

MINISTERUL EDUCAȚIEI



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

**DETECȚIA CODULUI NUMERIC PERSONAL FOLOSIND O
SOLUȚIE HIBRIDĂ BAZATĂ PE REȚELE NEURONALE**

LUCRARE DE LICENȚĂ

Absolvent: **Marius-Adrian STOICA**

Coordonator: **Asist. Prof. Drd. Ing. Mircea Paul**
științific: **MUREŞAN**

2021



FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Marius-Adrian STOICA**

**DETECȚIA CODULUI NUMERIC PERSONAL FOLOSIND O SOLUȚIE
HIBRIDĂ BAZATĂ PE REȚELE NEURONALE**

1. **Enunțul temei:** *Proiectarea unui sistem de detecție a cifrelor care formează codul numeric personal de pe o imagine cu actul de identitate.*
2. **Conținutul lucrării:** *Introducere, Obiectivele Proiectului, Studiu Bibliografic, Analiză și Fundamentare Teoretică, Proiectare de Detaliu și Implementare, Testare și Validare, Manual de Instalare și Utilizare, Concluzii, Bibliografie.*
3. **Locul documentării:** Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare
4. **Consultanți:** Asist. Prof. Drd. Ing. Mircea Paul MUREŞAN
5. **Data emiterii temei:** 1 noiembrie 2020
6. **Data predării:** 6 septembrie 2021

Absolvent: Marius-Adrian Stoica

Coordonator științific: Asist. Prof. Drd. Ing. Mircea Paul
MUREŞAN



**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE**

**Declarație pe proprie răspundere privind
autenticitatea lucrării de licență**

Subsemnatul(a) Stoica Marius-Adrian, legitimat(ă) cu CI seria CJ nr. 161759 CNP 1960821260063, autorul lucrării Detection Codului Numeric Personal Folosind o Soluție Hibridă Bazată pe Rețele Neuronale elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea Calculatoare din cadrul Universității Tehnice din Cluj-Napoca, sesiunea septembrie a anului universitar 2021, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

In cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data

Nume, Prenume

06.09.2021

Stoica Marius-Adrian

Semnătura

Cuprins

Capitolul 1. Introducere	1
1.1. Inteligenta artificială în viața cotidiană.....	1
1.2. Definiția și domenii ale inteligenței artificiale.....	2
1.3. Inteligenta artificială în birocratie	3
1.4. Contextul proiectului	4
1.5. Structura lucrării	4
Capitolul 2. Obiectivele Proiectului.....	5
2.1. Motivația	5
2.2. Obiectivele	5
2.3. Grupul țintă	6
2.4. Cerințele funcționale și cele non-funcționale ale sistemului.....	8
Capitolul 3. Studiu Bibliografic	12
3.1. Machine learning pe FPGA.....	12
3.1.1. Antrenarea reței neuronale	12
3.2. Rețelele neuronale pe FPGA.....	15
3.2.1. Antrenarea rețelei neuronale	15
3.2.2. Reprezentarea numerelor	16
3.2.3. Funcția de activare	17
3.2.4. Folosirea IP Cores	17
3.2.5. Proiectarea neuronului	18
3.2.6. Proiectarea layer-urilor	18
3.3. Recunoașterea cifrelor scrise de mână folosind rețelele neuronale artificiale	19
3.3.1. Introducere	19
3.3.2. Descrierea dataset-ului	19
3.3.3. Inițializarea clasificatorului.....	20
3.3.4. Testarea rețelei neuronale	21
3.3.5. Discuție pe baza eșecurilor	22
3.4. Optical Character Recognition (OCR)	22

3.4.1.	Introducere	22
3.4.2.	Pipeline de preprocesare	22
3.4.3.	Extragerea caracteristicilor	23
3.5.	Automatizarea aplicațiilor desktop	24
3.5.1.	Introducere și motivație	24
3.5.2.	Winium	25
Capitolul 4. Analiză și Fundamentare Teoretică	26	
4.1.	Flow-ul aplicațiilor.....	26
4.1.1.	VHDL	26
4.1.2.	Script licență	27
4.1.3.	Aplicație desktop.....	28
4.1.4.	Automatizare Winium.....	29
4.1.5.	MySQL	29
4.1.6.	Rețea neuronală conoluțională	30
4.2.	Diagrame flow pentru cazuri specifice	31
4.2.1.	Trimitere imagine spre procesare.....	31
4.2.2.	Înregistrarea utilizatorului în aplicația desktop.....	32
4.2.3.	Vizualizare imagine librărie.....	33
4.2.4.	Rularea aplicațiilor necesare procesării imaginii	34
4.3.	Algoritmi și protocoale utilizate	35
4.3.1.	Algoritmul de conversie a numerelor din virgulă mobilă în fixed point representation	35
4.3.2.	Algoritm identificare caractere CNP.....	37
4.3.3.	Algoritm conversie din caractere ASCII în siruri de biți	37
4.4.	Tehnologii.....	38
Capitolul 5. Proiectare de Detaliu și Implementare	41	
5.1.	VHDL	41
5.1.1.	UART_TX	41
5.1.2.	UART_RX	43
5.1.3.	Reprezentarea numerelor în fixed point	45
5.1.4.	Memoria BRAM	47
5.1.5.	Primul ANN	49
5.1.6.	ANN Final.....	52
5.2.	Python	56

5.2.1.	Rețeaua neuronală artificială.....	56
5.2.2.	Rețeaua neuronală conoluțională	58
5.2.3.	Optical Character Recognition.....	61
5.2.4.	Script licență	63
5.3.	Winium	64
5.4.	Delphi.....	66
5.4.1.	ULoginP	66
5.4.2.	USignUpP	67
5.4.3.	UPrincP	68
5.5.	MySQL	72
Capitolul 6.	Testare și validare	74
Capitolul 7.	Manual de Instalare și Utilizare	79
7.1.	Manual de Instalare.....	79
7.1.1.	Instalare Anaconda Navigator.....	79
7.1.2.	Crearea unui environment în Anaconda Navigator.....	80
7.1.3.	Instalarea librăriilor necesare aplicației în environment	80
7.1.4.	Instalare Xilinx Vivado	81
7.1.5.	Instalare XAMPP	81
7.1.6.	Instalare Hterm.....	81
7.2.	Manual de Utilizare.....	82
7.2.1.	Rulare MySQL folosind XAMPP	82
7.2.2.	Rulare script licență	82
7.2.3.	Rulare Winium Desktop Driver	84
7.2.4.	Rulare proiect VHDL.....	84
7.2.5.	Rulare aplicație desktop	86
7.2.6.	Funcționalități oferite de aplicația VHDL.....	89
Capitolul 8.	Concluzii	90
8.1.	Posibile îmbunătățiri ale proiectului	93
8.2.	Dezvoltări ulterioare	94
Bibliografie	95	

Capitolul 1. Introducere

1.1. Inteligența artificială în viața cotidiană

Inteligența artificială. Un termen, sau mai bine zis, o sintagmă, care pentru mulți oameni definește ‘tehnologia viitorului’. Când apare o știre despre ‘Inteligența Artificială’, când se discută despre aceasta în diferite medii (televiziune, internet, radio), majoritatea oamenilor au tendință să credă că veștile, informațiile, noutățile despre acest domeniu nu au un impact direct asupra lor. Inteligența artificială, cu sau fără acordul populației, a ajuns să facă parte din viața cotidiană. Implicarea inteligenței artificiale, și a diferitelor ei subramuri, poate trece neobservată pentru persoanele fără un background minim în acest segment al tehnologiei. Cu toate acestea, A.I. (Artificial Intelligence – Inteligența Artificială) are un aport din ce în ce mai mare în activitățile cotidiene.

Pentru mulți oameni, ziua începe prin a-și verifica telefonul. Multe telefoane folosesc recunoașterea facială pentru deblocare. Recunoașterea facială are la bază inteligența artificială, cu predilecție M.L. (Machine Learning) și Computer Vision. Serviciile de e-mail folosesc clasificarea automată inteligentă a inbox-ului, folosind algoritmi de N.L.P. (Natural Language Processing) și Machine Learning, precum și clasificarea diferitelor e-mail-uri în funcție de conținutul lor. În procesul de scriere a mail-urilor, spell checker-ul este folosit pentru a reliefa greșelile gramaticale, și implicit a le corecta, prin utilizarea Natural Language Processing. Rețelele de socializare folosesc inteligența artificială pentru a oferi informații bazate pe istoricul de pe internet al utilizatorului. Google deservește utilizatorii cu reclame pe baza accesărilor sale. Căutările predictive au de asemenea la bază informații colectate precum vârstă, locația, iar inteligența artificială încearcă pe baza acestor date să prezică posibilele căutări ale utilizatorului. În drumul către locul de muncă, oamenii pot folosi aplicații de navigație care folosesc A.I. și M.L., precum Google Maps, pentru a primi date în timp real despre trafic, cu scopul de a alege ruta cea mai convenabilă. Pentru a ajunge la locul de muncă, oamenii pot folosi mașini inteligente, în unele țări existând mijloace de transport în comun (autobuze inteligente) sau taxiuri fără șofer. Aplicațiile de voce pentru planificarea întâlnirilor, update-ul privind condițiile meteorologice, folosesc N.L.P. pentru a înțelege ceea ce utilizatorii le transmit. Platformele de streaming precum Netflix sau YouTube fac recomandări personalizate de filme sau video-uri folosind inteligența artificială.[24] Alte platforme verifică interesele unui utilizator, apoi creează un ‘profil’ al utilizatorului, verifică interesele altor utilizatori care se aseamănă aceluia profil, și ulterior transmit utilizatorului inițial recomandări ale celor care respectă acel pattern, acel profil.

Acestea sunt exemple comune prin care inteligența artificială afectează viața de zi cu zi a populației. Pe lângă acestea, oamenii prestează activități proprii, nemunite. Copiii merg la școală, adulții merg la locul de muncă. Multe dintre aceste activități implică un contact cu inteligența artificială. În educație există programe de plagiat care funcționează pe baza A.I., în sport sunt ceasuri inteligente pentru monitorizarea semnelor vitale, a performanței, în domeniul sănătății sunt dispozitive de monitorizare a sănătății. Agricultura poate folosi inteligența artificială pentru a monitoriza sănătatea solului, a plantelor. În domeniul hotelier există camere de hotel inteligente, în imobiliare pentru analiza pieței, în asigurări pentru identificare potențialelor riscuri. Casele inteligente folosesc A.I. pentru a putea fi controlate. În domeniul organizării de evenimente, pot exista sisteme de recunoaștere facială pentru a scana participanții. În domeniul aerospațial, A.I. poate prezice condițiile meteorologice sau poate fi folosit pentru

autopilot. Alte domenii precum securitatea cibernetică, apărare, politică, cumpărături online, comunicări, transport pot și folosesc sub o formă sau alta inteligența artificială. [25]

Astfel, inteligența artificială s-a infiltrat din ce în ce mai mult în viața cotidiană, fără ca majoritatea oamenilor să realizeze acest fapt. Ea poate fi regăsită atât în activitățile comune majorității oamenilor precum folosirea rețelelor sociale, a serviciilor de e-mail, a aplicațiilor de navigație, de voce sau platforme de streaming, cât și în activități necomune tuturor oamenilor, individuale, în funcție de locul de muncă sau domeniul de interes: educație, sport, sănătate, agricultură, home automation, organizare de evenimente, politică, comunicări, transport și multe altele.

1.2. Definiția și domenii ale inteligenței artificiale

Motivația redactării acestui subcapitol este dată, pe de o parte, de o oarecare obligativitate de a contextualiza sintagma “Inteligență Artificială”, regăsită cu precădere în lucrarea de față, iar pe de altă parte, de necesitatea folosirii unor termeni, cuvinte de specialitate din domeniul inteligenței artificiale. Astfel, scopul este crearea unui vocabular, astfel încât să se înțeleagă distincția dintre “Inteligență Artificială” și principalele domenii pe care aceasta le guvernează.

Pentru a oferi o definiție a inteligenței artificiale, este necesară, în prealabil, divizarea acestui termen: ‘inteligență’, respectiv ‘artificială’, și punerea în context, atât din perspectiva lexicului, cât și al științei. Din punctul de vedere al lexicului, inteligența reprezintă “capacitatea de a înțelege ușor și bine, de a sesiza ceea ce este esențial, de a rezolva situații sau probleme noi, pe baza experienței acumulate anterior” [1]. Din punct de vedere științific, cei mai importanți factori ai inteligenței sunt: învățarea generalizată, cea care permite celui care învăță să fie capabil să performeze în situații cu care nu s-a mai întâlnit, raționamentul, adică schițarea unei concluzii potrivite situației în cauză, rezolvarea problemei, percepția prin analizarea unui mediu, a caracteristicilor sale și a relațiilor dintre obiecte și înțelegerea limbajului, urmărind sintaxa și regulile, asemănător oamenilor. [4] A doua parte a sintagmei, artificială, se referă la ceva nenatural, care imită un produs al naturii. Astfel, inteligența artificială este un sistem, capabil să rezolve cerințe care necesită în mod normal inteligență umană.

Este importantă prezentarea domeniilor inteligenței artificiale, întrucât subcapitolul anterior conține termeni de specialitate care, dacă nu sunt bine definiți, ar putea genera confuzie. Domeniile inteligenței artificiale sunt: machine learning, deep learning, robotică, sisteme expert, fuzzy logic, natural language processing, computer vision.

1. Machine learning: permite calculatoarelor, mai degrabă să ia decizii pe baza datelor, decât să fie programate pentru a rezolva o anumită cerință.
2. Deep learning: simulează modul de a funcționa al creierului uman în procesarea datelor și crearea de pattern-uri în luarea deciziilor.
3. Robotics: ramură a tehnologiei care implică folosirea roboților, în colaborare cu alte domenii precum electronica, computer science, inteligență artificială sau mecatronica.
4. Expert systems: program care utilizează inteligența artificială pentru a simula modul de luare a deciziilor a omului.
5. Fuzzy logic: încercarea de a imita raționamentul uman, în procese de luare a deciziilor care implică mai multe posibilități.
6. Natural language processing: ramură a inteligenței artificiale care ajută calculatorul să înțeleagă, să interpreteze și să manipuleze limbajul uman.

7. Computer vision: scopul este de a replica capacitatele viziunii umane (human vision), prin recunoașterea obiectelor prezente și a caracteristicilor sale precum texturi, culori, mărimi pentru a obține o descriere cât mai amplă a unei imagini. [3]

1.3. Inteligența artificială în birocratie

Digitalizarea birocratiei este un proces care a luat amploare în ultimul deceniu, datorită evoluției în domeniul inteligenței artificiale. Acest proces poate fi privit din mai multe perspective.

La nivel global, un pas important în sensul digitalizării birocratiei a fost făcut de DBC (Digital Bureaucracy). DBC este un proiect care va permite oamenilor efectuarea tranzacțiilor biocratice folosind o rețea blockchain, simplu și rapid. DBC este un sistem biocratic descentralizat, care urmărește standardizarea documentelor oficiale. DBC combină inteligența artificială și tehnologia blockchain în vederea soluționării procedurilor biocratice, asigurând rapiditate prin descentralizare, simplitate prin standardizare, și securitate prin encripția datelor. Domeniile pe care DBC le va influența sunt: educația prin distribuirea cărților în format electronic, afișarea notelor pe cataloage electronice, imobiliar, prin procesarea mai rapidă a documentelor de vânzare/cumpărare, consiliu local, prin eliminarea documentelor fizice, a cozilor de la ghișee, medical, prin crearea unei conexiuni între spitale și farmacii pentru distribuirea documentelor în format electronic și multe alte domenii. [5]

La nivel național, a fost lansat ghidul de digitalizare pentru primăriile din România, cu scopul de a reduce birocracia. De asemenea, în contextul epidemiologic actual, eliminarea cozilor sau reducerea lor vine în sprijinul măsurilor de siguranță impuse. Zitec, o companie de IT este responsabilă pentru crearea acestui ghid. Un avantaj al României în sensul digitalizării birocratiei este viteza bună a internetului. [6]

La nivel local, Consiliul Județean Cluj și Primăria Municipiului Cluj-Napoca au luat măsuri în privința digitalizării instituțiilor. În 2020, Consiliul Județean Cluj a lansat un concurs în vederea digitalizării instituției, având ca scop devenirea primului ‘județ SMART’ al României. Beneficiarul acestui proiect este cetățeanul de rând. Scopul este de a asigura un proces rapid în domeniul autorizării lucrărilor de construcție, digitalizarea depunerii documentelor, digitalizarea procedurilor aferente actelor realizate de funcționari. Primăria Municipiului Cluj-Napoca a introdus în 2018 primul “funcționar public virtual” din România, numit Antonia. Antonia deservește 24 de ore din 24 cetățenii în vederea soluționării problemelor cu municipalitatea. De asemenea, proiectul Antonia ajungea în 2019 să proceseze 90 de tipuri de cereri. Proiectul a fost un real succes, motiv pentru care, Consiliul Local Cluj-Napoca a decis alocarea unei sume de bani în vederea creării unei holograme a unei persoane care ar discuta în locul funcționarului de la ghișeu. Astfel, hologramei i-ar putea fi adresate întrebări cu privire la diferite informații, asemenea întrebărilor puse funcționarului de la primărie. Holograma nu urma să dea soluția unei probleme, ci ar fi redirecționat persoana care dorea informații către departamentul corespunzător nevoii în cauză. [7]

Așadar, digitalizarea birocratiei este un obiectiv la nivel global. Scopul este de a simplifica procesul de transmitere a documentelor, eliminarea timpilor de așteptare la cozile fizice, securizarea informațiilor și a actelor, standardizarea anumitor procese și documente. Obiectivul acestui subcapitol este prezentarea principalelor avantaje pe care le implică digitalizarea, din moment ce aplicația de licență poate ajuta în această zonă a digitalizării birocratiei, fiind de asemenea o soluție viabilă în contextul epidemiologic actual.

1.4. Contextul proiectului

Pe fondul acestui trend al digitalizării, proiectul de licență propune o alternativă proceselor de preluare și prelucrare a datelor, folosind inteligență artificială și domenii pe care aceasta le guvernează, precum machine learning sau computer vision.

De asemenea, în contextul epidemiologic actual, găsirea oricărei măsuri de prevenire a răspândirii coronavirusului este binevenită. Astfel, eliminarea cozilor fizice ar contribui la distanțarea socială, atât de promovată în ultima vreme. Eliminarea acestor cozi fizice, de la ghișee sau alte puncte de preluare și prelucrare a datelor, ar duce și la înlăturarea timpilor de aşteptare aferenți. Toate acestea ar oferi viteza procesului de preluare și prelucrare a datelor de pe diverse formate, de la abonamente, documente, acte și introducerea informațiilor în mediul digital, baze de date sau spreadsheet-uri.

Preluarea automată a datelor poate oferi și un anumit nivel de securitate, în cazul în care informațiile preluate ar fi automat encriptate, eliminându-se astfel riscul de a se stoca informații "plain text" în baze de date, sau chiar mai rău, păstrarea datelor confidențiale pe hârtie. Nu în ultimul rând, preluarea automată a datelor ar putea duce la înlocuirea documentelor fizice, implicit a foii de hârtie, cu documente în format electronic, fiind o 'gură de aer' pentru natură.

1.5. Structura lucrării

În cele ce urmează, vor fi prezentate succint, principalele capitole ale lucrării de licență, respectiv rolul pe care îl deservesc.

În al doilea capitol sunt prezentate motivația creării sistemului, principalele obiective ale suitei de aplicații, grupul țintă căruia i se adresează sistemul, și cerințele funcționale, respectiv non-funcționale, ale aplicației.

Al treilea capitol, studiu bibliografic, prezintă principalele surse de inspirație pentru implementarea diferitelor componente ale suitei de aplicații.

Următorul capitol elaborează flow-urile aplicațiilor care formează întregul sistem, diagrame flow pentru cazuri specifice, algoritmii și protocolele utilizate, respectiv principalele tehnologii folosite.

În al cincilea capitol, proiectare de detaliu și implementare, sunt prezentate, în amănunt, principalele componente ale aplicației. Sistemul este alcătuit dintr-o suită de aplicații. Fiecare dintre aceste aplicații este descrisă, în detaliu, în capitolul de față.

Al sășelea capitol, testare și validare, prezintă modalitățile de testare ale sistemului. Pentru a ilustra testarea și validarea sistemului, este necesară prezentarea în detaliu a testării, pe diferite module ale aplicației.

Următorul capitol conține manualul de instalare și utilizare al aplicației. Sunt prezentate pașii pentru instalarea aplicațiilor necesare rulării suitei de programe, respectiv modalitatea de a rula aplicațiile.

În încheiere, concluziile au rolul de a reliefa principalele avantaje ale aplicației. De asemenea, sunt prezentate posibilele îmbunătățiri, care ar putea fi aduse proiectului de față, îmbunătățiri fezabile. Nu în ultimul rând, sunt descrise posibilitățile de dezvoltare ulterioară ale aplicației.

Capitolul 2. Obiectivele Proiectului

Principalul scop al lucrării de față este crearea unei aplicații care preia o imagine a cărții de identitate, localizează cele 13 cifre ale codului numeric personal, și clasifică fiecare cifră, folosind rețele neuronale conoluționale. Cifrele sunt detectate, iar rezultatele conoluțiilor aplicate pe acestea sunt trimise la placa de dezvoltare Basys3, folosind protocolul de transmitere UART. Rețea neuronală artificială implementată pe placa FPGA primește atât rezultatele intermediare, cât și modelul rețelei (weights-urile) și clasifică fiecare cifră în parte. Rezultatul final este afișat pe afișorul cu 7 segmente, folosind un barrell shifter.

2.1. Motivația

Principala motivație este căutarea unei metode de eficientizare a diferitelor procese care implică interacțiunea umană. Aplicația ar putea minimiza timpul de așteptare la coadă, sau chiar să îl elime. În scenariul minimizării timpului de așteptare, coada fizică la un ghișeu, spre exemplu, ar exista în continuare, dar persoana de la ghișeu ar trebui doar să facă o poză actului de identitate, iar datele ar fi automat introduse în baza de date. În scenariul eliminării cozii fizice, imaginea cu actul de identitate ar fi trimisă pe mail, datele urmândă a fi procesate automat.

Având în vedere contextul epidemiologic actual, eliminarea cozilor și implicit a interacțiunii umane necesare vine ca o măsură de preventie a răspândirii virusului. De multe ori, la ghișee, pentru că spațiul este restrâns, oamenii care stau la coadă sunt nevoiți să aștepte în afara incintei, indiferent de condițiile meteorologice.

De asemenea, aplicația vine ca o alternativă la preluarea datelor și scrierea lor în diferite registre fizice, contribuind astfel la reducerea necesarului de hârtie, ajutând astfel mediul înconjurător. În ziua de azi se caută alternative “verzi”, iar această aplicație vine în sprijinul misiunii de asigurare a unui mediu natural sănătos.

Aplicația ar putea ajuta la securitatea datelor personale, informațiile confidențiale precum CNP-ul fiind preluate automat din imagine, encriptate și salvate într-o bază de date. Astfel, s-ar evita scrierea manuală a informațiilor în registre fizice care pot fi alterate într-un mod sau altul, sau scrierea datelor în plain text în baze de date, spreadsheet-uri sau fișiere text. Ar fi o măsură salutată de susținătorii trendului GDPR.

2.2. Obiectivele

Obiectivul ideal, utopic, al acestei lucrări de licență, este crearea unui punct de pornire, a unei baze, pe care să se clădească o aplicație, care să ajute în procesul de digitalizare al birocrației. Pentru a pune bazele solide ale unei astfel de aplicații, este necesară crearea unui sistem funcțional, simplu și ușor de extins ulterior. Astfel, obiectivele personale propuse pentru aplicația de față sunt următoarele:

1. Crearea unui sistem, compus dintr-o suită de programe, care să detecteze codul numeric personal de pe o imagine a actului de identitate.

Suita de programe ar urma a fi reprezentată de script-uri în python pentru localizarea și clasificarea cifrelor din cadrul codului numeric persoanel, aplicație în Java pentru automatizarea proceselor de transmitere a modelului și input-urilor la placa de dezvoltare, aplicație desktop în Delphi care să unească toate componente folosind o interfață grafică intuitivă, program în limbajul VHDL care să clasifice, pe rând, cifrele din CNP.

2. Crearea unui program care să localizeze cifrele din CNP.

Script în Python care să detecteze cifrele care compun codul numeric personal, pe baza unei imagini. Cifrele detectate vor fi afişate sub codul numeric personal, cu o altă culoare, pentru a se face distincția.

3. Crearea unui program care să preia fiecare cifră din cele 13 ale CNP-ului, și să le clasifice, folosind o rețea neuronală convoluțională.

Script în Python care prin care cifrele din imagine vor fi preluate, redimensionate și introduse într-o rețea neuronală convoluțională pentru a fi clasificate.

4. Crearea unei aplicații desktop, cu o interfață grafică ușor de utilizat, intuitivă și rapidă și un aspect comercial.

O aplicație desktop, realizată în Delphi, care să unească toate componenetele sistemului, ca un liant. Aplicația ar urma să aibă o interfață grafică ușor de utilizat (fără prea multe ferestre, posibile hint-uri la butoane, un posibil help sub forma unui manual de utilizare în cadrul aplicației), intuitivă (denumirile butoanelor și a ferestrelor să fie sugestive pentru rolul pe care îl au în cadrul aplicației), rapidă (prin încercarea de a nu implementa un design cu prea multe ferestre care se deschid secvențial), iar aspectul comercial ar urma să fie dat de componente grafice realizate folosind diferite user experience design tooluri, precum Adobe XD sau GIMP.

5. Crearea unei baze de date pentru a stoca utilizatorii și datele despre aceștia (parole encriptate), imaginile originale și cele procesate, un istoric al evenimentelor operate în aplicație.

Bază de date SQL menită să stocheze informații relevante ale aplicației. Parolele utilizatorilor ar urma să fie encriptate, folosind obiecte de encriptie/decriptie puse la dispoziție de framework-ul Delphi.

6. Crearea unei rețele neuronale artificiale pe placa FPGA, pentru a clasifica cele 13 cifre ale CNP-ului.

Rețea neuronală artificială, proiectată în VHDL, care să aibă ca input ieșirile din urma convoluțiilor realizate de rețea neuronală convoluțională din Python, iar output-ul să fie valoarea cifrei clasificate.

7. Crearea unui program care să automatizeze aplicațiile desktop, cu scopul de a transmite input-ul rețelei neuronale artificiale și modelul acestei rețele la placa de dezvoltare Basys3.

Program scris în Java, menit să transmită automat input-urile și modelul rețelei neuronale artificiale la placa FPGA, folosind protocolul UART.

2.3. Grupul țintă

Alegerea grupului țintă este o sarcină vitală pentru un dezvoltator de aplicații. Este punctul de pornire al strategiei de marketing. Definirea lui stă la baza construirii unui plan de dezvoltare al produsului. Astfel, publicul sau grupul țintă nu este oricine. Scopul unui dezvoltator este de a alege acest grup țintă pentru a se nișa pe o anumită arie, pe care să încerce ulterior să creeze un monopol sau oligopol.

În lucrarea de față, grupul țintă este clasificat în actori principali (adevăratul grup țintă, cel pentru care se face nișarea) și actori secundari (oameni care ar avea de beneficiat în urma acestei aplicații).

Actorii principali, adevăratul grup țintă, cel pentru care se face nișarea, este reprezentat de:

1. Instituții publice

Începând de la primării, prefecturi, consilii județene, instituții centrale, instituții de sănătate, ambasade, consulate, până la unități de învățământ. Toate aceste instituții au procese de preluare și prelucrare a datelor care ar putea fi eficientizate. De exemplu, timpul de așteptare al cozilor de la diferite ghișee ale primăriei ar putea fi reduse sau chiar eliminate dacă preluarea datelor ar fi făcută pe baza unor imagini ale acelora/documentelor.

2. Companii din sectorul privat

Există multe procese în cadrul firmelor private care ar putea necesita o aplicație care să preia, în timp real, diferite informații de pe acte/documente. De exemplu, pentru hypermarketurile unde este necesară înregistrarea clienților în baza de date, aplicația ar eficientiza procesul. Un alt exemplu ar fi în cadrul unui proces de angajare, persoana care urmează a fi angajată, ar trimite o poză cu actul de identitate sau alte documente relevante acestui proces, iar informațiile necesare ar fi preluate simplu și rapid.

3. Organizatori de evenimente

Evenimente majore precum Untold, Neversea, Electric Castle sunt dezvoltate din punctul de vedere al digitalizării, prin urmare nu ele ar fi grupul țintă, ci mai degrabă evenimente locale de o anvergură mai redusă. De pildă, accesul la o nuntă, sau eveniment caritabil, cu sute de invitați, ar putea fi făcut pe baza unei aplicații care să detecteze CNP-ul de pe actul de identitate, și dacă acesta corespunde cu cnp-urile din baza de date, accesul să fie permis.

Actorii secundari, grupuri sau persoane fizice care ar acea de beneficiat în urma acestei aplicații sunt:

1. Dezvoltatorii care doresc să aibă ca punct de pornire lucrarea de față

Start-up-uri care doresc să contribuie la digitalizarea birocației și care ar putea folosi idei din aplicația de față pentru a implementa un sistem complet de detecție a datelor personale din documente. De exemplu, ar putea prelua modelul rețelei neuronale convoluționale, sau design-ul aplicației desktop, modul în care sunt interconectate ferestrele și secvențialitatea acestora.

2. Studenți care doresc să dezvolte aplicații pe baza tehnologiilor folosite în lucrarea de față.

În dorința de a crea sistemul de față, a fost necesară căutarea unor resurse sau proiecte care să stea la baza diferitelor module ale aplicației. De exemplu, pentru a crea rețeauna neuronală artificială pe FPGA, a fost nevoie de căutări intense pentru a găsi un proiect a cărui idee de bază să semene cătuși de puțin cu aplicația de față. Astfel, rețeauna neuronală artificială, odată implementată, poate reprezenta o sursă de inspirație pentru studenții care doresc să dezvolte o astfel de aplicație pentru placa de dezvoltare FPGA. Un alt student

ar putea prelua modelul rețelei neuronale convolutionale, pentru a realiza un clasificator de cifre, acesta putând fi folosit, de pildă, pentru a detecta cifrele de la numărul de înmatriculare a unei mașini.

3. Oameni pasionați de programare

Oameni care în general sunt la curent cu ultimele tenduri din domeniul IT, care nu au neapărat cunoștințe de specilitate, dar care sunt dornici să fie up-to-date cu ultimele aplicații apărute pe piață.

2.4. Cerințele funcționale și cele non-funcționale ale sistemului

Cerințe funcționale:

1. Detectia cifrelor.

Localizarea și clasificarea cifrelor din codul numeric personal de pe imaginea cu actul de identitate. De asemenea, scrierea codului numeric personal detectat, sub cifrele din imaginea originală, cu o culoare diferită, formând o aşa-numita imagine procesată.

2. Salvarea informațiilor în baza de date.

Datele privind utilizatorii (nume, prenume, username, parola, imagine de contact), imaginile salvate (denumire, cnp, extensie, tip, data salvării) și istoricul evenimentelor (acțiunea întreprinsă, denumirile fișierelor, data la care a avut loc evenimentul, user-ul care a întreprins acțiunea).

3. Sign up.

Pentru a deveni utilizator al aplicației, este necesară înscrierea, folosind numele, prenumele, username-ul, parola și o imagine de contact. Parola este encriptată și salvată în baza de date, alături de celelalte informații de logare.

4. Sign in.

Se va face doar dacă utilizatorul s-a înscris în prealabil. Pentru a se loga, user-ul trebuie să introducă username-ul și parola. Parola introdusă va fi encriptată și comparată cu cea salvată în baza de date. Dacă cele două parole corespund, utilizatorul va fi redirecționat spre pagina principală a aplicației, altfel va apărea un mesaj de avertizare în ceea ce privește corectitudinea username-ului sau parolei introduse.

5. Încărcare imagine.

Imaginea cu actul de identitate, aşa-numita imagine originală, va fi încărcată din sistemul de fișiere. Se va permite comutarea folderelor pentru găsirea imaginii dezirabile, și alegerea ei. Imaginea și denumirea fișierului vor fi afișate în aplicație.

6. Trimitere imagine.

Imaginea originală va fi procesată pentru a se detecta cnp-ul. Imaginea procesată, cu cnp-ul detectat, va fi afișată în aplicație, alături de imaginea originală. De asemenea, inputurile și modelul rețelei neuronale artificiale vor fi trimise la placa de dezvoltare, care va afișa rezultatul pe 7 segment display.

7. Salvare imagine.

Imaginile originale și procesate vor putea fi salvate în baza de date, și ulterior afișate din librărie (funcționalitate descrisă mai jos).

8. Afișare rezultate.

Rezultatele detecției cifrelor vor fi afișate în aplicație. Se dorește afișarea rezultatelor așteptate (cnp-ul propriu-zis, cel real) și a rezultatelor reale în urma clasificării cifrelor. Este de dorit ca cele două tipuri de rezultate să coincidă.

9. Afișare istoric.

Istoricul evenimentelor întreprinse de un utilizator al aplicației este redat de funcționalitatea intitulată istoric. Se vor afișa acțiunea care a avut loc (exemplu ‘Choose image’/‘Send image’/‘Save images’/‘Remove images’), denumirea fișierului original, al fișierului editat și data întreprinderii acțiunii.

10. Afișare librărie

Librăria conține atât imaginile originale, neprocesate, brute cu actele de identitate, cât și imaginile editate, procesate, care au cnp-ul detectat și scris sub cnp-ul real. Utilizatorul are opțiunea de a alege fișierul dorit, și de-al deschide într-o fereastră nouă, care va permite redimensionarea imaginii, oferind claritate.

Cerințe non-funcționale:

1. Adaptabilitate

Se referă la capacitatea unui sistem de a se adapta eficient și rapid la schimbări.[26] Adaptabilitatea acestui program este redată de faptul că pozele nu trebuie să respecte un anumit şablon, sau să fie cele mai clare. Rețea neuronală este antrenată folosind un dataset îndeajuns de mare și de reliable încât să recunoască și cifrele sterse.

2. Disponibilitate

Se referă la nivelul la care un sistem este pregătit la începutul unei misiuni, știind că misiunea va începe la o dată nedeterminată în viitor.[27] Modul în care se scriu cifrele romane nu se va schimba prea curând, astfel că rețea neuronală va continua să clasifice cifrele. Totuși, există posibilitatea ca actul de identitate să își schimbe formatul, caz în care va fi necesară modalitatea de a detecta cifrele. Cel mai probabil, totuși, varianta cu codul numeric personal se va păstra.

3. Integritatea datelor

Se referă la siguranța acurateții și consistenții datelor pe durata întregii existențe a sistemului.[28] Datele vor putea fi stocate într-o bază de date, aşadar există toate şansele ca datele să fie integre, spre deosebire de varianta în care datele personale sunt preluate și scrise pe foi de hârtie sau spreadsheet-uri salvate local.

4. Durabilitate

Reprezintă abilitatea unui sistem de a rămâne funcțional, fără a necesita reparații excesive pe durata sa de viață. [29] Aplicația va necesita un proces de menenanță, însă erorile care

ar putea să apară, nu ar trebui să fie majore sau să necesite un timp îndelungat pentru a fi remediate.

5. Integrarea sistemelor

Este capacitatea de a aduce laolaltă mai multe sub-sisteme, formând un sistem. [30] Sistemul de față este compus dintr-o suită de programe și este deschis spre a fi integrat cu alte sisteme, formând un sistem mai mare. De asemenea, există o deschidere și spre a adăuga alte sub-sisteme la sistemul de față, creând un sistem mai complex.

6. Interoperabilitate

Capacitatea unui sistem de a lucra cu alte sisteme, în prezent sau viitor.[31] Sistemul este deschis spre a fi integrat cu alte sisteme, formând un sistem mai mare. Acest lucru s-a întâmplat de-a lungul evoluției aplicației. Inițial, aplicația era formată doar din rețea neuronală artificială de pe placa FPGA, având ca scop clasificarea cifrelor. Cu timpul, aplicația s-a dezvoltat spre localizare cifrelor din CNP și clasificarea lor.

7. Mantenabilitate

Se referă la ușurința cu care un produs poate fi întreținut pentru a corecta defectele sau cauza apariției lor.[32] Sistemul nu ar trebui să prezinte erori majore, care să necesite timp îndelungat pentru reparații. Totuși, există anumite cazuri care ar necesita modificări substanțiale ale aplicației, precum modificarea formatului actului de identitate. În rest, cele mai probabile erori vor avea legătură cu baza de date sau use-casuri care nu au fost prevăzute.

8. Operabilitate

Este proprietatea unui sistem care îl face să lucreze bine în producție.[33] Aplicația nu prezintă crash-uri majore, nu necesită restartare frecventă.

9. Performanță

Se referă la acuratețea, eficiența și viteza execuției unui program.[34] Aplicația a fost optimizată în aşa fel încât timpul de așteptare dintre momentul în care o imagine este selectată spre a fi procesată, până în momentul finalizării procesării să fie cât mai redus. De asemenea, s-a încercat o eficientizare a aplicației de automatizare, păstrând însă anumite limite. De exemplu, pentru a apăsa automat pe anumite butoane ake unor aplicații, a fost nevoie de inserarea unor tempi de așteptare de două sau trei secunde, pentru a fi sigur că celelalte acțiuni sau sarcini au fost finalizate până în momentul respectiv.

10. Portabilitate

Proprietatea ca un program să fie utilizat în diferite medii.[35] Teoretic, aplicația, sub forma unui executabil, ar putea îndeplini această cerință.

11. Fiabilitate

Se referă la abilitatea unui sistem de a funcționa fără a da greș în timpul execuției.[36] Aplicația nu prezintă prea multe bug-uri care să determine oprirea din execuție a acesteia.

De asemenea, procesul de transmitere automată a fișierelor către placa de dezvoltare a fost conceput în aşa fel încât să se țină cont de eventualele întârzieri datorate de suprasolicitarea sistemului.

12. Securitate

Protecția unui sistem împotriva surgerii de informații, furt sau compromiterea componentelor hardware sau software.[40] Informații relevante, care necesită protejare, precum parolele, sunt encriptate înapoi de a fi introduse în baza de date. În momentul în care utilizatorul dorește să se logheze, acesta introduce parola. Parola este automat encriptată iar valoarea de după encriptie este comparată cu cea din baza de date. Dacă cele două valori corespund, utilizatorul va fi redirecționat către pagina principală a aplicației.

13. Scalabilitate

Proprietatea unui sistem de a gestiona o creștere a muncii, prin adăugarea unor resurse.[37] Baza de date a aplicației este cea mai predispusă să se încarce, mai ales că există tabela de evenimente care salvează fiecare acțiune a fiecărui utilizator. Relațiile dintre tabele au fost concepute în aşa fel încât să nu fie duplicate sau prea multe câmpuri goale, iar în ceea ce privește aplicația desktop, se vor prelua un număr limitat de informații din baza de date, pentru a evita blocarea programului. De exemplu, pentru funcționalitatea de afișare a istoricului evenimentelor, vor fi preluate doar ultimele 50 de evenimente. Dacă ar fi preluate toate (de pildă, 10.000 de evenimente), aplicația s-ar bloca. Un alt exemplu sugestiv este dat de filtrările care se pot face pe tabele. Delphi are opțiunea FilterSQL care va face filtrările la nivelul bazei de date, fără a fi necesar ca informațiile să fie aduse din baza de date până în aplicația desktop, și abia ulterior să se facă filtrarea.

14. Compatibilitate

Este capacitatea ca două sisteme să funcționeze împreună, fără a fi necesară alterarea sau modificarea lor pentru a performa. [38] Aplicația trebuie obligatoriu instalată pe computer. Nu a fost dezvoltată o aplicație pentru mobil. De asemenea, este necesară instalarea unor programe speciale pentru ca aplicația să funcționeze. De exemplu, aplicația HTerm pentru transmiterea fișierelor la placa de dezvoltare.

15. Uzabilitate

Capacitatea unui sistem de a oferi utilizatorilor șansa de a efectua sarcinile în siguranță, efectiv și eficient, bucurându-se în același timp de experiență. [39] Aplicația ar urma să aibă o interfață grafică ușor de utilizat (fără prea multe ferestre, posibile hint-uri la butoane, un posibil help sub forma unui manual de utilizare în cadrul aplicației), intuitivă (denumirile butoanelor și a ferestrelor să fie sugestive pentru rolul pe care îl au în cadrul aplicației), rapidă (prin încercarea de a nu implementa un design cu prea multe ferestre care se deschid secvențial), iar aspectul comercial ar urma să fie dat de componente grafice realizate folosind diferite user experience design tool-uri, precum Adobe XD sau GIMP.

Capitolul 3. Studiu Bibliografic

3.1. Machine learning pe FPGA

Aplicația din lucrarea de față are ca scop localizarea și clasificarea cifrelor din codul numeric personal, folosind imagini. Clasificarea cifrelor se va face folosind placa de dezvoltare Basys3. Este necesară proiectarea și implementarea unei rețele neuronale artificiale, care va prelua o imagine cu cifra, de 28X28, ca și input, iar output-ul rețelei va fi reprezentat de posibilele valori, de la 0 la 9, pe care le-ar putea avea cifra procesată. Valoarea cea mai mare din cele 10 output-uri ale rețelei va fi, practic, valoarea cifrei clasificate.

Aplicația de recunoaștere a semnelor de circulație, implementată de Prof. Dr. Marco Winzker, de la Universitatea Bonn-Rhein-Sieg din Germania, folosește machine learning în FPGA, pentru a clasifica semnele de circulație de pe drumurile din țara natală. Această aplicație reprezintă un punct de pornire foarte bun pentru sub-sistemul reprezentat de clasificatorul de cifre din FPGA. Limbajul folosit în acest program este VHDL, opțiune care coincide cu alegerea pentru lucrarea de față. O altă variată, a cărei implementare va fi descrisă într-un alt subcapitol, este folosind limbajul Verilog.

3.1.1. Antrenarea reței neuronale

Programul de recunoaștere a semnelor de circulație, dezvoltat de Prof. Dr. Marco Winzker, folosește învățarea supervizată, motiv pentru care este nevoie de date și de label-uri. Aplicația are ca scop localizarea și clasificarea semnelor de circulație din Germania. Semnele de circulație cu fundal albastru, din Figura 3.1 (a) [8], denotă un drum primar (de exemplu, o autostradă), iar cele cu fundal galben, din Figura 3.1 (b) [8], un drum secundar.



Figura 3.1 - Drum prioritar din Germania



Figura 3.2 - Drum secundar din Germania

În primă fază, se vor detecta doar semnele de circulație cu fundal galben. Un prim pas este determinarea label-urilor. Astfel, se va crea o imagine label, în care semnul de circulație este marcat cu alb, iar background-ul imaginii, tot ceea ce nu reprezintă semn de circulație (de pildă, drum, mașini, copaci) este marcat cu negru.



Figura 3.3 - Imagine label

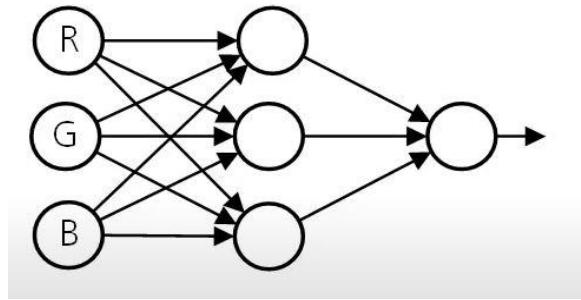


Figura 3.4 - Structura rețelei neuronale

Imaginea din Figura 3.3 [8] este imaginea label, creeată pe baza imaginii din Figura 3.2, cu semnul de circulație pe fundal galben. Semnul de circulație a fost tăiat și a fost aplicat un fundal alb în locul în care a fost semnul de circulație de culoare galbenă. Celelalte elemente ale imaginii originale au fost eliminate, prin setarea unui fundal negru în locul acestora. Cele două imagini, imaginea originală din Figura 3.2, și imaginea label din Figura 3.3, stau la baza antrenării rețelei neuronale.

Structura rețelei neuronale este una simplă, alcătuită dintr-un input layer, un hidden layer și un output layer. Input-ul are 3 noduri, câte unul pentru fiecare canal: roșu, verde și albastru. Toți pixelii din imagine vor constitui, pe rând, intrări ale rețelei neuronale. Hidden layer-ul are 3 noduri. Output layer-ul are un nod, care comunică dacă a fost sau nu detectată culoarea galbenă. Structura rețelei neuronale este redată în imaginea de mai jos din Figura 3.4 [8].

Modelul rețelei neuronale, parametrii reprezentați de weights-uri și bias-uri, vor fi determinate folosind un Octave script (matlab). Pentru antrenare se vor folosi imaginea originală cu semnul de circulație de culoarea galbenă din Figura 3.2, și imaginea label din Figura 3.3. Vor fi 400 de epoci, cu un learning rate de 0.00001. Se va crea o matrice labels care se populează cu 0.01, în loc de 0, pentru că sigmoid nu va ajunge să fie niciodată 0. De asemenea, în cadrul scriptului, se va defini structura rețelei: 3 noduri de intrare, 3 noduri în hidden layer, 1 nod de ieșire. Pentru antrenarea rețelei se folosește backpropagation. Imaginea de training este reprezentată în proporție de 16.73% de culoarea galbenă, iar 84.27% de background.

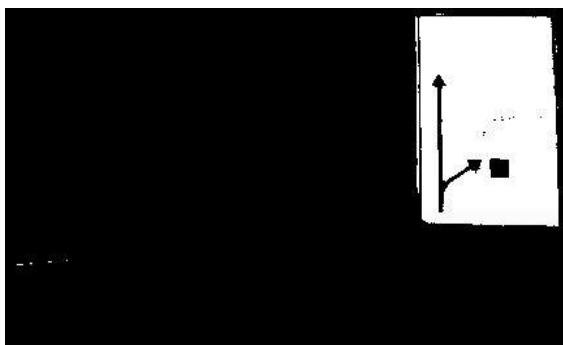


Figura 3.5 - Rezultatul antrenării

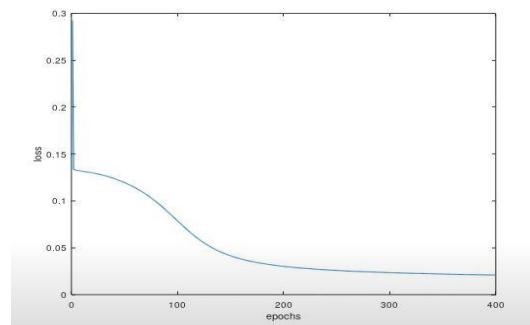


Figura 3.6 - Evoluție loss

Rezultatul antrenării, din Figura 3.5 [8], este o imagine cu aceleasi dimensiuni cu imaginea originală, în care fundalul cu galben al semnului de circulație este marcat cu alb, iar restul imaginii, cu negru. Chiar dacă semnul de circulație a fost detectat, se poate observa că au

fost detectate și alte porțiuni ale imaginii. De asemenea, a fost afișat un grafic care prezintă modul în care loss-ul a scăzut, după backpropagation, pe durata celor 400 de epoci, conform Figurii 3.6 [8]. Rețeaua neuronală ar putea fi îmbunătățită dacă s-ar folosi mai multe imagini pentru antrenare.

Scopul principal al antrenării a fost obținerea parametrilor rețelei neuronale. Se pot observa weights-urile și bias-urile pentru hidden layer, respectiv pentru output layer. Fiecare rând corespunde câte unui canal R, G și B, iar ultima coloană este bias-ul. Parametrii sunt salvați în floating point, dar vor fi convertiți pentru a putea fi utilizati în implementarea hardware. [8]

Diagrama bloc a rețelei neuronale, conform Figurii 3.7 [9], se bazează pe structura rețelei neuronale. Cele 3 semnale de intrare (R, G, B), 3 submodule pentru cei 3 neuroni ai hidden layer, 1 submodul pentru output layer. Neuronii vor primi parametrii rezultați în urma antrenării: 3 weights-uri și 1 bias pe neuron. Procesarea output-ului constă în următorul algoritm: dacă valoarea output-ului este mai mare sau egală cu 0.5, atunci pixelul de output este alb, altfel este negru.

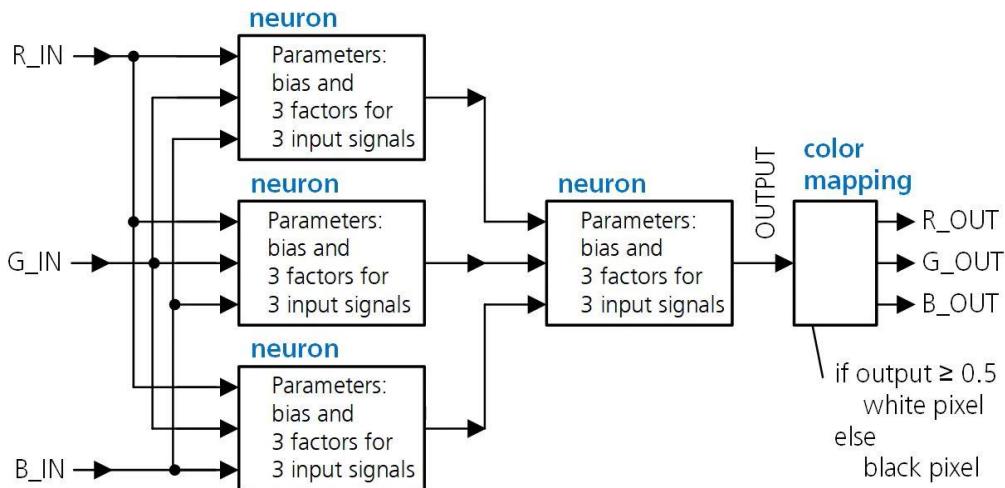


Figura 3.7 - Diagrama bloc a rețelei neuronale
Figura 3.7 - Diagrama bloc a rețelei neuronale

Implementarea unui neuron se face conform Figurii 3.8 [9]. Input-urile sunt x_1, x_2, x_3 , pe 8 biți (0-255) și parametrii generici w_1, w_2, w_3 pentru multiplicare. Se calculează suma produselor și se adaugă la final bias-ul. Valoarea va fi reprezentată pe 14 biți și va fi introdusă într-o memorie ROM care va avea ca output, pe 8 biți, valoarea sigmoid-ului.

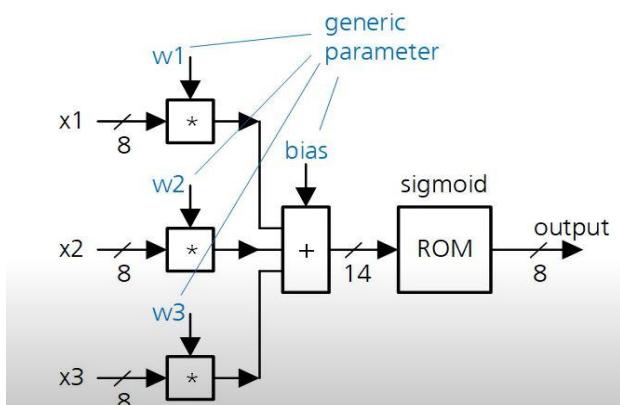


Figura 3.8 - Implementarea unui neuron

Avantajul implementării hardware este abilitatea de a obține o putere computațională înaltă. Calculele sunt făcute în hardware în paralel. De aceea, implementarea are nevoie de următorul efort de procesare: pentru fiecare ciclu de ceas, vor fi necesare 12 înmulțiri, 12 sume, 4 funcții sigmoid și maparea culorilor. Se folosesc strategii pipeline, iar numărul lor poate fi modificat, optimizat, pentru a se ajunge la frecvențe mai mari. Design-ul este implementat în fixed point values. Așadar, este nevoie de o conversie a numerelor (parametrilor) din floating point în fixed point. Conversia din floating point în fixed point se face conform următoarei strategii: weights-urile w_1, w_2, w_3 sunt shiftate la stânga cu 5 poziții, iar bias-ul cu 13 poziții la stânga. În Figura 3.9 [9] sunt reprezentate weights-urile în floating point pentru hidden layer și output layer, respectiv weights-urile în fixed point value pentru hidden layer și output layer. Pentru a înțelege cum au fost convertite valorile din floating point în fixed point, se va lua ca exemplu prima valoare din matricea de weights-uri pentru hidden layer. Astfel, valoarea 0.90314 a fost convertită în fixed point folosind o shiftare la stânga cu 5 poziții. Cu alte cuvinte, valoarea 0.90314 a fost înmulțită cu valoarea zecimală 32, iar rezultatul este 29. Pentru bias, valoarea 0.34728 a fost shiftată la stânga cu 13 poziții. Astfel, înmulțind 0,34728 cu valoarea zecimală 8192, corespunzătoare shiftării la stânga cu 13 poziții, rezultatul va fi 2845. [9]

```

Weight Matrix from the Input to the Hidden Layer
 0.90314   -1.41457   -2.71251   -2.22496
 -11.29638    3.94491   11.58380    0.34728
  -9.77221    2.99860   10.54126    0.55096
Weight Matrix from the Hidden to the Output Layer
  1.5896   -4.9519   -4.0163    5.0977

Finished Script
>> NN_convert_fixed_point
Starting Script

Fixed Point Matrix for Hidden Layer
  29     -45     -87   -18227
  -361     126     371    2845
  -313      96     337    4513
Fixed Point Matrix for Output Layer
   51     -158    -129   41760

```

Figura 3.9 - Conversia din floating point în fixed point

Reprezentarea numerelor în fixed point, propusă în această lucrare, modalitatea în care se face conversia din floating point și ideea implementării funcției sigmoid sub forma unei memorii ROM, reprezintă posibile puncte de reper pentru lucrarea de cercetare de față.

3.2. Rețele neuronale pe FPGA

3.2.1. Antrenarea rețelei neuronale

În cazul folosirii rețelelor neuronale pe FPGA, de cele mai multe ori, antrenarea se va face în prealabil. Există mai multe motive care stau în spatele acestei decizii. În primul rând, antrenarea necesită multe resurse hardware, iar algoritmi, în special backpropagation, nu sunt hardware-friendly. În al doilea rând, antrenarea se face o singură dată. Cu modelul obținut se vor putea face ulterior numeroase clasificări. Astfel, având în vedere că nu este nevoie de mai mult de o antrenare, antrenarea pe FPGA s-ar considera o pierdere inutilă. Nu în ultimul rând, antrenarea rețelei nu este time critical. Ea ar putea dura una-două zile pe calculator. Important este ca rețeaua neuronală preantrenată să clasifice imaginile în timp real. Având în vedere aceste

motive, este de preferat ca antrenarea rețelei neuronale să fie făcută în software, iar rețeaua neuronală preantrenată să fie folosită în implementarea hardware.

3.2.2. Reprezentarea numerelor

În general, input-ul rețelei neuronale este reprezentat de valori pozitive, valori ale pixelilor. Weights-urile și bias-urile, în schimb, pot fi numere pozitive sau negative. Mai mult decât atât, ele au și o parte frațională. Astfel, reprezentarea lor se rezumă la două modalități: IEEE floating point (32 biți precizie simplă sau 64 biți precizie dublă) sau fixed point representation. Floating point ajută în reprezentarea numerelor mari și oferă o precizie mai bună, dar implementarea și manipularea sunt dificile. Aici se adaugă și un consum mare de resurse. Din aceste motive, se va folosi reprezentarea în fixed point. Valorile de input folosite în rețele neuronale sunt, în general, normalizate (între 0 și 1 sau -1 și 1), astfel că reprezentarea numerelor mari nu va constitui o problemă. Dezavantajul la reprezentarea în fixed point este acuratețea, dar cât timp nu va apărea overflow varianta cu fixed point va avea o performanță mai bună decât reprezentarea în floating point.

Folosind reprezentarea în fixed point, este necesară specificarea numărului total de biți pe care îl va avea un număr, numărul de biți reprezentând partea întreagă și numărul de biți reprezentând partea frațională. De exemplu, un număr căruia i s-au alocat 15 biți, ar putea avea următoarea formă: 5 biți reprezintă partea întreagă, 10 biți reprezintă partea frațională.

În Figura 3.10 [10], este reprezentat numărul 15.84. Partea întreagă este 15, iar partea frațională este 0.84. Numărul 15 este reprezentat în binar ca fiind 01111. Partea frațională se calculează ca 2 la puterea minus indexul poziției. Astfel, primele 2 cifre din partea frațională, adică 11, reprezintă 0.5 plus 0.25, adică 0.75. Dacă a treia cifră din partea frațională ar fi fost 1, s-ar fi adăugat 0.12 la 0.75 și ar fi dat 0.87, care ar fi depășit 0.84.

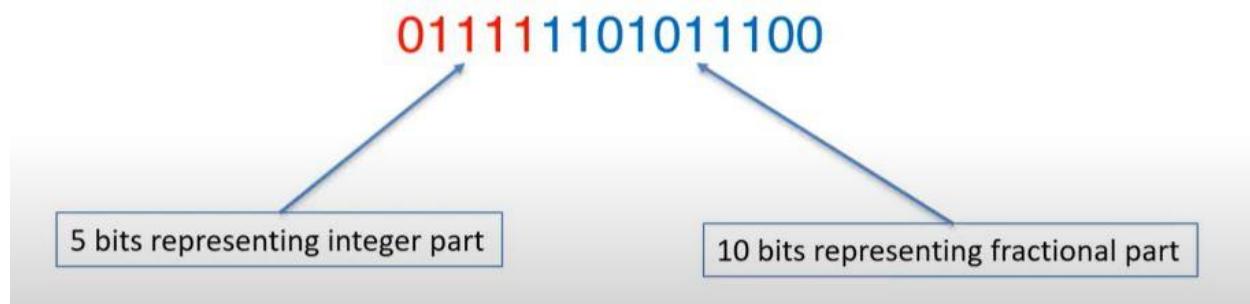


Figura 3.10 - Reprezentarea numerelor în fixed point

În cazul de față, partea frațională este 0.83984375, nu 0.84, astfel este introdusă o eroare de 0.00015625. Dacă eroarea se acumulează, performanța va fi afectată. Dacă în loc de reprezentarea pe 15 biți se optează pe o reprezentare pe 10 biți, unde 5 biți constituie partea întreagă iar 5 biți partea frațională, valoarea 15.84 va fi reprezentată ca și 15.8125 în reprezentarea în fixed point. În acest caz, eroarea este 0.0275, mult mai mare față de cea anterioară. Astfel, cu cât se alocă mai mulți biți pentru partea frațională, cu atât precizia va fi mai bună. Prin stabilirea numărului de biți pe care se vor reprezenta valorile, se va face un tradeoff între acuratețe și utilizarea resurselor.

Weights-urile și bias-urile pot fi pozitive sau negative. Astfel, se poate alege între 2 reprezentări, ori reprezentarea signed-magnitude, ori reprezentarea în complement față de 2. În reprezentarea în signed-magnitude, cel mai semnificativ bit (MSB) denotă semnul: 0 dacă este pozitiv, 1 dacă este negativ.

01111101011100 represents +15.83984375
11111101011100 represents -15.83984375

Figura 3.11 - Reprezentarea numerelor în signed-magnitude

În Figura 3.11 [10], primul bit, cel cu verde, denotă semnul numărului, restul bițiilor rămânând neschimbați. Reprezentarea numerelor în signed-magnitude este ușor de decodat pentru oameni. Dezavantajul este că există două reprezentări pentru 0 (+0 și -0). De asemenea, adunând două numere cu semn, nu va garanta că rezultatul va fi tot un număr cu semn. Din aceste motive, a fost introdusă reprezentarea numerelor în complement față de 2. La fel ca la signed-magnitude, în reprezentarea în complement față de 2, MSB denotă semnul numărului (0 pentru pozitive, 1 pentru negative).

01111101011100 represents +15.83984375
100000010100100 represents -15.83984375

Figura 3.12 - Reprezentarea numerelor în complement față de 2

Pentru a afla complementul față de 2 al unui număr, se va calcula complementul față de 1 al numărului pozitiv și se va adăuga 1. Reprezentarea în complement față de 2, conform Figurii 3.12 [10], este eficientă pentru operația de adunare (scădere), deoarece rezultatul va fi reprezentat tot în complement față de 2. Pentru operația de înmulțire, signed-magnitude ar putea fi mai eficientă. [10]

3.2.3. Funcția de activare

Multe rețele neuronale folosesc sigmoid sau hyperbolic tangent ca și funcții de activare. În construirea circuitelor digitale, aceste funcții vor folosi intensiv resursele, plus că ar fi greu de implementat. De aceea, valorile lor se precalculează (din moment ce se cunoaște range-ul input-ului x) și se vor stoca într-o memorie ROM. Aceste memorii ROM se vor numi LUT (Look Up Table). A nu se confunda aceste LUT-uri cu building block-ul LUT din FPGA. Aceste ROM LUT-uri se vor construi fie folosind BRAM (Block RAM), fie RAM distribuit (FFs and LUTs). [12]

3.2.4. Folosirea IP Cores

În general nu se folosesc Xilinx IP Cores în crearea rețelelor neuronale. Principalul motiv este că acestea nu sunt prea flexibile. De exemplu, dacă se folosesc IP Cores pentru a crea un ROM pentru activation function, și dacă se dorește modificarea mărimi ROM-ului, acest lucru nu se va putea face schimbând valoarea unui parametru. Cu toate acestea, IP Cores sunt eficiente

în implementare. Astfel, BRAM-urile sunt mult mai eficiente decât RAM-ul distribuit în descrierea ROM-urilor. [10]

3.2.5. Proiectarea neuronului

Arhitectura neuronului folosit în rețeaua neuronală este reprezentată în Figura 3.13 [11]. Weight memory va stoca weights-urile. Mărimea memoriei depinde de numărul de input-uri pe care le va primi neuronul. Fiecare locație din weight memory va fi o valoarea a unui weight. Weight memory poate fi implementată atât ca un ROM, cât și ca un RAM. Având în vedere că se lucrează cu valorile predefinite (antrenamentul a fost deja făcut), se poate folosi ROM. Fiecare weight va fi înmulțit cu fiecare input. Acest lucru se va realiza cu ajutorul ‘Interface and Control Logic’. Operația de acumulare, folosind un sumator, va aduna rezultatul înmulțirii dintre input și weight, cu valoarea acumulată până în momentul respectiv. La finalul adunării produselor dintre input și weight, se va adăuga bias-ul. Rezultatul final va reprezenta input-ul memoriei ROM pentru funcția de activare, unde vor fi stocate valorile pre-calulate pentru sigmoid, iar ieșirea acestei memorii coincide cu output-ul neuronului. [11]

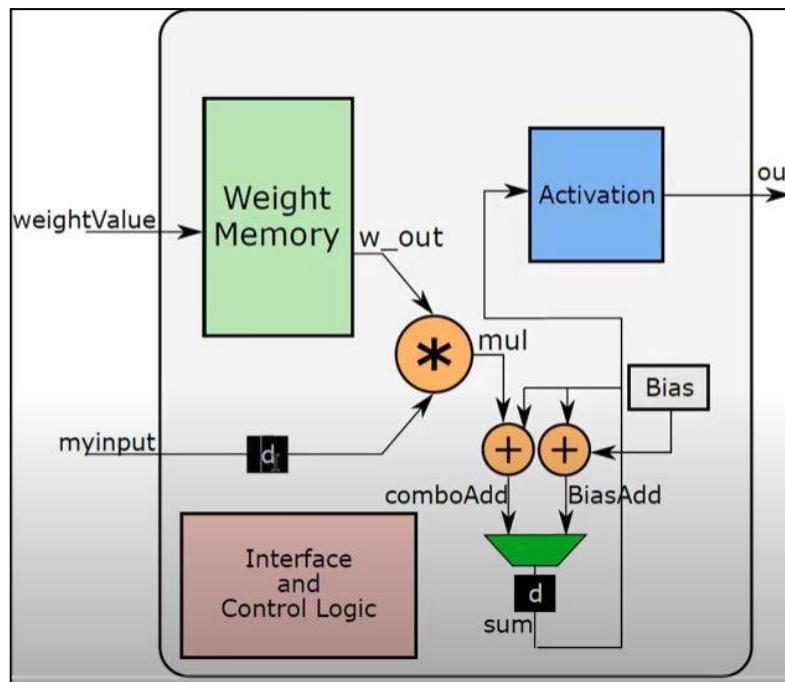


Figura 3.13 - Proiectarea neuronului [11]

3.2.6. Proiectarea layer-urilor

Rețeaua conține un input layer cu 784 neuroni, 3 hidden layer, două cu 30 neuroni și unul cu 10 neuroni, și un output layer cu 10 neuroni. Cele 784 de input-uri nu vor fi reprezentate ca și neuroni. Neuronii vor fi implementați începând de la primul hidden layer, până la output layer (inclusiv). În implementarea aleasă de V.K., fiecare layer este instanțiat separat. De asemenea, fiecare neuron al fiecărui layer este instanțiat separat. În mod normal, acest lucru ar putea fi făcut folosind o buclă de tip for generate. Partea dificilă este dată de faptul că weights-

urile sunt stocate în fișiere. Pentru fiecare fișier, trebuie introdus numele acestuia, de exemplu "w_1_0.mif" pentru a prelua datele. Verilog nu permite generarea string-urilor de forma menționată mai sus. [13]

3.3. Recunoașterea cifrelor scrise de mâna folosind rețele neuronale artificiale

Clasificarea cifrelor scrise de mâna este o sarcină complexă, implementată recent în tot mai multe aplicații. Este folosită de cercetători sau ingineri într-o varietate de aplicații, una dintre ele fiind citirea cifrelor scrise de mâna pe CEC-uri, folosită de aplicații bancare. Lucrarea de față prezintă implementarea unei rețele neuronale cu mai multe straturi complet conectate, pentru clasificarea cifrelor scrise de mâna. Antrenarea și testarea au fost făcute folosind baza de date publică pentru cifre scrise de mâna de la MNIST. Au fost folosite 28.000 de imagini pentru antrenare, respectiv 14.000 de imagini pentru testare. Acuratețea rețelei neuronale este de 99.60% pe dataset-ul de test.

3.3.1. Introducere

Recunoașterea cifrelor scrise de mâna este un subiect important în viața cotidiană, prin aplicațiile practice pe care le are. În ultimii ani, sisteme de recunoaștere a cifrelor au fost introduse în numeroase aplicații. Inițial, recunoașterea cifrelor scrise de mâna a fost folosită pentru coduri poștale. Ulterior, clasificarea a fost utilizată în aplicații bancare pentru recunoașterea CEC-urilor. Omul a fost principalul actor în recunoașterea obiectelor precum cifre, caractere, fețe, voci. Cu toate acestea, implementarea unui sistem computerizat de recunoaștere a acestor obiecte este o sarcină dificilă.

Lucrarea de față prezintă implementarea unei rețele neuronale artificiale și antrenarea ei, pentru a recunoaște cifre scrise de mâna, de la 0 la 9. Fiind vorba de o rețea neuronală complet conectată, fiecare nod este calculat în corelație cu alte noduri și weights-uri, care sunt ajustate, calibrate, în procesul de antrenare. Valoarea fiecărui nod este calculată pe baza valorilor din nodurile precedente, proces numit forward propagation. Ieșirea rețelei neuronale este comparată cu ieșirea așteptată, iar weights-urile sunt calibrate pentru a minimiza pierderea, într-un proces numit backpropagation. Motivul pentru care rețelele neuronale au mai multe layere, este pentru a contribui la o acuratețe mai ridicată. Într-o rețea neuronală complet conectată, nodurile din fiecare strat sunt conectate cu toate nodurile din straturile precedente, respectiv următoare.

3.3.2. Descrierea dataset-ului

Rețeaua neuronală artificială propusă folosește dataset-ul public de la MNIST, care conține zeci de mii de imagini cu cifre scrise de mâna (exemple de imagini din dataset conform Figurii 3.14 [17]). Au fost extrase 42.000 de imagini, 28.000 au fost folosite pentru antrenare, respectiv 14.000 pentru testare. Imaginele au o dimensiune de 28x28 pixeli. De asemenea, fiecare imagine grayscale conține o singură cifră. Este folosită o reprezentare unidimensională a imaginii. Astfel, fiecare imagine bidimensională, de 28x28 pixeli, a fost convertită într-o imagine unidimensională cu 784 de valori, fiecare valoare reprezentând câte un pixel din imagine.

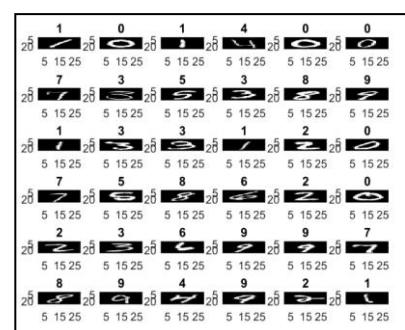


Figura 3.14 – Imagini dataset

3.3.3. Inițializarea clasificatorului

Rețeaua neuronală artificială conține 10 clase, câte o clasă pentru fiecare tip de cifră. Rețeaua este concepută să se antreneze pe baza dataset-ului de antrenare și să evalueze performanțam pe baza dataset-ului de testare. Rețeaua neuronală este implementată în Matlab și conține 3 layere, conform Figurii 3.15 [17]. Layerul de intrare este alcătuit din cei 784 de neuroni, fiecare reprezentând câte un pixel din imagine. Astfel, vectorul de intrare conține 28.000 x 784 de valori, pentru antrenarea rețelei. A fost folosit un singur hidden layer cu 100 neuroni, oferind cea mai bună performanță pentru aplicația de față. Output layer-ul are 10 neuroni, deoarece sistemul propus are 10 clase.

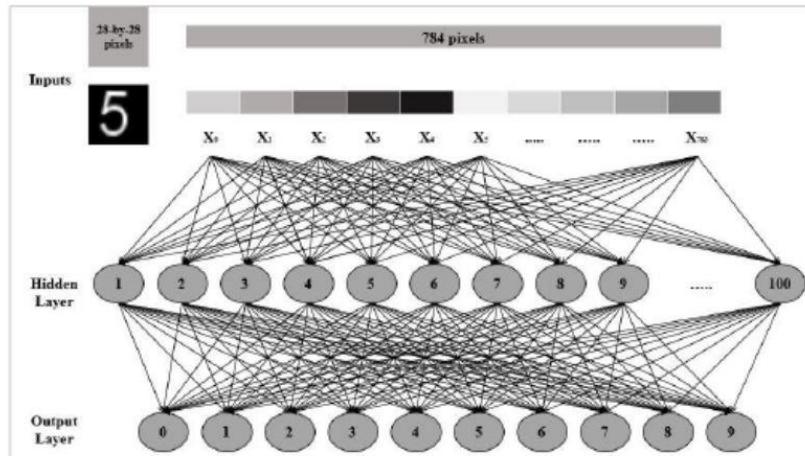


Figura 3.15 – Arhitectura rețelei neuronale artificiale

Dataset-ul a fost divizat în mai multe categorii, pentru antrenare, validare și testare. Scopul acestei împărțiri a fost evitarea overifitting-ului. Astfel, cele 28.000 de imagini au fost distribuite aleator, 90% pentru antrenare (25.200), 5% pentru validare (1.400) și 5% pentru testare (1.400). În timpul procesului de antrenare, modelul este adaptat în raport cu eroarea rețelei. Imaginele pentru validare sunt utilizate pentru a opri antrenarea atunci când observațiile măsurate pe rețeaua neuronală artificială nu se îmbunătățesc. Imaginele de test nu au nicio influență asupra procesului de antrenare. Output-ul este o matrice cu o combinație de 28.000 ori 10, precum este prezentat în Figura 3.16 [17]. Din figură, cifrei 0 îi corespunde formatul 1000000000, în timp ce lui 9 îi corespunde 0000000001.

Rows	Handwritten digits									
	0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1

Figura 3.16 – Output-ul rețelei neuronale

Figura 3.17 [17] afișează performanța antrenării. La începutul procesului de antrenare, eroarea era maximă. Pe măsură ce numărul de epoci crește, se incrementează, eroarea scade din ce în ce mai mult. La iterată 107 a epocilor, rețeaua a oferit cea mai bună performanță de validare, eroarea fiind de 0.01331.

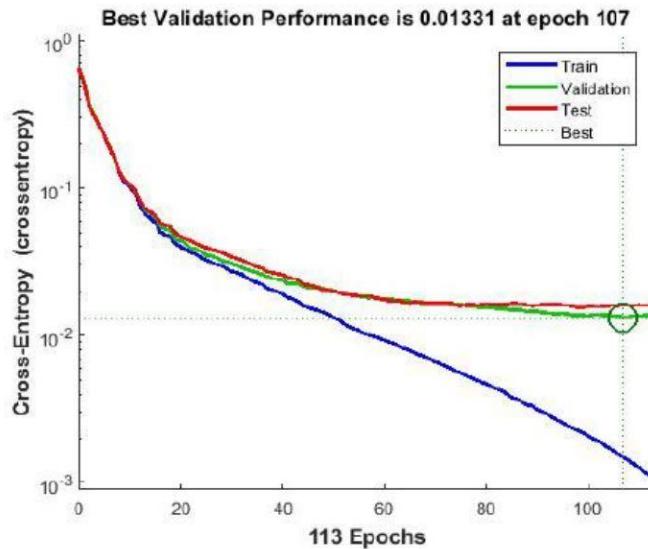


Figura 3.17 – Performanța antrenării

3.3.4. Testarea rețelei neuronale

Au fost alocate 14.000 de imagini pentru testare. Performanța testării este relevată de confuzion matrix. În matricea din Figura 3.18 [17], cuburile roșii reprezintă clasificările incorecte, cuburile verzi pe cele corecte. Cubul albastru din colțul din dreapta jos arată media procentajelor clasificărilor. Per total 99.60% din predicții sunt corecte, iar 0.40% sunt greșite în experimentul de față.

Confusion Matrix											
Output Class	1	2	3	4	5	6	7	8	9	10	
	1657 11.8%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	2 0.0%	1 0.0%	0 0.0%	0 0.0%	99.8% 0.2%
	0 0.0%	1400 10.0%	3 0.0%	0 0.0%	1 0.0%	1 0.0%	1 0.0%	1 0.0%	0 0.0%	0 0.0%	99.5% 0.5%
	0 0.0%	0 0.0%	1464 10.5%	0 0.0%	1 0.0%	1 0.0%	0 0.0%	4 0.0%	2 0.0%	0 0.0%	99.5% 0.5%
	1 0.0%	2 0.0%	0 0.0%	1290 9.2%	0 0.0%	0 0.0%	1 0.0%	1 0.0%	2 0.0%	0 0.0%	99.5% 0.5%
	0 0.0%	1 0.0%	6 0.0%	0 0.0%	1246 8.9%	1 0.0%	0 0.0%	2 0.0%	0 0.0%	1 0.0%	99.1% 0.9%
	0 0.0%	2 0.0%	0 0.0%	0 0.0%	3 0.0%	1385 9.9%	1 0.0%	1 0.0%	0 0.0%	1 0.0%	99.4% 0.6%
	0 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	1452 10.4%	1 0.0%	1 0.0%	0 0.0%	99.7% 0.3%
	0 0.0%	1 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1325 9.5%	1 0.0%	0 0.0%	99.8% 0.2%
	1 0.0%	0 0.0%	0 0.0%	1 0.0%	1 0.0%	0 0.0%	2 0.0%	2 0.0%	1369 9.8%	1 0.0%	99.4% 0.6%
	0 0.0%	1 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1351 9.7%	99.9% 0.1%
99.9% 0.1%	99.4% 0.6%	99.3% 0.7%	99.9% 0.1%	99.5% 0.5%	99.7% 0.3%	99.5% 0.5%	99.0% 1.0%	99.6% 0.4%	99.8% 0.2%	99.6% 0.4%	

Figura 3.18 – Confusion matrix

3.3.5. Discuție pe baza eșecurilor

Recunoașterea cifrelor scrise de mâna este o sarcină dificilă, datorită diferențelor unghiuri și stiluri de scriere. Există numeroase variații de scriere a cifrelor, de la o persoană la alta. Un sistem de recunoaștere a cifrelor scrise de mâna nu poate avea o acuratețe de 100%, deoarece, chiar și oamenii antrenați în acest sens, nu pot recunoaște toate cifrele scrise de mâna. Printre motivele pentru care clasificatorul nu recunoaște cifrele, se numără și contrastul slab sau mediul complicat al imaginii. Una dintre cele mai comune erori se referă la similaritățile dintre clase. Astfel, conform Figurii 3.19 [17], cifrele 1 și 7 au similarități. De asemenea, cifrele 4 și 6 pot avea similarități. [17]

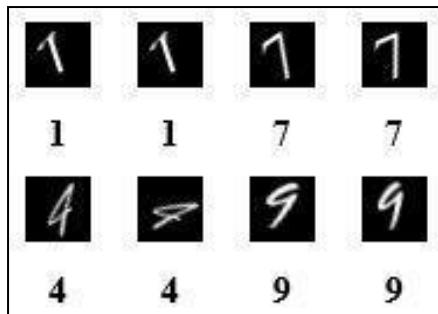


Figura 3.19 – Similarități între cifre

3.4. Optical Character Recognition (OCR)

3.4.1. Introducere

OCR este conversia electronică a imaginii cu scris de tipar, de mâna sau printat, în text editabil. Folosind OCR, un număr mare de documente pe hârtie, scrise în limbi diferite sau formate diferite, pot fi digitalizate și convertite în text editabil, oferind un avantaj, atât în stocarea mai ușoară a documentelor, cât și în permiterea manipulării unor date sau informații care înainte nu ar fi putut fi accesate atât de ușor.

3.4.2. Pipeline de procesare

Diferitele fonturi și modalități de scriere a unui caracter constituie o problemă dificil de rezolvat. Înainte de folosirea unui algoritm OCR, este necesară o procesare a imaginii. Pipeline-ul de procesare include un număr de 9 stagii.

1. De-înclinare

În cazul în care documentul nu a fost corect aliniat când a fost scanat, este posibil să fie necesară înclinarea sa în sensul arcelor de ceasornic sau invers, pentru a face ca textul să fie complet orizontal/vertical.

2. “Despeckle”

Eliminarea zonelor pozitive sau negative, netezirea muchiilor.

3. Binarizare

Convertirea unei imagini în alb și negru (numită imagine binară). Această sarcină reprezintă o modalitate ușoară de a deosebi textul de fundal.

4. Eliminarea liniilor

Ștergerea liniilor care nu aparțin unei imagini.

5. Analiza layout-ului sau “zoning”

Identifică paragrafe și coloane ca și blocuri. Este cu precădere folositor la tabele.

6. Detectia liniilor și a cuvintelor

Stabilirea formelor cuvintelor și caracterelor.

7. Recunoașterea script-ului

Este vitală înainte ca OCR să fie utilizat.

8. Izolarea caracterului

Caracterele ar trebui divizate, fiecare caracter împărțit în mai multe piese, care unite, formează caracterul.

9. Normalizare

Normalizarea rației aspectului, respectiv scalare.

3.4.3. Extragerea caracteristicilor

Există două metode principale de extragere a caracteristicilor în OCR.

1. În prima metodă, algoritmul de detecție a caracteristicilor definește un caracter prin evaluarea liniilor sale, conform Figurii 3.20 [16].

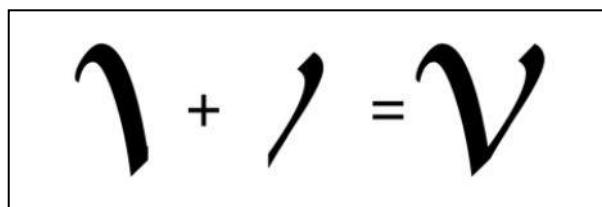


Figura 3.20 – Recunoașterea unui caracter pe baza caracteristicilor

2. În a doua metodă, recunoașterea formei se face prin identificarea întregului caracter. Izolarea caracterului poate fi observată în Figura 3.21 [16].



Figura 3.21 – Izolarea unui caracter

Pentru a izola un caracter, este nevoie de distingerea rândurilor și a coloanelor. Astfel, un rând poate fi recunoscut prin căutarea în imagine a rândurilor de pixeli albi, fără pixeli negri inserați. În mod similar se poate recunoaște unde începe și unde se termină un caracter, aplicând algoritmul anterior pe coloane. Odată ce caracterul a fost izolat, următorul pas este conversia imaginii cu caracterul într-o matrice binară, unde pixelii albi sunt 0 și pixelii negri sunt 1, conform Figurii 3.22 [16].



Figura 3.22 – Conversia unui caracter într-o matrice binară

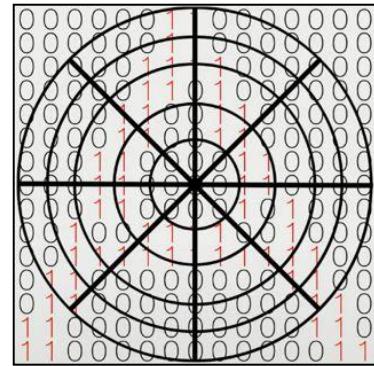


Figura 3.23 – Cercul cu raza cea mai mare distanță

Apoi, folosind formula distanței, se poate determina distanța de la centru matricii către cel mai îndepărtat 1. Se va crea un cerc, având raza cea mai mare distanță de la centru la cel mai îndepărtat 1. Acest cerc va fi divizat în secțiuni, conform Figurii 3.23 [16]. În această etapă, algoritmul compară fiecare secțiune cu o bază de date de matrici reprezentând caractere cu diferite fonturi, pentru a identifica acel caracter cu care are cel mai multe lucruri în comun, din punct de vedere statistic. [16]

3.5. Automatizarea aplicațiilor desktop

3.5.1. Introducere și motivație

Multe afaceri continuă să depindă de aplicații desktop pentru a executa diferite procese. Aplicațiile desktop pot fi automatizate în mai multe moduri. Primul mod este automatizarea testării, prin testarea componentelor din cadrul unei singure aplicații. Al doilea mod, este automatizarea testării, prin testarea tranzacțiilor dintre aplicații (desktop și non-desktop). Al treilea mod, este robotic process automation sau RPA, prin efectuarea unor acțiuni sau tranzacții automate în cadrul uneia sau mai multor aplicații, desktop sau non-desktop.

Primele două moduri au rolul de a asigura că procesele critice din cadrul unei afaceri sunt monitorizate și testate frecvent, permitând găsirea și eliminarea bug-urilor, înainte ca acestea să ajungă la clienții finali. Al treilea mod, RPA, are rolul de a automatiza procese care, altfel, ar fi efectuate de angajați ai afacerii. Toate cele 3 tipuri de automatizare ajută la eliminarea unor sarcini repetitive, ducând la o creștere a productivității și costuri reduse pentru afaceri.

Pentru a demonstra avantajele automatizării proceselor, este ilustrat un exemplu de proces pentru crearea, publicarea și validarea unei polițe de asigurare.

După cum se poate observa în Figura 3.24 [14], acest proces include atât aplicații desktop, cât și aplicații web. În cazul în care ar fi efectuat manual, ar dura mult timp doar pentru a executa un singur test din acest proces, și ar trebui repetat pentru fiecare utilizator. Este costisitor din punct de vedere al timpului, dar esențial afacerii, astfel încât nu poate fi omis sau grăbit. Procesele necesită testare continuă, iar singura modalitate de a face acest lucru, fără a face risipă de resurse, este prin automatizarea acestuia. Mai mult decât atât, acesta este unul dintre sutele sau chiar miile de procese pe care o afacere le are, relifând nevoia de automatizare. [14]

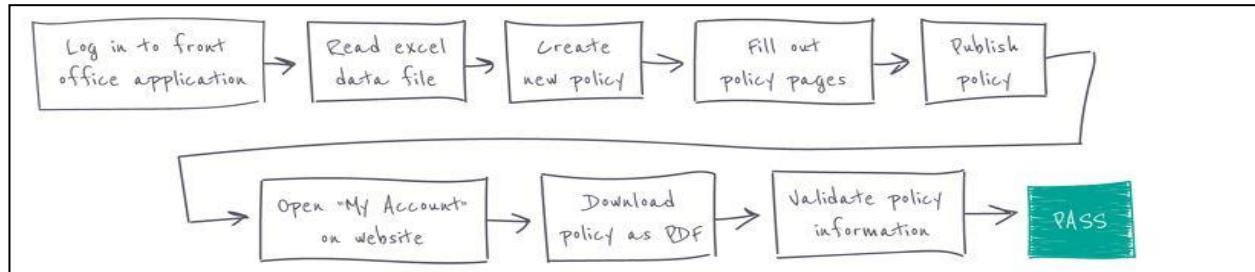


Figura 3.24 – Procesul unei polițe de asigurare

3.5.2. Winium

Pentru aplicații bazate pe Windows, automatizarea se poate face folosind Winium. Winium este un framework open source de automatizare bazat pe Selenium, pentru platforme Windows. Selenium, pe de altă parte, este un framework pentru automatizarea aplicațiilor web. Orice aplicație care rulează în Windows poate fi automatizată. Dezavantajul este că Winium lucrează doar cu aplicațiile de pe Windows, nefiind compatibil cu Mac sau Linux. Principalul avantaj al Winium este faptul că îl open-source. De asemenea, un alt avantaj este dat de faptul că seamănă la mod de utilizare cu Selenium. Așadar, tranziția dintre cele două framework-uri este una ușoară. Aplicația de automatizare poate fi scrisă în diferite limbi de programare: Java, Python, PHP, C#, JavaScript și se poate folosi orice framework de testare: Junit, TestNG, Nunit. [15]

Capitolul 4. Analiză și Fundamentare Teoretică

4.1. Flow-ul aplicațiilor

4.1.1. VHDL

Rețeaua neuronală artificială pentru clasificarea cifrelor din codul numeric personal de pe o imagine cu actul de identitate va fi implementată pe FPGA folosind limbajul VHDL. Rețeaua va avea un input layer și un output layer. Antrenarea rețelei neuronale artificiale este făcută în prealabil, modelul fiind ulterior trimis la placă. Implementarea antrenării pe placa de dezvoltare ar fi costisitoare din punctul de vedere al resurselor necesare. Rețeaua neuronală va fi antrenată folosind un script Python, iar modelul va fi salvat în fișierul "Weights.txt". Fișierul "Inputs.txt" va conține cele 13 input-uri ale rețelei neuronale artificiale, corespunzătoare celor 13 imagini cu cifrele detectate din CNP. Fiecare input din cele 13 va fi reprezentat de siruri de valori binare, constituind valorile neuronilor din rețeaua neuronală artificială.

Transmiterea datelor la placa de dezvoltare se va face folosind terminalul Hterm. Hterm permite transmiterea datelor de la PC la placa de dezvoltare, cât și receptia datelor de la placa de dezvoltare la PC. Transmiterea și receptia vor fi posibile prin implementarea protocolului UART. Algoritmul de implementare al protocolului este descris în subcapitolele următoare. Datele transmise vor fi stocate într-o memorie BRAM. Memoria BRAM este o memorie adițională care permite stocarea datelor de dimensiuni mari. Vor fi implementate câte o memorie pentru fiecare dintre cele două fișiere: o memorie BRAM pentru input-uri, respectiv o memorie BRAM pentru weights-uri.

Cifrele CNP-ului vor fi preluate rând pe rând și vor fi clasificate, folosind un automat cu stări finite. Automatul va prelua inputurile fiecareia din cele 13 imagini cu cifrele CNP-ului, și folosind weights-urile, va clasifica fiecare dintre cele 13 cifre. Output-ul FSM-ului reprezintă chiar array-ul de cifre clasificate de rețeaua neuronală. Acestea vor fi afișate pe SSD-ul plăcii de dezvoltare, iar cele 10 ponderi pentru fiecare cifră clasificată, vor fi afișate pe led-uri.

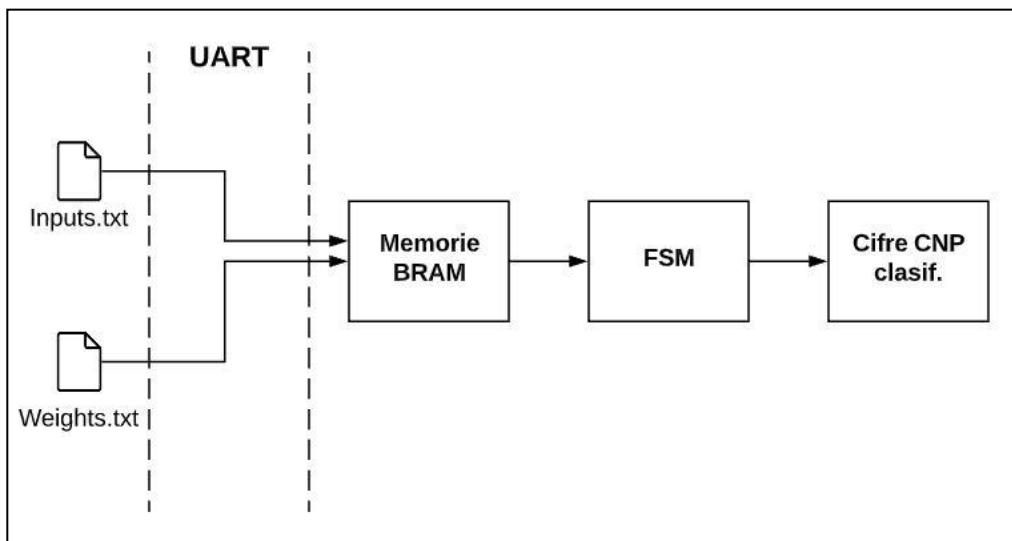


Figura 4.1 – Design-ul rețelei neuronale artificiale pe FPGA

4.1.2. Script licență

Fișierul de input trimis la placa de dezvoltare “Inputs.txt” este creat în urma unui pipeline de procesare a imaginii originale cu actul de identitate. Procesarea imaginii cu buletinul se va face folosind un script Python. Scriptul va citi imaginea aleasă spre a fi procesată. Preprocesarea are rolul de a elimina zgomotele din imagine și de a crea o imagine în care detecția cifrelor să fie cât mai ușor de realizat. Imaginea va fi convertită din RGB în grayscale, făcând ca o imagine cu 3 canale de culori să devină o imagine cu un singur canal de culoare. Apoi se va aplica o binarizare asupra imaginii, ținând cont de valoarea unui threshold. Ulterior se vor elimina zgomotele din imagine, prin intermediul unui proces numit opening. În finalul preprocesării, algoritmul canny va detecta muchiile.

Algoritmul OCR va clasifica și localiza cifrele din CNP. Algoritmul de implementare al OCR este descris în subcapitolele următoare. Ulterior detecției cifrelor din CNP, acestea vor trebui convertite în imagini care să compună dataset-ul rețelei neuronale convoluționale. Astfel, o serie de procesări vor avea loc asupra cifrelor, încât acestea să facă parte din imagini 28X28 pixeli, centrate. Dacă cifrele din imagine nu sunt centrate, clasificarea imaginii va fi greu de realizat. Va trebui ținut cont de faptul că există posibilitatea de apariție a unor neconcordanțe în predicția cifrelor. Trebuie amintit faptul că clasificare cifrelor se face pe un dataset cu cifre scrise de mână, pe când cifrele din CNP sunt cifre de tipar. Cele 13 imagini ale CNP-ului vor alcătui, după procesare, dataset-ul pe care se va face predicția.

Scriptul va conține rețea neuronală convoluțională antrenată. Rețea neuronală convoluțională va conține 3 layeruri de convoluție și 2 layeruri de max-pooling. Rezultatul ultimului layer de convoluție va reprezenta input-ul unei rețele neuronale artificiale complete conectată. Fiecare din cele 13 imagini ale CNP-ului va constitui input-ul rețelei neuronale convoluționale. Pe fiecare imagine se vor aplica layeruri de convoluție și max-pooling. Rezultatele din urma layerelor de convoluție și max-pooling vor compune fișierul “Inputs.txt”. Înainte de a fi salvate în fișierul text, valorile vor fi convertite din numere în virgulă mobilă în fixed point representation. Algoritmul de conversie al numerelor va fi prezentat în subcapitolele următoare. Fișierul text “Inputs.txt” va conține siruri de valori binare. Aceste valori vor fi trimise la placă folosind pipeline-ul descris în subcapitolul anterior. Practic output-ul pipeline-ului de procesare a imaginii cu buletinul constituie input-ul pipeline-ului de clasificare a cifrelor folosind rețea neuronală artificială din VHDL.

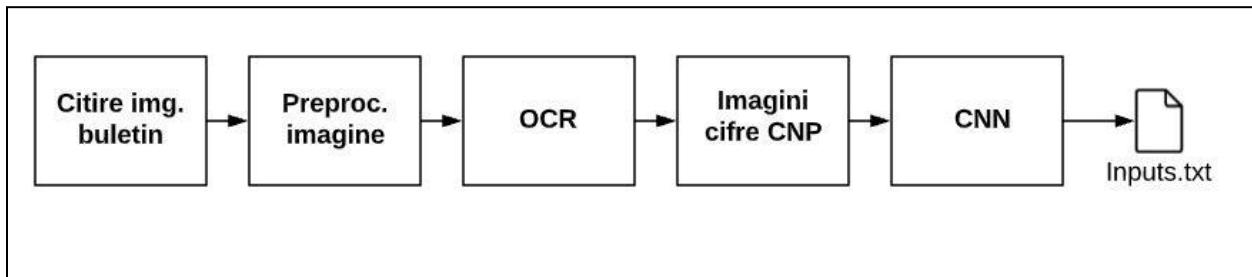


Figura 4.2 – Pipeline-ul de detecție a cifrelor din CNP și creare a fișierului “Inputs.txt”

4.1.3. Aplicație desktop

Aplicația desktop va fi realizată în Delphi. Avantajul implementării aplicației în Delphi este dat de ușurința realizării conexiunii cu baza de date. Un alt avantaj este implementarea facilă a query-urilor. Delphi oferă opțiunea filtrării datelor la nivelul bazei de date, fără a necesita preluarea informațiilor din baza de date și aducerea lor în aplicația desktop.

Utilizatorul va fi nevoie să se logheze pentru a avea acces la funcționalitățile aplicației. Pentru aceasta, trebuie întâi să se înregistreze, folosind numele, username-ul, parola și eventual o poză de profil. Dacă procesul de înregistrare a fost încheiat cu succes, utilizatorul va avea acces la pagina principală din aplicație.

Principalele funcționalități ale aplicației vor fi “Încărcare imagine buletin”, “Vizualizare librărie” și “Vizualizare istoric evenimente”. Pentru a detecta cifrele din CNP, este necesară încărcarea imaginii cu actul de identitate. Imaginea, odată încărcată, va putea fi trimisă la scriptul din Python, descris anterior, care va detecta cifrele din buletin și le va returna sub forma unor imagini individuale de 28X28. De asemenea, o imagine procesată cu buletinul, având cifrele din CNP scrise sub cifrele originale, va apărea în fereastra aplicației. Odată procesată imaginea, există opțiunile de vizualizare a imaginii editate, vizualizare a rezultatelor sub forma imaginilor individuale cu fiecare cifră din CNP, și salvarea imaginilor în baza de date.

O altă funcționalitate, independentă de încărcarea imaginii cu actul de identitate, este “Vizualizare librărie”. Imaginile originale și editate vor putea fi accesate și, în cazul în care dimensiunea lor este prea mică, vor putea fi deschise într-o fereastră nouă. Pentru fiecare imagine va exista o intrare în tabelele de imagini. O tabelă va fi tabela imaginilor originale, iar cealaltă, a imaginilor procesate.

Funcționalitatea “Vizualizare istoric evenimente” permite salvarea fiecărui eveniment, a fiecărei acțiuni, întreprinse de utilizator. Evenimentele vor fi afișate sub forma unei tabele de evenimente.

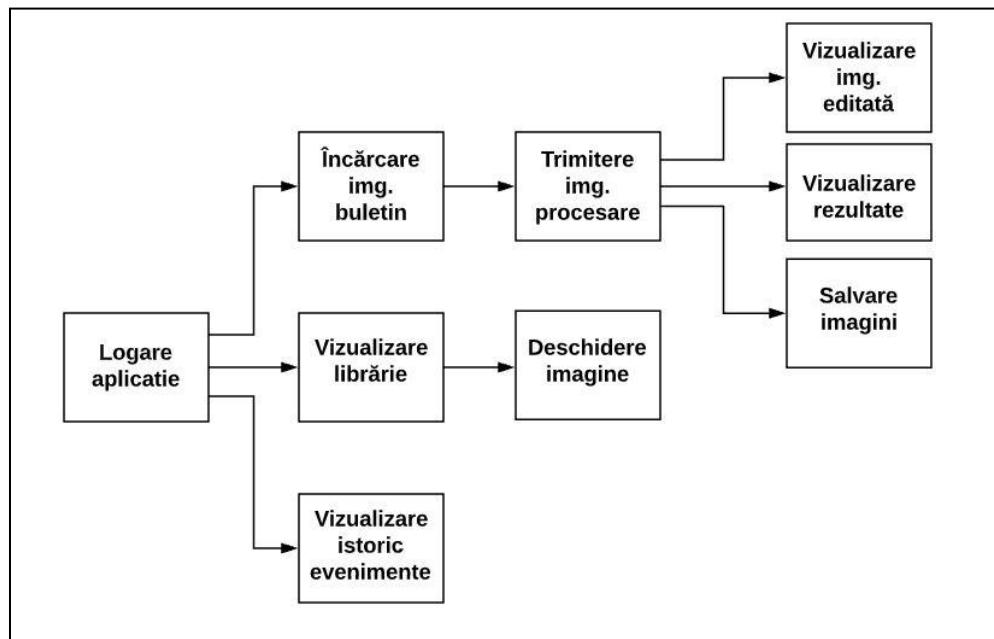


Figura 4.3 – Funcționalitățile aplicației desktop

4.1.4. Automatizare Winium

Aplicația de automatizare va fi scrisă în Java, folosind Winium. Winium permite automatizarea aplicațiilor desktop. Aplicația va fi salvată ca un fișier executabil, și rulată după ce a fost încheiată crearea fișierului de input al rețelei neuronale artificiale din VHDL. Scopul aplicației este de a trimite la placa de dezvoltare fișierul de input și fișierul cu modelul rețelei neuronale artificiale, pentru a clasifica cifrele din CNP, folosind placa de dezvoltare Basys3.

Aplicația va rula terminalul Hterm, folosit pentru a transmite date de la calculator la placă, implicit a recepționa date de la placă. Configurațiile Hterm vor fi făcute folosind un fișier de configurare. Aplicația va selecta fișierul de input din sistemul de fișiere și o va trimite la placa de dezvoltare. Procesul va dura câteva secunde, fapt consemnat de aplicația desktop, care va “dormi” pentru un număr dat de secunde, până când aplicația a fost trimisă cu succes. Următorul pas va fi trimiterea modelului la placa de dezvoltare. Modelul va fi realizat în urma antrenării rețelei neuronale convoluționale în Python. Fișierul cu weights-uri va fi selectat din sistemul de fișiere. Transmiterea fișierului va necesita, din nou, o anumită perioadă de timp. Aplicația va “dormi” în acel interval. Se va oferi o marjă de eroare, în aşa fel încât să nu apară situații neprevăzute. De exemplu, începerea transmisiiei celui de-al doilea fișier, fără ca primul fișier să fie complet trimis.

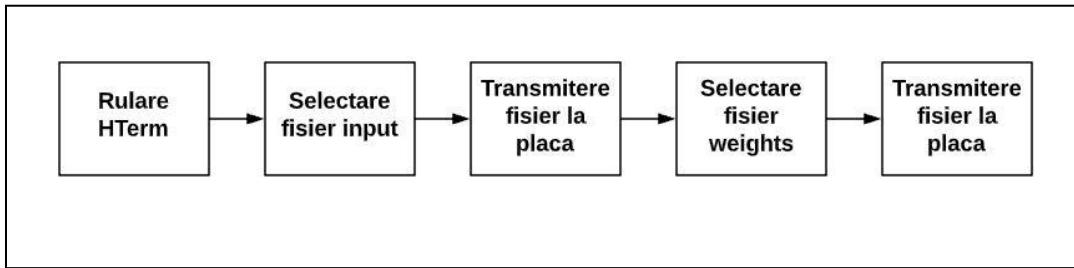


Figura 4.4 – Procesul de automatizare al trimiterii fișierelor la placă

4.1.5. MySQL

Sistemul de gestiune al bazelor de date ales este MySQL. Principalul avantaj este faptul că MySQL este open-source. De asemenea, MySQL acoperă majoritatea funcționalităților necesare aplicației de față. Crearea schemei, a tabelelor, posibilitatea vizualizării schemei. Dezavantajul major este faptul că nu are opțiunea de query builder, cu ajutorul căreia se pot crea ușor query-uri complexe. Cu toate acestea, aplicația de față nu necesită astfel de query-uri, fapt pentru care folosirea MySQL este oportună.

Aplicația desktop este cea care va interacționa cu baza de date. Interacțiunea este facilitată de folosirea obiectelor puse la dispoziție de Delphi pentru manipularea informațiilor din baza de date. Utilizatorii se vor înregistra în aplicație, folosind numele, prenumele, username-ul și parola. Toate acestea vor fi stocate în baza de date. De asemenea, parola va fi encriptată. Imaginele originale și cele procesate vor putea fi salvate și revăzute. Evenimentele sau acțiunile întreprinse de utilizatori, precum alegerea imaginii, transmiterea și salvarea ei, vor fi de asemenea salvate în baza de date. Informațiile vor fi salvate în tabele, aparținând unei scheme.

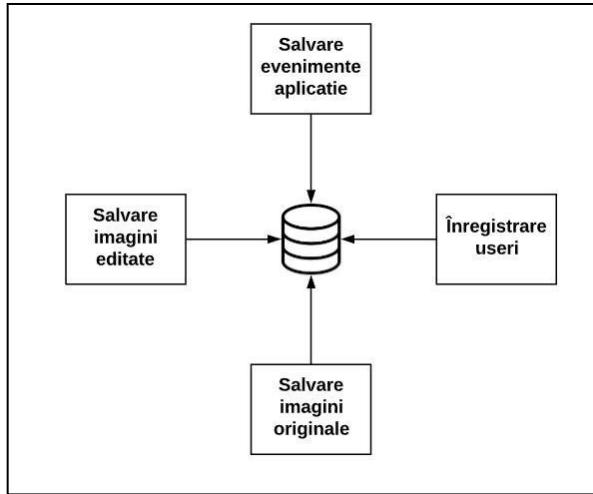


Figura 4.5 – Principalele acțiuni care implică folosirea bazei de date

4.1.6. Rețea neuronală conoluțională

Rețeaua neuronală conoluțională are rolul de a clasifica imagini cu cifre scrise de mâna. De asemenea, modelul determinat de antrenarea rețelei neuronale va fi folosit salvat într-un fișier și transmis la placa de dezvoltare FPGA pentru a putea fi folosit în rețeaua neuronală artificială din VHDL. Crearea dataset-ului de antrenare și testare se rezumă la încărcarea imaginilor din dataset-ul de la MNIST pentru cifre scrise de mâna. Preprocesarea dataset-urilor implică normalizarea pixelilor pentru a avea valori între 0 și 1. De asemenea, imaginile, ca array-uri bidimensionale vor fi convertite în array-uri unidimensionale. Se va adăuga o dimensiune pentru canalul de culoare. Fiind o imagine grayscale, există un singur canal de culoare.

Modelul rețelei neuronale conoluționale conține 3 layeruri de conoluție și 2 layeruri de max-pooling. Acestea au ca și funcție de activare relu. Funcția de activare relu funcționează astfel: dacă valoarea este mai mică decât 0, atunci valoarea va fi 0, altfel rămâne valoarea originală. Output-ul ultimului layer de conoluție este convertit într-un array unidimensional. Ulterior este o rețea neuronală artificială, complet conectată, care are ca input 576 de valori, iar output-ul 10 valori, corespunzătoare celor 10 cifre. Funcția de activare pentru output este sigmoid, făcând ca output-ul să aibă valori cuprinse între 0 și 1.

Modelul se va salva, pentru a fi utilizat din nou. Aceasta este modalitatea prin care weights-urile și bias-urile unui model rămân neschimbate. De fiecare dată când se compilează un model, valorile weights-urilor și bias-urilor se modifică.

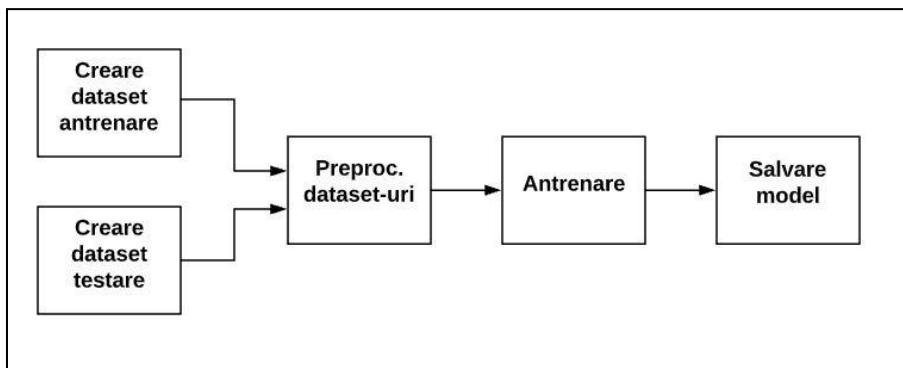


Figura 4.6 – Pipeline rețea neuronală conoluțională

4.2. Diagrame flow pentru cazuri specifice

4.2.1. Trimitere imagine spre procesare

1. Diagrama flow

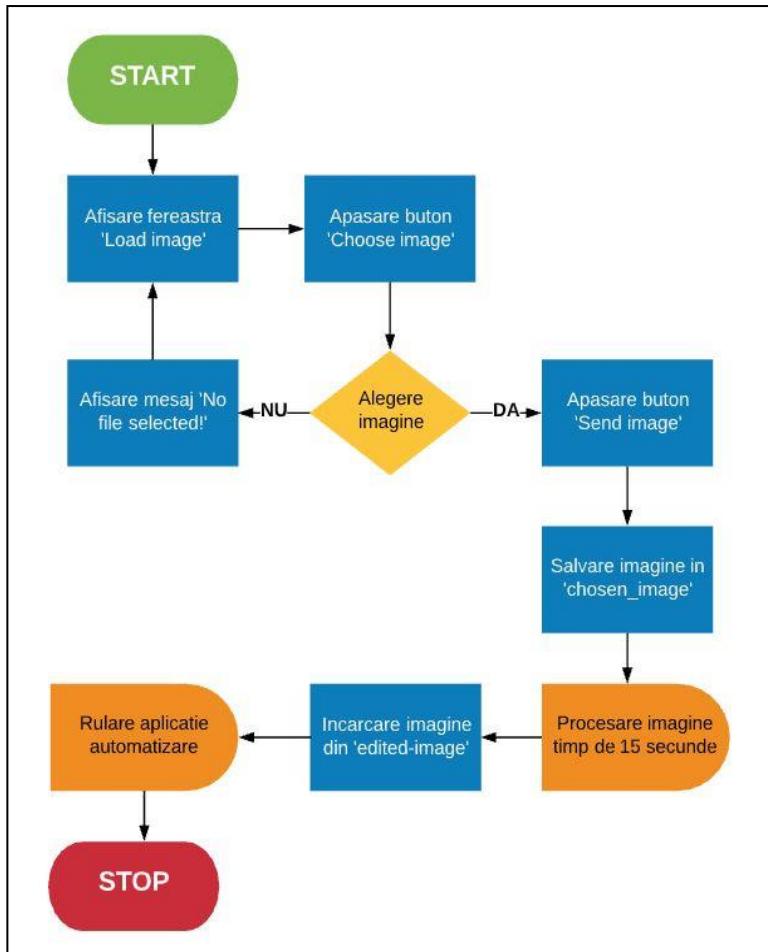


Figura 4.7 – Diagrama flow pentru trimiterea imaginii spre procesare

2. Precondiții

- Rularea aplicațiilor necesare procesării imaginilor.
- Înregistrarea utilizatorului în baza de date.
- Rularea aplicației desktop și logarea utilizatorului.

3. Postcondiții

- Afisarea imaginilor originale și editate în aplicația desktop.
- Afisarea cifrelor clasificare din CNP pe SSD-ul plăcii Basys3.

4. Explicații

Utilizatorul va fi logat în aplicația desktop, fiind afișată fereastra ‘Homepage’. Utilizatorul va apăsa pe butonul ‘Load image’ din meniu. Următorul pas este apăsarea butonului ‘Choose image’ pentru a alege imaginea din sistemul de fișiere. În cazul în care utilizatorul nu alege o imagine, va fi afișat un mesaj de forma ‘No file selected!’. În cazul în care utilizatorul alege o imagine cu un act de identitate, pentru a trimite acea imagine spre procesare, user-ul va apăsa butonul ‘Send image’. Va fi necesară, în prealabil, rularea aplicațiilor necesare procesării imaginilor: scriptul python, încărcarea programului pe placă Basys3, rularea Winium Desktop Driver. După apăsarea butonului ‘Send image’, imaginea originală va fi salvată în folderul ‘chosen_image’, de unde va fi preluată de scriptul din python. Procesarea imaginii de către script va dura în jur de 15 secunde, timp în care aplicația desktop va “dormi”. După terminarea celor 15 secunde, imaginea procesată, cu CNP-ul detectat, va fi încărcată din folderul ‘edited_image’, unde a fost salvată de scriptul python. Următorul pas este rularea aplicației de automatizare, care va transmite fișierele de input și weights la placă de dezvoltare, pentru a clasifica cifrele din CNP folosind rețeaua neuronală artificială implementată în VHDL.

4.2.2. Înregistrarea utilizatorului în aplicația desktop

1. Diagrama flow

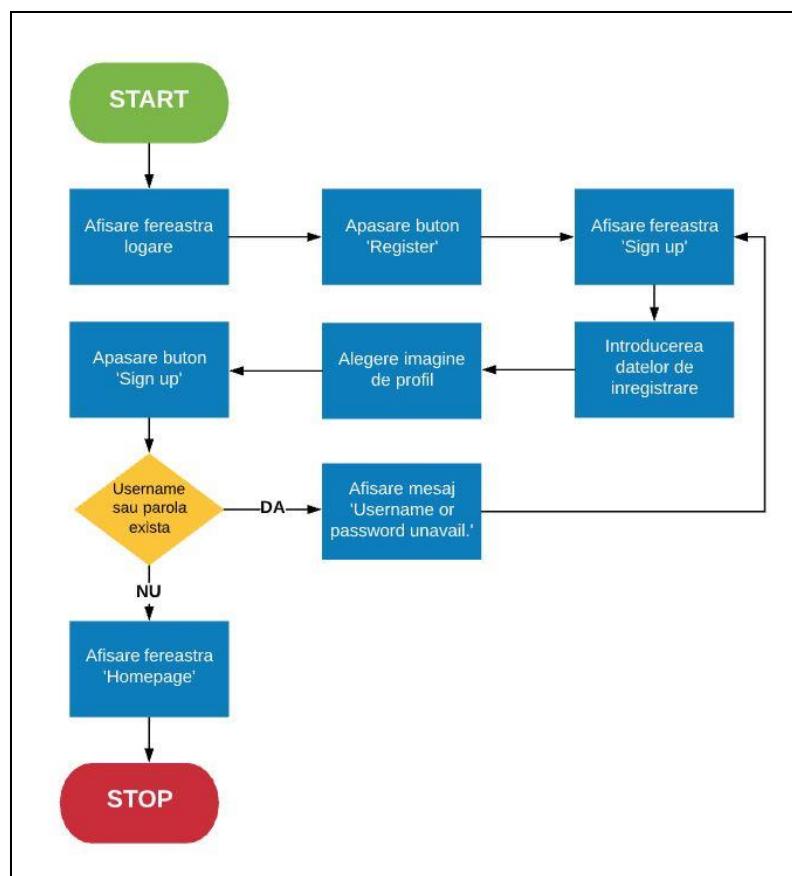


Figura 4.8 – Diagrama flow pentru înregistrarea user-ului în aplicație

2. Precondiții

- Rulare executabil aplicație desktop.
- Utilizatorul nu este înregistrat în baza de date.

3. Postcondiții

- Utilizatorul va fi înregistrat în baza de date.

4. Explicații

Utilizatorul a rulat executabilul aplicației desktop și se află în fereastra de logare. Pentru a se înregistra, utilizatorul va apăsa butonul ‘Register’. Se va afișa fereastra ‘Sign up’. Utilizatorul își va introduce datele precum prenumele, numele, username-ul și parola. De asemenea, utilizatorul are posibilitatea alegerii unei imagini de profil. Următorul pas este apăsarea butonului ‘Sign up’. În cazul în care username-ul sau parola sunt deja existente, se va afișa un mesaj de forma ‘Username or password unavailable!’ și va fi afișată în continuare fereastra de ‘Sign up’. Dacă username-ul și parola sunt disponibile, utilizatorul va fi înregistrat și redirecționat spre pagina principală a aplicației.

4.2.3. Vizualizare imagine librărie

1. Diagrama flow

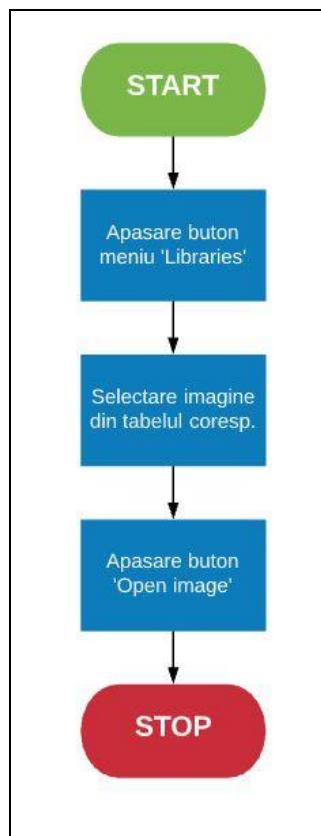


Figura 4.9 – Diagrama flow pentru vizualizarea unei imagini din librărie

2. Precondiții

- Înregistrarea utilizatorului în baza de date.
- Rularea aplicației desktop și logarea utilizatorului.

3. Postcondiții

- Vizualizare unei imagini din libărie într-o fereastră nouă.

4. Explicații

Utilizatorul va fi logat în aplicație și se va afla în fereastra ‘Homepage’. Utilizatorul va apăsa butonul ‘Libraries’ din meniu, pentru a avea acces la imagini. Se vor afișa două tabele, una a imaginilor originale, și cealaltă a imaginilor editate. Utilizatorul va alege imaginea dorită, și o va deschide într-o fereastră nouă, prin apăsarea butonului ‘Open image’.

4.2.4. Rularea aplicațiilor necesare procesării imaginii

1. Diagrama flow

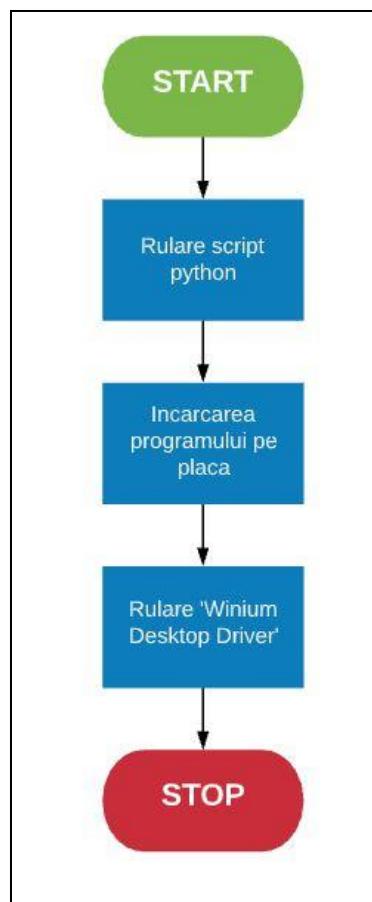


Figura 4.10 – Diagrama flow pentru rularea aplicațiilor necesare proceării imaginii

2. Precondiții
 - Rularea jupyter notebook.
 - Conectarea plăcii Basys3 la USB-ul PC-ului.
3. Postcondiții
 - Vizualizare unei imagini din libărie într-o fereastră nouă.
4. Explicații

Scriptul python va fi rulat pentru a prelua imaginea originală și a detecta cifrele din CNP. Este necesară rularea scriptului înainte ca utilizatorul să trimită imaginea aleasă spre procesare. Încărcarea programului pe placă este necesară pentru a detecta cifrele din CNP folosind rețea neuronală artificială implementată în VHDL. Rularea ‘Winium Desktop Driver’ este necesară pentru funcționarea aplicației care automatizează procesul de transmitere a fișierului de input, respectiv al celui cu modelul, la placa FPGA.

4.3. Algoritmi și protocoale utilizate

4.3.1. Algoritmul de conversie a numerelor din virgulă mobilă în fixed point representation

Valorile folosite de rețea neuronală artificială din VHDL ar fi putut fi reprezentate în una dintre următoarele două forme: IEEE floating point (32 biți precizie simplă sau 64 biți precizie dublă) sau fixed point representation. Floating point ajută în reprezentarea numerelor mari și oferă o precizie mai bună, dar implementarea și manipularea sunt dificile. Aici se adaugă și un consum mare de resurse. Din aceste motive, se va folosi reprezentarea în fixed point. Valorile de input folosite în rețele neuronale sunt, în general, normalize (între 0 și 1 sau -1 și 1), astfel că reprezentarea numerelor mari nu va constitui o problemă. Dezavantajul la reprezentarea în fixed point este acuratețea, dar cât timp nu va apărea overflow varianta cu fixed point va avea o performanță mai bună decât reprezentarea în floating point.

Valorile rețelei neuronale vor fi reprezentate pe 16 biți. Primii 8 biți vor constitui partea întreagă a numărului, iar ultimii 8 biți, partea frațională.

În cele ce urmează, va fi prezentat algoritmul de conversie a numerelor din virgulă mobilă în fixed point representation, atât din punct de vedere practic, cât și conceptual.

4.3.1.1. Algoritm din punct de vedere practic

Funcția convert(x)

Dacă x este un număr pozitiv, partea întreagă a lui x va fi $\text{int}(x)$, iar partea frațională va fi $x - \text{int}(x)$. De exemplu, pentru 1.5, partea întreagă este $\text{int}(1.5)$ adică 1, iar partea frațională este $1.5 - \text{int}(1.5)$, adică $1.5 - 1$, care este egal cu 0.5. Dacă x este un număr negativ, cu partea frațională diferită de 0.0, atunci partea întreagă se va calcula ca fiind $\text{int}(x) - 1$, iar partea frațională va fi $1 + (x - \text{int}(x))$. De exemplu, pentru numărul -2.3, partea întreagă va fi $-2.0 - 1$, adică -3, iar partea frațională va fi $1 + (-2.3 - (-2))$, adică $1 + (-0.3)$, adică 0.7.

Pe cele două valori, pi (partea întreagă) și pf (partea frațională), se vor aplica funcțiile convInt(pi) și convFrac(pf) , care vor converti cele două părți în valori fixed point. După conversia celor două părți, în câte 8 biți fiecare, acestea se vor concatena și vor compune un singur număr, valoarea conversiei din virgulă mobilă în fixed point representation.

Funcția convInt(pi)

Se va continua conversia numerelor pe exemplul cu valorile 1.5, respectiv -2.3. Pentru valoarea 1.5 se va apela convInt(1), iar pentru -2.3, se va apela convInt(-3). Pentru convInt(1), se va verifica dacă valoarea transmisă ca și parametru este mai mare sau egală cu 0, caz în care este convertită în binar, pe 8 biți. Astfel, pib (partea întreagă binar) va fi 0000 0001. Dacă valoarea transmisă ca și parametru este mai mică decât 0, precum în cazul convInt(-3), se va converti, în primă fază, în $-x$, adică 3. Valoarea 3 va fi reprezentată în binar, pe 8 biți, pib = 0000 0011. Se va calcula complementul lui pib, practic biții de 0 vor deveni 1, iar biții de 1 vor deveni 0. Astfel, complementul lui 0000 0011 va fi 1111 1100. Se va adăuga 1 la complement, iar valoarea finală va fi partea întreagă binară a numărului -2.3, adică 1111 1101.

Funcția convFrac(pf)

Se va continua conversia numerelor pe exemplul cu valorile 1.5, respectiv -2.3. Pentru valoarea 1.5 se va apela convFrac(0.5), iar pentru -2.3, se va apela convFrac(0.7). Se vor compune, rând pe rând, cei 8 biți ai părții fracționale. Se va calcula dublul lui 0.5. Dacă dublul este mai mare sau egal cu 1, atunci bitul corespunzător poziției va fi 1, și se va scădea 1 din valoarea dublului. Dacă dublul este mai mic decât 1, atunci bitul corespunzător poziției va fi 0. Acțiunea se repetă de 8 ori, pentru fiecare bit al părții fracționale. Astfel, după conversie, valoarea lui convFrac(0.5) va fi 1000 000, în timp ce valoarea lui convFrac(0.7) va fi 1011 0011.

4.3.1.2. Algoritmul din punct de vedere conceptual

Reprezentarea numărului -13.8125 în fixed point

Pas 1: Caut cel mai mare număr întreg, mai mic decât -13, și îl reprezint în binar.

Răspuns: Cel mai mare număr întreg mai mic decât -13 este -14.

$$\begin{array}{ll} 14: 0000\ 1110 & \\ C1: 1111\ 0001 & -14(10) = 11110010(2) \\ -14: 1111\ 0010 & \end{array}$$

Pas 2: Calculez diferența de la -14 la -13.8125.

Răspuns: $14 - 13.8125 = 0.1875$

Pas 3: Reprezint 0.1875 în binar.

$$\begin{array}{ll} 0.1875 * 2 = 0.375 & \rightarrow 0 \\ 0.375 * 2 = 0.75 & \rightarrow 0 \\ 0.75 * 2 = 1.5 & \rightarrow 1 \\ 0.5 * 2 = 1.0 & \rightarrow 1 \\ 0.0 * 2 = 0.0 & \rightarrow 0 \\ 0.0 * 2 = 0.0 & \rightarrow 0 \\ 0.0 * 2 = 0.0 & \rightarrow 0 \\ 0.0 * 2 = 0.0 & \rightarrow 0 \end{array} \quad \downarrow \quad 0.1875(10) = 00110000(2)$$

Răspuns final: $-13.8125(10) = 1111001000110000(2)$

Figura 4.11 – Algoritmul de conversie din punct de vedere conceptual

4.3.2. Algoritmul identificare caractere CNP

Optical Character Recognition sau OCR este un algoritm folosit pentru a detecta și localiza caracterele dintr-o imagine. Folosind OCR, vor fi detectate cifrele care formează codul numeric personal într-o imagine cu un act de identitate. OCR are mai multe implementări. Pentru lucrarea de cercetare de față, a fost aleasă implementarea OCR folosind tesseract.

Imaginea aleasă spre a fi editată va fi salvată în folderul chosen_image. Imaginea va fi citită de scriptul python. Imaginea va fi RGB. Este necesară o preprocesare a imaginii. Astfel, întâi va fi transformată într-o imagine grayscale, apoi se va aplica canny edge detection pentru a detecta muchiile. Aceste două transformări permit o detecție mai facilă a caracterelor.

Folosind tesseract, și având ca input imaginea canny, vor fi detectate caracterele din actul de identitate. Acestea vor fi salvate într-un array, alături de alte informații precum coordonatele x,y, width-ul și height-ul.

Array-ul de caractere va fi parcurs de la primul caracter, până la ultimul. Se va urmări găsirea a 13 caractere numerice succesive. Primul sir de 13 caractere numerice succesive va fi salvat într-un array separat. Apoi, folosind informațiile adiționale precum coordonatele, width-ul și height-ul, cifrele din CNP vor fi încadrate în bounding box-uri.

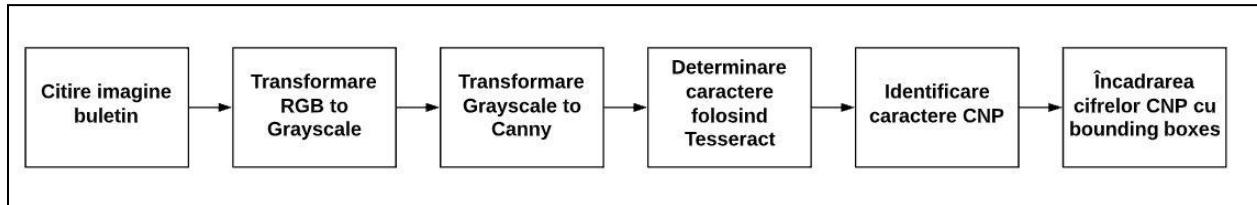


Figura 4.12 – Pipeline identificare caractere CNP

4.3.3. Algoritm conversie din caractere ASCII în șiruri de biți

Acest algoritm va fi folosit pentru a prelua caracterele trimise de calculator la placa de dezvoltare, și de a le stoca într-un mod corespunzător. Fișierele de input și de weights au ca valori șiruri de biți, constituind reprezentarea numerelor în fixed point. Astfel, o posibilă valoarea a unui weight ar putea fi 1111 0000 1111 0000. Numerele sunt salvate pe 16 biți, primii 8 biți fiind partea întreagă, și ultimii 8, partea frațională. O însiruire de două ponderi din fișierul weights ar arăta astfel: 1111000011110000 000011100001111. Șirurile sunt delimitate prin spațiu. Terminalul Hterm va trimite la placă, folosind protocolul UART, fiecare caracter în parte. Astfel, pe exemplul 1111000011110000, primul caracter trimis va fi ‘1’. Caracterul va fi recepționat de placă, sub forma unei valori ASCII, implicit 49. Algoritmul de conversie va scădea 48 de fiecare dată când întâlnește un caracter mai mare decât 48. Astfel, pentru primul caracter, se va face scăderea 49 minus 48, iar rezultatul, 1, va fi salvat la poziția 0 din șirul de caractere din VHDL. Poziția va fi incrementată. Următorul caracter va fi tot ‘1’, astfel că la poziția 1 va fi stocat tot 1, în urma diferenței 49 minus 48. Dacă în loc de 1 ar fi fost 0, atunci scăderea 48 minus 48 ar fi avut ca rezultat 0, care ar fi fost salvat la poziția 1. Algoritmul cotină până când se citește spațiul dintre șiruri. Spațiul este reprezentat în codul ASCII ca și 32. La citirea spațiului, poziția se va reinitializa cu 0 și se va incrementa adresa memoriei BRAM, pentru a salva următorul weight.

4.4. Tehnologii

Alegerea tehnologiilor pentru diferitele componente ale sistemului este o sarcină complexă, în care se ține cont de mai multe metrii. Alegerea și focusarea asupra unei singure metrii, poate duce la neglijarea celorlalte metrii. De exemplu, alegerea ca metrică a "dificultății de a crea un program", poate duce la folosirea unor medii de dezvoltare high-level, care ar putea genera programe de dimensiuni mai mari. Astfel, există un tradeoff, indiferent de metrica primordială aleasă.

Principalele metrii pentru a alege tehnologiile care stau la baza conceperii sistemului au fost „rapiditatea” (speed) de a crea suita de aplicații și „ușurința de utilizare” (ease of use). Din acest motiv, au fost folosite IDE-uri care au implementat librării complexe, care ar putea veni cu un cost, și anume dimensiunea mai mare a programelor.

În cele ce urmează, vor fi prezentate principalele tehnologii utilizate, în funcție de componenta sistemului care a fost implementată.

Rețea neuronală artificială

Mediu de dezvoltare: Xilinx Vivado 2020.

Limbaj: VHDL.

Resurse adiționale: Placa de dezvoltare Basys3, Hterm.

Rețea neuronală artificială folosită pentru a clasifica cifrele din codul numeric personal va fi implementată în limbajul VHDL, folosind placa de dezvoltare Basys3. Valorile din input și weights vor fi trimise de la calculator la placă, folosind terminalul Hterm. Valorile celor două fișiere vor fi stocate în două memorii block RAM. Mediul Xilinx Vivado permite folosire IP integrator pentru crearea celor două memorii BRAM. Valorile vor fi reprezentate în fixed point, folosind librăria ieee.fixed_pkg.all. Detectia se va face folosind un automat cu stări finite, pentru fiecare stare fiind implementată o procedură separată. Mediul Xilinx Vivado permite implementarea librăriilor pentru stocarea procedurilor. Output-ul rețelei neuronale va fi afișat pe SSD, folosind un barrel shifter. Se vor folosi butoanele, switch-urile și led-urile pentru afișarea diferitelor valori pe placa de dezvoltare Basys3.

Rețea neuronală conoluțională

Mediu de dezvoltare: Jupyter Notebook folosind Anaconda Navigator.

Limbaj: Python.

Resurse adiționale: Tensorflow.

Anaconda Navigator permite crearea și rularea unui environment. Environment-ul este folosit pentru instalarea librăriilor necesare anumitor proiecte. În funcție de proiect, se poate crea un environment separat, pentru a stoca librăriile utile aceluiași proiect. Jupyter Notebook permite crearea unor script-uri python interactive, în care se îmbină codul, cu texte sub formă de markdown, imagini, video și diferite grafice implementate folosind librăria Python. Pentru a crea rețea neuronală conoluțională s-a optat pentru limbajul Python întrucât este open-source, ușor de utilizat, intuitiv și permite folosirea unor librării specializate. Tensorflow este o librărie utilizată pentru antrenarea și testarea unor modele de rețele neuronale. Se va folosi dataset-ul de

la MNIST de cifre scrise de mâna pentru antrenarea și testarea modelului. Modelul, odată antrenat, va fi salvat și trimis la placa de dezvoltare Basys3, pentru a detecta cifrele din CNP.

Script OCR

Mediul de dezvoltare: Jupyter Notebook folosind Anaconda Navigator.

Limbaj: Python.

Resurse adiționale: Tesseract.

Anaconda Navigator permite crearea și rularea unui environment. Environment-ul este folosit pentru instalarea librăriilor necesare anumitor proiecte. În funcție de proiect, se poate crea un environment separat, pentru a stoca librăriile utile aceluiași proiect. Jupyter Notebook permite crearea unor script-uri python interactive, în care se îmbină codul, cu texte sub formă de markdown, imagini, video și diferite grafice implementate folosind librăria Python. Pentru a crea scriptul OCR s-a optat pentru limbajul Python întrucât este open-source, ușor de utilizat, intuitiv și permite folosirea unor librării specializate. Script-ul OCR are scopul de a clasifica și localiza cifrele din codul numeric personal, dintr-o imagine cu actul de identitate. Imaginele vor fi preprocesate, iar folosind librăria Tesseract, se vor detecta cifrele. Se va crea o imagine corespunzătoare fiecărei cifre din CNP, care va constitui input-ul rețelei neuronale artificiale din VHDL. Fișierul cu input-uri va fi trimis la placa FPGA folosind Hterm.

Aplicație automatizare

Mediul de dezvoltare: Eclipse.

Limbaj: Java.

Resurse adiționale: Winium, Inspect.exe, Winium Desktop Driver.

Aplicația de automatizare are scopul de a trimite automat fișierele de input și weights la placa de dezvoltare FPGA. S-a ales folosirea Winium în acest sens, care este un framework open source de automatizare bazat pe Selenium, pentru platforme Windows. Aplicația Winium se va crea folosind limbajul Java. Pentru a detecta id-ul sau denumirile butoanelor implicate în procesul de automatizare, se va folosi tool-ul Inspect.exe. Acest tool relevă principalele informații despre componente din interfață grafică, precum butoane, label-uri, textfield-uri, panel-uri. Pentru a rula aplicația de automatizare, trebuie întâi asignat un port procesului. Aceste lucru se face prin rularea Winium Desktop Driver, care asignează portul 9999.

Aplicație desktop

Mediul de dezvoltare: Delphi.

Limbaj: Objective Pascal.

Resurse adiționale: ImageButton.

Aplicația desktop va fi implementată folosind framework-ul Delphi. Principalul avantaj al acestui framework este viteza/rapiditatea de creare a aplicației. Se folosesc obiecte specializate, ușor de utilizat, pentru conexiunea și comunicarea cu baza de date. Dezavantajul

folosirii Delphi este imposibilitatea, teoretică, de a colora butoane sau de a aplica imagini pe butoane. Acest dezavantaj a fost eliminat prin implementarea librăriei ImageButton, care permite folosirea unor imagini pe fundalul butoanelor. Imaginele de pe butoane ar urma să fie implementate folosind tool-uri specializate precum Adobe XD. ImageButton permite și folosirea a 3 imagini separate, pentru diferitele stări în care se află butonul: neapăsat, atins și apăsat.

Baza de date

Mediu de dezvoltare: MYSQL Workbench.

Limbaj: SQL.

Resurse adiționale: XAMPP.

Informațiile despre utilizatori, imaginile originale și editate se vor salva într-o bază de date SQL. Datele vor fi preluate de aplicația desktop din tabelele bazei de date, folosind query-uri. Relațiile dintre tabele se vor face folosind opțiunea de forward și reverse engineering pusă la dispoziție de MYSQL Workbench. Principalul avantaj al MYSQL Workbench constă în faptul că este open-source. Pe de altă parte, unul dintre cele mai mari dezavantaje ale MYSQL Workbench este imposibilitatea de a folosi query builder pentru a crea query-uri complexe. Query builder ar duce la crearea mai rapidă a interogărilor. Pe de altă parte, aceste interogări pot fi scrise manual, astfel că dezavantajul poate fi anulat. Ca și resurse adiționale, XAMPP permite crearea unei baze de date MYSQL locale.

GUI

Mediu de dezvoltare: Adobe XD, GIMP.

Interfața grafică a aplicației desktop va fi creată folosind Adobe XD, respectiv GIMP. Adobe XD va fi folosit pentru crearea imaginilor de fundal ale butoanelor. Se vor realiza câte 3 imagini pentru fiecare buton, câte o imagine pentru fiecare stare: neapăsat, atins și apăsat. Imaginele de background ale aplicației vor fi realizate folosind GIMP.

Capitolul 5. Proiectare de Detaliu și Implementare

5.1. VHDL

5.1.1. UART_TX

Implementarea UART_TX permite transmiterea datelor de la placa de dezvoltare FPGA la calculator, pe baza protocolului UART. Astfel, transmisia datelor se realizează bit cu bit. Întâi este transmis bitul de start, urmat de un octet de date (8 biți), urmat de bitul de stop. Rata de transmisie a datelor poate varia. Valori comune pentru baud rate sunt 9600 sau 115200. Baud rate-ul se referă la numărul de biți transmiși pe secundă. Astfel, un baud rate de 9600 ar transmite 960 caractere pe secundă: pentru fiecare caracter se transmit 8 biți de date și biții de start respectiv stop. Bitul de start este 0 iar bitul de stop este 1. Tranzitia liniei TX din 1 în 0 denotă începerea transmisiei unui caracter. Datele transmise folosind UART sunt codificate folosind codul ASCII. Pentru a transmite caracterul ‘a’, biții de date ar trebui să fie de forma 01100001b, unde primul bit este LSB iar ultimul este MSB.

Atât transmisia cât și receptia datelor necesită un automat cu stări fininte (FSM). FSM-ul pentru transmisie are ca intrări TX_DATA pe 8 biți, reprezentând cei 8 biți de date, TX_EN semnal pe un bit care semnalează intenția de transmitere a unui character și care funcționează în corelație cu BAUD_EN, și un semnal RST de reset. Ieșirile FSM-ului sunt TX pe 1 bit, reprezentând bitul transmis și TX_RDY.

Placa de dezvoltare Basys3 are o frecvență de 100 MHz. Astfel, perioada pentru semnalul de ceas este de 10 ns. La un baud rate de 9600, se transmit 9600 biți pe secundă. Astfel, pentru a se transmite un singur bit, este nevoie de 0.00010416 secunde. Având în vedere că perioada este de 10 ns, se va împărti timpul necesar transmiterii unui bit la valoarea perioadei, pentru a determina de câte perioade este necesar pentru a transmite un bit. Rezultatul este 10416. Astfel, este nevoie de un counter care să numere până la 10416, iar când ajunge la acea valoare, semnalul BAUD_EN va fi ‘1’. Această metodă asigură faptul că un bit rămâne pe linia de transmisie pentru intervalul dat de baud rate. Tranzitia dintre stări se face doar dacă BAUD_EN este ‘1’.

FSM-ul pentru transmisie conține 4 stări (IDLE, START, BITT, STOP) iar tranzitia se face pe frontul de ceas cresător și semnalul BAUD_EN este ‘1’. FSM-ul conține și un bistabil cu set și reset, în care Set-ul este dat de TX_EN (prin apăsarea unui buton sau apariția unui eveniment care semnalează intenția de transmitere a unui caracter), iar reset-ul de BAUD_EN. Folosind această metodă, ieșirea acestui bistabil va fi ‘1’, chiar dacă butonul este lăsat, până cand BAUD_EN va fi ‘1’, caz în care se va activa reset-ul și ieșirea va fi ‘0’.

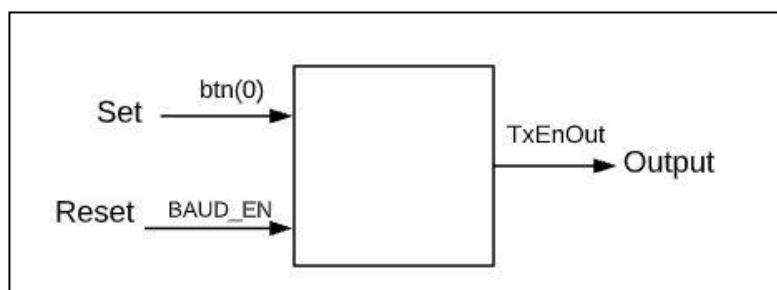


Figura 5.1 – Bistabil SR pentru controlarea FSM-ului

Cazuri bistabil SR:

1. Apăsare buton. Set-ul este ‘1’ și Reset-ul este ‘0’. Ieșirea bistabilului va fi ‘1’.
2. Eliberare buton: Set-ul devine ‘0’, Reset-ul este ‘0’. TxEnOut rămâne ‘1’, Dacă nu se folosea bistabilul SR, output-ul devinea ‘0’ și nu ar fi tranzitat din IDLE în START.
3. BAUD_EN devine ‘1’ când counter-ul ajunge la 10416. Semnalează faptul că se poate transmite alt bit de date. Set-ul este ‘0’ și Reset-ul este ‘1’, astfel ieșirea va deveni ‘0’.

Automatul are 4 stări, exemplificate în Figura 5.2 [19]. Stările automatului sunt:

1. IDLE: TX este ‘1’ și TX_RDY este ‘1’. Semnifică disponibilitatea transmiterii unui nou caracter. De asemenea, counter-ul folosit în starea BITT pentru a transmite rând pe rând biții de date este inițializat cu “000”. Trecerea din IDLE în START se face când TX_EN_OUT (ieșirea bistabilului cu Set TX_EN si Reset BAUD_EN) este ‘1’.
2. START: Denotă inițierea unei transmiteri de caracter. Conform protocolului, TX este ‘0’ (la start) și TX_RDY devine ‘0’, simbolizând că linia este ocupată.
3. BITT: Se transmit, rând pe rând, datele din TX_DATA, folosind un counter care se incrementează până la 7. TX_RDY este în continuare ‘0’. Se transmit toți biții din caracter și se trece în starea următoare.
4. STOP: Se transmite bitul de stop TX = ‘1’. Conform protocolului, TX_RDY este în continuare ‘0’.

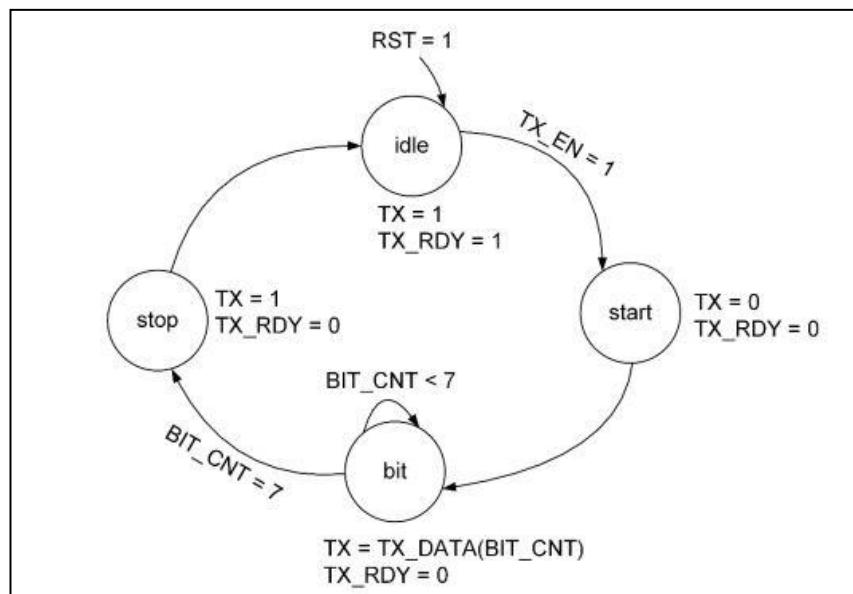


Figura 5.2 – FSM TX

Pentru a exemplifica funcționarea transmiterii datelor de la placa de dezvoltare la calculator, a fost realizată o aplicație care transmite datele de pe switch-uri (7 downto 0) la calculator, folosind HTERM. Parametrii pentru aplicație sunt: baud rate 9600, 8 biți de date, 1 bit de stop, fără paritate. Port-ul este detectat automat. Transmiterea caracterelor se face prin

apăsarea butonului din centru (btn(0)), iar datele sunt reprezentate pe switch-uri folosind codul ASCII. Astfel, pentru a transmite caracterul ‘a’, a fost nevoie de reprezentarea acestuia în ASCII (numărul 97 în zecimal).

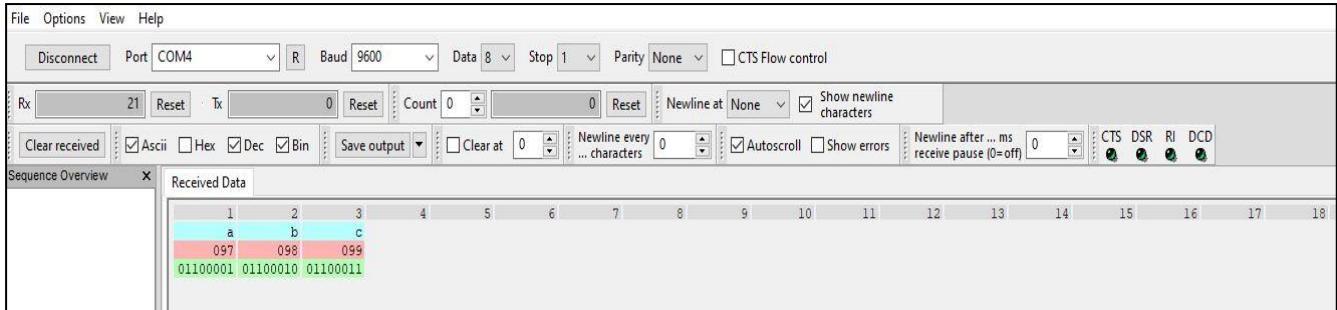


Figura 5.3 – Transmiterea caracterelor de la placă la calculator folosind HTerm

5.1.2. UART_RX

Implementarea UART_TX permite recepția datelor de la calculator la placa de dezvoltare FPGA, pe baza protocolului UART. Valori comune pentru baud rate sunt 9600 sau 115200. Baud rate-ul se referă la numărul de biți transmiși pe secundă. Astfel, un baud rate de 9600 ar transmite 960 caractere pe secundă: pentru fiecare caracter se transmit 8 biți de date și biții de start respectiv stop. Rata de eșantionare de la recepție nu trebuie să coincidă cu rata de eșantionare de la transmisie. Dacă cele două ar fi egale, ar putea determina apariția decalajelor, care ar duce la citirea dublă a unui bit sau chiar sărarea peste un bit. Astfel, este necesară suprareeșantionarea în cazul recepției. Biții de date ar urma să fie citiți aproximativ la mijlocul intervalului de date. Rata de recepție va fi de 16 ori mai mare decât cea de transmisie. Fiecare bit de pe linia serială va fi citit de 16 ori, doar una dintre acestea urmând a fi stocate.

Se va folosi un FSM pentru implementarea receptiei. Acest FSM are ca intrări bitul RX care reprezintă biții care vor fi transmiși de calculator la placă, fie de date, fie de start/stop, RST pentru a reseta automatul și BAUD_EN pentru a realiza tranziția dintre stări. Ieșirile FSM-ului sunt RX_DATA pe 8 biți, reprezentând biții de date transmiși de la calculator la placă, și RX_RDY care denotă disponibilitatea de a receptiona un caracter.

Rata de recepție trebuie să fie de 16 ori mai mare decât cea de transmisie. La un baud rate de 9600, se transmit 960 caractere pe secundă. Fiind nevoie de o eșantionare, baud rate-ul va deveni $16 * 9600$, adică 153600 biți pe secundă. Astfel, un bit se transferă în 0.00000651041 secunde. Având în vedere că perioada este de 10 ns, este nevoie de un counter care să numere până la 651 pentru a face ca BAUD_EN să devină ‘1’.

Automatul are 5 stări, exemplificate în Figura 5.4 [19]. Stările automatului sunt:

1. IDLE: RX_RDY este ‘0’. Sunt inițializate cu 0 baudCnt și bitCnt. BaudCnt este un counter care se incrementează de fiecare dată când BAUD_EN este ‘1’. Dacă RX este ‘0’, se va tranziționa în starea următoare de START.
2. START: Dacă RX este ‘1’, se trece în IDLE. Altfel, se incrementează baudCnt până la 7 (mijlocul intervalului de eșantionare). Când baudCnt ajunge la 7, acesta este reinițializat și se trece în starea următoare. RX_RDY este în continuare ‘0’.
3. BITT: BaudCnt numără până la 15. De fiecare dată când ajunge la 15, se incrementează bitCnt. BitCnt este folosit pentru a salva, rând pe rând, valorile venite pe linia serială RX, în vectorul RX_DATA. Când baudCnt este 15 și bitCnt

- este 7, se va trece în starea următoare. RX_RDY este ‘0’. BaudCnt este reininitializat.
4. STOP: BaudCnt este incrementat până la 15. Odată ce ajunge la 15, se trece în starea următoare. RX_RDY este ‘0’.
 5. WAITT: BaudCnt se incrementează până ajunge la 7 (numărul de biți rămași din moment ce amplasarea se face mereu la jumătatea eşantionării pentru a prelua datele și a asigura redundanță), și se va trece în IDLE. RX_RDY este de această dată ‘1’.

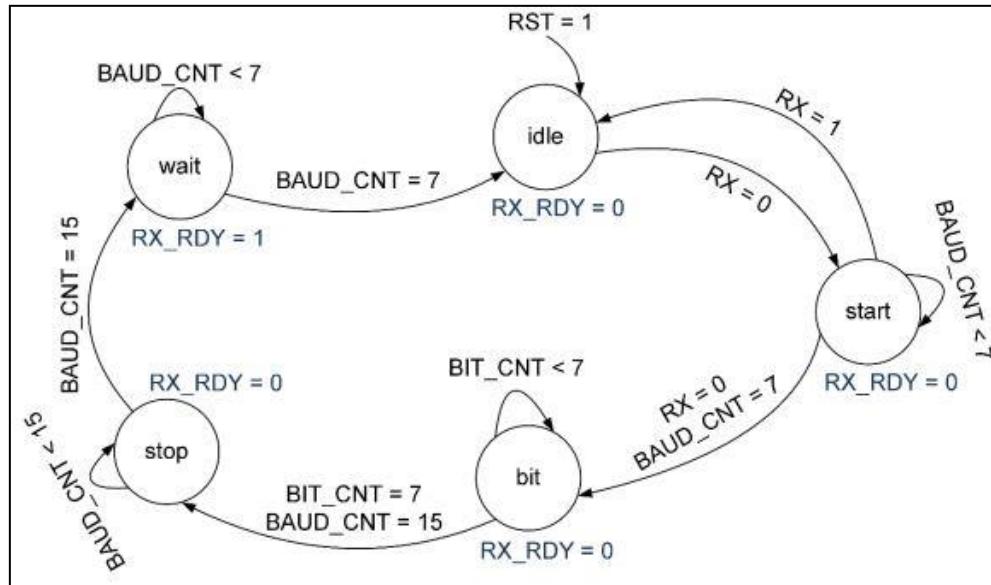


Figura 5.4 – FSM RX

Pentru a exemplifica funcționarea receptiei datelor de la calculator la placa de dezvoltare, a fost realizată o aplicație care preia datele transmise din HTERM și le afișează pe SSD-ul plăcii de dezvoltare Basys3. În exemplul de față, a fost transmis caracterul ‘a’ de la calculator la placă. Codul ASCII al caracterului ‘a’ este valoarea zecimală 97. Astfel, pe SSD va fi afișată valoarea în hexazecimal a caracterului, adică 61.

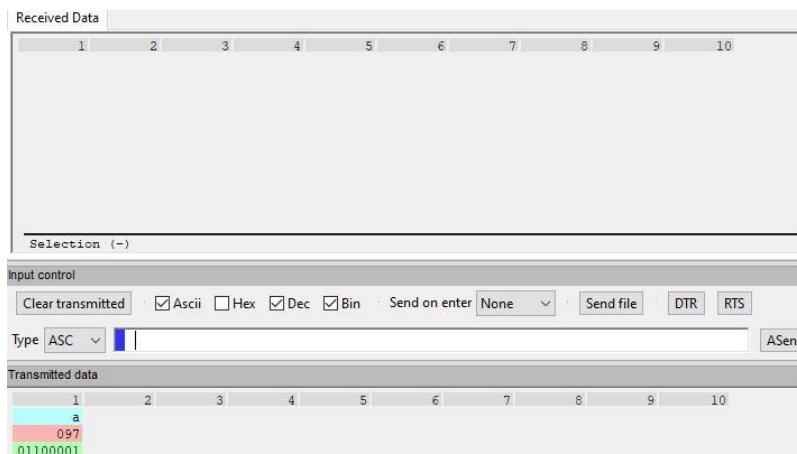


Figura 5.5 – Transmiterea caracterului de la calculator la placă

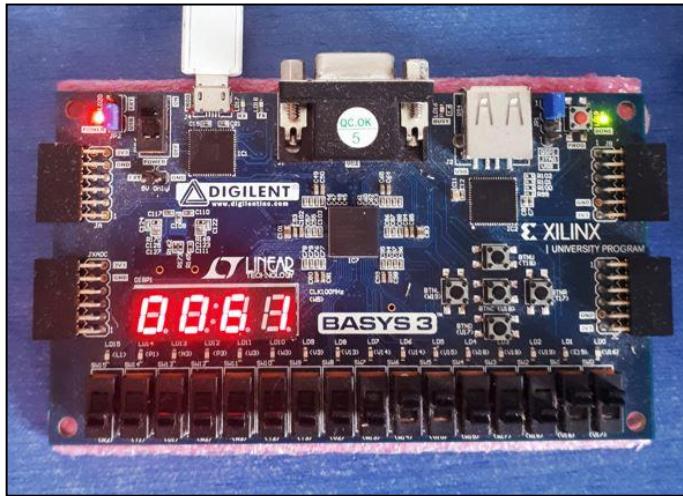


Figura 5.6 – Afişarea pe SSD a caracterului recepţionat de la calculator

5.1.3. Reprezentarea numerelor în fixed point

Rețeaua neuronală artificială implementată în VHDL necesită o reprezentare a numerelor. Rețeaua are un input layer și un output layer. Input-ul este reprezentat de 576 valori, care constituie output-ul din urma proceselor de convoluții și max-pooling efectuate pe imaginea cu cifra. Aceste input-uri sunt înmulțite cu weights-uri, construind output-ul. Teoretic, valorile de input și weights ar fi putut fi salvate sub formă de întregi. Avantajul major ar fi fost ușurința de a converti numerele din binar în zecimal. Dezavantajul, însă, a contribuit decisiv la decizia reprezentării numerelor în fixed point. Înmulțind numere întregi, eroarea rezultată ar fi fost foarte mare. Din acest punct de vedere, reprezentarea în floating point ar fi fost cea mai efectivă. Cu toate acestea, dificultatea reprezentării numerelor în virgulă mobilă în VHDL a determinat ca în final, folosirea reprezentării numerelor în fixed point să fie cea mai potrivită în situația de față.

A fost creată o aplicație de test care va realiza suma și produsul a două numere reprezentate în fixed point, și le va afișa pe led-uri. Aceasta va exemplifica perfect avantajele și dezavantajele folosirii acestui tip de reprezentare a numerelor.

Pentru a putea folosi reprezentarea în fixed point este necesară introducerea în TCL Console a liniilor de cod din Figura 5.7. De asemenea, trebuie inclusă în antet librăria, prin scrierea ”use ieee.fixed_pkg.all;”.

```
Tcl Console × Messages Serial I/O Links Serial I/O Scans
Q | X | ♦ | II | G | E | W |
add_files C:/Xilinx/Vivado/2020.1/scripts/rt/data/fixed_pkg_2008.vhd
WARNING: [filemgr: 56-12] File 'C:/Xilinx/Vivado/2020.1/scripts/rt/data/fixed_pkg_2008.vhd' cannot be added to the project because it already exists in the project, skipping this file
set_property library ieee [get_files C:/Xilinx/Vivado/2020.1/scripts/rt/data/fixed_pkg_2008.vhd]
```

Figura 5.7 – Includerea librăriei fixed point în proiect folosind TCL Console

Au fost declarate și inițializate două semnale a și b de tipul sfixed(7 downto -8). Acest lucru denotă că pentru a reprezenta un număr, vor fi folosiți 16 biți. Primii 8 biți reprezintă partea întreagă a numărului, (7 downto 0), iar ultimii 8 biți constituie partea fracțională (-1 downto -8). Algoritmul folosit pentru conversia numerelor a fost explicitat în capitolul anterior.

Semnalul a are valoarea -1.35 ("111111010100110" în fixed point), iar semnalul b are valoarea -2.38 ("1111110110011110" în fixed point). De asemenea, au fost declarate semnalele suma2 și produs2, folosite pentru a salva suma, respectiv produsul.

```
signal a: sfixed(7 downto -8) := "111111010100110";
signal b: sfixed(7 downto -8) := "1111110110011110";
signal suma2: sfixed(7 downto -8) := (others => '0');
signal produs2: sfixed(7 downto -8) := (others => '0');
```

Suma și produsul au fost calculate pe baza liniilor de cod de mai jos:

```
suma2 <=
resize(arg=>(a+b),size_res=>suma2,overflow_style=>fixed_saturate,round_style=>fixed_round);
produs2 <=
resize(arg=>(a*b),size_res=>produs2,overflow_style=>fixed_saturate,round_style=>fixed_round);
```

Suma celor două numere, a și b, este -3.73. Acest număr este reprezentat în fixed point sub forma "1111110001000101". Rezultatul sumei în aplicație este "1111110001000100". Se poate observa că ultimul bit diferă. Astfel, există o eroare, dar aceasta este neglijabilă în acest caz. Produsul celor două numere este "0000001100110110". În aplicație, produsul este "0000001100111000". Se poate observa și în acest caz o diferență între produsul real și produsul oferit de aplicație. În cazul adunării sau înmulțirii a două numere, eroare este infimă. Cu toate acestea, într-o rețea neuronală, unde pentru fiecare output se calculează suma de produse, eroarea crește considerabil. În alegerea modului de reprezentare a numerelor din rețeaua neuronală, trebuie să existe un tradeoff între acuratețe și ușurința de a reprezenta numerele. Reprezentarea în valori întregi ar fi oferit o acuratețe mai slabă decât cea a fixed point. Reprezentarea în floating point ar fi fost mai greu de calculat și ar fi folosit mai multe resurse.



Figura 5.8 – Suma lui a și b

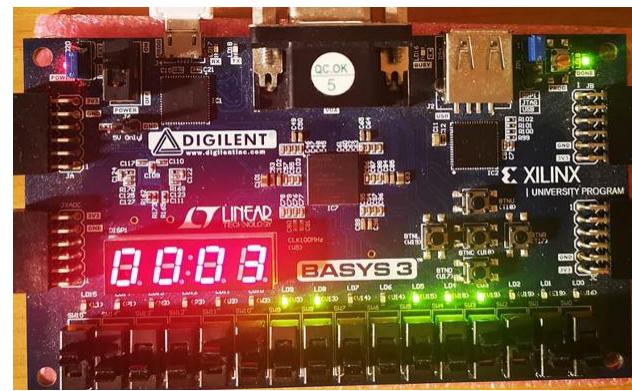


Figura 5.9 – Produsul lui a și b

5.1.4. Memoria BRAM

BlockRAM sau BRAM înseamnă Block Random Access Memory și poate fi folosit pentru a stoca date pe FPGA. În general, BRAM-ul este folosit pentru a stoca date de dimensiuni mari, pentru a lua responsabilitatea de pe LUTs (Look-Up Tables) și FFs (Flip-Flops).

Principalul avantaj al BRAM îl constituie posibilitatea de a stoca informații de dimensiuni mari. Astfel, Basys3 poate stoca 1800 kb, având 100 BRAM-uri cu 18 kb fiecare. Dezavantajul la BRAM, față de implementarea cu RAM folosind LUTs și FFs este dinamismul scăzut pe care îl prezintă. Dimensiunea RAM-ului poate fi modificată prin schimbarea unui parametru, spre deosebire de BRAM, care odată alocat, nu poate fi modificat.

În aplicația pentru lucrarea de licență, input-ul și weights-urile vor fi stocate în câte un BRAM. Input-ul este reprezentat de $13 * 576$ locații. Pentru fiecare cifră din cele 13 ale CNP-ului, se vor atribui 576 locații (input-ul rețelei neuronale artificiale). Fiecare locație va avea câte 16 biți, reprezentând numărul în fixed point representation. Primii 8 biți vor constitui partea întreagă, iar ultimii 8, partea fracțională. BRAM-ul pentru weights-uri va avea 5760 de locații a către 16 biți fiecare. În total, BRAM-urile vor stoca 13.248 de valori, fiecare cu 16 biți. Din acest motiv, necesitatea folosirii BRAM-urilor a fost vitală. De asemenea, prin folosirea BRAM-urilor, LUTs și FFs vor putea fi folosite cu predilecție pentru implementarea calculelor și algoritmilor, fără a fi necesară alocarea pentru memorii RAM sau ROM.

A fost creată o aplicație de test pentru folosirea memoriei BRAM. Memoria a fost creată în configurația single port. Intrările memoriei sunt adresa pe 13 biți, semnalul de clock, semnalul de input pe 16 biți, semnalul de enable și semnalul de write-enable. Ieșirea memoriei este semnalul de output pe 16 biți.

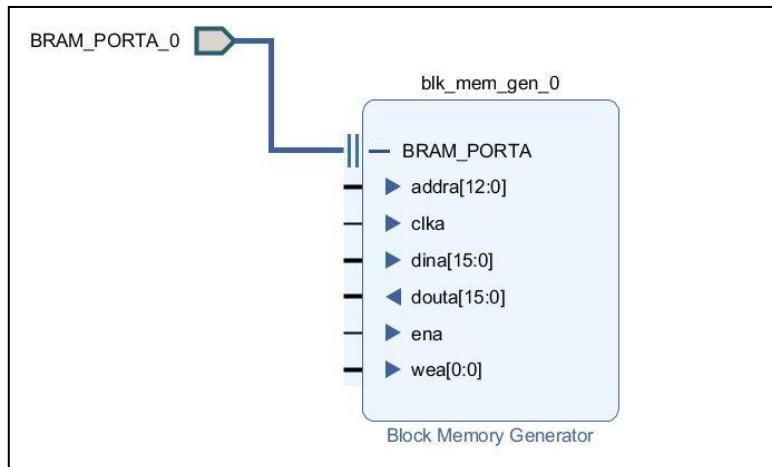


Figura 5.10 – Memoria BRAM din Block Design

Valoarea de 16 biți de pe intrare va fi stocată la adresa memoriei BRAM, dacă write-enable este ‘1’. Semnalul enable poate fi automat pus pe ‘1’. De asemenea există opțiunea alegării unei memorii BRAM fără semnal de enable. Pe ieșire se afișează conținutul locației de la adresa dată de semnalul pe 13 biți, address.

Aplicația de test va scrie la adresa determinată de semnalul address când se va apăsa butonul din centru. Dacă sw(15) este ‘1’ și se apasă butonul de sus, un semnal de counter care coincide ca mărime cu semnalul de adresă se va incrementa. Adresa va prelua automat valoarea semnalului de count. Dacă sw(15) este ‘0’, valoarea counter-ului, respectiv al adresei

determinate de acesta se va decrementa. Dacă se apasă pe butonul din stânga, led-urile vor primi ieșirile determinate de valoarea adresei.

Pentru a demonstra funcționarea memoriei BRAM, vom folosi următorul use-case: switchul 15 al plăcii va fi pus pe ‘1’, pentru a incrementa counter-ul. Valoarea inițială a counter-ului este 0. Prin apăsarea butonului de sus, cu switchul 15 pe ‘1’, valoarea va deveni 1. Se va apăsa pe butonul din stânga pentru a afișa valoarea de la adresa 1. Aceasta ar trebui initial să fie 0. La adresa 1 vom salva valoarea “1111”, folosind switch-urile de la 14 la 0. Odată aleasă valoarea “1111”, se va apăsa butonul din centru pentru a activa write-enable și a scrie la adresa 1, valoarea “1111”. Pentru a afișa valoarea de la adresa 1 pe led-uri, se va apăsa butonul din stânga.

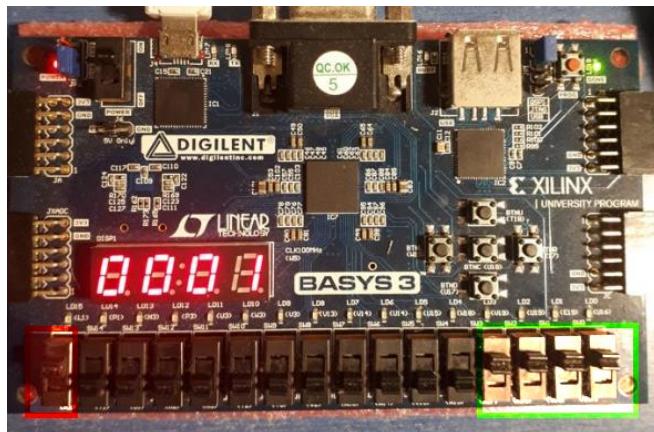


Figura 5.11 – Salvarea bițiilor “1111” la adresa 1

În imaginea din Figura 5.11, switch-ul 15 încadrat cu roșu este pus pe ‘1’. Astfel se incrementează adresa, prin apăsarea butonului de sus. De asemenea, biții de la switch-ul 3 până la switch-ul 0 au fost puși pe ‘1’. Prin apăsarea butonului din centru, write-enable va fi activ, iar valoarea “1111” va fi salvată la adresa 1. În Figura 5.12, se poate observa afișarea pe led-uri a conținutului de la adresa 1.

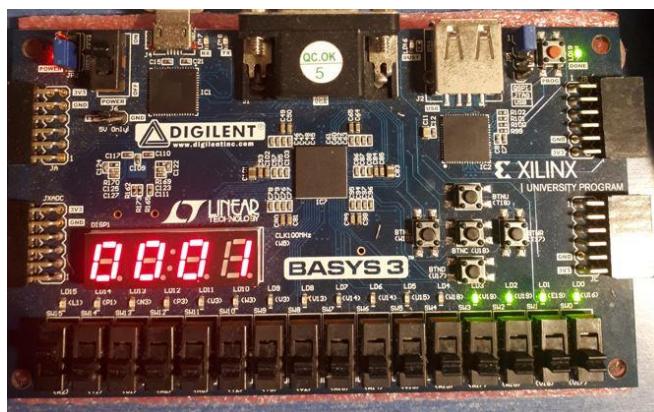


Figura 5.12 – Afisare conținut adresa 1

5.1.5. Primul ANN

Prima aplicație realizată pentru lucrarea de cercetare transmite o imagine de 28X28 a unei cifre, iar rețeaua neuronală artificială detectează cifra din imagine. Pentru a crea această aplicație, nu a fost necesară folosirea memoriei BRAM, întrucât protocolul de transmisie a datelor a fost conceput în aşa fel încât să nu fie necesară memoria adițională.

Rețeaua neuronală conține un input layer de 784 de neuroni, fiecare reprezentând câte un pixel din imaginea transmisă, și un output layer, cu 10 neuroni, câte unul pentru fiecare rezultat posibil, de la 0 la 9. Valorile de input și weights-urile sunt reprezentate folosind fixed point representation, fiecare valoare având 16 biți, primii 8 pentru partea întreagă și ultimii 8 pentru partea frațională. A fost creat tipul array de sfixed(7 downto -8). Folosind acest tip, au fost create array-uri de pixeli, weights, biases și output. Rețeaua neuronală are 784 de input-uri, reprezentând pixelii imaginii, 7840 de weights-uri, salvate, pe rând, de 10 ori câte 784 pentru fiecare output, 10 bias-uri și 10 output-uri.

```
type FixedPointArray is ARRAY (NATURAL RANGE <>) of sfixed(7 downto -8);

signal pixels : FixedPointArray (0 to 783) := (others => X"0000);
signal weights : FixedPointArray (0 to 783) := (others => X"0000);
signal biases : FixedPointArray (0 to 9) := (others => X"0000);
signal outputt : FixedPointArray (0 to 9) := (others => X"0000);
```

Declararea principalelor semnale

A fost implementat protocolul UART pentru receptia și transmisia datelor. Astfel, fișierele cu pixelii, weights-urile și bias-urile sunt transmise de la calculator la placă. De asemenea, la final clasificării, rezultatul este trimis înapoi la calculator. În implementarea rețelei neuronale artificiale, a fost folosit un automat cu stări finite. FSM-ul are 6 stări: idle, receive, show, compute, verify, transmit.

Starea idle are rolul de reinicializa principalele semnale: contoarele pentru pixeli, weights-uri, bias-uri și output-uri. Tranziția din starea idle în starea receive se face atunci când semnalul rxRdy este ‘1’, semn că datele sunt trimise de la calculator la placă.

Starea receive are rolul de a popula array-urile de pixeli, weights și bias. Datele vin de la calculator la placă sub forma de coduri ASCII. De exemplu, este transmis de la calculator la placă șirul “1111000011110000 0000111100001111”. Se vor citi caracterele, rând pe rând. Primul caracter este ‘1’, iar placa va citi codul ASCII al acestuia, adică valoarea zecimală 49. Codul ASCII este decrementat cu 48. Astfel, dacă a fost citit ‘0’, adică 48 în codul ASCII, se va salva valoarea $48 - 48$, adică 0. Dacă a fost citit ‘1’, adică 49 în codul ASCII, se va salva valoarea $49 - 48$, adică 1. Prima valoarea va fi salvată la poziția 7 a unui semnal sfixed(7 downto -8). Pentru următorul caracter, poziția va fi decrementată, până când se ajunge la ultimul caracter, când poziția va fi -8. După ce se va citi ultimul caracter al primului număr, placa va citi spațiul dintre numere. Spațiul are codul ASCII 32. În acest caz, valoarea pe 16 biți convertită va fi stocată în array-ul de pixeli, weights sau bias, în funcție de valoarea contorului de fișiere. Dacă contorul de fișiere este 0, se va salva în array-ul de pixeli. Dacă contorul de fișiere este 1, se va salva în array-ul de bias-uri, iar dacă valoarea contorului este 2 sau mai mare, se va salva în array-ul de weights. Dacă au fost salvate 784 de valori în cazul în care contorul de fișiere este 0, sau 10 valori pentru contorul de fișiere 1, sau 784 de valori pentru contorul de fișiere 2 sau mai mare, se va trece în starea următoare.

Starea show are rolul de a incrementa contorul de fișiere. Inițial, contorul de fișiere este 0. După ce se încheie transmisia pixelilor, conversia și stocarea în array-ul corespunzător, contorul de fișiere va fi incrementat, pentru a semnala că urmează să fie salvate bias-urile. La final, contorul de fișiere va avea valoarea 12: un fișier pentru pixeli, unul pentru bias și 10 pentru weights-urile corespunzătoare fiecărui output.

Starea verify verifică valoarea contorului de fișiere. Dacă valoarea acestuia este mai mică decât 3, se va tranzitiona în starea idle. Motivul pentru care se întâmplă acest lucru este că nu pot fi calculate output-urile, până nu au fost salvați pixelii, bias-urile și măcar primul set de weights-uri corespunzător primului output. Dacă valoarea este 3, înseamnă că au fost populate array-urile de pixeli, biases și primul set de weights-uri. Dacă valoarea contorului de fișiere este mai mare sau egală cu 3, se va trece în starea compute.

Starea compute are rolul de a crea, pe rând, cele 10 output-uri. Se va trece de 10 ori în starea compute, o dată pentru fiecare output. Astfel, primul output va fi calculat ca suma produselor fiecărui din cei 784 de pixeli cu weight-ul corespunzător primului output. Așadar, primul pixel va fi înmulțit cu primul weight al primului output. Al doilea pixel va fi înmulțit cu al doilea weight al primului output și aşa mai departe, până când se ajunge la ultimul pixel, care va fi înmulțit cu ultimul weight al primului output. Aceste produse vor fi adunate iar la final, va fi adăugat bias-ul. Rezultatul este calcularea primului output. La fel se procedează și pentru celelalte 9 output-uri. Valoarea output-ului calculat este salvată în array-ul de output-uri. Pe măsură ce se calculează output-urile, se verifică valoarea maximă a acestora și se salvează labelul corespunzător. În cazul în care contorul de fișiere este 12, adică au fost calculatate toate output-urile, se va trece în starea transmit, altfel în starea idle.

Starea transmit are rolul de transmite, după ce au fost calculate output-urile, valoarea maximă înapoi la calculator, pentru a fi preluată de aplicația desktop.

Rețeaua neuronală a fost testată folosind o imagine cu cifra 7. Au fost salvați pixelii, bias-urile și weights-urile. Cele 12 fișiere au fost trimise rând pe rând la placă, folosind HTERM și un program de automatizare a proceselor pentru aplicații desktop.

 InputBiasesVHDL.txt	6/1/2021 2:10 PM	Text Document	1 KB
 InputsAllVHDL - Copy.txt	6/22/2021 12:48 PM	Text Document	125 KB
 InputsAllVHDL.txt	8/6/2021 11:38 PM	Text Document	125 KB
 InputsAllVHDL1.txt	6/29/2021 10:30 AM	Text Document	125 KB
 InputTestImage0VHDL.txt	5/19/2021 5:59 PM	Text Document	14 KB
 InputTestImage1VHDL.txt	6/3/2021 12:45 PM	Text Document	14 KB
 InputTestImage2VHDL.txt	6/3/2021 1:17 PM	Text Document	14 KB
 InputTestImage3VHDL.txt	6/3/2021 1:20 PM	Text Document	14 KB
 InputTestImage4VHDL.txt	6/3/2021 1:28 PM	Text Document	14 KB
 InputWeightsVHDL0.txt	5/31/2021 12:00 PM	Text Document	14 KB
 InputWeightsVHDL1.txt	6/1/2021 1:10 AM	Text Document	14 KB
 InputWeightsVHDL2.txt	6/1/2021 1:33 AM	Text Document	14 KB
 InputWeightsVHDL3.txt	6/1/2021 1:39 AM	Text Document	14 KB
 InputWeightsVHDL4.txt	6/1/2021 1:45 AM	Text Document	14 KB
 InputWeightsVHDL5.txt	6/1/2021 1:51 AM	Text Document	14 KB
 InputWeightsVHDL6.txt	6/1/2021 1:57 AM	Text Document	14 KB
 InputWeightsVHDL7.txt	6/1/2021 1:19 AM	Text Document	14 KB
 InputWeightsVHDL8.txt	6/1/2021 2:02 AM	Text Document	14 KB
 InputWeightsVHDL9.txt	6/1/2021 2:07 AM	Text Document	14 KB

Figura 5.13 – Fișierele selectate pentru a fi trimise la placă

Programul permite afişarea valorilor semnalelor importante pe led-urile plăcii. Astfel, a fost afişat label-ul corespunzător valorii maxime din cele 10 output-uri de la finalul clasificării. Conform imaginii din Figura 5.14, label-ul corespunzător este 7, fiind afişat în binar.

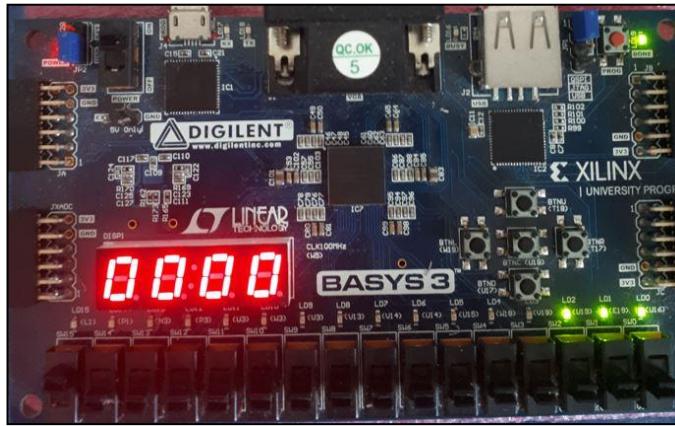


Figura 5.14 – Rezultatul clasificării afişat în binar pe led-uri

Cu toate că rețeaua neuronală a clasificat corect imaginea, iar clasamentul output-urilor este cel așteptat, este important de menționat că valoarea output-urilor este aproximativ jumătate din cea care ar trebui să fie. Astfel, pentru output-ul 0, conform imaginii din Figura 5.15, rezultatul este -4.75, față de -10.40 căt ar trebui să fie. Pentru output-ul 1, rezultatul este -11.80 față de -23.06 căt ar trebui să fie. La fel este și în cazul celorlalte output-uri, fapt relevat de Tabelul 1. Output-ul din VHDL este jumătate față de căt ar trebui să fie. Acest lucru este datorat pierderilor înregistrate cu conversia numerelor în fixed point, adunarea și înmulțirea lor făcând să fie cu atât mai pronunțată ideea de loss.

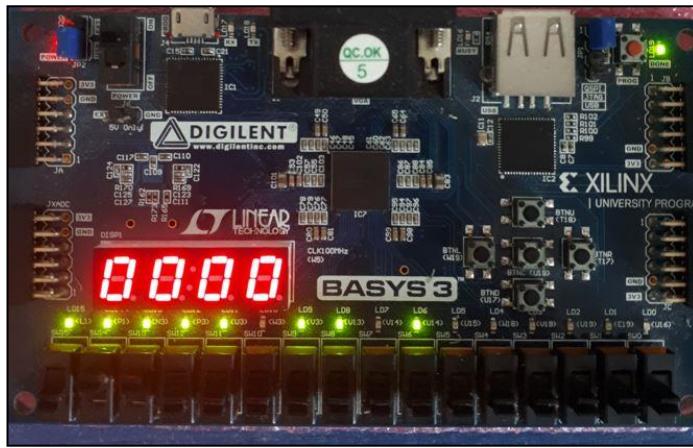


Figura 5.15 – Output-ul 0 al rețelei neuronale

Mai jos este tabelul în care au fost colectate rezultatele obținute respectiv cele așteptate ale output-urilor rețelei neuronale artificiale, pentru imaginea cu cifra 7. Se poate observa că valoarea maximă este cea a output-ului 7, fapt ce indică funcționarea corectă a rețelei neuronale. Problematică, în schimb, este diferența, aproape dublă, dintre rezultatele așteptate și cele obținute.

Număr output	Rezultatul obținut	Rezultatul așteptat
0	-4.75	-10.40
1	-11.80	-23.06
2	-4.44	-9.69
3	-2.01	-4.06
4	-6.75	-13.80
5	-5.25	-10.28
6	-10.6	-20.6
7	0.37	0.63
8	-4.35	-8.59
9	-3.8	-6.71

Tabel 5.1 – Rezultatele obținute și așteptate pentru fiecare output

5.1.6. ANN Final

Prima aplicație avea ca scop clasificarea unei imagini cu o singură cifră, folosind o rețea neuronală artificială implementată în VHDL. A doua aplicație, realizată ulterior, are ca scop localizarea cifrelor din codul numeric personal de pe o imagine cu buletinul, și clasificarea lor. Cifrele din CNP, cele 13, sunt clasificate folosind o rețea neuronală artificială implementată în VHDL. Structura rețelei neuronale artificiale din a doua aplicație este asemănătoare cu cea din prima aplicație. Există un input layer și un output layer, fără hidden layere. Aplicația anterioară avea 784 de neuroni ca input. Fiecare neuron a corespuns cu câte un pixel din imaginea de 28X28. De această dată, input-ul rețelei neuronale artificiale din VHDL are doar 576 de neuroni. Input layer-ul rețelei neuronale artificiale este, de această dată, output-ul determinat de layerele de conoluție, respectiv max-pooling ale rețelei neuronale convoluționale implementată în Python. Fiecare imagine, de 28X28, din cele 13 imagini ale CNP-ului, va trece prin layerele de conoluție și max-pooling ale rețelei neuronale convoluționale din Python, iar ieșirea de după ultimul layer de max-pooling, iar ieșirea de 576 de valori va fi adăugată, rând pe rând, într-un fișier text. Așadar, vor fi adăugate 13X576 valori în fișierul cu input-uri, care urmează să fie trimis la placa FPGA. Rețeaua neuronală convoluțională are, după layerele de conoluție și max-pooling, o structură identică cu rețeaua neuronală artificială din VHDL: 576 de intrări și 10 ieșiri, fără layere ascunse. Modelul acestei rețele va conține 576*10 weights-uri, care vor fi salvate în fișierul text corespunzător weights-urilor. Bias-urile nu au mai fost trimise la placă, pentru că nu influențează rezultatul final, iar trimitera lor necesită un timp de procesare a imaginii mai îndelungat.

Fișierele de input și weights sunt trimise la placă, folosind protocolul UART. A fost implementată atât receptia datelor de la calculator, cât și transmiterea datelor la calculator. Valorile din fișierul de input au fost salvate într-o memorie BRAM de input-uri, având adresa pe 13 biți. Este necesară stocarea a 7488 de valori, fiecare valoare având 16 biți. Valorile din fișierul de weights au fost salvate într-o altă memorie BRAM, având adresa tot pe 13 biți. Este necesară stocarea a 5760 de valori, fiecare având, la fel, 16 biți. Este folosită reprezentarea numerelor în fixed point. Folosirea memoriei BRAM prezintă anumite avantaje în aplicația de față a rețelei neuronale artificiale, față de cea anterioară. În primul rând, timpul de transmitere a datelor este redus. Este nevoie de transmiterea a două fișiere, nu 12 ca data trecută. Un alt avantaj este ușurința protocolului de transmitere a datelor de la calculator la placă. Aplicația

anterioară avea un sistem complicat de transmitere a fișierelor. Întâi erau transmiși pixelii și biasurile, apoi, rând pe rând, weights-urile corespunzătoare fiecărui output. De data aceasta, însă, au fost trimise doar 2 fișiere, într-o ordine aleatorie. Astfel, nu a fost necesară crearea unui protocol complex de transmisie a datelor de la calculator la placă. Principalul dezavantaj este că FSM-ul care stă la baza implementării rețelei neuronale artificiale a devenit mai complex, având 20 de stări, față de cele 6 ale celui anterior. Această complexitate a provenit și din faptul că de această dată, FSM-ul trebuie să clasifice 13 cifre, față de una singură în iterația precedentă. Memoriile BRAM, atât pentru inputs cât și pentru weights, au ca intrări adresele pe 13 biți, intrarea pe 16 biți, semnal de clock și write-enable, iar ieșirea este tot pe 16 biți. Comportamentul BRAM a fost prezentat în sub-capitolele anterioare. În Figura 5.16 se poate observa structura BRAM-ului pentru input-uri. Pentru memoria BRAM a weights-urilor, a fost necesară modificarea parametrilor de la Write Depth și Read Depth, din 7488, în 5760.

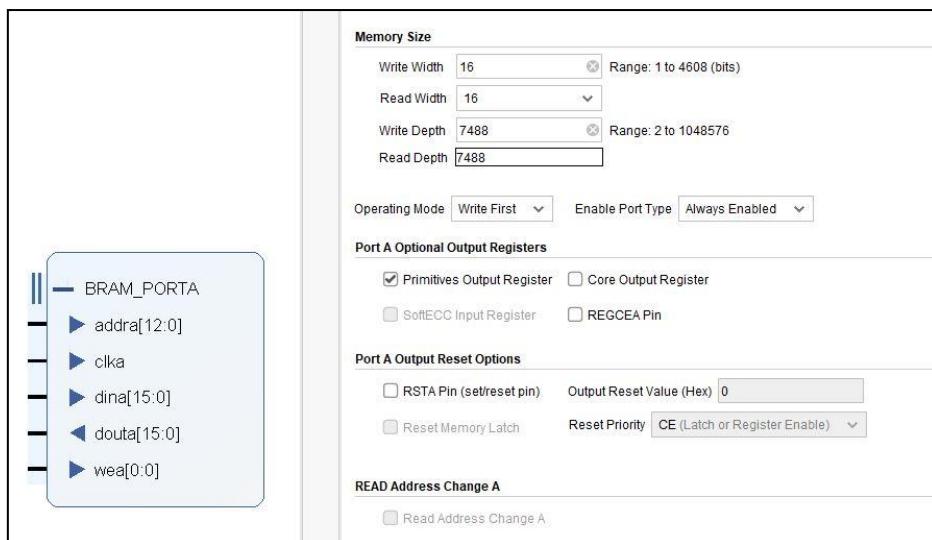


Figura 5.16 – BRAM pentru inputs

Rețeaua neuronală artificială a fost implementată, și de această dată, folosind un FSM. De această dată, însă, FSM-ul are 20 de stări, față de cele 6 ale celui anterior. Algoritmul a fost modificat. Este necesară procesarea a 13 cifre, față de una singură în aplicația anterioară. Fiecare stare a fost implementată folosind proceduri. Pentru a folosi proceduri, au fost create librării. În implementarea de față, au fost grupate procedurile în librării care, oarecum, se ocupă de sub-module ale aplicației. Practic, procedurile care acționează în același sens, au fost grupate în aceeași librărie. De asemenea, a fost creată o librărie în care au fost stocate structurile de date ale aplicației. Fișierul 'types_pack.vhd' conține declarări ale principalelor structuri de date folosite în cadrul proiectului. Entitatele care folosesc aceste tipuri de date (aproape toate) conțin în antet 'use work.types_pack.all'. În cele ce urmează, vor fi prezentate procedurile corespunzătoare celor 20 de stări ale rețelei neuronale artificiale.

Procedura 'idleState' inițializează cu 0 variabilele countValuesI (folosită pentru a număra input-urile: $576 * 13 = 7488$), countValuesW (folosită pentru a număra weights-urile: $576 * 10 = 5760$) și trece în starea receive dacă este semnalată primirea datelor pe linia RX (rxRdy devine '1').

Procedura 'receiveState' are rolul de a salva valorile în memoriile adiționale de input (inputArray) sau de weights-uri (weightsArray). Input-urile și weights-urile vin sub forma sirurilor de caractere (ex. "100010000010001 1010101010101010"). Scopul acestei stări este de a prelua fiecare byte (0 sau 1) și de a-l stoca într-un vector. La citirea caracterului spațiu, se va salva vectorul în memoria de input/weights și se va trece la formarea următorului vector. Dacă variabila countFiles este 0, se va salva în memoria de input-uri, iar dacă countFiles este 1, se va salva în memoria de weights-uri. Dacă rxRdy este '1' (s-a primit un byte pe linia RX), atunci se verifică valoarea acelui byte: byte-ul poate fi o cifră (0 sau 1) (cod ascii 48 sau 49) sau spațiu (cod ascii 32). Dacă este cifră, se va salva la poziția corespunzătoare din vectorul aflat în proces de formare. Pozițiile sunt de la 15 la 0. Vectorul se va salva la adresa determinată de variabilele countValuesI, respectiv countValuesW. Scrierea în memoriile de input-uri și weights-uri se face când weInputs/weWeights (semnal write enable) este '1'.

Procedura 'showState' este apelată atunci când se încheie procesul de transmitere a input-urilor (countValuesI egal cu 7488) sau al weights-urilor (countValuesW egal cu 5760). Dacă countFiles este 0, numberInputs va primi valoarea lui countValuesI, iar dacă countFiles este 1, numberWeights va primi valoarea lui countValuesW. Scopul acestei proceduri este de a salva numărul total de input-uri și numărul total de weights-uri în variabilele numberInputs și numberWeights, pentru a verifica dacă au fost primite toate datele necesare.

Procedura 'verifyFilesState' are rolul de a comuta automatul de stări către starea verifyDigit, în cazul în care memoriile de input-uri și weights-uri au fost populate (countFiles = 2). Dacă countFiles este 0 sau 1, înseamnă că memoriile nu au fost populate, astfel încât se tranzitionează spre starea idle, care așteaptă primirea datelor.

Procedura 'verifyDigitState' verifică valoarea variabilei counter1, folosită pentru a marca indexul cifrei din CNP care este introdusă în rețea neuronală artificială. De exemplu, pentru CNP (2760429264382), dacă cifra 7 este cea introdusă în rețea neuronală artificială, valoarea counter1 va fi 1. Dacă counter1 are valoarea 13, înseamnă că toate cifrele au fost introduse în rețea neuronală artificială, iar automatul va comuta în ultima stare (finală). Altfel, se va comuta în starea 'populateInputArray'.

Procedura 'populateInputArrayState' preia vectorul de input de la adresa addressInputs și îl salvează în valInput. Adresa de input este determinată de valoarea variabilei count. Se face tranziția către starea 'computeFixedPoint'.

Procedura 'computeFixedPointState' are rolul de a converti valoarea din vectorul valInput, care este std_logic_vector(15 downto 0), într-un vector de tipul fixed point (7 downto -8). Poziția 15 din vectorul std_logic_vector corespunde poziției 7 din fixed point vector, iar aceasta se decrementează, până când se ajunge la poziția 0 în std_logic_vector, respectiv poziția -8 din fixed point vector. În vectorul fixed point, pozițiile de la 7 la 0 sunt alocate părții întregi, iar pozițiile -1 până la -8 sunt alocate părții fractionale. Astfel, valoarea 1.5 va fi reprezentată ca 0000 0001 1000 0000, unde 0000 0001 este partea întreagă, și 1000 0000 este partea fractională. În momentul în care vectorul fixed point este populat, se va trece la starea saveValue.

Procedura 'saveValueState' are menirea de a salva vectorul fixed point în array-ul de fixed point values, numit inputArray, la poziția determinată de valoarea cntI.

Procedura 'incrementCountState' are rolul de a tranzitiona în starea populateInputArray în cazul în care array-ul inputArray nu a fost pe deplin populat. De exemplu, pentru cifra 2 (prima din CNP), se vor lua primele 576 de input-uri din memoria de inputs și vor fi salvate în inputArray. Odată încheiat procesul pentru cifra 2 (s-au determinat ponderile celor 10 output-

uri), se va trece la următoarea cifră din CNP, 7, și se va repeta procesul pentru cele 576 de input-uri ale cifrei 7.

Procedura 'verifyWeightIndexState' verifică valoarea variabilei counter2, și în funcție de aceasta, comută automatul de stări către starea corespunzătoare. Variabila counter2 este folosită pentru a urmări care este valoarea weights-urilor din weightsArray (576 weights-uri). Pentru prima cifră din CNP (cifra 2), se vor lua input-urile aferente și vor fi stocate în inputsArray. Apoi, pentru fiecare valoarea a counter2, (de la 0 la 9), se vor stoca în weightsArray, cele 576 de weights-uri corespunzătoare output-ului aferent counter2. De exemplu, pentru a crea output-ul 0 al cifrei 2, se vor stoca în weightsArray weights-urile corespunzătoare output 0. Ulterior se va face suma produselor pentru fiecare index al inputsArray și weightsArray, iar suma respectivă va reprezenta output 0. Counter2 se va incrementa, și se va relua procesul pentru output-ul 1, păstrând același inputArray (afferent cifrei 2 din CNP). Procesul pentru prima cifră din CNP se va încheia când counter2 va avea valoarea 10 (astfel cele 10 output-uri vor fi deja construite iar valoarea maximă a sumelor va reprezenta cifra corespunzătoare din CNP).

Procedura 'populateWeightArrayState' are rolul de a prelua valoarea weights-urilor din memoria adițională de weights-uri de la adresa determinată de valoarea variabilei countW.

Procedura 'computeFixedPointWState', în oglindă cu procedura computeFixedPointState, are rolul de a converti valoarea din vectorul valInputW care este std_logic_vector(15 downto 0), într-un vector de tipul fixed point (7 downto -8). Poziția 15 din vectorul std_logic_vector corespunde poziției 7 din fixed point vector, iar aceasta se decrementează, până când se ajunge la poziția 0 în std_logic_vector, respectiv pozitia -8 din fixed point vector. În vectorul fixed point, pozițiile de la 7 la 0 sunt alocate părții întregi, iar pozițiile -1 pana la -8 sunt alocate părții fractionale. Astfel, valoarea 1.5 va fi reprezentată ca 0000 0001 1000 0000, unde 0000 0001 este partea întreagă, și 1000 0000 este partea fractională. În momentul în care vectorul fixed point este populat, se va trece la starea saveValueW.

Procedura 'saveValueWState' are menirea de a salva vectorul fixed point în array-ul de fixed point values, numit weightsArray, la poziția determinată de valoarea cntW.

Procedura 'incrementCountWState' are rolul de a tranzitiona în starea populateWeightArray în cazul în care array-ul weightsArray nu a fost pe deplin populat. De exemplu, pentru output-ul 0 al cifrei 2 (prima din CNP), se vor lua primele 576 de weights-uri din memoria de weights și vor fi salvate în weightsArray. Odată încheiat procesul pentru output-ul 0 al cifrei 2, se va trece la output-ul 1 (următoarele 576 weights-uri), până se ajunge la ultimul output (output 9).

Procedura 'computeState' are rolul de a tranzitiona către starea computeProduct dacă nu au fost realizate înmulțirile input-urilor cu weights-urile corespunzătoare, sau de a salva output-ul corespunzător valorii variabilei counter2, în cazul în care s-a realizat suma tuturor celor 576 de produse. Dacă suma este mai mare decât valoarea maximă înregistrată de output-urile uneia dintre cifrele de input, atunci acea valoare va deveni maxValue. De exemplu, pentru prima cifră din CNP (cifra 2), al treilea output (output 2) va avea valoarea cea mai mare. Variabila labelMax are scopul de a salva indexul valorii maxime.

Procedura 'saveAllOutputsValueState' are menirea de a salva output-ul pentru fiecare cifră în array-ul allOutputs (FixedPointArray). Astfel, allOutputs va avea 130 de output-uri (13 cifre în CNP, fiecare cu câte 10 output-uri, de la 0 la 9).

Procedura 'computeProductState' are rolul de a înmulți input-ul de la indexul countOutput cu weight-ul de la indexul countOutput, iar rezultatul va fi stocat în variabila produs.

Procedura 'computeSumState' are rolul de a aduna la suma corespunzătoare unui output, produsul corespunzător înmulțirii dintre input-ul de la indexul countOutput și weights-ului de la indexul countOutput.

Procedura 'incrementCounterWeightState' incrementă variabila counter2 (care corespunde index-ului output-ului care se vrea a fi calculat). De exemplu, dacă pentru cifra 2 am calculat output 0, atunci counter2 se va incrementa, pentru a putea calcula output 1.

Imaginile sunt clasificate, iar label-ul corespunzător fiecărei imagini, este salvat în array-ul maxValues. Array-ul maxValues conține 13 valori, fiecare fiind reprezentată pe 4 biți. Inițial, pe SSD sunt afișate primele patru valori din maxValues, folosind un semnal left. Semnalul left este inițializat cu 0. De asemenea, pe SSD sunt afișate și valorile de la indexul left + 1, left + 2, left + 3. Afișarea rezultatului se face folosind un barrell shifter. A fost folosit un numărător, numit countBarrel, care se va incrementa la fiecare ciclu de ceas. Când ajunge la valoarea 200000000, care corespunde a două secunde, countBarrel se reinicializează, iar left se incrementă cu 1. Astfel, vor fi afișate valorile de la adresele 1,2,3 și 4 ale maxValues, față de adresele 0,1,2 și 3 înainte de incrementarea semnalului left. Acest procedeu se repeat la fiecare două secunde, până când left ajunge să aibă valoarea 9, caz în care sunt afișate ultimele 4 valori ale maxValues, iar left este reinicializat cu 0.

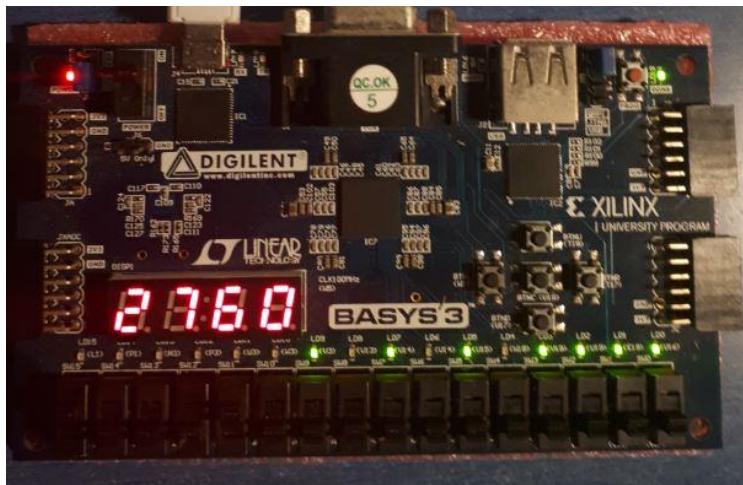


Figura 5.17 – Afisare pe SSD a primelor 4 cifre ale CNP-ului

5.2. Python

5.2.1. Rețeaua neuronală artificială

Reteaua neuronală artificială are rolul de a clasifica imagini cu cifre scrise de mâna. Imaginile au dimensiunea de 28X28 pixeli, sunt grayscale, și fac parte din dataset-ul MNIST. De asemenea, modelul determinat de antrenarea rețelei neuronale va fi folosit salvat într-un fișier și transmis la placa de dezvoltare FPGA pentru a putea fi folosit în rețeaua neuronală artificială din VHDL.

Reteaua neuronală a fost implementată folosind TensorFlow. Imaginile cu cifrele scrise de mâna au fost încărcate din dataset-ul de la MNIST, și divizate în imagini de antrenare și imagini de testare. De asemenea, au fost încărcate și labelurile corespunzătoare fiecărei imagini, atât pentru cele de antrenare, cât și pentru testare. Dimensiunea dataset-ului de antrenare este de 60.000 de imagini, fiecare imagine având dimensiunea de 28X28 pixeli. Imaginile de

antrenament sunt, inițial, bidimensionale. Dimensiunea dataset-ului de testare este de 10.000 de imagini, fiecare imagine având dimensiunea de 28X28 pixeli. Imaginele de testare sunt, de asemenea, inițial, bidimensionale.

Imaginea este grayscale și bidimensională. Valorile sunt numere întregi, cuprinse între 0 și 255. Aceste valori vor fi normalizate, însă rețeaua neuronală artificială nu poate lucra cu valori întregi, ci doar cu valori în virgulă mobilă. Valorile set-ului de antrenare, respectiv al celui de testare au fost divizate cu 255, pentru a avea valori cuprinse între 0 și 1. Aceasta reprezintă un avantaj pentru rețeaua neuronală artificială, care, în teorie, ar putea lucra cu numere până la 255, dar antrenarea ar fi ineficientă, și ar necesita prea multe resurse. Imaginele din setul de antrenare și de testare sunt bidimensionale. Fiecare imagine are 28 de rânduri, fiecare rând având 28 de valori. Acest lucru va fi modificat, însă input-ul rețelei neuronale artificiale este unidimensional. Astfel, imaginile vor fi convertite din array-uri 2D în array-uri 1D. Setul de antrenare va avea în continuare 60.000 de imagini, dar acestea vor fi reprezentate în array-uri unidimensionale, cu 784 de valori.

```
In [26]: model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')
])

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

model.fit(X_train_flattened,y_train,epochs = 5)

Epoch 1/5
1875/1875 [=====] - 1s 566us/step - loss: 0.4860 - accuracy: 0.8779
Epoch 2/5
1875/1875 [=====] - 1s 521us/step - loss: 0.3062 - accuracy: 0.9157
Epoch 3/5
1875/1875 [=====] - 1s 511us/step - loss: 0.2855 - accuracy: 0.9211
Epoch 4/5
1875/1875 [=====] - 1s 515us/step - loss: 0.2739 - accuracy: 0.9244
Epoch 5/5
1875/1875 [=====] - 1s 514us/step - loss: 0.2677 - accuracy: 0.9258

Out[26]: <tensorflow.python.keras.callbacks.History at 0x1f807e4bba0>
```

Figura 5.18 – Crearea modelului și compilarea sa

Crearea modelului a fost realizată folosind keras. În definirea modelului, proprietatea Dense denotă faptul că toți neuronii dintr-un layer sunt conectați cu toți neuronii din următorul layer. Input-ul are 784 de valori, reprezentând cele 784 de pixeli ai unei imagini. Output-ul are 10 valori, câte una pentru fiecare cifră existentă, de la 0 la 9. Funcția de activare este sigmoid, făcând ca output-ul să aibă valori între 0 și 1. Compilarea modelului are ca argumente optimizer, loss și metrics. Sparse_categorical_crossentropy se referă la faptul că output-urile, y, au valori întregi. Metrica este acuratețea, însă scopul principal este de a face rețeaua neuronală să aibă o acuratețe cât mai mare. Folosind funcția fit a modelului, se va face antrenarea, având ca parametrii setul de antrenare convertit într-un array unidimensional, output-ul corespunzător antrenării, și numărul de epoci.

Se evaluatează acuratețea pe dataset-ul de test, folosind funcția evaluate. Acuratețea modelului pe dataset-ul de test este de 92%. Folosind funcția predict a modelului, având ca parametru imaginile de test 1D, a fost creată y_predicted, o variabilă care conține, pentru fiecare imagine, predicția corespunzătoare a modelului. Array-ul de predicție pentru una dintre

imagini conține 10 valori, câte o probabilitate pentru fiecare cifră. Indexul valorii celei mai mari din array, corespunde cu cifra din imagine. Din acest motiv se calculează argmax pe `y_predicted[0]`. Valoarea rezultată este 7, care coincide cu output-ul primei imagini de test.

Confusion matrix-ul comunică de câte ori au fost prezise corect valorile. De exemplu, 0 a fost presuscoret de 962 de ori. Valoarea 1 a fost presuscoret de 1116 ori. Se poate observa că valorile de pe diagonală sunt cele mai mari, întrucât cifrele au fost, în general, prezise corect, rețeaua neuronală având o acuratețe de aproximativ 92%. Confusion matrix relevă și cifrele care pun probleme. De exemplu, cifra 4 a fost presuscoret de 49 de ori ca fiind 9.

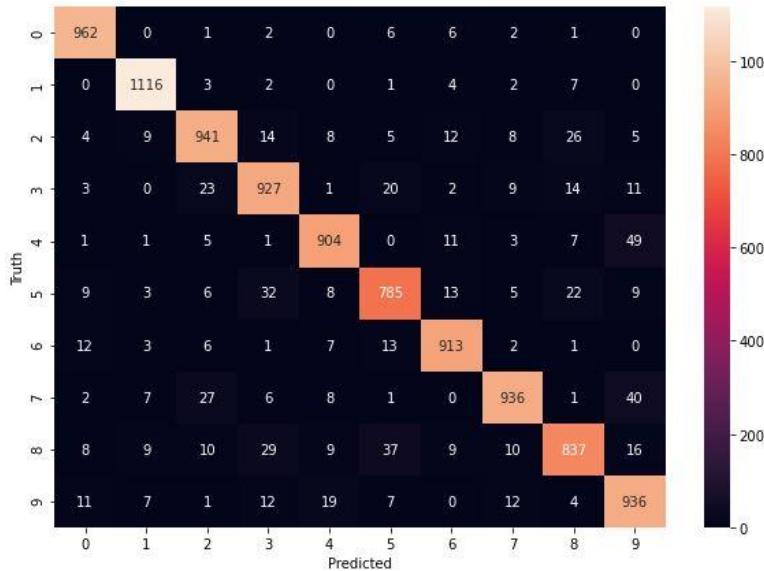


Figura 5.19 – Confusion matrix ANN

Modelul a fost salvat pentru a fi utilizat din nou. Aceasta este modalitatea prin care weights-urile și bias-urile unui model rămân neschimbate. De fiecare dată când se compilează un model, valorile weights-urilor și bias-urilor se modifică. Modelul poate fi încărcat din fișierul în care a fost salvat, și poate fi reutilizat. Valorile weights-urilor și bias-urilor rămân neschimbate.

5.2.2. Rețeaua neuronală convoluțională

Modelul rețelei neuronale convoluționale a fost preluat din [42].

Rețeaua neuronală convoluțională are rolul de a clasifica imagini cu cifre scrise de mână. Imaginile au dimensiunea de 28X28 pixeli, sunt grayscale, și fac parte din dataset-ul MNIST. De asemenea, modelul determinat de antrenarea rețelei neuronale va fi folosit salvat într-un fișier și transmis la placa de dezvoltare FPGA pentru a putea fi folosit în rețeaua neuronală artificială din VHDL. Imaginile cu cifrele scrise de mână au fost încărcate din dataset-ul de la MNIST, și divizate în imagini de antrenare și imagini de testare. De asemenea, au fost încărcate și labelurile corespunzătoare fiecărei imagini, atât pentru cele de antrenare, cât și pentru testare. Dimensiunea dataset-ului de antrenare este de 60.000 de imagini, fiecare imagine având dimensiunea de 28X28 pixeli. Imaginile de antrenament sunt, inițial, bidimensionale. Dimensiunea dataset-ului de testare este de 10.000 de imagini, fiecare imagine având dimensiunea de 28X28 pixeli. Imaginile de testare sunt, de asemenea, inițial, bidimensionale.

Imaginea este grayscale și bidimensională. Valorile sunt numere întregi, cuprinse între 0 și 255. Aceste valori vor fi normalizate, însă rețeaua neuronală artificială nu poate lucra cu valori întregi, ci doar cu valori în virgulă mobilă. Valorile set-ului de antrenare, respectiv al celui de testare au fost divizate cu 255, pentru a avea valori cuprinse între 0 și 1. Aceasta reprezintă un avantaj pentru rețeaua neuronală artificială, care, în teorie, ar putea lucra cu numere până la 255, dar antrenarea ar fi ineficientă, și ar necesita prea multe resurse. Imaginele din setul de antrenare și de testare sunt bidimensionale. Fiecare imagine are 28 de rânduri, fiecare rând având 28 de valori. Acest lucru va fi modificat, însă input-ul rețelei neuronale artificiale este unidimensional. Astfel, imaginile vor fi convertite din array-uri 2D în array-uri 1D.

Dimensiunile setului de antrenare au fost modificate. A fost adăugată o dimensiune, pentru canalul de culoare. Fiind o imagine grayscale, există un singur canal de culoare. În cazul în care dataset-ul ar fi conținut imagini color, atunci X_train ar fi avut 3 canale pentru culoare: (60.000, 28, 28, 3). Același lucru care s-a întâmplat pentru imaginile de antrenare se întâmplă și pentru dataset-ul de testare.

```
In [27]: model = keras.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.Flatten(),
    #layers.Dense(100, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])
```

Figura 5.20 – Crearea modelului rețelei neuronale convoluționale

Modelul rețelei neuronale convoluționale conține 3 layeruri de convoluție și 2 layeruri de max-pooling. Acestea au ca și funcție de activare relu. Funcția de activare relu funcționează astfel: dacă valoarea este mai mică decât 0, atunci valoarea va fi 0, altfel rămâne valoarea originală. Output-ul ultimului layer de convoluție este convertit într-un array unidimensional. Ulterior este o rețea neuronală artificială, complet conectată, care are ca input 576 de valori, iar output-ul 10 valori, corespunzătoare celor 10 cifre. Funcția de activare pentru output este sigmoid, făcând ca output-ul să aibă valori cuprinse între 0 și 1.

Compilarea modelului are ca argumente optimizer, loss și metrics. Sparse_categorical_crossentropy se referă la faptul că output-urile, y, au valori întregi. Metrica este acuratețea, însă scopul principal este de a face rețeaua neuronală să aibă o acuratețe cât mai mare. Folosind funcția fit a modelului, se va face antrenarea, având ca parametrii setul de antrenare convertit într-un array unidimensional, output-ul corespunzător antrenării, și numărul de epoci.

```

Epoch 1/5
1875/1875 [=====] - 19s 10ms/step - loss: 0.1616 - accuracy: 0.9498
Epoch 2/5
1875/1875 [=====] - 20s 11ms/step - loss: 0.0519 - accuracy: 0.9835
Epoch 3/5
1875/1875 [=====] - 21s 11ms/step - loss: 0.0379 - accuracy: 0.9880
Epoch 4/5
1875/1875 [=====] - 21s 11ms/step - loss: 0.0290 - accuracy: 0.9908
Epoch 5/5
1875/1875 [=====] - 21s 11ms/step - loss: 0.0215 - accuracy: 0.9931

```

Figura 5.21 – Compilarea rețelei neuronale convoluționale

Se evaluează acuratețea pe dataset-ul de test, folosind funcția evaluate. Acuratețea modelului pe dataset-ul de test este de 98%. Modelul a fost salvat pentru a fi utilizat din nou. Aceasta este modalitatea prin care weights-urile și bias-urile unui model rămân neschimbate. De fiecare dată când se compilează un model, valorile weights-urilor și bias-urilor se modifică. Modelul poate fi încărcat din fișierul în care a fost salvat, și poate fi reutilizat. Valorile weights-urilor și bias-urilor rămân neschimbate. Structura rețelei neuronale poate fi redată apelând funcția summary. Informațiile relevante de apelul acestei funcții sunt denumirea layer-ului, forma de output și numărul de parametri.

Folosind funcția predict a modelului, având ca parametru imaginile de test 1D, a fost creată y_predicted, o variabilă care conține, pentru fiecare imagine, predicția corespunzătoare a modelului.

Confusion matrix-ul comunică de câte ori au fost prezise corect valorile. Predicția numerelor este mai bună față de cea a rețelei neuronale artificiale, dat fiind faptul că acuratețea a crescut de la 92% la 98% pentru dataset-ul de testare. De exemplu, 0 a fost prezentat corect de 976 de ori, folosind rețeaua neuronală convoluțională, față de 962 cu rețeaua neuronală artificială. Se poate observa că valorile de pe diagonală sunt cele mai mari, întrucât cifrele au fost, în general, prezise corect, rețeaua neuronală având o acuratețe de aproximativ 98%. Confusion matrix relevă și cifrele care pun probleme. De exemplu, cifra 9 a fost prezentată de 12 de ori ca fiind 4.

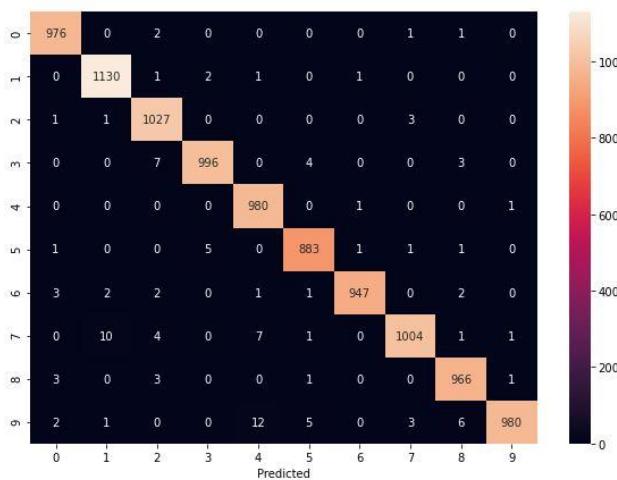


Figura 5.22 – Confusion matrix CNN

5.2.3. Optical Character Recognition

Optical Character Recognition sau OCR este un algoritm folosit pentru a detecta caracterele dintr-o imagine. Folosind OCR, vor fi detectate cifrele care formează codul numeric personal într-o imagine cu un act de identitate. OCR are mai multe implementări. Pentru lucrarea de cercetare de față, a fost aleasă implementarea OCR folosind tesseract.



Figura 5.23 – Imaginea originală cu actul de identitate

Folosind librăria OpenCV, este citită o imaginea cu un act de identitate. Imaginea a fost redimensionată, având ca și parametrii $fx = 2$ și $fy = 2$. Parametrii fx și fy au valori diferite pentru imagini diferite. Valorile vor fi stabilite prin încercări multiple, de la 0.5, fiind incrementat cu câte 0.5, uneori ajungând chiar până la 3.0.

Au fost implementate funcții de preprocesare a imaginii, înainte de a aplica algoritmul OCR. Funcția `get_grayscale` face ca o imagine RGB cu 3 canale de culoare, să devină o imagine grayscale, cu un singur canal de culoare. Funcția `thresholding` preia imaginea grayscale și o binarizează, folosind un `threshold`. Funcția `opening` are rolul de a elimina zgomotele din imagine. Funcția `canny` are rolul de a detecta muchiile.



Figura 5.24 – Imaginile grayscale, thresholding și canny

Folosind funcția `image_to_boxes`, aplicată pe imaginea canny, au fost detectate multe dintre caracterele din imagine. Caracterele au fost introduse într-un array unidimensional.

Algoritmul de mai jos are rolul de a detecta seria din buletin. Una dintre posibilele aplicații ale algoritmului ar fi putut detecta CNP-ul și blura cifrelor acestuia, fiind afișate doar datele care nu sunt atât de compromițătoare, precum seria. Array-ul unidimensional de caractere este parcurs. Dacă se ajunge la un caracter numeric, se marchează începutul unui sir de numere, folosind variabila okLeft, și se salvează indexul poziției de început, în variabila left. Dacă sirul de numere citit aparține seriei, vor fi citite 6 numere, urmate de un caracter care nu este numeric. Se salvează și indexul poziției ultimei cifre care aparține seriei.

Cifrele din serie au fost clasificate și localizate. Pentru fiecare cifră, se preiau coordonatele x,y, width-ul și height-ul. Fiecare cifră va fi încadrată într-o imagine de 18X18. Scopul este crearea unei imagini de 28X28, astfel încât cifra să fie cât mai mult centrată în imagine. Dacă cifra nu este centrată în imagine, clasificarea va fi greu de realizat. În jurul celor 18X18 pixeli cu cifra vor fi introdusi pixeli negri pentru a umple imaginea de 28X28. Cifrele detectate au culoarea negru pe fundal alb. A fost necesară modificarea imaginii într-una cu cifre albe pe fundal negru, de aici scăderea fiecărei valori din imagine cu 255. Cele 6 imagini cu cifrele din serie vor fi salvate pentru a fi utilizate în rețea neuronală convoluțională.

Algoritmul de mai jos are rolul de a detecta codul numeric personal din buletin. Array-ul unidimensional de caractere este parcurs. Dacă se ajunge la un caracter numeric, se marchează începutul unui sir de numere, folosind variabila okLeft, și se salvează indexul poziției de început, în variabila left. Dacă sirul de numere citit aparține CNP-ului, vor fi citite 13 numere, urmate de un caracter care nu este numeric. Se salvează și indexul poziției ultimei cifre care aparține CNP-ului. Cifrele din CNP au fost detectate și localizate. Pentru fiecare cifră, se preiau coordonatele x,y, width-ul și height-ul. Fiecare cifră va fi încadrată într-o imagine de 18X18. Scopul este crearea unei imagini de 28X28, astfel încât cifra să fie cât mai mult centrată în imagine. Dacă cifra nu este centrată în imagine, clasificarea va fi greu de realizat. În jurul celor 18X18 pixeli cu cifra vor fi introdusi pixeli negri pentru a umple imaginea de 28X28. Cifrele detectate au culoarea negru pe fundal alb. A fost necesară modificarea imaginii într-una cu cifre albe pe fundal negru, de aici scăderea fiecărei valori din imagine cu 255. Cele 13 imagini cu cifrele din serie vor fi salvate pentru a fi utilizate în rețea neuronală convoluțională.



Figura 5.25 – Rezultatul aplicării algoritmului OCR

5.2.4. Script licență

Acest script are ca scop preluarea imaginii cu actul de identitate ales, folosind aplicația desktop, pentru a fi procesat, detecția și localizarea cifrelor din CNP, preluarea fiecărei cifre și crearea unui dataset folosit pentru a prezice aceste cifre folosind rețea neuronală conoluțională, salvarea rezultatelor în urma aplicării convoluțiilor și max-pooling pe fiecare dintre cele 13 cifre, convertirea pixelilor din zecimal în fixed point și crearea unui fișier de input care ar urma să fie trimis la placa de dezvoltare.

Script-ul se va rula în întregime, folosind comanda "Run All". Execuția script-ului se va opri pentru a fi selectată imaginea care se dorește a fi procesată din aplicația desktop. Odată selectată imaginea, aceasta va fi salvată în folderul chosen_image cu denumirea buletin.jpg. În momentul în care este salvată, execuția scriptului va continua. În momentul în care este aleasă imaginea cu buletinul, vor fi salvați parametrii corespunzători imaginii în fișierul config.txt. Parametrii imaginii sunt fx și fy (coeficienții cu care se va face resize la imagine pentru a fi detectate cu ușurință cifrele din CNP) și threshold-ul pentru care se face binarizarea imaginilor cu cifrele.

Imaginea este redimensionată, având ca și parametrii fx și fy, preluati din fișierul de configurare. Parametrii fx și fy au valori diferite pentru imagini diferite. Valorile vor fi stabilite prin încercări multiple, de la 0.5, fiind incrementat cu câte 0.5, uneori ajungând chiar până la 3.0. Sunt implementate funcții de preprocesare a imaginii, înainte de a aplica algoritmul OCR. Funcția get_grayscale face ca o imagine RGB cu 3 canale de culoare, să devină o imagine grayscale, cu un singur canal de culoare. Funcția thresholding preia imaginea grayscale și o binarizează, folosind un threshold. Funcția opening are rolul de a elimina zgomotele din imagine. Funcția canny are rolul de a detecta muchiile.

Folosind funcția image_to_boxes, aplicată pe imaginea canny, au fost detectate multe dintre caracterele din imagine. Caracterele au fost introduse într-un array unidimensional. Array-ul unidimensional de caracter este parcurs. Dacă se ajunge la un caracter numeric, se marchează începutul unui sir de numere, folosind variabila okLeft, și se salvează indexul poziției de început, în variabila left. Dacă sirul de numere citit aparține CNP-ului, vor fi citite 13 numere, urmate de un caracter care nu este numeric. Se salvează și indexul poziției ultimei cifre care aparține CNP-ului. Cifrele din CNP au fost detectate și localizate. Pentru fiecare cifră, se preiau coordonatele x,y, width-ul și height-ul. Fiecare cifră va fi încadrată într-o imagine de 18X18. Scopul este crearea unei imagini de 28X28, astfel încât cifra să fie cât mai mult centrată în imagine. Dacă cifra nu este centrată în imagine, clasificarea va fi greu de realizat. În jurul celor 18X18 pixeli cu cifra vor fi introdusi pixeli negri pentru a umple imaginea de 28X28. Cifrele detectate au culoarea negru pe fundal alb. A fost necesară modificarea imaginii într-una cu cifre albe pe fundal negru, de aici scăderea fiecărei valori din imagine cu 255. Cele 13 imagini cu cifrele din serie vor fi salvate pentru a fi utilizate în rețea neuronală conoluțională.



Figura 5.26 – Salvarea primei valori din CNP ca și imagine

Cele 13 imagini cu cifrele CNP-ului sunt citite din folder și adăugate în array-ul test_arr1. Imaginele sunt binarizate, folosind threshold-ul din fișierul de configurare. Fiecare imagine are o altă valoare de threshold. Imaginele au valori cuprinse între 0 și 255. Pentru a fi introduse în rețea neuronală conoluțională, valorile trebuie să fie normalize între 0 și 1. Array-ul test_arr1 va avea 13 imagini cu dimensiunea de 28X28 pixeli normalizați. Se face o redimensionare a array-ului test_arr1, pentru a avea o reprezentare pe canale de culoare, chiar dacă are un singur canal, fiind o imagine grayscale. Această redimensionare este necesară înainte de a realiza predicția cifrelor folosind rețea neuronală conoluțională.

Modelul rețelei neuronale a fost salvat și este încărcat din fișirul 'my_model'. Se va face o predicție a imaginilor din CNP apelând metoda predict, având ca și parametru dataset-ul de imagini cu cele 13 cifre ale CNP-ului. Rezultatele predicției, adică valorile maxime ale y_pred, sunt salvate și scrise în fișierul resulted_cnp.txt. În intermediate_output sunt stocate rezultatele de după aplicarea layer-elor de conoluție și max-pooling pe cele 13 imagini cu cifrele CNP-ului, fiecare având dimensiunea de 576. Fiecare dintre cele 13 va fi, pe rând, o intrare a rețelei neuronale conoluționale din VHDL. Se va folosi algoritmul de conversie a numerelor din virgulă mobilă în fixed point, și vor fi salvate în fișierul InputsAllVHDL.txt.

5.3. Winium

Aplicația de automatizare are ca scop trimitera fișierului de input și a fișierului cu modelul rețelei neuronale artificiale la placa de dezvoltare, folosind aplicația Hterm. În acest scop a fost creat executabilul Winium.exe, care va fi rulat după ce este creat fișierul de input intitulat InputsAllVHDL.txt, în urma executării fișierului python Script_licență. Pentru a rula aplicația de automatizare, este necesară rularea Winium.Desktop.Driver.exe, pentru a rula Windows Desktop Driver pe portul 9999.

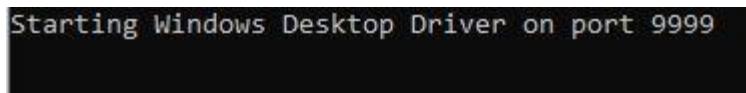


Figura 5.27 – Rularea Windows Desktop Driver

Aplicația va rula hterm.exe. Denumirile fișierelor care urmează a fi trimise la placă au fost salvate într-un array de string-uri. În momentul în care pornește hterm, parametrii de transmitere a datelor sunt citiți din fișierul de configurare. Astfel, baud-rate-ul este setat la 115200, 8 biți de date, 1 bit de stop. Aplicația va apăsa butonul ‘Send file’, pentru a alege fișierul care urmează a fi trimis. Se vor aștepta 3 secunde, datorită posibilelor întârzieri care pot apărea din folosirea concomitentă a mai multor programe. Este inserată denumirea primului fișier, “InputsALLVHDL.txt”, salvat în array-ul de string-uri, se apăsa ‘Open’ și ‘Send’. Aplicația va aștepta aproximativ 12 secunde până când primul fișier va fi trimis la placă. Același procedeu va avea loc și pentru cel de-al doilea fișier, “WeightsALLVHDL.txt”, care conține modelul rețelei neuronale.

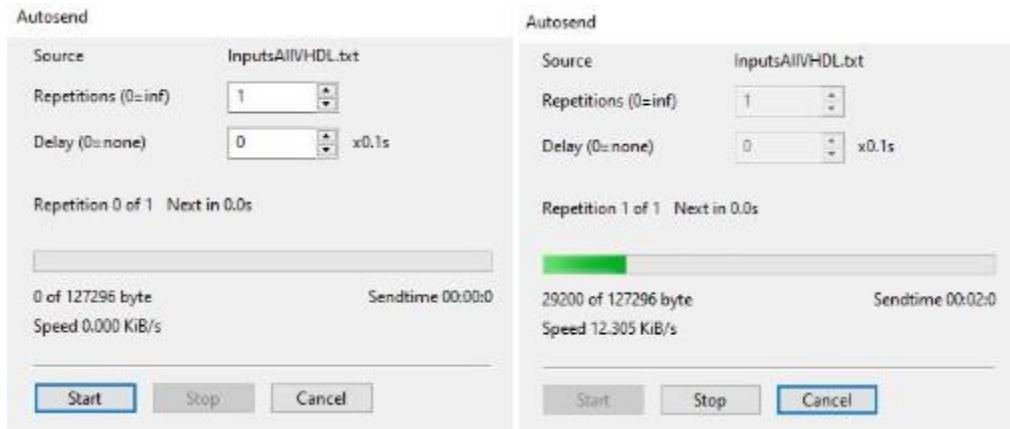


Figura 5.28 – Trimiterea fișierului “InputsALLVHDL.txt” la placa de dezvoltare

Butoanele au fost identificate folosind tool-ul Inspect.exe. Aplicația este rulată și apare o fereastră cu numeroși parametrii, folosiți pentru a identifica unic un buton. Cel mai ușor de folosit parametru este denumirea. Sunt cazuri, însă, când mai multe butoane dintr-o fereastră au aceeași denumire. Atunci este de preferat să se folosească id-ul. Principalul dezavantaj al aplicației de automatizare este că aceasta trebuie să ruleze în foreground. Astfel este afișat tot procesul de transmitere a fișierelor. Ar fi fost de preferat ca acțiunile aplicației să fie în background. Acest lucru ar fi fost posibil dacă ar fi fost folosit încă un monitor, pe care să fie afișat procesul de transmitere al fișierelor, sau dacă ar fi fost instalată o mașină virtuală.

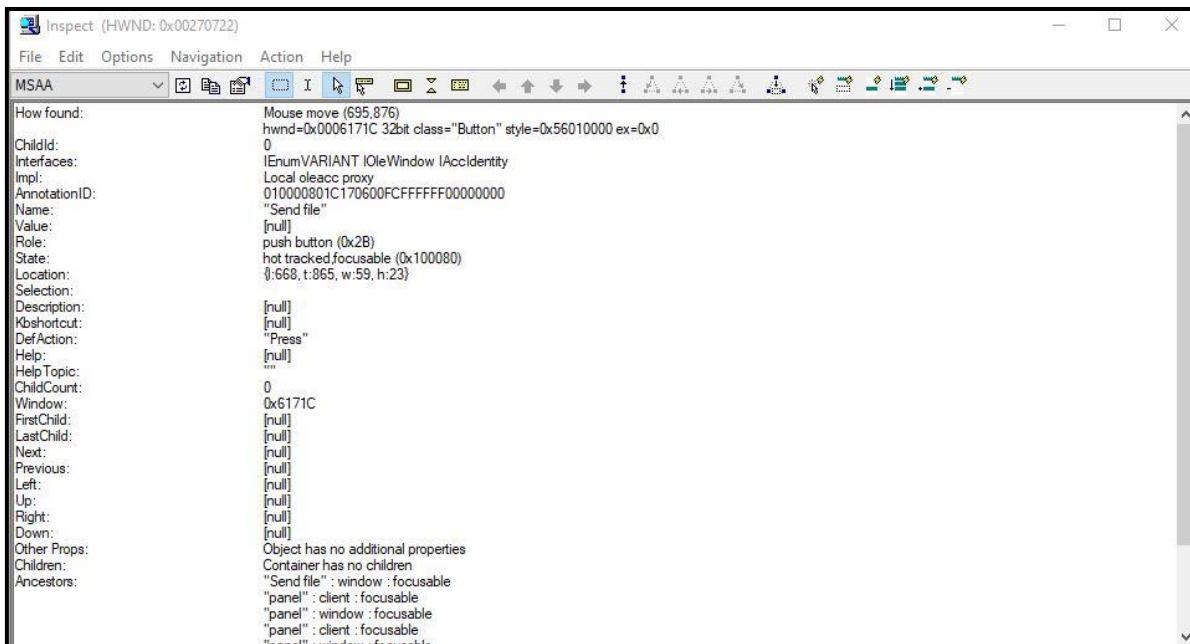


Figura 5.29 – Folosirea Inspect.exe pentru a prelua informații despre butoanele din HTerm

5.4. Delphi

Interfața grafică a aplicației desktop a fost realizată folosind resurse de pe site-ul [41].

5.4.1. ULoginP

5.4.1.1. Funcționalitate

Fereastra ‘ULoginP’ este fereastra default a aplicației. Dacă aplicația este rulată, aceasta este prima fereastră care va apărea. Fereastra permite logarea user-ului, pentru a accesa funcționalitățile aplicației. Pentru aceasta, utilizatorul trebuie să aibă un cont creat și o parolă. În cazul în care nu are, user-ul are opțiunea de a se înregistra, apăsând pe textul subliniat ‘here’ din sintagma “Don’t you have an account? Sign up here”. Parola introdusă este hash-uită. Pentru a se loga, este comparată valoarea hash-ului din aplicație, cu cea din baza de date, pentru utilizatorul cu username-ul scris. Dacă parola hash din aplicație corespunde cu cea din baza de date, utilizatorul va fi redirecționat spre fereastra principală a aplicației.

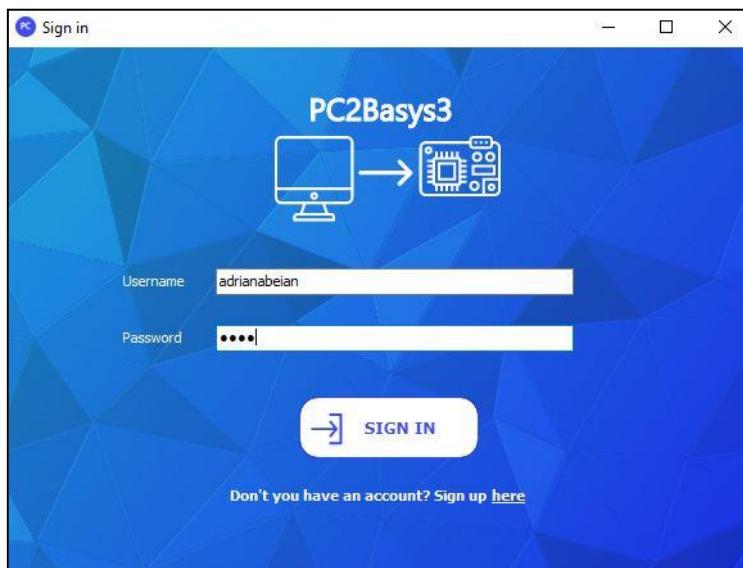


Figura 5.30 – Fereastra ULoginP

5.4.1.2. Obiecte relevante

FDConnection1 este obiectul folosit pentru a realiza conexiunea aplicației desktop cu baza de date. Principalii parametri sunt Driver ID (MySQL), Database (pc2basys3_schema), User_Name(root), Password(niciuna), Server (localhost), Port (3306). FDPhysMySQLDriverLink1 este obiectul care face referire la fișierul dll al MySQL. Parametrul VendorLib al obiectului conține calea către dll : E:\mysql_dlls\libmysql.dll. FDQuery1 este obiectul care conține query-ul care preia userii din baza de date, cu username și parola definite prin parametru. DataSource1 este un obiect care are ca dataset pe FDQuery1. Obiectul permite manipularea ușoară a datelor, și poate fi legat de un tabel în aplicația desktop, pentru a afișa coloanele tabelelor din baza de date. ImageList1 conține imaginile butonului de Sign In: imaginea când butonul nu este selectat, și imaginea când butonul este selectat.

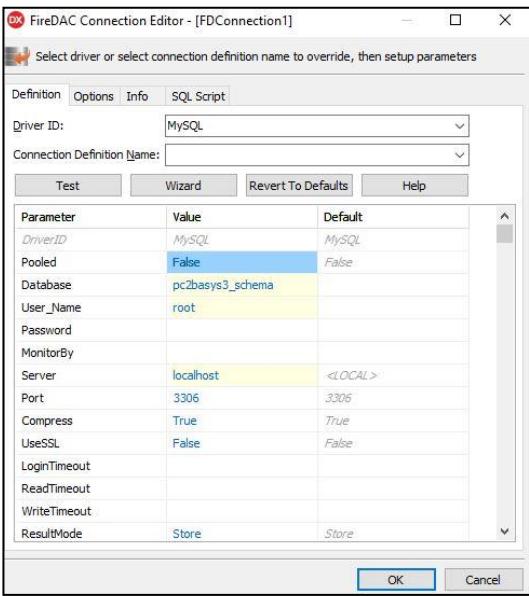


Figura 5.31 – FDConnection1

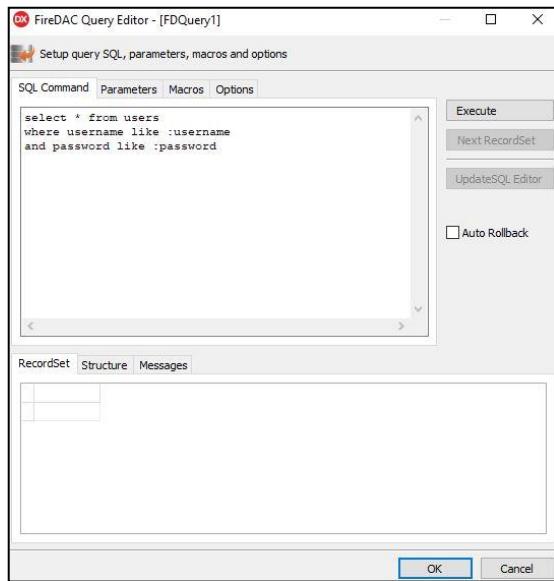


Figura 5.32 – FDQuery1

5.4.2. USignUpP

5.4.2.1. Funcționalitate

Fereastra ‘USignUpP’ are rolul de a înregistra utilizatorii care nu au un cont în baza de date. Utilizatorii au opțiunea de a-și alege o poză de profil, apăsând pe butonul ‘Choose image’. Poza de profil va putea fi încărcată din sistemul de fișiere. În cazul în care nu își aleg o poză de profil, va fi atribuită una default. Utilizatorul va trebui să își introducă prenumele, numele, username-ul și parola. Parola va fi hashuită înainte de a fi salvată în baza de date. După ce au fost introduse informațiile, utilizatorul va apăsa butonul ‘Sign Up’. Butonul ‘Clear’ are rolul de a șterge conținutul câmpurilor.

5.4.2.2. Obiecte relevante

OpenDialog1 este obiectul folosit pentru a alege o imagine din sistemul de fișiere. Utilizatorul va putea selecta imaginea dorită din folderele din sistem, și va apăsa butonul ‘Open’ pentru a confirma alegerea. Imaginea selectată va apărea în aplicație. FDQuery1 conține query-ul care preia datele despre utilizatori din baza de date. După ce user-ul a completat informațiile corespunzătoare înregistrării și după ce va apăsa butonul ‘Sign Up’, FDQuery1 va folosi modul Insert pentru a insera datele în tabela users. Imagile butoanelor au fost salvate în ImageList1 și ImageList2. A fost necesară folosirea a două obiecte ImageList pentru că dimensiunea butoanelor diferă: ‘Choose Image’ pe de-o parte, și ,’Sign Up’ și ’Clear’ pe de altă parte. ActionList1 are rolul de a stoca evenimentele, sub forma unor acțiuni. Astfel, saltul la o anumită metodă poate fi ușurat folosind acest obiect. Este nevoie doar de căutarea evenimentului corespunzător apăsării butonului. De asemenea, evenimentele pot fi grupate pe diferite categorii, fapt ce ușurează căutarea acestora. DataSource1 are ca dataset pe FDQuery1. Obiectul permite manipularea facilă a datelor, și poate fi folosit pentru a verifica preluarea datelor și inserarea lor în baza de date.

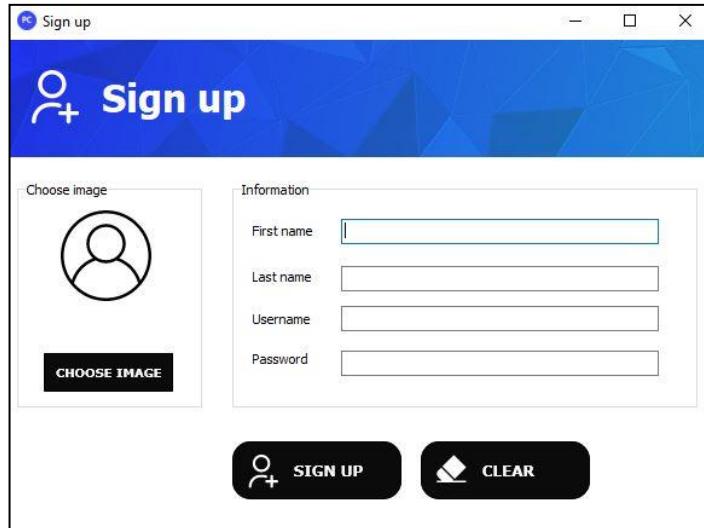


Figura 5.33 – Fereastra USignUpP

5.4.3. UPrincP

5.4.3.1. Funcționalitate

'UPrincP' este fereastra principală, care apare după ce utilizatorul se loghează în aplicație. În colțul din stânga sus vor fi afișate poza de profil, alături de prenumele și username-ul utilizatorului. Mai jos sunt butoanele care accesează principalele funcționalități ale aplicației.

'Load image' are scopul de a încărca o imagine cu actul de identitate din folderul id al sistemului de fișiere, și de a-l trimite spre procesare, atât pentru detecția cifrelor din CNP, cât și pentru transmiterea acestora la placa de dezvoltare FPGA. Alegerea imaginii se face prin apăsare butonului 'Choose Image'. Imaginea va putea fi selectată din sistemul de fișiere, iar odată selectată, aceasta va apărea în chenarul 'Original Image'. Prin apăsarea butonului 'Send image', imaginea originală va fi rulat scriptul care permite detecția cifrelor din CNP. După aproximativ 15 secunde, imaginea procesată cu cifrele din CNP detectate va apărea în chenarul 'Processed image'. Între timp, input-ul rețelei neuronale și modelul acesta vor fi trimise la placa de dezvoltare, pentru a clasifica cifrele. Butonul 'Save Images' permite salvarea imaginilor în baza de date. Butonul 'Remove Images' va șterge imaginile din cele 2 chenare, oferind posibilitatea de a începe o nouă procesare. Denumirile fișierelor originale și procesate vor fi afișate în dreptul label-urilor 'Original image filename', respectiv 'Processed image filename'.

'Results' va afișa imaginile corespunzătoare cifrelor detectate din CNP. Rezultatele apar doar în cazul în care, în prealabil, a fost încărcată o imagine cu buletinul și procesată. La rubrică 'Images' apar imaginile mărite ale cifrelor din CNP. La rubrica 'Expected results' sunt afișate rezultatele așteptate. De exemplu, dacă CNP-ul începe cu cifra 2, atunci 2 este cifra așteptată. La rubrica 'Results' apar rezultatele din urma clasificării cifrelor. În unele cazuri, rezultate așteptate și cele reale diferă. Atunci, rezultatele reale sunt scrise cu roșu. Dacă rezultatele așteptate și cele reale coincid, atunci rezultatele reale sunt scrise cu verde.

'Library' afișează imaginile salvate în baza de date, atât pe cele originale, needitate, cât și pe cele procesate. Există două tabele, unul al imaginilor originale, și celălalt al imaginilor procesate. În momentul în care este aleasă denumirea unei imagini din tabel, aceasta va fi afișată. În cazul în care imaginea afișată nu este îndeajuns de clară, aceasta va putea fi mărită, apăsând butonul 'Open Image'. Imaginea va fi deschisă într-o nouă fereastră.

‘History’ afișează istoricul evenimentelor care au avut loc în aplicație. Acțiunile user-ilor sunt salvate în baza de date. Principalele acțiuni salvate sunt alegerea imaginii, salvarea imaginilor și eliminarea acestora. Sunt afișate denumirile imaginilor originale, procesate și data în care a avut loc acțiunea.

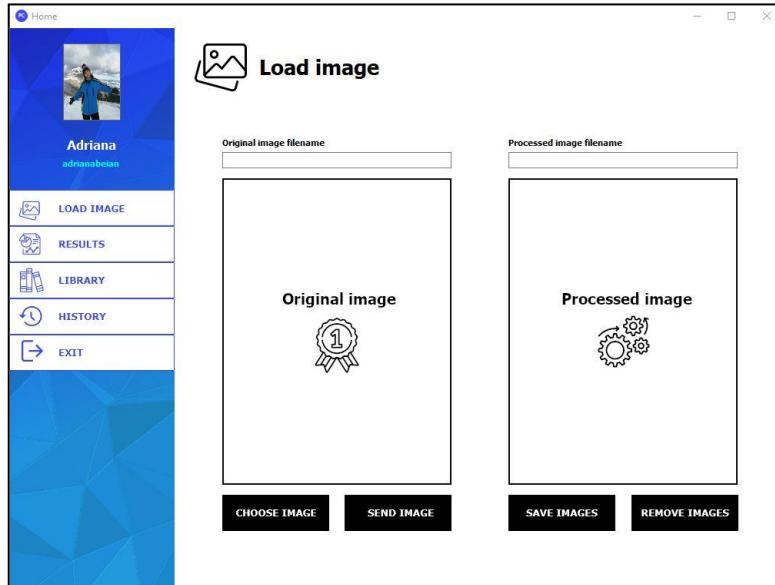


Figura 5.34 – Fereastra ‘Load image’

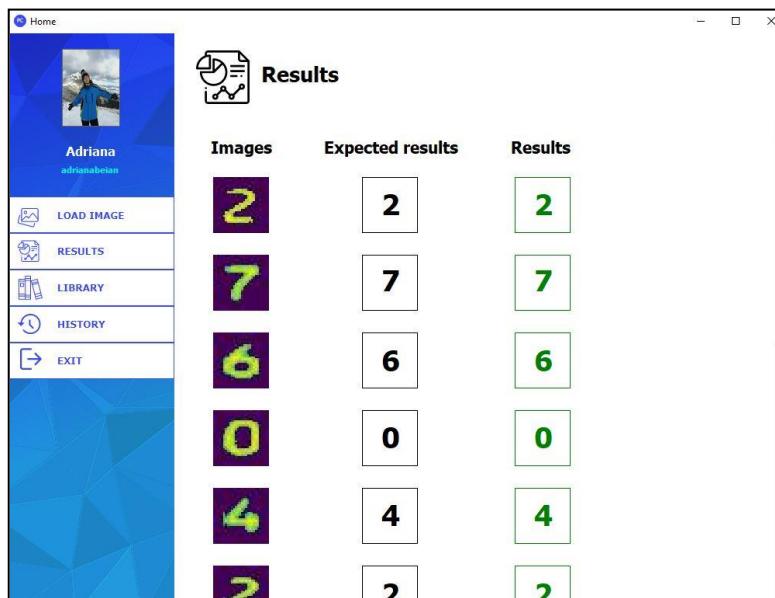


Figura 5.35 – Fereastra ‘Results’

Capitolul 5

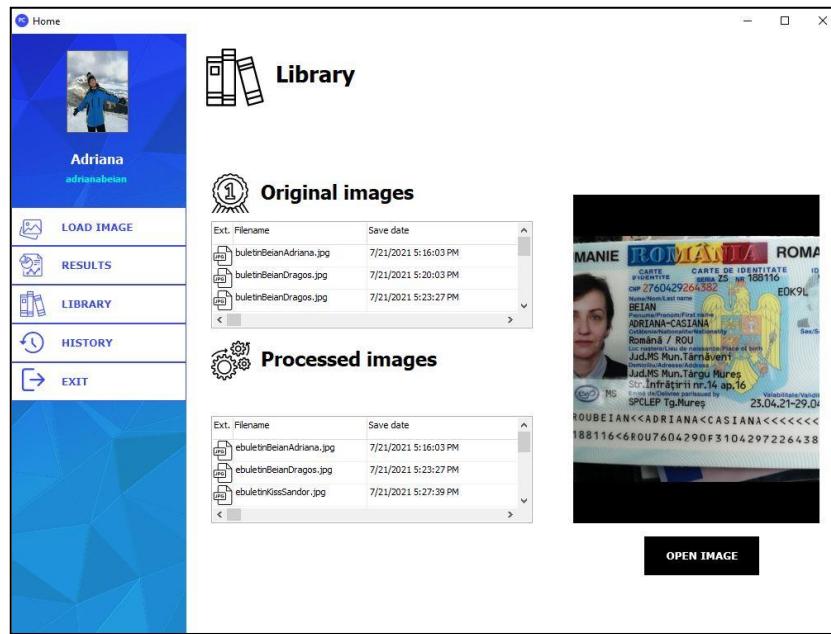


Figura 5.36 – Fereastra ‘Library’

History				
Icon	Action	Original filename	Edited filename	Save date
📁	Choose image	buletinBeanAdriana.jpg	-	7/21/2021 5:14:49 PM
📝	Save images	buletinBeanAdriana.jpg	ebuletinBeanAdriana.jpg	7/21/2021 5:16:03 PM
🗑️	Remove images	buletinBeanAdriana.jpg	ebuletinBeanAdriana.jpg	7/21/2021 5:16:17 PM
📁	Choose image	buletinBeanDragos.jpg	-	7/21/2021 5:17:34 PM
🗑️	Remove images	buletinBeanDragos.jpg	ebuletinBeanDragos.jpg	7/21/2021 5:18:39 PM
📁	Choose image	buletinBeanDragos.jpg	-	7/21/2021 5:19:06 PM
📁	Choose image	buletinBeanDragos.jpg	-	7/21/2021 5:21:55 PM
📝	Save images	buletinBeanDragos.jpg	ebuletinBeanDragos.jpg	7/21/2021 5:22:27 PM
🗑️	Remove images	buletinBeanDragos.jpg	ebuletinBeanDragos.jpg	7/21/2021 5:23:39 PM
📁	Choose image	buletinSandor.jpg	-	7/21/2021 5:23:44 PM
🗑️	Remove images	buletinSandor.jpg	-	7/21/2021 5:24:08 PM
📁	Choose image	buletinSandor.jpg	-	7/21/2021 5:24:13 PM
📁	Choose image	buletinKissSandor.jpg	-	7/21/2021 5:25:53 PM
📝	Save images	buletinKissSandor.jpg	ebuletinKissSandor.jpg	7/21/2021 5:27:39 PM
📁	Choose image	buletinKissSandor.jpg	-	7/21/2021 5:58:40 PM
📝	Save images	buletinKissSandor.jpg	ebuletinKissSandor.jpg	7/21/2021 5:59:40 PM
🗑️	Remove images	buletinKissSandor.jpg	ebuletinKissSandor.jpg	7/21/2021 6:00:23 PM

Figura 5.37 – Fereastra ‘History’

5.4.3.2. Procesarea actului de identitate

Imaginea cu actul de identitate va trebui aleasă din sistemul de fișiere, apăsând pe butonul ‘Choose Image’ și navigând printre folderele PC-ului. Odată aleasă imaginea, denumirea acesteia va fi afișată în dreptul label-ului ‘Original image filename’ iar imaginea va fi afișată în chenarul ‘Original image’. Pentru a fi procesată imaginea, trebuie apăsat butonul ‘Send Image’. În momentul în care butonul este apăsat, imaginea va fi salvată în folderul ‘chosen_image’, având denumirea ‘bulletin.jpg’. De asemenea, din baza de date vor fi preluăți parametrii necesari scriptului din Python pentru procesarea imaginii: fx,fy reprezentând coeficientul de mărire al imaginii, și threshold-ul pentru care se face binarizarea cifrelor din CNP. Acești parametri vor fi salvați în fișierul ‘config.txt’. Execuția aplicației va fi întreruptă timp de 15 secunde, folosind ‘thread.sleep(15000)’, timp în care va avea loc procesarea imaginii originale de către scriptul din Python. După cele 15 secunde, imaginea editată va apărea în folderul ‘edited_image’, și va fi preluată de aplicația desktop. În dreptul label-ului ‘Processed image filename’ va apărea denumirea imaginii editate, iar în chenarul ‘Processed image’ va apărea imaginea editată, având CNP-ul detectat scris cu galben sub cel real.

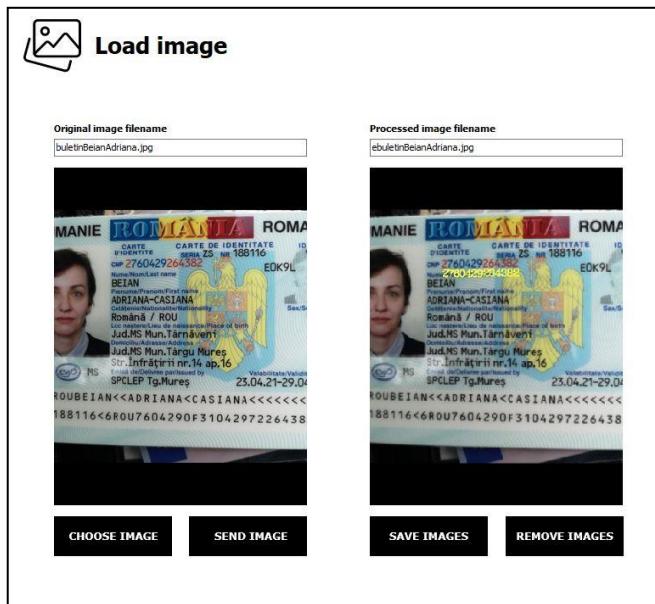


Figura 5.38 – Imaginea originală și imaginea procesată

Urmează transmiterea la placă a fișierului de input și a modelului. Pentru aceasta, va fi rulată aplicația de automatizare realizată în Java. Aplicația va transmite fișierul de input și fișierul de weights la placa de dezvoltare. Programul VHDL va fi rulat, în prealabil, pe placa FPGA. Rezultatele clasificării vor fi afișate pe SSD-ul plăcii, folosind un barrell shifter.

La închiderea ferestrei, vor fi șterse fișierele ‘edited.jpg’ din folderul ‘edited_image’, ‘bulletin.jpg’ din folderul ‘chosen_image’ și ‘config.txt’ din folderul ‘chosen_image’, pentru a putea relua procesul de transmitere când aplicația va fi repornită, fără a fi necesară ștergerea manuală a fișierelor mai sus menționate. Ștergerea fișierelor poate fi executată și apăsând pe butonul ‘Remove Images’, fără a fi necesară ieșirea din aplicație și rularea ei încă o dată, pentru a transmite o altă imagine spre procesare. Dacă se dorește încă o procesare, este necesară rularea în prealabil a scriptului din Python intitulat ‘Script_licenta’, și a programului în VHDL.

5.5. MySQL

Baza de date are ca scop salvarea informațiilor despre utilizatori (nume, prenume, username, parola, imaginea de profil), imaginile originale (denumire, CNP, parametrii fx, fy și threshold), imaginile originale și editate aparținând utilizatorilor (denumire, extensie, tipul imaginii, imaginea, data salvării și id-ul utilizatorului căruia îi aparține imaginea), istoricul evenimentelor din aplicația desktop (eveniment, denumire fișier original, denumire fișier procesat, data salvării și id-ul utilizatorului care a întreprins acțiunea).

Primul pas în crearea bazei de date a fost crearea unei scheme. Aceasta este intitulată ‘pc2basys3_schema’. Schema conține 4 tabele: users, images, images_users și histories.



Figura 5.39 – Schema și tabelele existente

Tabela ‘users’ conține id-ul utilizatorului, username, password, firstName, lastName și picture. Câmpul id este INT(10), cheie primară, în timp ce câmpurile username, password, firstName și lastName sunt VARCHAR(45). Imaginile sunt stocate în picture, care este un câmp de tipul LONGBLOB. Parola salvată este encriptată. Informațiile din tabela ‘users’ sunt salvate în baza de date în momentul înregistrării utilizatorului în aplicația desktop.

Tabela ‘images’ conține id-ul imaginii, denumirea, CNP-ul și parametrii fx,fy și threshold. Câmpul id este INT(10), cheie primară, name și cnp sunt VARCHAR(45), fx și fy sunt DOUBLE, iar threshold este INT(11). Informațiile din tabela ‘images’ sunt salvate înainte de folosirea imaginilor în aplicație. Este nevoie de testarea imaginilor în script-ul Python ‘Script_licenta’ pentru a determina valorile parametrilor fx, fy și threshold, și a-i salva în tabela ‘images’. Când utilizatorul optează pentru a trimite o anumită imagine spre procesare, aplicația preia informațiile din baza de date și le salvează într-un fișier de configurare.

Tabela ‘images_users’ conține denumirea fișierului salvat de utilizator, extensia imaginii, tipul imaginii (imagine originală sau imagine editată), imaginea în sine, data salvării și id-ul utilizatorului care a salvat imaginea. Imaginile sunt salvate din aplicație, după ce a avut loc procesarea. Fișierele originale vor avea tipul fișierului salvat ca ‘original’, iar fișierele editate, vor fi salvate ca ‘edited’. Cheia străină, users_id este în corespondență cu id-ul utilizatorului din tabela ‘users’. Aceste imagini vor putea fi preluate din baza de date, când se accesează funcționalitatea ‘Libraries’ din aplicația desktop.

Tabela ‘histories’ conține id-ul, evenimentul/acțiunea care a avut loc, denumirea fișierului original, denumirea fișierului editat, data salvării și id-ul utilizatorului care a întreprins acțiunea. Fiecare acțiune este salvată în momentul apăsării unuia dintre butoanele ‘Choose Image’, ‘Save Images’ sau ‘Remove Images’. Cheia străină, users_id este în corespondență cu id-ul utilizatorului din tabela ‘users’.

Tabelele pot fi vizualizate, făcând Reverse Engineering pe schema ‘pc2basys3_schema’. Rezultatul acestei acțiuni poate fi observat în Figura 5.40.

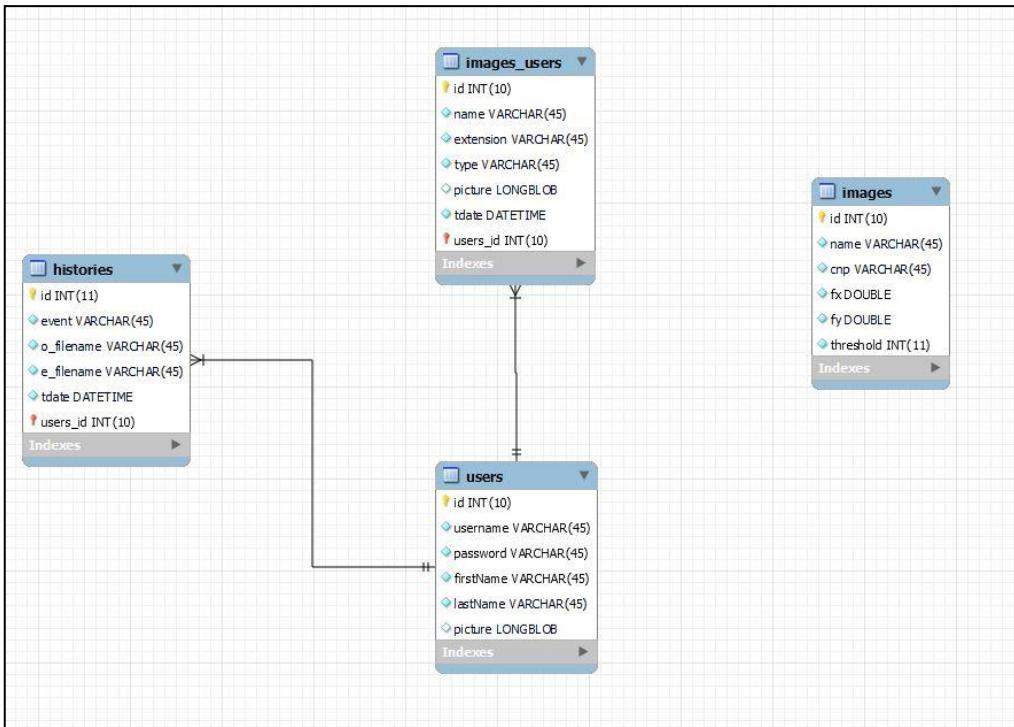


Figura 5.40 – Schema ‘pc2basys3_schema’

Tabela ‘users’ interacționează cu tabelele ‘images_users’ și ‘histories’. Tabela ‘images’ nu are legătură cu celelalte tabele. Între ‘users’ și ‘image_users’ este o relație one to many. Un user poate avea mai multe imagini. O imagine poate avea un singur user. De asemenea, între ‘users’ și ‘histories’ este o relație one to many. Un utilizator poate întreprinde mai multe evenimente, dar un eveniment poate avea un singur utilizator.

Capitolul 6. Testare și validare

Sistemul de detectie al cifrelor care formează codul numeric personal al unui buletin este compus din mai multe module. Fiecare modul a fost proiectat, implementat și testat. Principalele module ale aplicației sunt: rețea neuronală artificială din VHDL, rețea neuronală conlovuțională din Python, script-ul OCR, aplicația de automatizare a transmiterii fișierelor la placa de dezvoltare și aplicația desktop. La rândul lor, fiecare dintre module este realizat din mai multe componente. De exemplu, rețea neuronală din VHDL are ca și componente principale: componenta UART pentru transmisia și receptia datelor, componenta BRAM care are rolul de a salva input-urile și modelul rețelei, componenta FSM care implementează, folosind un automat de stări finite, partea de forwarding a rețelei neuronale. Astfel, pentru a ilustra testarea și validarea sistemului, este necesară prezentarea testării pe diferite module ale aplicației.

Rețea neuronală artificială VHDL

Rețea neuronală implementată în VHDL are mai multe componente. Pentru fiecare dintre aceste componente au fost create proiecte de test, pentru a verifica corectitudinea rezultatelor. În cazul în care aceste proiecte nu ar fi fost realizate, rețea neuronală artificială ar fi avut erori la fiecare componentă, fiind greu de depistat și eliminat. În cele ce urmează, vor fi enumerate proiectele de test create, alături de o scurtă descriere a funcționalității lor.

1. Testare UART Receiver

Sunt trimise caractere de la calculator la placa de dezvoltare, folosind terminalul Hterm. Baud-rate-ul este de 9600. De exemplu, caracterul ‘a’ este scris în terminalul Hterm și trimis la placă. Placa va afișa pe SSD valoarea codului ASCII a lui a, în hexazecimal. Proiectul are ca și componente implementare SSD pentru afișarea datelor pe placă, respectiv RX_FSM, componentă care permite receptia datelor de la calculator, folosind un automat cu stări finite. Ambele componente fac parte din proiectul final, ANN, reprezentând rețea neuronală artificială folosită pentru clasificarea cifrelor CNP-ului.

2. Testare UART Transmitter

Sunt trimise caractere de la placă la calculator, la un baud-rate de 9600. Datele sunt afișate în terminalul Hterm. Caracterele sunt reprezentate în binar, folosind switch-urile. De exemplu, caracterul ‘A’ are codul ASCII 65. Valoarea 65 a fost reprezentată în binar folosind switch-urile și trimisă la calculator, prin apăsarea unui buton de pe placă. Caracterul ‘A’ a fost recepționat de terminal și afișat. Proiectul are ca și componente implementate MPG pentru apăsarea butoanelor, respectiv TX_FSM, componentă care permite transmiterea datelor de la placă la calculator. Ambele componente fac parte din proiectul final, ANN, reprezentând rețea neuronală artificială folosită pentru clasificarea cifrelor CNP-ului.

3. Reprezentarea numerelor în fixed point

Proiectul are ca scop testarea reprezentării numerelor în fixed point și verificarea sumei și produsului a două valori predefinite. Astfel, au fost create semnalele a și b care conțin numerele pe 16 biți, sfixed, respectiv semnalele de sumă și produs. Cele două numere au fost adunate și înmulțite, iar rezultatele au fost afișate pe led-uri. Suma și produsul sunt două operații fundamentale pentru rețea neuronală artificială, așa că testarea lor în prealabil a fost

obligatorie. Reprezentarea numerelor în fixed point și formulele pentru sumă și produs au fost preluate din acest proiect și folosite în proiectul final al rețelei neuronale.

4. Memoria BRAM

A fost creată o aplicație de test care folosește o memorie BRAM pentru a stoca date pe 16 biți. Datele sunt stocate la o adresă, a cărei valoare este dată de un counter care se poate incrementa și decrementa, în funcție de valoarea unuia dintre switch-uri. Datele salvate sunt afișate pe leduri. Aplicația a stat la baza creării și folosirii memorii BRAM. În rețeaua neuronală artificială, memorii BRAM sunt folosite pentru a stoca input-urile și modelul.

5. Clasificarea unei singure cifre folosind o rețea neuronală artificială

A fost creat un proiect în VHDL, pentru a clasifica o singură cifră, din cele 13 ale CNP-ului, folosind o rețea neuronală artificială. Rețeaua neuronală folosea toate componentele descrise mai sus: protocolul UART pentru transmisia și receptia datelor a fost implementat pentru a putea transmite fișierele de input și weights de la calculator la placă, numerele au fost reprezentate în fixed point. S-a folosit un FSM cu 6 stări. Cifra trimisă sub forma unui fișier de input a fost clasificată.

Toate aceste proiecte de test, descrise pe scurt, au contribuit la proiectarea, implementarea și mai ales testarea aplicației finale din VHDL: rețeaua neuronală artificială care detectează toate cele 13 cifre ale CNP-ului.

Testarea rezultatelor rețelei neuronale artificiale s-a făcut în două moduri. În primul rând au fost testate rezultatele fiecărui output din cele 10 output-uri posibile, pentru fiecare cifră din CNP, din cele 13 posibile. Așadar, un total de 130 de output-uri au fost salvate într-un array. Valorile acestor output-uri au fost afișate pe led-uri, folosind un counter, și comparate cu valorile prezise de rețeaua neuronală în Python. A fost creat un algoritm pentru a prelua valorile fiecărui output, înainte de aplicarea funcției sigmoid pe acestea. Astfel, tabelul 1 prezintă rezultatele așteptate pentru fiecare output al primei cifre dintr-un CNP procesat de rețeaua neuronală (cifra 2 în acest caz), înainte de apicarea funcției sigmoid, rezultatele efective, și valoarea în fixed point representation.

<i>Output</i>	<i>Rezultat asteptat înainte de sigmoid</i>	<i>Rezultat efectiv înainte de sigmoid</i>	<i>Valoare în fixed point representation</i>
0	-10.33	-10.75	1111010101001010
1	-9.00	-9.44	1111011010010111
2	-1.90	-1.25	1111110111001010
3	-4.55	-4.94	1111101100011001
4	-15.33	-15.75	1111000001001111
5	-16.47	-16.82	1110111100110011
6	-17.24	-17.63	1110111001101111
7	-8.44	-8.88	1111011100101101
8	-7.58	-8.00	1111100000000111
9	-10.42	-10.75	1111010101001101

Tabelul 6.1 – Rezultatele output-urilor pentru cifra 2 dintr-un CNP

În Figura 6.1, a fost afișat al patrulea output al cifrei 2 clasificate folosind rețeaua neuronală artificială. Valoarea led-urilor corespunde cu valoarea fixed point representation din Tabelul 1, pentru output-ul 3. Cu cât valoarea din tabel este mai mare, cu atât probabilitatea ca cifra clasificată să coincidă cu acel output este mai mare. După cum se poate observa din Tabelul 1, output-ul 2 are valoarea cea mai mare, fiind de asemenea chiar valoarea primei cifre din CNP. SSD-ul afișează cifra 3, reprezentând output-ul al patrulea din cele 130 de output-uri salvate (câte 10 pentru fiecare din cele 13 cifre ale CNP-ului).

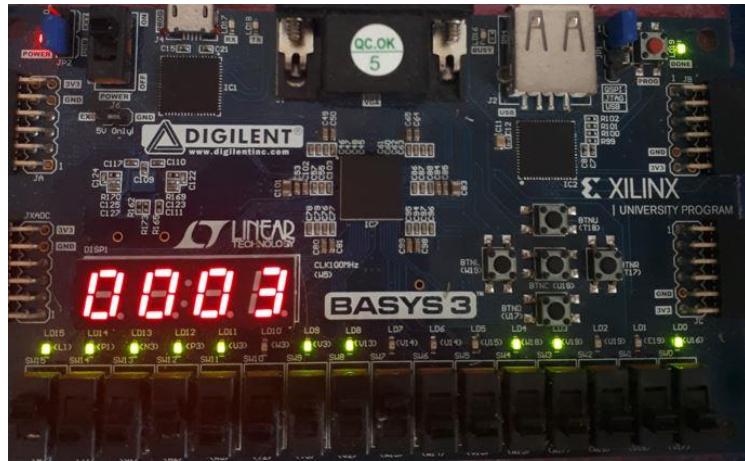


Figura 6.1 – Output-ul 3 al cifrei 2 din CNP

CNP-ul detectat de rețeaua neuronală artificială a fost afișat pe SSD, folosind un barrel shifter. Cifrele sunt shiftate la un interval de 2 secunde. Valoarea afișată pe SSD a fost comparată cu valoarea reală a CNP-ului, pentru a testa acuratețea clasificării. În urma acestei testări, a rezultat faptul că cifra 1 din CNP este clasificată ca fiind 7. Acest fapt este datorat antrenării rețelei neuronale convoluționale cu un dataset format din cifre scrise de mână, în timp ce cifrele de pe buletin sunt cifre scrise de tipar.

Rețeaua neuronală convoluțională Python

Rețeaua neuronală convoluțională are scopul de a clasifica cifrele scrise de mână. Antrenarea și testarea rețelei s-a făcut folosind dataset-ul de la MNIST pentru cifre scrise de mână. Dataset-ul de antrenare are 60.000 de imagini, iar cel de testare are 10.000 de imagini. În urma antrenării rețelei neuronale, folosind un model cu 3 layeruri de convoluție, 2 de max-pooling și o rețea neuronală artificială, complet conectată, cu un input layer și un output layer, s-a obținut o acuratețe de 99%. S-a verificat acuratețea și pe dataset-ul de testare, aceasta fiind de 98%. Împărțirea imaginilor în două dataset-uri și testarea antrenării pe dataset-ul de testare constituie o metodă de testare a acurateții rețelei neuronale convoluționale.

După ce a avut loc antrenarea rețelei neuronale convoluționale, s-a încercat o predicție pe baza imaginilor de test. Confusion matrix-ul relevă de câte ori au fost prezise corect valorile. Se poate observa că valorile de pe diagonală sunt cele mai mari, întrucât cifrele au fost, în general, prezise corect, rețeaua neuronală având o acuratețe de aproximativ 98%. Confusion matrix relevă și cifrele care pun probleme. De exemplu, cifra 9 a fost prezisă de 12 de ori ca fiind 4.

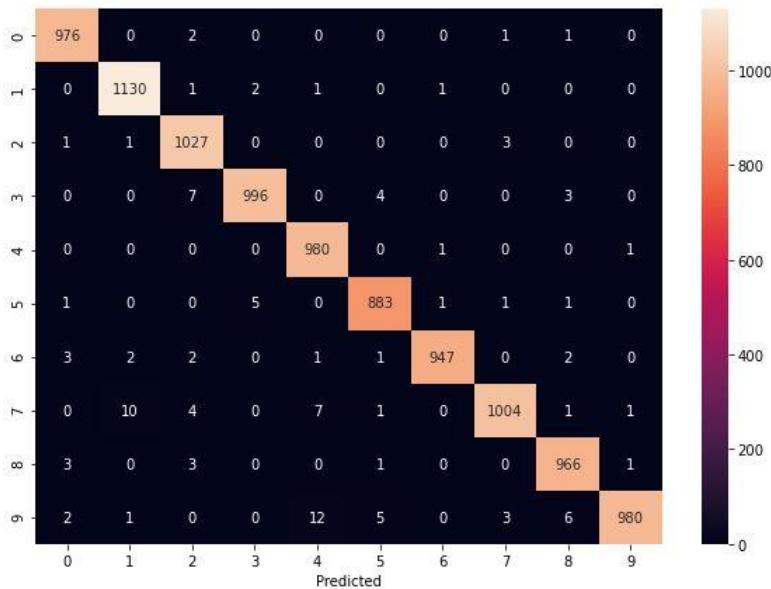


Figura 6.2 – Confusion matrix rețea neuronală conoluțională

Script OCR

Imaginea originală, cu actul de identitate, din care a fost detectat CNP-ul a necesitat o preprocesare înainte de a se aplica algoritmul OCR. Imaginea a fost transformată în grayscale, apoi din grayscale s-a aplicat algoritmul Canny pentru a detecta muchiile. Rezultatele acestor preprocesări au fost afișate după fiecare etapă a pipeline-ului de preprocesare. Ulterior, folosind OCR, au fost preluate caracterele din imagine. Acestea au fost afișate, fapt ce a dus la compunerea algoritmului de detecție al cifrelor CNP-ului: oricare 13 cifre consecutive formează codul numeric personal. Algoritmul OCR a preluat, pe lângă caractere, o serie de informații precum coordonatele x,y ale caracterului, width-ul și height-ul acestuia. Folosind aceste informații, a fost posibilă localizarea cifrelor și încadrarea lor într-un bounding box pentru testare. Cifrele au fost încadrate în chenare de culoare albăstră. De asemenea, sub fiecare cifră din CNP a fost scrisă, cu galben, valoarea cifrei detectate de script.



Figura 6.3 – Încadrarea cifrelor CNP-ului în bounding boxes

Aplicație automatizare

Aplicația de automatizare realizată în Java are ca scop trimiterea fișierelor de input și weights la placa de dezvoltare Basys3. Aplicația este rulată, iar procesul de transmitere al fișierelor este afișat pe ecran. Faptul că fișierele au fost trimise corect și în întregime la placă este confirmat, în primul rând, de verificarea memoriei BRAM aferentă input-urilor și weights-urilor. Verificarea valorilor stocate se face din VHDL, folosind un numărător pe post de adrese, iar valorile pot fi afișate pe led-uri și comparate cu cele din fișierele corespunzătoare.

Capitolul 7. Manual de Instalare și Utilizare

7.1. Manual de Instalare

Pentru a rula aplicația practică din lucrarea de față, este necesară folosirea unui PC sau laptop. De asemenea, aplicația se va rula în sistemul de operare Windows. Motivul pentru care este obligatorie folosirea Windows, este în primul rând aplicația de automatizare a trimiterii datelor de la calculator la placă de dezvoltare. În acest sens s-a folosit Winium, care poate rula doar pe Windows.

Este important de menționat că versiunile aplicațiilor din momentul realizării lucrării de licență ar putea să nu coincidă cu versiunile aplicațiilor la momentul testării proiectului de licență. De asemenea, link-urile sau interfața grafică a site-urilor ar putea suferi modificări.

7.1.1. Instalare Anaconda Navigator

1. Se va accesa link-ul [20] <https://www.anaconda.com/> folosind browserul preferat de Internet.
2. Se va apăsa butonul Downloads din colțul din dreapta, sus.
3. Vor apărea opțiuni pentru Windows, MacOS și Linux. Se va alege opțiunea de 32 sau 64 biți, cu versiunea de Python 3.6, pentru sistemul de operare Windows.
4. Installerul se va descărca.
5. După ce installerul s-a descărcat complet, se va rula executabilul aferent.
6. Se va instala aplicația, prin apăsarea Next-> I Agree->alegerea opțiunii All Users->Next->Next.
7. Se va ajunge la fereastra Advanced Installation Options. Se recomandă bifarea opțiunii “Add Anaconda to the system PATH environment variable”, dacă Anaconda sau Python nu au mai fost instalate. Dacă au fost instalate în trecut, se recomandă debifarea opțiunii. Opțiunea “Register Anaconda as the system Python 3.6” poate rămâne bifată.
8. Se va apăsa butonul Install.
9. După ce aplicația s-a instalat complet, se vor apăsa butoanele Next->Skip->Finish.

În Figura 7.1 au fost afișate ferestrele relevante pentru instalarea Anaconda Navigator [18].

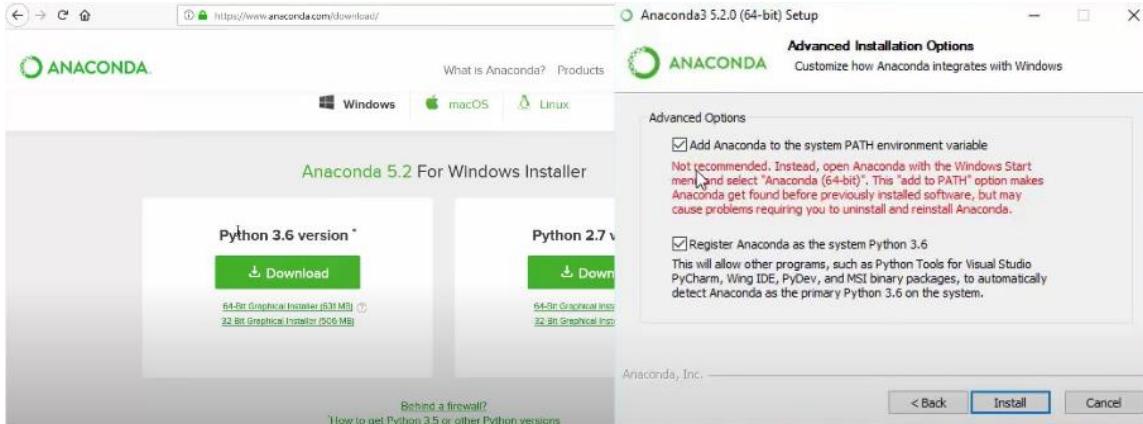


Figura 7.1 – Ferestre relevante pentru instalarea Anaconda Navigator

7.1.2. Crearea unui environment în Anaconda Navigator

1. După instalarea de la punctul 8.1.1., în tab-ul de la Windows se va căuta Anaconda Navigator și se va da click pe acesta.
2. Se va deschide o fereastră ca prima imagine din Figura 7.2.
3. În meniu din stânga, se va da click pe “Environments”.
4. Se va apăsa butonul “Create” pentru a crea un environment.
5. Se va deschide o fereastră ca a doua imagine din Figura 7.2. Se va introduce denumirea environment-ului și se va apăsa butonul “Create”.
6. Environment-ul nou creat va apărea precum ultima imagine din Figura 7.2.

După crearea unui environment, se vor instala librăriile necesare funcționării aplicației.



Figura 7.2 – Ferestre relevante pentru crearea unui environment în Anaconda Navigator

7.1.3. Instalarea librăriilor necesare aplicației în environment

1. Se va deschide fereastra “Environments” din Anaconda Navigator.
2. Se va da click pe triunghiul verde “Play” și se va apăsa “Open Terminal”.
3. Se va deschide un terminal precum cel din Figura 7.3.
4. Se vor instala următoarele librării, prin scrierea următoarelor linii de cod în terminal:
 - Tensorflow: conda install -c conda-forge tensorflow
 - Matplotlib: conda install -c conda-forge matplotlib
 - Numpy: conda install -c anaconda numpy
 - Pytesseract: conda install -c conda-forge pytesseract
 - OpenCV: conda install -c conda-forge opencv
 - Pillow: conda install -c conda-forge pillow
 - Pickle: conda install -c conda-forge pickle5
 - Scikit-learn: conda install scikit-learn

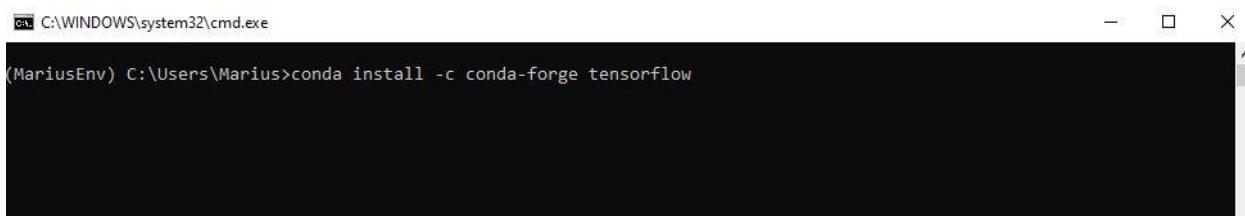


Figura 7.3 – Terminal deschis în environment-ul “MariusEnv”

7.1.4. Instalare Xilinx Vivado

1. Se va accesa link-ul [21] <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html>
2. Se va da click pe versiunea 2019.1.
3. În secțiunea “Vivado Design Suite-HLx Editions – 2019.1 Full Product Installation” se va alege “Vivado HLx 2019.1: WebPACK and Editions – Windows Self-Extracting Web Installer”.
4. În pagina afișată, se vor introduce credențialele.
5. Următoarea pagină se va completa automat după logare.
6. Se va apăsa butonul “Download” din josul paginii.
7. După descărcarea installerului, va urma instalarea programului.
8. Se va rula fișierul executabil de instalare.
9. Se va apăsa butonul Next.
10. În fereastra deschisă, la secțiunea “User Authentication” se vor introduce credențialele și se va apăsa butonul Next.
11. În următoarea fereastră, se vor alege toate opțiunile “I Agree” și se va apăsa butonul Next.
12. Se va selecta “Vivado HL WebPACK” și se va apăsa Next->Next->Install.

7.1.5. Instalare XAMPP

1. Se va accesa link-ul [22] <https://www.apachefriends.org/download.html>.
2. Se va alege una din opțiunile de descărcare din secțiunea “XAMPP for Windows” și se va apăsa butonul “Download (64 bit)”.
3. Se va rula installerul descărcat.
4. Se vor apăsa Ok->Next->Next->Next->Next->Next.
5. Se va aștepta instalarea XAMPP și se va apăsa Next->Finish.

7.1.6. Instalare Hterm

1. Se va accesa link-ul [23] <http://der-hammer.info/pages/terminal.html>.
2. Se va dezarhiva arhiva descărcată.
3. Se va rula fișierul “hterm.exe”.

7.2. Manual de Utilizare

7.2.1. Rulare MySQL folosind XAMPP

Rularea MySQL folosind XAMPP implică instalarea prealabilă a XAMPP, conform instrucțiunilor prezentate în subcapitolul 7.1.5. Pașii pentru pornirea localhost-ului:

1. Se va da click pe butonul Start din Windows.
2. Se va introduce textul "xampp" și se va alege "XAMPP Control Panel", conform Figurii 7.4.
3. În fereastra nou deschisă, se va apăsa butonul "Start" din rubrica "Actions", în dreptul modulului "MySQL".

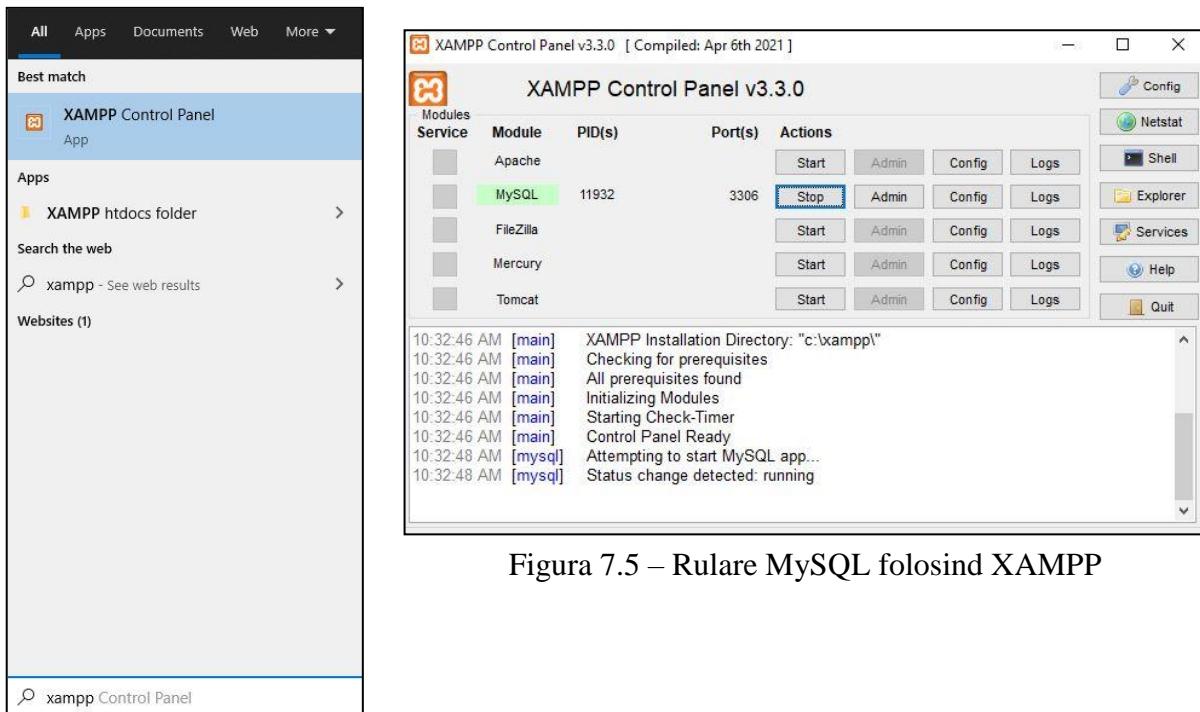


Figura 7.5 – Rulare MySQL folosind XAMPP

Figura 7.4 – Deschidere "XAMPP"

7.2.2. Rulare script licență

Rularea scriptului python implică efectuarea pașilor din sucapitolele 7.1.1, 7.1.2 și 7.1.3.

1. Se va da click pe butonul Start din Windows.
2. Se va introduce textul "anaconda" și se va alege "Anaconda Navigator", conform Figurii 7.6.
3. În fereastra "Anaconda Navigator", se va apăsa butonul "Environments" din meniu.
4. Se va da click pe environment-ul care conține libăriile necesare rulării scriptului.
5. Se va aștepta încărcarea pachetelor.
6. Se va apăsa pe triunghiul verde "Play" și se va alege "Open Terminal", conform Figurii 7.7.

7. În terminal, se va parcurge sistemul de fișiere pentru a ajunge în directorul care conține scriptul python intitulat “Script_licenta”. O alternativă este alegerea partitiei în care se află scriptul, de exemplu “E:” și apoi introducerea liniei “cd calea_absolută”, precum ”cd E:\Licenta”.
8. Odată accesat directorul care conține scriptul python, se va rula jupyter notebook prin scrierea în terminal a liniei “jupyter notebook”.
9. Se va deschide fișierul python “Script_licenta”.
10. În meniul de sus, se va alege “Cell->Run All”.

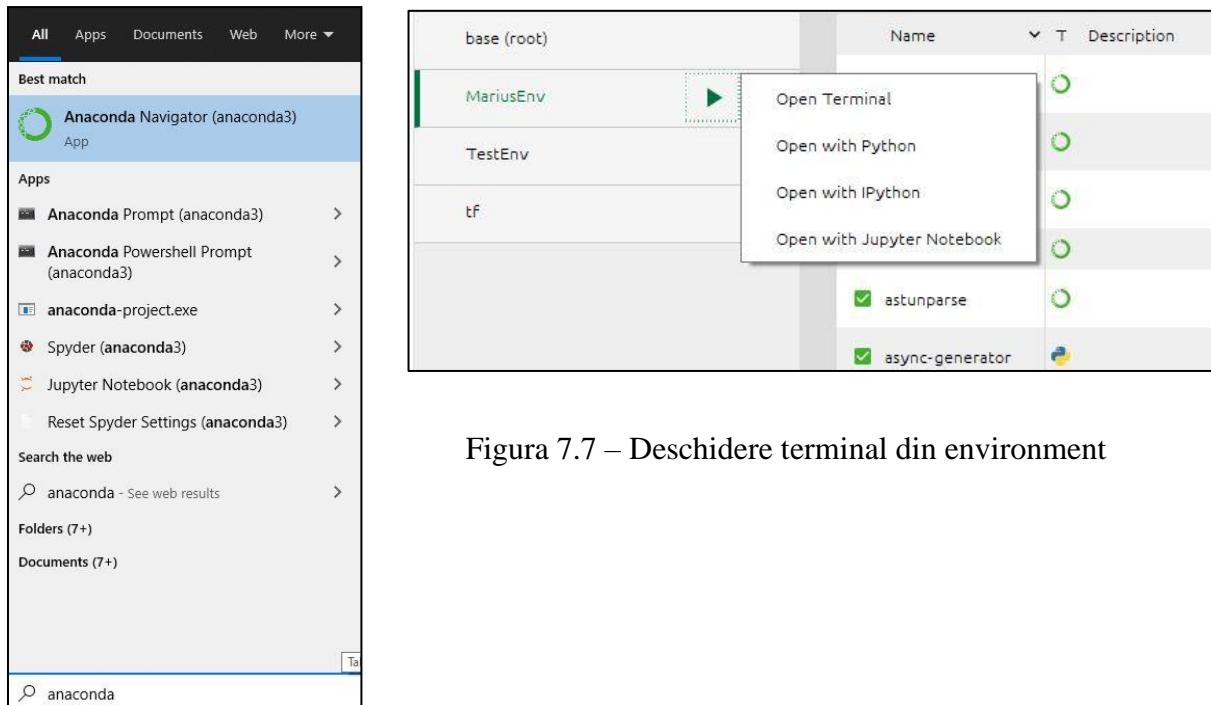


Figura 7.6 – Deschidere “Anaconda Navigator”

Figura 7.7 – Deschidere terminal din environment

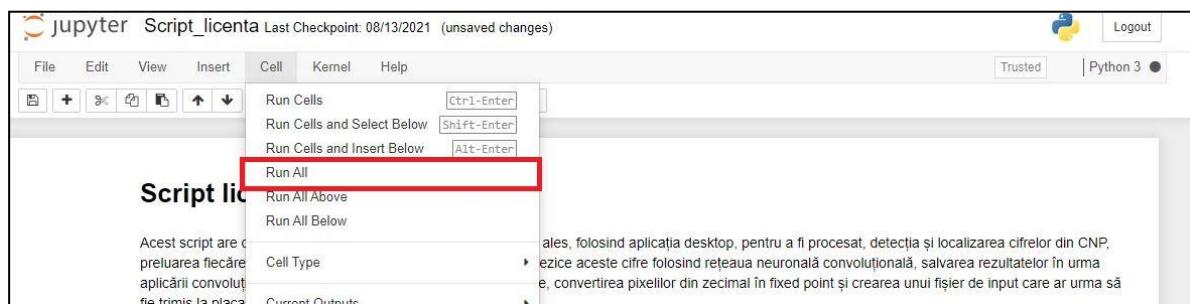


Figura 7.8 – Execuție comanda “Run All”

7.2.3. Rulare Winium Desktop Driver

Se va rula executabilul “Winium.Desktop.Driver.exe” pentru a permite funcționarea aplicației de automatizare a transmiterii fișierelor la placa de dezvoltare Basys3.

7.2.4. Rulare proiect VHDL

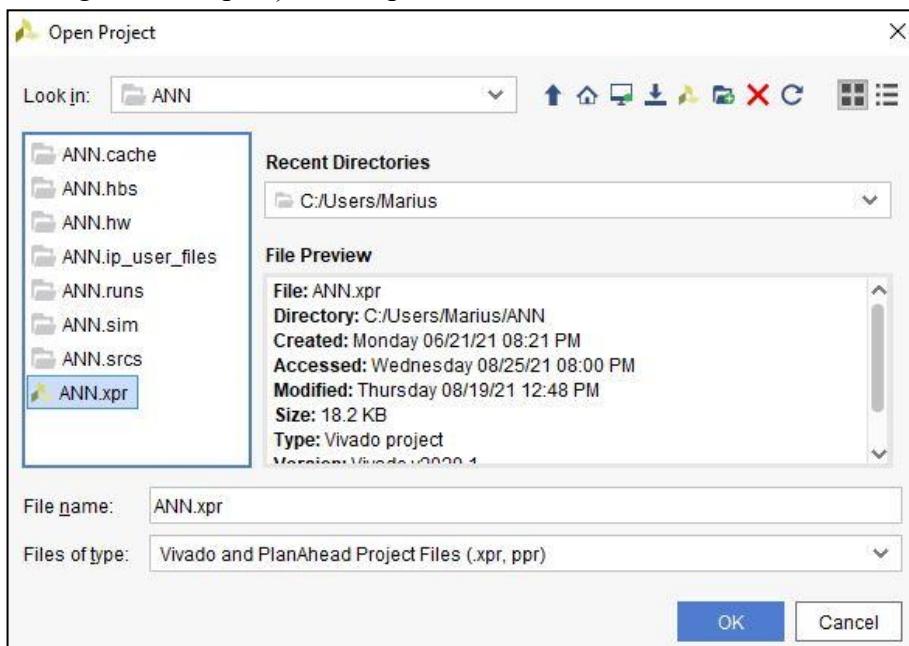
Rularea proiectului VHDL care conține rețeaua neuronală artificială pentru clasificarea cifrelor din CNP, implică efectuarea pașilor din subcapitolul 7.1.4.

1. Se va da click pe butonul Start din Windows.
2. Se va introduce textul “vivado” și se va alege “Vivado 2020.1”.
3. Se va apăsa butonul “Open Project” din rubrica “Quick Start”.



Figura 7.9 – Deschidere proiect VHDL

4. Se va parcurge sistemul de fișiere pentru a găsi proiectul “ANN.xpr”.
5. Se va alege “ANN.xpr” și se va apăsa butonul “OK”.



6. Proiectul ANN.xpr va fi deschis.

Figura 7.10 – Alegere proiect “ANN.xpr”

7. Din ”PROJECT MANAGER” se va alege “Run Synthesis” și se va aștepta până când proiectul se sintetizează.

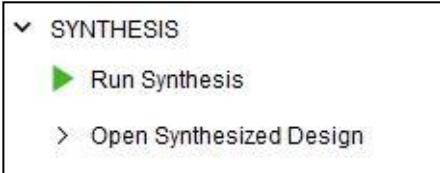


Figura 7.11 – Run Synthesis

8. Din ”PROJECT MANAGER” se va alege “Run Implementation” și se va aștepta până când proiectul se implementează.



Figura 7.12 – Run Implementation

9. Din ”PROJECT MANAGER” se va alege “Generate Bitstream” și se va aștepta până la încheierea procesului.

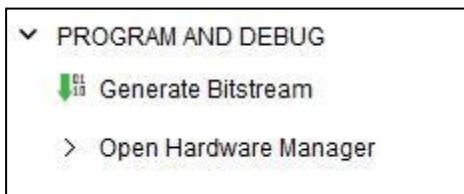


Figura 7.13 – Generate Bitstream

10. Se va conecta placa Basys3 la calculator, precum în imaginea de mai jos, folosind un cablu de date USB 3.0 A tata / micro B tata.

11. Din ”PROJECT MANAGER” se va alege “Open Hardware Manager”.Se va alege “Open Target” din partea de sus a proiectului, apoi “Auto Connect”.



Figura 7.14 – Open target

12. Se va apăsa “Program device” din partea de sus a proiectului.



Figura 7.15 – Program device

7.2.5. Rulare aplicație desktop

Pentru a beneficia la maxim de experiența utilizării aplicației, este necesară completarea pașilor din subcapitolele anterioare.

7.2.5.1. Sign up/Sign in

1. Se va rula executabilul “PC2Basys3.exe”.
2. În fereastra de “Sign In” se vor introduce username-ul și parola, și se va apăsa pe butonul de “SIGN IN”.
3. În cazul în care utilizatorul nu este înregistrat, acesta are opțiune de a face acest lucru prin apăsarea cuvântului subliniat “here” din sintagma “Don’t you have an account? Sign up here”.
4. Utilizatorul va fi redirecționat spre fereastra de “Sign Up”.
5. Utilizatorul se va înregistra folosind prenumele, numele, username-ul și parola. De asemenea, utilizatorul are opțiunea de a adăuga o imagine de profil.
6. După completarea datelor, utilizatorul va apăsa “SIGN UP” pentru a se înregistra.
7. Pentru a șterge datele, se va apăsa butonul “CLEAR”.
8. Dacă înregistrarea s-a încheiat cu succes, utilizatorul va fi redirecționat spre fereastra principală a aplicației.

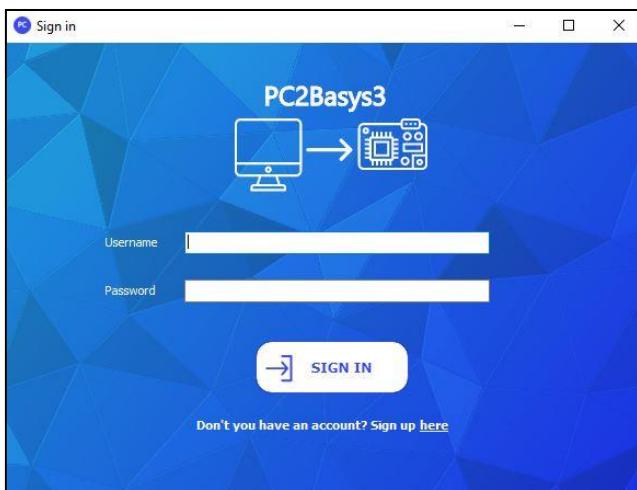


Figura 7.16 – Fereastra ‘Sign In’

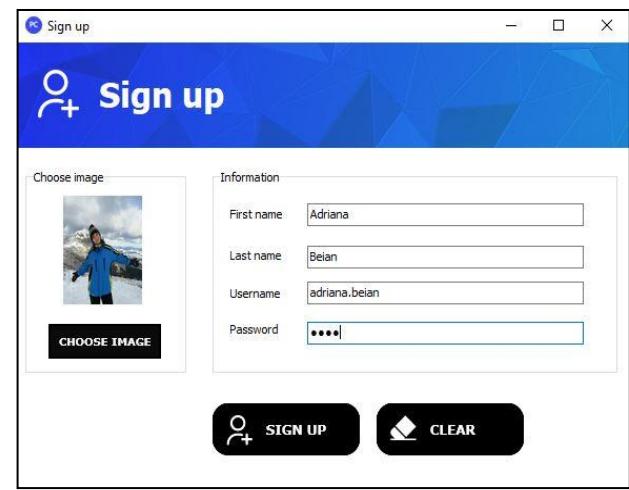


Figura 7.17 – Fereastra ‘Sign Up’

7.2.5.2. Load image

1. Se va apăsa butonul “LOAD IMAGE” din meniul din partea stângă a ferestrei principale.
2. Se va apăsa butonul “CHOOSE IMAGE” pentru a alege o imagine cu butonul din sistemul de fișiere.
3. După selectarea imaginii, aceasta va apărea în locul rezervat imaginii originale “Original image”.
4. Se va apăsa “SEND IMAGE” pentru a detecta cifrele din actul de identitate ales la pasul anterior.
5. După aproximativ 15 secunde, imaginea editată va apărea în locul rezervat imaginii editate “Processed image”.

6. Fișierele de input și modelul rețelei neuronale artificiale vor fi trimise la placă, folosind procesul de automatizare.
7. CNP-ul va apărea pe SSD-ul plăcii.
8. Se va apăsa “SAVE IMAGES” pentru a salva imaginile în baza de date.
9. Se va apăsa “REMOVE IMAGES” pentru a elimina imaginile din rubricile “Original image” și “Processed image”.

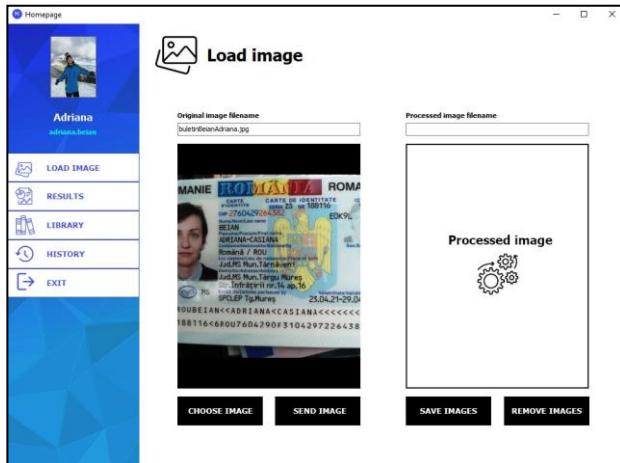


Figura 7.18 – Imaginea originală aleasă

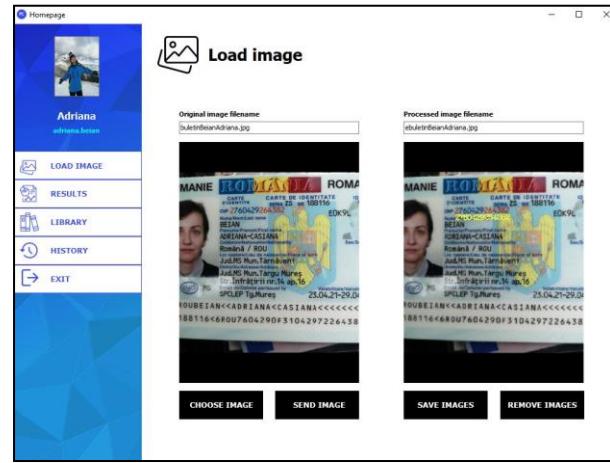


Figura 7.19 – Imaginea a fost procesată

7.2.5.3. Results

1. După ce se încheie procesul de detecție al cifrelor CNP-ului, există opțiunea de a se apăsa butonul “RESULTS” din meniu din partea stângă a fereastrăi principale.
2. Pe coloana “Images” se vor afișa imaginile cu fiecare cifră detectată din CNP.
3. Pe coloana “Expected results”, vor fi afișate cifrele reale care formează CNP-ul.
4. Pe coloana “Results”, vor fi afișate rezultatele în urma detecției CNP-ului. Cu verde vor fi încadrate rezultatele corecte, iar cu roșu, cele gresite.

Results		
Images	Expected results	Results
2	2	2
7	7	7
6	6	6
0	0	0
4	4	4

Figura 7.20 – Fereastra ’Results’

7.2.5.4. Library

1. Se va apăsa butonul “LIBRARY” din meniul din partea stângă a ferestrei principale.
2. Imaginele salvate vor fi afișate în două tabele distincte: “Original images” și “Processed images”.
3. Pentru a mări imaginile, se va apăsa pe butonul “OPEN IMAGE”.

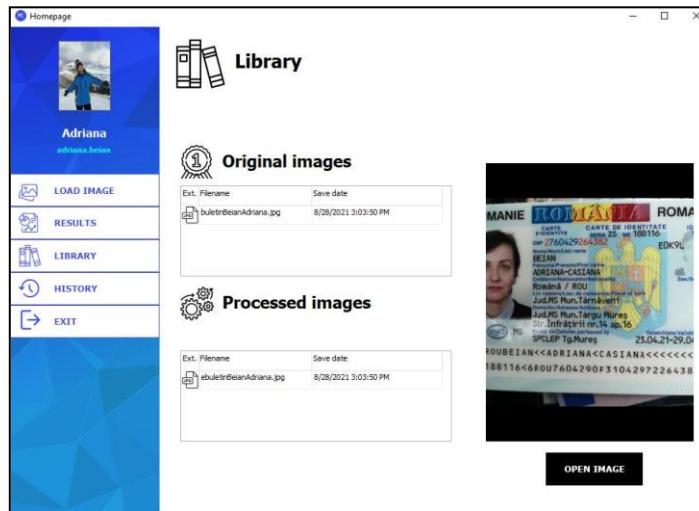


Figura 7.21 – Fereastra ’Library’

7.2.5.5. History

1. Se va apăsa butonul “HISTORY” din meniul din partea stângă a ferestrei principale.
2. Se va afișa o tabelă cu istoricul evenimentelor ce au avut loc în aplicație.

Icon	Action	Original filename	Edited filename	Save date
✉	Choose image	buletinBelanAdriana.jpg	-	8/28/2021 2:27:00 PM
✉	Choose image	buletinBelanAdriana.jpg	-	8/28/2021 2:56:07 PM
💾	Save images	buletinBelanAdriana.jpg	ebuletinBelanAdriana.jpg	8/28/2021 3:03:50 PM

Figura 7.22 – Fereastra ’History’

7.2.6. Funcționalități oferite de aplicația VHDL

În cazul în care utilizatorul trimite o imagine la placă, prin intermediul aplicației desktop, programul VHDL va permite afișarea unor informații ale rețelei neuronale artificiale, cu ajutorul switchurilor, ledurilor și butoanelor. Valorile afișate pe SSD vor fi în hexazecimal (mai puțin CNP-ul care va fi afișat în zecimal, folosind un barrell shifter), iar valorile de pe led-uri, în binar. Printre aceste informații se numără:

1. Afișarea valorilor din memoria BRAM destinată input-urilor.
 - Se va afișa, pe SSD, valoarea counter-ului destinat indexului array-ului care conține input-urile rețelei neuronale artificiale.
 - Pe leduri vor fi afișate valorile input-urilor.
 - Counter-ul va fi incrementat prin apăsarea butonului din stânga.
 - Switch-urile 15,14,13,12,11 vor avea valoarea: 01001.
 - Switch-urile 10,9,8,7 vor avea valoarea 0010.
2. Afișarea valorilor din memoria BRAM destinată weights-urilor.
 - Se va afișa, pe SSD, valoarea counter-ului destinat indexului array-ului care conține weights-urile rețelei neuronale artificiale.
 - Pe leduri vor fi afișate valorile weights-urilor.
 - Counter-ul va fi incrementat prin apăsarea butonului din dreapta.
 - Switch-urile 15,14,13,12,11 vor avea valoarea: 01010.
 - Switch-urile 10,9,8,7 vor avea valoarea 0011.
3. Afișarea numărului de input-uri.
 - Numărul de input-uri va fi afișat pe SSD;
 - Switch-urile 15,14,13,12,11 vor avea valoarea: 00111.
 - Numărul de input-uri ar trebui să fie de fiecare dată 7488, valoarea zecimală, respectiv 1d40, valoarea în hexazecimal.
4. Afișarea numărului de weights-uri.
 - Numărul de weights-uri va fi afișate pe SSD;
 - Switch-urile 15,14,13,12,11 vor avea valoarea: 01000.
 - Numărul de weights-uri ar trebui să fie de fiecare dată 5760, valoarea zecimală, respectiv 1680, valoarea în hexazecimal.
5. Afișarea cifrelor care formează CNP-ul.
 - Cele 13 cifre care formează CNP-ul vor fi afișate pe SSD, folosind un barrell shifter.
 - Fiecare cifră se va deplasa cu o poziție la stânga, o dată la 2 secunde.
 - Switch-urile 15,14,13,12,11 vor avea valoarea: 10001.
6. Afișarea output-urilor rețelei neuronale artificiale pentru fiecare cifră din CNP.
 - Switch-urile 15,14,13,12,11 vor avea valoarea: 10010, pentru a afișa pe SSD counter-ul corespunzător indexului array-ului care conține cele 130 de output-uri.
 - Switch-urile 10,9,8,7 vor avea valoarea 0111
 - Se va apăsa butonul de jos pentru a incrementa counter-ul.
 - Valorile celor 10 output-uri pentru fiecare cifră din CNP vor fi afișate pe leduri.

Capitolul 8. Concluzii

Proiectul de față poate fi considerat un punct de pornire pentru crearea unei aplicații sau a unei suite de aplicații care să ajute în domeniul digitalizării birocrației. Acest domeniu a luat amploare în ultimii ani, atât pe plan mondial, cât și național și mai ales local. Aspecte ale digitalizării birocrației pe toate cele 3 planuri au fost descrise în capitolul de introducere.

De asemenea, diferite componente sau aspecte ale aplicației ar putea fi considerate un punct de pornire sau o sursă de inspirație pentru studenți, programatori sau ingineri din diferite nișe/arii ale programării. Aplicația de față îmbină concepte din domenii precum inteligență artificială, aplicațiile desktop, baze de date, programarea hardware. Urmează a fi descrise principalele aspecte pozitive implementate în suita de aplicații care formează partea practică a lucrării de licență, pe diferite componente ale aplicației.

Rețea neuronală artificială VHDL

- Implementarea protocolului UART pentru transmiterea și receptia datelor.
- Reprezentarea numerelor în fixed point, pe 16 biți.
- Folosirea memoriei BRAM pentru salvarea input-urilor și a modelului rețelei.
- Folosirea unui FSM pentru a clasifica cifrele și crearea unui algoritm în acest sens.
- Crearea unui protocol de transmisie al datelor cât mai rapid, cu costul creșterii complexității algoritmului de preluare și prelucrare a datelor.

Rețea neuronală conoluțională Python

- Implementarea unei modele cu o acuratețe de peste 90%.
- Abilitatea de a prelua informații utile cu privire la modelul aplicației, indiferent de layer, informații care au stat la baza testării funcționării rețelei neuronale artificiale din VHDL.

Script OCR

- Crearea unui sistem prin care script-ul preia parametrii necesari pentru redimensionarea imaginii originale (parametrii de redimensionare) sau threshold-ul pentru binarizare, având scopul de a facilita procesul de detecție al cifrelor care formează codul numeric personal.

Aplicație automatizare

- Crearea unui protocol de transmisie a fișierelor, care include optimizarea timpilor de așteptare, pentru a asigura o procesare cât mai rapidă, și în același timp sigură, a fișierului de input și a fișierului cu modelul.

Aplicație desktop

- Folosirea cât mai eficientă a obiectelor destinate conexiunii cu baza de date, prin utilizarea unui număr restrâns de query-uri, pe care pot avea loc operațiile CRUD.
- Încercarea de a oferi un aspect comercial aplicației, prin crearea unor butoane personalizate, adăugarea background-urilor create folosind diferite tool-uri precum GIMP sau Adobe XD.

Aceste aspecte pozitive ale aplicației ar putea constitui o sursă de inspirație pentru programatorii care doresc implementarea unor sisteme care să conțină componente asemănătoare cu cele folosite în lucrarea de față. De exemplu, crearea unor aplicații care necesită comunicarea calculatorului cu placa FPGA pentru a rezolva o problemă în timp real. Calculatorul ar putea trimite informații la placă. Placa ar putea prelua și prelucra datele, iar răspunsul ar fi trimis înapoi la calculator. În acest sens, comunicarea UART din lucrarea de față ar acoperi această necesitate.

De asemenea, reprezentarea numerelor în fixed point pe 16 biți pe FPGA reprezintă o soluție bună pentru implementarea rețelelor neuronale artificiale sau convoluționale, sau a algoritmilor de procesare a imaginilor. Pixelii unei imagini pot fi ușor reprezentați folosind fixed point representation.

Un alt aspect important este folosirea BRAM-urilor pentru date de dimensiuni mari. În lucrarea de față, fișierul de input are 7488 valori, în timp ce fișierul cu modelul are 5760 valori. Salvarea acestor valori în memorii RAM ar fi dus la implementarea unui protocol dificil de transmisie al datelor și ar fi dus la folosirea inopportună a resurselor hardware.

Implementarea automatului cu stări finite și a procedurilor corespunzătoare fiecărei stări ar putea fi folosite în diferite aplicații care necesită FSM-uri. Folosirea FSM-urilor este comună în aplicațiile hardware, aşadar această implementare ar putea sta la baza multor aplicații.

Protocolul de transmitere al datelor este unul simplu. Sunt trimise două fișiere, un fișier de input și un fișier cu weights-uri. Această simplitate a procesului de transmitere a datelor la placa de dezvoltare vine cu un cost, acela al implementării unui algoritm dificil de preluare și prelucrare a datelor din punct de vedere hardware. Crearea unui protocol de transmitere complex, cu un fișier de input pentru fiecare cifră și un fișier cu modelul rețelei ar fi condus la implementarea unui algoritm mai simplu de preluare și prelucrare a datelor. Prin urmare, orice decizie de implementare constituie, până la urmă, un tradeoff între complexitatea algoritmilor și resursele necesare.

Rețea neuronală convoluțională are o acuratețe de 98% pe dataset-ul de test. Astfel, modelul rețelei neuronale cu 3 layere de convoluții, 2 layere de max-pooling și o rețea neuronală artificială complet conectată este un model care poate constitui o sursă de inspirație pentru programatorii dornici să folosească rețea neuronală în diferite aplicații. De exemplu, o aplicație care recunoaște numerele de telefon scrise de mână ar putea folosi modelul rețelei neuronale convoluționale implementată în lucrarea de față.

Imaginiile cu cele 13 cifre ale CNP-ului, având dimensiunea de 28X28 pixeli, au fost, pe rând, input în rețea neuronală convoluțională. Rezultatele de pe urma aplicării layer-elor de convoluție și max-pooling au compus fișierul de input transmis la rețea neuronală artificială. Modelul rețelei neuronale artificiale complet conectată a fost salvat, la rândul lui, în fișiereul de weights. Astfel, a fost implementat un sistem de extragere a datelor din diferite layere ale rețelei neuronale convoluționale.

Scriptul OCR are rolul de a detecta cifrele CNP-ului de pe actul de identitate. Pentru a detecta cifrele, este nevoie, în prealabil, de testarea imaginii cu buletinul și încercarea de a modifica parametrii de redimensionare a imaginii, respectiv threshold-ul pentru binarizare. Cu parametrii determinați și salvați în baza de date, actele de identitate pot fi procesate iar cifrele CNP-ului pot fi detectate. Desigur, această variantă nu este optimă. Ideal ar fi fost ca preprocesarea imaginilor cu actul de identitate să fie făcută corect și automat, fără a fi necesare încercările manuale din prealabil. Cu toate acestea, sistemul este funcțional, iar această modalitate de a implementa OCR-ul poate fi folosită pentru a detecta caracterele din imagini.

Programul care permite automatizarea procesului de transmitere a celor două fișiere (input și model) la placa de dezvoltare FPGA a fost conceput în aşa fel încât timpul de aşteptare să fie cât mai redus. Timpul de aşteptare redus este dat și de implementarea protocolului cu două fișiere în VHDL, și implicit folosirea memoriei BRAM. În ceea ce privește automatizarea, cu toate că s-a dorit un timp de aşteptare cât mai redus, a fost necesară și inserarea unor timpi pentru a se asigura că evenimentele definite vor avea loc. De exemplu, a fost inserat un timp de aşteptare de 3 secunde între apăsarea butonului “Send file” din HTerm și inserarea denumirii fișierului selectat. A fost necesară inserarea timpului de aşteptare, pentru că aplicația, de multe ori, nu mai încărca fișierul dorit, datorită multitudinii de task-uri executate succesiv. Timpul de aşteptare de 3 secunde constituie o marjă de eroare, pentru a evita suprasolicitarea sistemului.

Aplicația desktop are un flow intuitiv pentru utilizator. User-ul se va loga cu username-ul și parola, iar în cazul în care nu este înregistrat, are opțiunea să o facă. Odată intrat în aplicație user-ul are opțiune de a încărca și trimite o imagine spre a fi procesată. După procesare, vor fi afișate rezultate procesării, cifrele detectate corect fiind încadrate, în mod intuitiv, cu verde, iar cifrele clasificate greșit, cu roșu. Utilizatorul are opțiunea de a vizualiza imaginile încărcate anterior din librărie și poate accesa istoricul aplicației. Acest flow a fost realizat fără a fi necesară folosirea multor ferestre, ci mai degrabă se bazează pe un sistem de afișare a panel-urilor într-o singură fereastră. De exemplu, fereastra “Homepage” are 4 panel-uri, câte unul pentru fiecare funcționalitate. În funcție de butonul apăsat din meniu, un panel va fi mereu vizibil, iar celelalte vor fi invizibile. Cealaltă opțiune ar fi fost deschiderea câte unei ferestre pentru fiecare funcționalitate, fapt ce ar fi dus la o experiență neplăcută pentru utilizator.

Aplicația desktop a folosit puține query-uri. S-a urmărit folosirea query-urilor de select pentru a prelua datele din baza de date. Ulterior, s-a ținut cont de dinamisul oferit de Delphi pentru a insera, update sau șterge informații din baza de date. Nu s-au folosit query-uri separate pentru inserare, modificare sau ștergere. Modalitatea de a implementa relația aplicației desktop cu baza de date poate constitui un model pentru programatorii dormici de a folosi Delphi și de a profita de funcționalitățile și avantajele pe care le conferă.

S-a încercat conferirea unui aspect comercial, user-friendly, aplicației. Din acest motiv, imaginile de fundal din ferestrele de logare și cea principală au fost create folosind un tool specializat, numit GIMP. De asemenea, butoanele din aplicație au fost create folosind Adobe Experience Design. Delphi nu permite folosirea butoanelor cu imagini sau a culorilor pe fundalul butoanelor. Acest fapt a fost dejucat de implementarea unei clase separate pentru butoane, care permite oferirea culorilor de fundal butoanelor și adăugarea imaginilor pe butoane. Butoanele au fost implementate folosind multe linii de cod, spre deosebire de butoanele default oferite de Delphi, iar comportamentul butoanelor implementate a fost descris pas cu pas. Modalitatea în care butoanele au fost implementate poate reprezenta un model pentru un programator dormic să realizeze interfețe grafice remarcabile folosind Delphi.

În cele ce urmează, vor fi prezentate posibile îmbunătățiri care ar putea fi aduse părții practice a lucrării de licență. De asemenea, vor fi descrise posibilitățile de dezvoltare ulterioară ale aplicației. Îmbunătățirile care ar putea fi aduse și posibilitățile de dezvoltare ulterioară ar face ca aplicația de față să fie cu un pas mai aproape de o posibilă aplicație comercială, sau ar sta la baza creării unei aplicații comerciale, nu neapărat pentru detecția cifrelor din CNP, ci un sistem care să detecteze caracterele alfanumerice din diferite medii și de pe diferite formate.

8.1. Posibile îmbunătățiri ale proiectului

Scopul aplicației este de a detecta cifrele care formează codul numeric personal de pe o imagine cu actul de identitate. Sistemul de față îndeplinește această cerință. Cu toate acestea, există îmbunătățiri ce pot fi aduse sistemului, pentru a crește viteza procesului de detecție al cifrelor, acuratețea clasificării, limitarea input-ului uman și per total oferirea unei experiențe mai plăcute în utilizarea aplicației. Aspectele care pot fi îmbunătățite, vor fi descrise pe componente.

Rețea neuronală artificială VHDL

- Creșterea baud rate-ului de la 115.200 la 256.000, pentru o transmitere mai rapidă a fișierului de input și a modelului.
- Reprezentarea numerelor pe 32 biți: 16 biți pentru partea întreagă, 16 biți pentru partea fracțională. Eroarea ar fi mai mică, iar posibilitatea de overflow-ului ar fi mai redusă.
- Încercarea de a implementa o rețea neuronală cu un algoritm „mai puțin secvențial”. Automatul de stări finite secvențializează implementarea. Ideală ar fi procesarea datelor concurrent, procesare care ar putea fi îmbunătățită folosind diferite tehnici pipeline.
- Introducerea unor hidden layer în rețea neuronală artificială pentru a crește acuratețea, sau chiar încercarea de a implementa o rețea neuronală conlovțională, doar forwarding, nu și backpropagation.

Script OCR

- Preprocesarea corectă a imaginilor. Imaginea originală trebuie să parcurgă un pipeline de procesare, care nu a fost implementat integral în aplicația de față.
- În cazul în care preprocesarea nu asigură detecția cifrelor din CNP, atunci ar fi ideală determinarea automată a parametrilor de configurare.
- Detecția cifrelor care formează seria și încadrarea fiecarei din cele 6 într-un bounding box. De asemenea, blurarea cifrelor CNP-ului sau a altor date cu caracter personal din actul de identitate, într-o aplicație care necesită această funcționalitate.

Aplicație automatizare

- Afisarea procesului de transmitere a fișierelor pe un alt desktop sau pe o mașină virtuală, astfel încât aplicația desktop să rămână în prim plan pe durata transmiterii fișierelor la placa de dezvoltare, și chiar să afișeze un o fereastră cu un progress bar al procentajului de transmitere al fișierelor la placă și denumirea fișierului care este în curs de trimitere.

Aplicație desktop

- Realizarea unei interfețe grafice îmbunătățite, prin rotunjirea unor butoane, pictograme colorate, folosirea unui sistem de culori care să se potrivească.
- Introducerea unei funcționalități care returnează rezultatele clasificării pentru toate cele 13 cifre ale CNP-ului de la placa de dezvoltare, scrierea lor într-un fișier și preluarea de către aplicația desktop a datelor. Aplicația ar urma să creeze un graf care constituie modelul rețelei neuronale artificiale din VHDL, și ar afișa output-urile pentru fiecare din cele 13 cifre în parte. Fiecare cifră ar urma să fie reprezentată pe un tab-sheet diferit.
- Reprezentarea automatului cu stări finite și a stărilor sale sub forma unui graf. Aplicația desktop ar urma să preia de la placă ordinea prin care s-a trecut în stări și să afișeze sub forma unui graf secvențialitatea lor. Fiecare stări din FSM i-ar corespunde un nod în graf.

8.2. Dezvoltări ulterioare

Aplicația de față folosește o rețea neuronală conoluțională și o rețea neuronală artificială, antrenate pe un dataset cu cifre scrise de mână, pentru a clasifica cifrele de tipar ale codului numeric personal de pe o imagine cu actul de identitate. Acest fapt a dus la predicția greșită a cifrei 1 de tipar, fiind tratată drept 7 de rețele neuronale implementate. Pentru a evita această eroare, una dintre posibilitățile de dezvoltare ulterioară este de a antrena o rețea neuronală conoluțională, folosind un dataset cu cifre scrise de tipar. Aceasta ar face ca acuratețea clasificării cifrelor să fie mai bună.

De asemenea, aplicația desktop ar putea fi modificată, în aşa fel încât să conțină două funcționalități majore. Prima, de a detecta cifrele de mână dintr-o imagine în care textul este scris cu mâna. Astfel, o persoană ar putea scrie un număr de telefon pe o foaie, iar aplicația ar recunoaște cifrele, pe baza rețelei neuronale antrenate cu un dataset cu cifre scrise de mână. A doua funcționalitate, ar putea detecta cifrele de tipar dintr-o imagine cu cifre scrise de tipar. Aplicația de față ar putea detecta cifrele din CNP folosind o rețea neuronală antrenată pe un dataset cu cifre scrise de tipar. Acest lucru ar face ca cifra 1 să fie clasificată corect în mai multe cazuri.

Aplicația de față ar fi putut avea o funcționalitate de a folosi camera laptopului, pentru a detecta cifrele CNP-ului, fie pe baza unei fotografii, fie în timp real, folosind înregistrarea video. La fel de bine ar fi putut fi implementată o aplicație mobile care să folosească camera telefonului pentru a detecta cifrele din CNP, la fel, folosind o imagine sau o înregistrare video în real time. Aceste funcționalități, alături de posibilele îmbunătățiri ale proiectului, descrise în subcapitolul anterior, ar putea duce la o aplicație, sau o suita de aplicații, cu un potențial comercial.

Bibliografie

- [1] "Xilinx Vivado" [Online]. Available: https://en.wikipedia.org/wiki/Xilinx_Vivado
- [2] Definiție "inteligentă" [Online]. Available: <https://dexonline.ro/definitie/inteligenta>
- [3] Somen Das," Different domains of Artificial intelligence(AI)", *somenplus.blogspot.com*, Nov.31,2020 [Online]. Available: <https://somenplus.blogspot.com/2020/11/different-domains-of-artificial.html> [Accessed Jun.13, 2021].
- [4] " Artificial Intelligence In 5 Minutes | What Is Artificial Intelligence? | AI Explained | Simplilearn" [Online]. Available: <https://www.youtube.com/watch?v=ad79nYk2keg>
- [5] " Digital Bureaucracy with Blockchain infrastructure" [Online]. Available: <https://www.digitalbureaucracy.org/>
- [6] Lidia Neagu, "Zitec lansează Ghidul de Digitalizare dedicat primăriilor din România pentru reducerea birocrației și eliminarea cozilor," *Economica.net*, Noiembrie 16, 2012. [Online], Available: https://www.economica.net/zitec-lanseaza-ghidul-de-digitalizare-dedicat-primariilor-din-romania-pentru-reducerea-birocratiei-si-eliminarea-cozilor_192738.html , [Accessed Jul. 12, 2021].
- [7] Camelia Badea, "Se intampla tot la Cluj: Performantele si ce urmeaza pentru Antonia, primul "functionar public virtual" din Romania," *Ziare.com*, Iunie 23, 2019. [Online], Available: <https://ziare.com/social/administratia/se-intampla-tot-la-cluj-antonia-primul-functionar-public-virtual-din-romania-trece-in-etapa-de-holograma-1566394/> , [Accessed Jul. 12, 2021].
- [8] "Machine Learning on FPGAs: Training the Neural Network" [Online]. Available: <https://www.youtube.com/watch?v=YgA7LKUofY>
- [9] "Machine Learning on FPGAs: Circuit Architecture and FPGA Implementation" [Online]. Available: <https://www.youtube.com/watch?v=Qgjawf20v7Y&t=302s>
- [10] "Neural Networks on FPGA: Part 1: Introduction" [Online]. Available: https://www.youtube.com/watch?v=rw_JITpbh3k&t=982s
- [11] "Neural Networks on FPGA: Part 2: Designing a Neuron" [Online]. Available: https://www.youtube.com/watch?v=a2wOjxRf_xg&t=290s
- [12] " Neural Networks on FPGA: Part 3: Activation Functions" [Online]. Available: <https://www.youtube.com/watch?v=Tsu3q2iyab4&t=36s>
- [13] " Neural Networks on FPGA: Part 5: Designing the Layers" [Online]. Available: <https://www.youtube.com/watch?v=FDefiTq14T0&t=614s>

- [14] Maria Homann, "What is Desktop Automation?", *leapwork.com*, Available: <https://www.youtube.com/watch?v=juRKu9cBwQ0&t=974s> [Accessed Jun.13, 2021].
- [15] " How to Automate Windows Based Application using Winium and Selenium" [Online]. Available: <https://www.youtube.com/watch?v=juRKu9cBwQ0&t=974s>
- [16] " The Comprehensive Guide to Optical Character Recognition (OCR)" [Online]. Available: <https://moov.ai/en/blog/optical-character-recognition-ocr/>
- [17] Kh Tohidul Islam, Ghulam Mujtaba, Henry Friday Nweke, Dr. Ram Gopal Raj, " Handwritten Digits Recognition with Artificial Neural Network", in Proc. of the International Conference on Engineering Technologies and Technopreneurship (ICE2T 2017), 18-20 September 2017, Kuala Lumpur, Malaysia
- [18] " Install Anaconda Python, Jupyter Notebook And Spyder on Windows 10" [Online]. Available: <https://www.youtube.com/watch?v=5mDYijMfSzs&t=610s/>
- [19] M. Negru și F. Oniga, *ARHITECTURA CALCULATOARELOR Îndrumător de laborator*, Editura UTPRESS Cluj-Napoca, 2019
- [20] " Anaconda" [Online]. Available: <https://www.anaconda.com/>
- [21] " Download Xilinx" [Online]. Available: <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html/>
- [22] " Download XAMPP" [Online]. Available: <https://www.apachefriends.org/download.html/>
- [23] " Download HTerm" [Online]. Available: <http://der-hammer.info/pages/terminal.html/>
- [24] " Everyday applications of AI | AI in everyday life" [Online]. Available: <https://www.youtube.com/watch?v=MpR6JZdQ4B0>
- [25] "Infographic: How AI is Being Deployed Across Industries" [Online]. Available: <https://www.robotsbusinessreview.com/ai/infographic-how-ai-is-being-deployed-across-industries/>
- [26] " Adaptability" [Online]. Available: <https://en.wikipedia.org/wiki/Adaptability>
- [27] " Availability" [Online]. Available: <https://en.wikipedia.org/wiki/Availability>
- [28] " Data integrity" [Online]. Available: https://en.wikipedia.org/wiki/Data_integrity
- [29] " Durability" [Online]. Available: <https://en.wikipedia.org/wiki/Durability>
- [30] "System integration" [Online]. Available: https://en.wikipedia.org/wiki/System_integration

- [31] "Interoperability" [Online]. Available: <https://en.wikipedia.org/wiki/Interoperability>
- [32] "Maintainability" [Online]. Available: <https://en.wikipedia.org/wiki/Maintainability>
- [33] "What is software operability?" [Online]. Available: <https://confluxdigital.net/what-is-operability>
- [34] "Computer performance" [Online]. Available:
https://en.wikipedia.org/wiki/Computer_performance
- [35] "Software portability" [Online]. Available:
https://en.wikipedia.org/wiki/Software_portability
- [36] "Reliability engineering" [Online]. Available:
https://en.wikipedia.org/wiki/Reliability_engineering
- [37] "Scalability" [Online]. Available: <https://en.wikipedia.org/wiki/Scalability>
- [38] "DEFINITION compatibility" [Online]. Available:
<https://whatis.techtarget.com/definition/compatibility>
- [39] "Usability" [Online]. Available: <https://en.wikipedia.org/wiki/Usability>
- [40] "Computer security" [Online]. Available: https://en.wikipedia.org/wiki/Computer_security
- [41] "Access +5.3M vector icons & stickers" [Online]. Available: <https://www.flaticon.com/>
- [42] "Convolutional Neural Network (CNN)" [Online]. Available:
<https://www.tensorflow.org/tutorials/images/cnn>