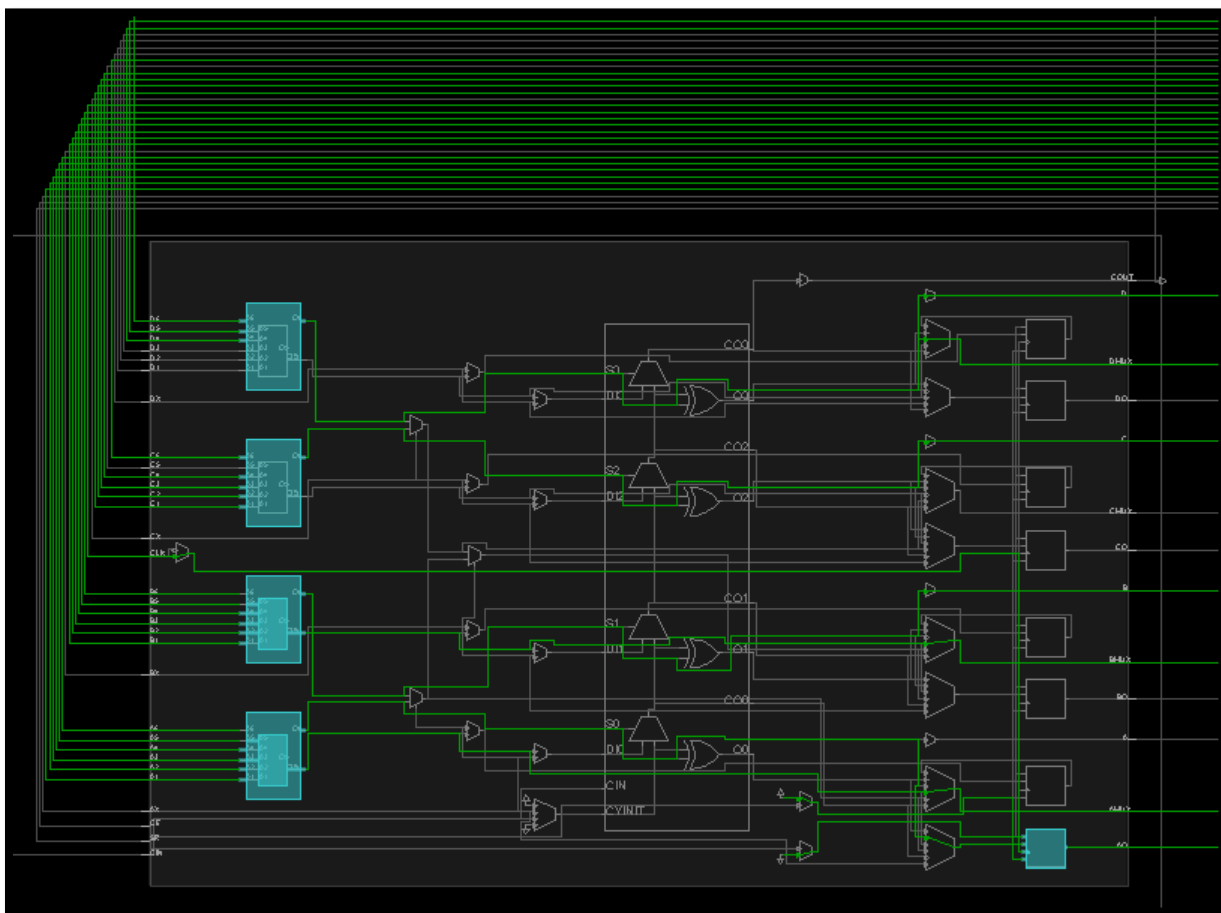
38. ábra: Nvidia Jetson Xavier beágyazott rendszer (forrás: <https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/>; 2022.02.25)

#### 4.1.12 FPGA implementáció



39. ábra: Fejlesztési környezet, FPGA implementáció (szerk. a szerző)

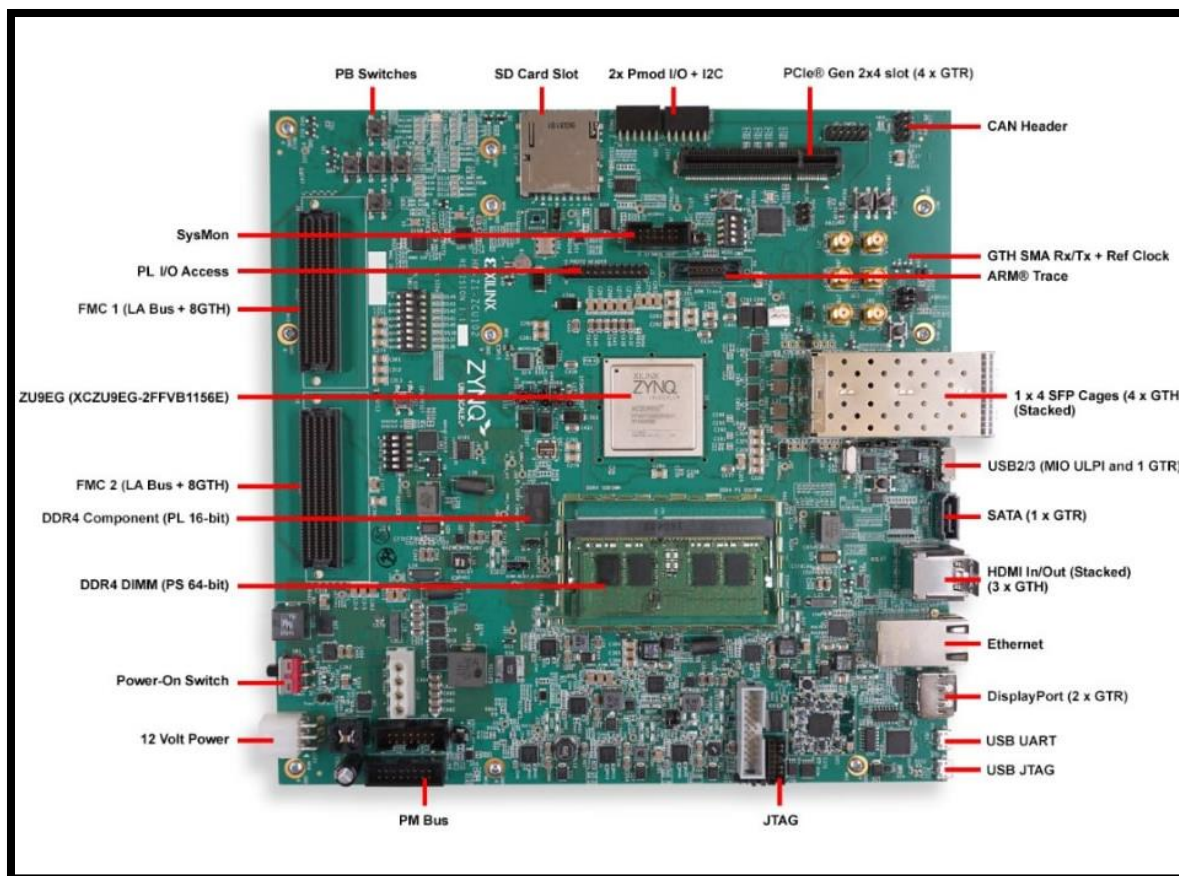
Az elérendő mérföldkövek közé tartozik (40. ábra) az arcfelismerő algoritmus FPGA-ba történő implementálása, mely a komplex mélytanulási hálózatok valósidejű futtatása miatt szükséges. Bár az FPGA órajele lényegesen kisebb egy korszerű személyi számítógépnél, a valós idejű, párhuzamosított program végrehajtás miatt számos kutatócsoport dolgozik a saját fejlesztésű hálózatok célhardveres implementálásán [50] [51] [52] [53]. A processzorokkal szemben előnye, hogy a programvégrehajtás teljes mértékig párhuzamosítható, mivel a hardverleíró nyelven leírt áramköri működés nem virtuális, hanem fizikailag is létrejön az eszközben (40. ábra).



**40. ábra: FPGA-ban implementált RTL kód (szerk. a szerző)**

A modell alapú fejlesztési irányelveket követve, első lépésként egy, a fejlesztő környezet által támogatott FPGA kártyára implementáljuk az arcfelismerő algoritmust. A választott kártya a Xilinx Ultrascale+ MPSoC ZCU102 fejlesztői rendszere (41. ábra).

A legfőbb szempont, amiért erre a kártyára esett a választás, hogy a MATLAB Deep Learning HDL Coder toolbox-sza támogatja, és lehetőséget biztosít arra is, hogy a betanított neurális hálózathoz „bit” fájlt generáljunk, az algoritmusunkat pedig HIL környezetben futtathassuk, teszteljük.



**41. ábra: A Zynq Ultrascale MPSoC ZCU102 fejlesztő kártya** (forrás: <https://hu.mouser.com/new/xilinx/xilinx-zynq-ultrascale-zcu102-eval-kit/>; 2022.02.16)

Az itt megszerzett tapasztalatok alapján folytatódik majd a fejlesztés. A projektterv szerint az FPGA implementálás lesz az a lépés, amikor a beágyazott rendszer képes lesz egyéb funkciókat is ellátni, kommunikálni szerverekkel, illetve vezérelni tudja a beléptető rendszer egyes fizikai komponenseit. Ezen a ponton már egy intellectual property mag (IP core) lesz az FPGA-ba kerülő MI algoritmus formátuma, aminek az implementálása részben MATLAB-bal, illetve a Xilinx gyári megoldásaival [53] történik majd.

# Összegzés

Jelen dolgozatban ismertetett kutatás hosszútávú célja tehát egy mélytanulási eljárásra épülő, arcfelismerés alapú beléptető rendszer prototípusának kifejlesztése. A dolgozatban a felhasználásra tervezett eszközrendszer került vizsgálatra, ami alapján definiálhatjuk céljainkat és a fejlesztési menetrendet. A jelenleg rendelkezésre álló eszközök segítségével a következő feladatokat sikerült végrehajtani, célokat elérni:

- Keras mélytanulási keretrendszer összeállítása Docker-ben:
  - Mélytanulási keretrendszer kialakítása.
- DCGAN arcgenerálás tesztelése processzoron:
  - Tesztadatok (képek) generálása az arcfelismerő algoritmushoz;
  - Ismerkedés a mélytanulási keretrendszerrel;
  - Tapasztalatgyűjtés a CPU-s számítási kapacitásokkal kapcsolatban.
- DCGAN arcgenerálás tesztelése kisteljesítményű GPU-n:
  - GPU beállítása a mélytanulási keretrendszerhez;
  - Tapasztalatgyűjtés a GPU-s számítási kapacitásokkal kapcsolatban.
- MATLAB/Simulink modell alapú keretrendszer összeállítása transfer learning-hez:
  - Modell alapú fejlesztési munkafolyamat megtervezése és kialakítása.
- ST Microelectronics STM32F746 mikrovezérlőre négyrétegű perceptron neurális hálózat betanítása szinuszos függvényre és felprogramozása TensorFlow Lite-tal:
  - Munkafolyamat kialakítása beágyazott rendszeren történő gépi tanulási fejlesztésekhez;
  - Beágyazott rendszerek gépi tanulással kapcsolatos képességeinek tesztelése.
- FPGA hardverimplementációs fázis fejlesztési feladatainak meghatározása:
  - Eszközök és az implementálás módjának meghatározása.

A projekt céljai és eszközei meghatározásra kerültek, mégis lényeges, hogy folyamatosan tájékozódjunk, az újonnan megjelenő technológiákat teszteljük, hatékonyságukat kielemezzünk. Ugyanakkor fontos, hogy a szembejövő előnyök mellett figyelembe vegyünk a fejlesztői környezetünk stabilitását, algoritmusaink hatékonyságát.

## Jövőbeli tervek:

- CNN hálózat fejlesztése transfer learning-gel;
- Egyéb szűrési, kép- és adatfeldolgozási műveletek implementálása, tesztelése;

- CNN hálózat GPU-n történő tanítása MATLAB/Simulink környezetben;
- FPGA implementáció megvalósítása;
- Beléptető rendszer prototípusának kiépítése.

# Ábrajegyzék

|                                                                                                                                                                                                                                              |    |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1. ábra: Idemia VisionPass (forrás: [3]) .....                                                                                                                                                                                               | 6  |
| 2. ábra: RFID alapú beléptető rendszer (szerk. a szerző).....                                                                                                                                                                                | 7  |
| 3. ábra: Gépi látás alapú beléptető rendszer (szerk. a szerző).....                                                                                                                                                                          | 9  |
| 4. ábra: Fejlesztési feladatok (szerk. a szerző) .....                                                                                                                                                                                       | 9  |
| 5. ábra: Mesterséges intelligencia csoportosítása (szerk. a szerző).....                                                                                                                                                                     | 11 |
| 6. ábra: Fejlesztési környezet, képfeldolgozás (szerk. a szerző) .....                                                                                                                                                                       | 14 |
| 7. ábra: Viola Jones [19] .....                                                                                                                                                                                                              | 15 |
| 8. ábra: Fejlesztési környezet, mélytanulósos neurális hálózatok (szerk. a szerző) .....                                                                                                                                                     | 15 |
| 9. ábra: Keras felépítése (szerk. a szerző) .....                                                                                                                                                                                            | 17 |
| 10. ábra: Modell alapú fejlesztéshez felhasznált szoftverek (szerk. a szerző) .....                                                                                                                                                          | 19 |
| 11. ábra: Simulink modell (szerk. a szerző) .....                                                                                                                                                                                            | 20 |
| 12. ábra: Fejlesztési környezet, arcgenerálás (szerk. a szerző) .....                                                                                                                                                                        | 25 |
| 13. ábra: A DCGAN Generátor hálózatának felépítése (szerk. a szerző) .....                                                                                                                                                                   | 26 |
| 14. ábra: A DCGAN működése (szerk. a szerző).....                                                                                                                                                                                            | 27 |
| 15. ábra: Tensorflow/Keras keretrendszer kialakítása (szerk. a szerző) .....                                                                                                                                                                 | 29 |
| 16. ábra: Teszt kimenete (szerk. a szerző).....                                                                                                                                                                                              | 31 |
| 17. ábra: Könyvtárszerkezet (szerk. a szerző).....                                                                                                                                                                                           | 32 |
| 18. ábra: Passzív hűtésű videokártya hűtőbordájának melegedése .....                                                                                                                                                                         | 34 |
| 19. ábra: Xeon vs. GT730 (szerk. a szerző) .....                                                                                                                                                                                             | 34 |
| 20. ábra: Arcgenerálás 49 epoch alatt (szerk. a szerző) .....                                                                                                                                                                                | 35 |
| 21. ábra: Fejlesztési környezet, konvolúciós neurális hálózat (szerk. a szerző) .....                                                                                                                                                        | 36 |
| 22. ábra: GoogleNet módosítása Deep Learning Toolbox segítségével (szerk. a szerző) .....                                                                                                                                                    | 37 |
| 23. ábra: Fejlesztési környezet, STM32 implementálás.....                                                                                                                                                                                    | 38 |
| 24. ábra: Állapotgép (Simulink Stateflow) (szerk. a szerző) .....                                                                                                                                                                            | 39 |
| 25. ábra: Véletlenszerű adat generálása (szerk. a szerző) .....                                                                                                                                                                              | 41 |
| 26. ábra: Szinusz függvény diszkrét adatpontjainak generálása (szerk. a szerző) .....                                                                                                                                                        | 42 |
| 27. ábra: Véletlenszerű adatpontok a szinusz függvényhez rendelve (szerk. a szerző) .....                                                                                                                                                    | 42 |
| 28. ábra: Adatok szegmentálása (szerk. a szerző) .....                                                                                                                                                                                       | 43 |
| 29. ábra: Négyrétegű perceptron hálózat felépítése (forrás: <a href="https://www.tensorflow.org/lite/microcontrollers/get_started_low_level">https://www.tensorflow.org/lite/microcontrollers/get_started_low_level</a> ; 2022.02.16) .....  | 44 |
| 30. ábra - Négyrétegű perceptron hálózat felépítése (forrás: <a href="https://www.tensorflow.org/lite/microcontrollers/get_started_low_level">https://www.tensorflow.org/lite/microcontrollers/get_started_low_level</a> ; 2022.02.16) ..... | 45 |

|                                                                                                                                                                                                                                                                          |    |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 31. ábra: Átlagos abszolút hiba (szerk. a szerző) .....                                                                                                                                                                                                                  | 46 |
| 32. ábra: Betanított szinusz függvény pontossága (szerk. a szerző) .....                                                                                                                                                                                                 | 46 |
| 33. ábra: A legenerált modellek (szerk. a szerző) .....                                                                                                                                                                                                                  | 47 |
| 34. ábra: Perceptron hálózatunk importálása STM32 Cube IDE-be (szerk. a szerző).....                                                                                                                                                                                     | 48 |
| 35. ábra: Terminál kimenet (szerk. a szerző) .....                                                                                                                                                                                                                       | 51 |
| 36. ábra: Teszt eredménye 2000 mintával (szerk. a szerző).....                                                                                                                                                                                                           | 52 |
| 37. ábra: Teszt eredménye 3000 mintával (szerk. a szerző).....                                                                                                                                                                                                           | 52 |
| 38. ábra: Nvidia Jetson Xavier beágyazott rendszer (forrás:<br><a href="https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/">https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/</a> ;<br>2022.02.25)..... | 54 |
| 39. ábra: Fejlesztési környezet, FPGA implementáció (szerk. a szerző) .....                                                                                                                                                                                              | 54 |
| 40. ábra: FPGA-ban implementált RTL kód (szerk. a szerző).....                                                                                                                                                                                                           | 55 |
| 41. ábra: A Zynq Ultrascale MPSoC ZCU102 fejlesztő kártya (forrás:<br><a href="https://hu.mouser.com/new/xilinx/xilinx-zynq-ultrascale-zcu102-eval-kit/">https://hu.mouser.com/new/xilinx/xilinx-zynq-ultrascale-zcu102-eval-kit/</a> ; 2022.02.16).....                 | 56 |

# Irodalomjegyzék

- [1] Otti Csaba: „Biometriaalapú beléptetőrendszerek alkalmazhatósága tömegtartózkodású helyeken”, Óbudai Egyetem Doktori (PhD) értekezés, 2019.
- [2] F. Chollet: „Back to Neural Networks” in *Deep Learning with Python*, New York, Manning Publications, 2018, p. 17.
- [3] Idemia: „VisionPass Installation Guide,” 2019.  
<https://biometricdevices.idemia.com/s/visionpass>. [Hozzáférés dátuma: 2022.03.02.]
- [4] Négyesi Imre: „A mesterséges intelligencia és az etika” *Hadtudomány*, 30. kötet, 1. szám, pp. 103-113, 2019.
- [5] Dr. Németh András, Dr. Hegedűs Ernő, Wippelhauser András, Simó Réka: „A katonai alkalmazású autonóm terepjáró járművek fejlesztésének egyes kérdései I. rész” *Haditechnika*, 53. évfolyam, 4. szám, pp. 11-16, 2019.
- [6] Tóth Levente, Tóth Attila: „Személyazonosítás”, *Biztonságtechnika*, Budapest, Nemzeti Közszerológati és Tankönyv Kiadó Kft, 2014, pp. 201-209.
- [7] Otti Csaba: „Biometrikus rendszerek felhasználói minta pozicionálásának kérdései”, *Tavaszi Szél 2016 Konferencia*, 3. kötet, pp. 251-260, 2016.
- [8] Németh András, Virágh Krisztián: „Mesterséges Intelligencia és haderő - A mesterséges intelligencia fejlődéstörténete I. rész”, *Haditechnika*, 56. évfolyam, 1. szám, pp. 17-22., 2022.
- [9] F. Chollet: „Don't Believe in the Short-term Hype”, *Deep Learning with Python*, New York, Manning Publications, 2018, pp. 12-13.
- [10] Y. B. A. C. Ian Goodfellow: „Historical Trends in Deep Learning”, *Deep Learning*, Cambridge, The MIT Press, 2016, pp. 12-28.
- [11] F. Chollet: „Four Branches of Machine Learning”, *Deep Learning with Python*, New York, Manning Publications, 2018, pp. 94-96.
- [12] Németh András, Virágh Krisztián: „Mesterséges intelligencia és a haderő - A mesterséges intelligencia területei III. rész”, *Haditechnika*, 56. évfolyam, 3. szám, pp. 2-7, 2022.
- [13] Négyesi Imre: „A mesterséges intelligencia katonai felhasználásának társadalmi kérdései”, *Honvédegi Szemle*, 149. kötet, 1. szám, pp. 133-144, 2021.
- [14] E. Bizottság, „A BIZOTTSÁG KÖZLEMÉNYE AZ EURÓPAI PARLAMENTNEK, A TANÁCSNAK, AZ EURÓPAI GAZDASÁGI ÉS SZOCIÁLIS BIZOTTSÁGNAK ÉS A RÉGIÓK BIZOTTSÁGÁNAK” 2018.12.07.

- [15] M. I. Koalíció, „Magyarország Mesterséges Intelligencia Stratégiája”, *Információs és Technológiai Minisztérium*, 2020.
- [16] Virágh Krisztián: „A mesterséges intelligencia fejlődési tendenciái és katonai alkalmazásának perspektívái”, *Nemzeti Köszolgálati Egyetem Hadtudományi és Honvédtisztképző Kar Intézményi Tudományos Diákköri Konferencia*, 2020.
- [17] Négyesi Imre: „A mesterséges intelligencia és a hadseregek”, *Hadtudomány*, 29. kötet, 3. szám, pp. 71-79, 2020.
- [18] Dr. Németh András, Dr. Hegedűs Ernő, Wippelhauser András, Simó Réka: „A katonai alkalmazású autonóm terepjáró járművek fejlesztésének egyes kérdései II. rész”, *Haditechnika*, 53. évfolyam, 5. szám, pp. 2-7, 2019.
- [19] P. Viola a. M. Jone:, „Robust real-time face detection JCV 57(2), 137–154, 2004”, *IEEE*, 2004.
- [20] M. Ali: „How The Facial Detection Algorithms Work: Viola-Jones Algorithm And OpenCV”, 2021.02.22. <https://medium.datadriveninvestor.com/how-the-facial-detection-algorithms-work-viola-jones-algorithm-and-opencv-bd694936512f>. [Hozzáférés dátuma: 2021.03.10.]
- [21] G. S. C. R. M. K. C. M. D. Q. V. L. M. Z. M. M. R. A. S. P. T. K. Y. a. A. Y. N. J. Dean: „Large scale distributed deep networks”, *NIPS*, pp. 1232-1240, 2012.
- [22] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray és B. Steiner: „TensorFlow: A System for Large-Scale Machine Learning”, 2016. <https://gitlab.com/p4892/camera-ai/-/wikis/uploads/45b62af591cf93d409815ad408cf020a/osdi16-abadi.pdf>. [Hozzáférés dátuma: 2022.01.15.]
- [23] E. S. J. D. S. K. J. L. R. G. S. G. a. T. D. Y. Jia: „Caffee: Convolutional architecture for fast feature embedding”, *ACM Multimedia*, pp. 675-678, 2014.
- [24] G. A. A. A. C. A. D. B. N. B. F. B. J. B. A. B. A. B. Y. B. A. B. J. B. V. B. J. B. S. N. B. N. B.-L. X. B. A. d. R. Al-Rfou: „Theano: A Python framework for fast”, 2016. <https://arxiv.org/abs/1605.02688>. [Hozzáférés dátuma: 2022.02.26.]
- [25] S. B. a. J. M. R. Collobert: „Torch: A modular machine learning software library”, 30 10 2002.10.30. <http://infoscience.epfl.ch/record/82802/files/rr02-46.pdf>. [Hozzáférés dátuma: 2022.02.05.]
- [26] Microsoft: „The Microsoft Cognitive Toolkit”, 2017. <https://docs.microsoft.com/en-us/cognitive-toolkit/>. [Hozzáférés dátuma: 2022.02.02]
- [27] G.-T. B. T. D. L. T. C. G. S. H. I. Y. N. A. L. K. O. K. K. A. E. E. Tian Jin: „Compiling ONNX Neural Network Models Using MLIR”, 2020.10.01. <https://arxiv.org/abs/2008.08272>. [Hozzáférés dátuma: 2022.02.02]

- [28] F. Chollet: „Introduction to Keras”, in *Deep Learning with Python*, New York, Manning Publications, 2018, pp. 61-64.
- [29] L. Kechiche: „Hardware acceleration for deep learning of image classification”, *2021 International Conference of Women in Data Science at Taif University (WiDSTaif)*, pp. 1-5, 2021.
- [30] Mathwork: „MATLAB Coder”, <https://uk.mathworks.com/products/matlab-coder.html>. [Hozzáféres dátuma: 2022.02.02]
- [31] Mathworks: „Simulink Coder”, <https://uk.mathworks.com/products/simulink-coder.html>. [Hozzáféres dátuma: 2022.02.02]
- [32] Mathworks: „Parallel Computing”, <https://uk.mathworks.com/products/parallel-computing.html>. [Hozzáféres dátuma: 2022.02.02]
- [33] Mathworks, „GPU Coder”, <https://uk.mathworks.com/products/gpu-coder.html>. [Hozzáféres dátuma: 2022.02.02]
- [34] Mathworks: „Signal Processing”, <https://uk.mathworks.com/products/signal.html>. [Hozzáféres dátuma: 2022.02.02]
- [35] Mathworks: „Image Processing”, <https://uk.mathworks.com/products/image.html>. [Hozzáféres dátuma: 2022.02.02]
- [36] Mathworks: „Computer Vision”, <https://uk.mathworks.com/products/computer-vision.html>. [Hozzáféres dátuma: 2022.02.02]
- [37] Mathworks: „Statistics Toolbox”, <https://uk.mathworks.com/products/statistics.html>. [Hozzáféres dátuma: 2022.02.02]
- [38] Mathworks: „Deep Learning”, <https://uk.mathworks.com/products/statistics.html>. Hozzáféres dátuma: 2022.02.02]
- [39] Mathworks: „Image Acquisition”, <https://uk.mathworks.com/products/image-acquisition.html>. [Hozzáféres dátuma: 2022.02.02]
- [40] Mathworks: „Instrument Control”, <https://uk.mathworks.com/products/instrument.html>. [Hozzáféres dátuma: 2022.02.02]
- [41] Mathworks: „Deep Learning HDL Coder”, <https://uk.mathworks.com/products/deep-learning-hdl.html>. [Hozzáféres dátuma: 2022.02.02]
- [42] Mathworks: „Fixed Point”, <https://uk.mathworks.com/products/fixed-point-designer.html>. [Hozzáféres dátuma: 2022.02.02]
- [43] Mathworks: „HDL Coder”, <https://uk.mathworks.com/products/hdl-coder.html>. [Hozzáféres dátuma: 2022.02.02]

- [44] Mathworks: „HDL Verifier”, <https://uk.mathworks.com/products/hdl-verifier.html>. [Hozzáférés dátuma: 2022.02.02]
- [45] J. P.-A. M. M. B. X. D. W.-F. S. O. A. C. Y. B. Ian Goodfellow: „Generative Adversarial Networks”, 2014.06.10. <https://arxiv.org/abs/1406.2661>. [Hozzáférés dátuma: 2022.01.03.]
- [46] F. Chollet: „Introduction to Generative Adversarial Networks”, *Deep Learning with Python*, New York, Manning Publications, 2018, pp. 305-313.
- [47] Y. B. A. C. Ian Goodfellow: „Generative Adversarial Networks” *Deep Learning*, Cambridge, The MIT Press, 2016, pp. 690-693.
- [48] K. Freund: „NVIDIA Reveals Xavier SOC Details”, 2018.08.24. <https://www.forbes.com/sites/moorinsights/2018/08/24/nvidia-reveals-xavier-soc-details/#7ac42dbb47e5>. [Hozzáférés dátuma: 2022.01.10.]
- [49] S. S. K. Srinivasan: „Performance Characterization of NVIDIA Xavier”, *Academia Letters*, 2021.
- [50] V. P. E. Kavvousanos: „Optimizing Deep Learning Decoders for FPGA Implementation”, *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 271-272, 2021.
- [51] J. W. J. Han Sung Lee: „Accelerating Deep Neural Networks Using FPGAs and ZYNQ”, *IEEE Region 10 Symposium (TENSYP)*, 2021.
- [52] L. D. L. W. F. C. Xuelei Li: „FPGA Accelerates Deep Residual Learning for Image Recognition”, *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 837-840, 2017.
- [53] Y. Z. H. L. Y. Z. L. W. Jian Zhao: „HLS-Based FPGA Implementation of Convolutional Deep Belief Network for Signal Modulation Recognition”, *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium*, pp. 6985-6988, 09 2020.



NEMZETI KUTATÁSI, FEJLESZTÉSI  
ÉS INNOVÁCIÓS HIVATAL

AZ NKFI ALAPBÓL  
MEGVALÓSULÓ  
PROJEKT