

Acceleratoare pentru aplicații de inteligență artificială

Scopul acestei lucrări este de a ilustra modul în care este posibilă implementarea în timp real a prelucrărilor necesare în aplicații ce utilizează rețele neuronale cu ajutorul structurilor hardware specializate integrate în arhitectura microcontrolerelor de ultimă generație. Se exemplifică o rețea neuronală convoluțională (Convolutional Neural Network – CNN) folosită pentru clasificarea imaginilor și se prezintă modul de dezvoltare a unei aplicații cu CNN și rezultatele obținute.

1. Cerințe de calcul și de memorie pentru aplicații de inteligență artificială

Aplicațiile de inteligență artificială folosesc rețele neuronale care, în principiu, au structura din figura 1:

Deep Neural Networks

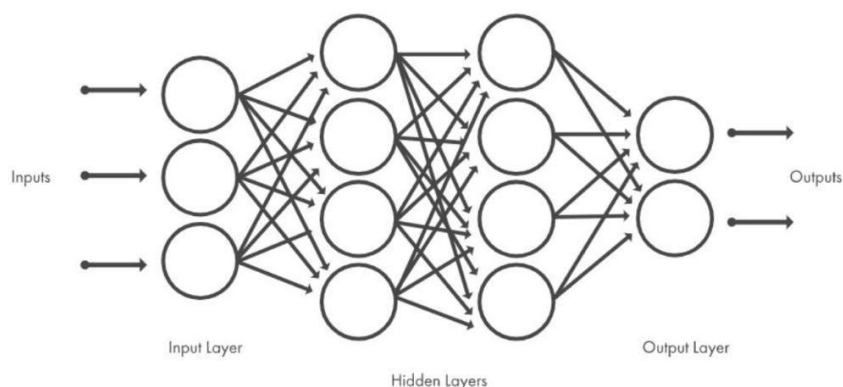
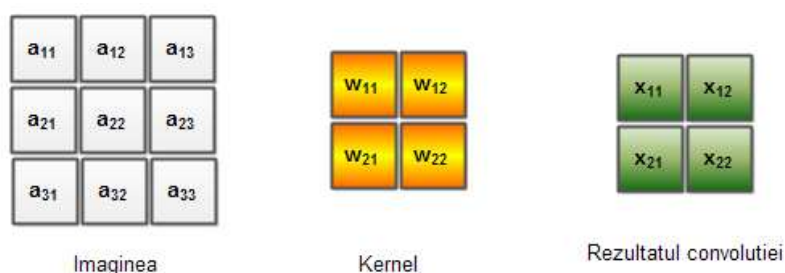
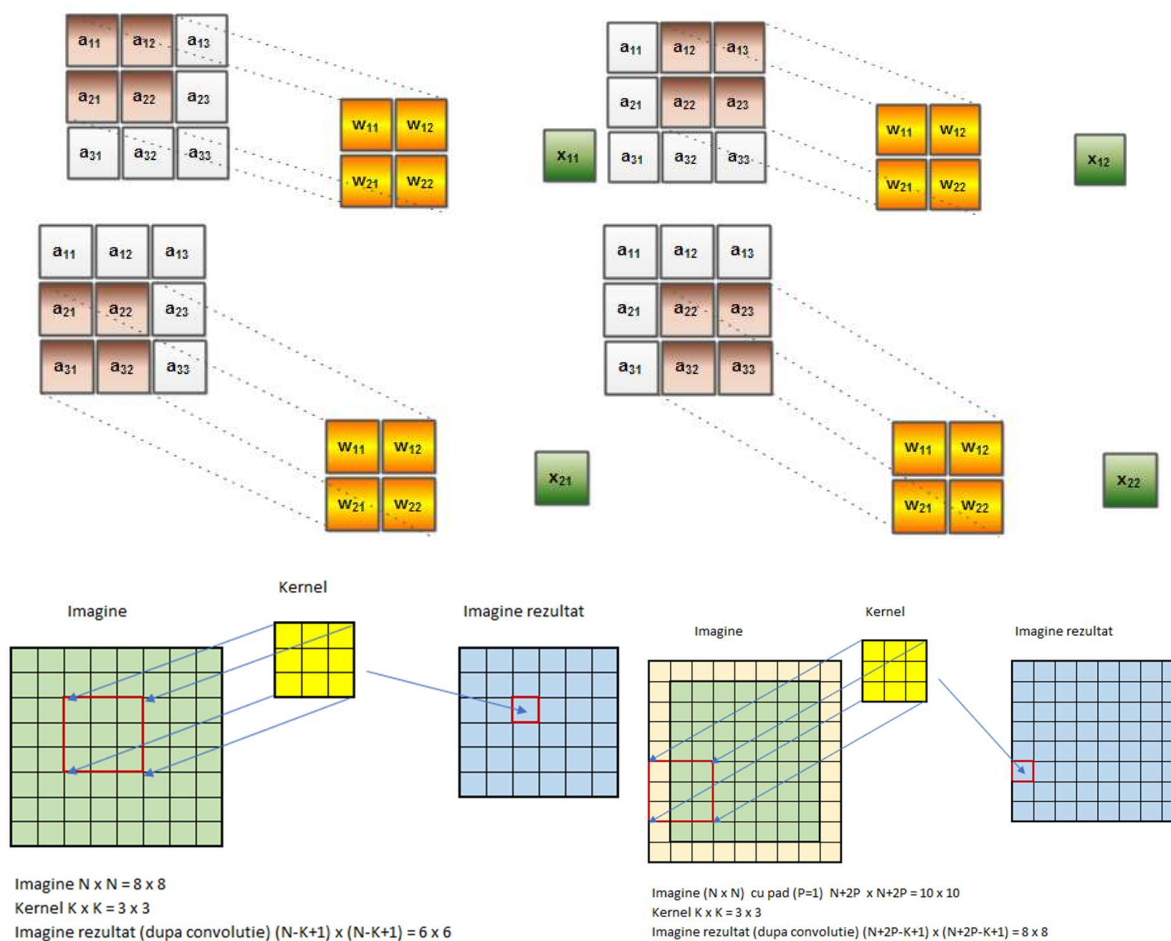


Figura 1. Rețea neuronală [1]

O astfel de rețea are în componență un nivel (*layer*) de intrare, mai multe niveluri ascunse (numărul mare al acestor niveluri va permite extragerea unui număr mai mare de caracteristici (*feature*) ale intrărilor și un nivel de ieșire care reprezintă rezultatul (de exemplu o decizie). Operațiile necesare într-o astfel de rețea sunt sume de produse (într-un număr foarte mare), convoluții 1D sau 2D, alte funcții pentru validarea (activarea) rezultatelor între niveluri. De exemplu, ieșirea unui nod j poate fi calculată ca $out(j) = \sum_{i=1}^N in(i) \cdot w(i, j)$, unde $in(i)$ este intrarea i , iar $w(i, j)$ este ponderea dintre nodul i și nodul j . Există N noduri care contribuie la stabilirea valorii nodului j . Alte operații în rețelele neuronale (pentru prelucrarea imaginilor) pot fi: convoluția 2D (cu sau fără adăugarea de elemente pe linii și coloane ($pad=0$ sau $pad=1$)) și cu pasul de deplasare al nucleului (*kernel*), $stride = 1$ și MaxPool (o operație de reducere a dimensiunii imaginilor cu $stride = 2$ și dimensiunea $kernel = 2$), reprezentate în figurile 2 și 3.





Convoluția fără și cu padding - pad = (1,1), stride = (1,1)

Figura 2. Operația de convoluție 2D

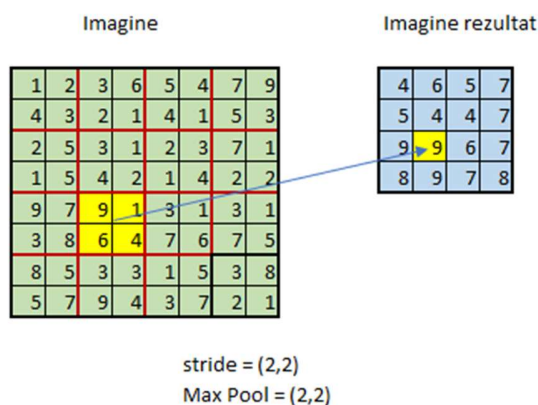


Figura 3. Operația MaxPool

Pentru operația MaxPool, imaginea se împarte în blocuri (2,2) și se calculează maximul acestor elemente care va reprezenta un element în imaginea rezultat. Pentru o rețea neuronală trebuie determinate toate ponderile și elementele nucleului (*kernel*) care sunt utilizate pentru determinarea fiecărei ieșiri. Determinarea acestora se face în faza de antrenare (*training*) care presupune ca se considera o mulțime cu un număr mare de elemente de intrare pentru care se știe rezultatul. Prin iterații succesive se calculează toate elementele rețelei, urmărindu-se minimizarea unei funcții de eroare (diferența în medie pătratică dintre rezultatul dorit și cel determinat de rețeaua neuronală).

Pentru rețele neuronale cu un număr mare de niveluri, numărul ponderilor este foarte mare, ceea ce necesita o capacitate mare de memorie. De asemenea, volumul de calcul este mare (ceea ce poate întârzia decizia) si precizia efectuării calculului poate afecta rezultatul final, in raport cu cel obținut cu precizie infinita).

2. Acceleratoare pentru rețele neuronale convolutionale

Circuitul MAX78000 are in compunere si un accelerator pentru rețele neuronale convolutionale (Convolutional Neural Network - CNN) care poate implementa operațiile de convoluție 1D, 2D si MaxPool prezentate in figura 4a.[7]

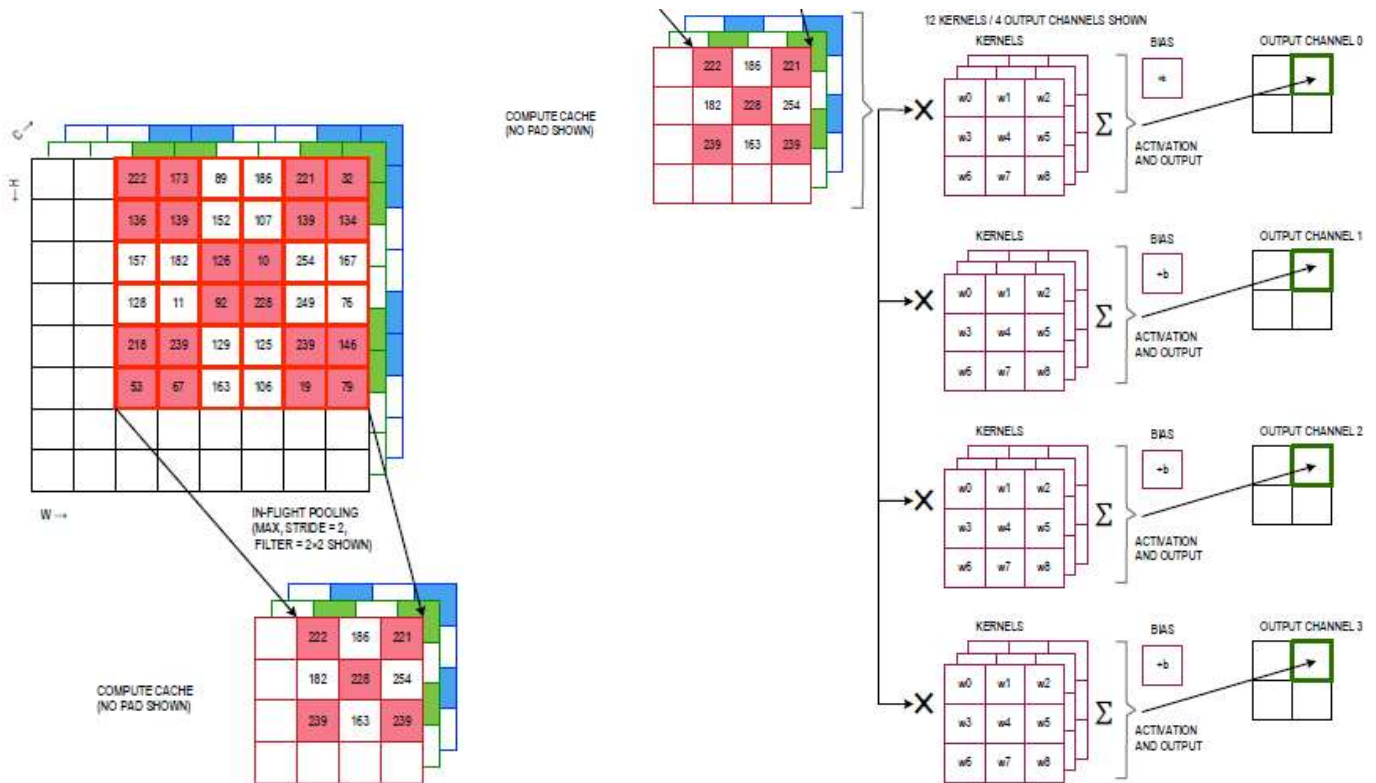


Figura 4a. Operațiile CNN – MAX78000

Acceleratorul CNN are structura din figura 4b [6, 7].

Exista 4 sisteme multiprocesor (cuadrante) fiecare cu cite 16 procesoare. Procesoarele din cadrante partajează memorii rapide specializate pentru stocarea datelor de intrare, a ponderilor rețelei neurale si a rezultatelor. Pentru a crește viteza de prelucrare, exista si memorii FIFO care permit preluarea datelor de intrare in flux continuu (pe modul *streaming*). Acest mod se bazează pe suprapunerea operațiilor de acces la memorie cu operații de calcul si este prezentat in secțiunea de implementare a rețelei neuronale.

Paralelismul prelucrărilor se obține prin posibila utilizare in paralel a celor 64 de procesoare precum si in utilizarea unor memorii multiple si rapide pentru accesarea datelor necesare (date de intrare, ponderi). Procesoarele nu au o structura foarte complexa; sunt de fapt elemente de procesare specializate in realizarea sumelor de produse. Exista posibilitatea de concatenare a rezultatelor parțiale (obținute intr-un grup de procesoare) si de a efectua operații pe biți. Operațiile de tip *maxpool*, activare si pe biți sunt integrate cu operațiile de convoluție (nu necesita timp de prelucrare suplimentar).

Programarea CNN este complexa deoarece exista un număr foarte mare de registre de control. Din acest motiv, programarea (generarea codului) se face automatizat cu unelte specializate care integrează antrenarea rețelei CNN, cuantizarea ponderilor, generarea codului in limbaj C pentru programarea CNN.

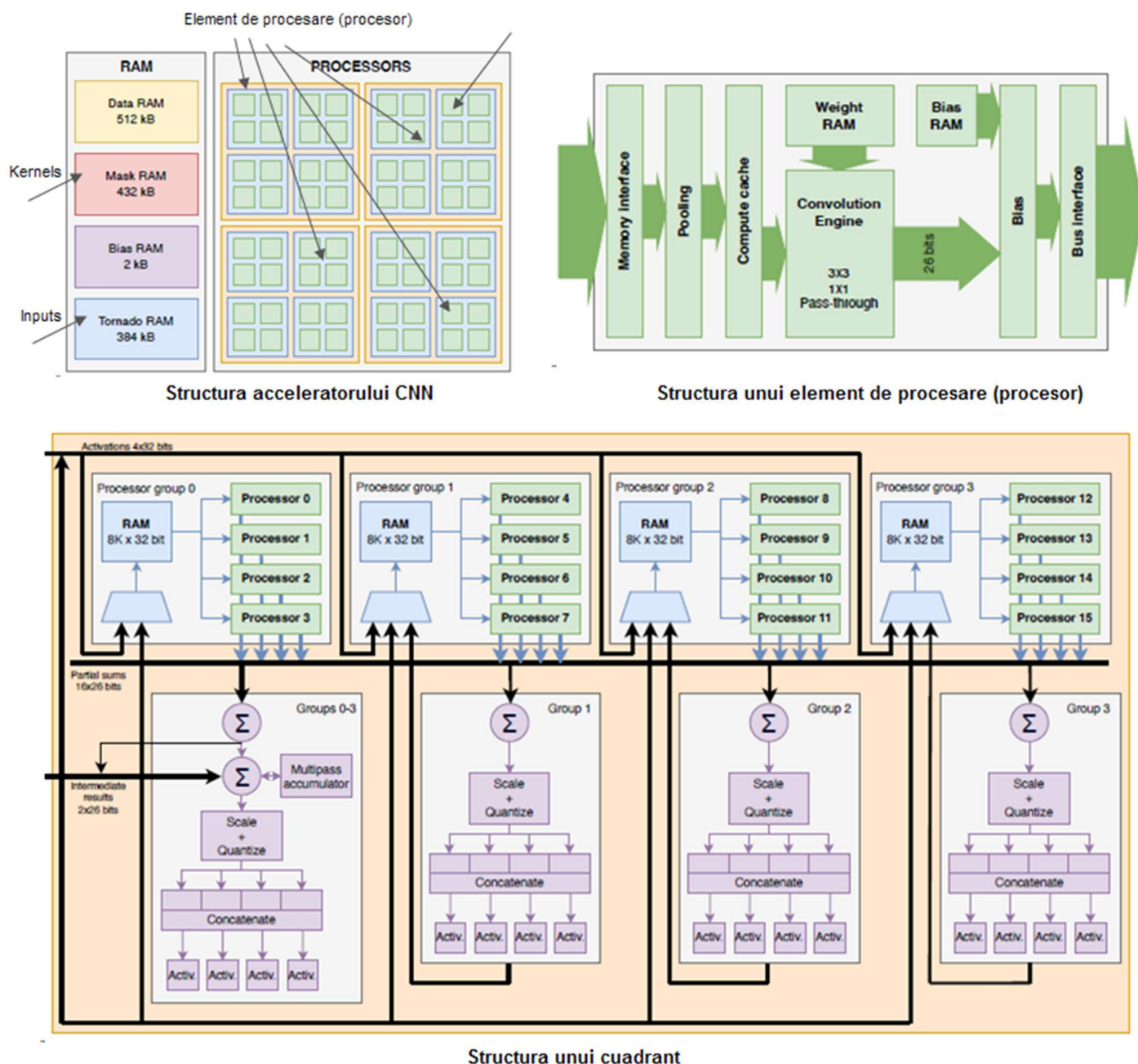


Figura 4b. Structura acceleratorului CNN – MAX78000

Aplicațiile ce utilizează acceleratorul CNN integrează codul de programare efectivă a CNN cu programarea celorlalte resurse de intrare-ieșire (camera, microfon, pini IO) și cu o structură de control general a aplicației.

3. Exemplu de aplicație de clasificare cu rețele neuronale convoluționale

Se exemplifica o rețea neuronală convoluțională ca în figura 5.

Intrarea este o imagine color, cu dimensiunile 128 x 128 x 3, urmează 6 niveluri convoluționale, un nivel cu interconectare totală și un bloc de decizie pentru 2 clase. Nivelurile 0 – 6 sunt ilustrate în figurile 6 – 12.

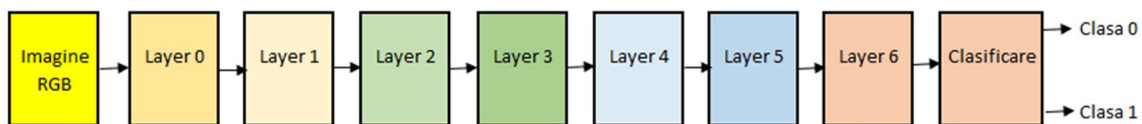


Figura 5. Retea neuronală convolutională

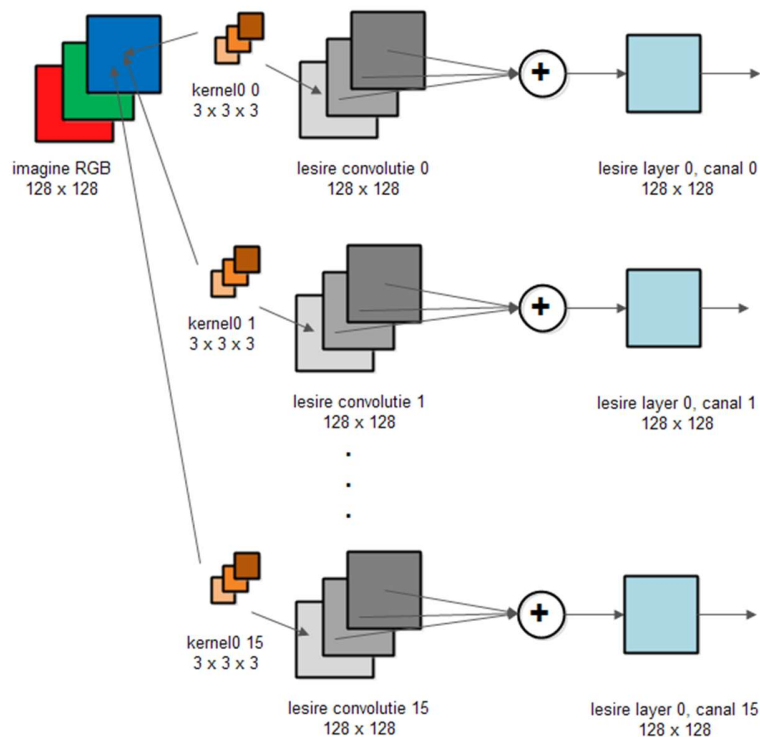


Figura 6. Layer 0

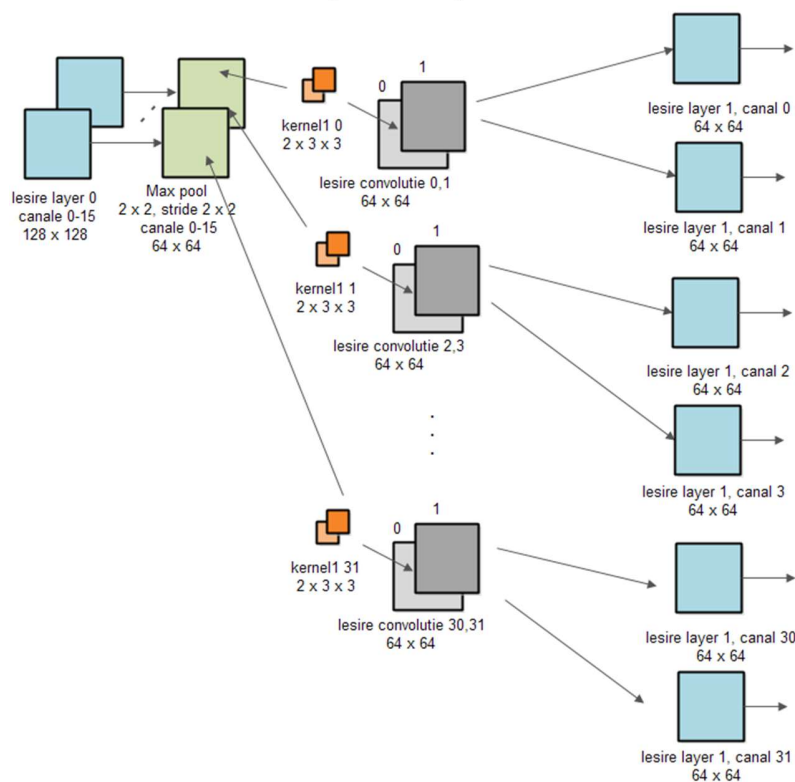


Figura 7. Layer 1

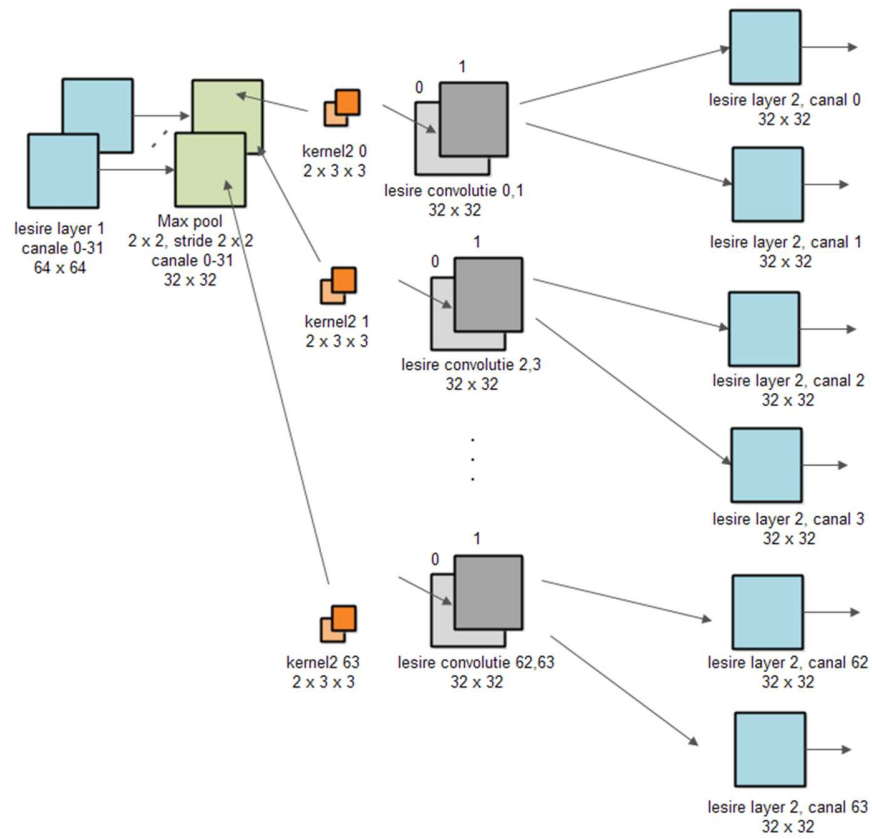


Figura 8. Layer 2

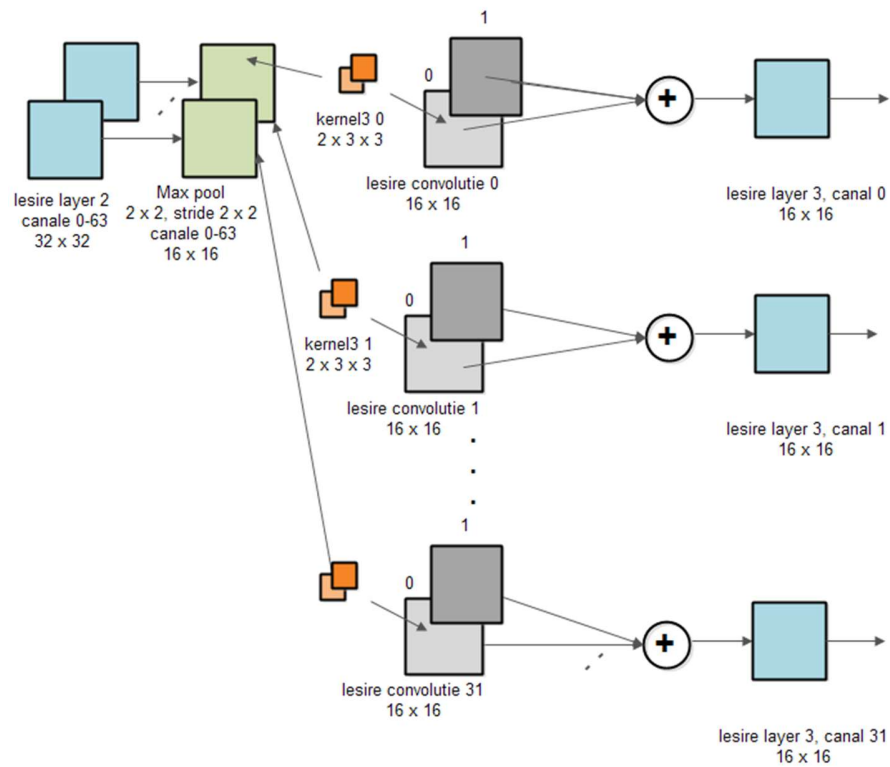


Figura 9. Layer 3

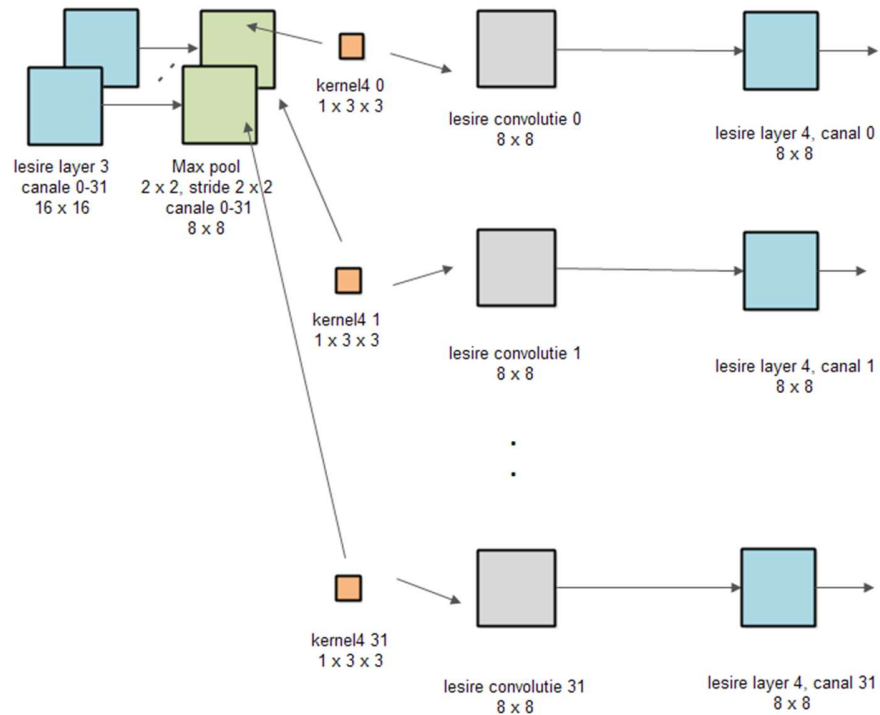


Figura 10. Layer 4

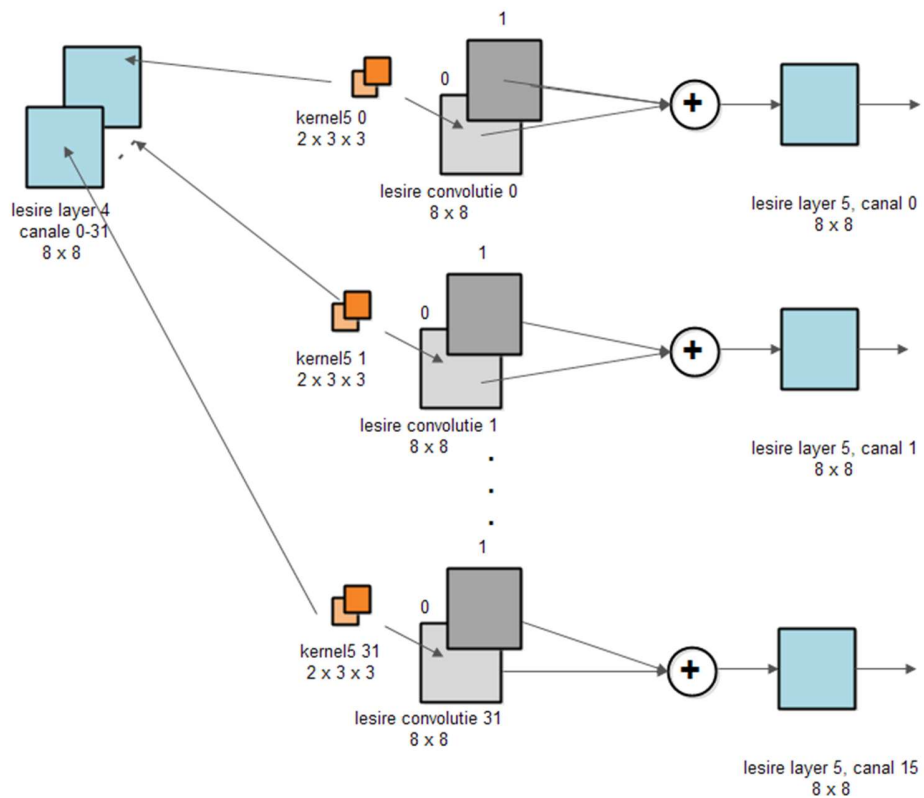


Figura 11. Layer 5

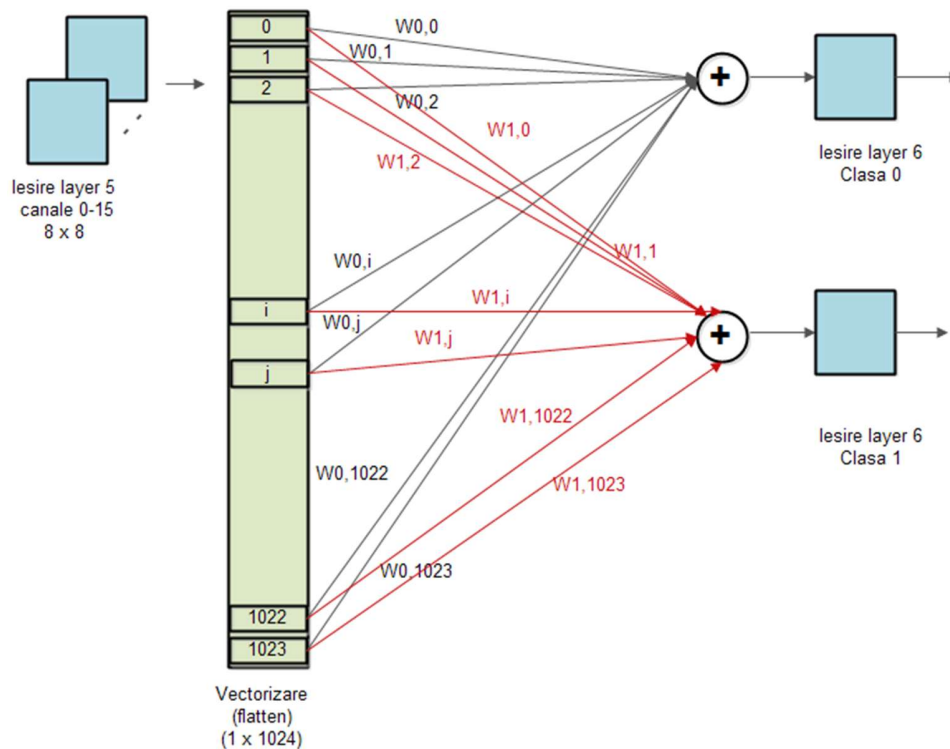


Figura 12. Layer 6

Decizia se ia prin transformarea scorurilor obținute de cele doua clase in probabilități (confidente procentuale) folosind funcția $softmax(x_i) = \frac{x_i \cdot e^{-x_i}}{\sum_{i=1}^N x_i \cdot e^{-x_i}}$.

4. Implementarea cu placa de evaluare MAX78000 Evkit

Implementarea rețelei neuronale presupune antrenarea si determinarea ponderilor (parametrilor învățabili – *learning parameters*). Acest lucru se realizează separat folosind programe specializate care generează automat si codul pentru încărcarea in memorie si configurarea registrelor de comanda ale CNN din arhitectura MAX78000 [3, 4, 5]. In figura 12b este prezentat fluxul de dezvoltare a unei aplicații cu inteligenta artificiala folosind acceleratorul CNN din MAX78000 [3,4,5].

Antrenarea rețelei neuronale se realizează in PyTorch (care poate importa imaginile necesare antrenării si in care se poate defini arhitectura rețelei neuronale – număr de straturi, definirea fiecărui strat). Antrenarea se realizează cu ponderi cuantizate pe maxim 8 biți (pentru ca ponderile sa fie compatibile cu modul de reprezentare numerica in CNN). In urma antrenării, se generează un fișier care conține ponderile (*checkpoint*). Urmează etapa de sinteza care preia un fișier de descriere a arhitecturii rețelei neuronale (*.yaml*), fișierul cu ponderi si un fișier cu date de intrare (cu imagini de test pentru verificarea corectitudinii codului). Codul generat este compilat si încărcat in memoria MAX78000 pentru a fi rulat in timp real.

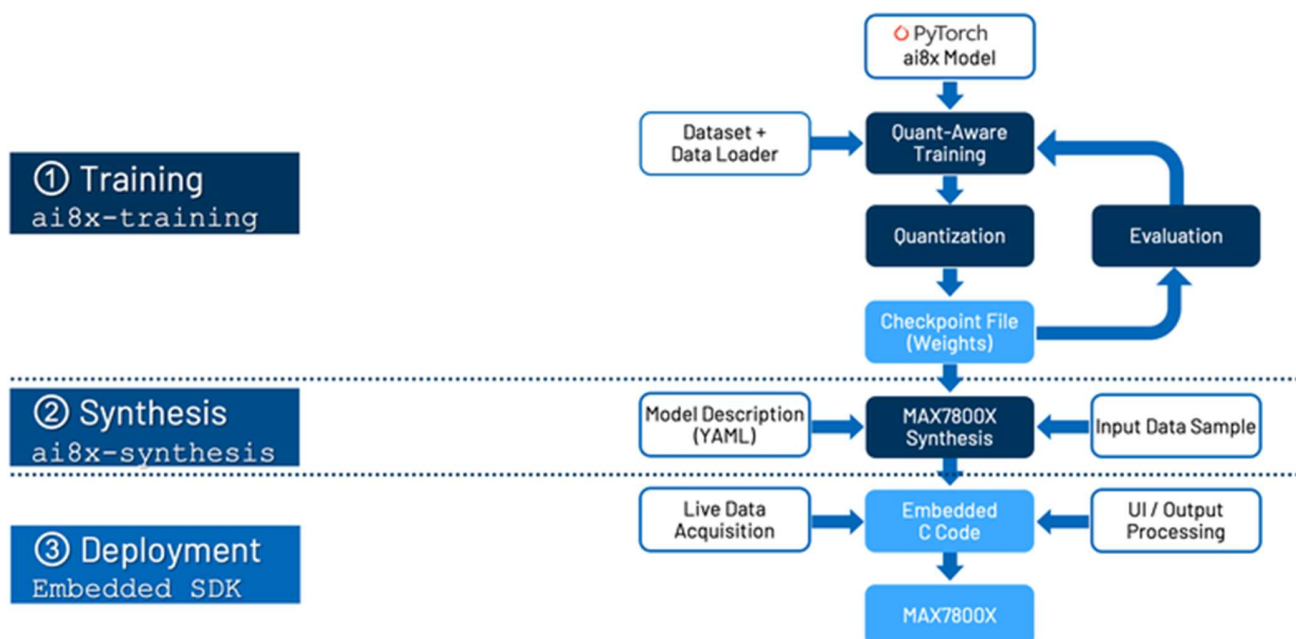


Figura 12b. Fluxul de dezvoltare a unei aplicații CNN din MAX78000 [3,4,5]

În exemplul prezentat, se considera ca toate ponderile (pentru toate straturile rețelei CNN) sunt definite și preluate de aplicație conform modului de lucru ilustrat în secțiunea 4.1. De asemenea formatul imaginii este RGB împachetat pe 4 octeți: *00, BB, GG, RR*, unde *BB* reprezintă componenta *blue*, *GG* componenta *green* și *RR* componenta *red*.

Organigrama aplicației de clasificare este ilustrată în figura 13.

Figurile 14 și 15 ilustrează procedurile de setare a modului de lucru și de start a aplicației.

Există două moduri de lucru: cu preluarea imaginilor din fișiere de test și cu preluarea imaginii de la camera. Două butoane, PB1 și PB2 asigură setarea modului de lucru și preluarea imaginii. Rezultatele obținute (decizia de clasificare, timpul de execuție, confidencele asociate fiecărei clase) sunt afișate pe un display sau pe un port serial virtual.

Organigramele din figurile 13, 14 și 15 vor fi urmărite la desfășurarea practică a lucrării, pentru a putea realiza experimentele cerute cu placa de evaluare MAX78000 Evkit.

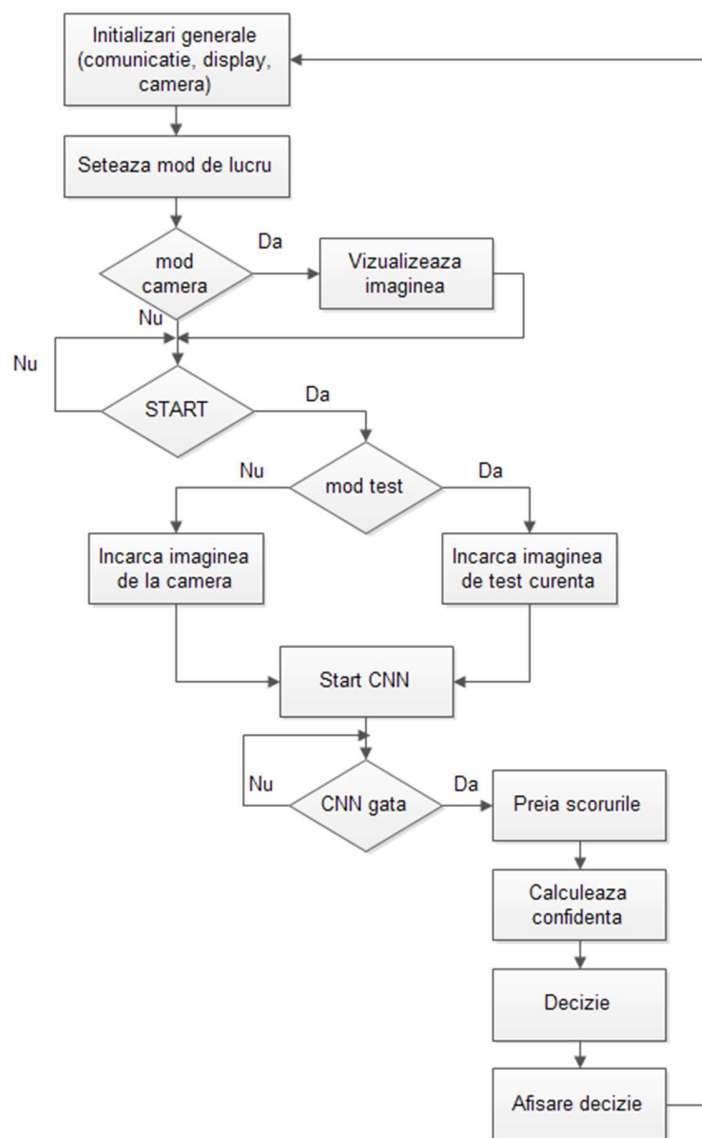


Figura 13. Organigrama aplicației de clasificare

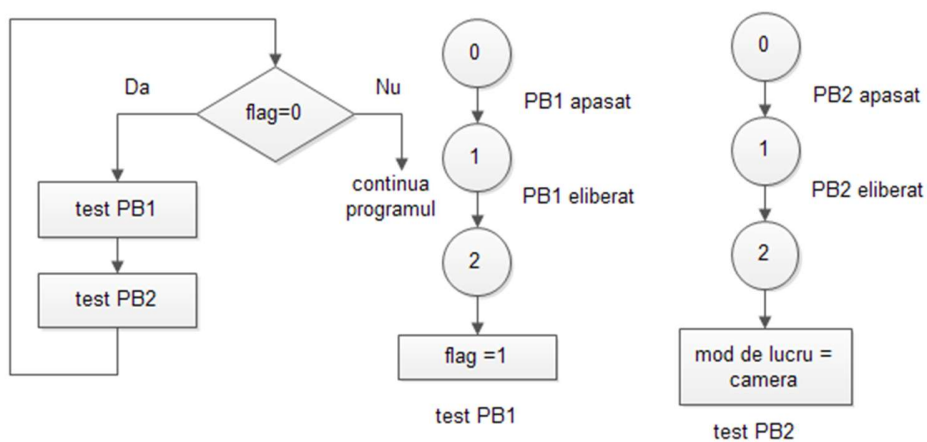


Figura 14. Organigrama procedurii de setare a modului de lucru

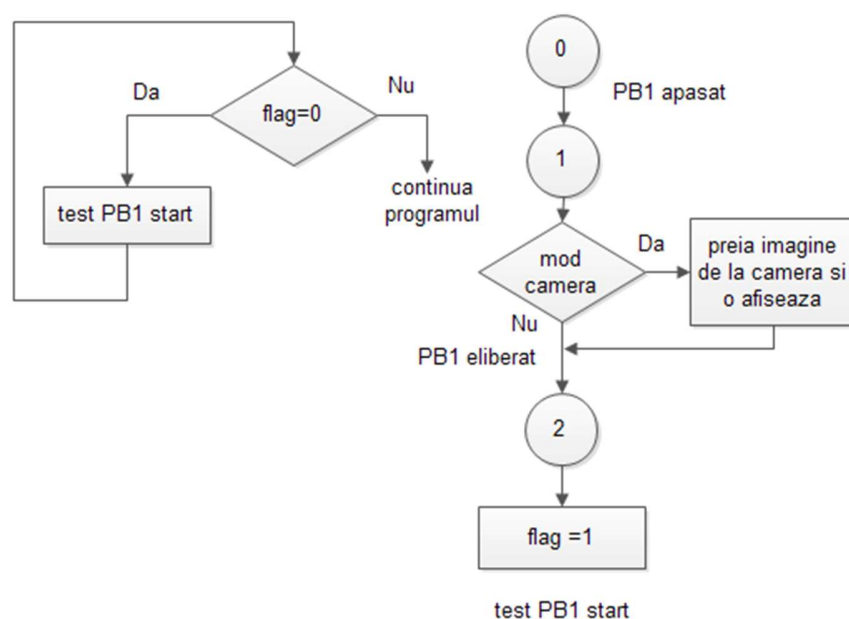
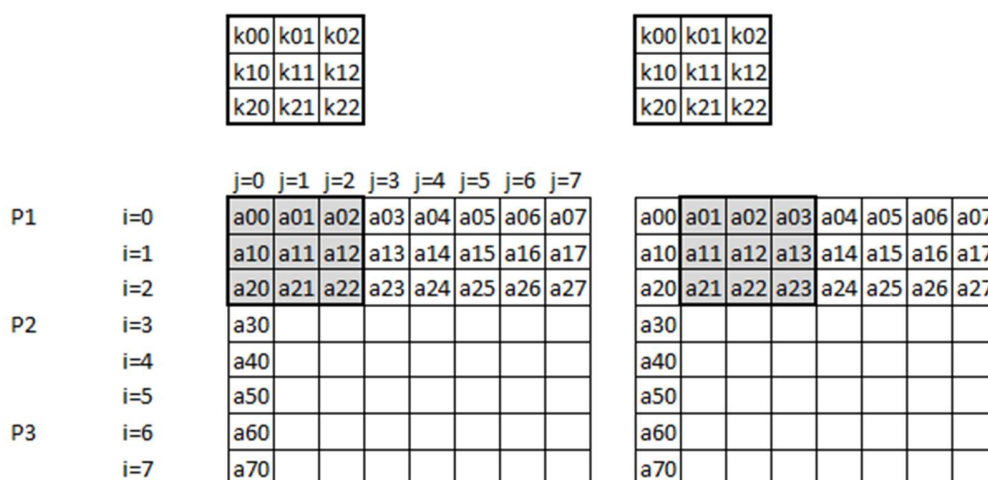


Figura 15. Organigrama procedurii de start a aplicației

Modul de implementare a convoluției (cu replicarea kernel-urilor) este ilustrată în figurile 16 și 17. Replicarea kernel-urilor crește viteza de prelucrare (există o replică a fiecărui kernel pentru fiecare procesor care prelucrează o linie a imaginii de intrare). Modul de calcul al convoluției este ilustrat în figura 16. Un exemplu de logica de comandă pentru realizarea hardware a convoluțiilor este prezentat în figura 17.



Imaginea este A

Kernel-ul B este replicat pentru fiecare procesor

Fiecare procesor calculează o linie din rezultatul convoluției C

Procesoarele funcționează în paralel

Pentru P1 (primele 2 rezultate):

A	a00	B	k00	C	c00	A	a00	B	k00	C	c00
A+1	a01	B+1	k01	C+1		A+1	a01	B+1	k01	C+1	c01
A+2	a02	B+2	k02	C+2		A+2	a02	B+2	k02	C+2	
A+3	a03	B+3	k10	C+3		A+3	a03	B+3	k10	C+3	
A+4	a04	B+4	k12	C+4		A+4	a04	B+4	k12	C+4	
A+5	a05	B+5	k13	C+5		A+5	a05	B+5	k13	C+5	
A+6	a06	B+6	k20			A+6	a06	B+6	k20		
A+7	a07	B+7	k21			A+7	a07	B+7	k21		
A+8	a10	B+8	k22			A+8	a10	B+8	k22		
A+9	a11					A+9	a11				
A+10	a12					A+10	a12				
A+11	a13					A+11	a13				
A+12	a14					A+12	a14				
A+13	a15					A+13	a15				
A+14	a16					A+14	a16				
A+15	a17					A+15	a17				
A+16	a20					A+16	a20				
A+17	a21					A+17	a21				
A+18	a22					A+18	a22				
A+19	a23					A+19	a23				
A+20	a24					A+20	a24				
A+21	a25					A+21	a25				
A+22	a26					A+22	a26				
A+23	a27					A+23	a27				

Figura 16. Organizarea prelucrărilor pentru calculul convoluției

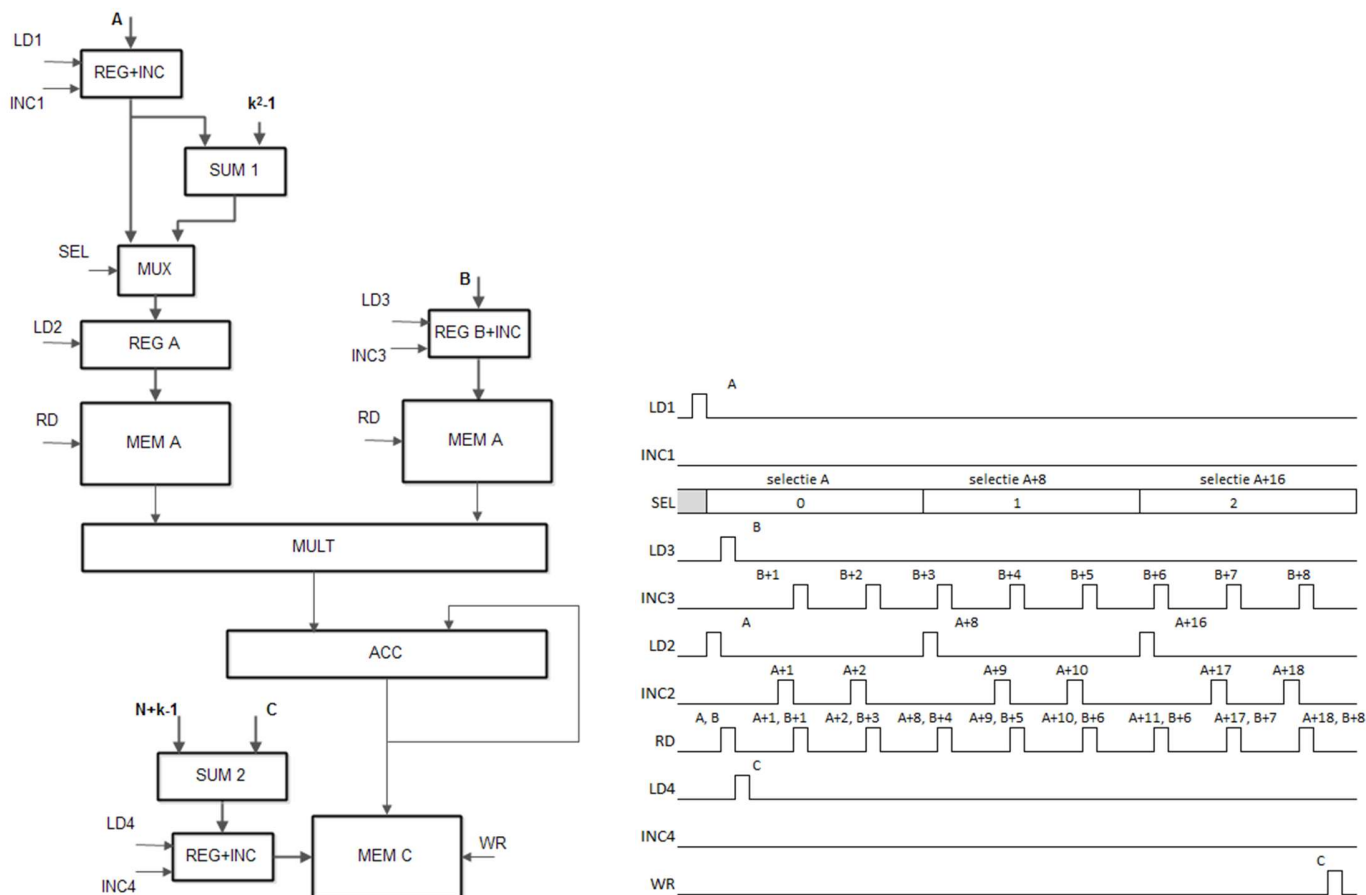


Figura 17. Exemplu de logica de comanda pentru calculul convoluției

Pentru accelerarea prelucrărilor se poate aplica modul de lucru *streaming* care suprapune operațiile de încărcare din memorie cu operațiile de calcul, ca în figura 18.

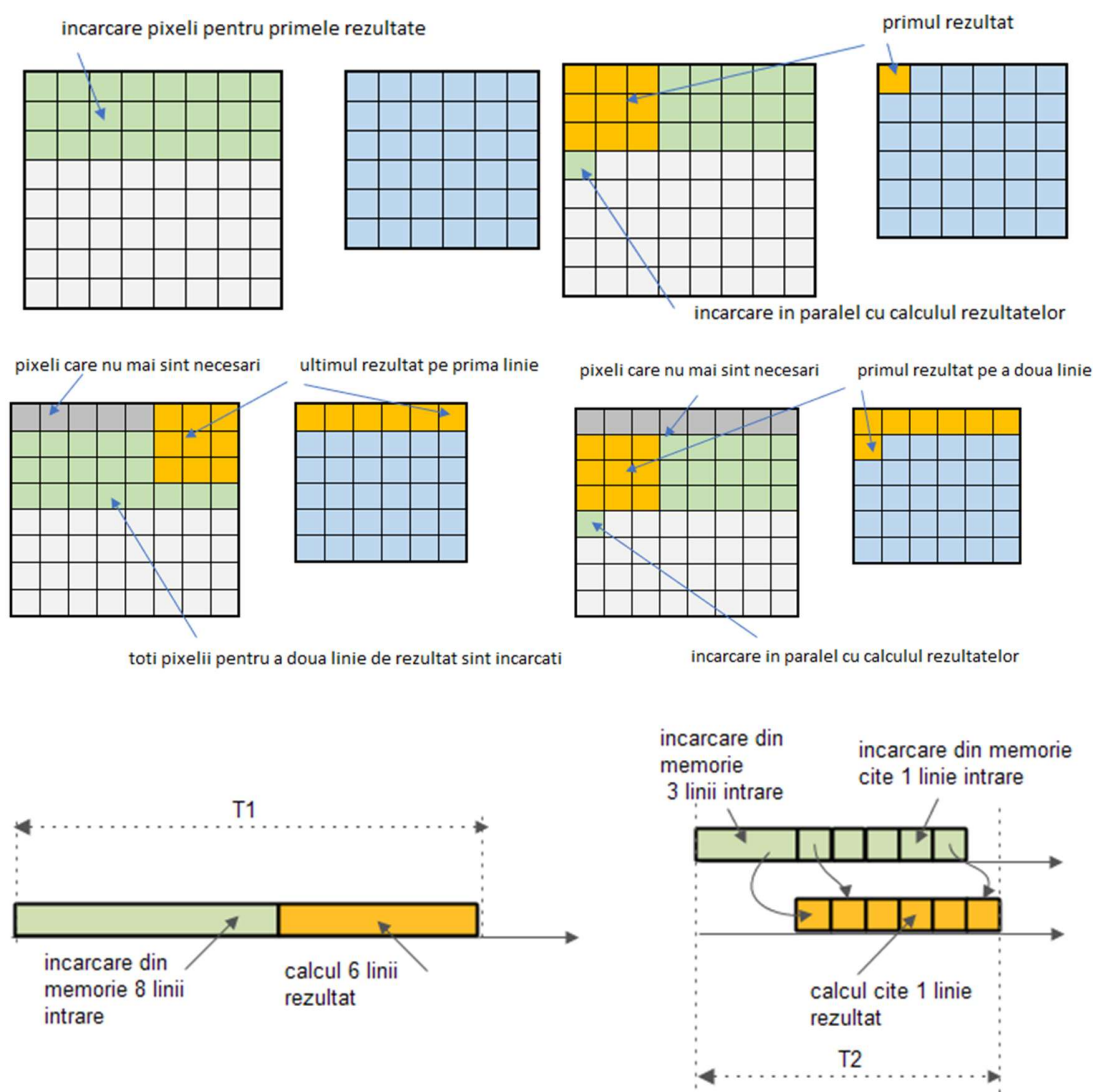


Figura 18. Modul de lucru cu suprapunere operații IO și operații de calcul (*streaming*)

4.1 Extrase din codul aplicației

Această secțiune prezintă succint modul de programare a aplicației și a CNN. În principiu, programarea acceleratorului CNN presupune scrierea unor cuvinte de comandă pentru:

- stabilirea arhitecturii CNN - număr de *layere* (straturi), modul de realizare a convoluțiilor (dimensiunea intrării, *pad*, *stride*, *pool*)
- stabilirea modului de acces la memorii (pentru citirea intrărilor și a ponderilor și scrierea rezultatelor)

Detalii pentru programarea CNN sunt ilustrate în secțiunea 4.2

Programul principal implementează organigramele din figurile 13, 14 și 15.

Programul principal

```
// Extras din codul pentru implementarea rețelei neuronale CNN cu doua clase (cats-dogs)
const char classes[CNN_NUM_OUTPUTS][10] = { "Cat", "Dog" };
// Layer de clasificare (full connected):
static int32_t ml_data[CNN_NUM_OUTPUTS];
// Calculul probabilitatilor cu softmax
static q15_t ml_softmax[CNN_NUM_OUTPUTS];
// timer pentru calculul timpului de prelucrare
volatile uint32_t cnn_time; // Stopwatch
// Buffer pentru display in format RGB565
uint8_t data565[128];
// imagini de test in format HWC
#include "sampledata_1.h"
#include "sampledata_2.h"
// buffer pentru imaginea de la camera in format RGB888
static uint32_t input_c[128*128];

int sample_mod=-1; // mod de lucru -1 initial, 0 si 1 test, 2 camera

int QPB1=0; // variabila de stare PB1
int QPB2=0; // variabila de stare PB2
int b1; // valoare PB1
int b2; // valoare PB2
int flag=0; // flag asteptare apasari butoane
int prag; // prag de decizie

/* ***** */
void cnn_load_input(void)
{
    int i;
    const uint32_t *in0; // = input_0;

    //
    if (sample_mod==0) in0=input_0;
    if (sample_mod==1) in0=input_1;
    if (sample_mod==2) in0=input_c;

    //
    for (i = 0; i < 16384; i++) {
        *((volatile uint32_t *)0x50000008) = *in0++; // scrie FIFO 0
    }
}

// afisarea pe display a imaginii de test
void display_sampledata(void)
{
    uint32_t w;
    uint8_t r, g, b;
    uint16_t rgb;

    int j = 0;
    uint32_t temp;

    int cnt = 0;

    w = IMAGE_SIZE_X; // IMAGE_SIZE_X = IMAGE_SIZE_Y = 128

    // Preia imaginea de test linie cu linie
```

```

    for (int row = 0; row < IMAGE_SIZE_Y; row++) {
        j = IMAGE_SIZE_X * 2 - 2; // oglindire pe display

        for (int k = 0; k < 4 * w; k += 4) {
// esantioanele de test sint in intervalul [-128,127]
// se convertesc in intervalul [0,255] pentru afisare

            if (sample_mod == 0) temp = input_0[cnt] ^ 0x00808080;
            if (sample_mod == 1) temp = input_1[cnt] ^ 0x00808080;

            // formatul HWC: 0x00bbgrr
            r = temp & 0xFF;
            g = (temp >> 8) & 0xFF;
            b = (temp >> 16) & 0xFF;
            cnt++;

            // convert la RGB656 pentru afisare
            rgb = ((r & 0b11111000) << 8) | ((g & 0b11111100) << 3) | (b >> 3);
            data565[j] = (rgb >> 8) & 0xFF;
            data565[j + 1] = rgb & 0xFF;

            j -= 2; // oglindire pe display
        }
        // afisare pe display
        MXC_TFT_ShowImageCameraRGB565(TFT_X_START, TFT_Y_START + row, data565, w, 1);
    }

// captura de imagine de la camera
void capture_process_camera(void)
{
    uint8_t *raw;
    uint32_t imgLen;
    uint32_t w, h;

    int cnt = 0;

    uint8_t r, g, b;
    uint16_t rgb;
    int j = 0;

    uint8_t *data = NULL;
    stream_stat_t *stat;

    camera_start_capture_image();

    // preia detaliile imaginii latime , inaltime, dimensiune
    camera_get_image(&raw, &imgLen, &w, &h);
    printf("W:%d H:%d L:%d \n", w, h, imgLen);

    // preia imaginea linie cu linie
    for (int row = 0; row < h; row++) {
// asteapta umplerea bufferului de la camera (comunicarea dintre camera si memorie - prin DMA)
        while ((data = get_camera_stream_buffer()) == NULL) {
            if (camera_is_image_rcv()) {
                break;
            }
        }

        j = IMAGE_SIZE_X * 2 - 2; // oglindire pe display
    }
}

```

```

    for (int k = 0; k < 4 * w; k += 4) {
        // formatul esantionului: 0x00bbggrr
        r = data[k];
        g = data[k + 1];
        b = data[k + 2];
        //se sare peste k+3

        // schimbare interval de la [0,255] la [-128,127] pentru CNN
        input_c[cnt++] = ((b << 16) | (g << 8) | r) ^ 0x00808080;
        // conversie RGB888 la RGB656 pentru afisare
        rgb = ((r & 0b11111000) << 8) | ((g & 0b11111100) << 3) | (b >> 3);
        data565[j] = (rgb >> 8) & 0xFF;
        data565[j + 1] = rgb & 0xFF;

        j -= 2; //oglindire pe display

    // afisare imagine
    MXC_TFT_ShowImageCameraRGB565(TFT_X_START, TFT_Y_START + row, data565, w, 1);

    // opreste camera
    release_camera_stream_buffer();
}

}

/* ***** */
int main(void)
{
    int i;
    int digs, tens;
    int ret = 0;
    int result[CNF_NUM_OUTPUTS]; // = {0};
    int dma_channel;

    char buff[TFT_BUFF_SIZE];

    printf("\n\nExemplu de retea neuronală CNN cu 2 clase (Cats-Dogs)\n");

    /* Valideaza cache */
    MXC_ICC_Enable(MXC_ICC0);

    /* Seteaza frecventa de tact la 100 MHz */
    MXC_SYS_Clock_Select(MXC_SYS_CLOCK_IPO);
    SystemCoreClockUpdate();

    /* Valideaza CNN, intreruperi CNN */
    /* CNN clock: 50 MHz */
    cnn_enable(MXC_S_GCR_PCLKDIV_CNNCLKSEL_PCLK, MXC_S_GCR_PCLKDIV_CNNCLKDIV_DIV1);

    /* porneste CNN boost (pinul P2.5 */
    cnn_boost_enable(MXC_GPIO2, MXC_GPIO_PIN_5);

    /* Initializare CNN - definire numar de layere */
    cnn_init();
    /* Incarcare ponderi pentru fiecare layer */
    cnn_load_weights();
    /* Incarcare bias - layer fully connected */

```

```

cnn_load_bias();
/* Configurare CNN */
cnn_configure();

printf("Initializare LCD\n");
// initializare TFT
MXC_TFT_Init();
MXC_TFT_ClearScreen();
MXC_TFT_ShowImage(0, 0, image_bitmap_1);

// intirziere 10 secunde
MXC_Delay(1000000);

// Initializare DMA pentru camera
MXC_DMA_Init();
dma_channel = MXC_DMA_AcquireChannel();

// Initializare camera
printf("Initializare Camera\n");
camera_init(CAMERA_FREQ);
ret=camera_setup
(IMAGE_SIZE_X,IMAGE_SIZE_Y,PIXFORMAT_RGB888,FIFO_THREE_BYTE,STREAMING_DMA,dma_channel);

if (ret != STATUS_OK) {
    printf("Error returned from setting up camera. Error %d\n", ret);
    return -1;
}

// setare camera clock prescaler pentru a preveni overflow-ul
camera_write_reg(0x11, 0x1);

// setare display (TFT)
MXC_TFT_SetPalette(image_bitmap_2);
MXC_TFT_SetBackgroundColor(4);

memset(buff, 32, TFT_BUFF_SIZE);
TFT_Print(buff, 20, 10, font_2, snprintf(buff, sizeof(buff), "Analog Devices"));
TFT_Print(buff, 20, 50, font_2, snprintf(buff, sizeof(buff), "Laborator ASC 2026"));
TFT_Print(buff, 20, 90, font_1, snprintf(buff, sizeof(buff), "Retea CNN - Clasificare"));
TFT_Print(buff, 20, 130, font_2, snprintf(buff, sizeof(buff), "Camera: PB2(SW2), PB1(SW1)"));
TFT_Print(buff, 20, 170, font_2, snprintf(buff, sizeof(buff), "Test: PB1(SW1)"));
TFT_Print(buff, 20, 210, font_2, snprintf(buff, sizeof(buff), "Start: PB1(SW1)"));

//   asteapta combinatia de butoane PB1 si PB2 pentru a seta modul de lucru

printf("Apasa PB1(SW1) pentru modul de test \r\n");
printf("Apasa PB2(SW2), apoi apasa PB1(SW1) pentru modul camera \r\n");

// apasare butoane
flag = 0;
while(flag==0)
{
    switch(QPB1)
    {
    case 0:
        b1 = PB_Get(0);
        if (b1 == 1) QPB1=1;

```

```

        break;
case 1:
    b1 = PB_Get(0);
    if (b1 == 0) QPB1=2;
    break;
case 2:
    flag=1;
    break;
}
switch(QPB2)
{
case 0:
    b2 = PB_Get(1);
    if (b2 == 1) QPB2=1;
    break;
case 1:
    b2 = PB_Get(1);
    if (b2 == 0) QPB2=2;
    break;
case 2:
    sample_mod=2;
    break;
}
}

if (sample_mod == -1 || sample_mod==0 || sample_mod==1)
{
    printf("Mod test\r\n");
    printf("Apasa PB1(SW1) pentru a schimba imaginea de test \r\n");
    LED_On(LED1);
    LED_Off(LED2);

}
if (sample_mod==2)
{
    printf("Mod camera \r\n");
    printf("Apasa PB1(SW1) pentru a captura o imagine noua \r\n");
    LED_On(LED2);
    LED_Off(LED1);
}

// Bucla principala de program
while (1) {

// asteapta PB1
flag = 0;
QPB1=0;
while(flag==0)
{
    switch(QPB1)
    {
    case 0:
        b1 = PB_Get(0);
        if (b1 == 1) QPB1=1;
        break;
    case 1:
        //
        if (sample_mod == 2) {
            #ifdef TFT_ENABLE
                MXC_TFT_ClearScreen();
            #endif
        }
    }
}
}

```

```

        #endif
        capture_process_camera();
    }
    //
        b1 = PB_Get(0);
        if (b1 == 0) QPB1=2;
        break;

    case 2:
        flag=1;
        break;
    }
}

if (sample_mod== -1 || sample_mod == 0 || sample_mod == 1)
{
    sample_mod = (sample_mod + 1) % 2;
    if (sample_mod == 0) printf("Intrarea - Imaginea de test 0 \r\n");
    if (sample_mod == 1) printf("Intrarea - Imaginea de test 1 \r\n");
}

if (sample_mod == 2) printf("Intrarea - de la camera\r\n");

// sterge display
MXC_TFT_ClearScreen();

//valideaza CNN clock
MXC_SYS_ClockEnable(MXC_SYS_PERIPH_CLOCK_CNN);

// preia imaginea
if (sample_mod == 0 || sample_mod == 1)
{
    display_sampledata();
}

if (sample_mod == 2)
{
    capture_process_camera();
}

// start CNN
cnn_start();
// incarca imaginea in CNN
cnn_load_input();

SCB->SCR &= ~SCB_SCR_SLEEPDEEP_Msk; // MAX78000 ARM in SLEEPDEEP
while (cnn_time == 0) {
    __WFI(); // asteapta in Sleep ca CNN sa termine (se va genera intrerupere la terminare)
}

// Preia datele de la CNN
cnn_unload((uint32_t *)ml_data);
// CNN stop
cnn_stop();

// Softmax
softmax_q17p14_q15((const q31_t *)ml_data, CNN_NUM_OUTPUTS, ml_softmax);

```

```

// afişare timp de prelucrare CNN
printf("Timp de prelucrare CNN: %d us\n\n", cnn_time);
// afişare rezultate clasificare la consola
printf("Rezultatul clasificarii:\n");

for (i = 0; i < CNN_NUM_OUTPUTS; i++) {
    digs = (1000 * ml_softmax[i] + 0x4000) >> 15;
    tens = digs % 10;
    digs = digs / 10;
    result[i] = digs;
    printf("[%7d] -> Clasa %d %8s: %d.%d%%\r\n", ml_data[i], i, classes[i], result[i], tens);
}

printf("\n");

// afisare rezultate clasificare pe display
area_t area;
area.x = 0;
area.y = 0;
area.w = 320;
area.h = TFT_Y_START - 1;
MXC_TFT_ClearArea(&area, 4);

memset(buff, 32, TFT_BUFF_SIZE);

if (sample_mod <2) prag=90;
if (sample_mod==2) prag=90;
printf("Prag de decizie: %d%%\n",prag);
if (result[0] > prag)
{
    TFT_Print(buff, TFT_X_START + 10, TFT_Y_START - 30, font_1,
               snprintf(buff, sizeof(buff), "%s (%d%%)", classes[0], result[0]));
    printf("Decizie: %s\n",classes[0]);
}
if (result[1] > prag)
{
    TFT_Print(buff, TFT_X_START + 10, TFT_Y_START - 30, font_1,
               snprintf(buff, sizeof(buff), "%s (%d%%)", classes[1], result[1]));
    printf("Decizie: %s\n",classes[1]);
}

if ( result[0] <=prag && result[1] <=prag)
{
    TFT_Print(buff, TFT_X_START + 10, TFT_Y_START - 30, font_1,
               snprintf(buff, sizeof(buff), "Unknown"));
    printf("Decizie: Unknown\n");
}

memset(buff, 32, TFT_BUFF_SIZE);
TFT_Print(buff, TFT_X_START + 40, TFT_Y_START + IMAGE_SIZE_Y + 10, font_1,
           snprintf(buff, sizeof(buff), "%dms", cnn_time / 1000));

TFT_Print(buff, 20, TFT_Y_START + IMAGE_SIZE_Y + 35, font_2,
           snprintf(buff, sizeof(buff), "Start: PB1(SW1)"));

return 0;
}

```

Programare CNN

```
// cats-dogs
// This file was @generated by ai8xsize.py --test-dir sdk/Examples/MAX78000/CNN --prefix cats-dogs
--checkpoint-file trained/ai85-catsdogs-qat8-q.pth.tar --config-file networks/cats-dogs-hwc.yaml -
-fifo --softmax --device MAX78000 --timer 0 --display-checkpoint --verbose --overwrite

// Configuring 7 layers
// Input data: HWC
// Layer 0: 3x128x128 streaming, no pooling, conv2d with kernel size 3x3, stride 1/1, pad 1/1, ReLU,
16x128x128 output
// Layer 1: 16x128x128 streaming, max pool 2x2 with stride 2/2, conv2d with kernel size 3x3, stride
1/1, pad 1/1, ReLU, 32x64x64 output
// Layer 2: 32x64x64, max pool 2x2 with stride 2/2, conv2d with kernel size 3x3, stride 1/1, pad
1/1, ReLU, 64x32x32 output
// Layer 3: 64x32x32, max pool 2x2 with stride 2/2, conv2d with kernel size 3x3, stride 1/1, pad
1/1, ReLU, 32x16x16 output
// Layer 4: 32x16x16, max pool 2x2 with stride 2/2, conv2d with kernel size 3x3, stride 1/1, pad
1/1, ReLU, 32x8x8 output
// Layer 5: 32x8x8, no pooling, conv2d with kernel size 3x3, stride 1/1, pad 1/1, ReLU, 16x8x8
output
// Layer 6: 16x8x8 flattened to 1024x1x1, no pooling, linear, no activation, 2x1x1 output

// masurarea timpului de calcul
// la CNN start
MXC_TMR_SW_Start(CNN_INFERENCE_TIMER);
// la CNN stop
cnn_time = MXC_TMR_SW_Stop(CNN_INFERENCE_TIMER);

// incarcarea ponderilor (din fisierul de ponderi impachetate)
int cnn_load_weights(void)
{
    uint32_t len;
    volatile uint32_t *addr;
    const uint32_t *ptr = kernels;

    while ((addr = (volatile uint32_t *) *ptr++) != 0) {
        len = *ptr++;
        while (len-- > 0)
            *addr++ = *ptr++;
    }

    return CNN_OK;
}

// preluarea rezultatelor
int cnn_unload(uint32_t *out_buf)
{
    volatile uint32_t *addr;

    // Custom unload for this network, layer 6: 32-bit data, shape: (2, 1, 1)
    addr = (volatile uint32_t *) 0x50401000;
    *out_buf++ = *addr++;
    *out_buf++ = *addr++;

    return CNN_OK;
}
```

Softmax

```
y_i = 2^(x_i/16384) / sum(2^(x_j/16384))  
y_i = e^(x_i) / sum(e^(x_j))
```

Definirea ponderilor

```
// bias pentru layer 6  
static const uint8_t bias_0[]={  
    0x0a, 0xf2  
};  
  
static const uint32_t kernels[]={  
    0x50180000, // adresa  
    0x00000166, // lungime  
    0xbcee09ee, 0x2f18ddd7, 0x7f30f5df, 0xff441eeb, 0xd2e2cb02, 0xd311f2fa, // ponderi  
    .....  
    0x50184000, // adresa  
    0x00000166, // lungime  
    0xddfded0d5, 0x0e40eaff, 0x3319f314, 0xf2140802, 0xe1ec2922, 0xcc0ef429, // ponderi  
    .....  
    0x50998000, // adresa  
    0x0000006c, // lungime  
    0x2011c9f0, 0xf71cd10e, 0xed0614f3, 0xe303321c, 0xf401fce4, 0x1e020128, // ponderi  
    .....  
    0x00000000 // terminator  
};
```

4.2.Programarea arhitecturii rețelei neurale (detalii)

Numărul maxim de niveluri = 32

Numărul de niveluri (layer) - 7

CNNx16_n Layer Count Maximum - CNNx16_n_LCNT_MAX [0x0008]
Layer Count Maximum = 6 (se numără de la 0)

Registre CNN

(pentru fiecare layer definit in arhitectura rețelei neurale si pentru fiecare cuadrant din arhitectura CNN)

- controlul adresării memoriei (FIFO si SRAM)
 - adresele pentru preluarea intrărilor si scrierea rezultatelor
- controlul procesului de convoluție
 - dimensiunea imaginii, dimensiunea kernel-ului, pad, stride, reprezentare ponderi (1, 2, 4 sau 8 biți)
- controlul procesului de pool - tip pooling, stride
- controlul procesului de flatten (activare)
- controlul bias (activare)
- controlul procesării (streaming sau nu, numărul de procesoare implicate)
- controlul postprocesării (scalare)

Configurare pe layere (exemplificare dimensiuni ale intrărilor de prelucrat)

Layer 0:

intrare 3x128x128, streaming, fără pooling, dimensiune kernel 3x3, stride 1/1, pad 1/1, ReLU, ieșire 16x128x128

CNNx16_n_Ly Row Count - CNNx16_n_Ly_RCNT [0x0010]-[0x008C]

Cuvântul de comandă: 0x00010081 = 0000 0000 0000 0001 0000 0000 1000 0001

Pad Rows = 01

Layer Row Count Maximum = 0x81=129 (128 linii plus pad-ul)

CNNx16_n_Ly_CCNT [0x0090]-[0x010C]

Cuvântul de comandă: 0x00010081 = 0000 0000 0000 0001 0000 0000 1000 0001

Layer Column Pad Count = 01

Layer Column Count Maximum = 0x81=129 (128 coloane plus pad-ul)

*((volatile uint32_t *) 0x50100010) = 0x00010081; // Rows

*((volatile uint32_t *) 0x50100090) = 0x00010081; // Columns

Layer 1:

intrare, 16x128x128, streaming, pooling max pool 2x2, cu stride 2/2, dimensiune kernel 3x3, stride 1/1, pad 1/1, ReLU, iesire 32x64x64

CNNx16_n_Ly Row Count - CNNx16_n_Ly_RCNT [0x0010]-[0x008C]

Cuvântul de comandă: 0x00010081 = 0000 0000 0000 0001 0000 0000 1000 0001

Pad Rows = 01

Layer Row Count Maximum = 0x81=129 (128 linii plus pad-ul)

CNNx16_n_Ly Column Count - CNNx16_n_Ly_CCNT [0x0090]-[0x010C]

Cuvântul de comandă: 0x00010081 = 0000 0000 0000 0001 0000 0000 1000 0001

Layer Column Pad Count = 01

Layer Column Count Maximum = 0x81=129 (128 coloane plus pad-ul)

*((volatile uint32_t *) 0x50100014) = 0x00010081; // Rows

*((volatile uint32_t *) 0x50100094) = 0x00010081; // Columns

Layer 2:

intrare 32x64x64, pooling max pool 2x2 cu stride 2/2, dimensiune kernel 3x3, stride 1/1, pad 1/1, ReLU, iesire 64x32x32

CNNx16_n_Ly Row Count - CNNx16_n_Ly_RCNT [0x0010]-[0x008C]

Cuvântul de comandă: 0x00010041 = 0000 0000 0000 0001 0000 0000 0100 0001

Pad Rows = 01

Layer Row Count Maximum = 0x41=65 (64 linii plus pad-ul)

CNNx16_n_Ly Column Count - CNNx16_n_Ly_CCNT [0x0090]-[0x010C]

Cuvântul de comandă: 0x00010041 = 0000 0000 0000 0001 0000 0000 0100 0001

Layer Column Pad Count = 01

Layer Column Count Maximum = 0x41=65 (64 coloane plus pad-ul)

*((volatile uint32_t *) 0x50100018) = 0x00010041; // Rows

*((volatile uint32_t *) 0x50100098) = 0x00010041; // Columns

Layer 3:

intrare 64x32x32, pooling max pool 2x2 cu stride 2/2, dimensiune kernel 3x3, stride 1/1, pad 1/1, ReLU, iesire 32x16x16

CNNx16_n_Ly Row Count - CNNx16_n_Ly_RCNT [0x0010]-[0x008C]

Cuvântul de comandă: 0x00010021 = 0000 0000 0000 0001 0000 0000 0010 0001

Pad Rows = 01

Layer Row Count Maximum = 0x21=33 (32 linii plus pad-ul)

CNNx16_n_Ly Column Count - CNNx16_n_Ly_CCNT [0x0090]-[0x010C]

Cuvintul de comanda: 0x00010021 = 0000 0000 0000 0001 0000 0000 0010 0001
Layer Column Pad Count = 01
Layer Column Count Maximum = 0x21=33 (32 coloane plus pad-ul)

```
((volatile uint32_t *) 0x5010001c) = 0x00010021; // Rows  
((volatile uint32_t *) 0x5010009c) = 0x00010021; // Columns
```

Layer 4:

intrare 32x16x16, pooling max pool 2x2 cu stride 2/2, dimensiune kernel 3x3, stride 1/1, pad 1/1, ReLU, iesire 32x8x8

CNNx16_n_Ly Row Count - CNNx16_n_Ly_RCNT [0x0010]-[0x008C]

Cuvintul de comanda: 0x00010011 = 0000 0000 0000 0001 0000 0000 0001 0001

Pad Rows = 01

Layer Row Count Maximum = 0x11=17 (16 linii plus pad-ul)

CNNx16_n_Ly Column Count - CNNx16_n_Ly_CCNT [0x0090]-[0x010C]

Cuvintul de comanda: 0x00010011 = 0000 0000 0000 0001 0000 0000 0001 0001

Layer Column Pad Count = 01

Layer Column Count Maximum= 0x11=17 (16 coloane plus pad-ul)

```
((volatile uint32_t *) 0x50100020) = 0x00010011; // Rows  
((volatile uint32_t *) 0x501000a0) = 0x00010011; // Columns
```

Layer 5:

intrare 32x8x8, fără pooling, dimensiune kernel 3x3, stride 1/1, pad 1/1, ReLU, iesire 16x8x8

CNNx16_n_Ly Row Count - CNNx16_n_Ly_RCNT [0x0010]-[0x008C]

Cuvintul de comanda: 0x00010009 = 0000 0000 0000 0001 0000 0000 0000 1001

Pad Rows = 01

Layer Row Count Maximum = 0x09=9 (8 linii plus pad-ul)

CNNx16_n_Ly Column Count - CNNx16_n_Ly_CCNT [0x0090]-[0x010C]

Cuvintul de comanda: 0x00010009 = 0000 0000 0000 0001 0000 0000 0000 1001

Layer Column Pad Count = 01

Layer Column Count Maximum = 0x09=9 (8 coloane plus pad-ul)

```
((volatile uint32_t *) 0x50100024) = 0x00010009; // Rows  
((volatile uint32_t *) 0x501000a4) = 0x00010009; // Columns
```

Layer 6:

intrare 16x8x8, activare flatten la 1024x1x1, fără pooling, iesire 2x1x1

CNNx16_n_Ly One Dimensional Control - CNNx16_n_Ly_ONED [0x0110]-[0x018C]

Cuvintul de comanda: 0x00000100

One Dimensional Convolution Start Mask Address = 1

```
((volatile uint32_t *) 0x50100128) = 0x00000100; // 1D
```

Observați modul de împachetare a ponderilor (pagina 22, variabila *kernels*) si modul de implementare a funcției *softmax* (pagina 22 – Softmax si formula de la pagina 8) .

Configurarea fiecărui nivel (strat, *layer*) se precizează într-un fișier de configurarea (*.yaml) ca in exemplul următor(fiecare procesor activ are 1 logic in cuvintul *processors*):

layers:

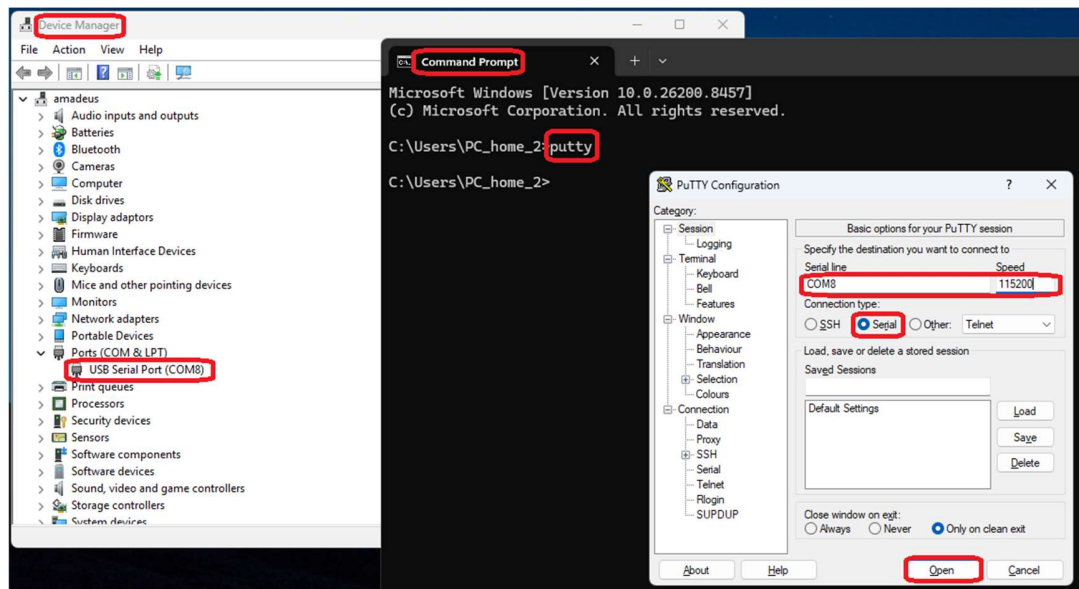
- pad: 1
activate: ReLU
out_offset: 0x1000
processors: 0x0000000000000007
data_format: HWC
operation: Conv2d
- max_pool: 2
pool_stride: 2
pad: 1
activate: ReLU
out_offset: 0x2000
processors: 0x000ffff000000000
operation: Conv2d
- max_pool: 2
pool_stride: 2
pad: 1
activate: ReLU
out_offset: 0x0000
processors: 0x00000000ffffffff
operation: Conv2d
- max_pool: 2
pool_stride: 2
pad: 1
activate: ReLU
out_offset: 0x2000
processors: 0xffffffffffffffff
operation: Conv2d
- max_pool: 2
pool_stride: 2
pad: 1
activate: ReLU
out_offset: 0x0000
processors: 0x00000000ffffffff
operation: Conv2d
- pad: 1
activate: ReLU
out_offset: 0x2000
processors: 0xffffffff00000000
operation: Conv2d
- op: mlp
flatten: true
out_offset: 0x1000
output_width: 32
processors: 0x0000000000000fff
activate: None

Layer	Intrare	Iesire	Operatii	Nr procesoare	Nr de iteratii per procesor
0	3x128x128	16x128x128	Conv2D	3	16
1	16x128x128	32x64x64	Maxpool, Conv2D	16	2
2	32x64x64	64x32x32	Maxpool, Conv2D	32	1
3	64x32x32	32x16x16	Maxpool, Conv2D	64	1
4	32x16x16	32x8x8	Maxpool, Conv2D	32	1
5	32x8x8	16x8x8	Conv2D	32	1
6	16x8x8	2x1x1	Conv1D (flatten. MLP)	16	2

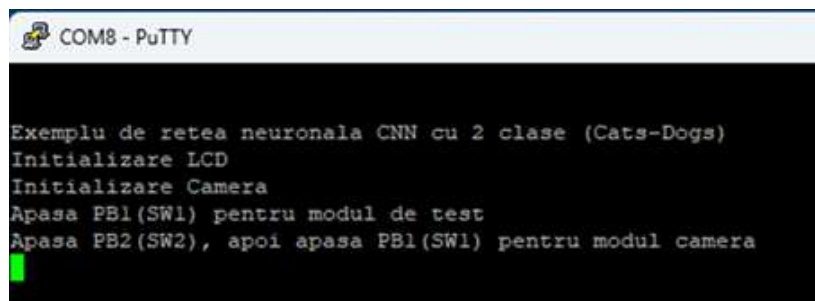
5. Rezultate obținute si modul de operare

5.1 Stabilirea comunicatiei seriale intre MAX78000 Evkit si calculator (MAX78000 este conectat prin USB la calculator; prin USB se asigura alimentarea si un port serial virtual)

- se determina portul serial din device manager (de exemplu COM8)
- se ruleaza, in fereastra de comanda cmd, programul putty pentru comunicatie seriala, pe portul definit anterior, cu viteza de comunicatie 115200



5.2 Mesajele trimise pe interfata seriala la pornirea sistemului (stabilire mod de lucru)



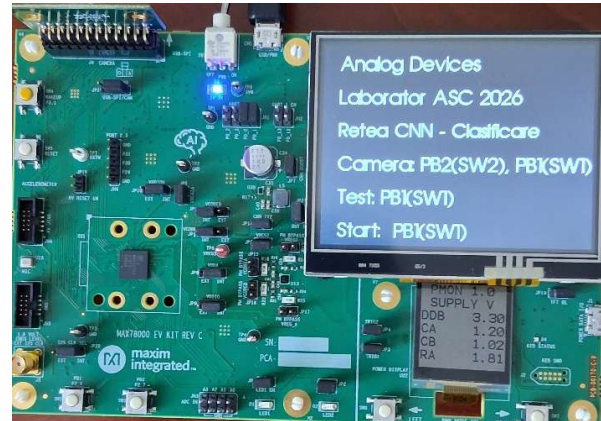
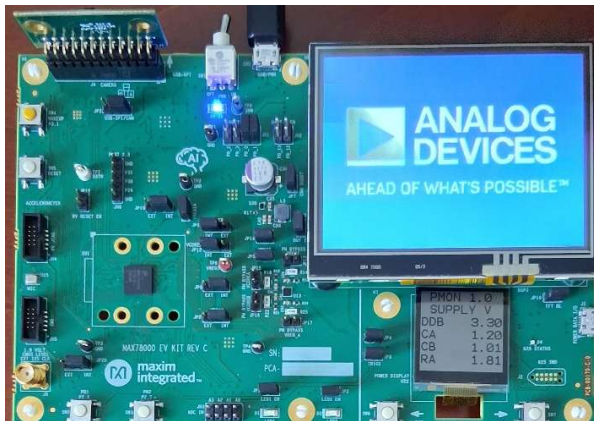
5.3 Semnificatia butoanelor pe MAX78000 Evkit

PB1(SW1), PB2(SW2) – stabilire mod de lucru si start clasificare

LED1 (verde) – aprins in mod test, clipeste la start clasificare pe mod camera

LED2 (rosu) – aprins mod camera, clipeste la start clasificare pe mod test

Exista un display (TFT LCD - *thin-film-transistor liquid-crystal display*) pe care se afiseaza imaginile si mesajele necesare. Exista un al doilea display LCD pe care se afiseaza puterea consumata pe magistralele de alimentare (DDB – pentru memorie, CA si CB pentru CNN si RA pentru MCU). Butoanele SW3 si SW4 (de langa acest display) comuta intre afisarea tensiunilor, curentului consumat si puterii consumate).



5.4 Mesajele transmise pe interfața serială după o clasificare (imagine de la camera). Se afișează scorurile, procentul de confidență, timpul de prelucrare, pragul de decizie minim și decizia.

Scoruri pentru fiecare clasa

Procent de confidență pentru fiecare clasa

```

Apasa PB1(SW1) pentru a captura o imagine noua
Intrarea de la camera
W:128 H:128 L:65536
Timp de prelucrare CNN: 12052 us

Rezultatul clasificării:
[ 8572] -> Clasa 0      Cat: 66.7%
[ -3452] -> Clasa 1    Dog: 33.3%

Prag de decizie: 90%
Decizie: Unknown
  
```



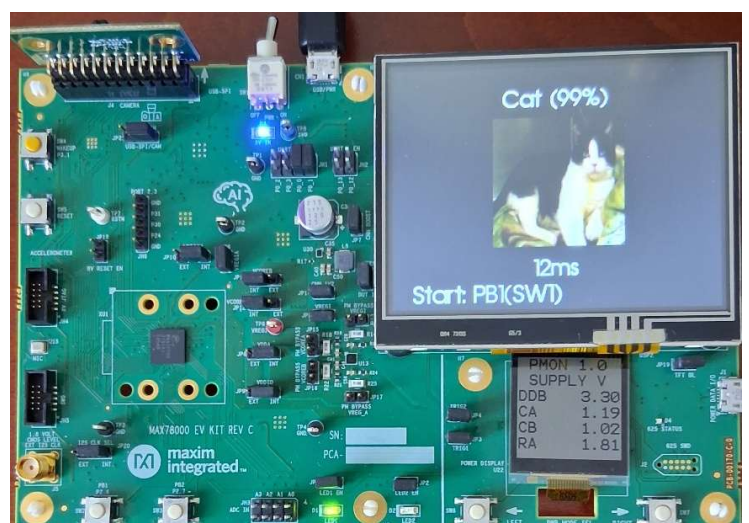
5.5 Mesajele transmise pe interfața serială după o clasificare pentru imaginile de test.

```

Mod test
Apasa PB1(SW1) pentru a schimba imaginea de test
Intrarea - Imaginea de test 0
Timp de prelucrare CNN: 12157 us

Rezultatul clasificării:
[ 70045] -> Clasa 0      Cat: 99.8%
[ -69649] -> Clasa 1    Dog: 0.2%

Prag de decizie: 90%
Decizie: Cat
  
```



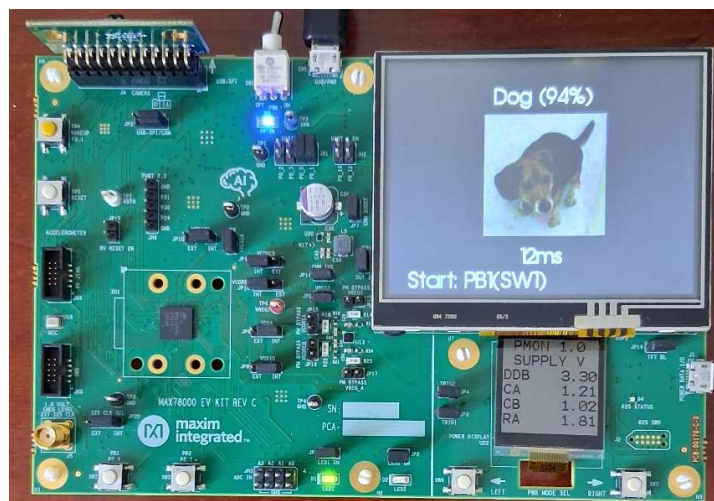
```

Mod test
Apasa PB1(SW1) pentru a schimba imaginea de test
Intrarea - Imaginea de test 0
Timp de prelucrare CNN: 12157 us

Rezultatul clasificarii:
[ -35088] -> Clasa 0      Cat: 5.9%
[  29101] -> Clasa 1      Dog: 94.1%

Prag de decizie: 90%
Decizie: Dog

```



5.6 Mesajele transmise pe interfața seriala după clasificări pentru imagini de la camera.

Imaginile testate:



Rezultatele clasificarii:

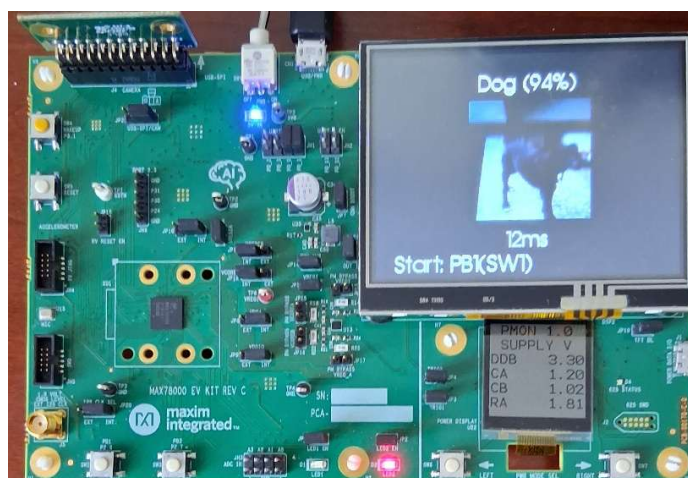
```

Intrarea - de la camera
W:128 H:128 L:65536
Timp de prelucrare CNN: 12052 us

Rezultatul clasificarii:
[ -35088] -> Clasa 0      Cat: 5.9%
[  29101] -> Clasa 1      Dog: 94.1%

Prag de decizie: 90%
Decizie: Dog

```



```

Intrarea - de la camera
W:128 H:128 L:65536
Timp de prelucrare CNN: 12052 us

Rezultatul clasificarii:
[ 39197] -> Clasa 0      Cat: 94.1%
[ -33520] -> Clasa 1    Dog: 5.9%

Prag de decizie: 90%
Decizie: Cat

```



6. Desfășurarea lucrării

- 6.1 Studiați arhitectura rețelei neuronale CNN pentru clasificarea cu doua clase
- 6.2 Pentru exemplul de logica de comanda din figura 17, estimați timpul de calcul pentru toate convoluțiile din arhitectura rețelei neuronale din figura 5.
- 6.3. Studiați modul de programare al CNN, pentru aceasta arhitectura. Se vor evidenția, in principiu, etapele necesare in programare (fără detalii despre conținutul registrelor de comanda – doar funcționalitatea acestora)
- 6.4. Studiați modul de implementare a programului de stabilire a modului de lucru (test, camera)
- 6.5. Studiați modul de preluare si împachetare a ponderilor si a imaginilor pentru realizarea clasificarii
- 6.6. Experimentați pe placa de evaluare MAX78000 Evkit, clasificarea unor imagini preluate cu camera (de pe ecranul PC si directorul cu imagini de test) si determinați precizia clasificarii. Se va completa următorul tabel (decizia 1 se va lua prin majoritate, la egalitate clasa va fi “unknown”, iar decizia 2 se va lua cu un prag de confidenta peste 90%), pentru minim 10 imagini din fiecare clasa:

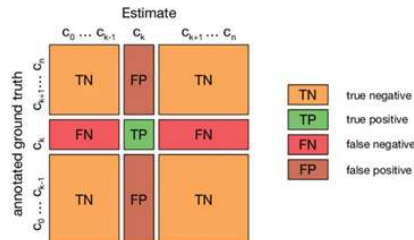
Nr. crt	Clasa reala	Confidenta clasa 1	Confidenta clasa 2	Decizie 1	Decizie 2	Timp de execuție	Energie consumata
1							
2							
...							
20							

Se vor calcula următoarele metrice de performanta [2] pentru ambele tipuri de decizie, ca in exemplul de mai jos:

Clase adevărate	Clasa 3 Unknown	4	6	90
	Clasa 2 Dogs	3	90	7
	Clasa 1 Cats	90	6	4
		Clasa 1 Cats	Clasa 2 Dogs	Clasa 3 Unknown
Clase prezise				

Clase adevărate	Clasa 1 Cats	4	6	90
	Clasa 2 Dogs	3	90	7
	Clasa 3 Unknown	90	6	4
		Clasa 1 Cats	Clasa 2 Dogs	Clasa 3 Unknown
Clase prezise				

Clase adevărate	Clasa 3 Unknown	4	6	90
	Clasa 2 Dogs	3	90	7
	Clasa 1 Cats	90	6	4
		Clasa 1 Cats	Clasa 2 Dogs	Clasa 3 Unknown
Clase prezise				



	TP	TN	FP	FN
Clasa 1 Cats	90	193	7	10
Clasa 2 Dogs	90	188	12	10
Clasa 3 Unknown	90	189	11	10
Total	270	570	30	30

Global

Acuratetea = $(TP+TN)/(TP+TN+FP+FN)=$	0.93
Probabilitatea de detectie = $TP/(TP+FN)=$	0.90
Probabilitatea de alarma falsa = $FP/(FP+TN)=$	0.05
Precizia = $TP/(TP+FP)=$	0.90

Clasa 1

Cats

Acuratetea = $(TP+TN)/(TP+TN+FP+FN)=$	0.94
Probabilitatea de detectie = $TP/(TP+FN)=$	0.90
Probabilitatea de alarma falsa = $FP/(FP+TN)=$	0.04
Precizia = $TP/(TP+FP)=$	0.93

Clasa 2

Dogs

Acuratetea = $(TP+TN)/(TP+TN+FP+FN)=$	0.93
Probabilitatea de detectie = $TP/(TP+FN)=$	0.90
Probabilitatea de alarma falsa = $FP/(FP+TN)=$	0.06
Precizia = $TP/(TP+FP)=$	0.88

Clasa 3

Unknown

Acuratetea = $(TP+TN)/(TP+TN+FP+FN)=$	0.93
Probabilitatea de detectie = $TP/(TP+FN)=$	0.90
Probabilitatea de alarma falsa = $FP/(FP+TN)=$	0.06
Precizia = $TP/(TP+FP)=$	0.89

6.7. Vizualizați, pe display-ul monitorului de putere (din dreapta jos pe placa de evaluare), tensiunile, curentul, puterea si energia pentru implementarea rețelei neuronale CNN.

7. Întrebări

1. De ce clasificarea poate fi eronata ? Discuție.
2. Cum s-ar putea corecta clasificarea ? Discuție.
3. Cum influențează intrarea corectitudinea deciziei ?
4. Cum variază acuratețea in funcție de numărul de teste efectuate?
5. Implementarea nivelului Fully Connected se poate realiza printr-o convoluție 1D?
6. Estimați creșterea de viteza in cazul utilizării modului *streaming*.
7. Funcția *softmax* este strict necesara pentru clasificare? Argumentați.

8. Referințe

- [1]. <https://courses.grainger.illinois.edu/cs433/fa2022/projects/Google-TPU.pdf>
- [2]. Krüger, Frank. (2016). Activity, Context, and Plan Recognition with Computational Causal Behaviour Models.
- [3]. <https://github.com/analogdevicesinc/ai8x-synthesis>
- [4]. <https://github.com/analogdevicesinc/ai8x-training>
- [5]. <https://www.analog.com/en/resources/media-center/videos/6313217809112.html>
- [6]. Martina Maria Balbi Antunes, Lance Doherty, Thomas Watteyne. Embedded Artificial Intelligence for IoT Applications Using the MAX78000. IEEE Access, 2025
- [7]. <https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.analog.com/media/en/technical-documentation/user-guides/max78000-user-guide.pdf>