UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería Electrónica

Implementación mediante OpenCL de Convolutional Neural Networks sobre plataformas FPGA de bajo consumo

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

AUTOR/A: Rocabado Rocha, José Luís

Tutor/a: Gadea Gironés, Rafael

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

IMPLEMENTACIÓN MEDIANTE OPENCL DE CONVOLUTIONAL NEURAL NETWORKS SOBRE PLATAFORMAS FPGA DE BAJO CONSUMO

Autor: Jose Luis Rocabado Rocha

Tutor: Rafael Gadea Gironés

Trabajo Fin de Máster presentado en el Departamento de Ingeniería Electrónica de la Universitat Politècnica de València para la obtención del Título de Máster Universitario en Ingeniería de Sistemas Electrónicos

Curso 2022-23

Valencia, Julio de 2023

Resumen

El incesante desarrollo de arquitecturas de *Deep Learning*, cuyas arquitecturas requieren de máquinas con una gran potencia de computación, es necesario un nuevo enfoque para acercar las grandes ventajas de la inteligencia artificial a la electrónica con recursos limitados, donde premia el bajo coste y bajo consumo.

Resum

Degut al incesant desenvolupament de arquitectures de *Deep Learning*, amb arquitectures que requereixen dispositius amb una gran potencia de computació, es necesari un nou punt de vista per a apropar els grans avantatges de la intel·ligencia artificial a la eléctrónica amb recursos limitats, on se valorá mes el baix cost i el baix consum.

Abstract

The ongoing development of Deep Learning architectures, whose architectures require machines with significant computing power, requires a new approach to bring the great advantages of artificial intelligence to electronics with limited resources, where low cost and low consumption are rewarded.

El presente trabajo se dedica principalmente a mis padres que me han mostrado siempre su apoyo de forma incondicional. Gracias a ellos poseo los valores necesarios que me convierten en la persona que soy. También se lo dedico a mi tutor, Rafael Gadea Gironés, por su gran implicación y soporte mostrado a lo largo del desarrollo del proyecto. Finalmente, se lo quiero dedicar a mis compañeros y amigos. En especial a aquellas personas que han compartido momentos conmigo en la Casa del Alumno facilitando y amenizando las jornadas de trabajo.

Índice general

Índice de figuras

Índice de tablas

Listado de siglas empleadas

2D 2 Dimensiones.

ALM Módulo Lógico Adaptativo (Adaptative Logic Module).

API Interfaz de Programación de Aplicaciones (Application Programming Interface).

BSP Board Support Package.

CNN Red Neuronal Convolucional (Convolutional Neural Network).

CPU Unidad Central de Procesamiento (Central Processing Unit).

DL Deep Learning.

DSP Procesamiento Digital de Señales (Digital Signal Processing).

FPGA Matriz de Puertas Lógicas Programable en Campo (Field-Programmable Gate Array).

GPU Unidad de Procesamiento Gráfico (Graphics Processing Unit).

HDL Hardware Description Lenguage.

HLS High Level Synthesis.

HPS Hard Processor System.

ICD Installable Client Driver.

IDE Entorno de Desarrollo Integrado (Integrated Development Environment).

II Intervalo de Iniciación.

IP Propiedad Intelectual (Intelectual Property).

MACC Multiplicador Accumulador.

OS Sistema Operativo (Operating System).

SDK Kit de Desarrollo de Software (Software Development Kit).

SIMD Single Instruction, Multiple Data.

SoC System on Chip.

Parte I

Memoria

Capítulo 1

Introducción y objetivos

A lo largo de los últimos años, se ha podido observar un auge de las tecnologías basadas en inteligencia artificial mediante algoritmos de *machine learning*. Las áreas de aplicación de estas tecnologías son infinitas. Desde clasificación de imágenes o reconocimiento de objetos para un sin fin de objetivos como soporte en el diagnóstico de enfermedades o el control de calidad, hasta generación de texto, siendo un claro ejemplo de esto el conocido ChatGPT.

Sin embargo, estas tecnologías dependen fuertemente de máquinas con una potencia de computación considerable, por lo que normalmente el procesado suele llevarse a cabo en servidores en la nube para facilitar el acceso al usuario final. No obstante, si queremos acercar la inteligencia artificial a la electrónica, este enfoque no es el adecuado debido a restricciones en latencia, consumo y tamaño.

En respuesta a ello, y gracias en las mejoras en los últimos procesadores, aparece un nuevo enfoque conocido como inteligencia artificial *on the edge*. La filosofía de este enfoque consiste en poder llevar el procesado de los datos adquiridos en un nodo dentro del mismo nodo, de forma que pueda actuar una vez obtenido los resultados, otorgándole independencia a la electrónica.

Por lo tanto, es posible encontrar nuevas arquitecturas cuyo objetivo es conseguir las mismas funcionalidades que el resto de tecnologías, si bien a cambio de reducir la exactitud de los resultados, reduciendo su complejidad.

En este trabajo nos centraremos en arquitecturas basadas en convoluciones, *convolutional neural networks* debido a que son una tendencia clara en el panorama actual de la computación *on the edge*. Este tipo de arquitecturas permiten extraer rasgos o características de las imágenes de entrada, siendo capaces de clasificarlas o de detectar diferentes objetos.

Podemos encontrar diferentes arquitecturas de este estilo tales como Darknet o VGG. Podemos encontrar también una opción interesante conocida como SqueezeNet v1.1, que podemos encontrar en el repositorio oficial en [1]. Esta arquitectura consiste en una nueva versión de [2] manteniendo la misma precisión y reduciendo el coste computacional por un factor 2.4. Por todo esto, el siguiente trabajo pretende realizar una implementación de la *convolutional neural network*, SqueezeNet v1.1, en plataformas FPGA SoC de bajo coste, en específico la placa de desarrollo DE10-Nano. Por lo tanto, nuestro sistema se compondrá de un procesador HPS ARM coexistiendo con la aceleración *hardware* a diseñar.

Para reducir el tiempo de diseño, se recurrirá a compiladores de alto nivel HLS mediante el

cual se aumenta el nivel de abstracción para generar la síntesis del *hardware* mediante un código de C/C++. Una de las opciones más interesantes proporcionadas por el fabricante que nos permite aprovechar las capacidades de las FPGAs para optimizar los resultados con eficiencia energética y baja latencia es Intel® FPGA SDK para OpenCL™.

Gracias a esta herramienta seremos capaces de implementar nuestro algoritmo de clasificación de imágenes, infiriendo un *hardware* de aceleración con capacidad de paralelismo, tal y como establece el estándar de OpenCL, dentro de la placa de desarrollo de bajo coste, DE10-Nano.

En la figura 1.1 podemos encontrar un diagrama conceptual que resume lo mencionado anteriormente.

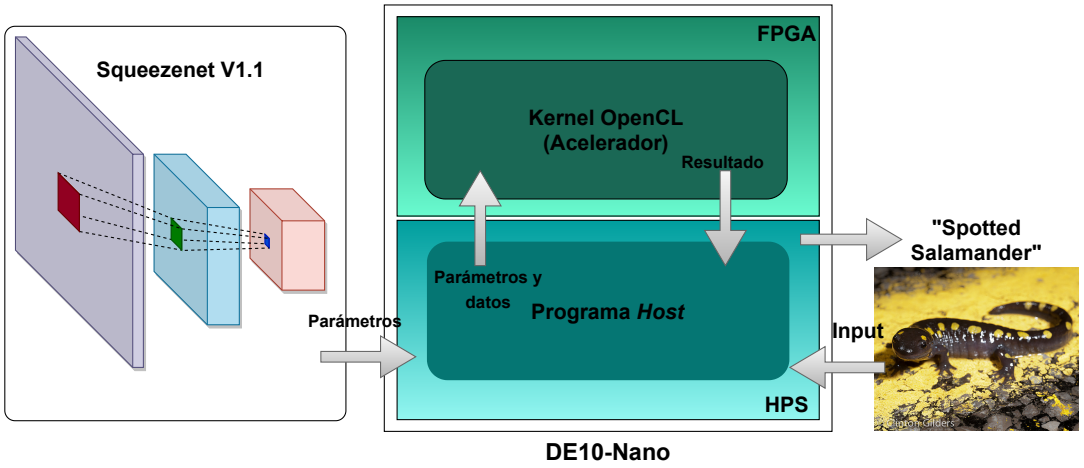

Figura 1.1: Diagrama conceptual del objetivo principal.

Además, como suele ser común en el ámbito del *machine learning*, se creará una infraestructura de validación basada en Python y específicamente mediante Jupyter Notebooks. En dicha estructura se confeccionarán los diferentes bancos de prueba que nos permitirán validar nuestros diseños. Por lo tanto, podemos establecer los siguientes ítems como objetivos del presente trabajo:

- Implementación de la CNN SqueezeNet V1.1, mediante el HLS de Intel basado en OpenCL, en la placa de desarrollo DE10-Nano, una FPGA de bajo coste y bajo consumo.
- Creación de una infraestructura de diseño y validación de los aceleradores *hardware diseñados* mediante Jupyter Notebook y Python.
- Creación de los bancos de prueba que nos permitirán validar los distintos diseños.
- Estudio comparativo de las aproximaciones sugeridas por Intel para la inferencia de aceleradores *hardware* con el HLS de OpenCL.

Capítulo 2

Revisión de la literatura y marco teórico

En la siguiente sección, se explicará y analizará la arquitectura CNN seleccionada, SqueezeNet, para asentar las bases sobre las que se desarrollará este trabajo. Posteriormente, se presentarán los diferentes proyectos previos a este trabajo, conformando de esta forma la revisión literaria. Por último, se expondrá el marco teórico, desde la operación básica de la arquitectura de DL, la convolución, hasta las diferentes opciones que nos ofrece la herramienta de trabajo, Intel® FPGA SDK para OpenCL™, para llevar a cabo la implementación del acelerador *hardware*.

2.1. Squeezenet

SqueezeNet, [2], es una arquitectura CNN cuyo objetivo principal ha sido reducir el número de parámetros sin llegar a afectar significativamente al nivel de precisión de las imágenes clasificadas. El hecho de reducir el número de parámetros de la red convolucional, hace que su implementación sea viable en FPGA o en sistemas embebidos. Además, permite guardar los parámetros en la memoria *on-chip* reduciendo el cuello de botella generado por el ancho de banda del acceso a memoria.

Para llevar a cabo esto, en [2] se presentan los módulos Fire que compondrán la arquitectura global. Estos módulos se conforman con una capa Squeeze, correspondiente a una convolución 1x1,cuya salida conectará directamente con una capa Expand, formado por una mezcla de convoluciones 1x1 y 3x3, cuyos resultados concatenados generarán la salida del módulo.

Por un lado, los módulos Fire consiguen reducir los parámetros de la arquitectura debido al uso de convoluciones 1x1 que reducen en un factor 9 los parámetros si se utilizarán filtros 3x3.

Por otro lado, el hecho de utilizar una capa Squeeze antes de la convolución 3x3, permite reducir el número de canales de entrada, reduciendo también el número de filtros usados. En la figura 2.1 podemos encontrar la disposición de las diferentes convoluciones en un módulo Fire.

Posteriormente, se presenta una nueva versión de SqueezeNet disponible en el repositorio oficial [1], SqueezeNet v1.1. Esta versión modifica la arquitectura original, manteniendo los módulos Fire, reduciendo ligeramente el número de parámetros y obteniendo una reducción de 2.4 en el coste de computación sin llegar a perder precisión.

Una comparativa entre versiones del error de las clasificaciones para el modelo preentrenado,

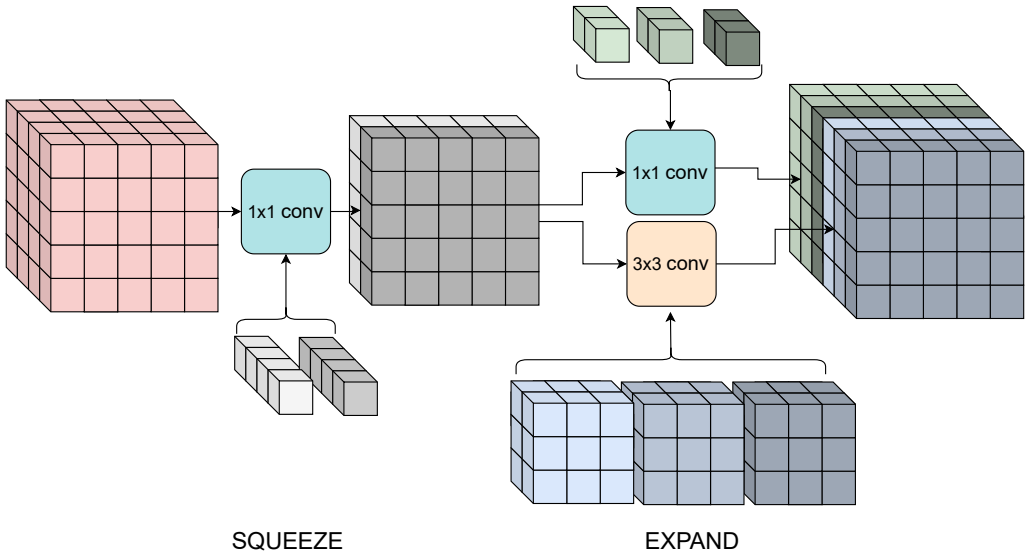

Figura 2.1: Estructura del módulo Fire conformado por los diferentes filtros de convolución. Para el ejemplo tenemos un factor Squeeze, S_{1x1}, de 2; un factor Expand de la convolución 1x1, E_{1x1}, de 3 y un factor Expand de la convolución 3x3, E_{3x3} de 3.

con las imágenes de Imagenet, se encuentra disponible en la documentación de la librería de código abierto para aprendizaje automático PyTorch [3]. Podemos encontrar los resultados en la tabla 2.1.

Estructura del módelo	Error Top-1	Error Top-5
SqueezeNet V1.0	41.90 %	19.58 %
SqueezeNet V1.1	41.81 %	19.38 %

Tabla 2.1: Tabla comparativa de los errores de predicción de las versiones de SqueezeNet con el set de datos de Imagenet. Datos obtenidos de [3].

Finalmente, en la tabla 2.2 podemos encontrar la estructura de SqueezeNet V1.1.

Nombre/tipo de capa	Tamaño de salida	*size*/*stride* del filtro (no módulos Fire)	S_{1x1}	E_{1x1}	E_{3x3}
imagen de entrada	224x224x3				
conv1	111x111x64	3x3/2 (x64)			
maxpool1	55x55x64	3x3/2			
fire1	55x55x128		16	64	64
fire2	55x55x128		16	64	64
maxpool2	27x27x128	3x3/2			
fire3	27x27x256		32	128	128
fire4	27x27x256		32	128	128
maxpool3	13x13x256	3x3/2			
fire5	13x13x384		48	192	192
fire6	13x13x384		48	192	192
fire7	13x13x512		64	256	256
fire8	13x13x512		64	256	256
conv2	13x13x1000	1x1/1 (x1000)			
avgpool1	1x1x1000	13x13/1			

Tabla 2.2: Tabla con la arquitectura del modelo SqueezeNet V1.1. Los datos se han obtenido del archivo de Google Colab en conjunto con el archivo de descripción de modelos, ".pth", accesibles en [3].

Es importante remarcar que después de cada convolución tenemos la función de activación ReLU. En cuanto a los módulos Fire, la convolución 3x3 se configura con un *stride* y un *padding* de 1 mientras que la convolución 1x1 solo se configura con un *stride* de 1. Podemos comprobar esto en los tamaños de salida de los módulos Fire de la tabla 2.2.

2.2. Trabajos relacionados

Poco después de la aparición del modelo SqueezeNet, David Gschwend [4] realiza una implementación del modelo en una plataforma de desarrollo de bajo coste Zynqbox que incluye una FPGA Xilinx Zynq-7000 SoC pues combina además una ARM Cortex-A9. La implementación llevada a cabo, que recibe el nombre de ZynqNet, se divide en dos partes.

Por un lado, se realizan modificaciones al modelo original de SqueezeNet para adaptar el modelo a los requerimientos de la FPGA. Una de las modificaciones más notable consiste en la eliminación de las capas maxpool, siguiendo la filosofía de "*all-convolutional networks*" donde la red estará compuesta únicamente por capas convolucionales. Para obtener igualmente el mismo resultado en la reducción del tamaño de salida, añaden un *stride* de 2 en la convolución inmediatamente posterior. Sin embargo, en SqueezeNet esta capa consistirá siempre en una convolución 1x1, lo que resultaría en perdida de información. Por ello sustituyen esa primera capa por un filtro 3x3. Implementan también la capa final de tipo *average pool* que podría considerarse una convolución cuyos filtros se componen de los valores $kernel_{size}^{-1}$.

Otra modificación que cabe mencionar consiste en el redimensionamiento del tamaño de las capas, haciendo que su altura y anchura sean potencias de 2.
Por otro lado, en la implementación del acelerador se observa como el autor se decanta por el uso de coma flotante de simple precisión para la representación de los datos para mantener la compatibilidad con su implementación de ZynqNet en GPU.

Además, se inclinan con el uso de memorias caché para facilitar la lectura y escritura de memoria y también optan por desenrollar los bucles de las convoluciones 3x3, llegando así a conseguir un error top-5 sobre Imagenet del 15.4 % usando una representación de datos de coma flotante de 32 bits. Podemos observar el diagrama de bloques de su implementación final en la figura 2.2. Observamos que el acelerador está optimizado para realizar convoluciones 3x3 y que posiblemente cualquier otra configuración requerirían de lógica adicional, resultando en una baja utilización de los recursos MACC 3x3. Cabe remarcar que han realizado su implementación mediante el HLS de Vivado.

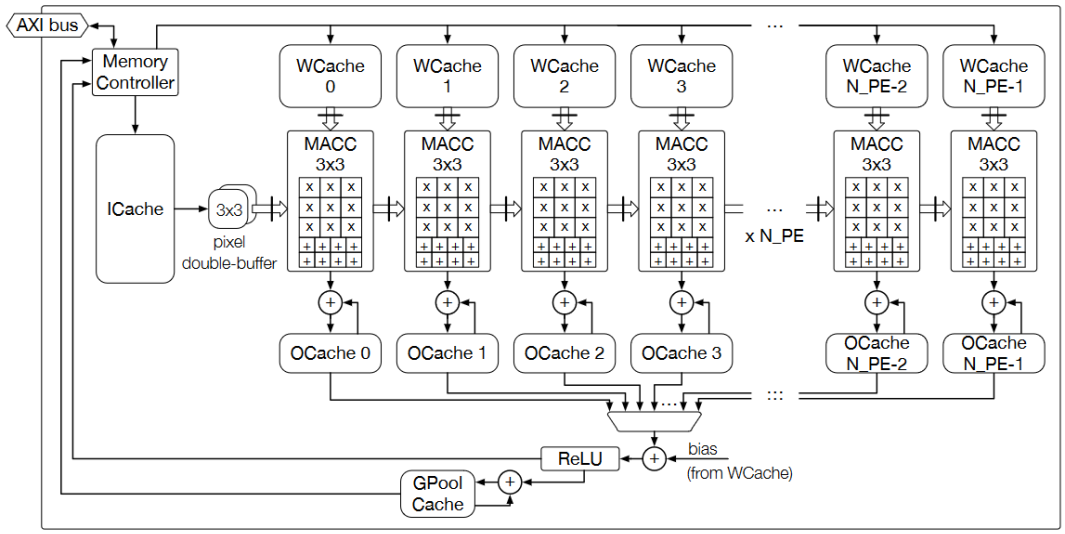

Figura 2.2: Estructura del acelerador implementado en ZynqNet, imagen obtenida de [4].

Un trabajo similar a ZynqNnet se lleva a cabo en [5], dando como resultado EdgeNet. Esta propuesta de red convolucional se implementa en una placa DE10-Nano y su propuesta de acelerador consiste en un bloque de computación configurable. La unidad diseñada implementa un módulo Fire de manera inversa, es decir, la entrada del acelerador equivale a una capa de tipo Expand siendo la salida la capa Squeeze. De esta forma, dado que la entrada de la unidad de computación será mayormente el resultado de una convolución 1x1 de la capa Squeeze, pueden reducir el acceso a memoria debido a que hay menos canales de entrada. Lo mismo aplica a la salida, puesto que la salida de la unidad corresponde con la capa Squeeze.

Implementa también las capas de tipo pool que serán accesibles según la configuración del *datapath*. Podemos ver la arquitectura del bloque computacional en la figura 2.3

Cabe destacar que los resultados obtenidos son con representación de los datos y parámetros usando coma flotante de 8 bits (5 bits de exponente y 2 de mantisa) de los parámetros resultantes del entrenamiento de [4], llegando a obtener una reducción del 6 % de exactitud en las clasificaciones consiguiendo un 51 % de precisión top-1 con Imagenet.

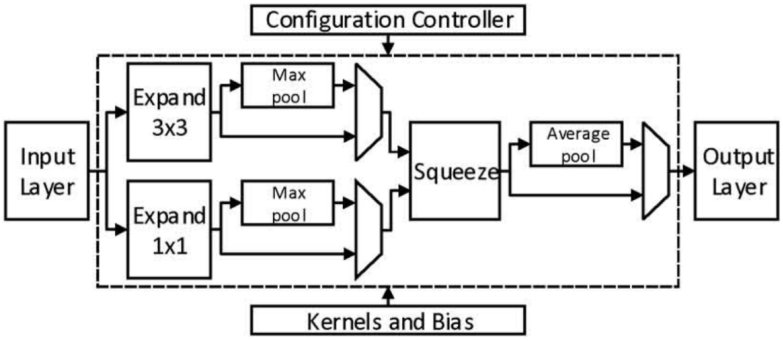

Figura 2.3: Estructura del acelerador implementado en EdgeNet, imagen obtenida de [5].

Siguiendo con la cuantificación de los parámetros de las redes neuronales convolucionales, encontramos nuevamente en [6] una implementación de SqueezeNet V1.1 donde previamente al diseño del acelerador realizan un estudio del efecto de la cuantificación en la precisión del modelo.

Observan que usando el tipo de datos entero de 8 bits, int8, consiguen reducir considerablemente tanto los recursos de la FPGA como los accesos a memoria, obteniendo una perdida del 0.69 % y 0.72 % resultando en un 57.49 % y un 79.90 % en la precisión top-1 y top-5 respectivamente.

Cabe mencionar que esta implementación se lleva a cabo en la placa de bajo coste DE10-Nano con el uso del estándar de OpenCL y HLS.

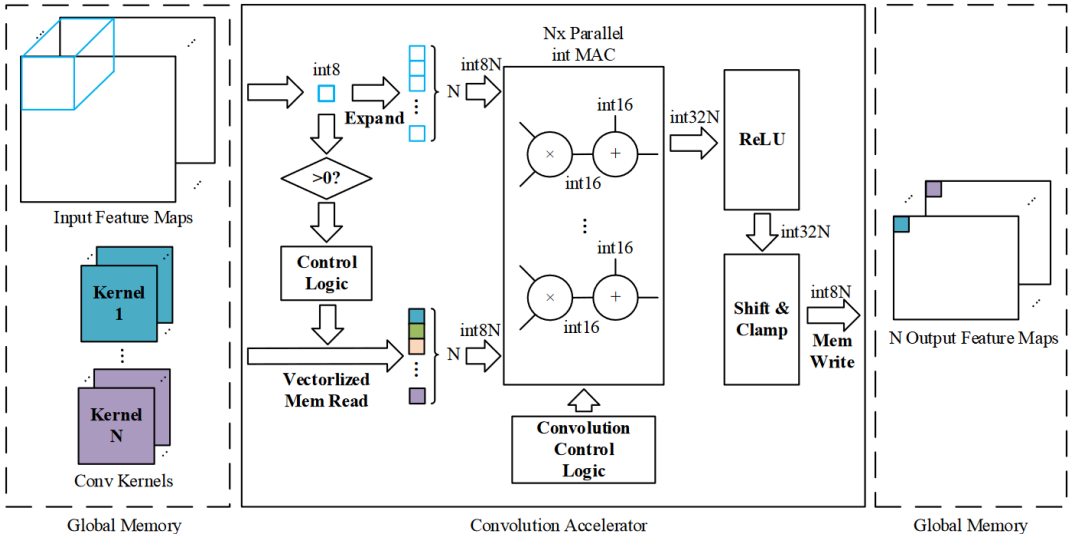

Figura 2.4: Estructura del acelerador implementado en [6]. Imagen obtenida de la misma referencia.

La arquitectura presentada (figura 2.4) se caracteriza principalmente por un paralelismo controlado por la variable N, que replica los bloques de multiplicación-acumulación con cuantificación que llevan a cabo la convolución.

Además, incorporan lógica adicional que permite gestionar la lectura de los datos de entrada, los filtros de la convolución y la escritura de los resultados en la memoria global. También, debido a que la función de activación ReLU genera mapas de características con resultados iguales a 0,

introducen un control específico con el que evitan realizar operaciones con valores de entrada nulas.

Posteriormente, en [7] podemos encontrar un proyecto donde desarrollan, nuevamente sobre la placa SoC DE10-Nano, la implementación de SqueezeNet V1.1 infiriendo paralelismo a nivel de multiples hilos, usando el HLS de OpenCL.

Poseen una documentación detallada de los pasos seguidos para llevar a cabo su diseño, así como diferentes versiones donde aplicando técnicas de optimización de *kernels* como el acceso de memoria coalescente y el *loop unrolling* consiguen ir reduciendo el tiempo requerido en la predicción de la red. Revisando la implementación, se observa que el hecho de usar memoria coalescente mediante el uso de la estructura de datos float4 limitan el número de canales de cada capa de convolución, pues deben de ser múltiplo de 4 debido a que es el número de datos que se leen de forma coalescente.

Lo más remarcable sería el método de depuración del código de OpenCL mediante un documento de *Jupyter Notebook*, el API de OpenCL implementado en Python, PyOpenCL, y la librería de aprendizaje automático Pytorch.

De esta forma, son capaces de depurar al mismo tiempo la ejecución del *host* y la implementación del *Kernel*, puesto que:

- El modelo de Squeezenet es fácilmente instanciable gracias a Pytorch convirtiéndose así en un buen modelo de referencia o *Golden Model*.
- Los resultados de cada nodo del modelo de Pytorch pueden ser convertidos a matrices de NumPy, facilitando una comparación capa a capa para la depuración del diseño.
- La implementación del *host* es más flexible y *user-friendly* mediante PyOpenCL.

2.3. Marco teórico

En esta sección se establecerán los fundamentos teóricos en los que se basa el presente trabajo.

Se explica el funcionamiento básico de las capas de convolución, que son parte fundamental en las CNN, con el fin de conocer las particularidades que este tipo de capas presentan, con el fin de realizar un correcto diseño e implementación.

Se comenta, también, otro tipo de capas que son comúnmente utilizadas en arquitecturas convolucionales y que, además, son utilizadas en el modelo seleccionado, SqueezeNet V1.1.

Finalmente, se discute sobre el estándar de OpenCL y sus dos puntos principales de enfoque que propone para abordar los diferentes tipos de problemas, desde un punto de vista de soluciones en FPGA.

2.3.1. Capa de convolución

La capa de convolución es una capa utilizada normalmente para procesar entradas o *feature maps* cuya distribución espacial es de dos dimensiones, como es el caso de las imágenes cuyo tamaño viene expresado normalmente por un ancho y un alto de píxeles, $H_{pixel} \times W_{pixel}$.

Sin embargo, las imágenes normalmente suelen tener diferentes canales, que contienen información del color en el formato *Red-Green-Blue*. Por lo tanto, las entradas a las capas de convolución suelen tener una dimensión $H_{in} \times W_{in} \times CH_{in}$, siendo, respectivamente, la altura, el ancho y la profundidad de canal del *feature map* de entrada.

En consecuencia, las capas de convolución 2D están formadas por filtros con dimensión $H_k \times W_k \times CH_{in}$, donde los canales de los diferentes filtros son iguales a los canales de la entrada para mantener la concordancia. En cuanto al tamaño de los filtros, $H_k \times W_k$, podemos encontrar diferentes combinaciones, siendo muy comunes los tamaños 1x1,3x3 y 5x5. Otro parámetro que forma parte de las convoluciones es el denominado *bias*. Consiste en un valor escalar por cada filtro de la convolución.

La figura 2.5 representa de forma visual las dimensiones, con su nomenclatura, de un *feature map* y un filtro.

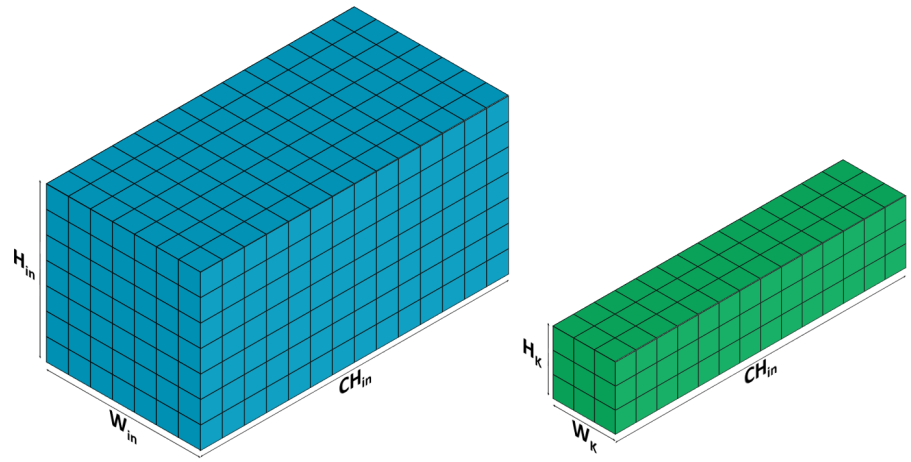

Figura 2.5: Representación de un *feature map* y un filtro con la nomenclatura de sus dimensiones.

El resultado de la convolución se corresponderá con la suma del producto del filtro con los datos de entrada elemento a elemento. El filtro se irá deslizando a través del tamaño de los datos de entrada, produciendo un valor de salida por cada desplazamiento. En caso de haber más de un canal se aplicará el filtro a su canal correspondiente y se sumarán los resultados de todos los canales.

Supongamos un *feature map* de dimensiones $Hin \times W_{in} \times 1$ y un único filtro $H_k \times W_k \times 1$. El primer elemento de la salida corresponderá con:

$$out(0,0) = bias + \sum_{i=0}^{W_k} \sum_{j=0}^{H_k} filtro(i,j) \cdot feature(i,j) \tag{2.1}$$

El tamaño de la salida dependerá principalmente del número de filtros utilizados en la convolución, del factor de deslizamiento o *stride* y de una configuración opcional conocido como *padding* que permite añadir un marco, normalmente de ceros, alrededor de la entrada.

Siendo la entrada y el filtro cuadrados con tamaños in_{size} y k_{size} respectivamente, podemos calcular el tamaño de la salida out_{size} como:

$$out_{size} = \frac{in_{size} + 2 \cdot padding - k_{size}}{stride} + 1 \tag{2.2}$$

Donde *padding* es el tamaño del marco añadido y *stride* son las posiciones que se desplaza el filtro sobre los datos de entrada.

En cuanto al número de canales de la salida, dependerá del número de filtros que aplica la capa de convolución.

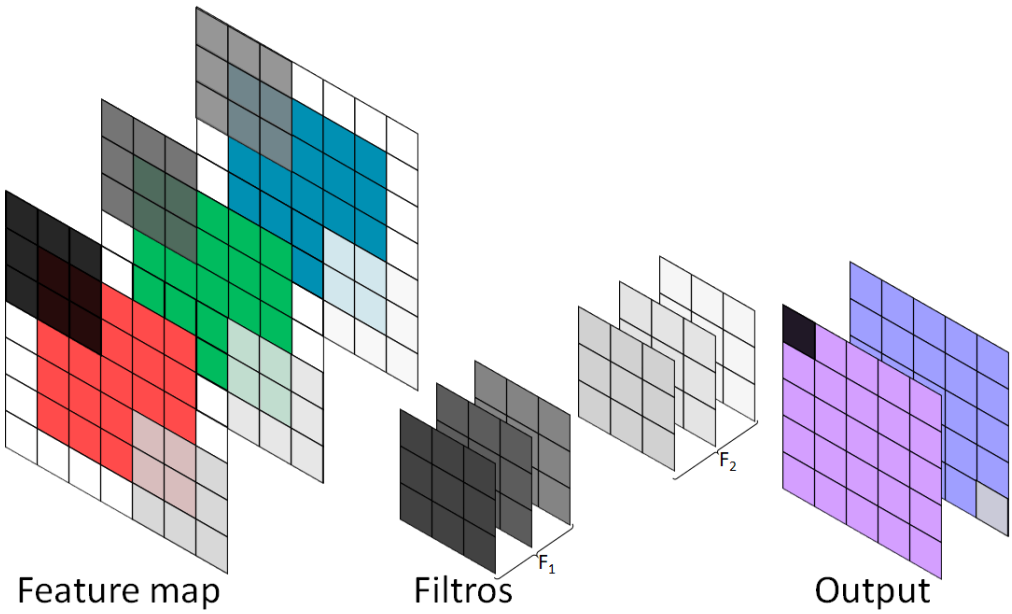

Figura 2.6: Ejemplo de convolución 3x3 con una imagen de entrada de dimensiones 5x5x3 y dos filtros (sin *bias*) con *padding* y *stride* 1.

En la figura 2.6 podemos encontrar un ejemplo visual de una operación de una capa de convolución. Se observa un *feature map* de entrada con tamaño in_{size} de 5 y un filtro con k_{size} de 3, pues se trata de una convolución 3x3, con un *stride* y *padding* de 1.

Aplicando la formula 2.2, podemos comprobar que el tamaño de la salida, out_{size} es de 5. Además, dado que tenemos dos filtros, los canales de la salida serán 2 obteniendo una dimensión de salida de 5x5x2.

Para finalizar, es necesario comentar que una práctica muy común es aplicar una función de activación a los resultados obtenidos en la convolución para aplicar no-linealidades a los datos, permitiendo que la red pueda abordar problemas más complejos. La más común es conocida como ReLU cuya función a trozos se encuentra en la ecuación 2.3.

$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Para mayor información sobre convoluciones aplicadas a DL, con animaciones que facilitan la comprensión, revisar [8].

2.3.2. Capas de tipo *pool*

Este tipo de capas son también ampliamente utilizadas en modelos CNN y uno de los principales objetivos consiste en reducir dimensionalidad de los *features* reduciendo el coste computacional de la red sin llegar a perder información en el proceso.

Podemos encontrar mayormente dos tipos, *max-pooling* y *average pooling* (ambas presentes en SqueezeNet V1.1), según el tipo de extracción de datos que llevan a cabo. Estas capas tienen un procedimiento similar a las convoluciones, en el sentido en el que consisten es un filtro de un tamaño determinado que se desliza a través de los datos de entrada realizando la operación correspondiente. Se puede configurar, además, tanto el *stride* como el *padding*. Sin embargo, no afectan para nada a la dimensionalidad de canales del *feature map* de entrada.

Por un lado, la capa maxpool realiza la operación de extraer el dato de entrada cuyo valor sea el mayor de todos los que se encuentran dentro del tamaño del *kernel*.

Por otro lado, la capa avgpool calculará el promedio de los *features* de los datos de entrada.

2.3.3. Definiciones básicas de OpenCL en FPGA

El estándar de OpenCL, de forma general, tiene como objetivo establecer una comunicación entre todos los recursos en una plataforma heterogénea. De esta forma, se realiza una diferenciación entre un dispositivo *Host* que se comunicará con los diferentes dispositivos, realizando una programación de los *kernels* sobre los mismos. Este hecho implica que la programación de los programas de OpenCL se haga en tiempo de ejecución. Podemos encontrar un diagrama conceptual en la figura 2.7

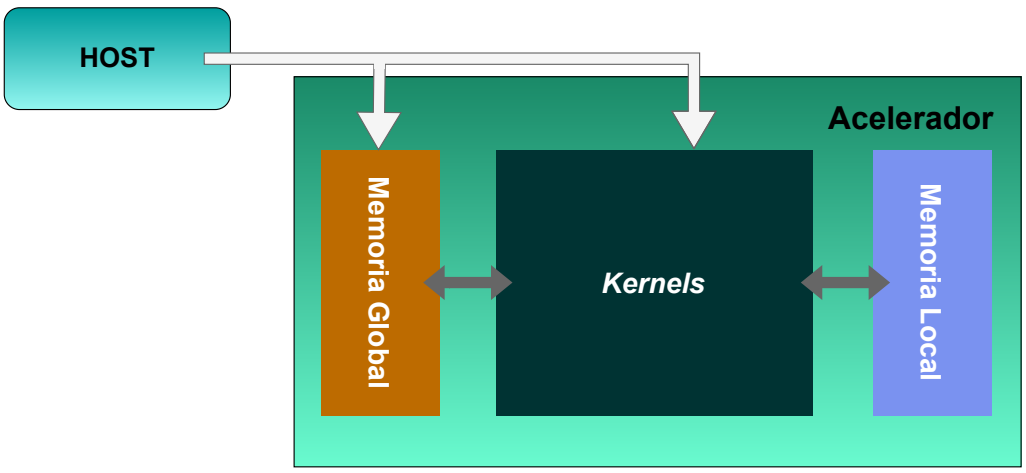

Figura 2.7: Diagrama de bloques conceptual con los componentes básicos en una solución basada en el estándar de OpenCL.

El código del *host* controla la ejecución de los *kernel* dentro de una plataforma mediante colas de comandos, *command queue*. Del mismo modo, dirige los datos que se procesarán por el acelerador *hardware*, en el caso de tecnologías con FPGA, y gestiona los diferentes dispositivos de memoria utilizados.

El compilador HLS de Intel® FPGA SDK para OpenCL sintetizará el código del estándar para

generar el *bitstream* de programación de la FPGA, mediante un archivo ".aocx", implementando el acelerador *hardware* diseñado (2.8). Adicionalmente, en el proceso se genera también un proyecto de Quartus con información de la implementación, así como un archivo de Platform Designer y diferentes informes sobre los resultados del compilador.

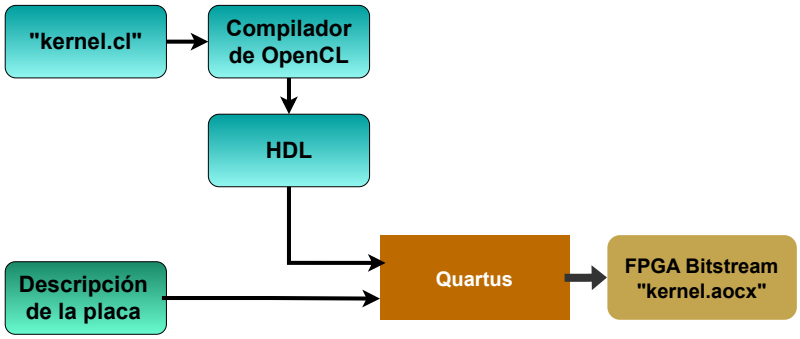

Figura 2.8: Diagrama de bloques de la sintetización del acelerado *hardware*.

2.3.4. Comparativa de las aproximaciones NDRange y *Single Task* en FPGA

Si revisamos la documentación proporcionada por el fabricante sobre el Intel® FPGA SDK para OpenCL™, tanto el documento de *best practices* accesible a partir de [9] como los recursos de *online training* de Intel [10], podemos comprobar que es posible abordar las implementaciones de las soluciones mediante inferencia de paralelismo en FPGA según las características del problema.

La primera aproximación, conocida como NDRange, consiste en dividir la dimensión espacial del problema en *workitems* asemejándose al estilo de paralelización SIMD.

Intel realiza esta aproximación mediante paralelización a nivel de *pipelines* en la que la intención es enviar un dato de entrada para cada hilo nuevo en cada ciclo de reloj.
Para poder conseguir esto, es necesario cumplir las siguientes suposiciones:

- Existe en el kernel paralelismo a nivel de datos.
- Los datos pueden ser divididos en grupo de *work-items*.
- Dependencia mínima entre *work-items* debido a que, en OpenCL, el orden de ejecución de los *threads* no está definido.

En la siguiente figura (2.9), podemos encontrar un ejemplo sencillo de la paralelización de *pipeline* generada por un código sencillo de OpenCL que realiza la suma de dos vectores.

Por motivos de visualización, la ejecución de los *work-items* está serializada, pero en realidad no está definido. Se puede apreciar como en cada ciclo de reloj se intenta lanzar la ejecución de un nuevo *work-item* de forma que cada etapa del *pipeline* esté ocupado utilizando todos los recursos.

La segunda aproximación consiste en gestionar el problema únicamente con un único *workitem*. En el estándar OpenCL, este tipo de *kernels* se denominan tareas, por lo que reciben el nombre de *Single Task*.

Intel recomienda, siempre que sea posible, implementar las soluciones como *Single Task*, pues el compilador se encargará de acelerar el *kernel* para un mejor rendimiento.

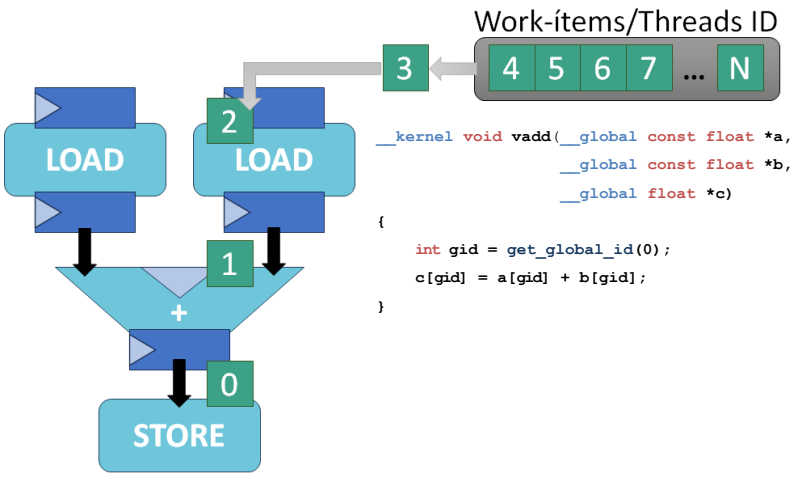

Figura 2.9: Ejemplo de la implementación del paralelismo a nivel de *pipeline* en un *kernel* de tipo NDRange. Imagen creada usando la información de [10].

Del mismo modo, Intel realizará una paralelización a nivel de *pipeline*. La diferencia reside en que en NDRange, el *loop* se despliega en los diferentes *work-items*, pero en *Single Task*, se crea un *pipelining* de las diferentes iteraciones, por ello también se le denomina *loop parallelization*. De esta forma, se consigue:

- Ejecutar eficientemente múltiples iteraciones de un bucle
- Las dependencias entre datos son gestionadas por el compilador
- La transferencia de datos entre iteraciones de un bucle son sencillas y sin necesidad de consumir demasiados recursos de la FPGA gracias a realimentaciones en el *pipeline*

En la figura 2.10 tenemos un ejemplo de paralelización de bucle. Se observa como, gracias a que los índices del bucle están definidos, las dependencias entre los datos de diferentes iteraciones es posible. Además, este estilo de programación es más adaptable a la mayoría de algoritmos, ya que sigue un estilo secuencial facilitando su programación.

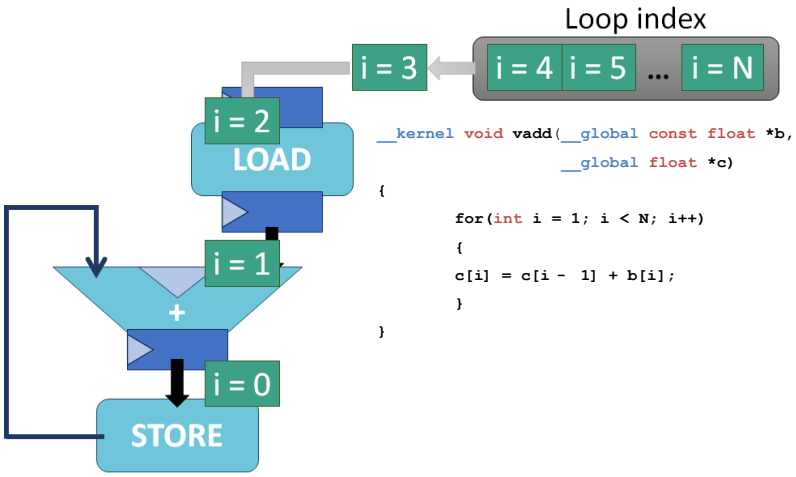

Figura 2.10: Ejemplo de la implementación del paralelismo de *loop* en un *kernel* de tipo *Single Task*. Imagen creada usando la información de [10].

Cuando la dependencia dentro del bucle es compleja, ha de resolverse antes de que la siguiente iteración pueda empezar, lo que añade latencia, ya que el *pipeline* debe esperar hasta que finalice limitando la capacidad de concurrencia del acelerador hardware. Esto se conoce como *loop carried dependency* y afecta directamente al intervalo de iniciación, II, que se refiere al intervalo necesario para que sea posible lanzar una nueva iteración.

La siguiente fórmula que se encuentra en la documentación del fabricante nos permitirá definir el nivel máximo de concurrencia alcanzable por el *kernel* a través de los datos de los informes del compilador HLS:

$$Concurrency_{\ max} = \frac{Latency_{\ block} \cdot Interleave\ Iterations_{\ max}}{II} \tag{2.4}$$

Siendo $Latency_{\ block}$ la latencia del bucle interno y $Interleave\ Iterations_{\ max}$ el número de iteraciones máxima que se pueden ejecutar de forma intercalada.

Cuando se trata de *kernels* en formato de *pipelines*, el número máximo de iteraciones intercaladas es de 1. De esta forma, podemos corroborar, simplificando la ecuación 2.4, que para que la concurrencia del bloque sea la máxima, el II debe de ser de 1 optimizando completamente el *pipeline*.

Capítulo 3

Metodología

A continuación, se procede a describir la metodología que se ha seguido para la realización del presente trabajo de final de máster. Aquí se describirán las características y el procedimiento de instalación tanto del entorno de trabajo como del sistema de validación.

Posteriormente, se establecerá el algoritmo de diseño seguido para la creación de los *kernels* de OpenCL para finalmente exponer el desarrollo seguido en la implementación de la CNN, SqueezeNet V1.1.

3.1. Creación del entorno de trabajo

El entorno de trabajo está conformado por el kit de desarrollo de bajo coste DE10-Nano y el archivo de configuración del acelerador hardware generado por el compilador de Intel® FPGA SDK para OpenCL™.

3.1.1. DE10-Nano

La placa de desarrollo DE10-nano presenta una plataforma con el SoC FPGA de Intel Cyclone V. Integra un procesador ARM Cortex-A9 de doble core (HPS) equipado con una memoria DDR3. Posee un conector RJ45 para conectividad Ethernet, conectores USB, un puerto de tarjetas MicroSD. Podemos encontrar el diagrama de bloques del sistema en la figura 3.1.

En cuanto a los recursos de la FPGA, integra:

- 41910 módulos lógicos adaptativos (ALMs)
- 112 bloques de procesamiento digital de señales (bloques DSP)
- 5570 kilobits de memoria

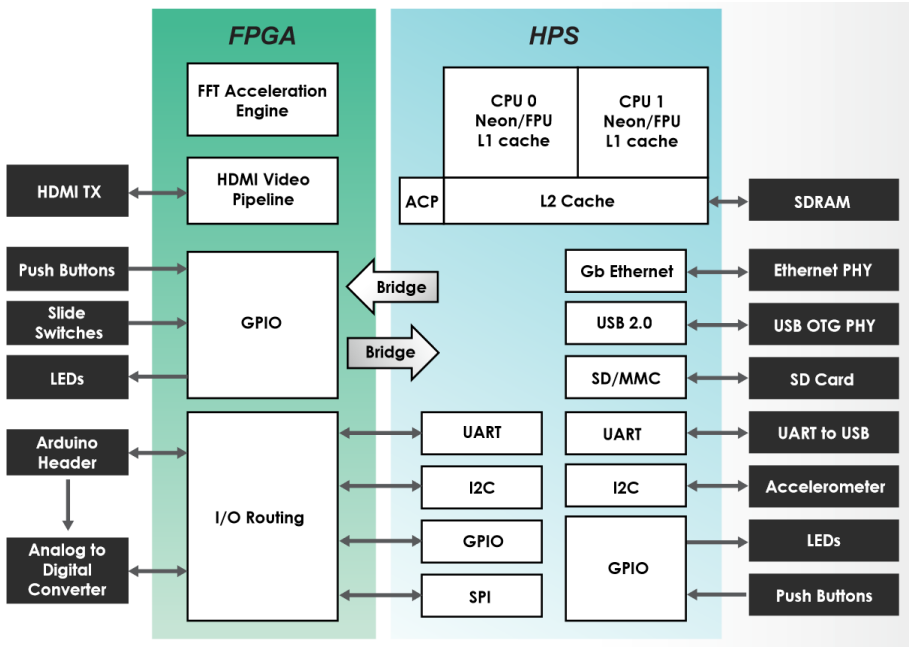

Figura 3.1: Diagrama de bloques del sistema del kit de desarrollo DE10-nano. Imagen proporcionada por el fabricante en [11].

El sistema operativo del HPS es la distribución de Linux, Ubuntu 18.04. La imagen del sistema posee una versión del kernel customizada, 3.18.0-gc11c2db7, y ha sido facilitada por el tutor de este proyecto.

Tiene instalado Python 3.8.0 con la plataforma de computación interactiva de Jupyter Notebook para ejecutar de forma interactiva los diferentes aceleradores *hardware*. Por ello también se encuentran instalados los siguientes paquetes:

- La librería de soporte de matrices multidimensionales Numpy con la versión 1.13.3-2ub.
- El API que implementa las funcionalidades del estandar OpenCL en python, PyOpenCL (versión 2017.2.2-1).
- La librería de aprendizaje automático Pytorch con la versión 1.13.0a0.

3.1.2. Instalación del software

Para poder utilizar la herramienta de Intel® FPGA SDK para OpenCL™ es necesario realizar la instalación de Quartus previamente. El compilador que genera el archivo de configuración del acelerador hardware se instalará en una máquina con el sistema operativo Windows. En el *FPGA Software Download Center* de Intel [12], podemos encontrar los archivos de instalación de Quartus. Según [11], la versión necesaria es la versión estándar 18.1 con licencia necesaria. Una vez encontrado en el centro de descargas, para realizar una instalación con los requerimientos mínimos para nuestro entorno de trabajo, debemos descargar los siguientes archivos:

- Intel® Quartus® Prime Standard Edition Software (sin soporte a los dispositivos) disponible en la pestaña de "*Individual Files*".

- ModelSim-Intel® FPGA Edition (herramienta de simulación HDL), en la misma pestaña.
- Intel® Cyclone® V Device Support.
- Intel® FPGA SDK for OpenCL™ disponible en la pestaña de "*Additional Software*".

Una vez descargado todos los archivos, procederemos con la instalación por defecto siguiendo al *wizard* de Windows. Mencionar que la instalación del soporte de la Cyclone V se realizará conjuntamente con Quartus. También encontramos disponible una actualización del *Software* que se recomienda realizar cuyos archivos de instalación se hallan en la pestaña de "*Updates*" Posteriormente, se debe crear la variable del entorno INTELFPGAOCLSDKROOT con la dirección del directorio "intelFPGA\18.1\hld" para después añadir en la variable PATH las siguientes rutas:

- %INTELFPGAOCLSDKROOT %\bin.
- %INTELFPGAOCLSDKROOT %\host\windows64\bin.

A continuación, para que el HLS pueda generar resultados para la DE10-Nano, necesitamos instalar el paquete de soporte de la placa,BSP, disponible en [11]. Debemos de descomprimir este archivo en el directorio" %INTELFPGAOCLSDKROOT %\board\terasic\de10_nano" y asignar esta dirección a una variable de entorno llamada AOCL_BOARD_PACKAGE_ROOT. Para verificar la instalación del SDK para OpenCL™ podemos comprobar la versión en un PowerShell de Windows mediante el comando "aocl version" (figura 3.2). También podemos comprobar que la placa del BSP es visible para el HLS con el comando "aoc −list −boards" (figura 3.3).

```
PS C:\Users\jose_> aocl version
aocl 18.1.1.646 (Intel(R) FPGA SDK for OpenCL(TM), Version 18.1.1
Build 646 Standard Edition, Copyright (C) 2019 Intel Corporation)
PS C:\Users\jose_>
```

Figura 3.2: Comprobación de la correcta instalación Intel® FPGA SDK para OpenCL™ mediante la comprobación de la versión en el PowerSell de Windows.

```
PS C:\Users\jose_> aoc -list-boards
Board list:
  de10_nano_sharedonly
     Board Package: C:\intelFPGA\18.1\hld\board\terasic\de10_nano
```

Figura 3.3: Comprobación de la correcta instalación del BSP de la placa de desarrollo DE10-nano mediante la comprobación del comando de consulta de placas instaladas del SDK para OpenCL con el PowerSell de Windows.

La compilación se realizará usando el siguiente comando:

```
aoc <kernel_filename>.cl -o <kernel_filename>.aocx
-board=de10_nano_sharedonly -v -report
```

3.2. Creación del entorno de verificación

Una vez realizada la instalación del apartado anterior, ya somos capaces de sintetizar nuestras soluciones de aceleración *hardware*. Sin embargo, el tiempo de compilación requerido por el HLS para generar una solución es considerable, requiriendo alrededor de unas 3 horas dependiendo de la complejidad del diseño.

Por este motivo, se plantea un entorno de validación basado en emulación y simulación para reducir considerablemente el tiempo invertido en el proceso de diseño de los *kernels* de OpenCL.

De forma general, se validará el código que ejecutará el dispositivo *host* mediante el uso de PyOpenCL. Por lo tanto, y al igual que en el entorno de trabajo, se realizará la ejecución en Python en conjunto con Jupyter Notebook de forma dinámica y se comparará con el modelo de referencia que implementa la librería Pytorch.

Posteriormente, se validará la funcionalidad del *kernel* de OpenCL mediante el uso del modo emulación, cuya compilación requiere no mucho más que un par de minutos.

Finalmente, se recurrirá a una validación mediante simulación en el que se validará el diseño en una placa con características similares al kit de desarrollo DE10 nano.
Por facilidad de uso e instalación, se recomienda un OS basado en Linux, por lo que se utilizará una máquina virtual.

3.2.1. Características de la máquina virtual

Mediante el uso de VirtualBox, vamos a instalar una máquina virtual en un *host* de Windows 11. Gracias al manual de usuario DE10-Nano OpenCL descargable desde [11], observamos que el fabricante recomienda para trabajar la distribución de Linux CentOS 7 como sistema operativo. La versión del Kernel instalada es la versión 3.10.0-1160.el7.x86_64

3.2.2. Instalación del *software*

Una vez tenemos la máquina funcionando correctamente, procedemos a instalar los paquetes necesarios para establecer nuestro entorno de validación.

Para facilitar la instalación y gestión de los diferentes paquetes requeridos, mediante la creación de entornos virtuales, se va a utilizar la herramienta de Anaconda. Podemos descargar el archivo de instalación de directamente desde su página web oficial.

Se recomienda crear el entorno virtual de Anaconda con Python 3.8 y activarlo para proceder correctamente con la instalación de los paquetes. Posteriormente, procedemos a instalar pyOpenCL mediante el canal de paquetes Conda-forge de la siguiente forma:

```
conda install -c conda-forge pyopencl
```

Tras terminar la instalación, en “$CONDA_PREFIX/etc/OpenCL/vendors/” tendremos el directorio donde necesitamos situar los drivers del dispositivo compatible con OpenCL o ICD. Para obtener los drivers del CPU de Intel podemos instalar el compilador de clásico mediante

"conda install -c intel icc_rt". Con este paquete se nos instalarán las dependencias necesarias para que OpenCL pueda reconocer las plataformas y dispositivos compatibles.

Es posible que tras la instalación obtengamos un error donde no se encuentra la librería matemática de Intel ("libvml.so error"). Para solucionarlo nos tenemos que instalar la librería *Intel Math Kernel*.

Posteriormente, y si se obtiene "pyopencl._cl.LogicError PLATFORM_NOT_FOUND_KHR", deberemos revisar el directorio "/vendors" de nuestro entorno virtual y comprobar que existe el archivo ".icd". Estos archivos consisten en documentos de texto donde, en una única línea, contienen la dirección de la librería con la implementación del *driver* del dispositivo. De esta forma, mediante un ICD *loader*, nuestra implementación accederá a los *drivers* sin tener que realizar el enlace manualmente.

El driver de la CPU cuya dirección debe aparecer en el archivo ICD se corresponde con el archivo "libintelocl.so". Podremos obtener su localización mediante el siguiente comando:

```
locate libintelocl.so
```

Si no hay ningún documento ICD con la información del driver requerido, habrá que crearlo manualmente. Una rápida comprobación de la instalación se podría realizar, utilizando el intérprete de Python en el terminal de CentOS, ejecutando "pyopencl.get_platforms()" tras importar la librería. Debería aparecer como mínimo la plataforma de la CPU. Podemos encontrar un ejemplo en la figura 3.4.

```
(base) [joerock@localhost ~]$ conda activate TFM
(TFM) [joerock@localhost ~]$ python
Python 3.8.16 (default, Mar  2 2023, 03:21:46)
[GCC 11.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pyopencl
>>> pyopencl.get_platforms()
[<pyopencl.Platform 'Intel(R) OpenCL' at 0xcd7668>, <pyopencl.Platform 'Intel(R) FPGA Emu
lation Platform for OpenCL(TM)' at 0xce1570>, <pyopencl.Platform 'Intel(R) FPGA SDK for O
penCL(TM)' at 0x7f35b0054060>]
>>>
```

Figura 3.4: Comprobación de la correcta instalación de PyOpenCL consultando las plataformas visibles.

Finalmente, podemos proceder a la instalación de los demás paquetes necesarios, así como el IDE Jupyter Lab. Se puede realizar todo este procedimiento mediante los siguientes comandos:

```
conda install -c conda-forge jupyterlab
conda install -c anaconda pillow
conda install -c conda-forge matplotlib
conda install -c pytorch torchvision
```

La instalación de del paquete torchvision también instalará la librería de aprendizaje automático, Pytorch.

En cuanto a la emulación y simulación, es necesario instalar nuevamente Quartus e Intel® FPGA SDK for OpenCL™. Sin embargo, estas funcionalidades se encuentran en versiones superiores de las ediciones profesionales y no presentan soporte para los dispositivos Cyclone V por lo que para la simulación, se utilizará una placa con características similares pero de una gama superior.

Además, en cuanto al *software* de simulación HDL, se utilizará QuestaSim por temas de compatibilidad con la simulación. Esta herramienta nos permitirá obtenerlas formas de onda de la simulación que podrán servir de utilidad en caso de necesitar una depuración más extensa.

Nuevamente, en [12] encontraremos los documentos necesarios para completar la instalación en nuestra máquina virtual. La versión seleccionada para la instalación ha sido la 21.1 cuya instalación mínima requiere la descarga de los siguientes documentos (requiere licencia):

- Intel® Quartus® Prime (includes Nios® II EDS).
- Questa*-Intel® FPGA Edition (incluye Starter Edition).
- Intel® Arria® 10 Device Support.
- Intel® FPGA SDK for OpenCL™ Pro Edition.

Tras la instalación de los archivos con las direcciones por defecto, es necesario crear las diferentes variables de entorno:

- INTELFPGAOCLSDKROOT con valor "/home/joerock/intelFPGA_pro/21.1/hld".
- QSYS_ROOTDIR con valor "/home/joerock/intelFPGA_pro/21.1/qsys/bin".
- QUARTUS_ROOTDIR con valor "/home/joerock/intelFPGA_pro/21.1/quartus".
- QUARTUS_ROOTDIR_OVERRIDE con valor el mismo valor que QUARTUS_ROOTDIR. Esta variable será necesaria para la instalación de los BSPs y para realizar las emulaciones y simulaciones correspondientes.

También se debe añadir los subdirectorios ".../bin" dentro del directorio INTELFPGAOCLSDKROOT a la variable del sistema PATH. Se recomienda crear un *script* de configuración para establecer las variables de entorno automáticamente.

3.2.3. Configuración de la emulación

Para que OpenCL sea capaz de encontrar la plataforma de emulación, es necesario realizar la siguiente configuración:

- Copiar "Intel_FPGA_SSG_Emulator.icd" a "etc\OpenCL\vendors". Este archivo se encuentra en "$INTELFPGAOCLSDKROOT/icd"
- Añadir a la variable de entorno PATH el directorio "$INTELFPGAOCLSDKROOT/llvm/aocl-bin" para poder realizar la compilación en modo emulación correctamente.
- Opcionalmente, establecer la variable de entorno CL_CONFIG_CPU_EMULATE_DEVICES con el número de dispositivos que se quieren emular.

Tras completar la configuración, es posible compilar los *kernels* de OpenCL en modo emulación desde el Jupyter Notebook gracias al comando mágico:

```
%%bash
aoc -march=emulator <kernel_filename>.cl -o <kernel_filename>.aocx
```

3.2.4. Configuración de la simulación

Para la simulación, será necesario instalar el *bsp* de la FPGA Arria 10GX. Para poder obtener estos archivos deberemos acceder a la pestaña "*Linux BSP*" dentro de la página de instalación de "Intel® FPGA SDK para OpenCL™ Pro Edition Software Version 19.4" accesible a través de [12]. Tras su descarga, se debe extraer los archivos en "$INTELFPGAOCLSDKROOT/board/a10_ref" y declarar la variable de entorno AOCL_BOARD_PACKAGE_ROOT.

A continuación, se deberá ejecutar el comando aocl install para completar la instalación de la placa. Al igual que en la instalación del entorno de trabajo, podemos comprobar que la placa se ha instalado correctamente mediante el comando "aoc −list −boards" (figura 3.5).

```
(base) [joerock@localhost Documents]$ aoc -list-boards
Board list:
  a10gx (default)
     Board Package: /home/joerock/intelFPGA_pro/21.1/hld/board/a10_ref
```

Figura 3.5: Comprobación de la correcta instalación del BSP de la placa de soporte Arria 10GX mediante la comprobación del comando de consulta de placas instaladas del SDK para OpenCL en el terminal de CentOS.

Finalmente, para que la plataforma de simulación sea visible para OpenCL debe de ajustarse la variable de entorno CL_CONTEXT_MPSIM_DEVICE_INTELFPGA a 1. Para realizar la compilación desde la plataforma de Jupyter Notebook, en conjunto con el comando mágico "$ $ bash", se utilizará el comando "aoc −march=simulator −v −ghdl <kernel_filename>.cl −o < kernel_file name>.aocx −board=a10gx". Con este comando se generan, además, las formas de onda mediante el simulador HDL QuestaSim.

3.3. Algoritmo de diseño

A continuación, tras finalizar la instalación y configuración de los entornos de trabajo y simulación, se procede a exponer el algoritmo de diseño de los códigos de OpenCL que sintetizarán los aceleradores *hardware* que componen el modelo de CNN, SqueezeNet V1.1.

De forma general, el código OpenCL que diseña el algoritmo de aceleración se validará siguiendo el mismo proceso. Todos los resultados se compararán con los resultados del *golden model* implementado mediante PyTorch. De esta forma, se comprobarán que las operaciones realizadas en el *kernel* son las adecuadas.

Una vez tenemos confeccionado el código de OpenCL, se puede realizar una ejecución en la plataforma de la CPU. De esta forma podremos encontrar posibles errores de sintaxis de forma rápida y sencilla. A continuación, se realizará una compilación del código en modo emulación. Tras obtener el *bitstream* procederemos a validar tanto el código del *Host*, implementado mediante PyOpenCL en el entorno de programación interactiva de Jupyter Notebook, como el código del *kernel*.

Esta primera etapa nos confirmará si el acelerador es funcional a nivel de operaciones, pero será necesario comprobar si es implementable en una placa FPGA. En nuestro caso, el dispositivo

seleccionado es una Arria 10GX con características similares a la Cyclone V pero de una gama superior.

De esta forma, en la segunda etapa de la validación, se comprobará si el diseño es sintetizable en una FPGA. Por ello, se realizará la compilación en modo simulación con la placa objetivo seleccionada. Como el código del *Host* ha sido validado en la etapa anterior, únicamente será necesario modificar el archivo ".aocx" durante la programación del kernel. Nuevamente, se realizará la comparación de los resultados de la simulación con los resultados de PyTorch para proceder con la última etapa de validación.

Es necesario comentar que el proceso de simulación puede conllevar unos tiempos de ejecución elevados, pues se realiza la simulación HDL con QuestaSim. Por ello, se debe ajustar los bancos de prueba para que se pueda reducir los tiempos de simulación sin poder llegar a afectar a la validación funcional de las operaciones llevadas a cabo en el *kernel*.

Por último, a pesar de que las anteriores etapas sean satisfactorias, se deberá comprobar si el código de OpenCL es sintetizable para el dispositivo del entorno de trabajo. Tras finalizar la compilación, se generarán varios informes, siendo dos los más importantes. Un primer informe en formato HTML que contendrá un análisis de los bucles del*kernel* con información sobre el II y la latencia del sistema. El segundo informe consistirá en un documento de texto con información de los recursos de la FPGA consumidos por el acelerador, además de la frecuencia máxima de trabajo del sistema.

Tras comparar los resultados de la ejecución sobre la DE10-nano, se revisarán los informes para comprobar si la implementación es óptima o si es posible aplicar las diferentes técnicas de optimización documentadas por el fabricante en la guía de "*Best Practices*" disponible a través de [9]. Es necesario tener en cuenta que normalmente las técnicas de optimización de latencia conllevan un aumento en el uso de recursos, por lo que es necesario llegar siempre a un compromiso entre ambos.

En la figura 3.6 podemos encontrar el diagrama de flujos que sintetiza el algoritmo de diseño que se ha utilizado en el presente trabajo.

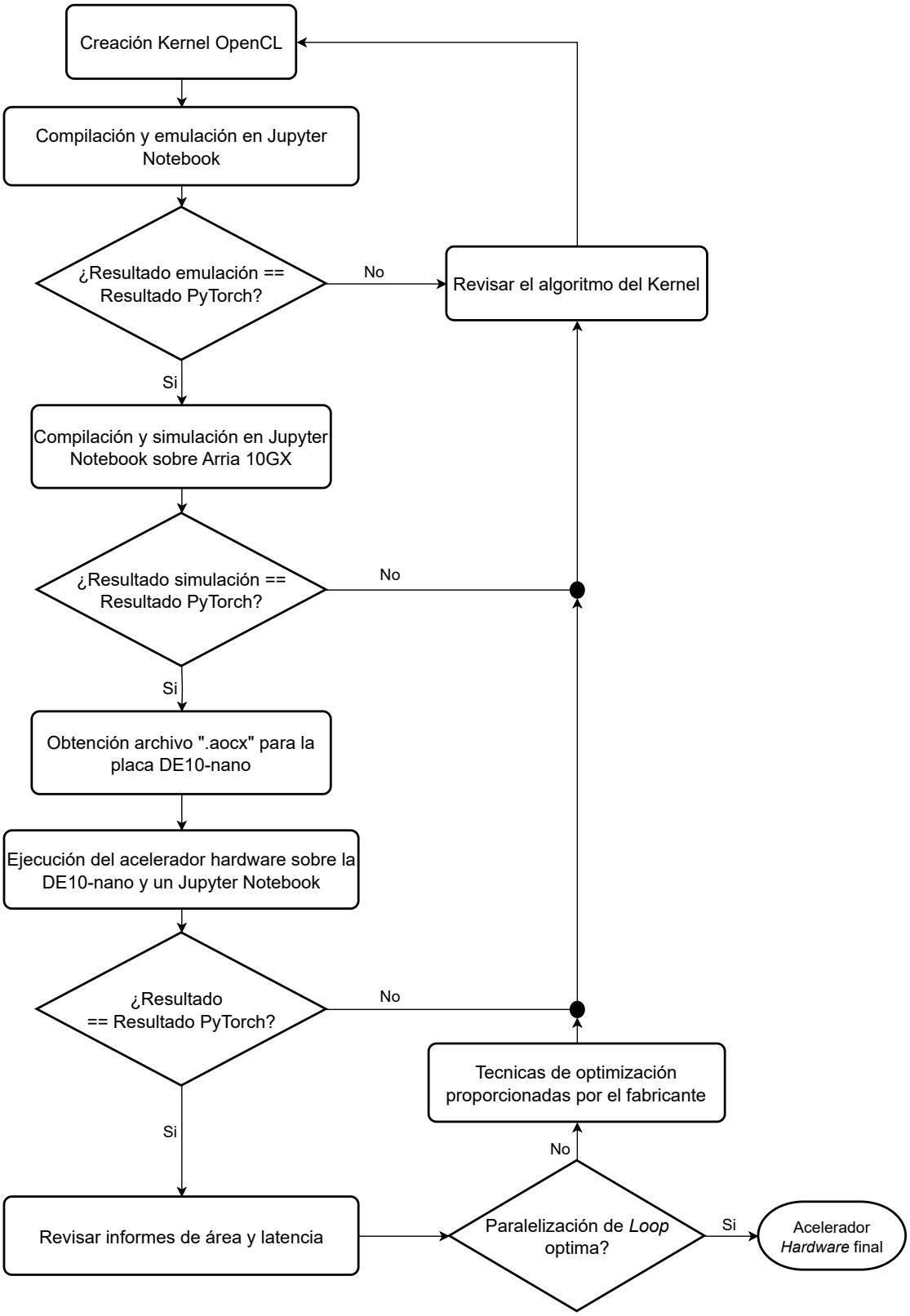

Figura 3.6: Algoritmo de diseño seguido para la validación del acelerador *hardware* diseñado.

3.4. Procedimiento

En este apartado, tras establecer como será el flujo de trabajo, se procede a exponer el procedimiento que se ha seguido en el desarrollo de la implementación de SqueezeNet V1.1 en la placa de desarrollo de bajo coste, DE10-nano.

Basándose en el marco teórico, se realizará una implementación de tipo *Single Task* de cada una de las capas presentes dentro del modelo de SqueezeNet V1.1. Se decide comenzar con el desarrollo de la capa de convolución 1x1, ya que es lo suficientemente compleja como para poder tener una primera toma de contacto con todos los entornos del algoritmo de diseño. A continuación se comentarán el resto de capas, desde la convolución 3x3, la más compleja, hasta la capa de tipo maxpool, la más asequible.

Por último, se juntarán las piezas del modelo para realizar una validación de todo el sistema a diferentes niveles. Puesto que es posible que se tenga que ajustar nuevamente cada capa para poder realizar la síntesis de todos los *kernels* dentro de la placa de desarrollo.

3.4.1. Convolucion 1x1

Este *Kernel*, debido a su relativa complejidad y similitud con el resto de capas, ha sido el primero en ser desarrollado. Por ello, podemos encontrar diferentes versiones que nos han sido útiles para poder conocer las herramientas de diseño y depuración facilitadas por Intel, observar las capacidades de la placa de desarrollo DE10-Nano y asimilar las características del compilador, HLS así como las directrices que lo controlan para poder perfilar el diseño. A continuación, se expondrán las versiones más relevantes.

Cabe remarcar que las implementaciones de las diferentes versiones se lleva a cabo utilizando las indicaciones que podemos encontrar en la guía de “Best Practices” que se encuentra en [9].

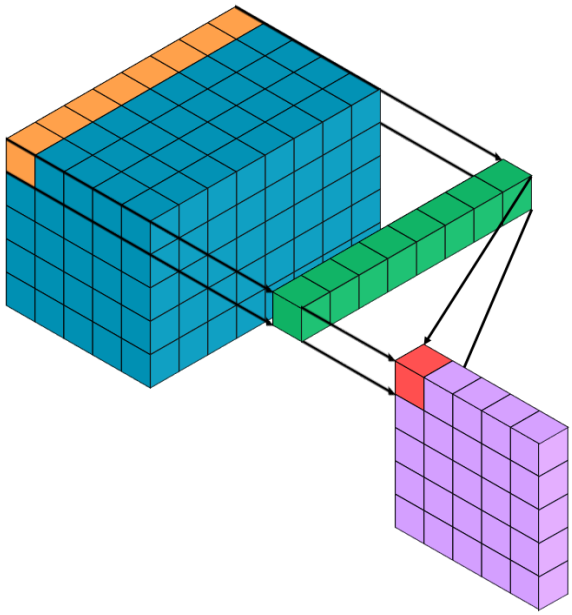

Figura 3.7: Ejemplo de convolución 1x1 para visualización del procedimiento.

La implementación de la primera versión de la convolución 1x1 se realiza de la forma más directa posible. Se realizan diferentes iteraciones a través de los diferentes filtros de la convolución, posteriormente a través del tamaño del *feature map* de entrada para seleccionar los píxeles de un canal y realizar la suma de los productos a través de los diferentes canales de entrada más el parámetro de *bias*, podemos usar la figura anterior para visualizar el procedimiento. La lectura de las variables de *weights* se realiza con memoria coalescente mediante la estructura de datos float4.

Tal y como observamos en el apartado 2.1, añadir *padding* y *stride* a la convolución 1x1 correspondería en un aumento del coste computacional innecesario y perdida de información respectivamente. Por este motivo, nuestro *kernel* no tendrá en cuenta estás configuraciones.

La comparación con el modelo de convolución de Pytorch es satisfactoria, al igual que la compilación del acelerador. Sin embargo, observamos en los resultados, que a pesar de que se consigue una frecuencia de reloj elevada, el II es de 16 ciclos de reloj debido a la operación de suma de datos float. Esto nos indica que la paralelización del bucle no es completamente óptima. Además, cabe remarcar que en el diseño final, la frecuencia de reloj se verá limitada por el *kernel* más restrictivo, por lo que la frecuencia del reloj será menor, reduciendo el rendimiento de la convolución 1x1.

En cuanto a la segunda versión, el fabricante aconseja no utilizar operaciones sobre los punteros de acceso a memoria, por lo que se decide realizar la indexación de los datos de entrada mediante el estilo $j + W\_in * i + offset$ siendo i y j la fila y la columna del elemento que se quiere indexar respectivamente. Se añade además el offset que seleccionará el canal al que pertenece dicho elemento. Se observa una mejoría en cuanto al tiempo de ejecución del kernel en adición de un ligero aumento de los DSPs utilizados.

Posteriormente, revisando el IP del subsistema generado por el SDK para OpenCL en FPGA de Intel mediante Platform Designer, observamos que la CPU se conecta con la memoria global mediante un Avalon-MM bidireccional de 256 bits, como podemos observar en la figura 3.8. Dado que la version de PyOpenCL instalada en la placa de desarrollo es compatible con la utilización de las estructuras de datos desde float4 hasta float16 se decide realizar una versión del kernel con float8 además de realizar una prueba de la directriz "pragma unroll" por un factor 2.

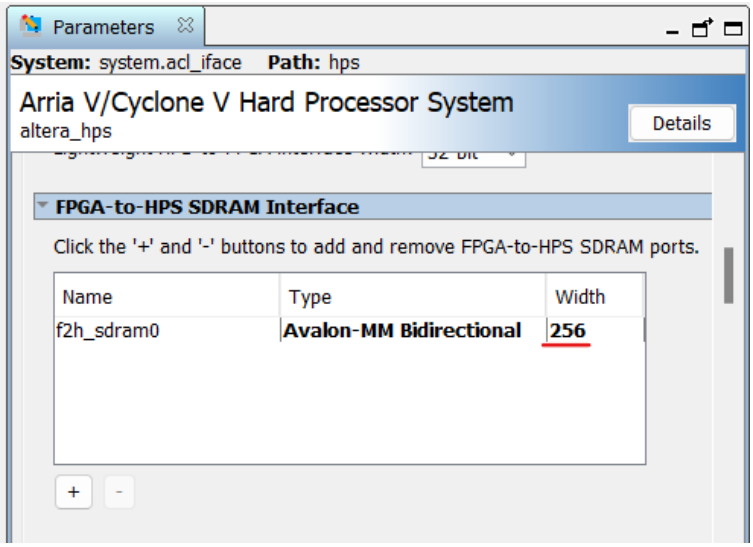

Figura 3.8: Captura de pantalla de la configuración de la conexión de la memoria global entre HPS y el acelerador de la FPGA.

Hasta ahora, todas estas versiones obtienen un índice II de más de 1 ciclo de reloj debido a la operación de suma de dos datos de tipo float del acumulador. Normalmente, en los informes de la DE10-Nano, esta operación suele consumir 16 ciclos de reloj.

Siguiendo el algoritmo de diseño, se revisa la documentación indicada donde se observa una técnica para reducir el II mediante la inferencia de un registro de desplazamiento o *shift register*. La documentación específica que el tamaño del registro de desplazamiento debe de ser aproximadamente del factor II del código sin optimizar. Suponiendo un acumulador con un intervalo de iniciación de 16 ciclos de reloj, el código optimizado se implementaría de la siguiente forma:

```
#define II_CYCLES 16

__kernel void conv2d1x1(
const int N,
__global const float* restrict B,
__global float* restrict C)
{
    float shift_reg[II_CYCLES];
    // Inicializando el registro de desplazamiento
    for(int k = 0; k < II_CYCLES; k++)
    {
        shift_reg[k] = 0;
    }

    for(int i = 0; i < N; i++)
    {
        shift_reg[II_CYCLES - 1] = shift_reg[0] + B[i];

        // Desplazando los valores
        #pragma unroll
        for(int k = 0; k < (II_CYCLES - 1); k++)
        {
            shift_reg[k] = 0;
        }
    }

    // Sumando los elementos del registro de desplazamiento
    float tmp = 0.0;
    for(int k = 0; k < (II_CYCLES - 1); k++)
    {
        tmp += shift_reg[k];
    }

    C[0] = tmp;
}
```

La siguiente versión realizará la implementación de la versión 2 añadiendo la inferencia de un registro de desplazamiento cuyo tamaño es de 16 registros, consiguiendo la reducción del intervalo de inicio por un factor 4. Se observa que esto es debido a que float4 requiere 4 sumadores, limitando de esta forma el II puesto que para obtener un intervalo de inicio de 1 se requiere un registro de desplazamiento de 64 ($16 \cdot 4$) registros agotando los recursos de la placa, siendo imposible realizar la síntesis para este caso.

Debido a esto, la versión 5 realiza una adaptación de la segunda versión sin el uso de la estructura de datos float4.

Finalmente, la última versión adaptará el código de la versión anterior para inferir un registro de desplazamiento, siendo posible una reducción del II a 1 optimizando completamente el *kernel* de OpenCL.

3.4.2. Convolucion 3x3

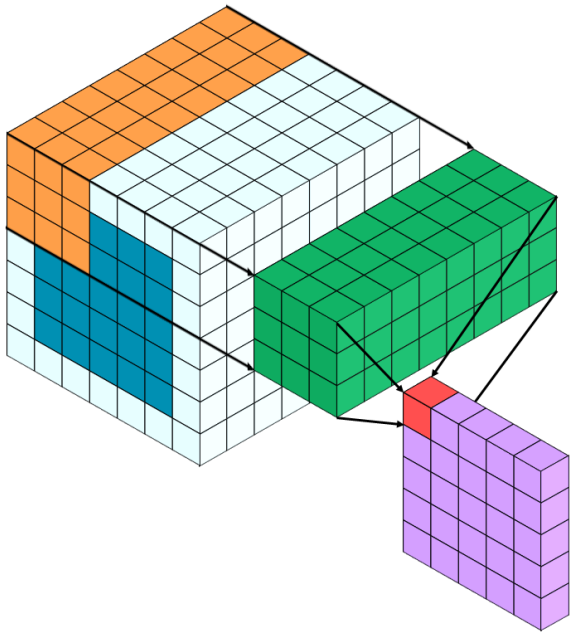

Figura 3.9: Ejemplo de convolución 3x3 con *padding* para visualizar el procedimiento.

Para la convolución 3x3 se decide realizar el mismo procedimiento que el que se ha seguido en la convolución 1x1. Sin embargo, solo se implementan versiones con la optimización mediante un registro de desplazamiento (V2) y la indexación sin realizar operaciones sobre los punteros con registro de desplazamiento (V3).

Se arreglan las iteraciones para poder acceder a los 9 píxeles de cada canal del *feature map* de entrada (figura 3.9), mediante el pragma "unroll" para realizar las 9 operaciones de forma concurrente replicando el hardware, teniendo en cuenta el *stride* y el *padding* para calcular la suma de los productos a través de todos los canales de entrada.

Dado que la convolución 3x3 forma parte de la capa *expand*, para poder realizar la concatenación de las salidas de esta capa, se añade una variable de offset que controlará la posición del *feature map* en la que se guardarán los datos de salida.

Nuevamente, se consigue validar la funcionalidad del acelerador dado que la comparación con PyTorch es satisfactoria. Por otro lado, se observa un intervalo de inicio con unas magnitudes bastante mayores que en la convolución 1x1. Las operaciones de suma de datos float generan un II de 136 ciclos de reloj, por lo que se observa un acelerador poco optimizado.

Se comprueba que, en este caso, es inviable realizar la implementación del registro de despla-

zamiento, puesto que los recursos requeridos para la implementación son demasiados al tratarse de datos de tipo flotante de 32 bits, no se consigue implementar esta capa en la DE10nano.

Sin embargo, es posible implementar el registro de desplazamiento con un menor tamaño y reducir igualmente el intervalo de iniciación. Mediante un ajuste de prueba y error, se consigue obtener un tamaño de registro de desplazamiento mínimo que permite reducir el II a 1. Con esto, se consigue obtener un *kernel* optimizado.

3.4.3. Avgpool

Podemos observar que esta capa presente en SqueezeNet (tabla 2.2) se encuentra al final del modelo de CNN y solo se utiliza una única vez. Por ello, podemos realizar una implementación más específica para poder reducir la complejidad del acelerador. Por ello, el *kernel* consistirá en un bucle que iterará a través de los diferentes canales, calculando el promedio de los 13x13 datos de entrada del *feature map*.

Dado que nuevamente se trata de una suma de datos de tipo flotante, el intervalo de iniciación es de 16 ciclos de reloj. Dado que esta capa solo se utiliza una única vez, se decide no inferir un registro de desplazamiento para optimizar el bucle paralelizado para no consumir demasiados recursos.

3.4.4. Maxpool

En cuanto al último tipo de capa a implementar, se observa en la tabla 2.2 que en SqueezeNet, consistirá siempre en un filtro de dimensiones 3x3 con un *stride* de 2. Por lo tanto, al igual que en la capa avgpool, podemos realizar un diseño más ajustado, reduciendo nuevamente la complejidad del acelerador a diseñar.

Este filtro consistirá, por lo tanto, en un bucle que iterará a través de los canales del *feature map* de entrada para acceder a los 3x3 datos de entrada. Dado que la capa se configura con un *stride* de 2, el tamaño de los *fetures* de salida será menor, reduciendo también el número de iteraciones.

Tras la compilación del código de OpenCL, se obtiene un acelerador optimizado, pues el intervalo de iniciación es de 1 ciclo de reloj. Esto se debe a que la operación realizada consiste en una comparación que es menos complejo que la operación suma.

3.4.5. Modelo completo

Finalmente, para el modelo completo, se observa que no es posible introducir todos los *kernels* optimizados, dado que los recursos de la placa de desarrollo DE10-nano son limitadas. Por ello, se realizan pruebas con tres versiones diferentes. Una primera versión, con los *kernels* sin optimizar, las otras dos tendrán las versiones optimizadas con un II de 1 de las dos capas de convolución 1x1 y 3x3 respectivamente. Así, se podrá escoger la implementación con un mejor rendimiento.

Es necesario comentar que se decide priorizar las capas cuyo II es de 1, dado que los tiempos de ejecución pueden variar debido a diferentes frecuencias de reloj de las implementaciones individuales de cada capa. Sin embargo, cuando juntamos todas las capas en un mismo acelerador, la

frecuencia de reloj se ve limitada, siendo el intervalo de inicio la mejor opción para optimizar los *kernels*.

Dado que la convolución 1x1 es la capa que más se utiliza, se observa que su rendimiento es bastante crítico en cuanto al tiempo de clasificación.

Para evaluar su impacto, se llevarán a cabo diferentes comparaciones a diferentes niveles. En la arquitectura SqueezeNet, encontramos diferentes divisiones en las que se puede seccionar el modelo. La unidad básica de esta CNN son los módulos Fire. Posteriormente, podemos encontrar 3 bloques diferentes correspondientes a agrupaciones de diferentes módulos Fire. En los dos primeros bloques, se hallan dos módulos Fire seguidos de una capa de tipo maxpool. En el último bloque, solo encontraremos una agrupación de 4 módulos Fire (Fires del 5 al 8), convirtiéndose en el bloque con mayor coste computacional.

Por último, se realizará la validación, a nivel global, de toda la arquitectura del modelo SqueezeNet comprobando el rendimiento completo de la síntesis del los aceleradores con estilo *Single Task* que implementa la *convolutional neural network* seleccionada.

3.4.6. Banco de pruebas

Tal y como hemos observado en nuestro algoritmo de diseño (apartado 3.3), es necesario realizar validaciones funcionales de nuestros diseños a diferentes niveles, desde la emulación, pasando por la simulación, hasta la validación en la placa de desarrollo.

Gracias a nuestro entorno de trabajo, somos capaces de realizar validaciones de nuestros diseños en OpenCL en diferentes plataformas, puesto que podemos llevar a cabo, tanto la ejecución directa del código de OpenCL como las emulaciones y simulaciones en la CPU y la GPU. Además, como es de esperar, podemos ejecutarlo también en la FPGA tras realizar la compilación HLS.

Por ello, se han desarrollado diferentes bancos de prueba. Por un lado, se ha desarrollado el código que se utilizará para la validación mediante emulación y simulación dentro de un Jupyter Notebook. Con este tipo de banco de prueba, somos capaces de ejecutar de forma directa el código OpenCL para encontrar pequeños fallos de sintaxis o de algoritmo. Posteriormente, es posible lanzar la compilación de nuestro diseño para emulación y simulación para comprobar que la implementación de nuestro diseño es factible en la FPGA objetivo.

Por otro lado, se adapta el código para poder ejecutar el mismo banco de pruebas sobre la FPGA para poder tener así el diseño final completamente validado.

Los bancos de prueba generados, nos permiten controlar sencillamente los datos de entrada de forma paramétrica y obtener resultados temporales, además de comprobar que el resultado es coherente con el modelo ideal implementado con PyTorch y comparar los resultados con su implementación en formato NDRange. También es posible configurar una adquisición de resultados temporales mediante el promedio de varias ejecuciones. La cantidad de datos usados para el promedio es también paramétrico.

Se han confeccionado diferentes Notebooks para realizar la validación tanto en placa como en emulación y simulación de los siguientes diseños y niveles:

- Convolución 1x1
- Convolución 3x3

- Average pool
- Max pool
- Nivel de Fire
- Nivel de bloque 1 y 2
- Nivel de bloque 3
- Nivel de modelo completo

En los anexos (II) se encuentran algunos códigos de ejemplo de los bancos de prueba desarrollados.

Capítulo 4

Resultados obtenidos y conclusiones

En el siguiente apartado, se mostrarán los resultados obtenidos tras la compilación de los *kernels* de OpenCL a los diferentes niveles expuestos en el capítulo anterior. Siempre que sea posible, se añadirán los resultados de los demás trabajos expuestos en la revisión literaria. Cabe comentar, que por simplicidad, los resultados que se presentan solo tienen en cuenta las versiones que produjeron un resultado significativo.

Dado que [7] proporciona el código de su implementación en estilo NDRange, podremos realizar una comparativa más extensa. Se utilizará la versión 1.3 que se encuentra en su repositorio de GitHub. Por último, se sintetizarán las conclusiones que se han podido extraer del desarrollo del proyecto

4.1. Convolución 1x1

A continuación, se presentan los resultados de tiempo de ejecución e intervalos de inicio obtenidos para las diferentes versiones desarrolladas. Para llevar a cabo estas comparaciones, se configurará la capa del filtro de convolución para ejecutar una entrada de dimensión 13x13x512 con 1000 filtros. Esta convolución se corresponde a la "conv2" de SqueezeNet V1.1 (tabla 2.2).

	Tiempo de ejecución (s)	Intervalo de inicio (II)
V0	3.4381	16
V1	1.7096	16
V2	1.0423	31
V5	2.4772	16
V6	1.2000	1

Tabla 4.1: Tabla con resultados de tiempos de ejecución e intervalo de inicio, II, de diferentes versiones del *kernel* conv1x1.

Como observamos, el hecho de no realizar operaciones sobre los punteros acelera considerablemente el tiempo de ejecución. Además, el único método que consigue optimizar considerablemente el *pipeline* de bucle es la implementación de un registro de desplazamiento, siendo la versión 6 el

kernel de la capa de convolución 1x1 que se ha optimizado mejor.

Descartamos la versión 2, debido a la cantidad de recursos necesarios para implementarlo y su elevado II, ya que a pesar de que sea la versión con menor tiempo de ejecución, no será compatible con la sintetización del *kernel* con todas las capas de SqueezeNet.

En la siguiente tabla encontramos los resultados obtenidos de la sintetización del *hardware* de la convolución 1x1 (V6) con inferencia de un *shift register*. Nuevamente, obtenemos el tiempo de ejecución requerido para la ejecución de una convolución 1x1 con un *feature map* de entrada de 13x13x512 con 1000 filtros.

	Single Task	NDRange [7]	PyTorch
ALUTs	16421	11536	-
Registros	28536	21064	-
Utilización lógica	12413 / 41910 (30 %)	6962 / 41910 (24 %)	-
Bloques DSP	9 / 112 (8 %)	20 / 112 (18 %)	-
Bits de memoria	351936 / 5662720 (6 %)	1092608 / 5662720 (19 %)	-
Bloques RAM	101 / 553 (18 %)	168 / 553 (30 %)	-
Frecuencia máxima (MHz)	116.13	120.81	-
Tiempo de ejecución (s)	1.2000	0.36890	0.1823

Tabla 4.2: Tabla con resultados de compilación y ejecución del *kernel* conv1x1 V6.

4.2. Convolución 3x3

En cuanto a la convolución 3x3, se utilizará una entrada de 13x13x64 con 256 filtros. Corresponde con una configuración de la convolución 3x3 de una capa *expand* de los módulos Fire 7 u 8 (tabla 2.2).

Podemos encontrar los tiempos de ejecución y los intervalos de iniciación de las versiones del *kernel* en la siguiente tabla:

	Tiempo de ejecución (s)	Intervalo de inicio (II)
V1	0.2482	136
V2	0.2174	1
V3	0.2135	1

Tabla 4.3: Tabla con resultados de tiempos de ejecución e intervalo de inicio, II, de diferentes versiones del *kernel* conv3x3.

Nuevamente, comprobamos que la indexación sin operaciones sobre los punteros de referencia mejora el tiempo de ejecución, aunque en este caso es una mejora sutil. Sin embargo, la imple-

mentación de un registro de desplazamiento, optimiza en mayor consideración, convirtiendo a la versión 3 del *kernel* de la convolución 3x3 la más óptima.

A continuación se exponen los resultados obtenidos de la sintetización del *hardware* de la versión de la convolución 3x3 seleccionada para comprobar el rendimiento de una paralelización de bucle optimizada. Para la obtención de los resultados de ejecución, nuevamente, se configura la ejecución para realizar una convolución 3x3 con un *feature map* de entrada de 13x13x64 con 256 filtros.

	Single Task	NDRange [7]	PyTorch
ALUTs	30109	255353	-
Registros	51345	38286	-
Utilización lógica	23431 / 41910 (56 %)	20028 / 41910 (48 %)	-
Bloques DSP	31 / 112 (28 %)	29 / 112 (26 %)	-
Bits de memoria	966560 / 5662720 (17 %)	2831104 / 5662720 (50 %)	-
Bloques RAM	198 / 553 (36 %)	403 / 553 (73 %)	-
Frecuencia máxima (MHz)	107.21	113.23	-
Tiempo de ejecución (s)	0.2135	0.0554	0.0921

Tabla 4.4: Tabla con resultados de compilación y ejecución del *kernel* conv3x3.

4.3. Avgpool

Para la capa avgpool, dado que no hay diferentes versiones del *kernel*, se presenta en la siguiente tabla los resultados obtenidos por la síntesis del código de OpenCL de la capa avgpool. Esta capa corresponde con la última etapa de la arquitectura de la CNN seleccionada y nos dará el resultado de la clasificación. Por ello, la entrada consistirá en un *feature map* de dimensión 13x13x1000.

	Single Task	NDRange [7]	PyTorch
ALUTs	6178	6045	-
Registros	9303	8492	-
Utilización lógica	4736 / 41910 (11 %)	4633 / 41910 (11 %)	-
Bloques DSP	4 / 112 (4 %)	4 / 112 (4 %)	-
Bits de memoria	182798 / 5662720 (3 %)	209162 / 5662720 (4 %)	-
Bloques RAM	59 / 553 (11 %)	49 / 553 (9 %)	-
Frecuencia máxima (MHz)	133.16	117.28	-
Tiempo de ejecución (s)	0.0028	0.0034	0.0057

Tabla 4.5: Tabla con resultados de compilación y ejecución del *kernel* avgpool.

4.4. Maxpool

En esta sección, dado que la capa maxpool tampoco tiene más versiones, se presentan los recursos de la DE10-nano tras su compilación con el HLS de OpenCL. Los resultados presentados en la siguiente tabla corresponden a la ejecución del *kernel* que implementa la capa maxpool con la configuración de la capa maxpool1 de la arquitectura de SqueezeNet V1.1 (tabla 2.2).

	Single Task	NDRange [7]	PyTorch
ALUTs	6274	6318	-
Registros	10757	12317	-
Utilización lógica	5432 / 41910 (13 %)	6032 / 41910 (14 %)	-
Bloques DSP	8 / 112 (7 %)	12 / 112 (11 %)	-
Bits de memoria	183644 / 5662720 (3 %)	234864 / 5662720 (4 %)	-
Bloques RAM	68 / 553 (12 %)	58 / 553 (10 %)	-
Frecuencia máxima (MHz)	122.24	118.51	-
Tiempo de ejecución (s)	0.0269	0.0471	0.0521

Tabla 4.6: Tabla con resultados de compilación y ejecución del *kernel* maxpool.

4.5. Modelo completo

Finalmente, tras realizar las pruebas se confecciona el código de OpenCL con todas las capas diseñadas. Se realizarán varias versiones con diferentes combinaciones de las versiones de los *ker-*

nels de las convoluciones 1x1 y 3x3 debido a que no hay recursos suficientes para implementar ambas capas optimizadas. Las versiones más significativas corresponden con la primera versión, que consistirá en los *kernels* sin ninguna optimización, la versión 6, que contiene la versión 6 de la convolución 1x1 y la versión 3 de la convolución 3x3 sin la optimización del registro de desplazamiento, y la versión 7 con la versión 1 de la convolución 1x1 y la versión 3 de la convolución 3x3.

Se escogen estas versiones, puesto que se corresponden con las versiones más optimizadas sin la implementación del registro de desplazamiento. En la siguiente tabla se exponen los resultados de las diferentes versiones del modelo de SqueezeNet completo, donde se presentan los tiempos de ejecución y frecuencia de reloj del acelerador.

	Tiempo de ejecución (s)	Frecuencia máxima (MHz)
V1	5.9842	105.42
V6	4.2992	106.43
V7	4.4380	104.97

Tabla 4.7: Tabla con resultados de tiempos de ejecución y frecuencia de reloj máxima de las diferentes versiones del acelerador *hardware* del modelo completo.

Se observa como la versión 6 consigue los mejores resultados además de una capa de convolución 1x1 con un II de 1. Esto podría deberse a que la presencia de esta capa en el modelo Squeezenet es elevada. Además, como hemos visto en los resultados, la capa "conv2" (tabla 2.2) es la que más tiempo consume en ejecutarse, pues su carga computacional es elevada.

Cabe comentar que, dado que el HDL realizá una implementación de todos los *kernels* de OpenCL es posible realizar la ejecución paralela de dos capas diferentes si utilizamos otra cola de comandos. Será necesario únicamente una sincronización mediante el comando de PyOpenCL "queue. finish ()". Se ha aplicado esta técnica a todas las pruebas de los *kernels* de tipo *Single Task* y también se hacen pruebas sobre el modelo de tipo NDRange de [7] mejorando ligeramente el tiempo de ejecución.

En la tabla 4.8, se encuentran los recursos consumidos por la placa de desarrollo para poder llevar a cabo la síntesis de los *kernels*. Además, se expondrán los tiempos de ejecución de los diferentes niveles que encontramos en el modelo de SqueezeNet.

	Single Task (V6)	NDRange [7]	Our NDRange	ZynqNet [4]	Edgenet [5]	Jingyuan Zhao et al [6]
ALUTs	46469	44951	-	-	-	
Registros	81572	73337	-	137k (FF)	-	-
Utilización lógica	36226 / 41910 (86 %)	36353 / 41910 (87 %)	-	154k / 218 (70 %) LUT	-	110k
Bloques DSP	52 / 112 (46 %)	65 / 112 (58 %)	-	739 / 900 (82 %)	-	-
Bits de memoria	1469514 / 5662720 (26 %)	4096538 / 5662720 (72 %)	-	-	-	-
Bloques RAM	377 / 553 (68 %)	553 / 553 (100 %)	-	996 / 1090 (91 %)	-	-
Frecuencia máxima (MHz)	106.43	103.07	-	200	100	101.7
Tiempo de ejecución Fire 8 (s)	0.3464	0.0901	0.0977	-	-	-
Tiempo de ejecución Block 3 (s)	1.0206	0.3358	0.3374	-	-	-
Tiempo de ejecución global (s)	4.2992	1.0454	0.9996	-	0.110	0.121

Tabla 4.8: Tabla con resultados de compilación y ejecuciones del código de OpenCL completo que implementa el modelo de SqueezeNet V1.1. Se realiza una comparación también con los trabajos previos.

Podemos encontrar el código de la versión 6 del *kernel* del modelo completo en los anexos (II).

Por último, se presentan los resultados del informe en formato web que genera el compilador de Intel OpenCL. Aquí podremos observar información respecto a la paralelización inferida tanto en la implementación *Single Task* (figura 4.1) como en la implementación NDRange (figura 4.2). A primera vista, se aprecia que hay menos bucles inferidos. También en las figuras 4.3 y 4.4, respectivamente para *Single Task* y NDRange, tendremos la visualización de los *kernels* facilitada por el informe.

Kernel: avgpool2d (squeezenet_ST_V6.cl:180)				Single work-item execution
avgpool2d.B1 (squeezenet_ST_V6.cl:187)	Yes	>=1	n/a	
avgpool2d.B2 (squeezenet_ST_V6.cl:192)	Yes	~16	II	Data dependency
Kernel: conv2d1x1 (squeezenet_ST_V6.cl:123)				Single work-item execution
conv2d1x1.B1 (squeezenet_ST_V6.cl:126)	Yes	>=1	n/a	
conv2d1x1.B2 (squeezenet_ST_V6.cl:134)	Yes	>=1	n/a	
Fully unrolled loop (squeezenet_ST_V6.cl:140)	n/a	n/a	n/a	Auto-unrolled
Fully unrolled loop (squeezenet_ST_V6.cl:160)	n/a	n/a	n/a	Auto-unrolled
conv2d1x1.B3 (squeezenet_ST_V6.cl:145)	Yes	~1	n/a	II is an approximation.
Fully unrolled loop (squeezenet_ST_V6.cl:152)	n/a	n/a	n/a	Unrolled by #pragma unroll
Kernel: conv2d3x3 (squeezenet_ST_V6.cl:65)				Single work-item execution
conv2d3x3.B1 (squeezenet_ST_V6.cl:71)	Yes	>=1	n/a	
conv2d3x3.B2 (squeezenet_ST_V6.cl:75)	Yes	>=1	n/a	
conv2d3x3.B3 (squeezenet_ST_V6.cl:77)	Yes	>=1	n/a	
conv2d3x3.B4 (squeezenet_ST_V6.cl:83)	Yes	~136	II	Data dependency
Fully unrolled loop (squeezenet_ST_V6.cl:86)	n/a	n/a	n/a	Unrolled by #pragma unroll
Fully unrolled loop (squeezenet_ST_V6.cl:89)	n/a	n/a	n/a	Auto-unrolled
Kernel: maxpool2d (squeezenet_ST_V6.cl:18)				Single work-item execution
maxpool2d.B1 (squeezenet_ST_V6.cl:20)	Yes	>=1	n/a	
maxpool2d.B2 (squeezenet_ST_V6.cl:23)	Yes	>=1	n/a	
maxpool2d.B3 (squeezenet_ST_V6.cl:25)	Yes	>=1	n/a	
maxpool2d.B4 (squeezenet_ST_V6.cl:32)	Yes	>=1	n/a	
maxpool2d.B5 (squeezenet_ST_V6.cl:35)	Yes	~1	n/a	II is an approximation.

Figura 4.1: "*Loop analysis*" del reporte generado por el compilador HLS para la solución de estilo *Single Task*.

Kernel: avgpool2d (squeezenet_NDRange.cl:140)				ND-Range
avgpool2d.B1 (squeezenet_NDRange.cl:147)	Yes	537529691	n/a	Thread capacity = 157
Kernel: conv2d1x1 (squeezenet_NDRange.cl:102)				ND-Range
conv2d1x1.B1 (squeezenet_NDRange.cl:115)	Yes	48	n/a	Thread capacity = 217
conv2d1x1.B2 (squeezenet_NDRange.cl:120)	Yes	4294967295	n/a	Thread capacity = 205
Kernel: conv2d3x3 (squeezenet_NDRange.cl:53)				ND-Range
conv2d3x3.B1 (squeezenet_NDRange.cl:64)	Yes	538976288	n/a	Thread capacity = 422
conv2d3x3.B2 (squeezenet_NDRange.cl:70)	Yes	538976288	n/a	Thread capacity = 399
Kernel: maxpool2d (squeezenet_NDRange.cl:9)				ND-Range
maxpool2d.B1 (squeezenet_NDRange.cl:16)	Yes	1978633691	n/a	Thread capacity = 169
Coalesced loop (squeezenet_NDRange.cl:18)	n/a	n/a	n/a	• Auto-coale...
maxpool2d.B2 (squeezenet_NDRange.cl:25)	No	n/a	n/a	Thread capacity = 155

Figura 4.2: "*Loop analysis*" del reporte generado por el compilador HLS para la solución de estilo NDRange.

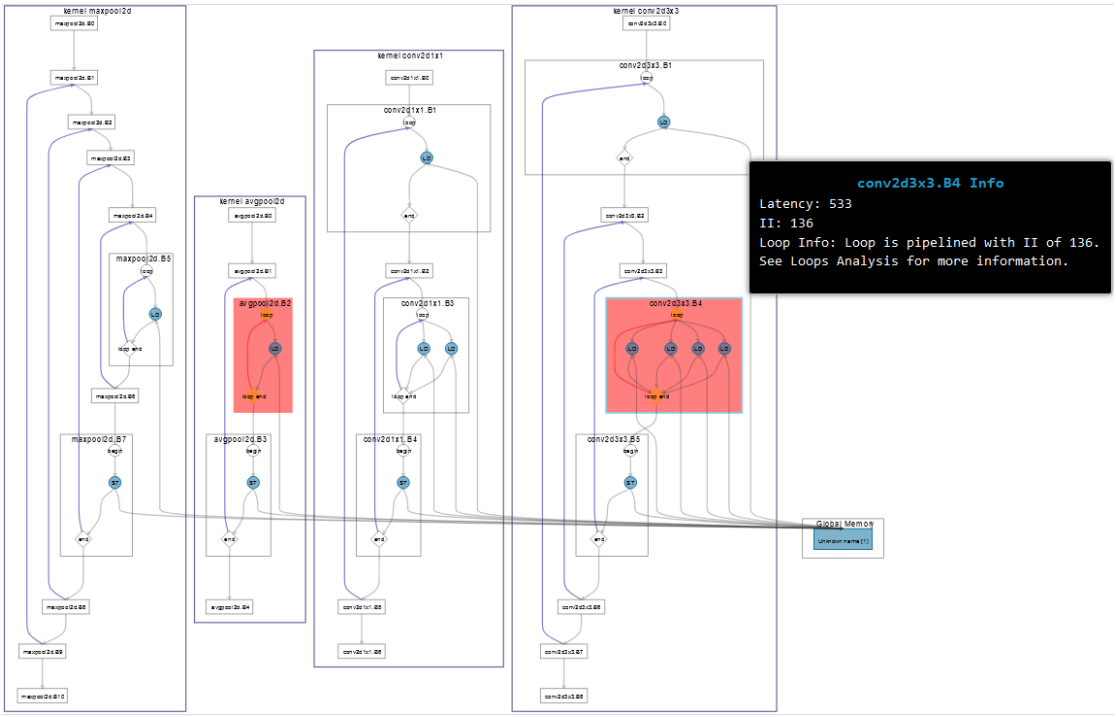

Figura 4.3: "*System viewer*" del reporte generado por el compilador HLS para la solución de estilo *Single Task*.

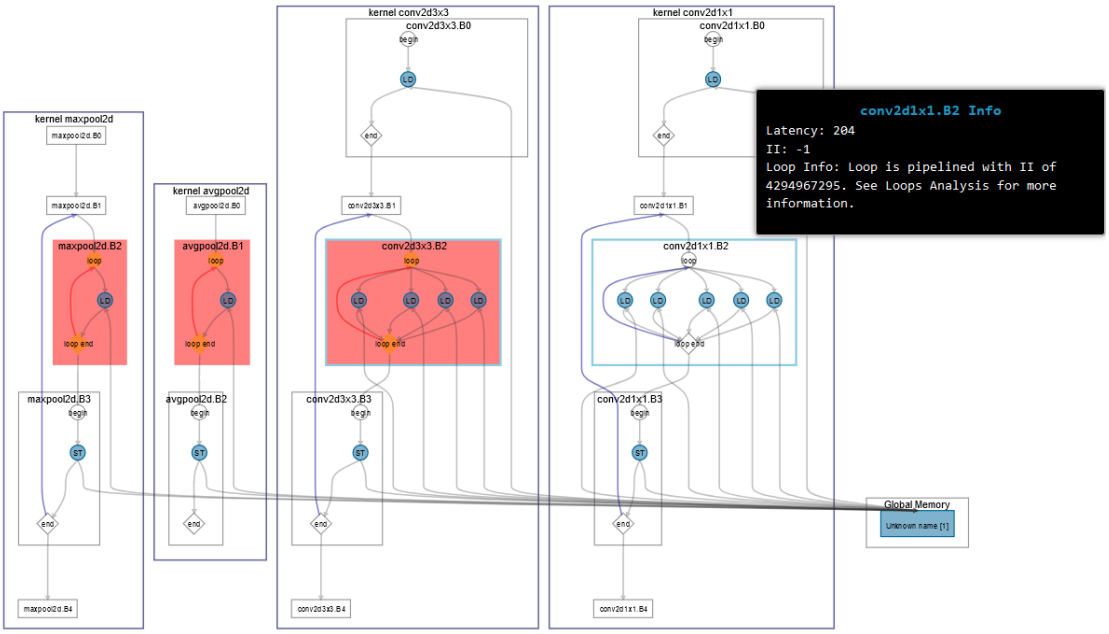

Figura 4.4: "*System viewer*" del reporte generado por el compilador HLS para la solución de estilo NDRange.

4.6. Conclusiones

Para concluir, ha sido posible crear una infraestructura multiplataforma basada en Jupyter Notebooks que nos permite realizar el diseño y la validación de nuestros *kernels* a diferentes niveles y en diferentes plataformas.

Esta infraestructura es accesible fácilmente para otros colaboradores, puesto que se encuentra todo en una imagen de disco de una máquina virtual y de la imagen del HPS que implementa nuestra placa de desarrollo.

Además, para poder realizar dichas validaciones sobre nuestra infraestructura, se han confeccionado diferentes bancos de prueba, fáciles de configurar. Esta parte es especialmente importante, puesto que es donde más tiempo del proyecto se ha invertido, con el objetivo de que el diseño y la validación sean lo más fácil, rápido y flexible, agilizando la obtención del *kernel* final.

Tras la correcta validación de nuestro diseño, somos capaces de observar en 4.8 que nuestro diseño de *kernel* del modelo SqueezeNet ha sido sintetizable correctamente para la placa DE10-Nano siendo un diseño factible para una FPGA de bajo coste y bajo consumo.

En cuanto a la comparación entre *Single Task* y NDRange, podemos establecer de forma general, para nuestro diseño, que las implementaciones de tipo *Single Task* utilizan considerablemente una menor cantidad de memoria. Sin embargo, en cuanto a la lógica utilizada, este tipo de implementaciones requieren ligeramente un mayor uso de estos recursos. Además, en cuanto a los registros, se observa como la aproximación *Single Task* utiliza una mayor cantidad de registros. Esto es debido a que se utilizan métodos de optimización basados en la inferencia de registros de desplazamientos.

Se comprueba que ambas capas de tipo pool, maxpool y average pool, presentan un mejor rendimiento cuando se implementan en con el estilo *Single Task*. Esto podría deberse a que estas capas no son lo suficientemente complejas, facilitando al compilador una síntesis óptima, a pesar de que por ejemplo en la capa avgpool se tenga un intervalo de iniciación de 16 ciclos de reloj.

También, podemos observar como en los informes de las implementaciones NDRange, nos ofrecen la información de la capacidad de hilos de los diferentes *kernels sintetizados*. Se comprueba que cuando la operación a realizar es mayor que el *Thread capacity*, el *kernel* reduce su rendimiento. Sin embargo, esto no parece ocurrir en la aproximación *Single Task*, puesto que es capaz de procesar de forma paralela a nivel de bucle cualquier carga de trabajo.

Por lo tanto, podemos establecer que ninguna aproximación, NDRange o *Single Task*, es mejor que otra y que la selección de una implementación de un tipo u otro dependerá del algoritmo a diseñar, así de como las dependencias que encontremos entre los datos siendo factible una implementación mixta para obtener una aceleración *hardware* lo más óptima posible.

Con todo esto, podemos concluir que los diferentes ítems establecidos como objetivos han sido conseguidos satisfactoriamente.

Capítulo 5

Discusión y futuras líneas de trabajo

Tras la obtención de los resultados del trabajo, se puede observar como el resultado del acelerador *Hardware* no nos habilita para aplicaciones en tiempo real. Sin embargo, en un contexto con especificaciones más relajadas podemos beneficiarnos del presente resultado. También cabe comentar que los *kernels* sintetizados no ocupan todos los recursos, dejando un pequeño espacio para añadir funcionalidades adicionales, otorgándole a nuestro sistema cierta flexibilidad. Además, gracias a la infraestructura que hemos establecido, puede resultar más ágil cualquier mejora que se quiera implementar para reducir el tiempo por predicción.

Sin embargo, si nos situamos en una situación en la que no hay requerimientos de tiempo estrictos, como en la monitorización y detección de incendios, por ejemplo, un tiempo de clasificación de 4 segundos no es para nada crítico, pero sí que lo es la eficiencia energética y el bajo consumo. Por ello, nuestra implementación en la placa de bajo coste y bajo consumo como la DE10-nano sería ideal para abordar esta situación.

En cuanto a lo relacionado con modelos de DL, la implementación en estilo *Single Task* permite mayor flexibilidad en cuanto a diferentes configuraciones de las convoluciones, pues se observa como en la implementación NDRange [7] requiere que los canales de entrada sean múltiplos de 4 para aprovechar al máximo el acelerador.

Sin embargo, es posible plantear diferentes líneas de trabajo con el objetivo de llegar a optimizar la implementación planteada en este trabajo.

Por un lado, con el objetivo de reducir el consumo de los recursos de forma considerable, se puede plantear la utilización de diferentes tipos de datos con una menor cantidad de bits, reduciendo la lógica de las operaciones a llevar a cabo en la aceleración *hardware*. Esta línea de trabajo ha sido explorada por la literatura previa a este trabajo, consiguiendo buenos resultados a cambio de incrementar en cierta medida la complejidad del diseño a llevar a cabo.

Por otro lado, siguiendo dentro de la línea de diseño *hls* con OpenCL, podemos plantear una solución mixta en la carga de trabajo sea repartida entre el HPS y PyTorch y las diferentes implementaciones en estilo *Single Task* y NDRange. En esta solución, sería necesario tener en cuenta el tiempo que se necesita para la transferencia de datos entre la FPGA y el HPS y viceversa.

También sería posible, dado que hay menos consumo de los recursos de memoria de la placa DE10-nano, profundizar en la utilización de memorias locales. Esta es una idea interesante, puesto que una vez la paralelización de bucle es óptima, con un II de 1, el siguiente cuello de botella se

encuentra en los accesos a la memoria global. Un claro ejemplo de esto se puede observar en la convolución 3x3, dado que hay píxeles que se acceden múltiples veces al realizar el desliz del filtro con un *stride* de 1.

Finalmente, una línea de trabajo interesante, facilitada por el fabricante Intel, consiste en la utilización de canales de Intel y *kernels* de tipo *autorun*. Los canales son nos permiten establecer una comunicación entre *kernels* de forma directa y sin necesidad de intervención del *host*. De esta forma, el *kernel autorun* irá procesando los datos de forma automática a medida que le van llegando. De esta forma, se debe establecer la estructura de la solución en una arquitectura de estilo productor/consumidor. El productor proporcionará los datos de forma ordenada al acelerador en modo *autorun* que simplemente se encargará en procesar los datos y entregárselas al consumidor para que ordene los resultados de la forma deseable,

Bibliografía

[1] Forrest N. Iandola. *SqueezeNet*. Commit: 51147dc. 2018. URL: https://github.com/forresti/SqueezeNet.

[2] Forrest N. Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size". En: (2016). arXiv: 1602.07360 [cs.CV].

[3] Pytorch Team. *SQUEEZENET model*. URL: https://pytorch.org/hub/pytorch_vision_squeezenet/ (visitado 27-06-2023).

[4] David Gschwend. "ZynqNet: An FPGA-Accelerated Embedded Convolutional Neural Network". En: *CoRR* abs/2005.06892 (2020). arXiv: 2005.06892. URL: https://arxiv.org/abs/2005.06892.

[5] Kathirgamaraja Pradeep et al. "EdgeNet: SqueezeNet like Convolution Neural Network on Embedded FPGA". En: *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 2018, págs. 81-84. DOI: 10.1109/ICECS.2018.8617876.

[6] Jingyuan Zhao et al. "Scalable FPGA-Based Convolutional Neural Network Accelerator for Embedded Systems". En: *2019 4th International Conference on Computational Intelligence and Applications (ICCIA)*. 2019, págs. 36-40. DOI: 10.1109/ICCIA.2019.00014.

[7] Er1cZ. *Deploying_CNN_on_FPGA_using_OpenCL*. Commit: 05f176f. 2017. URL: https://github.com/Er1cZ/Deploying_CNN_on_FPGA_using_OpenCL.

[8] Kunlun Bai. *A Comprehensive Introduction to Different Types of Convolutions in Deep Learning*. URL: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215 (visitado 28-06-2023).

[9] Intel. *Intel® FPGA SDK for OpenCL™ Software Technology*. URL: https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html (visitado 28-06-2023).

[10] Intel. *OpenCL™: Single-Threaded vs. Multi-Threaded Kernels*. URL: https://www.youtube.com/watch?v=2uqddr5UYm0 (visitado 29-06-2023).

[11] TerasIC. *DE10-Nano Kit*. URL: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=1046&PartNo=4#contents (visitado 20-06-2023).

[12] Intel. *FPGA Software Download Center*. URL: https://www.intel.com/content/www/us/en/collections/products/fpga/software/downloads.html?edition=standard&platform=windows&download_manager=dlm3&q=quartus%20prime%2018.1&s=Relevancy (visitado 30-06-2023).

Parte II

Anexos

Código de OpenCL que implementa la arquitectura CNN SqueezeNet en una aproximación Single Task

Documento “squeezenet_ST_V2.cl”:

```
/*
/*
////////////////////////////////////////////////////////////////////////////////

reduction of II for the kernel covn2d1x1 test (succesful)

////////////////////////////////////////////////////////////////////////////////
*/

//maxPool2d
//kernel_size=3 stride=2
//output one feature map per kernel
__kernel void maxpool2d(
        const int input_size,
        const int output_size,
    const int channel_size,
        __global const float* restrict input_im,
    __global float* restrict output_im)
{

    for(int channel_index = 0; channel_index < channel_size; channel_index++)
    {
        //loop over output feature map
        for(int i = 0; i < output_size; i++)//row
        {
            for(int j = 0; j < output_size; j++)//col
            {
                //find the max value in 3x3 reigon
                //to be one element in the output feature map
                float tmp = 0.0;

                #pragma unroll 1
                for(int k = 0; k < 3; k++)//row
                {
                    #pragma unroll 1
                    for(int l = 0; l < 3; l++)//col
                    {
                        float value = input_im[(i * 2 + k) * input_size\
                        + j * 2 + l ];
                        if(value > tmp)
                            tmp = value;
                    }
                }
                //store the result to output feature map
                output_im[i * output_size + j] = tmp;
            }
        }
```

```
        input_im += input_size * input_size;
        output_im += output_size * output_size;
    }
}

//3x3 convolution layer
//output one feature map per kernel
__kernel void conv2d3x3(
        const int input_channels, const int input_size,
        const int pad, const int stride,
        const int start_channel, //offset for 1x1 feature map in expand
        const int output_size,
        const int filter_size,
        __global const float* restrict input_im,
        __global const float* restrict filter_weight,
        __global const float* restrict filter_bias,
        __global float *restrict output_im
        )
{

        //filter_weight += filter_index * input_channels * 9;
        output_im += start_channel * output_size * output_size;

        //loop over output feature map
        for(int filter_index = 0; filter_index < filter_size; filter_index++)
        {
        float bias = filter_bias[filter_index];

                for(int i = 0; i < output_size; i++)
                {
            for(int j = 0; j < output_size; j++)
            {
                //compute one element in the output feature map
                float tmp = bias;

                //compute dot product of 2 input_channels x 3 x 3 matrix
                for(int k = 0; k < input_channels; k++)
                {
                    #pragma unroll
                    for(int l = 0; l < 3; l++)
                    {
                        int h = i * stride + l - pad;
                        for(int m = 0; m < 3; m++)
                        {
                            int w = j * stride + m - pad;
                            if((h >= 0) && (h < input_size) \
                            && (w >= 0) && (w < input_size))
                            {
                                tmp += input_im[k * input_size * input_size \
                                    + h * input_size + w] \
                                    * filter_weight[9 * k + 3 * l + m \
                                    + filter_index * input_channels * 9];
                            }
```

```
                        }
                    }
                }

                //add relu activation after conv
                output_im[i * output_size + j \
                + filter_index * output_size * output_size] \
                = (tmp > 0.0) ? tmp : 0.0;
            }
                }

        // filter_weight += input_channels * 9;
        // output_im += output_size * output_size;
        }
}

//1x1 convolution layer as a single kernel
//output one feature map per kernel
#define II_CYCLES 16

__kernel void conv2d1x1(
        const int input_channels,
        const int input_size,
        const int filter_size,
        __global const float* restrict input_im,
        __global const float* restrict filter_weight,
        __global const float* restrict filter_bias,
        __global float *restrict output_im)
{
    //loop over filters
        for(int f_i = 0; f_i < filter_size; f_i++)
        {
        float bias = filter_bias[f_i];

                for(int ij = 0; ij < (input_size * input_size); ij++)
                {

                        float shift_reg[II_CYCLES];

            for(int i = 0; i < II_CYCLES; i++)
            {
                shift_reg[i] = 0;
            }

                        for(int k = 0; k < input_channels; k++)
                        {
                                shift_reg[II_CYCLES - 1] = shift_reg[0] \
            + input_im[k * input_size * input_size + ij] \
            * filter_weight[k + f_i * input_channels];

                                #pragma unroll
                                for(int j = 0; j < (II_CYCLES - 1); j++)
                                {
```

```
                                        shift_reg[j] = shift_reg[j + 1];
                                }
                        }

                        float tmp = bias;
                        for(int i = 0; i < (II_CYCLES - 1); i++)
                        {
                                tmp += shift_reg[i];
                        }

                        //add relu after conv
                        output_im[ij + (input_size * input_size * f_i)] \
            = (tmp > 0.0) ? tmp : 0.0;
                    }
        }
}

//last layer use a 13 x 13 avgPool layer as classifier
//one class score per kernel
__kernel void avgpool2d(
        __global const float* restrict input_im,
        __global float* restrict output_im)
{
    //Since it's the final layer, we know that there are only 1000 classes

        for(int class_index = 0; class_index < 1000; class_index++)
    {

        float tmp = 0.0f;

        for(int i = 0; i < 169; i++)
        {
            tmp += input_im[class_index * 169 + i];
        }

        output_im[class_index] = tmp / 169.0;
    }
}
```

Código Python de ejemplo de los Jupyter Notebooks usados como bancos de prueba

Remarcar que los anexos de los Jupyter Notebook se añaden como pdf, por lo que no aparece la numeración de las páginas.

Documento “SqueezeNet_conv3x3_test_DE10NANO.ipynb”:

1 Debug SqueezeNet v1.3 (Simple Task) OpenCL implement with PyOpenCL and PyTorch

Partial code are copied heavily from https://github.com/pytorch/vision/blob/master/torchvision/models/squeezen
SqueezeNet Paper:https://arxiv.org/abs/1602.07360
SqueezeNet 1.1 model from https://github.com/DeepScale/SqueezeNet/tree/master/SqueezeNet_v1.1
SqueezeNet 1.1 has 2.4x less computation and slightly fewer parameters than SqueezeNet 1.0, without sacrificing accuracy.

TEST DE IMPLEMENTACIÓN CONV3x3

[1]:
```
#some set up
import os
import numpy as np
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
from torch.autograd import Variable
import torch.utils.data
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from PIL import Image
import math
import time
from time import sleep, perf_counter as pc
from matplotlib.pyplot import imshow
%matplotlib inline
```

[2]:
```
# OpenCL setup
import pyopencl as cl
import sys
sys.path.append('../python_common')
import deviceinfo
from time import time

#wksp = '../device/v1.3/conv3x3'
```

1.1 Veamos ahora solo conv3x3 con opencl

Step1: OpenCL preparation

[3]:
```
platforms = cl.get_platforms()
context = cl.Context(
        dev_type=cl.device_type.ALL,
        properties=[(cl.context_properties.PLATFORM, platforms[0])])

device = platforms[0].get_devices()
```

```
queue = cl.CommandQueue(context)

context
```

[3]:
```
<pyopencl.Context at 0x-46565ae0 on <pyopencl.Device 'de10_nano_sharedonly :
Cyclone V SoC Development Kit' on 'Intel(R) FPGA SDK for OpenCL(TM)' at
0x-5b088690>>
```

Step 2: creat kernels Creat & build program

Create kernels

[4]:
```
wksp = ''

file_dir = wksp + 'conv3x3_NDRange.aocx'

kernelSource = open(file_dir, mode='rb').read()
program_NDR = cl.Program(context, device, [kernelSource]).build()

# file_dir = wksp + 'conv3x3_ST.aocx'
# file_dir = wksp + 'conv3x3_ST_V2.aocx'
file_dir = wksp + 'conv3x3_ST_V3.aocx'
# file_dir = wksp + 'squeezenet_ST_V3.aocx'
# file_dir = wksp + 'squeezenet_ST_V4.aocx'
# file_dir = wksp + 'squeezenet_ST_V5.aocx'
# file_dir = wksp + 'squeezenet_ST_V6.aocx'
# file_dir = wksp + 'squeezenet_ST_V7.aocx'

kernelSource = open(file_dir, mode='rb').read()
program_ST = cl.Program(context, device, [kernelSource]).build()
```

```
Reprogramming device [0] with handle 1
/usr/local/lib/python3.8/site-packages/pyopencl-2022.2.4-py3.8-linux-
armv7l.egg/pyopencl/__init__.py:270: CompilerWarning: Non-empty compiler output
encountered. Set the environment variable PYOPENCL_COMPILER_OUTPUT=1 to see
more.
  warn("Non-empty compiler output encountered. Set the "
```

Creat kernels

[5]:
```
conv3x3_NDR = program_NDR.conv2d3x3
conv3x3_NDR.set_scalar_arg_dtypes([np.int32, np.int32, np.int32, np.int32, np.
 ↪int32, np.int32, None, None, None, None])

conv3x3_ST = program_ST.conv2d3x3
conv3x3_ST.set_scalar_arg_dtypes([np.int32, np.int32, np.int32, np.int32, np.
 ↪int32, np.int32, np.int32, None, None, None, None])
```

Run OpenCL implement

[6]:
```
tamanyo=13 #input_size
canales_iniciales=64 #input_channels
canales_contraidos=64 #filter_size
canales_finales = 256

acumulado_pytorch=0

imagen = np.random.randint(10,size=(1,canales_contraidos, tamanyo, tamanyo)).
 ↪astype(np.float32)
#imagen = np.ones((1,canales_contraidos, tamanyo, tamanyo)).astype(np.float32)

weights1=np.random.randint(10,size=(canales_finales, canales_iniciales, 3, 3)).
 ↪astype(np.float32)
bias1=np.random.randint(10,size=(canales_finales,)).astype(np.float32)

#weights1=np.ones((canales_iniciales, canales_finales, 3, 3)).astype(np.float32)
#bias1=np.ones((canales_iniciales,)).astype(np.float32)

squeeze_activation = nn.ReLU(inplace=True)

tic=pc()
squeeze1=nn.Conv2d(canales_iniciales, canales_finales, kernel_size=3,␣
 ↪bias=False, padding=1)
squeeze1.weight = nn.Parameter(torch.from_numpy(weights1))
squeeze1.bias = nn.Parameter(torch.from_numpy(bias1))

imagen1  = torch.from_numpy(imagen).float()

salida1=squeeze1(imagen1)
salida1_activation=squeeze_activation(salida1)

salida1_a_numpy=salida1_activation.detach().numpy()

toc=pc()
acumulado_pytorch=toc-tic+acumulado_pytorch

####### OPENCL COMPARISON #######
```

[7]:
```
# NDRANGE

h_sample = imagen.reshape(-1).astype(np.float32)
d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=h_sample)

fire1_expand3x3_weight = weights1.reshape(-1)
fire1_expand3x3_bias = bias1
```

```
d_fire1_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_weight)
d_fire1_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_bias)

h_result_fire1_expand = np.empty(1 * canales_finales * tamanyo * tamanyo).
 ↪astype(np.float32) # we wil only check 3x3 convolituional performance
d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire1_expand.nbytes)

#offset = np.int32(canales_iniciales)
offset = np.int32(0)

tic2 = pc()

conv3x3_NDR(queue,(canales_finales, tamanyo), None, canales_contraidos, tamanyo,␣
 ↪1, 1, offset, tamanyo, d_result_fire1_squeeze, d_fire1_expand3x3_weight,␣
 ↪d_fire1_expand3x3_bias, d_result_fire1_expand)

queue.finish()

cl.enqueue_copy(queue, h_result_fire1_expand, d_result_fire1_expand)

queue.finish()

veamos = h_result_fire1_expand.reshape(-1,tamanyo,tamanyo)

rtime = pc() - tic2
```

[9]:
```
# Simple task

h_sample = imagen.reshape(-1).astype(np.float32)
d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=h_sample)

fire1_expand3x3_weight = weights1.reshape(-1)
fire1_expand3x3_bias = bias1

d_fire1_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_weight)
d_fire1_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_bias)

h_result_fire1_expand = np.empty(1 * canales_finales * tamanyo * tamanyo).
 ↪astype(np.float32) # we wil only check 3x3 convolituional performance
```

```
d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire1_expand.nbytes)

#offset = np.int32(canales_iniciales)
offset = np.int32(0)

tic3 = pc()

conv3x3_ST(queue,(1,), None, np.int32(canales_iniciales), tamanyo, 1, 1, offset,␣
 ↪tamanyo, canales_finales, d_result_fire1_squeeze, d_fire1_expand3x3_weight,␣
 ↪d_fire1_expand3x3_bias, d_result_fire1_expand)

queue.finish()

cl.enqueue_copy(queue, h_result_fire1_expand, d_result_fire1_expand)

queue.finish()

veamos1 = h_result_fire1_expand.reshape(-1,tamanyo,tamanyo)

rtime1 = pc() - tic3
```

[10]:
```
print ("tiempo en segundos con pytorch= ", toc-tic)
print ("tiempo en segundos con opencl (NDRANGE)=",rtime)
print ("tiempo en segundos con opencl (Simple Task)=",rtime1)

comparativa1=np.allclose(salida1_a_numpy, veamos,rtol=1e-01, atol=1e-01)
comparativa2=np.allclose(salida1_a_numpy, veamos1,rtol=1e-01, atol=1e-01)
comparativa3=np.allclose(veamos, veamos1,rtol=1e-01, atol=1e-01)

print("comparativa (pytorch == NDRange): ",comparativa1)
print("comparativa (pytorch == Simple Task): ",comparativa2)
print("comparativa (NDRange == Simple Task): ",comparativa3)
```

```
tiempo en segundos con pytorch=  0.17989878799926373
tiempo en segundos con opencl (NDRANGE)= 0.0644559569991543
tiempo en segundos con opencl (Simple Task)= 0.22025866200056043
comparativa (pytorch == NDRange):  True
comparativa (pytorch == Simple Task):  True
comparativa (NDRange == Simple Task):  True
```

Obtención tiempo de ejecución ST

[11]:
```
tamanyo=13 #input_size
canales_iniciales=64 #input_channels
canales_contraidos=64
canales_finales = 256 #filter_size
```

```
count = 50

acumulado_pytorch=0
acumulado_kernel=0
comparativa4 = True

for i in range(count):
    imagen = np.random.randint(10,size=(1,canales_contraidos, tamanyo, tamanyo)).
 ↪astype(np.float32)
    #imagen = np.ones((1,canales_contraidos, tamanyo, tamanyo)).astype(np.
 ↪float32)

    weights1=np.random.randint(10,size=(canales_finales, canales_contraidos, 3,␣
 ↪3)).astype(np.float32)
    bias1=np.random.randint(10,size=(canales_finales,)).astype(np.float32)

    #weights1=np.ones((canales_finales, canales_contraidos, 3, 3)).astype(np.
 ↪float32)
    #bias1=np.ones((canales_finales,)).astype(np.float32)

    squeeze_activation = nn.ReLU(inplace=True)

    tic=pc()
    squeeze1=nn.Conv2d(canales_iniciales, canales_finales, kernel_size=3,␣
 ↪bias=False, padding=1)
    squeeze1.weight = nn.Parameter(torch.from_numpy(weights1))
    squeeze1.bias = nn.Parameter(torch.from_numpy(bias1))

    imagen1  = torch.from_numpy(imagen).float()

    salida1=squeeze1(imagen1)
    salida1_activation=squeeze_activation(salida1)

    salida1_a_numpy=salida1_activation.detach().numpy()

    toc=pc()
    acumulado_pytorch=toc-tic+acumulado_pytorch

    ####### OPENCL COMPARISON #######

    h_sample = imagen.reshape(-1).astype(np.float32)
    d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=h_sample)

    fire1_expand3x3_weight = weights1.reshape(-1)
    fire1_expand3x3_bias = bias1
```

```
    d_fire1_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_weight)
    d_fire1_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_bias)

    h_result_fire1_expand = np.empty(1 * canales_finales * tamanyo * tamanyo).
↪astype(np.float32) # we wil only check 3x3 convolituional performance
    d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_expand.nbytes)

    #offset = np.int32(canales_iniciales)
    offset = np.int32(0)

    tic5 = pc()

    conv3x3_ST(queue,(1,), None, np.int32(canales_contraidos), tamanyo, 1, 1,␣
↪offset, tamanyo, canales_finales, d_result_fire1_squeeze,␣
↪d_fire1_expand3x3_weight, d_fire1_expand3x3_bias, d_result_fire1_expand)

    queue.finish()

    cl.enqueue_copy(queue, h_result_fire1_expand, d_result_fire1_expand)

    queue.finish()

    veamos1 = h_result_fire1_expand.reshape(-1,tamanyo,tamanyo)

    toc5 = pc()

    acumulado_kernel = toc5 - tic5 + acumulado_kernel

    comparativa4 &= np.allclose(salida1_a_numpy, veamos1,rtol=1e-01, atol=1e-01)

    if (comparativa4 == False):
        print('Error en : ', i)
        break

print ("tiempo en segundos con pytorch= ", acumulado_pytorch/count)
print("tiempo en segundos con opencl (", file_dir[:-5], ")=",acumulado_kernel/
↪count)
print("comparativa (pytorch ==", file_dir[:-5], "): ",comparativa4)
```

```
tiempo en segundos con pytorch=  0.09690514955986146
tiempo en segundos con opencl ( conv3x3_ST_V3 )= 0.21774989399997138
comparativa (pytorch == conv3x3_ST_V3 ):  True
```

Obtención de tiempo de ejecución de NDRange

[19]:
```
tamanyo=13 #input_size
canales_iniciales=64 #input_channels
canales_contraidos=64
canales_finales = 256 #filter_size

count = 50

acumulado_pytorch=0
acumulado_kernel=0
comparativa4 = True

for i in range(count):
    imagen = np.random.randint(10,size=(1,canales_contraidos, tamanyo, tamanyo)).
↪astype(np.float32)
    #imagen = np.ones((1,canales_contraidos, tamanyo, tamanyo)).astype(np.
↪float32)

    weights1=np.random.randint(10,size=(canales_finales, canales_contraidos, 3,␣
↪3)).astype(np.float32)
    bias1=np.random.randint(10,size=(canales_finales,)).astype(np.float32)

    #weights1=np.ones((canales_finales, canales_contraidos, 3, 3)).astype(np.
↪float32)
    #bias1=np.ones((canales_finales,)).astype(np.float32)

    squeeze_activation = nn.ReLU(inplace=True)

    tic=pc()
    squeeze1=nn.Conv2d(canales_iniciales, canales_finales, kernel_size=3,␣
↪bias=False, padding=1)
    squeeze1.weight = nn.Parameter(torch.from_numpy(weights1))
    squeeze1.bias = nn.Parameter(torch.from_numpy(bias1))

    imagen1  = torch.from_numpy(imagen).float()

    salida1=squeeze1(imagen1)
    salida1_activation=squeeze_activation(salida1)

    salida1_a_numpy=salida1_activation.detach().numpy()

    toc=pc()

    acumulado_pytorch=toc-tic+acumulado_pytorch

    # NDRANGE

    h_sample = imagen.reshape(-1).astype(np.float32)
```

```
    d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
↪mem_flags.COPY_HOST_PTR, hostbuf=h_sample)

    fire1_expand3x3_weight = weights1.reshape(-1)
    fire1_expand3x3_bias = bias1

    d_fire1_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_weight)
    d_fire1_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_bias)

    h_result_fire1_expand = np.empty(1 * canales_finales * tamanyo * tamanyo).
↪astype(np.float32) # we wil only check 3x3 convolituional performance
    d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_expand.nbytes)

    #offset = np.int32(canales_iniciales)
    offset = np.int32(0)

    tic5 = pc()

    conv3x3_NDR(queue,(canales_finales, tamanyo), None, canales_contraidos,␣
↪tamanyo, 1, 1, offset, tamanyo, d_result_fire1_squeeze,␣
↪d_fire1_expand3x3_weight, d_fire1_expand3x3_bias, d_result_fire1_expand)

    queue.finish()

    cl.enqueue_copy(queue, h_result_fire1_expand, d_result_fire1_expand)

    queue.finish()

    veamos = h_result_fire1_expand.reshape(-1,tamanyo,tamanyo)

    toc5 = pc()

    acumulado_kernel = toc5 - tic5 + acumulado_kernel

    comparativa4 &= np.allclose(salida1_a_numpy, veamos, rtol=1e-01, atol=1e-01)

print ("tiempo en segundos con pytorch= ", acumulado_pytorch/count)
print ("tiempo en segundos con opencl (NDRange)=",acumulado_kernel/count)

print("comparativa (pytorch == NDRange): ",comparativa4)
```

```
tiempo en segundos con pytorch=  0.09741130668000324
tiempo en segundos con opencl (NDRange)= 0.05543484802001785
comparativa (pytorch == NDRange):  True
```

Comprobación de que elementos del resultado no coinciden con el modelo ideal

[11]:
```
for i in range(canales_contraidos):
    for j in range(tamanyo):
        for k in range(tamanyo):
            if (abs(salida1_a_numpy.reshape(-1,tamanyo,tamanyo)[i][j][k] -␣
↪veamos1[i][j][k])) > 1e-01:
                print("i:", i, "j:", j, "k:", k, salida1_a_numpy.
↪reshape(-1,tamanyo,tamanyo)[i][j][k], veamos1[i][j][k])
```

Documento “SqueezeNet_full_test_DE10NANO.ipynb”:

1 Debug SqueezeNet v1.3 (Simple Task) OpenCL implement with PyOpenCL and PyTorch

Partial code are copied heavily from https://github.com/pytorch/vision/blob/master/torchvision/models/squeezen
SqueezeNet Paper:https://arxiv.org/abs/1602.07360
SqueezeNet 1.1 model from https://github.com/DeepScale/SqueezeNet/tree/master/SqueezeNet_v1.1
SqueezeNet 1.1 has 2.4x less computation and slightly fewer parameters than SqueezeNet 1.0, without sacrificing accuracy.

Tests arquitecura completa

[1]:
```
#some set up
import os
import numpy as np
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
from torch.autograd import Variable
import torch.utils.data
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from PIL import Image
import math
import time
from time import sleep, perf_counter as pc
from matplotlib.pyplot import imshow
%matplotlib inline
```

[2]:
```
# OpenCL setup
import pyopencl as cl
import sys
sys.path.append('../python_common')
import deviceinfo
from time import time

#wksp = '../device/v1.3/squeezenet'
```

1.1 Veamos ahora con opencl

Step1: OpenCL preparation

[3]:
```
## Devices and compute context
platforms = cl.get_platforms()
context = cl.Context(
        dev_type=cl.device_type.ALL,
        properties=[(cl.context_properties.PLATFORM, platforms[0])])
device = platforms[0].get_devices()
```

```
# Create a command queue
queue = cl.CommandQueue(context)
queue1 = cl.CommandQueue(context)

context
```

[3]:
```
<pyopencl.Context at 0x-46610b30 on <pyopencl.Device 'de10_nano_sharedonly :
Cyclone V SoC Development Kit' on 'Intel(R) FPGA SDK for OpenCL(TM)' at
0x-5b12d690>>
```

Step 2: creat kernels Creat & build program

Create kernels

[13]:
```
wksp = ''

#file_dir = wksp + 'squeezenet_NDRange.aocx'
file_dir = wksp + 'squeezenet.aocx'

kernelSource = open(file_dir, mode='rb').read()
program_NDR = cl.Program(context, device, [kernelSource]).build()

# file_dir = wksp + 'squeezenet_ST.aocx'
# file_dir = wksp + 'squeezenet_ST_fp_relaxed.aocx'
# file_dir = wksp + 'squeezenet_ST_V2.aocx' #4s
#file_dir = wksp + 'squeezenet_ST_V3.aocx' #40s
# file_dir = wksp + 'squeezenet_ST_V4.aocx' #5s
#file_dir = wksp + 'squeezenet_ST_V5.aocx' #45s
# file_dir = wksp + 'squeezenet_ST_V2_fp_relaxed.aocx' # 4.3671s
file_dir = wksp + 'squeezenet_ST_V6.aocx' #4s
# file_dir = wksp + 'squeezenet_ST_V7.aocx' #4

kernelSource = open(file_dir, mode='rb').read()
program_ST = cl.Program(context, device, [kernelSource]).build()
```

Creat kernels

[14]:
```
conv3x3_NDR = program_NDR.conv2d3x3
conv3x3_NDR.set_scalar_arg_dtypes([np.int32, np.int32, np.int32, np.int32, np.
 ↪int32, np.int32, None, None, None, None])

maxpool_NDR = program_NDR.maxpool2d
maxpool_NDR.set_scalar_arg_dtypes([np.int32, np.int32, None, None])

conv1x1_NDR = program_NDR.conv2d1x1
conv1x1_NDR.set_scalar_arg_dtypes([np.int32, np.int32, None, None, None, None])

avgpool_NDR = program_NDR.avgpool2d
```

```
avgpool_NDR.set_scalar_arg_dtypes([None, None])

conv3x3_ST = program_ST.conv2d3x3
conv3x3_ST.set_scalar_arg_dtypes([np.int32, np.int32, np.int32, np.int32, np.
 ↪int32, np.int32, np.int32, None, None, None, None])

maxpool_ST = program_ST.maxpool2d
maxpool_ST.set_scalar_arg_dtypes([np.int32, np.int32, np.int32, None, None])

conv1x1_ST = program_ST.conv2d1x1
conv1x1_ST.set_scalar_arg_dtypes([np.int32, np.int32, np.int32, None, None,␣
 ↪None, None])

avgpool_ST = program_ST.avgpool2d
avgpool_ST.set_scalar_arg_dtypes([None, None])
```

Run OpenCL implement

[6]:
```
squeeze_activation = nn.ReLU(inplace=True)

acumulado_pytorch = 0

transform = transforms.Compose([
transforms.Resize(256),
transforms.CenterCrop(224),
transforms.ToTensor(),
transforms.Normalize(mean = [ 0.485, 0.456, 0.406 ],
                     std = [ 0.229, 0.224, 0.225 ]),])

imagen = np.random.rand(3, 224, 224).astype(np.float32)
imagen = Image.fromarray(imagen,'RGB')
imagen = transform(imagen).numpy()
imagen = imagen[np.newaxis,:]

# imagen = np.ones((1, 3, 224, 224))
#print(imagen.shape)

### First Conv3x3 and maxpool

weights0=np.random.rand(64, 3,3,3).astype(np.float32)
bias0=np.random.rand(64,).astype(np.float32)

#### FIRE 1 ####
weights1=np.random.rand(16, 64,1,1).astype(np.float32)
bias1=np.random.rand(16,).astype(np.float32)

weights2a=np.random.rand(64, 16,1,1).astype(np.float32)
```

```
bias2a=np.random.rand(64,).astype(np.float32)

weights2b=np.random.rand(64, 16,3,3).astype(np.float32)
bias2b=np.random.rand(64,).astype(np.float32)

#### FIRE 2 ####
weights3=np.random.rand(16, 128,1,1).astype(np.float32)
bias3=np.random.rand(16,).astype(np.float32)

weights4a=np.random.rand(64, 16,1,1).astype(np.float32)
bias4a=np.random.rand(64,).astype(np.float32)

weights4b=np.random.rand(64, 16,3,3).astype(np.float32)
bias4b=np.random.rand(64,).astype(np.float32)

#### FIRE 3 ####
weights5=np.random.rand(32, 128,1,1).astype(np.float32)
bias5=np.random.rand(32,).astype(np.float32)

weights6a=np.random.rand(128, 32,1,1).astype(np.float32)
bias6a=np.random.rand(128,).astype(np.float32)

weights6b=np.random.rand(128, 32,3,3).astype(np.float32)
bias6b=np.random.rand(128,).astype(np.float32)

#### FIRE 4 ####
weights7=np.random.rand(32, 256,1,1).astype(np.float32)
bias7=np.random.rand(32,).astype(np.float32)

weights8a=np.random.rand(128, 32,1,1).astype(np.float32)
bias8a=np.random.rand(128,).astype(np.float32)

weights8b=np.random.rand(128, 32,3,3).astype(np.float32)
bias8b=np.random.rand(128,).astype(np.float32)

#### FIRE 5 ####
weights9=np.random.rand(48, 256,1,1).astype(np.float32)
bias9=np.random.rand(48,).astype(np.float32)

weights10a=np.random.rand(192, 48,1,1).astype(np.float32)
bias10a=np.random.rand(192,).astype(np.float32)

weights10b=np.random.rand(192, 48,3,3).astype(np.float32)
bias10b=np.random.rand(192,).astype(np.float32)

#### FIRE 6 ####
weights11=np.random.rand(48, 384,1,1).astype(np.float32)
```

```
bias11=np.random.rand(48,).astype(np.float32)

weights12a=np.random.rand(192, 48,1,1).astype(np.float32)
bias12a=np.random.rand(192,).astype(np.float32)

weights12b=np.random.rand(192, 48,3,3).astype(np.float32)
bias12b=np.random.rand(192,).astype(np.float32)

#### FIRE 7 ####
weights13=np.random.rand(64, 384,1,1).astype(np.float32)
bias13=np.random.rand(64,).astype(np.float32)

weights14a=np.random.rand(256, 64,1,1).astype(np.float32)
bias14a=np.random.rand(256,).astype(np.float32)

weights14b=np.random.rand(256, 64,3,3).astype(np.float32)
bias14b=np.random.rand(256,).astype(np.float32)

#### FIRE 8 ####
weights15=np.random.rand(64, 512,1,1).astype(np.float32)
bias15=np.random.rand(64,).astype(np.float32)

weights16a=np.random.rand(256, 64,1,1).astype(np.float32)
bias16a=np.random.rand(256,).astype(np.float32)

weights16b=np.random.rand(256, 64,3,3).astype(np.float32)
bias16b=np.random.rand(256,).astype(np.float32)

### Classifier Conv3x3 and maxpool

weights17=np.random.rand(1000, 512,1,1).astype(np.float32)
bias17=np.random.rand(1000,).astype(np.float32)

#########################################################

params = torch.load('squeezenet1_1.pth')

### First Conv3x3 and maxpool
weights0=params['features.0.weight'].numpy()
bias0=params['features.0.bias'].numpy()

######## BLOCK 1 ########
#fire - fire - maxpool
#### FIRE 1 ####
weights1=params['features.3.squeeze.weight'].numpy()
bias1=params['features.3.squeeze.bias'].numpy()
```

```
weights2a=params['features.3.expand1x1.weight'].numpy()
bias2a=params['features.3.expand1x1.bias'].numpy()

weights2b=params['features.3.expand3x3.weight'].numpy()
bias2b=params['features.3.expand3x3.bias'].numpy()

#### FIRE 2 ####
weights3=params['features.4.squeeze.weight'].numpy()
bias3=params['features.4.squeeze.bias'].numpy()

weights4a=params['features.4.expand1x1.weight'].numpy()
bias4a=params['features.4.expand1x1.bias'].numpy()

weights4b=params['features.4.expand3x3.weight'].numpy()
bias4b=params['features.4.expand3x3.bias'].numpy()

######## BLOCK 2 ########
#fire - fire - maxpool
#### FIRE 3 ####
weights5=params['features.6.squeeze.weight'].numpy()
bias5=params['features.6.squeeze.bias'].numpy()

weights6a=params['features.6.expand1x1.weight'].numpy()
bias6a=params['features.6.expand1x1.bias'].numpy()

weights6b=params['features.6.expand3x3.weight'].numpy()
bias6b=params['features.6.expand3x3.bias'].numpy()

#### FIRE 4 ####
weights7=params['features.7.squeeze.weight'].numpy()
bias7=params['features.7.squeeze.bias'].numpy()

weights8a=params['features.7.expand1x1.weight'].numpy()
bias8a=params['features.7.expand1x1.bias'].numpy()

weights8b=params['features.7.expand3x3.weight'].numpy()
bias8b=params['features.7.expand3x3.bias'].numpy()

######## BLOCK 3 ########
#fire - fire - fire - fire
#### FIRE 5 ####
weights9=params['features.9.squeeze.weight'].numpy()
bias9=params['features.9.squeeze.bias'].numpy()

weights10a=params['features.9.expand1x1.weight'].numpy()
bias10a=params['features.9.expand1x1.bias'].numpy()
```

```
weights10b=params['features.9.expand3x3.weight'].numpy()
bias10b=params['features.9.expand3x3.bias'].numpy()

#### FIRE 6 ####
weights11=params['features.10.squeeze.weight'].numpy()
bias11=params['features.10.squeeze.bias'].numpy()

weights12a=params['features.10.expand1x1.weight'].numpy()
bias12a=params['features.10.expand1x1.bias'].numpy()

weights12b=params['features.10.expand3x3.weight'].numpy()
bias12b=params['features.10.expand3x3.bias'].numpy()

#### FIRE 7 ####
weights13=params['features.11.squeeze.weight'].numpy()
bias13=params['features.11.squeeze.bias'].numpy()

weights14a=params['features.11.expand1x1.weight'].numpy()
bias14a=params['features.11.expand1x1.bias'].numpy()

weights14b=params['features.11.expand3x3.weight'].numpy()
bias14b=params['features.11.expand3x3.bias'].numpy()

#### FIRE 8 ####
weights15=params['features.12.squeeze.weight'].numpy()
bias15=params['features.12.squeeze.bias'].numpy()

weights16a=params['features.12.expand1x1.weight'].numpy()
bias16a=params['features.12.expand1x1.bias'].numpy()

weights16b=params['features.12.expand3x3.weight'].numpy()
bias16b=params['features.12.expand3x3.bias'].numpy()

######## Classifier ########
#conv3x3 - avgpool
### Classifier Conv3x3 and avgpool
weights17=params['classifier.1.weight'].numpy()
bias17=params['classifier.1.bias'].numpy()

tic=pc()

squeeze0=nn.Conv2d(3, 64, kernel_size=3, bias=False, stride=2)
squeeze0.weight = nn.Parameter(torch.from_numpy(weights0))
squeeze0.bias = nn.Parameter(torch.from_numpy(bias0))

maxpool=nn.MaxPool2d(3, stride=2)
```

```
squeeze1=nn.Conv2d(64, 16, kernel_size=1, bias=False)
squeeze1.weight = nn.Parameter(torch.from_numpy(weights1))
squeeze1.bias = nn.Parameter(torch.from_numpy(bias1))

squeeze2a=nn.Conv2d(16, 64, kernel_size=1, bias=False)
squeeze2a.weight = nn.Parameter(torch.from_numpy(weights2a))
squeeze2a.bias = nn.Parameter(torch.from_numpy(bias2a))

squeeze2b=nn.Conv2d(16, 64, kernel_size=3, bias=False, padding=1)
squeeze2b.weight = nn.Parameter(torch.from_numpy(weights2b))
squeeze2b.bias = nn.Parameter(torch.from_numpy(bias2b))

squeeze3=nn.Conv2d(128, 16, kernel_size=1, bias=False)
squeeze3.weight = nn.Parameter(torch.from_numpy(weights3))
squeeze3.bias = nn.Parameter(torch.from_numpy(bias3))

squeeze4a=nn.Conv2d(16, 64, kernel_size=1, bias=False)
squeeze4a.weight = nn.Parameter(torch.from_numpy(weights4a))
squeeze4a.bias = nn.Parameter(torch.from_numpy(bias4a))

squeeze4b=nn.Conv2d(16, 64, kernel_size=3, bias=False, padding=1)
squeeze4b.weight = nn.Parameter(torch.from_numpy(weights4b))
squeeze4b.bias = nn.Parameter(torch.from_numpy(bias4b))

squeeze5=nn.Conv2d(128, 32, kernel_size=1, bias=False)
squeeze5.weight = nn.Parameter(torch.from_numpy(weights5))
squeeze5.bias = nn.Parameter(torch.from_numpy(bias5))

squeeze6a=nn.Conv2d(32, 128, kernel_size=1, bias=False)
squeeze6a.weight = nn.Parameter(torch.from_numpy(weights6a))
squeeze6a.bias = nn.Parameter(torch.from_numpy(bias6a))

squeeze6b=nn.Conv2d(32, 128, kernel_size=3, bias=False, padding=1)
squeeze6b.weight = nn.Parameter(torch.from_numpy(weights6b))
squeeze6b.bias = nn.Parameter(torch.from_numpy(bias6b))

squeeze7=nn.Conv2d(256, 32, kernel_size=1, bias=False)
squeeze7.weight = nn.Parameter(torch.from_numpy(weights7))
squeeze7.bias = nn.Parameter(torch.from_numpy(bias7))

squeeze8a=nn.Conv2d(32, 128, kernel_size=1, bias=False)
squeeze8a.weight = nn.Parameter(torch.from_numpy(weights8a))
squeeze8a.bias = nn.Parameter(torch.from_numpy(bias8a))

squeeze8b=nn.Conv2d(32, 128, kernel_size=3, bias=False, padding=1)
squeeze8b.weight = nn.Parameter(torch.from_numpy(weights8b))
squeeze8b.bias = nn.Parameter(torch.from_numpy(bias8b))
```

```
squeeze9=nn.Conv2d(256, 48, kernel_size=1, bias=False)
squeeze9.weight = nn.Parameter(torch.from_numpy(weights9))
squeeze9.bias = nn.Parameter(torch.from_numpy(bias9))

squeeze10a=nn.Conv2d(48, 192, kernel_size=1, bias=False)
squeeze10a.weight = nn.Parameter(torch.from_numpy(weights10a))
squeeze10a.bias = nn.Parameter(torch.from_numpy(bias10a))

squeeze10b=nn.Conv2d(48, 192, kernel_size=3, bias=False, padding=1)
squeeze10b.weight = nn.Parameter(torch.from_numpy(weights10b))
squeeze10b.bias = nn.Parameter(torch.from_numpy(bias10b))

squeeze11=nn.Conv2d(384, 48, kernel_size=1, bias=False)
squeeze11.weight = nn.Parameter(torch.from_numpy(weights11))
squeeze11.bias = nn.Parameter(torch.from_numpy(bias11))

squeeze12a=nn.Conv2d(48, 192, kernel_size=1, bias=False)
squeeze12a.weight = nn.Parameter(torch.from_numpy(weights12a))
squeeze12a.bias = nn.Parameter(torch.from_numpy(bias12a))

squeeze12b=nn.Conv2d(48, 192, kernel_size=3, bias=False, padding=1)
squeeze12b.weight = nn.Parameter(torch.from_numpy(weights12b))
squeeze12b.bias = nn.Parameter(torch.from_numpy(bias12b))

squeeze13=nn.Conv2d(384, 64, kernel_size=1, bias=False)
squeeze13.weight = nn.Parameter(torch.from_numpy(weights13))
squeeze13.bias = nn.Parameter(torch.from_numpy(bias13))

squeeze14a=nn.Conv2d(64, 256, kernel_size=1, bias=False)
squeeze14a.weight = nn.Parameter(torch.from_numpy(weights14a))
squeeze14a.bias = nn.Parameter(torch.from_numpy(bias14a))

squeeze14b=nn.Conv2d(64, 256, kernel_size=3, bias=False, padding=1)
squeeze14b.weight = nn.Parameter(torch.from_numpy(weights14b))
squeeze14b.bias = nn.Parameter(torch.from_numpy(bias14b))

squeeze15=nn.Conv2d(512, 64, kernel_size=1, bias=False)
squeeze15.weight = nn.Parameter(torch.from_numpy(weights15))
squeeze15.bias = nn.Parameter(torch.from_numpy(bias15))

squeeze16a=nn.Conv2d(64, 256, kernel_size=1, bias=False)
squeeze16a.weight = nn.Parameter(torch.from_numpy(weights16a))
squeeze16a.bias = nn.Parameter(torch.from_numpy(bias16a))

squeeze16b=nn.Conv2d(64, 256, kernel_size=3, bias=False, padding=1)
squeeze16b.weight = nn.Parameter(torch.from_numpy(weights16b))
```

```
squeeze16b.bias = nn.Parameter(torch.from_numpy(bias16b))

conv_class=nn.Conv2d(512, 1000, kernel_size=1, bias=False)
conv_class.weight = nn.Parameter(torch.from_numpy(weights17))
conv_class.bias = nn.Parameter(torch.from_numpy(bias17))

avgpool=nn.AvgPool2d(13)

imagen1  = torch.from_numpy(imagen).float()

salida0=squeeze0(imagen1)
salida0_activation=squeeze_activation(salida0)

salida_pool1 = maxpool(salida0_activation)

salida1=squeeze1(salida_pool1)
salida1_activation=squeeze_activation(salida1)

salida2a=squeeze2a(salida1_activation)
salida2a_activation=squeeze_activation(salida2a)
salida2b=squeeze2b(salida1_activation)
salida2b_activation=squeeze_activation(salida2b)
salida2_total=torch.cat([salida2a_activation,salida2b_activation], 1)

salida3=squeeze3(salida2_total)
salida3_activation=squeeze_activation(salida3)

salida4a=squeeze4a(salida3_activation)
salida4a_activation=squeeze_activation(salida4a)
salida4b=squeeze4b(salida3_activation)
salida4b_activation=squeeze_activation(salida4b)
salida4_total=torch.cat([salida4a_activation,salida4b_activation], 1)

salida_pool2 = maxpool(salida4_total)

salida5=squeeze5(salida_pool2)
salida5_activation=squeeze_activation(salida5)

salida6a=squeeze6a(salida5_activation)
salida6a_activation=squeeze_activation(salida6a)
salida6b=squeeze6b(salida5_activation)
salida6b_activation=squeeze_activation(salida6b)
salida6_total=torch.cat([salida6a_activation,salida6b_activation], 1)

salida7=squeeze7(salida6_total)
salida7_activation=squeeze_activation(salida7)
```

```
salida8a=squeeze8a(salida7_activation)
salida8a_activation=squeeze_activation(salida8a)
salida8b=squeeze8b(salida7_activation)
salida8b_activation=squeeze_activation(salida8b)
salida8_total=torch.cat([salida8a_activation,salida8b_activation], 1)

salida_pool3 = maxpool(salida8_total)

salida9=squeeze9(salida_pool3)
salida9_activation=squeeze_activation(salida9)

salida10a=squeeze10a(salida9_activation)
salida10a_activation=squeeze_activation(salida10a)
salida10b=squeeze10b(salida9_activation)
salida10b_activation=squeeze_activation(salida10b)
salida10_total=torch.cat([salida10a_activation,salida10b_activation], 1)

salida11=squeeze11(salida10_total)
salida11_activation=squeeze_activation(salida11)

salida12a=squeeze12a(salida11_activation)
salida12a_activation=squeeze_activation(salida12a)
salida12b=squeeze12b(salida11_activation)
salida12b_activation=squeeze_activation(salida12b)
salida12_total=torch.cat([salida12a_activation,salida12b_activation], 1)

salida13=squeeze13(salida12_total)
salida13_activation=squeeze_activation(salida13)

salida14a=squeeze14a(salida13_activation)
salida14a_activation=squeeze_activation(salida14a)
salida14b=squeeze14b(salida13_activation)
salida14b_activation=squeeze_activation(salida14b)
salida14_total=torch.cat([salida14a_activation,salida14b_activation], 1)

salida15=squeeze15(salida14_total)
salida15_activation=squeeze_activation(salida15)

salida16a=squeeze16a(salida15_activation)
salida16a_activation=squeeze_activation(salida16a)
salida16b=squeeze16b(salida15_activation)
salida16b_activation=squeeze_activation(salida16b)
salida16_total=torch.cat([salida16a_activation,salida16b_activation], 1)

salida17=conv_class(salida16_total)
salida17_activation=squeeze_activation(salida17)
salida18=avgpool(salida17_activation)
```

```
salida18_a_numpy=salida18.detach().numpy()

toc=pc()

acumulado_pytorch=toc-tic+acumulado_pytorch

####### OPENCL COMPARISON #######
```

[15]:
```
## NDRANGE
h_sample = imagen.reshape(-1).astype(np.float32)
d_sample = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=h_sample)

conv1_weight = weights0.reshape(-1)
conv1_bias = bias0

fire1_squeeze_weight = weights1.reshape(-1)
fire1_squeeze_bias = bias1
fire1_expand1x1_weight = weights2a.reshape(-1)
fire1_expand1x1_bias = bias2a
fire1_expand3x3_weight =weights2b.reshape(-1)
fire1_expand3x3_bias = bias2b

fire2_squeeze_weight = weights3.reshape(-1)
fire2_squeeze_bias = bias3
fire2_expand1x1_weight = weights4a.reshape(-1)
fire2_expand1x1_bias = bias4a
fire2_expand3x3_weight =weights4b.reshape(-1)
fire2_expand3x3_bias = bias4b

fire3_squeeze_weight = weights5.reshape(-1)
fire3_squeeze_bias = bias5
fire3_expand1x1_weight = weights6a.reshape(-1)
fire3_expand1x1_bias = bias6a
fire3_expand3x3_weight =weights6b.reshape(-1)
fire3_expand3x3_bias = bias6b

fire4_squeeze_weight = weights7.reshape(-1)
fire4_squeeze_bias = bias7
fire4_expand1x1_weight = weights8a.reshape(-1)
fire4_expand1x1_bias = bias8a
fire4_expand3x3_weight =weights8b.reshape(-1)
fire4_expand3x3_bias = bias8b

fire5_squeeze_weight = weights9.reshape(-1)
fire5_squeeze_bias = bias9
```

```
fire5_expand1x1_weight = weights10a.reshape(-1)
fire5_expand1x1_bias = bias10a
fire5_expand3x3_weight =weights10b.reshape(-1)
fire5_expand3x3_bias = bias10b

fire6_squeeze_weight = weights11.reshape(-1)
fire6_squeeze_bias = bias11
fire6_expand1x1_weight = weights12a.reshape(-1)
fire6_expand1x1_bias = bias12a
fire6_expand3x3_weight =weights12b.reshape(-1)
fire6_expand3x3_bias = bias12b

fire7_squeeze_weight = weights13.reshape(-1)
fire7_squeeze_bias = bias13
fire7_expand1x1_weight = weights14a.reshape(-1)
fire7_expand1x1_bias = bias14a
fire7_expand3x3_weight =weights14b.reshape(-1)
fire7_expand3x3_bias = bias14b

fire8_squeeze_weight = weights15.reshape(-1)
fire8_squeeze_bias = bias15
fire8_expand1x1_weight = weights16a.reshape(-1)
fire8_expand1x1_bias = bias16a
fire8_expand3x3_weight =weights16b.reshape(-1)
fire8_expand3x3_bias = bias16b

classifier_conv_weight = weights17.reshape(-1)
classifier_conv_bias = bias17

h_result_conv = np.empty(1 * 64 * 111 * 111).astype(np.float32)
h_result_pool1 = np.empty(1 * 64 * 55 * 55).astype(np.float32)

h_result_fire1_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
h_result_fire1_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
h_result_fire2_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
h_result_fire2_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
h_result_pool2 = np.empty(1 * 128 * 27 * 27).astype(np.float32)

h_result_fire3_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
h_result_fire3_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
h_result_fire4_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
h_result_fire4_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
h_result_pool3 = np.empty(1 * 256 * 13 * 13).astype(np.float32)

h_result_fire5_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
h_result_fire5_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
h_result_fire6_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
```

```
h_result_fire6_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
h_result_fire7_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
h_result_fire7_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)
h_result_fire8_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
h_result_fire8_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)

h_result_classifier_conv = np.empty(1 * 1000 * 13 * 13).astype(np.float32)
h_result_classifier = np.empty(1 * 1000).astype(np.float32)

d_conv1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=conv1_weight)
d_conv1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=conv1_bias)

d_fire1_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_squeeze_weight)
d_fire1_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire1_squeeze_bias)
d_fire1_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand1x1_weight)
d_fire1_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand1x1_bias)
d_fire1_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_weight)
d_fire1_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_bias)

d_fire2_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_squeeze_weight)
d_fire2_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire2_squeeze_bias)
d_fire2_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand1x1_weight)
d_fire2_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand1x1_bias)
d_fire2_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand3x3_weight)
d_fire2_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand3x3_bias)

d_fire3_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_squeeze_weight)
d_fire3_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire3_squeeze_bias)
d_fire3_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand1x1_weight)
```

```
d_fire3_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand1x1_bias)
d_fire3_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand3x3_weight)
d_fire3_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand3x3_bias)

d_fire4_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_squeeze_weight)
d_fire4_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire4_squeeze_bias)
d_fire4_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand1x1_weight)
d_fire4_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand1x1_bias)
d_fire4_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand3x3_weight)
d_fire4_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand3x3_bias)

d_fire5_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_squeeze_weight)
d_fire5_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire5_squeeze_bias)
d_fire5_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand1x1_weight)
d_fire5_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand1x1_bias)
d_fire5_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand3x3_weight)
d_fire5_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand3x3_bias)

d_fire6_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_squeeze_weight)
d_fire6_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire6_squeeze_bias)
d_fire6_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand1x1_weight)
d_fire6_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand1x1_bias)
d_fire6_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand3x3_weight)
d_fire6_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand3x3_bias)
```

```
d_fire7_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_squeeze_weight)
d_fire7_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire7_squeeze_bias)
d_fire7_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand1x1_weight)
d_fire7_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand1x1_bias)
d_fire7_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand3x3_weight)
d_fire7_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand3x3_bias)

d_fire8_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_squeeze_weight)
d_fire8_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire8_squeeze_bias)
d_fire8_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand1x1_weight)
d_fire8_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand1x1_bias)
d_fire8_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand3x3_weight)
d_fire8_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand3x3_bias)

d_classifier_conv_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=classifier_conv_weight)
d_classifier_conv_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=classifier_conv_bias)

d_result_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.nbytes)
d_result_pool1 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
 ↪nbytes)

d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire1_squeeze.nbytes)
d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire1_expand.nbytes)
d_result_fire2_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire2_squeeze.nbytes)
d_result_fire2_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire2_expand.nbytes)
d_result_pool2 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool2.
 ↪nbytes)
```

```
d_result_fire3_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire3_squeeze.nbytes)
d_result_fire3_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire3_expand.nbytes)
d_result_fire4_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire4_squeeze.nbytes)
d_result_fire4_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire4_expand.nbytes)
d_result_pool3 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool3.
 ↪nbytes)

d_result_fire5_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire5_squeeze.nbytes)
d_result_fire5_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire5_expand.nbytes)
d_result_fire6_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire6_squeeze.nbytes)
d_result_fire6_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire6_expand.nbytes)
d_result_fire7_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire7_squeeze.nbytes)
d_result_fire7_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire7_expand.nbytes)
d_result_fire8_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire8_squeeze.nbytes)
d_result_fire8_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire8_expand.nbytes)

d_result_classifier_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_classifier_conv.nbytes)
d_result_classifier = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_classifier.nbytes)

tic2 = pc()

#first conv layer
conv3x3_NDR(queue,(64, 111), None, 3, 224, 0, 2, 0, 111, d_sample,␣
 ↪d_conv1_weight, d_conv1_bias, d_result_conv)
maxpool_NDR(queue, (64, ), None, 111, 55, d_result_conv, d_result_pool1)

#block1
conv1x1_NDR(queue,(16, 55), None, np.int32(64/4), 55, d_result_pool1,␣
 ↪d_fire1_squeeze_weight, d_fire1_squeeze_bias, d_result_fire1_squeeze)
queue.finish()
```

```
conv1x1_NDR(queue1,(64, 55), None, np.int32(16/4), 55, d_result_fire1_squeeze,␣
 ↪d_fire1_expand1x1_weight, d_fire1_expand1x1_bias, d_result_fire1_expand)
conv3x3_NDR(queue,(64, 55), None, 16, 55, 1, 1, 64, 55, d_result_fire1_squeeze,␣
 ↪d_fire1_expand3x3_weight, d_fire1_expand3x3_bias, d_result_fire1_expand)
queue.finish()
queue1.finish()

conv1x1_NDR(queue,(16, 55), None, np.int32(128/4), 55, d_result_fire1_expand,␣
 ↪d_fire2_squeeze_weight, d_fire2_squeeze_bias, d_result_fire2_squeeze)
queue.finish()
conv1x1_NDR(queue1,(64, 55), None, np.int32(16/4), 55, d_result_fire2_squeeze,␣
 ↪d_fire2_expand1x1_weight, d_fire2_expand1x1_bias, d_result_fire2_expand)
conv3x3_NDR(queue,(64, 55), None, 16, 55, 1, 1, 64, 55, d_result_fire2_squeeze,␣
 ↪d_fire2_expand3x3_weight, d_fire2_expand3x3_bias, d_result_fire2_expand)
queue.finish()
queue1.finish()

maxpool_NDR(queue, (128, ), None, 55, 27, d_result_fire2_expand, d_result_pool2)

#block2
conv1x1_NDR(queue,(32, 27), None, np.int32(128/4), 27, d_result_pool2,␣
 ↪d_fire3_squeeze_weight, d_fire3_squeeze_bias, d_result_fire3_squeeze)
queue.finish()
conv1x1_NDR(queue1,(128, 27), None, np.int32(32/4), 27, d_result_fire3_squeeze,␣
 ↪d_fire3_expand1x1_weight, d_fire3_expand1x1_bias, d_result_fire3_expand)
conv3x3_NDR(queue,(128, 27), None, 32, 27, 1, 1, 128, 27,␣
 ↪d_result_fire3_squeeze, d_fire3_expand3x3_weight, d_fire3_expand3x3_bias,␣
 ↪d_result_fire3_expand)
queue.finish()
queue1.finish()

conv1x1_NDR(queue,(32, 27), None, np.int32(256/4), 27, d_result_fire3_expand,␣
 ↪d_fire4_squeeze_weight, d_fire4_squeeze_bias, d_result_fire4_squeeze)
queue.finish()
conv1x1_NDR(queue1,(128, 27), None, np.int32(32/4), 27, d_result_fire4_squeeze,␣
 ↪d_fire4_expand1x1_weight, d_fire4_expand1x1_bias, d_result_fire4_expand)
conv3x3_NDR(queue,(128, 27), None, 32, 27, 1, 1, 128, 27,␣
 ↪d_result_fire4_squeeze, d_fire4_expand3x3_weight, d_fire4_expand3x3_bias,␣
 ↪d_result_fire4_expand)
queue.finish()
queue1.finish()

maxpool_NDR(queue, (256, ), None, 27, 13, d_result_fire4_expand, d_result_pool3)

#block3
```

```
conv1x1_NDR(queue,(48, 13), None, np.int32(256/4), 13, d_result_pool3,␣
 ↪d_fire5_squeeze_weight, d_fire5_squeeze_bias, d_result_fire5_squeeze)
queue.finish()
conv1x1_NDR(queue1,(192, 13), None, np.int32(48/4), 13, d_result_fire5_squeeze,␣
 ↪d_fire5_expand1x1_weight, d_fire5_expand1x1_bias, d_result_fire5_expand)
conv3x3_NDR(queue,(192, 13), None, 48, 13, 1, 1, 192, 13,␣
 ↪d_result_fire5_squeeze, d_fire5_expand3x3_weight, d_fire5_expand3x3_bias,␣
 ↪d_result_fire5_expand)
queue.finish()
queue1.finish()

conv1x1_NDR(queue,(48, 13), None, np.int32(384/4), 13, d_result_fire5_expand,␣
 ↪d_fire6_squeeze_weight, d_fire6_squeeze_bias, d_result_fire6_squeeze)
queue.finish()
conv1x1_NDR(queue1,(192, 13), None, np.int32(48/4), 13, d_result_fire6_squeeze,␣
 ↪d_fire6_expand1x1_weight, d_fire6_expand1x1_bias, d_result_fire6_expand)
conv3x3_NDR(queue,(192, 13), None, 48, 13, 1, 1, 192, 13,␣
 ↪d_result_fire6_squeeze, d_fire6_expand3x3_weight, d_fire6_expand3x3_bias,␣
 ↪d_result_fire6_expand)
queue.finish()
queue1.finish()

conv1x1_NDR(queue,(64, 13), None, np.int32(384/4), 13, d_result_fire6_expand,␣
 ↪d_fire7_squeeze_weight, d_fire7_squeeze_bias, d_result_fire7_squeeze)
queue.finish()
conv1x1_NDR(queue1,(256, 13), None, np.int32(64/4), 13, d_result_fire7_squeeze,␣
 ↪d_fire7_expand1x1_weight, d_fire7_expand1x1_bias, d_result_fire7_expand)
conv3x3_NDR(queue,(256, 13), None, 64, 13, 1, 1, 256, 13,␣
 ↪d_result_fire7_squeeze, d_fire7_expand3x3_weight, d_fire7_expand3x3_bias,␣
 ↪d_result_fire7_expand)
queue.finish()
queue1.finish()

conv1x1_NDR(queue,(64, 13), None, np.int32(512/4), 13, d_result_fire7_expand,␣
 ↪d_fire8_squeeze_weight, d_fire8_squeeze_bias, d_result_fire8_squeeze)
queue.finish()
conv1x1_NDR(queue1,(256, 13), None, np.int32(64/4), 13, d_result_fire8_squeeze,␣
 ↪d_fire8_expand1x1_weight, d_fire8_expand1x1_bias, d_result_fire8_expand)
conv3x3_NDR(queue,(256, 13), None, 64, 13, 1, 1, 256, 13,␣
 ↪d_result_fire8_squeeze, d_fire8_expand3x3_weight, d_fire8_expand3x3_bias,␣
 ↪d_result_fire8_expand)
queue.finish()
queue1.finish()

# classifier
```

```
conv1x1_NDR(queue,(1000, 13), None, np.int32(512/4), 13, d_result_fire8_expand,␣
 ↪d_classifier_conv_weight, d_classifier_conv_bias, d_result_classifier_conv)

avgpool_NDR(queue,(1000, ), None, d_result_classifier_conv, d_result_classifier)

cl.enqueue_copy(queue, h_result_classifier, d_result_classifier)

queue.finish()

veamos = h_result_classifier

rtime = pc() - tic2
```

```
Reprogramming device [0] with handle 118
```

[17]:
```
# Simple task
h_sample = imagen.reshape(-1).astype(np.float32)
d_sample = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=h_sample)

conv1_weight = weights0.reshape(-1)
conv1_bias = bias0

fire1_squeeze_weight = weights1.reshape(-1)
fire1_squeeze_bias = bias1
fire1_expand1x1_weight = weights2a.reshape(-1)
fire1_expand1x1_bias = bias2a
fire1_expand3x3_weight =weights2b.reshape(-1)
fire1_expand3x3_bias = bias2b

fire2_squeeze_weight = weights3.reshape(-1)
fire2_squeeze_bias = bias3
fire2_expand1x1_weight = weights4a.reshape(-1)
fire2_expand1x1_bias = bias4a
fire2_expand3x3_weight =weights4b.reshape(-1)
fire2_expand3x3_bias = bias4b

fire3_squeeze_weight = weights5.reshape(-1)
fire3_squeeze_bias = bias5
fire3_expand1x1_weight = weights6a.reshape(-1)
fire3_expand1x1_bias = bias6a
fire3_expand3x3_weight =weights6b.reshape(-1)
fire3_expand3x3_bias = bias6b

fire4_squeeze_weight = weights7.reshape(-1)
fire4_squeeze_bias = bias7
fire4_expand1x1_weight = weights8a.reshape(-1)
```

```
fire4_expand1x1_bias = bias8a
fire4_expand3x3_weight =weights8b.reshape(-1)
fire4_expand3x3_bias = bias8b

fire5_squeeze_weight = weights9.reshape(-1)
fire5_squeeze_bias = bias9
fire5_expand1x1_weight = weights10a.reshape(-1)
fire5_expand1x1_bias = bias10a
fire5_expand3x3_weight =weights10b.reshape(-1)
fire5_expand3x3_bias = bias10b

fire6_squeeze_weight = weights11.reshape(-1)
fire6_squeeze_bias = bias11
fire6_expand1x1_weight = weights12a.reshape(-1)
fire6_expand1x1_bias = bias12a
fire6_expand3x3_weight =weights12b.reshape(-1)
fire6_expand3x3_bias = bias12b

fire7_squeeze_weight = weights13.reshape(-1)
fire7_squeeze_bias = bias13
fire7_expand1x1_weight = weights14a.reshape(-1)
fire7_expand1x1_bias = bias14a
fire7_expand3x3_weight =weights14b.reshape(-1)
fire7_expand3x3_bias = bias14b

fire8_squeeze_weight = weights15.reshape(-1)
fire8_squeeze_bias = bias15
fire8_expand1x1_weight = weights16a.reshape(-1)
fire8_expand1x1_bias = bias16a
fire8_expand3x3_weight =weights16b.reshape(-1)
fire8_expand3x3_bias = bias16b

classifier_conv_weight = weights17.reshape(-1)
classifier_conv_bias = bias17

h_result_conv = np.empty(1 * 64 * 111 * 111).astype(np.float32)
h_result_pool1 = np.empty(1 * 64 * 55 * 55).astype(np.float32)

h_result_fire1_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
h_result_fire1_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
h_result_fire2_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
h_result_fire2_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
h_result_pool2 = np.empty(1 * 128 * 27 * 27).astype(np.float32)

h_result_fire3_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
h_result_fire3_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
h_result_fire4_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
```

```
h_result_fire4_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
h_result_pool3 = np.empty(1 * 256 * 13 * 13).astype(np.float32)

h_result_fire5_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
h_result_fire5_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
h_result_fire6_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
h_result_fire6_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
h_result_fire7_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
h_result_fire7_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)
h_result_fire8_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
h_result_fire8_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)

h_result_classifier_conv = np.empty(1 * 1000 * 13 * 13).astype(np.float32)
h_result_classifier = np.empty(1 * 1000).astype(np.float32)

d_conv1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=conv1_weight)
d_conv1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=conv1_bias)

d_fire1_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_squeeze_weight)
d_fire1_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire1_squeeze_bias)
d_fire1_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand1x1_weight)
d_fire1_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand1x1_bias)
d_fire1_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_weight)
d_fire1_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_bias)

d_fire2_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_squeeze_weight)
d_fire2_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire2_squeeze_bias)
d_fire2_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand1x1_weight)
d_fire2_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand1x1_bias)
d_fire2_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand3x3_weight)
d_fire2_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand3x3_bias)
```

```
d_fire3_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_squeeze_weight)
d_fire3_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire3_squeeze_bias)
d_fire3_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand1x1_weight)
d_fire3_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand1x1_bias)
d_fire3_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand3x3_weight)
d_fire3_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand3x3_bias)

d_fire4_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_squeeze_weight)
d_fire4_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire4_squeeze_bias)
d_fire4_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand1x1_weight)
d_fire4_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand1x1_bias)
d_fire4_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand3x3_weight)
d_fire4_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand3x3_bias)

d_fire5_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_squeeze_weight)
d_fire5_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire5_squeeze_bias)
d_fire5_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand1x1_weight)
d_fire5_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand1x1_bias)
d_fire5_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand3x3_weight)
d_fire5_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand3x3_bias)

d_fire6_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_squeeze_weight)
d_fire6_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire6_squeeze_bias)
d_fire6_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand1x1_weight)
```

```
d_fire6_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand1x1_bias)
d_fire6_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand3x3_weight)
d_fire6_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand3x3_bias)

d_fire7_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_squeeze_weight)
d_fire7_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire7_squeeze_bias)
d_fire7_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand1x1_weight)
d_fire7_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand1x1_bias)
d_fire7_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand3x3_weight)
d_fire7_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand3x3_bias)

d_fire8_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_squeeze_weight)
d_fire8_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire8_squeeze_bias)
d_fire8_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand1x1_weight)
d_fire8_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand1x1_bias)
d_fire8_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand3x3_weight)
d_fire8_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand3x3_bias)

d_classifier_conv_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=classifier_conv_weight)
d_classifier_conv_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=classifier_conv_bias)

d_result_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.nbytes)
d_result_pool1 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
 ↪nbytes)

d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire1_squeeze.nbytes)
d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire1_expand.nbytes)
```

```
d_result_fire2_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire2_squeeze.nbytes)
d_result_fire2_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire2_expand.nbytes)
d_result_pool2 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool2.
 ↪nbytes)

d_result_fire3_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire3_squeeze.nbytes)
d_result_fire3_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire3_expand.nbytes)
d_result_fire4_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire4_squeeze.nbytes)
d_result_fire4_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire4_expand.nbytes)
d_result_pool3 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool3.
 ↪nbytes)

d_result_fire5_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire5_squeeze.nbytes)
d_result_fire5_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire5_expand.nbytes)
d_result_fire6_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire6_squeeze.nbytes)
d_result_fire6_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire6_expand.nbytes)
d_result_fire7_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire7_squeeze.nbytes)
d_result_fire7_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire7_expand.nbytes)
d_result_fire8_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire8_squeeze.nbytes)
d_result_fire8_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_fire8_expand.nbytes)

d_result_classifier_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_classifier_conv.nbytes)
d_result_classifier = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
 ↪h_result_classifier.nbytes)
```

[18]:
```
tic3 = pc()

#first conv layer
conv3x3_ST(queue,(1,), None, 3, 224, 0, 2, 0, 111, 64, d_sample, d_conv1_weight,␣
 ↪d_conv1_bias, d_result_conv)
maxpool_ST(queue, (1, ), None, 111, 55, 64, d_result_conv, d_result_pool1)
```

```
#block1
conv1x1_ST(queue,(1,), None, 64, 55, 16, d_result_pool1, d_fire1_squeeze_weight,␣
 ↪d_fire1_squeeze_bias, d_result_fire1_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, 16, 55, 64, d_result_fire1_squeeze,␣
 ↪d_fire1_expand1x1_weight, d_fire1_expand1x1_bias, d_result_fire1_expand)
conv3x3_ST(queue,(1,), None, 16, 55, 1, 1, 64, 55, 64, d_result_fire1_squeeze,␣
 ↪d_fire1_expand3x3_weight, d_fire1_expand3x3_bias, d_result_fire1_expand)
queue.finish()
queue1.finish()

conv1x1_ST(queue,(1,), None, 128, 55, 16, d_result_fire1_expand,␣
 ↪d_fire2_squeeze_weight, d_fire2_squeeze_bias, d_result_fire2_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, 16, 55, 64, d_result_fire2_squeeze,␣
 ↪d_fire2_expand1x1_weight, d_fire2_expand1x1_bias, d_result_fire2_expand)
conv3x3_ST(queue,(1,), None, 16, 55, 1, 1, 64, 55, 64, d_result_fire2_squeeze,␣
 ↪d_fire2_expand3x3_weight, d_fire2_expand3x3_bias, d_result_fire2_expand)
queue.finish()
queue1.finish()

maxpool_ST(queue, (1, ), None, 55, 27, 128, d_result_fire2_expand,␣
 ↪d_result_pool2)

#block2
conv1x1_ST(queue,(1,), None, 128, 27, 32, d_result_pool2,␣
 ↪d_fire3_squeeze_weight, d_fire3_squeeze_bias, d_result_fire3_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, 32, 27, 128, d_result_fire3_squeeze,␣
 ↪d_fire3_expand1x1_weight, d_fire3_expand1x1_bias, d_result_fire3_expand)
conv3x3_ST(queue,(1,), None, 32, 27, 1, 1, 128, 27, 128, d_result_fire3_squeeze,␣
 ↪d_fire3_expand3x3_weight, d_fire3_expand3x3_bias, d_result_fire3_expand)
queue.finish()
queue1.finish()

conv1x1_ST(queue,(1,), None, 256, 27, 32, d_result_fire3_expand,␣
 ↪d_fire4_squeeze_weight, d_fire4_squeeze_bias, d_result_fire4_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, 32, 27, 128, d_result_fire4_squeeze,␣
 ↪d_fire4_expand1x1_weight, d_fire4_expand1x1_bias, d_result_fire4_expand)
conv3x3_ST(queue,(1,), None, 32, 27, 1, 1, 128, 27, 128, d_result_fire4_squeeze,␣
 ↪d_fire4_expand3x3_weight, d_fire4_expand3x3_bias, d_result_fire4_expand)
queue.finish()
queue1.finish()
```

```
maxpool_ST(queue, (1, ), None, 27, 13, 256, d_result_fire4_expand,␣
 ↪d_result_pool3)

#block3
conv1x1_ST(queue,(1,), None, 256, 13, 48, d_result_pool3,␣
 ↪d_fire5_squeeze_weight, d_fire5_squeeze_bias, d_result_fire5_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, 48, 13, 192, d_result_fire5_squeeze,␣
 ↪d_fire5_expand1x1_weight, d_fire5_expand1x1_bias, d_result_fire5_expand)
conv3x3_ST(queue,(1,), None, 48, 13, 1, 1, 192, 13, 192, d_result_fire5_squeeze,␣
 ↪d_fire5_expand3x3_weight, d_fire5_expand3x3_bias, d_result_fire5_expand)
queue.finish()
queue1.finish()

conv1x1_ST(queue,(1,), None, 384, 13, 48, d_result_fire5_expand,␣
 ↪d_fire6_squeeze_weight, d_fire6_squeeze_bias, d_result_fire6_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, 48, 13, 192, d_result_fire6_squeeze,␣
 ↪d_fire6_expand1x1_weight, d_fire6_expand1x1_bias, d_result_fire6_expand)
conv3x3_ST(queue,(1,), None, 48, 13, 1, 1, 192, 13, 192, d_result_fire6_squeeze,␣
 ↪d_fire6_expand3x3_weight, d_fire6_expand3x3_bias, d_result_fire6_expand)
queue.finish()
queue1.finish()

conv1x1_ST(queue,(1,), None, 384, 13, 64, d_result_fire6_expand,␣
 ↪d_fire7_squeeze_weight, d_fire7_squeeze_bias, d_result_fire7_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, 64, 13, 256, d_result_fire7_squeeze,␣
 ↪d_fire7_expand1x1_weight, d_fire7_expand1x1_bias, d_result_fire7_expand)
conv3x3_ST(queue,(1,), None, 64, 13, 1, 1, 256, 13, 256, d_result_fire7_squeeze,␣
 ↪d_fire7_expand3x3_weight, d_fire7_expand3x3_bias, d_result_fire7_expand)
queue.finish()
queue1.finish()

conv1x1_ST(queue,(1,), None, 512, 13, 64, d_result_fire7_expand,␣
 ↪d_fire8_squeeze_weight, d_fire8_squeeze_bias, d_result_fire8_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, 64, 13, 256, d_result_fire8_squeeze,␣
 ↪d_fire8_expand1x1_weight, d_fire8_expand1x1_bias, d_result_fire8_expand)
conv3x3_ST(queue,(1,), None, 64, 13, 1, 1, 256, 13, 256, d_result_fire8_squeeze,␣
 ↪d_fire8_expand3x3_weight, d_fire8_expand3x3_bias, d_result_fire8_expand)
queue.finish()
queue1.finish()

# classifier
```

```
conv1x1_ST(queue,(1,), None, 512, 13, 1000, d_result_fire8_expand,␣
 ↪d_classifier_conv_weight, d_classifier_conv_bias, d_result_classifier_conv)

avgpool_ST(queue, (1, ), None, d_result_classifier_conv, d_result_classifier)

cl.enqueue_copy(queue, h_result_classifier, d_result_classifier)

veamos1 = h_result_classifier

rtime1 = pc() - tic3
```

```
Reprogramming device [0] with handle 119
```

[21]:
```
tic3 = pc()

#first conv layer
conv3x3_ST(queue,(1,), None, 3, 224, 0, 2, 0, 111, 64, d_sample, d_conv1_weight,␣
 ↪d_conv1_bias, d_result_conv)
maxpool_ST(queue, (1, ), None, 111, 55, 64, d_result_conv, d_result_pool1)

#block1
conv1x1_ST(queue,(1,), None, np.int32(64/4), 55, 16, d_result_pool1,␣
 ↪d_fire1_squeeze_weight, d_fire1_squeeze_bias, d_result_fire1_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, np.int32(16/4), 55, 64, d_result_fire1_squeeze,␣
 ↪d_fire1_expand1x1_weight, d_fire1_expand1x1_bias, d_result_fire1_expand)
conv3x3_ST(queue,(1,), None, 16, 55, 1, 1, 64, 55, 64, d_result_fire1_squeeze,␣
 ↪d_fire1_expand3x3_weight, d_fire1_expand3x3_bias, d_result_fire1_expand)
queue.finish()
queue1.finish()

conv1x1_ST(queue,(1,), None, np.int32(128/4), 55, 16, d_result_fire1_expand,␣
 ↪d_fire2_squeeze_weight, d_fire2_squeeze_bias, d_result_fire2_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, np.int32(16/4), 55, 64, d_result_fire2_squeeze,␣
 ↪d_fire2_expand1x1_weight, d_fire2_expand1x1_bias, d_result_fire2_expand)
conv3x3_ST(queue,(1,), None, 16, 55, 1, 1, 64, 55, 64, d_result_fire2_squeeze,␣
 ↪d_fire2_expand3x3_weight, d_fire2_expand3x3_bias, d_result_fire2_expand)
queue.finish()
queue1.finish()

maxpool_ST(queue, (1, ), None, 55, 27, 128, d_result_fire2_expand,␣
 ↪d_result_pool2)

#block2
conv1x1_ST(queue,(1,), None, np.int32(128/4), 27, 32, d_result_pool2,␣
 ↪d_fire3_squeeze_weight, d_fire3_squeeze_bias, d_result_fire3_squeeze)
```

```
queue.finish()
conv1x1_ST(queue1,(1,), None, np.int32(32/4), 27, 128, d_result_fire3_squeeze,␣
 ↪d_fire3_expand1x1_weight, d_fire3_expand1x1_bias, d_result_fire3_expand)
conv3x3_ST(queue,(1,), None, 32, 27, 1, 1, 128, 27, 128, d_result_fire3_squeeze,␣
 ↪d_fire3_expand3x3_weight, d_fire3_expand3x3_bias, d_result_fire3_expand)
queue.finish()
queue1.finish()

conv1x1_ST(queue,(1,), None, np.int32(256/4), 27, 32, d_result_fire3_expand,␣
 ↪d_fire4_squeeze_weight, d_fire4_squeeze_bias, d_result_fire4_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, np.int32(32/4), 27, 128, d_result_fire4_squeeze,␣
 ↪d_fire4_expand1x1_weight, d_fire4_expand1x1_bias, d_result_fire4_expand)
conv3x3_ST(queue,(1,), None, 32, 27, 1, 1, 128, 27, 128, d_result_fire4_squeeze,␣
 ↪d_fire4_expand3x3_weight, d_fire4_expand3x3_bias, d_result_fire4_expand)
queue.finish()
queue1.finish()

maxpool_ST(queue, (1, ), None, 27, 13, 256, d_result_fire4_expand,␣
 ↪d_result_pool3)

#block3
conv1x1_ST(queue,(1,), None, np.int32(256/4), 13, 48, d_result_pool3,␣
 ↪d_fire5_squeeze_weight, d_fire5_squeeze_bias, d_result_fire5_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, np.int32(48/4), 13, 192, d_result_fire5_squeeze,␣
 ↪d_fire5_expand1x1_weight, d_fire5_expand1x1_bias, d_result_fire5_expand)
conv3x3_ST(queue,(1,), None, 48, 13, 1, 1, 192, 13, 192, d_result_fire5_squeeze,␣
 ↪d_fire5_expand3x3_weight, d_fire5_expand3x3_bias, d_result_fire5_expand)
queue.finish()
queue1.finish()

conv1x1_ST(queue,(1,), None, np.int32(384/4), 13, 48, d_result_fire5_expand,␣
 ↪d_fire6_squeeze_weight, d_fire6_squeeze_bias, d_result_fire6_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, np.int32(48/4), 13, 192, d_result_fire6_squeeze,␣
 ↪d_fire6_expand1x1_weight, d_fire6_expand1x1_bias, d_result_fire6_expand)
conv3x3_ST(queue,(1,), None, 48, 13, 1, 1, 192, 13, 192, d_result_fire6_squeeze,␣
 ↪d_fire6_expand3x3_weight, d_fire6_expand3x3_bias, d_result_fire6_expand)
queue.finish()
queue1.finish()

conv1x1_ST(queue,(1,), None, np.int32(384/4), 13, 64, d_result_fire6_expand,␣
 ↪d_fire7_squeeze_weight, d_fire7_squeeze_bias, d_result_fire7_squeeze)
queue.finish()
```

```
conv1x1_ST(queue1,(1,), None, np.int32(64/4), 13, 256, d_result_fire7_squeeze,␣
 ↪d_fire7_expand1x1_weight, d_fire7_expand1x1_bias, d_result_fire7_expand)
conv3x3_ST(queue,(1,), None, 64, 13, 1, 1, 256, 13, 256, d_result_fire7_squeeze,␣
 ↪d_fire7_expand3x3_weight, d_fire7_expand3x3_bias, d_result_fire7_expand)
queue.finish()
queue1.finish()

conv1x1_ST(queue,(1,), None, np.int32(512/4), 13, 64, d_result_fire7_expand,␣
 ↪d_fire8_squeeze_weight, d_fire8_squeeze_bias, d_result_fire8_squeeze)
queue.finish()
conv1x1_ST(queue1,(1,), None, np.int32(64/4), 13, 256, d_result_fire8_squeeze,␣
 ↪d_fire8_expand1x1_weight, d_fire8_expand1x1_bias, d_result_fire8_expand)
conv3x3_ST(queue,(1,), None, 64, 13, 1, 1, 256, 13, 256, d_result_fire8_squeeze,␣
 ↪d_fire8_expand3x3_weight, d_fire8_expand3x3_bias, d_result_fire8_expand)
queue.finish()
queue1.finish()

# classifier
conv1x1_ST(queue,(1,), None, np.int32(512/4), 13, 1000, d_result_fire8_expand,␣
 ↪d_classifier_conv_weight, d_classifier_conv_bias, d_result_classifier_conv)

avgpool_ST(queue, (1, ), None, d_result_classifier_conv, d_result_classifier)

cl.enqueue_copy(queue, h_result_classifier, d_result_classifier)

veamos1 = h_result_classifier

rtime1 = pc() - tic3
```

[22]:
```
print ("tiempo en segundos con pytorch= ", toc-tic)
print ("tiempo en segundos con opencl (NDRANGE)=",rtime)
print ("tiempo en segundos con opencl (Simple Task)=",rtime1)

comparativa1=np.allclose(salida18_a_numpy.reshape(-1), veamos,rtol=1e-01,␣
 ↪atol=1e-01)
comparativa2=np.allclose(salida18_a_numpy.reshape(-1), veamos1,rtol=1e-01,␣
 ↪atol=1e-01)
comparativa3=np.allclose(veamos, veamos1,rtol=1e-01, atol=1e-01)

print("comparativa (pytorch == NDRange): ",comparativa1)
print("comparativa (pytorch ==", file_dir[:-5], "): ",comparativa2)
print("comparativa (NDRange ==", file_dir[:-5], "): ",comparativa3)
```

```
tiempo en segundos con pytorch=  1.1246508879994508
tiempo en segundos con opencl (NDRANGE)= 1.6290928150001491
tiempo en segundos con opencl (Simple Task)= 3.2147812559996964
comparativa (pytorch == NDRange):  True
```

```
comparativa (pytorch == squeezenet_ST_V6 ):  False
comparativa (NDRange == squeezenet_ST_V6 ):  False
```

Obtención de resultados sin ST Float4

[24]:
```
squeeze_activation = nn.ReLU(inplace=True)

pytorch_time = 0.0
NDRange_time = 0.0
Our_NDRange_time = 0.0
ST_time = 0.0

COUNT = 50

comparativa1 = True
comparativa2 = True
comparativa3 = True
comparativa4 = True

#####################################################################

transform = transforms.Compose([
transforms.Resize(256),
transforms.CenterCrop(224),
transforms.ToTensor(),
transforms.Normalize(mean = [ 0.485, 0.456, 0.406 ],
                     std = [ 0.229, 0.224, 0.225 ]),])

params = torch.load('squeezenet1_1.pth')

### First Conv3x3 and maxpool
weights0=params['features.0.weight'].numpy()
bias0=params['features.0.bias'].numpy()

######## BLOCK 1 ########
#fire - fire - maxpool
#### FIRE 1 ####
weights1=params['features.3.squeeze.weight'].numpy()
bias1=params['features.3.squeeze.bias'].numpy()

weights2a=params['features.3.expand1x1.weight'].numpy()
bias2a=params['features.3.expand1x1.bias'].numpy()

weights2b=params['features.3.expand3x3.weight'].numpy()
bias2b=params['features.3.expand3x3.bias'].numpy()

#### FIRE 2 ####
weights3=params['features.4.squeeze.weight'].numpy()
```

```
bias3=params['features.4.squeeze.bias'].numpy()

weights4a=params['features.4.expand1x1.weight'].numpy()
bias4a=params['features.4.expand1x1.bias'].numpy()

weights4b=params['features.4.expand3x3.weight'].numpy()
bias4b=params['features.4.expand3x3.bias'].numpy()

######## BLOCK 2 ########
#fire - fire - maxpool
#### FIRE 3 ####
weights5=params['features.6.squeeze.weight'].numpy()
bias5=params['features.6.squeeze.bias'].numpy()

weights6a=params['features.6.expand1x1.weight'].numpy()
bias6a=params['features.6.expand1x1.bias'].numpy()

weights6b=params['features.6.expand3x3.weight'].numpy()
bias6b=params['features.6.expand3x3.bias'].numpy()

#### FIRE 4 ####
weights7=params['features.7.squeeze.weight'].numpy()
bias7=params['features.7.squeeze.bias'].numpy()

weights8a=params['features.7.expand1x1.weight'].numpy()
bias8a=params['features.7.expand1x1.bias'].numpy()

weights8b=params['features.7.expand3x3.weight'].numpy()
bias8b=params['features.7.expand3x3.bias'].numpy()

######## BLOCK 3 ########
#fire - fire - fire - fire
#### FIRE 5 ####
weights9=params['features.9.squeeze.weight'].numpy()
bias9=params['features.9.squeeze.bias'].numpy()

weights10a=params['features.9.expand1x1.weight'].numpy()
bias10a=params['features.9.expand1x1.bias'].numpy()

weights10b=params['features.9.expand3x3.weight'].numpy()
bias10b=params['features.9.expand3x3.bias'].numpy()

#### FIRE 6 ####
weights11=params['features.10.squeeze.weight'].numpy()
bias11=params['features.10.squeeze.bias'].numpy()

weights12a=params['features.10.expand1x1.weight'].numpy()
```

```
bias12a=params['features.10.expand1x1.bias'].numpy()

weights12b=params['features.10.expand3x3.weight'].numpy()
bias12b=params['features.10.expand3x3.bias'].numpy()

#### FIRE 7 ####
weights13=params['features.11.squeeze.weight'].numpy()
bias13=params['features.11.squeeze.bias'].numpy()

weights14a=params['features.11.expand1x1.weight'].numpy()
bias14a=params['features.11.expand1x1.bias'].numpy()

weights14b=params['features.11.expand3x3.weight'].numpy()
bias14b=params['features.11.expand3x3.bias'].numpy()

#### FIRE 8 ####
weights15=params['features.12.squeeze.weight'].numpy()
bias15=params['features.12.squeeze.bias'].numpy()

weights16a=params['features.12.expand1x1.weight'].numpy()
bias16a=params['features.12.expand1x1.bias'].numpy()

weights16b=params['features.12.expand3x3.weight'].numpy()
bias16b=params['features.12.expand3x3.bias'].numpy()

######## Classifier ########
#conv3x3 - avgpool
### Classifier Conv3x3 and avgpool
weights17=params['classifier.1.weight'].numpy()
bias17=params['classifier.1.bias'].numpy()

##### Parameters for OpenCL

conv1_weight = weights0.reshape(-1)
conv1_bias = bias0

fire1_squeeze_weight = weights1.reshape(-1)
fire1_squeeze_bias = bias1
fire1_expand1x1_weight = weights2a.reshape(-1)
fire1_expand1x1_bias = bias2a
fire1_expand3x3_weight =weights2b.reshape(-1)
fire1_expand3x3_bias = bias2b

fire2_squeeze_weight = weights3.reshape(-1)
fire2_squeeze_bias = bias3
fire2_expand1x1_weight = weights4a.reshape(-1)
fire2_expand1x1_bias = bias4a
```

```
fire2_expand3x3_weight =weights4b.reshape(-1)
fire2_expand3x3_bias = bias4b

fire3_squeeze_weight = weights5.reshape(-1)
fire3_squeeze_bias = bias5
fire3_expand1x1_weight = weights6a.reshape(-1)
fire3_expand1x1_bias = bias6a
fire3_expand3x3_weight =weights6b.reshape(-1)
fire3_expand3x3_bias = bias6b

fire4_squeeze_weight = weights7.reshape(-1)
fire4_squeeze_bias = bias7
fire4_expand1x1_weight = weights8a.reshape(-1)
fire4_expand1x1_bias = bias8a
fire4_expand3x3_weight =weights8b.reshape(-1)
fire4_expand3x3_bias = bias8b

fire5_squeeze_weight = weights9.reshape(-1)
fire5_squeeze_bias = bias9
fire5_expand1x1_weight = weights10a.reshape(-1)
fire5_expand1x1_bias = bias10a
fire5_expand3x3_weight =weights10b.reshape(-1)
fire5_expand3x3_bias = bias10b

fire6_squeeze_weight = weights11.reshape(-1)
fire6_squeeze_bias = bias11
fire6_expand1x1_weight = weights12a.reshape(-1)
fire6_expand1x1_bias = bias12a
fire6_expand3x3_weight =weights12b.reshape(-1)
fire6_expand3x3_bias = bias12b

fire7_squeeze_weight = weights13.reshape(-1)
fire7_squeeze_bias = bias13
fire7_expand1x1_weight = weights14a.reshape(-1)
fire7_expand1x1_bias = bias14a
fire7_expand3x3_weight =weights14b.reshape(-1)
fire7_expand3x3_bias = bias14b

fire8_squeeze_weight = weights15.reshape(-1)
fire8_squeeze_bias = bias15
fire8_expand1x1_weight = weights16a.reshape(-1)
fire8_expand1x1_bias = bias16a
fire8_expand3x3_weight =weights16b.reshape(-1)
fire8_expand3x3_bias = bias16b

classifier_conv_weight = weights17.reshape(-1)
classifier_conv_bias = bias17
```

```
d_conv1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=conv1_weight)
d_conv1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=conv1_bias)

d_fire1_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_squeeze_weight)
d_fire1_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire1_squeeze_bias)
d_fire1_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand1x1_weight)
d_fire1_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand1x1_bias)
d_fire1_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_weight)
d_fire1_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_bias)

d_fire2_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_squeeze_weight)
d_fire2_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire2_squeeze_bias)
d_fire2_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand1x1_weight)
d_fire2_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand1x1_bias)
d_fire2_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand3x3_weight)
d_fire2_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand3x3_bias)

d_fire3_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_squeeze_weight)
d_fire3_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire3_squeeze_bias)
d_fire3_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand1x1_weight)
d_fire3_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand1x1_bias)
d_fire3_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand3x3_weight)
d_fire3_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand3x3_bias)
```

```
d_fire4_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_squeeze_weight)
d_fire4_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire4_squeeze_bias)
d_fire4_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand1x1_weight)
d_fire4_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand1x1_bias)
d_fire4_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand3x3_weight)
d_fire4_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand3x3_bias)

d_fire5_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_squeeze_weight)
d_fire5_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire5_squeeze_bias)
d_fire5_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand1x1_weight)
d_fire5_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand1x1_bias)
d_fire5_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand3x3_weight)
d_fire5_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand3x3_bias)

d_fire6_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_squeeze_weight)
d_fire6_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire6_squeeze_bias)
d_fire6_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand1x1_weight)
d_fire6_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand1x1_bias)
d_fire6_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand3x3_weight)
d_fire6_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand3x3_bias)

d_fire7_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_squeeze_weight)
d_fire7_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire7_squeeze_bias)
d_fire7_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand1x1_weight)
```

```
d_fire7_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand1x1_bias)
d_fire7_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand3x3_weight)
d_fire7_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand3x3_bias)

d_fire8_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_squeeze_weight)
d_fire8_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire8_squeeze_bias)
d_fire8_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand1x1_weight)
d_fire8_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand1x1_bias)
d_fire8_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand3x3_weight)
d_fire8_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand3x3_bias)

d_classifier_conv_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=classifier_conv_weight)
d_classifier_conv_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=classifier_conv_bias)

##################################################################

for i in range(COUNT):

    imagen = np.random.rand(3, 224, 224).astype(np.float32)
    imagen = Image.fromarray(imagen,'RGB')
    imagen = transform(imagen).numpy()
    imagen = imagen[np.newaxis,:]

    #imagen = np.ones((1, 3, 224, 224))
    #print(imagen.shape)

    ### CONV1 ###
    tic=pc()

    squeeze0=nn.Conv2d(3, 64, kernel_size=3, bias=False, stride=2)
    squeeze0.weight = nn.Parameter(torch.from_numpy(weights0))
    squeeze0.bias = nn.Parameter(torch.from_numpy(bias0))

    imagen1  = torch.from_numpy(imagen).float()
```

```
salida0=squeeze0(imagen1)
salida0_activation=squeeze_activation(salida0)

### MAXPOOL1 ###

maxpool=nn.MaxPool2d(3, stride=2)

salida_pool1 = maxpool(salida0_activation)

### SQUEEZE1 ###

squeeze1=nn.Conv2d(64, 16, kernel_size=1, bias=False)
squeeze1.weight = nn.Parameter(torch.from_numpy(weights1))
squeeze1.bias = nn.Parameter(torch.from_numpy(bias1))

salida1=squeeze1(salida_pool1)
salida1_activation=squeeze_activation(salida1)

### EXPAND1 ###

squeeze2a=nn.Conv2d(16, 64, kernel_size=1, bias=False)
squeeze2a.weight = nn.Parameter(torch.from_numpy(weights2a))
squeeze2a.bias = nn.Parameter(torch.from_numpy(bias2a))

squeeze2b=nn.Conv2d(16, 64, kernel_size=3, bias=False, padding=1)
squeeze2b.weight = nn.Parameter(torch.from_numpy(weights2b))
squeeze2b.bias = nn.Parameter(torch.from_numpy(bias2b))

salida2a=squeeze2a(salida1_activation)
salida2a_activation=squeeze_activation(salida2a)
salida2b=squeeze2b(salida1_activation)
salida2b_activation=squeeze_activation(salida2b)
salida2_total=torch.cat([salida2a_activation,salida2b_activation], 1)

### SQUEEZE2 ###

squeeze3=nn.Conv2d(128, 16, kernel_size=1, bias=False)
squeeze3.weight = nn.Parameter(torch.from_numpy(weights3))
squeeze3.bias = nn.Parameter(torch.from_numpy(bias3))

salida3=squeeze3(salida2_total)
salida3_activation=squeeze_activation(salida3)

### EXPAND2 ###

squeeze4a=nn.Conv2d(16, 64, kernel_size=1, bias=False)
squeeze4a.weight = nn.Parameter(torch.from_numpy(weights4a))
```

```
    squeeze4a.bias = nn.Parameter(torch.from_numpy(bias4a))

    squeeze4b=nn.Conv2d(16, 64, kernel_size=3, bias=False, padding=1)
    squeeze4b.weight = nn.Parameter(torch.from_numpy(weights4b))
    squeeze4b.bias = nn.Parameter(torch.from_numpy(bias4b))

    salida4a=squeeze4a(salida3_activation)
    salida4a_activation=squeeze_activation(salida4a)
    salida4b=squeeze4b(salida3_activation)
    salida4b_activation=squeeze_activation(salida4b)
    salida4_total=torch.cat([salida4a_activation,salida4b_activation], 1)

    ### MAXPOOL2 ###

    maxpool=nn.MaxPool2d(3, stride=2)

    salida_pool2 = maxpool(salida4_total)

    ### SQUEEZE3 ###

    squeeze5=nn.Conv2d(128, 32, kernel_size=1, bias=False)
    squeeze5.weight = nn.Parameter(torch.from_numpy(weights5))
    squeeze5.bias = nn.Parameter(torch.from_numpy(bias5))

    salida5=squeeze5(salida_pool2)
    salida5_activation=squeeze_activation(salida5)

    ### EXPAND3 ###

    squeeze6a=nn.Conv2d(32, 128, kernel_size=1, bias=False)
    squeeze6a.weight = nn.Parameter(torch.from_numpy(weights6a))
    squeeze6a.bias = nn.Parameter(torch.from_numpy(bias6a))

    squeeze6b=nn.Conv2d(32, 128, kernel_size=3, bias=False, padding=1)
    squeeze6b.weight = nn.Parameter(torch.from_numpy(weights6b))
    squeeze6b.bias = nn.Parameter(torch.from_numpy(bias6b))

    salida6a=squeeze6a(salida5_activation)
    salida6a_activation=squeeze_activation(salida6a)
    salida6b=squeeze6b(salida5_activation)
    salida6b_activation=squeeze_activation(salida6b)
    salida6_total=torch.cat([salida6a_activation,salida6b_activation], 1)

    ### SQUEEZE4 ###

    squeeze7=nn.Conv2d(256, 32, kernel_size=1, bias=False)
    squeeze7.weight = nn.Parameter(torch.from_numpy(weights7))
```

```
    squeeze7.bias = nn.Parameter(torch.from_numpy(bias7))

    salida7=squeeze7(salida6_total)
    salida7_activation=squeeze_activation(salida7)

    ### EXPAND4 ###

    squeeze8a=nn.Conv2d(32, 128, kernel_size=1, bias=False)
    squeeze8a.weight = nn.Parameter(torch.from_numpy(weights8a))
    squeeze8a.bias = nn.Parameter(torch.from_numpy(bias8a))

    squeeze8b=nn.Conv2d(32, 128, kernel_size=3, bias=False, padding=1)
    squeeze8b.weight = nn.Parameter(torch.from_numpy(weights8b))
    squeeze8b.bias = nn.Parameter(torch.from_numpy(bias8b))

    salida8a=squeeze8a(salida7_activation)
    salida8a_activation=squeeze_activation(salida8a)
    salida8b=squeeze8b(salida7_activation)
    salida8b_activation=squeeze_activation(salida8b)
    salida8_total=torch.cat([salida8a_activation,salida8b_activation], 1)

    ### MAXPOOL3 ###

    maxpool=nn.MaxPool2d(3, stride=2)

    salida_pool3 = maxpool(salida8_total)

    ### SQUEEZE5 ###

    squeeze9=nn.Conv2d(256, 48, kernel_size=1, bias=False)
    squeeze9.weight = nn.Parameter(torch.from_numpy(weights9))
    squeeze9.bias = nn.Parameter(torch.from_numpy(bias9))

    salida9=squeeze9(salida_pool3)
    salida9_activation=squeeze_activation(salida9)

    ### EXPAND5 ###

    squeeze10a=nn.Conv2d(48, 192, kernel_size=1, bias=False)
    squeeze10a.weight = nn.Parameter(torch.from_numpy(weights10a))
    squeeze10a.bias = nn.Parameter(torch.from_numpy(bias10a))

    squeeze10b=nn.Conv2d(48, 192, kernel_size=3, bias=False, padding=1)
    squeeze10b.weight = nn.Parameter(torch.from_numpy(weights10b))
    squeeze10b.bias = nn.Parameter(torch.from_numpy(bias10b))

    salida10a=squeeze10a(salida9_activation)
```

```
salida10a_activation=squeeze_activation(salida10a)
salida10b=squeeze10b(salida9_activation)
salida10b_activation=squeeze_activation(salida10b)
salida10_total=torch.cat([salida10a_activation,salida10b_activation], 1)

### SQUEEZE6 ###

squeeze11=nn.Conv2d(384, 48, kernel_size=1, bias=False)
squeeze11.weight = nn.Parameter(torch.from_numpy(weights11))
squeeze11.bias = nn.Parameter(torch.from_numpy(bias11))

salida11=squeeze11(salida10_total)
salida11_activation=squeeze_activation(salida11)

## EXPAND6 ###

squeeze12a=nn.Conv2d(48, 192, kernel_size=1, bias=False)
squeeze12a.weight = nn.Parameter(torch.from_numpy(weights12a))
squeeze12a.bias = nn.Parameter(torch.from_numpy(bias12a))

squeeze12b=nn.Conv2d(48, 192, kernel_size=3, bias=False, padding=1)
squeeze12b.weight = nn.Parameter(torch.from_numpy(weights12b))
squeeze12b.bias = nn.Parameter(torch.from_numpy(bias12b))

salida12a=squeeze12a(salida11_activation)
salida12a_activation=squeeze_activation(salida12a)
salida12b=squeeze12b(salida11_activation)
salida12b_activation=squeeze_activation(salida12b)
salida12_total=torch.cat([salida12a_activation,salida12b_activation], 1)

### SQUEEZE7 ###

squeeze13=nn.Conv2d(384, 64, kernel_size=1, bias=False)
squeeze13.weight = nn.Parameter(torch.from_numpy(weights13))
squeeze13.bias = nn.Parameter(torch.from_numpy(bias13))

salida13=squeeze13(salida12_total)
salida13_activation=squeeze_activation(salida13)

### EXPAND7 ###

squeeze14a=nn.Conv2d(64, 256, kernel_size=1, bias=False)
squeeze14a.weight = nn.Parameter(torch.from_numpy(weights14a))
squeeze14a.bias = nn.Parameter(torch.from_numpy(bias14a))

squeeze14b=nn.Conv2d(64, 256, kernel_size=3, bias=False, padding=1)
squeeze14b.weight = nn.Parameter(torch.from_numpy(weights14b))
```

```
    squeeze14b.bias = nn.Parameter(torch.from_numpy(bias14b))

    salida14a=squeeze14a(salida13_activation)
    salida14a_activation=squeeze_activation(salida14a)
    salida14b=squeeze14b(salida13_activation)
    salida14b_activation=squeeze_activation(salida14b)
    salida14_total=torch.cat([salida14a_activation,salida14b_activation], 1)

    ### SQUEEZE8 ###

    squeeze15=nn.Conv2d(512, 64, kernel_size=1, bias=False)
    squeeze15.weight = nn.Parameter(torch.from_numpy(weights15))
    squeeze15.bias = nn.Parameter(torch.from_numpy(bias15))

    salida15=squeeze15(salida14_total)
    salida15_activation=squeeze_activation(salida15)

    ### EXPAND8 ###

    squeeze16a=nn.Conv2d(64, 256, kernel_size=1, bias=False)
    squeeze16a.weight = nn.Parameter(torch.from_numpy(weights16a))
    squeeze16a.bias = nn.Parameter(torch.from_numpy(bias16a))

    squeeze16b=nn.Conv2d(64, 256, kernel_size=3, bias=False, padding=1)
    squeeze16b.weight = nn.Parameter(torch.from_numpy(weights16b))
    squeeze16b.bias = nn.Parameter(torch.from_numpy(bias16b))

    salida16a=squeeze16a(salida15_activation)
    salida16a_activation=squeeze_activation(salida16a)
    salida16b=squeeze16b(salida15_activation)
    salida16b_activation=squeeze_activation(salida16b)
    salida16_total=torch.cat([salida16a_activation,salida16b_activation], 1)

    ### CONV2 ###

    conv_class=nn.Conv2d(512, 1000, kernel_size=1, bias=False)
    conv_class.weight = nn.Parameter(torch.from_numpy(weights17))
    conv_class.bias = nn.Parameter(torch.from_numpy(bias17))

    salida17=conv_class(salida16_total)
    salida17_activation=squeeze_activation(salida17)

    ### AVGPOOL1 ###

    avgpool=nn.AvgPool2d(13)

    salida18=avgpool(salida17_activation)
```

```
   salida18_a_numpy=salida18.detach().numpy()

   toc=pc()

   pytorch_time += (toc - tic)

   ####### OPENCL COMPARISON #######

   ## NDRANGE

   h_sample = imagen.reshape(-1).astype(np.float32)
   d_sample = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
↪COPY_HOST_PTR, hostbuf=h_sample)

   h_result_conv = np.empty(1 * 64 * 111 * 111).astype(np.float32)
   h_result_pool1 = np.empty(1 * 64 * 55 * 55).astype(np.float32)

   h_result_fire1_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
   h_result_fire1_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
   h_result_fire2_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
   h_result_fire2_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
   h_result_pool2 = np.empty(1 * 128 * 27 * 27).astype(np.float32)

   h_result_fire3_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
   h_result_fire3_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
   h_result_fire4_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
   h_result_fire4_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
   h_result_pool3 = np.empty(1 * 256 * 13 * 13).astype(np.float32)

   h_result_fire5_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
   h_result_fire5_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
   h_result_fire6_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
   h_result_fire6_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
   h_result_fire7_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
   h_result_fire7_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)
   h_result_fire8_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
   h_result_fire8_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)

   h_result_classifier_conv = np.empty(1 * 1000 * 13 * 13).astype(np.float32)
   h_result_classifier = np.empty(1 * 1000).astype(np.float32)

   d_result_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.
↪nbytes)
   d_result_pool1 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
↪nbytes)
```

```
    d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_squeeze.nbytes)
    d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_expand.nbytes)
    d_result_fire2_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_squeeze.nbytes)
    d_result_fire2_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_expand.nbytes)
    d_result_pool2 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool2.
↪nbytes)

    d_result_fire3_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_squeeze.nbytes)
    d_result_fire3_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_expand.nbytes)
    d_result_fire4_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_squeeze.nbytes)
    d_result_fire4_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_expand.nbytes)
    d_result_pool3 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool3.
↪nbytes)

    d_result_fire5_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_squeeze.nbytes)
    d_result_fire5_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_expand.nbytes)
    d_result_fire6_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_squeeze.nbytes)
    d_result_fire6_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_expand.nbytes)
    d_result_fire7_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_squeeze.nbytes)
    d_result_fire7_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_expand.nbytes)
    d_result_fire8_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_squeeze.nbytes)
    d_result_fire8_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_expand.nbytes)

    d_result_classifier_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier_conv.nbytes)
    d_result_classifier = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier.nbytes)

    ##### DUMMY FOR REPROGRAMMING ####
```

```
    d_dummy_in = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.
↪nbytes)
    d_dummy_out = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
↪nbytes)

    maxpool_NDR(queue, (64, ), None, 111, 55, d_dummy_in, d_dummy_out)

    ##################################

    ### CONV1 ###
    tic1=pc()

    conv3x3_NDR(queue,(64, 111), None, 3, 224, 0, 2, 0, 111, d_sample,␣
↪d_conv1_weight, d_conv1_bias, d_result_conv)

    ### MAXPOOL1 ###

    maxpool_NDR(queue, (64, ), None, 111, 55, d_result_conv, d_result_pool1)

    ### SQUEEZE1 ###

    conv1x1_NDR(queue,(16, 55), None, np.int32(64/4), 55, d_result_pool1,␣
↪d_fire1_squeeze_weight, d_fire1_squeeze_bias, d_result_fire1_squeeze)
    queue.finish()

    ### EXPAND1 ###

    conv1x1_NDR(queue,(64, 55), None, np.int32(16/4), 55,␣
↪d_result_fire1_squeeze, d_fire1_expand1x1_weight, d_fire1_expand1x1_bias,␣
↪d_result_fire1_expand)
    conv3x3_NDR(queue,(64, 55), None, 16, 55, 1, 1, 64, 55,␣
↪d_result_fire1_squeeze, d_fire1_expand3x3_weight, d_fire1_expand3x3_bias,␣
↪d_result_fire1_expand)
    queue.finish()

    ### SQUEEZE2 ###

    conv1x1_NDR(queue,(16, 55), None, np.int32(128/4), 55,␣
↪d_result_fire1_expand, d_fire2_squeeze_weight, d_fire2_squeeze_bias,␣
↪d_result_fire2_squeeze)
    queue.finish()

    ### EXPAND2 ###
```

```
    conv1x1_NDR(queue,(64, 55), None, np.int32(16/4), 55,␣
↪d_result_fire2_squeeze, d_fire2_expand1x1_weight, d_fire2_expand1x1_bias,␣
↪d_result_fire2_expand)
    conv3x3_NDR(queue,(64, 55), None, 16, 55, 1, 1, 64, 55,␣
↪d_result_fire2_squeeze, d_fire2_expand3x3_weight, d_fire2_expand3x3_bias,␣
↪d_result_fire2_expand)
    queue.finish()

    ### MAXPOOL2 ###

    maxpool_NDR(queue, (128, ), None, 55, 27, d_result_fire2_expand,␣
↪d_result_pool2)

    ### SQUEEZE3 ###

    conv1x1_NDR(queue,(32, 27), None, np.int32(128/4), 27, d_result_pool2,␣
↪d_fire3_squeeze_weight, d_fire3_squeeze_bias, d_result_fire3_squeeze)
    queue.finish()

    ### EXPAND3 ###

    conv1x1_NDR(queue,(128, 27), None, np.int32(32/4), 27,␣
↪d_result_fire3_squeeze, d_fire3_expand1x1_weight, d_fire3_expand1x1_bias,␣
↪d_result_fire3_expand)
    conv3x3_NDR(queue,(128, 27), None, 32, 27, 1, 1, 128, 27,␣
↪d_result_fire3_squeeze, d_fire3_expand3x3_weight, d_fire3_expand3x3_bias,␣
↪d_result_fire3_expand)
    queue.finish()

    ### SQUEEZE4 ###

    conv1x1_NDR(queue,(32, 27), None, np.int32(256/4), 27,␣
↪d_result_fire3_expand, d_fire4_squeeze_weight, d_fire4_squeeze_bias,␣
↪d_result_fire4_squeeze)
    queue.finish()

    ### EXPAND4 ###

    conv1x1_NDR(queue,(128, 27), None, np.int32(32/4), 27,␣
↪d_result_fire4_squeeze, d_fire4_expand1x1_weight, d_fire4_expand1x1_bias,␣
↪d_result_fire4_expand)
    conv3x3_NDR(queue,(128, 27), None, 32, 27, 1, 1, 128, 27,␣
↪d_result_fire4_squeeze, d_fire4_expand3x3_weight, d_fire4_expand3x3_bias,␣
↪d_result_fire4_expand)
    queue.finish()
```

```
    ### MAXPOOL3 ###

    maxpool_NDR(queue, (256, ), None, 27, 13, d_result_fire4_expand,␣
↪d_result_pool3)

    ### SQUEEZE5 ###

    conv1x1_NDR(queue,(48, 13), None, np.int32(256/4), 13, d_result_pool3,␣
↪d_fire5_squeeze_weight, d_fire5_squeeze_bias, d_result_fire5_squeeze)
    queue.finish()

    ### EXPAND5 ###

    conv1x1_NDR(queue,(192, 13), None, np.int32(48/4), 13,␣
↪d_result_fire5_squeeze, d_fire5_expand1x1_weight, d_fire5_expand1x1_bias,␣
↪d_result_fire5_expand)
    conv3x3_NDR(queue,(192, 13), None, 48, 13, 1, 1, 192, 13,␣
↪d_result_fire5_squeeze, d_fire5_expand3x3_weight, d_fire5_expand3x3_bias,␣
↪d_result_fire5_expand)
    queue.finish()

    ### SQUEEZE6 ###

    conv1x1_NDR(queue,(48, 13), None, np.int32(384/4), 13,␣
↪d_result_fire5_expand, d_fire6_squeeze_weight, d_fire6_squeeze_bias,␣
↪d_result_fire6_squeeze)
    queue.finish()

    ### EXPAND6 ###

    conv1x1_NDR(queue,(192, 13), None, np.int32(48/4), 13,␣
↪d_result_fire6_squeeze, d_fire6_expand1x1_weight, d_fire6_expand1x1_bias,␣
↪d_result_fire6_expand)
    conv3x3_NDR(queue,(192, 13), None, 48, 13, 1, 1, 192, 13,␣
↪d_result_fire6_squeeze, d_fire6_expand3x3_weight, d_fire6_expand3x3_bias,␣
↪d_result_fire6_expand)
    queue.finish()

    ### SQUEEZE7 ###

    conv1x1_NDR(queue,(64, 13), None, np.int32(384/4), 13,␣
↪d_result_fire6_expand, d_fire7_squeeze_weight, d_fire7_squeeze_bias,␣
↪d_result_fire7_squeeze)
    queue.finish()

    ### EXPAND7 ###
```

```
    conv1x1_NDR(queue,(256, 13), None, np.int32(64/4), 13,␣
↪d_result_fire7_squeeze, d_fire7_expand1x1_weight, d_fire7_expand1x1_bias,␣
↪d_result_fire7_expand)
    conv3x3_NDR(queue,(256, 13), None, 64, 13, 1, 1, 256, 13,␣
↪d_result_fire7_squeeze, d_fire7_expand3x3_weight, d_fire7_expand3x3_bias,␣
↪d_result_fire7_expand)
    queue.finish()

    ### SQUEEZE8 ###

    conv1x1_NDR(queue,(64, 13), None, np.int32(512/4), 13,␣
↪d_result_fire7_expand, d_fire8_squeeze_weight, d_fire8_squeeze_bias,␣
↪d_result_fire8_squeeze)
    queue.finish()

    ### EXPAND8 ###

    conv1x1_NDR(queue,(256, 13), None, np.int32(64/4), 13,␣
↪d_result_fire8_squeeze, d_fire8_expand1x1_weight, d_fire8_expand1x1_bias,␣
↪d_result_fire8_expand)
    conv3x3_NDR(queue,(256, 13), None, 64, 13, 1, 1, 256, 13,␣
↪d_result_fire8_squeeze, d_fire8_expand3x3_weight, d_fire8_expand3x3_bias,␣
↪d_result_fire8_expand)
    queue.finish()

    ### CONV2 ###

    conv1x1_NDR(queue,(1000, 13), None, np.int32(512/4), 13,␣
↪d_result_fire8_expand, d_classifier_conv_weight, d_classifier_conv_bias,␣
↪d_result_classifier_conv)

    ### AVGPOOL ###

    avgpool_NDR(queue,(1000, ), None, d_result_classifier_conv,␣
↪d_result_classifier)

    cl.enqueue_copy(queue, h_result_classifier, d_result_classifier)

    queue.finish()

    veamos = h_result_classifier

    toc1=pc()

    NDRange_time += (toc1 - tic1)
```

```
   ## OUR NDRANGE

   h_sample = imagen.reshape(-1).astype(np.float32)
   d_sample = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
↪COPY_HOST_PTR, hostbuf=h_sample)

   h_result_conv = np.empty(1 * 64 * 111 * 111).astype(np.float32)
   h_result_pool1 = np.empty(1 * 64 * 55 * 55).astype(np.float32)

   h_result_fire1_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
   h_result_fire1_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
   h_result_fire2_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
   h_result_fire2_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
   h_result_pool2 = np.empty(1 * 128 * 27 * 27).astype(np.float32)

   h_result_fire3_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
   h_result_fire3_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
   h_result_fire4_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
   h_result_fire4_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
   h_result_pool3 = np.empty(1 * 256 * 13 * 13).astype(np.float32)

   h_result_fire5_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
   h_result_fire5_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
   h_result_fire6_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
   h_result_fire6_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
   h_result_fire7_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
   h_result_fire7_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)
   h_result_fire8_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
   h_result_fire8_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)

   h_result_classifier_conv = np.empty(1 * 1000 * 13 * 13).astype(np.float32)
   h_result_classifier = np.empty(1 * 1000).astype(np.float32)

   d_result_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.
↪nbytes)
   d_result_pool1 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
↪nbytes)

   d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_squeeze.nbytes)
   d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_expand.nbytes)
   d_result_fire2_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_squeeze.nbytes)
```

```
    d_result_fire2_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_expand.nbytes)
    d_result_pool2 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool2.
↪nbytes)

    d_result_fire3_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_squeeze.nbytes)
    d_result_fire3_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_expand.nbytes)
    d_result_fire4_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_squeeze.nbytes)
    d_result_fire4_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_expand.nbytes)
    d_result_pool3 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool3.
↪nbytes)

    d_result_fire5_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_squeeze.nbytes)
    d_result_fire5_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_expand.nbytes)
    d_result_fire6_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_squeeze.nbytes)
    d_result_fire6_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_expand.nbytes)
    d_result_fire7_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_squeeze.nbytes)
    d_result_fire7_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_expand.nbytes)
    d_result_fire8_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_squeeze.nbytes)
    d_result_fire8_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_expand.nbytes)

    d_result_classifier_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier_conv.nbytes)
    d_result_classifier = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier.nbytes)

    ### CONV1 ###
    tic2=pc()

    conv3x3_NDR(queue,(64, 111), None, 3, 224, 0, 2, 0, 111, d_sample,␣
↪d_conv1_weight, d_conv1_bias, d_result_conv)

    ### MAXPOOL1 ###
```

```
    maxpool_NDR(queue, (64, ), None, 111, 55, d_result_conv, d_result_pool1)

    ### SQUEEZE1 ###

    conv1x1_NDR(queue,(16, 55), None, np.int32(64/4), 55, d_result_pool1,␣
↪d_fire1_squeeze_weight, d_fire1_squeeze_bias, d_result_fire1_squeeze)
    queue.finish()

    ### EXPAND1 ###

    conv1x1_NDR(queue1,(64, 55), None, np.int32(16/4), 55,␣
↪d_result_fire1_squeeze, d_fire1_expand1x1_weight, d_fire1_expand1x1_bias,␣
↪d_result_fire1_expand)
    conv3x3_NDR(queue,(64, 55), None, 16, 55, 1, 1, 64, 55,␣
↪d_result_fire1_squeeze, d_fire1_expand3x3_weight, d_fire1_expand3x3_bias,␣
↪d_result_fire1_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE2 ###

    conv1x1_NDR(queue,(16, 55), None, np.int32(128/4), 55,␣
↪d_result_fire1_expand, d_fire2_squeeze_weight, d_fire2_squeeze_bias,␣
↪d_result_fire2_squeeze)
    queue.finish()

    ### EXPAND2 ###

    conv1x1_NDR(queue1,(64, 55), None, np.int32(16/4), 55,␣
↪d_result_fire2_squeeze, d_fire2_expand1x1_weight, d_fire2_expand1x1_bias,␣
↪d_result_fire2_expand)
    conv3x3_NDR(queue,(64, 55), None, 16, 55, 1, 1, 64, 55,␣
↪d_result_fire2_squeeze, d_fire2_expand3x3_weight, d_fire2_expand3x3_bias,␣
↪d_result_fire2_expand)
    queue.finish()
    queue1.finish()

    ### MAXPOOL2 ###

    maxpool_NDR(queue, (128, ), None, 55, 27, d_result_fire2_expand,␣
↪d_result_pool2)

    ### SQUEEZE3 ###

    conv1x1_NDR(queue,(32, 27), None, np.int32(128/4), 27, d_result_pool2,␣
↪d_fire3_squeeze_weight, d_fire3_squeeze_bias, d_result_fire3_squeeze)
```

```
    queue.finish()

    ### EXPAND3 ###

    conv1x1_NDR(queue1,(128, 27), None, np.int32(32/4), 27,␣
↪d_result_fire3_squeeze, d_fire3_expand1x1_weight, d_fire3_expand1x1_bias,␣
↪d_result_fire3_expand)
    conv3x3_NDR(queue,(128, 27), None, 32, 27, 1, 1, 128, 27,␣
↪d_result_fire3_squeeze, d_fire3_expand3x3_weight, d_fire3_expand3x3_bias,␣
↪d_result_fire3_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE4 ###

    conv1x1_NDR(queue,(32, 27), None, np.int32(256/4), 27,␣
↪d_result_fire3_expand, d_fire4_squeeze_weight, d_fire4_squeeze_bias,␣
↪d_result_fire4_squeeze)
    queue.finish()

    ### EXPAND4 ###

    conv1x1_NDR(queue1,(128, 27), None, np.int32(32/4), 27,␣
↪d_result_fire4_squeeze, d_fire4_expand1x1_weight, d_fire4_expand1x1_bias,␣
↪d_result_fire4_expand)
    conv3x3_NDR(queue,(128, 27), None, 32, 27, 1, 1, 128, 27,␣
↪d_result_fire4_squeeze, d_fire4_expand3x3_weight, d_fire4_expand3x3_bias,␣
↪d_result_fire4_expand)
    queue.finish()
    queue1.finish()

    ### MAXPOOL3 ###

    maxpool_NDR(queue, (256, ), None, 27, 13, d_result_fire4_expand,␣
↪d_result_pool3)

    ### SQUEEZE5 ###

    conv1x1_NDR(queue,(48, 13), None, np.int32(256/4), 13, d_result_pool3,␣
↪d_fire5_squeeze_weight, d_fire5_squeeze_bias, d_result_fire5_squeeze)
    queue.finish()

    ### EXPAND5 ###
```

```
    conv1x1_NDR(queue1,(192, 13), None, np.int32(48/4), 13,␣
↪d_result_fire5_squeeze, d_fire5_expand1x1_weight, d_fire5_expand1x1_bias,␣
↪d_result_fire5_expand)
    conv3x3_NDR(queue,(192, 13), None, 48, 13, 1, 1, 192, 13,␣
↪d_result_fire5_squeeze, d_fire5_expand3x3_weight, d_fire5_expand3x3_bias,␣
↪d_result_fire5_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE6 ###

    conv1x1_NDR(queue,(48, 13), None, np.int32(384/4), 13,␣
↪d_result_fire5_expand, d_fire6_squeeze_weight, d_fire6_squeeze_bias,␣
↪d_result_fire6_squeeze)
    queue.finish()

    ### EXPAND6 ###

    conv1x1_NDR(queue1,(192, 13), None, np.int32(48/4), 13,␣
↪d_result_fire6_squeeze, d_fire6_expand1x1_weight, d_fire6_expand1x1_bias,␣
↪d_result_fire6_expand)
    conv3x3_NDR(queue,(192, 13), None, 48, 13, 1, 1, 192, 13,␣
↪d_result_fire6_squeeze, d_fire6_expand3x3_weight, d_fire6_expand3x3_bias,␣
↪d_result_fire6_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE7 ###

    conv1x1_NDR(queue,(64, 13), None, np.int32(384/4), 13,␣
↪d_result_fire6_expand, d_fire7_squeeze_weight, d_fire7_squeeze_bias,␣
↪d_result_fire7_squeeze)
    queue.finish()

    ### EXPAND7 ###

    conv1x1_NDR(queue1,(256, 13), None, np.int32(64/4), 13,␣
↪d_result_fire7_squeeze, d_fire7_expand1x1_weight, d_fire7_expand1x1_bias,␣
↪d_result_fire7_expand)
    conv3x3_NDR(queue,(256, 13), None, 64, 13, 1, 1, 256, 13,␣
↪d_result_fire7_squeeze, d_fire7_expand3x3_weight, d_fire7_expand3x3_bias,␣
↪d_result_fire7_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE8 ###
```

```
   conv1x1_NDR(queue,(64, 13), None, np.int32(512/4), 13,␣
↪d_result_fire7_expand, d_fire8_squeeze_weight, d_fire8_squeeze_bias,␣
↪d_result_fire8_squeeze)
   queue.finish()

   ### EXPAND8 ###

   conv1x1_NDR(queue1,(256, 13), None, np.int32(64/4), 13,␣
↪d_result_fire8_squeeze, d_fire8_expand1x1_weight, d_fire8_expand1x1_bias,␣
↪d_result_fire8_expand)
   conv3x3_NDR(queue,(256, 13), None, 64, 13, 1, 1, 256, 13,␣
↪d_result_fire8_squeeze, d_fire8_expand3x3_weight, d_fire8_expand3x3_bias,␣
↪d_result_fire8_expand)
   queue.finish()
   queue1.finish()

   ### CONV2 ###

   conv1x1_NDR(queue,(1000, 13), None, np.int32(512/4), 13,␣
↪d_result_fire8_expand, d_classifier_conv_weight, d_classifier_conv_bias,␣
↪d_result_classifier_conv)

   ### AVGPOOL ###

   avgpool_NDR(queue,(1000, ), None, d_result_classifier_conv,␣
↪d_result_classifier)

   cl.enqueue_copy(queue, h_result_classifier, d_result_classifier)

   queue.finish()

   veamos1 = h_result_classifier

   toc2=pc()

   Our_NDRange_time += (toc2 - tic2)

   ## SINGLE TASK

   h_sample = imagen.reshape(-1).astype(np.float32)
   d_sample = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
↪COPY_HOST_PTR, hostbuf=h_sample)

   h_result_conv = np.empty(1 * 64 * 111 * 111).astype(np.float32)
   h_result_pool1 = np.empty(1 * 64 * 55 * 55).astype(np.float32)
```

```
    h_result_fire1_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
    h_result_fire1_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
    h_result_fire2_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
    h_result_fire2_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
    h_result_pool2 = np.empty(1 * 128 * 27 * 27).astype(np.float32)

    h_result_fire3_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
    h_result_fire3_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
    h_result_fire4_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
    h_result_fire4_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
    h_result_pool3 = np.empty(1 * 256 * 13 * 13).astype(np.float32)

    h_result_fire5_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
    h_result_fire5_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
    h_result_fire6_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
    h_result_fire6_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
    h_result_fire7_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
    h_result_fire7_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)
    h_result_fire8_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
    h_result_fire8_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)

    h_result_classifier_conv = np.empty(1 * 1000 * 13 * 13).astype(np.float32)
    h_result_classifier = np.empty(1 * 1000).astype(np.float32)

    d_result_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.
↪nbytes)
    d_result_pool1 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
↪nbytes)

    d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_squeeze.nbytes)
    d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_expand.nbytes)
    d_result_fire2_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_squeeze.nbytes)
    d_result_fire2_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_expand.nbytes)
    d_result_pool2 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool2.
↪nbytes)

    d_result_fire3_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_squeeze.nbytes)
    d_result_fire3_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_expand.nbytes)
```

```
    d_result_fire4_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_squeeze.nbytes)
    d_result_fire4_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_expand.nbytes)
    d_result_pool3 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool3.
↪nbytes)

    d_result_fire5_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_squeeze.nbytes)
    d_result_fire5_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_expand.nbytes)
    d_result_fire6_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_squeeze.nbytes)
    d_result_fire6_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_expand.nbytes)
    d_result_fire7_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_squeeze.nbytes)
    d_result_fire7_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_expand.nbytes)
    d_result_fire8_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_squeeze.nbytes)
    d_result_fire8_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_expand.nbytes)

    d_result_classifier_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier_conv.nbytes)
    d_result_classifier = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier.nbytes)

    ##### DUMMY FOR REPROGRAMMING ####

    d_dummy_in = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.
↪nbytes)
    d_dummy_out = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
↪nbytes)

    maxpool_ST(queue, (1, ), None, 111, 55, 64, d_dummy_in, d_dummy_out)

    ##################################

    ### CONV1 ###
    tic3=pc()

    conv3x3_ST(queue,(1,), None, 3, 224, 0, 2, 0, 111, 64, d_sample,␣
↪d_conv1_weight, d_conv1_bias, d_result_conv)
```

```
    ### MAXPOOL1 ###

    maxpool_ST(queue, (1, ), None, 111, 55, 64, d_result_conv, d_result_pool1)

    ### SQUEEZE1 ###

    conv1x1_ST(queue,(1,), None, 64, 55, 16, d_result_pool1,␣
↪d_fire1_squeeze_weight, d_fire1_squeeze_bias, d_result_fire1_squeeze)
    queue.finish()

    ### EXPAND1 ###

    conv1x1_ST(queue1,(1,), None, 16, 55, 64, d_result_fire1_squeeze,␣
↪d_fire1_expand1x1_weight, d_fire1_expand1x1_bias, d_result_fire1_expand)
    conv3x3_ST(queue,(1,), None, 16, 55, 1, 1, 64, 55, 64,␣
↪d_result_fire1_squeeze, d_fire1_expand3x3_weight, d_fire1_expand3x3_bias,␣
↪d_result_fire1_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE2 ###


    conv1x1_ST(queue,(1,), None, 128, 55, 16, d_result_fire1_expand,␣
↪d_fire2_squeeze_weight, d_fire2_squeeze_bias, d_result_fire2_squeeze)
    queue.finish()

    ### EXPAND2 ###

    conv1x1_ST(queue1,(1,), None, 16, 55, 64, d_result_fire2_squeeze,␣
↪d_fire2_expand1x1_weight, d_fire2_expand1x1_bias, d_result_fire2_expand)
    conv3x3_ST(queue,(1,), None, 16, 55, 1, 1, 64, 55, 64,␣
↪d_result_fire2_squeeze, d_fire2_expand3x3_weight, d_fire2_expand3x3_bias,␣
↪d_result_fire2_expand)
    queue.finish()
    queue1.finish()

    ### MAXPOOL2 ###

    maxpool_ST(queue, (1, ), None, 55, 27, 128, d_result_fire2_expand,␣
↪d_result_pool2)

    ### SQUEEZE3 ###

    conv1x1_ST(queue,(1,), None, 128, 27, 32, d_result_pool2,␣
↪d_fire3_squeeze_weight, d_fire3_squeeze_bias, d_result_fire3_squeeze)
```

```
    queue.finish()

    ### EXPAND3 ###

    conv1x1_ST(queue1,(1,), None, 32, 27, 128, d_result_fire3_squeeze,␣
↪d_fire3_expand1x1_weight, d_fire3_expand1x1_bias, d_result_fire3_expand)
    conv3x3_ST(queue,(1,), None, 32, 27, 1, 1, 128, 27, 128,␣
↪d_result_fire3_squeeze, d_fire3_expand3x3_weight, d_fire3_expand3x3_bias,␣
↪d_result_fire3_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE4 ###

    conv1x1_ST(queue,(1,), None, 256, 27, 32, d_result_fire3_expand,␣
↪d_fire4_squeeze_weight, d_fire4_squeeze_bias, d_result_fire4_squeeze)
    queue.finish()

    ### EXPAND4 ###

    conv1x1_ST(queue1,(1,), None, 32, 27, 128, d_result_fire4_squeeze,␣
↪d_fire4_expand1x1_weight, d_fire4_expand1x1_bias, d_result_fire4_expand)
    conv3x3_ST(queue,(1,), None, 32, 27, 1, 1, 128, 27, 128,␣
↪d_result_fire4_squeeze, d_fire4_expand3x3_weight, d_fire4_expand3x3_bias,␣
↪d_result_fire4_expand)
    queue.finish()
    queue1.finish()

    ### MAXPOOL3 ###

    maxpool_ST(queue, (1, ), None, 27, 13, 256, d_result_fire4_expand,␣
↪d_result_pool3)

    ### SQUEEZE5 ###

    conv1x1_ST(queue,(1,), None, 256, 13, 48, d_result_pool3,␣
↪d_fire5_squeeze_weight, d_fire5_squeeze_bias, d_result_fire5_squeeze)
    queue.finish()

    ### EXPAND5 ###

    conv1x1_ST(queue1,(1,), None, 48, 13, 192, d_result_fire5_squeeze,␣
↪d_fire5_expand1x1_weight, d_fire5_expand1x1_bias, d_result_fire5_expand)
    conv3x3_ST(queue,(1,), None, 48, 13, 1, 1, 192, 13, 192,␣
↪d_result_fire5_squeeze, d_fire5_expand3x3_weight, d_fire5_expand3x3_bias,␣
↪d_result_fire5_expand)
```

```
    queue.finish()
    queue1.finish()

    ### SQUEEZE6 ###

    conv1x1_ST(queue,(1,), None, 384, 13, 48, d_result_fire5_expand,␣
↪d_fire6_squeeze_weight, d_fire6_squeeze_bias, d_result_fire6_squeeze)
    queue.finish()

    ### EXPAND6 ###

    conv1x1_ST(queue1,(1,), None, 48, 13, 192, d_result_fire6_squeeze,␣
↪d_fire6_expand1x1_weight, d_fire6_expand1x1_bias, d_result_fire6_expand)
    conv3x3_ST(queue,(1,), None, 48, 13, 1, 1, 192, 13, 192,␣
↪d_result_fire6_squeeze, d_fire6_expand3x3_weight, d_fire6_expand3x3_bias,␣
↪d_result_fire6_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE7 ###

    conv1x1_ST(queue,(1,), None, 384, 13, 64, d_result_fire6_expand,␣
↪d_fire7_squeeze_weight, d_fire7_squeeze_bias, d_result_fire7_squeeze)
    queue.finish()

    ### EXPAND7 ###

    conv1x1_ST(queue1,(1,), None, 64, 13, 256, d_result_fire7_squeeze,␣
↪d_fire7_expand1x1_weight, d_fire7_expand1x1_bias, d_result_fire7_expand)
    conv3x3_ST(queue,(1,), None, 64, 13, 1, 1, 256, 13, 256,␣
↪d_result_fire7_squeeze, d_fire7_expand3x3_weight, d_fire7_expand3x3_bias,␣
↪d_result_fire7_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE8 ###

    conv1x1_ST(queue,(1,), None, 512, 13, 64, d_result_fire7_expand,␣
↪d_fire8_squeeze_weight, d_fire8_squeeze_bias, d_result_fire8_squeeze)
    queue.finish()

    ### EXPAND8 ###

    conv1x1_ST(queue1,(1,), None, 64, 13, 256, d_result_fire8_squeeze,␣
↪d_fire8_expand1x1_weight, d_fire8_expand1x1_bias, d_result_fire8_expand)
```

```
    conv3x3_ST(queue,(1,), None, 64, 13, 1, 1, 256, 13, 256,␣
 ↪d_result_fire8_squeeze, d_fire8_expand3x3_weight, d_fire8_expand3x3_bias,␣
 ↪d_result_fire8_expand)
    queue.finish()
    queue1.finish()

    ### CONV2 ###

    conv1x1_ST(queue,(1,), None, 512, 13, 1000, d_result_fire8_expand,␣
 ↪d_classifier_conv_weight, d_classifier_conv_bias, d_result_classifier_conv)

    ### TEST ###
#     cl.enqueue_copy(queue, h_result_classifier_conv, d_result_classifier_conv)

    ### AVGPOOL ###

    avgpool_ST(queue, (1, ), None, d_result_classifier_conv, d_result_classifier)

    cl.enqueue_copy(queue, h_result_classifier, d_result_classifier)

    veamos2 = h_result_classifier

    toc3=pc()

    ST_time += (toc3 - tic3)

    comparativa1&=np.allclose(salida18_a_numpy.reshape(-1), veamos,rtol=1e-01,␣
 ↪atol=1e-01)
    comparativa2&=np.allclose(salida18_a_numpy.reshape(-1), veamos1,rtol=1e-01,␣
 ↪atol=1e-01)
    comparativa2&=np.allclose(salida18_a_numpy.reshape(-1), veamos2,rtol=1e-01,␣
 ↪atol=1e-01)
    comparativa4&=np.allclose(veamos1, veamos2,rtol=1e-01, atol=1e-01)

    if (comparativa1 == False or comparativa2 == False or comparativa3 == False␣
 ↪or comparativa4 == False):
        print('Error en : ', i)
        break


print("comparativa (pytorch == NDRange): ",comparativa1)
print("comparativa (pytorch == Our NDRange): ",comparativa2)
print("comparativa (pytorch ==", file_dir[:-5], "): ",comparativa3)
print("comparativa (Our NDRange == Simple Task): ",comparativa4)

pytorch_time = pytorch_time/(float(COUNT))
```

```
NDRange_time = NDRange_time/(float(COUNT))
Our_NDRange_time = Our_NDRange_time/(float(COUNT))
ST_time = ST_time/(float(COUNT))

print ("tiempo en segundos con pytorch= ", pytorch_time)
print ("tiempo en segundos con opencl (NDRange)=",NDRange_time)
print ("tiempo en segundos con opencl (Our NDRange)=",Our_NDRange_time)
print ("tiempo en segundos con opencl (", file_dir[:-5], ")=",ST_time)
```

```
Reprogramming device [0] with handle 220
...
Reprogramming device [0] with handle 318
Reprogramming device [0] with handle 319
```

```
comparativa (pytorch == NDRange):  True
comparativa (pytorch == Our NDRange):  True
comparativa (pytorch == squeezenet_ST_V6 ):  True
comparativa (Our NDRange == Simple Task):  True
tiempo en segundos con pytorch=  1.1778390008799762
tiempo en segundos con opencl (NDRange)= 1.041179874300069
tiempo en segundos con opencl (Our NDRange)= 1.000556468840041
tiempo en segundos con opencl ( squeezenet_ST_V6 )= 4.28209365277995
```

Obtención de resultados con ST Float4

[12]:
```
squeeze_activation = nn.ReLU(inplace=True)

pytorch_time = 0.0
NDRange_time = 0.0
Our_NDRange_time = 0.0
ST_time = 0.0

COUNT = 50

comparativa1 = True
comparativa2 = True
comparativa3 = True
comparativa4 = True

################################################################

transform = transforms.Compose([
transforms.Resize(256),
transforms.CenterCrop(224),
transforms.ToTensor(),
transforms.Normalize(mean = [ 0.485, 0.456, 0.406 ],
                     std = [ 0.229, 0.224, 0.225 ]),])
```

```
params = torch.load('squeezenet1_1.pth')

### First Conv3x3 and maxpool
weights0=params['features.0.weight'].numpy()
bias0=params['features.0.bias'].numpy()

######## BLOCK 1 ########
#fire - fire - maxpool
#### FIRE 1 ####
weights1=params['features.3.squeeze.weight'].numpy()
bias1=params['features.3.squeeze.bias'].numpy()

weights2a=params['features.3.expand1x1.weight'].numpy()
bias2a=params['features.3.expand1x1.bias'].numpy()

weights2b=params['features.3.expand3x3.weight'].numpy()
bias2b=params['features.3.expand3x3.bias'].numpy()

#### FIRE 2 ####
weights3=params['features.4.squeeze.weight'].numpy()
bias3=params['features.4.squeeze.bias'].numpy()

weights4a=params['features.4.expand1x1.weight'].numpy()
bias4a=params['features.4.expand1x1.bias'].numpy()

weights4b=params['features.4.expand3x3.weight'].numpy()
bias4b=params['features.4.expand3x3.bias'].numpy()

######## BLOCK 2 ########
#fire - fire - maxpool
#### FIRE 3 ####
weights5=params['features.6.squeeze.weight'].numpy()
bias5=params['features.6.squeeze.bias'].numpy()

weights6a=params['features.6.expand1x1.weight'].numpy()
bias6a=params['features.6.expand1x1.bias'].numpy()

weights6b=params['features.6.expand3x3.weight'].numpy()
bias6b=params['features.6.expand3x3.bias'].numpy()

#### FIRE 4 ####
weights7=params['features.7.squeeze.weight'].numpy()
bias7=params['features.7.squeeze.bias'].numpy()

weights8a=params['features.7.expand1x1.weight'].numpy()
bias8a=params['features.7.expand1x1.bias'].numpy()
```

```
weights8b=params['features.7.expand3x3.weight'].numpy()
bias8b=params['features.7.expand3x3.bias'].numpy()

######## BLOCK 3 ########
#fire - fire - fire - fire
#### FIRE 5 ####
weights9=params['features.9.squeeze.weight'].numpy()
bias9=params['features.9.squeeze.bias'].numpy()

weights10a=params['features.9.expand1x1.weight'].numpy()
bias10a=params['features.9.expand1x1.bias'].numpy()

weights10b=params['features.9.expand3x3.weight'].numpy()
bias10b=params['features.9.expand3x3.bias'].numpy()

#### FIRE 6 ####
weights11=params['features.10.squeeze.weight'].numpy()
bias11=params['features.10.squeeze.bias'].numpy()

weights12a=params['features.10.expand1x1.weight'].numpy()
bias12a=params['features.10.expand1x1.bias'].numpy()

weights12b=params['features.10.expand3x3.weight'].numpy()
bias12b=params['features.10.expand3x3.bias'].numpy()

#### FIRE 7 ####
weights13=params['features.11.squeeze.weight'].numpy()
bias13=params['features.11.squeeze.bias'].numpy()

weights14a=params['features.11.expand1x1.weight'].numpy()
bias14a=params['features.11.expand1x1.bias'].numpy()

weights14b=params['features.11.expand3x3.weight'].numpy()
bias14b=params['features.11.expand3x3.bias'].numpy()

#### FIRE 8 ####
weights15=params['features.12.squeeze.weight'].numpy()
bias15=params['features.12.squeeze.bias'].numpy()

weights16a=params['features.12.expand1x1.weight'].numpy()
bias16a=params['features.12.expand1x1.bias'].numpy()

weights16b=params['features.12.expand3x3.weight'].numpy()
bias16b=params['features.12.expand3x3.bias'].numpy()

######## Classifier ########
```

```
#conv3x3 - avgpool
### Classifier Conv3x3 and avgpool
weights17=params['classifier.1.weight'].numpy()
bias17=params['classifier.1.bias'].numpy()

##### Parameters for OpenCL

conv1_weight = weights0.reshape(-1)
conv1_bias = bias0

fire1_squeeze_weight = weights1.reshape(-1)
fire1_squeeze_bias = bias1
fire1_expand1x1_weight = weights2a.reshape(-1)
fire1_expand1x1_bias = bias2a
fire1_expand3x3_weight =weights2b.reshape(-1)
fire1_expand3x3_bias = bias2b

fire2_squeeze_weight = weights3.reshape(-1)
fire2_squeeze_bias = bias3
fire2_expand1x1_weight = weights4a.reshape(-1)
fire2_expand1x1_bias = bias4a
fire2_expand3x3_weight =weights4b.reshape(-1)
fire2_expand3x3_bias = bias4b

fire3_squeeze_weight = weights5.reshape(-1)
fire3_squeeze_bias = bias5
fire3_expand1x1_weight = weights6a.reshape(-1)
fire3_expand1x1_bias = bias6a
fire3_expand3x3_weight =weights6b.reshape(-1)
fire3_expand3x3_bias = bias6b

fire4_squeeze_weight = weights7.reshape(-1)
fire4_squeeze_bias = bias7
fire4_expand1x1_weight = weights8a.reshape(-1)
fire4_expand1x1_bias = bias8a
fire4_expand3x3_weight =weights8b.reshape(-1)
fire4_expand3x3_bias = bias8b

fire5_squeeze_weight = weights9.reshape(-1)
fire5_squeeze_bias = bias9
fire5_expand1x1_weight = weights10a.reshape(-1)
fire5_expand1x1_bias = bias10a
fire5_expand3x3_weight =weights10b.reshape(-1)
fire5_expand3x3_bias = bias10b

fire6_squeeze_weight = weights11.reshape(-1)
fire6_squeeze_bias = bias11
```

```
fire6_expand1x1_weight = weights12a.reshape(-1)
fire6_expand1x1_bias = bias12a
fire6_expand3x3_weight =weights12b.reshape(-1)
fire6_expand3x3_bias = bias12b

fire7_squeeze_weight = weights13.reshape(-1)
fire7_squeeze_bias = bias13
fire7_expand1x1_weight = weights14a.reshape(-1)
fire7_expand1x1_bias = bias14a
fire7_expand3x3_weight =weights14b.reshape(-1)
fire7_expand3x3_bias = bias14b

fire8_squeeze_weight = weights15.reshape(-1)
fire8_squeeze_bias = bias15
fire8_expand1x1_weight = weights16a.reshape(-1)
fire8_expand1x1_bias = bias16a
fire8_expand3x3_weight =weights16b.reshape(-1)
fire8_expand3x3_bias = bias16b

classifier_conv_weight = weights17.reshape(-1)
classifier_conv_bias = bias17

d_conv1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=conv1_weight)
d_conv1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=conv1_bias)

d_fire1_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_squeeze_weight)
d_fire1_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire1_squeeze_bias)
d_fire1_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand1x1_weight)
d_fire1_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand1x1_bias)
d_fire1_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_weight)
d_fire1_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire1_expand3x3_bias)

d_fire2_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_squeeze_weight)
d_fire2_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire2_squeeze_bias)
d_fire2_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand1x1_weight)
```

```
d_fire2_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand1x1_bias)
d_fire2_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand3x3_weight)
d_fire2_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire2_expand3x3_bias)

d_fire3_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_squeeze_weight)
d_fire3_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire3_squeeze_bias)
d_fire3_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand1x1_weight)
d_fire3_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand1x1_bias)
d_fire3_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand3x3_weight)
d_fire3_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire3_expand3x3_bias)

d_fire4_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_squeeze_weight)
d_fire4_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire4_squeeze_bias)
d_fire4_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand1x1_weight)
d_fire4_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand1x1_bias)
d_fire4_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand3x3_weight)
d_fire4_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire4_expand3x3_bias)

d_fire5_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_squeeze_weight)
d_fire5_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire5_squeeze_bias)
d_fire5_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand1x1_weight)
d_fire5_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand1x1_bias)
d_fire5_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand3x3_weight)
d_fire5_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire5_expand3x3_bias)
```

```
d_fire6_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_squeeze_weight)
d_fire6_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire6_squeeze_bias)
d_fire6_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand1x1_weight)
d_fire6_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand1x1_bias)
d_fire6_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand3x3_weight)
d_fire6_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire6_expand3x3_bias)

d_fire7_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_squeeze_weight)
d_fire7_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire7_squeeze_bias)
d_fire7_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand1x1_weight)
d_fire7_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand1x1_bias)
d_fire7_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand3x3_weight)
d_fire7_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire7_expand3x3_bias)

d_fire8_squeeze_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_squeeze_weight)
d_fire8_squeeze_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
 ↪COPY_HOST_PTR, hostbuf=fire8_squeeze_bias)
d_fire8_expand1x1_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand1x1_weight)
d_fire8_expand1x1_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand1x1_bias)
d_fire8_expand3x3_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand3x3_weight)
d_fire8_expand3x3_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=fire8_expand3x3_bias)

d_classifier_conv_weight = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=classifier_conv_weight)
d_classifier_conv_bias = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.
 ↪mem_flags.COPY_HOST_PTR, hostbuf=classifier_conv_bias)
```

```
################################################################

for i in range(COUNT):

    imagen = np.random.rand(3, 224, 224).astype(np.float32)
    imagen = Image.fromarray(imagen,'RGB')
    imagen = transform(imagen).numpy()
    imagen = imagen[np.newaxis,:]

    #imagen = np.ones((1, 3, 224, 224))
    #print(imagen.shape)

    ### CONV1 ###
    tic=pc()

    squeeze0=nn.Conv2d(3, 64, kernel_size=3, bias=False, stride=2)
    squeeze0.weight = nn.Parameter(torch.from_numpy(weights0))
    squeeze0.bias = nn.Parameter(torch.from_numpy(bias0))

    imagen1  = torch.from_numpy(imagen).float()

    salida0=squeeze0(imagen1)
    salida0_activation=squeeze_activation(salida0)

    ### MAXPOOL1 ###

    maxpool=nn.MaxPool2d(3, stride=2)

    salida_pool1 = maxpool(salida0_activation)

    ### SQUEEZE1 ###

    squeeze1=nn.Conv2d(64, 16, kernel_size=1, bias=False)
    squeeze1.weight = nn.Parameter(torch.from_numpy(weights1))
    squeeze1.bias = nn.Parameter(torch.from_numpy(bias1))

    salida1=squeeze1(salida_pool1)
    salida1_activation=squeeze_activation(salida1)

    ### EXPAND1 ###

    squeeze2a=nn.Conv2d(16, 64, kernel_size=1, bias=False)
    squeeze2a.weight = nn.Parameter(torch.from_numpy(weights2a))
    squeeze2a.bias = nn.Parameter(torch.from_numpy(bias2a))

    squeeze2b=nn.Conv2d(16, 64, kernel_size=3, bias=False, padding=1)
    squeeze2b.weight = nn.Parameter(torch.from_numpy(weights2b))
```

```
    squeeze2b.bias = nn.Parameter(torch.from_numpy(bias2b))

    salida2a=squeeze2a(salida1_activation)
    salida2a_activation=squeeze_activation(salida2a)
    salida2b=squeeze2b(salida1_activation)
    salida2b_activation=squeeze_activation(salida2b)
    salida2_total=torch.cat([salida2a_activation,salida2b_activation], 1)

    ### SQUEEZE2 ###

    squeeze3=nn.Conv2d(128, 16, kernel_size=1, bias=False)
    squeeze3.weight = nn.Parameter(torch.from_numpy(weights3))
    squeeze3.bias = nn.Parameter(torch.from_numpy(bias3))

    salida3=squeeze3(salida2_total)
    salida3_activation=squeeze_activation(salida3)

    ### EXPAND2 ###

    squeeze4a=nn.Conv2d(16, 64, kernel_size=1, bias=False)
    squeeze4a.weight = nn.Parameter(torch.from_numpy(weights4a))
    squeeze4a.bias = nn.Parameter(torch.from_numpy(bias4a))

    squeeze4b=nn.Conv2d(16, 64, kernel_size=3, bias=False, padding=1)
    squeeze4b.weight = nn.Parameter(torch.from_numpy(weights4b))
    squeeze4b.bias = nn.Parameter(torch.from_numpy(bias4b))

    salida4a=squeeze4a(salida3_activation)
    salida4a_activation=squeeze_activation(salida4a)
    salida4b=squeeze4b(salida3_activation)
    salida4b_activation=squeeze_activation(salida4b)
    salida4_total=torch.cat([salida4a_activation,salida4b_activation], 1)

    ### MAXPOOL2 ###

    maxpool=nn.MaxPool2d(3, stride=2)

    salida_pool2 = maxpool(salida4_total)

    ### SQUEEZE3 ###

    squeeze5=nn.Conv2d(128, 32, kernel_size=1, bias=False)
    squeeze5.weight = nn.Parameter(torch.from_numpy(weights5))
    squeeze5.bias = nn.Parameter(torch.from_numpy(bias5))

    salida5=squeeze5(salida_pool2)
    salida5_activation=squeeze_activation(salida5)
```

```
### EXPAND3 ###

squeeze6a=nn.Conv2d(32, 128, kernel_size=1, bias=False)
squeeze6a.weight = nn.Parameter(torch.from_numpy(weights6a))
squeeze6a.bias = nn.Parameter(torch.from_numpy(bias6a))

squeeze6b=nn.Conv2d(32, 128, kernel_size=3, bias=False, padding=1)
squeeze6b.weight = nn.Parameter(torch.from_numpy(weights6b))
squeeze6b.bias = nn.Parameter(torch.from_numpy(bias6b))

salida6a=squeeze6a(salida5_activation)
salida6a_activation=squeeze_activation(salida6a)
salida6b=squeeze6b(salida5_activation)
salida6b_activation=squeeze_activation(salida6b)
salida6_total=torch.cat([salida6a_activation,salida6b_activation], 1)

### SQUEEZE4 ###

squeeze7=nn.Conv2d(256, 32, kernel_size=1, bias=False)
squeeze7.weight = nn.Parameter(torch.from_numpy(weights7))
squeeze7.bias = nn.Parameter(torch.from_numpy(bias7))

salida7=squeeze7(salida6_total)
salida7_activation=squeeze_activation(salida7)

### EXPAND4 ###

squeeze8a=nn.Conv2d(32, 128, kernel_size=1, bias=False)
squeeze8a.weight = nn.Parameter(torch.from_numpy(weights8a))
squeeze8a.bias = nn.Parameter(torch.from_numpy(bias8a))

squeeze8b=nn.Conv2d(32, 128, kernel_size=3, bias=False, padding=1)
squeeze8b.weight = nn.Parameter(torch.from_numpy(weights8b))
squeeze8b.bias = nn.Parameter(torch.from_numpy(bias8b))

salida8a=squeeze8a(salida7_activation)
salida8a_activation=squeeze_activation(salida8a)
salida8b=squeeze8b(salida7_activation)
salida8b_activation=squeeze_activation(salida8b)
salida8_total=torch.cat([salida8a_activation,salida8b_activation], 1)

### MAXPOOL3 ###

maxpool=nn.MaxPool2d(3, stride=2)

salida_pool3 = maxpool(salida8_total)
```

```
### SQUEEZE5 ###

squeeze9=nn.Conv2d(256, 48, kernel_size=1, bias=False)
squeeze9.weight = nn.Parameter(torch.from_numpy(weights9))
squeeze9.bias = nn.Parameter(torch.from_numpy(bias9))

salida9=squeeze9(salida_pool3)
salida9_activation=squeeze_activation(salida9)

### EXPAND5 ###

squeeze10a=nn.Conv2d(48, 192, kernel_size=1, bias=False)
squeeze10a.weight = nn.Parameter(torch.from_numpy(weights10a))
squeeze10a.bias = nn.Parameter(torch.from_numpy(bias10a))

squeeze10b=nn.Conv2d(48, 192, kernel_size=3, bias=False, padding=1)
squeeze10b.weight = nn.Parameter(torch.from_numpy(weights10b))
squeeze10b.bias = nn.Parameter(torch.from_numpy(bias10b))

salida10a=squeeze10a(salida9_activation)
salida10a_activation=squeeze_activation(salida10a)
salida10b=squeeze10b(salida9_activation)
salida10b_activation=squeeze_activation(salida10b)
salida10_total=torch.cat([salida10a_activation,salida10b_activation], 1)

### SQUEEZE6 ###

squeeze11=nn.Conv2d(384, 48, kernel_size=1, bias=False)
squeeze11.weight = nn.Parameter(torch.from_numpy(weights11))
squeeze11.bias = nn.Parameter(torch.from_numpy(bias11))

salida11=squeeze11(salida10_total)
salida11_activation=squeeze_activation(salida11)

## EXPAND6 ###

squeeze12a=nn.Conv2d(48, 192, kernel_size=1, bias=False)
squeeze12a.weight = nn.Parameter(torch.from_numpy(weights12a))
squeeze12a.bias = nn.Parameter(torch.from_numpy(bias12a))

squeeze12b=nn.Conv2d(48, 192, kernel_size=3, bias=False, padding=1)
squeeze12b.weight = nn.Parameter(torch.from_numpy(weights12b))
squeeze12b.bias = nn.Parameter(torch.from_numpy(bias12b))

salida12a=squeeze12a(salida11_activation)
salida12a_activation=squeeze_activation(salida12a)
```

```
    salida12b=squeeze12b(salida11_activation)
    salida12b_activation=squeeze_activation(salida12b)
    salida12_total=torch.cat([salida12a_activation,salida12b_activation], 1)

    ### SQUEEZE7 ###

    squeeze13=nn.Conv2d(384, 64, kernel_size=1, bias=False)
    squeeze13.weight = nn.Parameter(torch.from_numpy(weights13))
    squeeze13.bias = nn.Parameter(torch.from_numpy(bias13))

    salida13=squeeze13(salida12_total)
    salida13_activation=squeeze_activation(salida13)

    ### EXPAND7 ###

    squeeze14a=nn.Conv2d(64, 256, kernel_size=1, bias=False)
    squeeze14a.weight = nn.Parameter(torch.from_numpy(weights14a))
    squeeze14a.bias = nn.Parameter(torch.from_numpy(bias14a))

    squeeze14b=nn.Conv2d(64, 256, kernel_size=3, bias=False, padding=1)
    squeeze14b.weight = nn.Parameter(torch.from_numpy(weights14b))
    squeeze14b.bias = nn.Parameter(torch.from_numpy(bias14b))

    salida14a=squeeze14a(salida13_activation)
    salida14a_activation=squeeze_activation(salida14a)
    salida14b=squeeze14b(salida13_activation)
    salida14b_activation=squeeze_activation(salida14b)
    salida14_total=torch.cat([salida14a_activation,salida14b_activation], 1)

    ### SQUEEZE8 ###

    squeeze15=nn.Conv2d(512, 64, kernel_size=1, bias=False)
    squeeze15.weight = nn.Parameter(torch.from_numpy(weights15))
    squeeze15.bias = nn.Parameter(torch.from_numpy(bias15))

    salida15=squeeze15(salida14_total)
    salida15_activation=squeeze_activation(salida15)

    ### EXPAND8 ###

    squeeze16a=nn.Conv2d(64, 256, kernel_size=1, bias=False)
    squeeze16a.weight = nn.Parameter(torch.from_numpy(weights16a))
    squeeze16a.bias = nn.Parameter(torch.from_numpy(bias16a))

    squeeze16b=nn.Conv2d(64, 256, kernel_size=3, bias=False, padding=1)
    squeeze16b.weight = nn.Parameter(torch.from_numpy(weights16b))
    squeeze16b.bias = nn.Parameter(torch.from_numpy(bias16b))
```

```
    salida16a=squeeze16a(salida15_activation)
    salida16a_activation=squeeze_activation(salida16a)
    salida16b=squeeze16b(salida15_activation)
    salida16b_activation=squeeze_activation(salida16b)
    salida16_total=torch.cat([salida16a_activation,salida16b_activation], 1)

    ### CONV2 ###

    conv_class=nn.Conv2d(512, 1000, kernel_size=1, bias=False)
    conv_class.weight = nn.Parameter(torch.from_numpy(weights17))
    conv_class.bias = nn.Parameter(torch.from_numpy(bias17))

    salida17=conv_class(salida16_total)
    salida17_activation=squeeze_activation(salida17)

    ### AVGPOOL1 ###

    avgpool=nn.AvgPool2d(13)

    salida18=avgpool(salida17_activation)

    salida18_a_numpy=salida18.detach().numpy()

    toc=pc()

    pytorch_time += (toc - tic)

    ####### OPENCL COMPARISON #######

    ## NDRANGE

    h_sample = imagen.reshape(-1).astype(np.float32)
    d_sample = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
↪COPY_HOST_PTR, hostbuf=h_sample)

    h_result_conv = np.empty(1 * 64 * 111 * 111).astype(np.float32)
    h_result_pool1 = np.empty(1 * 64 * 55 * 55).astype(np.float32)

    h_result_fire1_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
    h_result_fire1_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
    h_result_fire2_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
    h_result_fire2_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
    h_result_pool2 = np.empty(1 * 128 * 27 * 27).astype(np.float32)

    h_result_fire3_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
    h_result_fire3_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
```

```
    h_result_fire4_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
    h_result_fire4_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
    h_result_pool3 = np.empty(1 * 256 * 13 * 13).astype(np.float32)

    h_result_fire5_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
    h_result_fire5_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
    h_result_fire6_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
    h_result_fire6_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
    h_result_fire7_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
    h_result_fire7_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)
    h_result_fire8_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
    h_result_fire8_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)

    h_result_classifier_conv = np.empty(1 * 1000 * 13 * 13).astype(np.float32)
    h_result_classifier = np.empty(1 * 1000).astype(np.float32)

    d_result_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.
↪nbytes)
    d_result_pool1 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
↪nbytes)

    d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_squeeze.nbytes)
    d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_expand.nbytes)
    d_result_fire2_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_squeeze.nbytes)
    d_result_fire2_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_expand.nbytes)
    d_result_pool2 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool2.
↪nbytes)

    d_result_fire3_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_squeeze.nbytes)
    d_result_fire3_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_expand.nbytes)
    d_result_fire4_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_squeeze.nbytes)
    d_result_fire4_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_expand.nbytes)
    d_result_pool3 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool3.
↪nbytes)

    d_result_fire5_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_squeeze.nbytes)
```

```
    d_result_fire5_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_expand.nbytes)
    d_result_fire6_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_squeeze.nbytes)
    d_result_fire6_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_expand.nbytes)
    d_result_fire7_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_squeeze.nbytes)
    d_result_fire7_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_expand.nbytes)
    d_result_fire8_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_squeeze.nbytes)
    d_result_fire8_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_expand.nbytes)

    d_result_classifier_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier_conv.nbytes)
    d_result_classifier = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier.nbytes)

    ##### DUMMY FOR REPROGRAMMING ####

    d_dummy_in = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.
↪nbytes)
    d_dummy_out = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
↪nbytes)

    maxpool_NDR(queue, (64, ), None, 111, 55, d_dummy_in, d_dummy_out)

    ###################################

    ### CONV1 ###
    tic1=pc()

    conv3x3_NDR(queue,(64, 111), None, 3, 224, 0, 2, 0, 111, d_sample,␣
↪d_conv1_weight, d_conv1_bias, d_result_conv)

    ### MAXPOOL1 ###

    maxpool_NDR(queue, (64, ), None, 111, 55, d_result_conv, d_result_pool1)

    ### SQUEEZE1 ###

    conv1x1_NDR(queue,(16, 55), None, np.int32(64/4), 55, d_result_pool1,␣
↪d_fire1_squeeze_weight, d_fire1_squeeze_bias, d_result_fire1_squeeze)
    queue.finish()
```

```
    ### EXPAND1 ###

    conv1x1_NDR(queue,(64, 55), None, np.int32(16/4), 55,␣
↪d_result_fire1_squeeze, d_fire1_expand1x1_weight, d_fire1_expand1x1_bias,␣
↪d_result_fire1_expand)
    conv3x3_NDR(queue,(64, 55), None, 16, 55, 1, 1, 64, 55,␣
↪d_result_fire1_squeeze, d_fire1_expand3x3_weight, d_fire1_expand3x3_bias,␣
↪d_result_fire1_expand)
    queue.finish()

    ### SQUEEZE2 ###

    conv1x1_NDR(queue,(16, 55), None, np.int32(128/4), 55,␣
↪d_result_fire1_expand, d_fire2_squeeze_weight, d_fire2_squeeze_bias,␣
↪d_result_fire2_squeeze)
    queue.finish()

    ### EXPAND2 ###

    conv1x1_NDR(queue,(64, 55), None, np.int32(16/4), 55,␣
↪d_result_fire2_squeeze, d_fire2_expand1x1_weight, d_fire2_expand1x1_bias,␣
↪d_result_fire2_expand)
    conv3x3_NDR(queue,(64, 55), None, 16, 55, 1, 1, 64, 55,␣
↪d_result_fire2_squeeze, d_fire2_expand3x3_weight, d_fire2_expand3x3_bias,␣
↪d_result_fire2_expand)
    queue.finish()

    ### MAXPOOL2 ###

    maxpool_NDR(queue, (128, ), None, 55, 27, d_result_fire2_expand,␣
↪d_result_pool2)

    ### SQUEEZE3 ###

    conv1x1_NDR(queue,(32, 27), None, np.int32(128/4), 27, d_result_pool2,␣
↪d_fire3_squeeze_weight, d_fire3_squeeze_bias, d_result_fire3_squeeze)
    queue.finish()

    ### EXPAND3 ###

    conv1x1_NDR(queue,(128, 27), None, np.int32(32/4), 27,␣
↪d_result_fire3_squeeze, d_fire3_expand1x1_weight, d_fire3_expand1x1_bias,␣
↪d_result_fire3_expand)
```

```
    conv3x3_NDR(queue,(128, 27), None, 32, 27, 1, 1, 128, 27,␣
↪d_result_fire3_squeeze, d_fire3_expand3x3_weight, d_fire3_expand3x3_bias,␣
↪d_result_fire3_expand)
    queue.finish()

    ### SQUEEZE4 ###

    conv1x1_NDR(queue,(32, 27), None, np.int32(256/4), 27,␣
↪d_result_fire3_expand, d_fire4_squeeze_weight, d_fire4_squeeze_bias,␣
↪d_result_fire4_squeeze)
    queue.finish()

    ### EXPAND4 ###

    conv1x1_NDR(queue,(128, 27), None, np.int32(32/4), 27,␣
↪d_result_fire4_squeeze, d_fire4_expand1x1_weight, d_fire4_expand1x1_bias,␣
↪d_result_fire4_expand)
    conv3x3_NDR(queue,(128, 27), None, 32, 27, 1, 1, 128, 27,␣
↪d_result_fire4_squeeze, d_fire4_expand3x3_weight, d_fire4_expand3x3_bias,␣
↪d_result_fire4_expand)
    queue.finish()

    ### MAXPOOL3 ###

    maxpool_NDR(queue, (256, ), None, 27, 13, d_result_fire4_expand,␣
↪d_result_pool3)

    ### SQUEEZE5 ###

    conv1x1_NDR(queue,(48, 13), None, np.int32(256/4), 13, d_result_pool3,␣
↪d_fire5_squeeze_weight, d_fire5_squeeze_bias, d_result_fire5_squeeze)
    queue.finish()

    ### EXPAND5 ###

    conv1x1_NDR(queue,(192, 13), None, np.int32(48/4), 13,␣
↪d_result_fire5_squeeze, d_fire5_expand1x1_weight, d_fire5_expand1x1_bias,␣
↪d_result_fire5_expand)
    conv3x3_NDR(queue,(192, 13), None, 48, 13, 1, 1, 192, 13,␣
↪d_result_fire5_squeeze, d_fire5_expand3x3_weight, d_fire5_expand3x3_bias,␣
↪d_result_fire5_expand)
    queue.finish()

    ### SQUEEZE6 ###
```

```
    conv1x1_NDR(queue,(48, 13), None, np.int32(384/4), 13,␣
↪d_result_fire5_expand, d_fire6_squeeze_weight, d_fire6_squeeze_bias,␣
↪d_result_fire6_squeeze)
    queue.finish()

    ### EXPAND6 ###

    conv1x1_NDR(queue,(192, 13), None, np.int32(48/4), 13,␣
↪d_result_fire6_squeeze, d_fire6_expand1x1_weight, d_fire6_expand1x1_bias,␣
↪d_result_fire6_expand)
    conv3x3_NDR(queue,(192, 13), None, 48, 13, 1, 1, 192, 13,␣
↪d_result_fire6_squeeze, d_fire6_expand3x3_weight, d_fire6_expand3x3_bias,␣
↪d_result_fire6_expand)
    queue.finish()

    ### SQUEEZE7 ###

    conv1x1_NDR(queue,(64, 13), None, np.int32(384/4), 13,␣
↪d_result_fire6_expand, d_fire7_squeeze_weight, d_fire7_squeeze_bias,␣
↪d_result_fire7_squeeze)
    queue.finish()

    ### EXPAND7 ###

    conv1x1_NDR(queue,(256, 13), None, np.int32(64/4), 13,␣
↪d_result_fire7_squeeze, d_fire7_expand1x1_weight, d_fire7_expand1x1_bias,␣
↪d_result_fire7_expand)
    conv3x3_NDR(queue,(256, 13), None, 64, 13, 1, 1, 256, 13,␣
↪d_result_fire7_squeeze, d_fire7_expand3x3_weight, d_fire7_expand3x3_bias,␣
↪d_result_fire7_expand)
    queue.finish()

    ### SQUEEZE8 ###

    conv1x1_NDR(queue,(64, 13), None, np.int32(512/4), 13,␣
↪d_result_fire7_expand, d_fire8_squeeze_weight, d_fire8_squeeze_bias,␣
↪d_result_fire8_squeeze)
    queue.finish()

    ### EXPAND8 ###

    conv1x1_NDR(queue,(256, 13), None, np.int32(64/4), 13,␣
↪d_result_fire8_squeeze, d_fire8_expand1x1_weight, d_fire8_expand1x1_bias,␣
↪d_result_fire8_expand)
```

```
   conv3x3_NDR(queue,(256, 13), None, 64, 13, 1, 1, 256, 13,␣
↪d_result_fire8_squeeze, d_fire8_expand3x3_weight, d_fire8_expand3x3_bias,␣
↪d_result_fire8_expand)
   queue.finish()

   ### CONV2 ###

   conv1x1_NDR(queue,(1000, 13), None, np.int32(512/4), 13,␣
↪d_result_fire8_expand, d_classifier_conv_weight, d_classifier_conv_bias,␣
↪d_result_classifier_conv)

   ### AVGPOOL ###

   avgpool_NDR(queue,(1000, ), None, d_result_classifier_conv,␣
↪d_result_classifier)

   cl.enqueue_copy(queue, h_result_classifier, d_result_classifier)

   queue.finish()

   veamos = h_result_classifier

   toc1=pc()

   NDRange_time += (toc1 - tic1)

   ## OUR NDRANGE

   h_sample = imagen.reshape(-1).astype(np.float32)
   d_sample = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
↪COPY_HOST_PTR, hostbuf=h_sample)

   h_result_conv = np.empty(1 * 64 * 111 * 111).astype(np.float32)
   h_result_pool1 = np.empty(1 * 64 * 55 * 55).astype(np.float32)

   h_result_fire1_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
   h_result_fire1_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
   h_result_fire2_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
   h_result_fire2_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
   h_result_pool2 = np.empty(1 * 128 * 27 * 27).astype(np.float32)

   h_result_fire3_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
   h_result_fire3_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
   h_result_fire4_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
   h_result_fire4_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
   h_result_pool3 = np.empty(1 * 256 * 13 * 13).astype(np.float32)
```

```
    h_result_fire5_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
    h_result_fire5_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
    h_result_fire6_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
    h_result_fire6_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
    h_result_fire7_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
    h_result_fire7_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)
    h_result_fire8_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
    h_result_fire8_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)

    h_result_classifier_conv = np.empty(1 * 1000 * 13 * 13).astype(np.float32)
    h_result_classifier = np.empty(1 * 1000).astype(np.float32)

    d_result_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.
↪nbytes)
    d_result_pool1 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
↪nbytes)

    d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_squeeze.nbytes)
    d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_expand.nbytes)
    d_result_fire2_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_squeeze.nbytes)
    d_result_fire2_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_expand.nbytes)
    d_result_pool2 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool2.
↪nbytes)

    d_result_fire3_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_squeeze.nbytes)
    d_result_fire3_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_expand.nbytes)
    d_result_fire4_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_squeeze.nbytes)
    d_result_fire4_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_expand.nbytes)
    d_result_pool3 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool3.
↪nbytes)

    d_result_fire5_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_squeeze.nbytes)
    d_result_fire5_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_expand.nbytes)
    d_result_fire6_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_squeeze.nbytes)
```

```
    d_result_fire6_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_expand.nbytes)
    d_result_fire7_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_squeeze.nbytes)
    d_result_fire7_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_expand.nbytes)
    d_result_fire8_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_squeeze.nbytes)
    d_result_fire8_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_expand.nbytes)

    d_result_classifier_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier_conv.nbytes)
    d_result_classifier = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier.nbytes)

    ### CONV1 ###
    tic2=pc()

    conv3x3_NDR(queue,(64, 111), None, 3, 224, 0, 2, 0, 111, d_sample,␣
↪d_conv1_weight, d_conv1_bias, d_result_conv)

    ### MAXPOOL1 ###

    maxpool_NDR(queue, (64, ), None, 111, 55, d_result_conv, d_result_pool1)

    ### SQUEEZE1 ###

    conv1x1_NDR(queue,(16, 55), None, np.int32(64/4), 55, d_result_pool1,␣
↪d_fire1_squeeze_weight, d_fire1_squeeze_bias, d_result_fire1_squeeze)
    queue.finish()

    ### EXPAND1 ###

    conv1x1_NDR(queue1,(64, 55), None, np.int32(16/4), 55,␣
↪d_result_fire1_squeeze, d_fire1_expand1x1_weight, d_fire1_expand1x1_bias,␣
↪d_result_fire1_expand)
    conv3x3_NDR(queue,(64, 55), None, 16, 55, 1, 1, 64, 55,␣
↪d_result_fire1_squeeze, d_fire1_expand3x3_weight, d_fire1_expand3x3_bias,␣
↪d_result_fire1_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE2 ###
```

```
    conv1x1_NDR(queue,(16, 55), None, np.int32(128/4), 55,␣
↪d_result_fire1_expand, d_fire2_squeeze_weight, d_fire2_squeeze_bias,␣
↪d_result_fire2_squeeze)
    queue.finish()

    ### EXPAND2 ###

    conv1x1_NDR(queue1,(64, 55), None, np.int32(16/4), 55,␣
↪d_result_fire2_squeeze, d_fire2_expand1x1_weight, d_fire2_expand1x1_bias,␣
↪d_result_fire2_expand)
    conv3x3_NDR(queue,(64, 55), None, 16, 55, 1, 1, 64, 55,␣
↪d_result_fire2_squeeze, d_fire2_expand3x3_weight, d_fire2_expand3x3_bias,␣
↪d_result_fire2_expand)
    queue.finish()
    queue1.finish()

    ### MAXPOOL2 ###

    maxpool_NDR(queue, (128, ), None, 55, 27, d_result_fire2_expand,␣
↪d_result_pool2)

    ### SQUEEZE3 ###

    conv1x1_NDR(queue,(32, 27), None, np.int32(128/4), 27, d_result_pool2,␣
↪d_fire3_squeeze_weight, d_fire3_squeeze_bias, d_result_fire3_squeeze)
    queue.finish()

    ### EXPAND3 ###

    conv1x1_NDR(queue1,(128, 27), None, np.int32(32/4), 27,␣
↪d_result_fire3_squeeze, d_fire3_expand1x1_weight, d_fire3_expand1x1_bias,␣
↪d_result_fire3_expand)
    conv3x3_NDR(queue,(128, 27), None, 32, 27, 1, 1, 128, 27,␣
↪d_result_fire3_squeeze, d_fire3_expand3x3_weight, d_fire3_expand3x3_bias,␣
↪d_result_fire3_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE4 ###

    conv1x1_NDR(queue,(32, 27), None, np.int32(256/4), 27,␣
↪d_result_fire3_expand, d_fire4_squeeze_weight, d_fire4_squeeze_bias,␣
↪d_result_fire4_squeeze)
    queue.finish()

    ### EXPAND4 ###
```

```
    conv1x1_NDR(queue1,(128, 27), None, np.int32(32/4), 27,␣
↪d_result_fire4_squeeze, d_fire4_expand1x1_weight, d_fire4_expand1x1_bias,␣
↪d_result_fire4_expand)
    conv3x3_NDR(queue,(128, 27), None, 32, 27, 1, 1, 128, 27,␣
↪d_result_fire4_squeeze, d_fire4_expand3x3_weight, d_fire4_expand3x3_bias,␣
↪d_result_fire4_expand)
    queue.finish()
    queue1.finish()

    ### MAXPOOL3 ###

    maxpool_NDR(queue, (256, ), None, 27, 13, d_result_fire4_expand,␣
↪d_result_pool3)

    ### SQUEEZE5 ###

    conv1x1_NDR(queue,(48, 13), None, np.int32(256/4), 13, d_result_pool3,␣
↪d_fire5_squeeze_weight, d_fire5_squeeze_bias, d_result_fire5_squeeze)
    queue.finish()

    ### EXPAND5 ###

    conv1x1_NDR(queue1,(192, 13), None, np.int32(48/4), 13,␣
↪d_result_fire5_squeeze, d_fire5_expand1x1_weight, d_fire5_expand1x1_bias,␣
↪d_result_fire5_expand)
    conv3x3_NDR(queue,(192, 13), None, 48, 13, 1, 1, 192, 13,␣
↪d_result_fire5_squeeze, d_fire5_expand3x3_weight, d_fire5_expand3x3_bias,␣
↪d_result_fire5_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE6 ###

    conv1x1_NDR(queue,(48, 13), None, np.int32(384/4), 13,␣
↪d_result_fire5_expand, d_fire6_squeeze_weight, d_fire6_squeeze_bias,␣
↪d_result_fire6_squeeze)
    queue.finish()

    ### EXPAND6 ###

    conv1x1_NDR(queue1,(192, 13), None, np.int32(48/4), 13,␣
↪d_result_fire6_squeeze, d_fire6_expand1x1_weight, d_fire6_expand1x1_bias,␣
↪d_result_fire6_expand)
```

```
    conv3x3_NDR(queue,(192, 13), None, 48, 13, 1, 1, 192, 13,␣
↪d_result_fire6_squeeze, d_fire6_expand3x3_weight, d_fire6_expand3x3_bias,␣
↪d_result_fire6_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE7 ###

    conv1x1_NDR(queue,(64, 13), None, np.int32(384/4), 13,␣
↪d_result_fire6_expand, d_fire7_squeeze_weight, d_fire7_squeeze_bias,␣
↪d_result_fire7_squeeze)
    queue.finish()

    ### EXPAND7 ###

    conv1x1_NDR(queue1,(256, 13), None, np.int32(64/4), 13,␣
↪d_result_fire7_squeeze, d_fire7_expand1x1_weight, d_fire7_expand1x1_bias,␣
↪d_result_fire7_expand)
    conv3x3_NDR(queue,(256, 13), None, 64, 13, 1, 1, 256, 13,␣
↪d_result_fire7_squeeze, d_fire7_expand3x3_weight, d_fire7_expand3x3_bias,␣
↪d_result_fire7_expand)
    queue.finish()
    queue1.finish()

    ### SQUEEZE8 ###

    conv1x1_NDR(queue,(64, 13), None, np.int32(512/4), 13,␣
↪d_result_fire7_expand, d_fire8_squeeze_weight, d_fire8_squeeze_bias,␣
↪d_result_fire8_squeeze)
    queue.finish()

    ### EXPAND8 ###

    conv1x1_NDR(queue1,(256, 13), None, np.int32(64/4), 13,␣
↪d_result_fire8_squeeze, d_fire8_expand1x1_weight, d_fire8_expand1x1_bias,␣
↪d_result_fire8_expand)
    conv3x3_NDR(queue,(256, 13), None, 64, 13, 1, 1, 256, 13,␣
↪d_result_fire8_squeeze, d_fire8_expand3x3_weight, d_fire8_expand3x3_bias,␣
↪d_result_fire8_expand)
    queue.finish()
    queue1.finish()

    ### CONV2 ###
```

```
   conv1x1_NDR(queue,(1000, 13), None, np.int32(512/4), 13,␣
↪d_result_fire8_expand, d_classifier_conv_weight, d_classifier_conv_bias,␣
↪d_result_classifier_conv)

   ### AVGPOOL ###

   avgpool_NDR(queue,(1000, ), None, d_result_classifier_conv,␣
↪d_result_classifier)

   cl.enqueue_copy(queue, h_result_classifier, d_result_classifier)

   queue.finish()

   veamos1 = h_result_classifier

   toc2=pc()

   Our_NDRange_time += (toc2 - tic2)

   ## SINGLE TASK

   h_sample = imagen.reshape(-1).astype(np.float32)
   d_sample = cl.Buffer(context, cl.mem_flags.READ_ONLY | cl.mem_flags.
↪COPY_HOST_PTR, hostbuf=h_sample)

   h_result_conv = np.empty(1 * 64 * 111 * 111).astype(np.float32)
   h_result_pool1 = np.empty(1 * 64 * 55 * 55).astype(np.float32)

   h_result_fire1_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
   h_result_fire1_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
   h_result_fire2_squeeze = np.empty(1 * 16 * 55 * 55).astype(np.float32)
   h_result_fire2_expand = np.empty(1 * 128 * 55 * 55).astype(np.float32)
   h_result_pool2 = np.empty(1 * 128 * 27 * 27).astype(np.float32)

   h_result_fire3_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
   h_result_fire3_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
   h_result_fire4_squeeze = np.empty(1 * 32 * 27 * 27).astype(np.float32)
   h_result_fire4_expand = np.empty(1 * 256 * 27 * 27).astype(np.float32)
   h_result_pool3 = np.empty(1 * 256 * 13 * 13).astype(np.float32)

   h_result_fire5_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
   h_result_fire5_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
   h_result_fire6_squeeze = np.empty(1 * 48 * 13 * 13).astype(np.float32)
   h_result_fire6_expand = np.empty(1 * 384 * 13 * 13).astype(np.float32)
   h_result_fire7_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
   h_result_fire7_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)
   h_result_fire8_squeeze = np.empty(1 * 64 * 13 * 13).astype(np.float32)
```

```
    h_result_fire8_expand = np.empty(1 * 512 * 13 * 13).astype(np.float32)

    h_result_classifier_conv = np.empty(1 * 1000 * 13 * 13).astype(np.float32)
    h_result_classifier = np.empty(1 * 1000).astype(np.float32)

    d_result_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.
↪nbytes)
    d_result_pool1 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
↪nbytes)

    d_result_fire1_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_squeeze.nbytes)
    d_result_fire1_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire1_expand.nbytes)
    d_result_fire2_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_squeeze.nbytes)
    d_result_fire2_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire2_expand.nbytes)
    d_result_pool2 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool2.
↪nbytes)

    d_result_fire3_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_squeeze.nbytes)
    d_result_fire3_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire3_expand.nbytes)
    d_result_fire4_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_squeeze.nbytes)
    d_result_fire4_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire4_expand.nbytes)
    d_result_pool3 = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool3.
↪nbytes)

    d_result_fire5_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_squeeze.nbytes)
    d_result_fire5_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire5_expand.nbytes)
    d_result_fire6_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_squeeze.nbytes)
    d_result_fire6_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire6_expand.nbytes)
    d_result_fire7_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_squeeze.nbytes)
    d_result_fire7_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire7_expand.nbytes)
    d_result_fire8_squeeze = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_squeeze.nbytes)
```

```
    d_result_fire8_expand = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_fire8_expand.nbytes)

    d_result_classifier_conv = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier_conv.nbytes)
    d_result_classifier = cl.Buffer(context, cl.mem_flags.WRITE_ONLY,␣
↪h_result_classifier.nbytes)

    ##### DUMMY FOR REPROGRAMMING ####

    d_dummy_in = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_conv.
↪nbytes)
    d_dummy_out = cl.Buffer(context, cl.mem_flags.WRITE_ONLY, h_result_pool1.
↪nbytes)

    maxpool_ST(queue, (1, ), None, 111, 55, 64, d_dummy_in, d_dummy_out)

    ###################################

    ### CONV1 ###
    tic3=pc()

    #first conv layer
    conv3x3_ST(queue,(1,), None, 3, 224, 0, 2, 0, 111, 64, d_sample,␣
↪d_conv1_weight, d_conv1_bias, d_result_conv)
    maxpool_ST(queue, (1, ), None, 111, 55, 64, d_result_conv, d_result_pool1)

    #block1
    conv1x1_ST(queue,(1,), None, np.int32(64/4), 55, 16, d_result_pool1,␣
↪d_fire1_squeeze_weight, d_fire1_squeeze_bias, d_result_fire1_squeeze)
    queue.finish()
    conv1x1_ST(queue1,(1,), None, np.int32(16/4), 55, 64,␣
↪d_result_fire1_squeeze, d_fire1_expand1x1_weight, d_fire1_expand1x1_bias,␣
↪d_result_fire1_expand)
    conv3x3_ST(queue,(1,), None, 16, 55, 1, 1, 64, 55, 64,␣
↪d_result_fire1_squeeze, d_fire1_expand3x3_weight, d_fire1_expand3x3_bias,␣
↪d_result_fire1_expand)
    queue.finish()
    queue1.finish()

    conv1x1_ST(queue,(1,), None, np.int32(128/4), 55, 16, d_result_fire1_expand,␣
↪d_fire2_squeeze_weight, d_fire2_squeeze_bias, d_result_fire2_squeeze)
    queue.finish()
    conv1x1_ST(queue1,(1,), None, np.int32(16/4), 55, 64,␣
↪d_result_fire2_squeeze, d_fire2_expand1x1_weight, d_fire2_expand1x1_bias,␣
↪d_result_fire2_expand)
```

```
    conv3x3_ST(queue,(1,), None, 16, 55, 1, 1, 64, 55, 64,␣
↪d_result_fire2_squeeze, d_fire2_expand3x3_weight, d_fire2_expand3x3_bias,␣
↪d_result_fire2_expand)
    queue.finish()
    queue1.finish()

    maxpool_ST(queue, (1, ), None, 55, 27, 128, d_result_fire2_expand,␣
↪d_result_pool2)

    #block2
    conv1x1_ST(queue,(1,), None, np.int32(128/4), 27, 32, d_result_pool2,␣
↪d_fire3_squeeze_weight, d_fire3_squeeze_bias, d_result_fire3_squeeze)
    queue.finish()
    conv1x1_ST(queue1,(1,), None, np.int32(32/4), 27, 128,␣
↪d_result_fire3_squeeze, d_fire3_expand1x1_weight, d_fire3_expand1x1_bias,␣
↪d_result_fire3_expand)
    conv3x3_ST(queue,(1,), None, 32, 27, 1, 1, 128, 27, 128,␣
↪d_result_fire3_squeeze, d_fire3_expand3x3_weight, d_fire3_expand3x3_bias,␣
↪d_result_fire3_expand)
    queue.finish()
    queue1.finish()

    conv1x1_ST(queue,(1,), None, np.int32(256/4), 27, 32, d_result_fire3_expand,␣
↪d_fire4_squeeze_weight, d_fire4_squeeze_bias, d_result_fire4_squeeze)
    queue.finish()
    conv1x1_ST(queue1,(1,), None, np.int32(32/4), 27, 128,␣
↪d_result_fire4_squeeze, d_fire4_expand1x1_weight, d_fire4_expand1x1_bias,␣
↪d_result_fire4_expand)
    conv3x3_ST(queue,(1,), None, 32, 27, 1, 1, 128, 27, 128,␣
↪d_result_fire4_squeeze, d_fire4_expand3x3_weight, d_fire4_expand3x3_bias,␣
↪d_result_fire4_expand)
    queue.finish()
    queue1.finish()

    maxpool_ST(queue, (1, ), None, 27, 13, 256, d_result_fire4_expand,␣
↪d_result_pool3)

    #block3
    conv1x1_ST(queue,(1,), None, np.int32(256/4), 13, 48, d_result_pool3,␣
↪d_fire5_squeeze_weight, d_fire5_squeeze_bias, d_result_fire5_squeeze)
    queue.finish()
    conv1x1_ST(queue1,(1,), None, np.int32(48/4), 13, 192,␣
↪d_result_fire5_squeeze, d_fire5_expand1x1_weight, d_fire5_expand1x1_bias,␣
↪d_result_fire5_expand)
```

```
    conv3x3_ST(queue,(1,), None, 48, 13, 1, 1, 192, 13, 192,␣
↪d_result_fire5_squeeze, d_fire5_expand3x3_weight, d_fire5_expand3x3_bias,␣
↪d_result_fire5_expand)
    queue.finish()
    queue1.finish()

    conv1x1_ST(queue,(1,), None, np.int32(384/4), 13, 48, d_result_fire5_expand,␣
↪d_fire6_squeeze_weight, d_fire6_squeeze_bias, d_result_fire6_squeeze)
    queue.finish()
    conv1x1_ST(queue1,(1,), None, np.int32(48/4), 13, 192,␣
↪d_result_fire6_squeeze, d_fire6_expand1x1_weight, d_fire6_expand1x1_bias,␣
↪d_result_fire6_expand)
    conv3x3_ST(queue,(1,), None, 48, 13, 1, 1, 192, 13, 192,␣
↪d_result_fire6_squeeze, d_fire6_expand3x3_weight, d_fire6_expand3x3_bias,␣
↪d_result_fire6_expand)
    queue.finish()
    queue1.finish()

    conv1x1_ST(queue,(1,), None, np.int32(384/4), 13, 64, d_result_fire6_expand,␣
↪d_fire7_squeeze_weight, d_fire7_squeeze_bias, d_result_fire7_squeeze)
    queue.finish()
    conv1x1_ST(queue1,(1,), None, np.int32(64/4), 13, 256,␣
↪d_result_fire7_squeeze, d_fire7_expand1x1_weight, d_fire7_expand1x1_bias,␣
↪d_result_fire7_expand)
    conv3x3_ST(queue,(1,), None, 64, 13, 1, 1, 256, 13, 256,␣
↪d_result_fire7_squeeze, d_fire7_expand3x3_weight, d_fire7_expand3x3_bias,␣
↪d_result_fire7_expand)
    queue.finish()
    queue1.finish()

    conv1x1_ST(queue,(1,), None, np.int32(512/4), 13, 64, d_result_fire7_expand,␣
↪d_fire8_squeeze_weight, d_fire8_squeeze_bias, d_result_fire8_squeeze)
    queue.finish()
    conv1x1_ST(queue1,(1,), None, np.int32(64/4), 13, 256,␣
↪d_result_fire8_squeeze, d_fire8_expand1x1_weight, d_fire8_expand1x1_bias,␣
↪d_result_fire8_expand)
    conv3x3_ST(queue,(1,), None, 64, 13, 1, 1, 256, 13, 256,␣
↪d_result_fire8_squeeze, d_fire8_expand3x3_weight, d_fire8_expand3x3_bias,␣
↪d_result_fire8_expand)
    queue.finish()
    queue1.finish()

    # classifier
    conv1x1_ST(queue,(1,), None, np.int32(512/4), 13, 1000,␣
↪d_result_fire8_expand, d_classifier_conv_weight, d_classifier_conv_bias,␣
↪d_result_classifier_conv)
```

```
    ### TEST ###
#     cl.enqueue_copy(queue, h_result_classifier_conv, d_result_classifier_conv)

    avgpool_ST(queue, (1, ), None, d_result_classifier_conv, d_result_classifier)

    cl.enqueue_copy(queue, h_result_classifier, d_result_classifier)

    veamos2 = h_result_classifier

    toc3=pc()

    ST_time += (toc3 - tic3)

    comparativa1&=np.allclose(salida18_a_numpy.reshape(-1), veamos,rtol=1e-01,␣
 ↪atol=1e-01)
    comparativa2&=np.allclose(salida18_a_numpy.reshape(-1), veamos1,rtol=1e-01,␣
 ↪atol=1e-01)
    comparativa2&=np.allclose(salida18_a_numpy.reshape(-1), veamos2,rtol=1e-01,␣
 ↪atol=1e-01)
    comparativa4&=np.allclose(veamos1, veamos2,rtol=1e-01, atol=1e-01)

    if (comparativa1 == False or comparativa2 == False or comparativa3 == False␣
 ↪or comparativa4 == False):
        print('Error en : ', i)
        break


print("comparativa (pytorch == NDRange): ",comparativa1)
print("comparativa (pytorch == Our NDRange): ",comparativa2)
print("comparativa (pytorch ==", file_dir[:-5], "): ",comparativa3)
print("comparativa (Our NDRange == Simple Task): ",comparativa4)

pytorch_time = pytorch_time/(float(COUNT))
NDRange_time = NDRange_time/(float(COUNT))
Our_NDRange_time = Our_NDRange_time/(float(COUNT))
ST_time = ST_time/(float(COUNT))

print ("tiempo en segundos con pytorch= ", pytorch_time)
print ("tiempo en segundos con opencl (NDRange)=",NDRange_time)
print ("tiempo en segundos con opencl (Our NDRange)=",Our_NDRange_time)
print ("tiempo en segundos con opencl (", file_dir[:-5], ")=",ST_time)
```

```
Reprogramming device [0] with handle 18
...
Reprogramming device [0] with handle 115
```

```
Reprogramming device [0] with handle 116
Reprogramming device [0] with handle 117

comparativa (pytorch == NDRange):  True
comparativa (pytorch == Our NDRange):  True
comparativa (pytorch == squeezenet_ST_V7 ):  True
comparativa (Our NDRange == Simple Task):  True
tiempo en segundos con pytorch=  1.1818468008400123
tiempo en segundos con opencl (NDRange)= 1.0446473936999792
tiempo en segundos con opencl (Our NDRange)= 0.999690948860025
tiempo en segundos con opencl ( squeezenet_ST_V7 )= 4.402192037339992
```