Proyecto 3: Optimizing Traditional and Deep Learning Algorithms for Autonomous Driving

Nombre: Ing. Luis Carlos Alvarez Mata
Carne: 200902144
Nombre: Ing. Verny Moreles Soto
Carne: 201116212

1. Optimización del algoritmo SGBM

1.1. Perfilado del algoritmo SGBM para detectar *hotspots*

El primer paso hacia el objetivo de la optimización es determinar qué se debe optimizar exactamente para obtener los mejores resultados mientras se invierte la menor cantidad de esfuerzo posible. Para ello, se realiza la perfilación de la aplicación, para encontrar sus puntos de acceso y abordarlos con métodos de optimización. En este caso, se utilizará el generador de perfiles Valgrind, que está integrado en QTCreator.

Para realizar el perfilado se utiliza como caso de prueba las imágenes de entrada *Prof_L.png* y *Prof_R.png* y no se selecciona ninguna imagen *Ground Truth* debido a que solo se quiere evaluar el cálculo del SGBM.

En la figura 9 se muestra el resultado del perfilado donde se observa que las funciones *find_min*, *find_minLri* y *compute_hamming_distance* son las que tienen el mayor número de llamadas. Con este resultado nos da una referencia de las funciones que se pueden optimizar. Al analizar las funciones se observan que estas solo poseen un loop los cuales tienen un numero pequeño de iteraciones. Analizando más fondo el código se descubre que estas tres funciones están dentro de otras dos funciones, *compute_hamming* y *cost_computation* las cuales tienen tiene 3 y 4 loops anidados respectivamente, como se muestra en la figura 10.

Function	Location	Called	Self Cost: Ir	Incl. Cost: Ir
find_min(int, int, int, int)	/home/luis/P...	11524350	92 194 800	92 194 800
StereoFPGA::find_minLri(int*)	/home/luis/P...	11524350	10 464 109 800	10 464 109 800
StereoFPGA::compute_hamming_distance(unsigned long, unsigned long)	/home/luis/P...	910800	530 085 600	530 085 600
strcmp	/build/glibc-2...	343502	11 095 363	11 095 363
0x0000000000020de0	/usr/lib/x86_...	270915	13 906 167	17 493 621
__strchr_sse2	/build/glibc-2...	248072	8 158 914	8 158 914
__strlen_sse2	/build/glibc-2...	191861	3 795 872	3 795 872
__strcmp_sse2_unaligned	/build/glibc-2...	188188	6 479 527	6 479 527
0x00000000000216a0	/usr/lib/x86_...	187007	4 307 524	4 471 766
free	/build/glibc-2...	164854	12 727 796	12 769 245
malloc	/build/glibc-2...	164110	9 044 975	24 194 679
0x0000000000026a30	/usr/lib/x86_...	135832	2 716 640	2 716 640
_dl_name_match_p	/build/glibc-2...	134799	3 644 555	11 801 453
0x000000000001a050	/usr/lib/x86_...	118104	3 682 504	9 563 196
0x000000000018a440	/usr/lib/x86_...	107014	642 084	3 831 094
memcpy@GLIBC_2.2.5	/build/glibc-2...	104735	2 324 030	2 324 030
_int_malloc	/build/glibc-2...	102289	16 917 928	19 692 242
g_ascii_strncasecmp	/usr/lib/x86_...	101426	3 829 692	3 829 692
g_free	/usr/lib/x86_...	98518	393 072	8 667 778
g_malloc	/usr/lib/x86_...	83607	836 078	12 412 012
0x000000000001846f0	/usr/lib/x86_...	83086	2 653 529	8 947 379

Figura 1: Perfilado del algoritmo SGBM.

Por lo tanto, se seleccionaron las funciones *compute_hamming*, *cost_computation* y *cost_aggregation* las cuales tendrían la mayor cantidad de loops anidados, 4, 5 y 4 respectivamente para ser optimizadas.

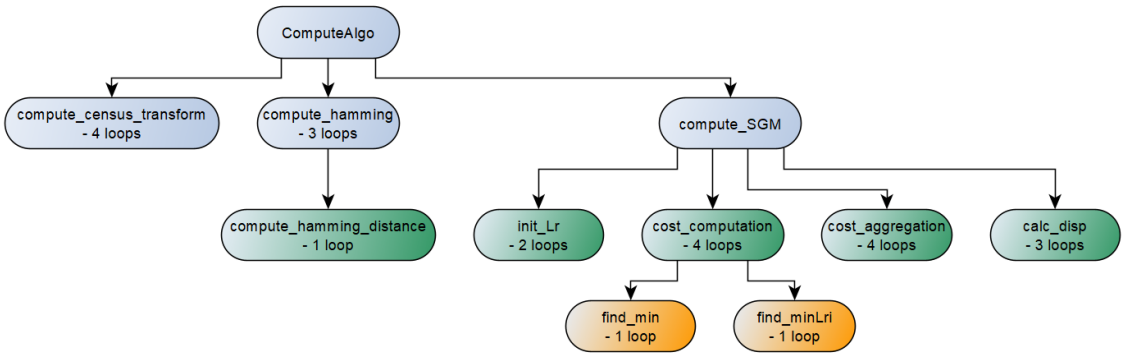

Figura 2: Diagrama de distribución de funciones y loops.

1.2. Reducción de la huella de memoria de la implementación de línea base

Al analizar el código se visualiza que la mayoría de las variables toman en cuenta la máxima profundidad de los datos, por lo tanto, nos enfocaremos en la optimización de la profundidad de las 4 variables a definir su dato máximo:

- BlockSize: 5 pixels - uint8_t m_u8BlockSize
- P2: 128 - uint16_t m_u16P2
- NumDir: 4 - uint8_t m_u8Directions
- Cmax(p, d) - uint64_t *ct1

Visualizando la profundidad máxima de las variables, se decide cambiar el tipo de variable para P2 de *uint16_t* a *uint8_t* dando que su dato máximo seria 128. para las demás variables se conserva la profundidad original ya que no se visualiza una reducción posible. Por lo tanto, quedaría la siguiente forma:

- BlockSize: 5 pixels - uint8_t m_u8BlockSize
- P2: 128 - uint8_t m_u16P2
- NumDir: 4 - uint8_t m_u8Directions
- Cmax(p, d) - uint64_t *ct1

1.3. Técnica multi-threading para acelerar Loops costosos

Para realizar la optimización por medio de multi-threading se implementó el API OpenMP, debido a que brindo una buena optimización en tiempo de ejecución y facilidad de implementación en la tarea 2. Además, debido a que se tienen funciones con loops anidados, se decidió utilizar clausula *#pragma omp parallel for collapse* la cual da la capacidad de especificar cuántos loops están asociados con la construcción y de especificar las variables compartidas. Se dejo que el API seleccionara de forma automática el número de threads ya que en la tarea 2 se observaron buenos resultados con esta técnica.

La cláusula *collapse* nos la desventaja de que, si la ejecución de cualquier ciclo asociado cambia alguno de los valores utilizados para calcular cualquiera de los recuentos de iteraciones, entonces el comportamiento no está especificado. Por lo tanto, no en todos los casos se puso a utilizar con la profundidad total de los loops anidados.

En el fragmento de código a continuación el cual describe la función *compute_hamming* se puede visualizar que se utilizó un numero de 3 para la cláusula *collapse*, número que fue factible debido a que se hace manejo de variables solo en el loop interno.

```
void StereoFPGA::compute_hamming(uint64_t *ct1, uint64_t *ct2, int *accumulatedCost) {
    #pragma omp parallel for collapse(3) shared(ct1,ct2, accumulatedCost)
    for (int i=m_u16yMin; i<m_u16yMax; i++) {
        for (int j=m_u16xMin; j<m_u16xMax; j++) {
            for (int d=0; d<m_u16TotalDisp; d++) {
                int dist = StereoFPGA::compute_hamming_distance(ct1[i*m_u16width_after_census+j],
                    ct2[i*m_u16width_after_census+j-(d+m_s16MinDisp)]);
                int Loop = (i*m_u16width_after_census+j)*m_u16TotalDisp + d;
                accumulatedCost[Loop] = dist;
            }
        }
    }
}
```

Para la función *cost_computation* no se realizó la paralelización a partir del primer loop, debido a que este tiene diversas comparaciones y asignaciones, lo cual no permitía utilizar la cláusula *#pragma omp parallel for collapse* y la paralización no era tan efectiva, tomando también en cuenta que es un loop con pocas iteraciones, máximo 4. Buscando esta mejora en el tiempo de cómputo se agregó la cláusula a partir del segundo loop y realizando un collapse de 2.

```
void StereoFPGA::cost_computation(int *Lr, int *initCost) {
    int iDisp, jDisp;
    for (int r=0; r<m_u8Directions; r++) {
        if (r==0) {
            iDisp = 0; jDisp = 0-m_u8DEBUGLeftNeighbor;
        }
        else if (r==1) {
            iDisp = -1; jDisp = -1;
        }
        else if (r==2) {
            iDisp = -1; jDisp = 0;
        }
        else if (r==3) {
            iDisp = -1; jDisp = 1;
        }
        else if (r==4) {
            iDisp = 0; jDisp = 1;
        }
        #pragma omp parallel for collapse(2) shared(iDisp, jDisp, Lr, initCost, r)
        for (int i=0; i<m_u16height_after_census; i++) {
            for (int j=0; j<m_u16width_after_census; j++) {
                int iNorm = i + iDisp; //height
                int jNorm = j + jDisp; //width
                int *Lrpr =
                    Lr+((r*m_u16height_after_census+iNorm)*m_u16width_after_census+jNorm)*m_u16TotalDisp;
                for (int d=0; d<m_u16TotalDisp; d++) {
                    int Cpd = initCost[(i*m_u16width_after_census+j)*m_u16TotalDisp+d];
                    int tmp;
                    if ( (((r==0)||(r==1))&&(jNorm<0)) || (((r==1)||(r==2)||(r==3))&&(i==0)) ||
                         ((r==3)&&(j==m_u16width_after_census-1))) {
                        tmp = Cpd;
                    } else {
                        int minLri = find_minLri(Lrpr);
                        int Lrpdm1, Lrpdp1;
                        if (d==0)
                            Lrpdm1 = INT_MAX-m_u16P1;
```

```
                    else
                        Lrpdm1 = Lrpr[d-1];
                    if (d==m_u16TotalDisp-1)
                        Lrpdp1 = INT_MAX-m_u16P1;
                    else
                        Lrpdp1 = Lrpr[d+1];

                    int v1 = find_min(Lrpr[d], Lrpdm1+m_u16P1, Lrpdp1+m_u16P1, minLri+m_u16P2);
                    tmp = Cpd + v1 - minLri;
                }
                Lr[((r*m_u16height_after_census+i)*m_u16width_after_census+j)*m_u16TotalDisp+d] = tmp;
            }
        }
    }
}
```

El siguiente fragmento de código representa la función *cost_aggregation*, la cual posee 4 loops anidados, pero se observa que a partir de tercer loop se realizan asignación de variables, lo cual nos permite realizar un collapse de 3.

```
void StereoFPGA::cost_aggregation(int *aggregatedCost, int *Lr) {
    #pragma omp parallel for collapse(3) shared(aggregatedCost, Lr)
    for (int i=0; i<m_u16height_after_census; i++) {
        for (int j=0; j<m_u16width_after_census; j++) {
            for (int d=0; d<m_u16TotalDisp; d++) {
                int *ptr = aggregatedCost + (i*m_u16width_after_census+j)*m_u16TotalDisp+d;
                ptr[0] = 0;
                for (int r=0; r<m_u8Directions; r++) {
                    ptr[0] +=
                        Lr[((r*m_u16height_after_census+i)*m_u16width_after_census+j)*m_u16TotalDisp+d];
                }
            }
        }
    }
}
```

1.4. Optimización CUDA

Para realizar la optimización utilizando *Compute Unified Device Architecture (CUDA)* se optimizaron las mismas funciones. Donde se tuvo que modificar la implementación creando un archivo *.cu* Por medio de este archivo se implementa las funciones las cuales van a ser paralelizadas en el GPU. Donde por medio de la instrucción *__global__* o *__device__* la cual nos indica si las funciones pueden ser llamadas desde el host o el device.

Las funciones *compute_hamming*, *cost_computation* y *cost_aggregation* son declaradas globales, debido a que son las funciones llamadas por el host o por el código C++. Y las funciones emphfind_min, *find_minLri* y *compute_hamming_distance* son declaradas devices ya que serán llamadas por el GPU. Para poder realizar el link entre el flujo del código y las funciones que fueron movidas al archivo *.cu* se realiza en la siguiente manera:

```
extern "C++"
{
void cost_computation(int *Lr, int *initCost, uint8_t m_u8Directions, uint8_t m_u8DEBUGLeftNeighbor,
    uint16_t m_u16height_after_census, uint16_t m_u16width_after_census, uint16_t m_u16TotalDisp, uint8_t
    m_u16P1, uint8_t m_u16P2);
void compute_hamming(uint64_t *ct1, uint64_t *ct2, int *accumulatedCost, uint64_t m_u16yMin, uint64_t
    m_u16yMax, uint64_t m_u16xMin, uint64_t m_u16xMax, uint64_t m_u16TotalDisp, uint64_t
    m_u16width_after_census, uint64_t m_s16MinDisp);
```

```
void cost_aggregation(int *aggregatedCost, int *Lr, uint16_t m_u16height_after_census, uint16_t
    m_u16width_after_census, uint16_t m_u16TotalDisp, uint16_t m_u8Directions);
}
```

2. Resultados

2.1. Reducción de la huella de memoria de la implementación de línea base

Las variables m_u8BlockSize, m_u16P2, m_u8Directions y ct1 son de las utilizadas, por lo tanto, se puede visualizar en las figuras 3 y 4 se muestras el costo de acceso a cada de las instrucciones para cada función. Donde se visualiza que la cantidad se llamadas se mantiene para todas las funciones, pero el costo de acceso para la función *find_minLri* se ve disminuido hasta en un 50 %.

Function	Location	Called	Self Cost: Ir	Incl. Cost: Ir
StereoFPGA::find_minLri(int*)	/home/luis/P...	11524350	10 464 109 800	10 464 109 800
StereoFPGA::cost_computation(int*, int*)	/home/luis/P...	1	777 054 657	11 333 359 257
StereoFPGA::compute_hamming_distance(unsigned long, unsigned long)	/home/luis/P...	910800	530 085 600	530 085 600
StereoFPGA::cost_aggregation(int*, int*)	/home/luis/P...	1	112 933 549	112 933 549
find_min(int, int, int, int)	/home/luis/P...	11524350	92 194 800	92 194 800
StereoFPGA::init_Lr(int*, int*)	/home/luis/P...	1	58 212 045	58 212 045
0x000000000085c550	/usr/lib/x86_...	512	40 764 164	40 764 164
do_lookup_x	/build/glibc-2...	28570	34 492 859	41 702 814
StereoFPGA::calc_disp(int*, cv::Mat*)	/home/luis/P...	1	31 425 427	31 867 404
StereoFPGA::compute_hamming(unsigned long*, unsigned long*, int*)	/home/luis/P...	1	22 825 729	552 911 329
cycle 8	/build/glibc-2...	4167	22 595 891	12 446 721 296
StereoFPGA::compute_census_transform(cv::Mat, unsigned long*)	/home/luis/P...	2	20 391 573	20 391 573
_int_malloc	/build/glibc-2...	102289	16 917 928	19 692 242
0x0000000000020de0	/usr/lib/x86_...	270915	13 906 167	17 493 621
0x0000000000337500	/usr/lib/x86_...	3	13 220 698	13 220 698
free	/build/glibc-2...	164854	12 727 796	12 769 245
strcmp	/build/glibc-2...	343502	11 095 363	11 095 363
malloc	/build/glibc-2...	164110	9 044 975	24 194 679
__strchr_sse2	/build/glibc-2...	248072	8 158 914	8 158 914
_dl_lookup_symbol_x	/build/glibc-2...	20574	6 658 829	48 370 193
__strcmp_sse2_unaligned	/build/glibc-2...	188188	6 479 527	6 479 527

Figura 3: Costo de acceso a lectura de instrucciones gastado sin reducción de la huella de memoria.

Function	Location	Called	Self Cost: Ir	Incl. Cost: Ir
StereoFPGA::find_minLri(int*)	/home/luis...	11524350	5 341 605 000	5 341 605 000
StereoFPGA::cost_computation(int*, int*)	/home/luis...	1	765 238 729	6 199 038 529
StereoFPGA::compute_hamming_distance(unsi...	/home/luis...	910800	530 085 600	530 085 600
StereoFPGA::cost_aggregation(int*, int*)	/home/luis...	1	112 855 282	112 855 282
find_min(int, int, int, int)	/home/luis...	11524350	92 194 800	92 194 800
0x00000000008e9260	/usr/lib/x86...	512	40 764 164	40 764 164
do_lookup_x	/build/glibc...	28656	35 561 325	43 310 949
StereoFPGA::calc_disp(int*, cv::Mat*)	/home/luis...	1	31 423 856	31 865 303
cycle 8	/usr/lib/x86...	5712	24 818 342	7 287 202 361
StereoFPGA::compute_hamming(unsigned lon...	/home/luis...	1	23 736 661	553 822 261
StereoFPGA::compute_census_transform(cv::M...	/home/luis...	2	20 340 105	20 340 105
StereoFPGA::init_Lr(int*, int*)	/home/luis...	1	17 463 744	17 463 744
_int_malloc	/build/glibc...	103162	17 043 044	19 827 837
0x0000000000020de0	/usr/lib/x86...	270915	13 906 167	17 493 621
0x0000000000337500	/usr/lib/x86...	3	13 220 698	13 220 698
free	/build/glibc...	168079	13 006 899	13 051 061
malloc	/build/glibc...	167159	9 189 024	24 431 111
__strchr_sse2	/build/glibc...	248059	8 159 520	8 159 520
strcmp	/build/glibc...	363694	7 653 620	7 653 620
__strcmp_sse2_unaligned	/build/glibc...	189044	6 733 601	6 733 601

Figura 4: Costo de acceso a lectura de instrucciones gastado con reducción de la huella de memoria.

2.2. Optimización del algoritmo SGBM

Para poder cuantificar la optimización se tomó el tiempo de ejecución para la función *ComputeAlgo*. La figura 5 muestra los resultados de las configuraciones.

- Código base - Source
- Reducción de la huella de memoria - Mem Opt
- Técnica multi-threading - Multithreding
- Optimización CUDA - Cuda

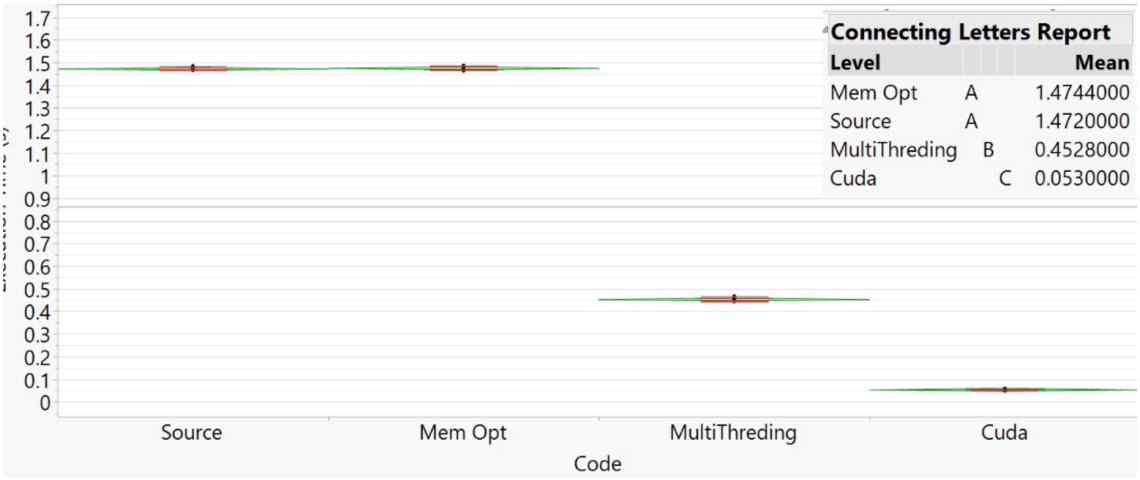

Figura 5: Tiempos de ejecución.

Utilizando como prueba las imágenes de entrada *Prof_L.png* y *Prof_R.png* donde como resultado un tiempo mayor de ejecución de 1.47s para la aplicación sin optimización. Al aplicar la optimización basada en multi-threading se da como resultado un tiempo de 0.452s, esto se debe a la paralelización de los diferentes loops aninados. La paralelización se distribuye en cuatro núcleos los cual visualiza en la figura 7 con lo cual al no ser paralelizadle toda a ejecución tener una mejora en el tiempo de un 30 % menor es un resultado esperado.

Cuando se aplicó la optimización basada en CUDA, da un tiempo de 0.053s, la cual es una reducción muy grande. La posibilidad de dividir la ejecución de los loop anidados en 128 unidades funcióneles hace que la optimización en la ejecución sea muy alta. A pesar de distribuir las mismas funciones que se hicieron en la optimización con multi-threading se observa que el GPU solo llega a necesitar un rendimiento de un 35 % de su capacidad máxima, con cierto rendimiento requerido por el CPU para las otras funciones que no fueron optimizadas para ser ejecutadas con CUDA.

2.3. Problema red convulcional

2.3.1. Detección de restricción vehicular

El problema que se escogió para el estudio de la red convolucional es el de la detección automática de la restricción vehicular por medio del análisis de las placas. Se van a utilizar dos modelos, uno inical que detecta si existen placas en la imagen, y un segundo modelo el cual es el último numero de la placa, en caso de que esta sea una. Para el entrenamiento de la red se utiliza un set de datos genérico llamado MNIST el cual se puede encontrar en la libreria de keras de manera gratuita. La idea del algoritmo es determinar si una placa puede circular o no, a continuación se explicará la implementación y entrenamiento de este código.

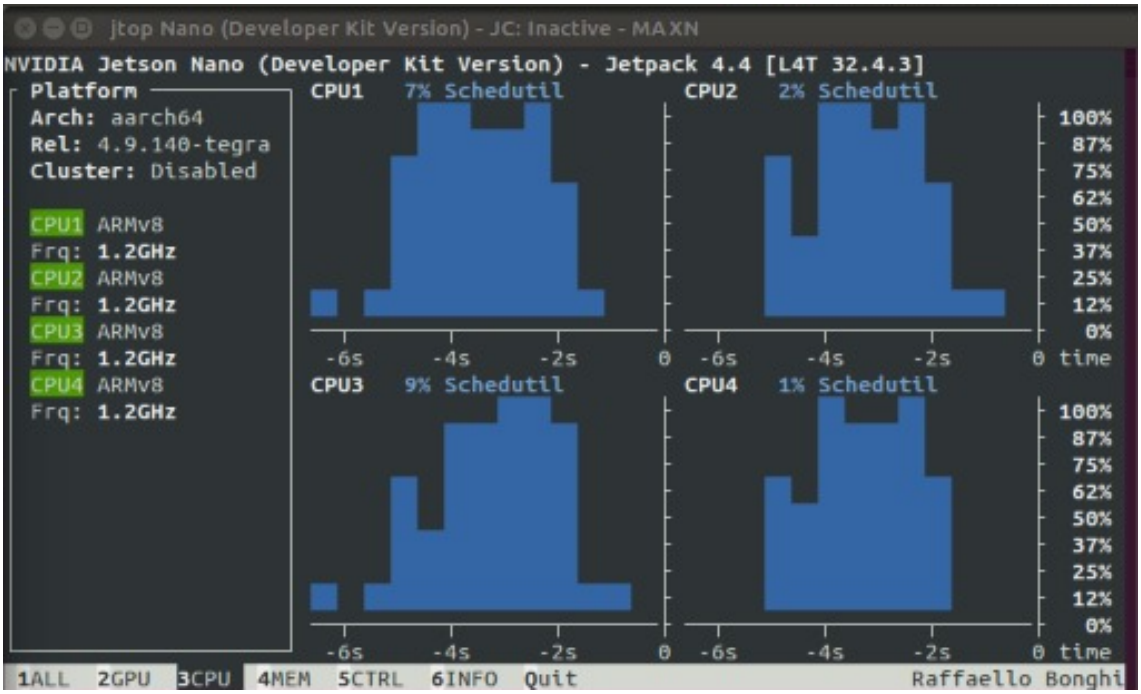

Figura 6: Tiempos de ejecución.

2.3.2. Código en python

Se importan todas las librerías

```
from __future__ import absolute_import, division, print_function, unicode_literals
import cv2
import tensorflow as tf
import numpy as np

import pickle
import pandas as pd
import json
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from tensorflow import keras

from PIL import Image
from skimage.transform import resize

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
```

A continuación se cargan los dos modelos, uno para detectar placas. y el otro para detectar números. Como se puede observar el modelo fue previamente entrenado y ejecutado en una máquina distinta. Se carga únicamente los .ckpt, los cuales contienen los pesos del modelo. En la siguiente sección se explica como se obtienen estos pesos y se entrena al modelo.

```
IMG_SIZE=128
```

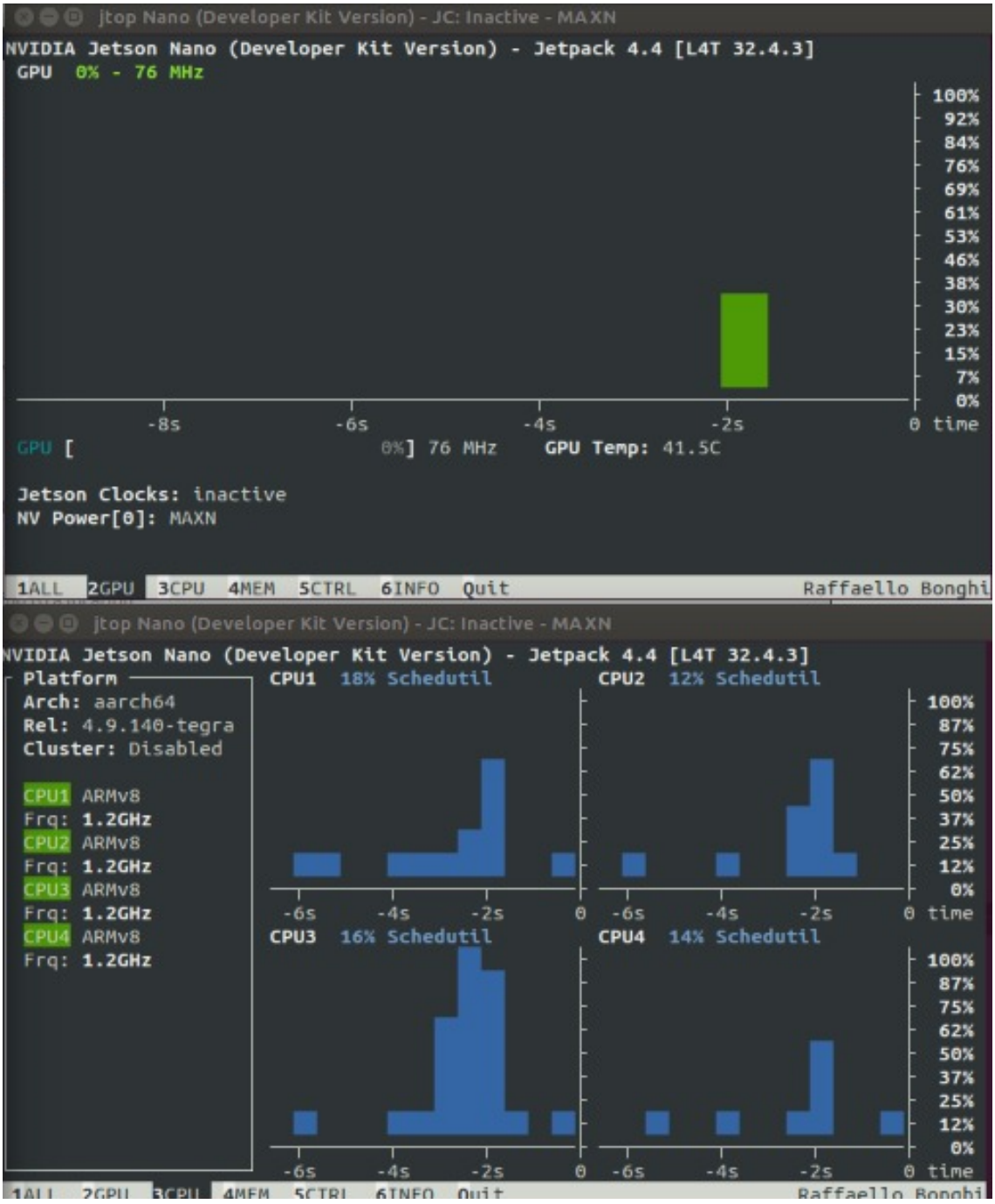

Figura 7: Tiempos de ejecución.

```
def load_first_model():
    # Modelo secuencial, se aaden los layers de la CNN
    model = Sequential()
    model.add(Conv2D(256, (3, 3), input_shape=(128, 128, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
    model.add(Conv2D(256, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(64))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))

    checkpoint_path = "/home/verny/projecto3/jetson/modelo_verny/cp.ckpt"

    model.load_weights(checkpoint_path)
    return model

def load_second_model():
    # Modelo secuencial, se aaden los layers de la CNN
    model = Sequential()
    model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation=tf.nn.relu))
    model.add(Dropout(0.2))
    model.add(Dense(10,activation=tf.nn.softmax))
    checkpoint_path = "/home/verny/projecto3/jetson/modelo_verny/Model_Verny2.ckpt"

    model.load_weights(checkpoint_path)
    return model
```

Las siguientes funciones se utilizan para utilizar la camara y visualizar la imagen y guardarla en un arreglo. Sin embargo, en nuestro experimento no se pudo hacer el experimento en vivo, por lo que se utilizaron imágenes previamente descargadas en la jetson nano. Esto se puede observar en el último bloque de codigo (el main) donde estamos tomando las imágenes previamente cargadas.

```
def get_image_in_format(filepath):
    im = np.array(Image.open(filepath))
    print(im.shape)
    new_array = cv2.resize(im, (IMG_SIZE, IMG_SIZE))
    return new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 3)

def get_image(filepath):
    im = np.array(Image.open(filepath))
    return im
```

A continaución se muestra la lógica que nos indica primero si es placa y además el día en el que puede circular la placa. Esto nos va a ayudar a determinar si el carro cumple o no con la restricción vehicular.

```
def get_second_model_evaluation(image_original,model):
    img_rows, img_cols = 28, 28
    image_original_gray = rgb2gray(image_original)
    image_original_gray_res = resize(image_original_gray, (28,28))

    image_index = 3333
    plt.imshow(image_original_gray_res.reshape(28, 28),cmap='Greys')
    pred = model.predict(image_original_gray_res.reshape(1, img_rows, img_cols, 1))

    dia_semana = 0 # El dia es lunes
    pred = model.predict(image_original_gray_res.reshape(1, img_rows, img_cols, 1))
```

```
if pred.argmax() == 0:
    if dia_semana == 0:
        print("El carro no cumple con la restriccion vehicular")
    elif dia_semana == 1:
        print("El carro no cumple con la restriccion vehicular")
    else:
        print("El carro cumple con la restriccion vehicular")
if pred.argmax() == 1:
    if dia_semana == 0:
        print("El carro no cumple con la restriccion vehicular")
    elif dia_semana == 1:
        print("El carro no cumple con la restriccion vehicular")
    else:
        print("El carro cumple con la restriccion vehicular")

if pred.argmax() == 2:
    if dia_semana == 2:
        print("El carro no cumple con la restriccion vehicular")
    elif dia_semana == 3:
        print("El carro no cumple con la restriccion vehicular")
    else:
        print("El carro cumple con la restriccion vehicular")
if pred.argmax() == 3:
    if dia_semana == 2:
        print("El carro no cumple con la restriccion vehicular")
    elif dia_semana == 3:
        print("El carro no cumple con la restriccion vehicular")
    else:
        print("El carro cumple con la restriccion vehicular")

if pred.argmax() == 4:
    if dia_semana == 4:
        print("El carro no cumple con la restriccion vehicular")
    elif dia_semana == 5:
        print("El carro no cumple con la restriccion vehicular")
    else:
        print("El carro cumple con la restriccion vehicular")
if pred.argmax() == 5:
    if dia_semana == 4:
        print("El carro no cumple con la restriccion vehicular")
    elif dia_semana == 5:
        print("El carro no cumple con la restriccion vehicular")
    else:
        print("El carro cumple con la restriccion vehicular")

if pred.argmax() == 6:
    if dia_semana == 6:
        print("El carro no cumple con la restriccion vehicular")
    elif dia_semana == 7:
        print("El carro no cumple con la restriccion vehicular")
    else:
        print("El carro cumple con la restriccion vehicular")
if pred.argmax() == 7:
    if dia_semana == 6:
        print("El carro no cumple con la restriccion vehicular")
    elif dia_semana == 7:
        print("El carro no cumple con la restriccion vehicular")
```

```
        else:
            print("El carro cumple con la restriccion vehicular")

    if pred.argmax() == 8:
        if dia_semana == 8:
            print("El carro no cumple con la restriccion vehicular")
        elif dia_semana == 9:
            print("El carro no cumple con la restriccion vehicular")
        else:
            print("El carro cumple con la restriccion vehicular")
    if pred.argmax() == 9:
        if dia_semana == 8:
            print("El carro no cumple con la restriccion vehicular")
        elif dia_semana == 9:
            print("El carro no cumple con la restriccion vehicular")
        else:
            print("El carro cumple con la restriccion vehicular")

    plt.imshow(image_original_gray_res.reshape(28, 28),cmap='Greys')
    return pred.argmax()
```

A continuación se presenta el bloque main, el cual cuenta con funciones importantes como el de cargar la imagen a utilizar, el recortado de la misma para obtener la última posición de la placa (esto es una aproximación y se realizaron varias iteraciones para concluir la mejor aproximación).

```
def main():
    first_model=load_first_model()
    second_model=load_second_model()
    #metodo esttico para agregar imagen
    plate_image=get_image_in_format('/home/verny/projecto3/model_inputs/plate1.jpeg')
    raw_image=get_image('/home/verny/projecto3/model_inputs/plate1.jpeg')
    prediction_upon_image_captured=first_model.predict(plate_image)
    if prediction_upon_image_captured==0:
        print("Es una placa, hora de buscar la ultima letra")
        #manera de recortar la imagen para obtener la ultima posicion de la placa
        x_point_top = raw_image.shape[1]/8*6
        y_point_top = 0
        x_point_bot = raw_image.shape[1]
        y_point_bot = raw_image.shape[0]
        plateImage = Image.fromarray(raw_image).crop((x_point_top, y_point_top, x_point_bot, y_point_bot))
        fig, ax = plt.subplots(1, 1, constrained_layout=True)
        print(type(plateImage))
        plate_number=get_second_model_evaluation(np.asarray(plateImage),second_model)
        print(plate_number)
    else:
        print("No es placa")

if __name__== "__main__":
  main()
```

En general, las imágenes que se utilizaron para el estudio de las placas son las siguientes, y además como se observa su salida. Se puede ver como el algoritmo busca una imagen del set de datos previamente cargada, y como cada una de estas imágenes tienen un tag de identificación se puede saber cual es el número final de la placa.

2.3.3. Entrenamiento de la red

Se importa el set de datos con el que se va a trabajar.

Figura 8: Imagen placa con número final 8.

```
El carro cumple con la restriccion vehicular
8
```

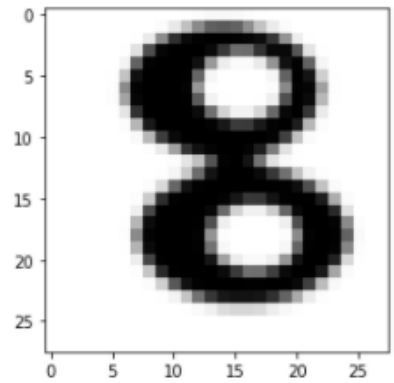

Figura 9: Resultado al procesar la figura anterior por el modelo CNN.

```
import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

Se verifica que el set de datos se muestre correctamente.

```
import matplotlib.pyplot as plt
#Verificar que el set de datos se cargara correctamente
image_index = 1000
print(y_train[image_index])
plt.imshow(x_train[image_index], cmap='Greys')
```

A continuación se van a manejar los datos de tal manera que sean de un mismo tamañi y puedan alimentar al modelo.

```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
```

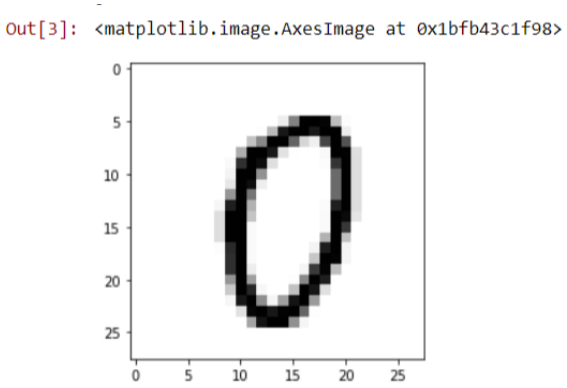

Figura 10: Imagen aleatoria del set de datos MNIST

```
# convierte los datos en flotantes
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# Se normalizan los datos para obtener unicos tamanos.
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print('Number of images in x_train', x_train.shape[0])
print('Number of images in x_test', x_test.shape[0])
x_train shape: (60000, 28, 28, 1)
Number of images in x_train 60000
Number of images in x_test 10000
```

Como ya se mencionó anteriormente se decide trabajar con Keras. En resumen, Keras es una biblioteca de redes neuronales escrita en python y se ejecuta sobre TensorFlow. A continuación se muestra como se importan los modelos que vamos a utilizar de Keras para CNNs.

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
# Se crea el modelo secuencial de layers
model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten()) # Flattening para la salida de las layers
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(10,activation=tf.nn.softmax))
```

En este punto es donde se compila y exporta el modelo a los .cpk que serán utilizados en el modelo de la jetson. De manera muy similar se realizar el proceso par aidentificar si es placa o no.

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

checkpoint_path = (r"C:\Users\vernyjmo\Google Drive\Maestria\HPCE\Proyecto_Final\num_predictor.ckpt")
checkpoint_dir = os.path.dirname(checkpoint_path)
cp_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path, save_weights_only=True, verbose=1)
```

```
model.fit(x=x_train,y=y_train, epochs=3, callbacks=[cp_callback])
```

Referencias

[1]"Tutorial 01: Say Hello to CUDA - CUDA Tutorial." https://cuda-tutorial.readthedocs.io/en/latest/tutorials/tutorial01/ (accessed Aug. 30, 2020).

[2]"Keras: the Python deep learning API." https://keras.io/ (accessed Aug. 30, 2020).

[3]R. Lopez, "Redes neuronales convolucionales con TensorFlow." https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/ (accessed Aug. 30, 2020).

[4]R. Friedman, "Reference Guides," OpenMP. https://www.openmp.org/resources/refguides/ (accessed Aug. 30, 2020).

[5]M. Zăvoian, mihaits/Qt-CUDA-example. 2020.

[6]K. Willems, "TensorFlow Tutorial For Beginners," DataCamp Community, Jan. 16, 2019. https://www.datacamp.com/comm tutorial (accessed Aug. 30, 2020).

[7]E. Dulshan, "1D, 2D and 3D thread allocation for loops in CUDA," Medium, Aug. 26, 2018. https://medium.com/@erangadu 2d-and-3d-thread-allocation-for-loops-in-cuda-e0f908537a52 (accessed Aug. 30, 2020).

[8]A. Sharma, "Convolutional Neural Networks in Python," DataCamp Community, Dec. 05, 2017. https://www.datacamp.com neural-networks-python (accessed Aug. 30, 2020).

[9]Pablo, "Qt Creator + CUDA + Linux – Review," cudaspace, Jul. 05, 2012. https://cudaspace.wordpress.com/2012/07/05/qt creator-cuda-linux-review/ (accessed Aug. 30, 2020).

[10]G. Ruetsch and B. Oster, "Getting Started with CUDA," p. 65, 2008.