Anexo 1. Formato para los protocolos de investigación

1. Portada

Dr. Omar Piña Ramírez
Coordinador Académico de la Maestría en Sistemas Embebidos
Infotec Aguascalientes
omar.pina@infotec.mx
Tel. 55 5624 2800 ext. 6130
Doctor en Ciencias (Ingeniería Biomédica)

Este documento presenta la propuesta de proyecto titulada:

Generador de código VHDL que describe Máquinas de Soporte Vectorial optimizadas para su implementación en FPGA para aplicaciones en Internet de las Cosas.

La cual se enmaraca en la línea de investigación en sistemas embebidos para el desarrollo de internet de las cosas.

2. Antecedentes

2.1. Máquinas de Soporte Vectorial

Las máquinas de soporte vectorial (SVM por sus siglas en inglés), son métodos de *machine learning* supervisado; generalmente utilizadas para las tareas de clasificación y regresión, tanto de series de tiempo como de imágenes. Las SVM son clasificadores que discriminan entre dos clases, es decir, clasificadores binarios [**Bishop**].

La función de decisión general de una SVM es ec.1

$$y(\mathbf{x}) = sign\left(\sum_{i=1}^{M} \alpha_i y_i \mathcal{K}\left(\mathbf{x}, \mathbf{x_i}\right) + b\right), \tag{1}$$

donde los vectores $\mathbf{x}, \mathbf{x_i} \in \mathbb{R}^N$ corresponden al vector de clase desconocida que será etiquetado automáticamente por las SVM y el i-ésimo vector de soporte; $\alpha_i \in \mathbb{R}^+$ es una constante de ponderación; $y_i \in \{-1, 1\}$ es la etiqueta de clase del i-ésimo vector de soporte.

Por otra parte, la función *sign* es la que asigna la etiqueta de clase al vector $\mathbf{x}$ de acuerdo a la siguiente regla de correspondencia

$$sign(a) = \begin{cases} 1 & : \text{Si } a \geq 0 \\ 0 & : \text{Otro} \end{cases}. \tag{2}$$

Sean $\mathbf{r}, \mathbf{s} \in \mathbb{R}^N$, se denomina función kernel $\mathcal{K}$ a la transformación, generalmente no lineal, que evalua indirectamente el producto interior entre transformaciones de los vectores, es decir,

$$\mathcal{K}(\mathbf{r}, \mathbf{s}) = < \phi(\mathbf{r}), \phi(\mathbf{s}) > , \tag{3}$$

siendo $\phi(\cdot) : \mathbb{R}^N \rightarrow \mathbb{R}^K$ la transformación realizada sobre los vectores, con K generalmente distinto de N. La ventaja de utilizar la función kernel es la reducción de la complejidad computacional ya que la evaluación de la función ϕ no se realiza directamente, así mismo, en ocaciones no se conoce una formula cerrada para dicha función.

Cuando la función ϕ es la identidad, la ec.1 se reduce a

$$y(\mathbf{x}) = sign\left(\sum_{i=1}^{M} \alpha_i y_i \mathbf{x}^T \mathbf{x_i} + b\right) , \tag{4}$$

que es un producto punto entre $\mathbf{x}$ y $\mathbf{x_i}$; es por eso que a la ec.4 se le conoce como SVM de kernel lineal (LSVM).

Otro kernel ampliamente utilizado en SVM es la Función de Base Radial (RBF por sus siglas en inglés) que se describe como

$$\mathcal{K}(\mathbf{x}, \mathbf{x}_i) = exp\left\{-\gamma ||\mathbf{x}, \mathbf{x}_i||^2\right\} , \tag{5}$$

donde $\gamma \in \mathbb{R}^+$.

De esta forma, etiquetar al vector $\mathbf{x}$ con una SVM de kernel RBF o RSVM se realiza al evaluar

$$y(\mathbf{x}) = sign\left(\sum_{i=1}^{M} \alpha_i y_i \, exp\left\{-\gamma ||\mathbf{x}, \mathbf{x}_i||^2\right\} + b\right) , \tag{6}$$

Las SVM son un método de aprendizaje supervisado, es decir, se requiere de un conjunto $\mathcal{C}$ de entrenamiento que está constituido de parejas de vectores con sus correspondientes etiquetas de clase:

$$\mathcal{C} = \left\{(\mathbf{x}_j, y_j)\right\} : \mathbf{x}_j \in \mathbb{R}^N, y_j \in \{-1, 1\} . \tag{7}$$

Como resultado del entrenamiento de las SVM se obtinen las α_i que ponderan a los vectores de soporte y sus correspondientes etiquetas de clase $(\mathbf{x}_i, y_i)$ como indica la ec. 1

2.2. Implementación de SVM en FPGA

Se han realizado implementaciones en FPGA de SVM para aplicaciones médicas, procesamiento de imágenes, entre otras. La ventaja de implementar SVM en FPGA, en lugar de una arquitectura tipo von Neumann, consiste en: el alto nivel de paralelismo incluso con otros sistemas implementados en FPGA; solución de propósito específico sin necesidad de un sistema operativo en tiempo real; y con enfoque de sistemas embebidos [1, 4, 5, 11].

3. Planteamiento y definición del problema

A diferencia de un lenguaje de programación imperativo en donde se describe una serie ordenada de sentancias destinadas a resolver un problema, el lenguaje de descripción de hardware VHDL, que es en donde se describen las interconexiones del FPGA, intrinsecamente está relacionado a la conexión compuertas lógicas y flip-flops que se comportarán como la máquina de estados finitos para la que fueron diseñados, lo cual, es un enfoque completamente distinto a una arquitectura von Neumann convencional. Posiblemente, este cambio de paradigma es lo que dificulte la descripción óptima de algoritmos en FPGA.

Si bien es cierto que existen implementaciones FPGA de SVM, estas son para un uso particular y únicamente expertos en lenguajes de descripción de hardware los pueden reutilizar [2-4, 6-11].

Por tal motivo, en un trabajo previo se realizó un generador automático de código Handel-C que describía LSVM para implementarse en FPGA para aplicaciones en segmentación de imágenes cerebrales [8]; sin embargo, dicho trabajo ha quedado obstoleto ya que el lenguaje Handel-C ya no es vigentes.

Por tal motivo se propone dar continuidad a dicho proyecto al desarrollar un prototipo de framework que permita generar código VHDL que describa SVM de kernel lineal y RBF, los cuales son los más utilizados en las aplicaciones.

4. Justificación o sustentación

El desarrollo de este framework se propone en tres fases, una por año. En la primera fase se propone implementar el sistema que entrene las SVM, evalue su desempeño y posteriormente genere el código VHDL de la implementación. Una vez realizado, el framework deberá ser capaz de evaluar los desempeños de clasificación sobre datos no vistos entre las implementaciones: punto fijo (FPGA), punto flotante (computadora en donde se realizó el entrenamiento). Esta perspectiva, permite que las aplicaciones del framework sean de diversa ídole; tanto para señales como para imágenes. En una segunda fase este generador de código incorporará *shield* para cloudino lo cual permitirá tener sistemas de reconocimiento de patrones embebidos que reunan SVM, FPGA e IoT, lo cual, es una de las líneas estratégicas de Infotec. Finalmente, en la tercera etapa, se proyecta la aplicación del framework con IoT en entornos de procesamiento masivo-paralelo en tiempo real.

5. Objetivos (general y particulares)

El objetivo general del proyecto es desarrollar un prototipo de framework que permita entrenar modelos de SVM para posteriormente implementarlos en FPGA a partir de la generación automática del código VHDL optimizado que los describe. Dicho framwork también contemplará herramientas de análisis de desempeño de las implementaciones FPGA.

Para lograr dicho objetivo, se plantean tres objetivos particulares:

- Desarrollar un prototipo de framework open source que, dado un conjunto de datos etiquetados para una aplicación dada: 1) entrene, optimice y evalúe un modelo SVM

- Analizar los modelos de SVM y pra generar el código optimizado de VHDL que los implementa en FPGA
- Comparar el desempeño de las implementaciones con representación de punto fijo (FPGA) respecto a las de punto flotante (computadora)

6. Hipótesis de trabajo

Se podrá generar código VHDL, y por consiguiente implementaciones FPGA en representación de punto fijo de modelos de SVM entrenados con punto flotante, de tal forma que ambas implementaciones generen resultados de clasificación equivalentes, es decir que no exista diferencia estadísticamente significativa entre los desempeños de ambas implementaciones a pesar de la reducción de la precisión en la representación debida al punto fijo. Lo anterior se estima sea cierto en por lo menos el 80 % de los conjuntos de prueba que se analizarán.

7. Elementos de valor

Dicho desarrollo proveerá de una herramienta open source que permitirá la implementación del método de aprendizaje maquinal SVM directamente en FPGA sin que el usuario sea un experto en cómputo reconfigurable y FPGA, garantizando una implementación optimizada, personalizada, y altamente paralelizable.

8. Alcance

Solo se podrá utilizar cuando SVM sea un buen clasificador para la aplicación, ya que la implementación en FPGA será óptima, pero heredará las mismas cualidades de discriminación que su contraparte en punto flotante. Por otra parte, se cuenta con la limitación del número de SVM que se podrán implementar, en este respecto, se podrán implementar tantas SVM como recursos disponibles tenga el FPGA diponibles.

9. Beneficios

Infotec tendrá un framework que le permita realizar implementaciones de SVM para las aplicaciones que crea pertinentes, adicionalmente, la conceptualización del framework lo hace extendible a otras plataformas y métodos de aprendizaje maquinal. Y esto, en etapas posteriores, podrá integrarse a IoT, en particular, Cloudino que es un producto clave de Infotec.

10. Índice tentativo

1. Introducción
2. Máquinas de Soporte Vectorial
3. Implementación de Máquinas de Soporte Vectorial en FPGA
4. Generador de código VHDL que implementa Máquinas de Soporte Vectorial

5. Evaluación de Implementaciones

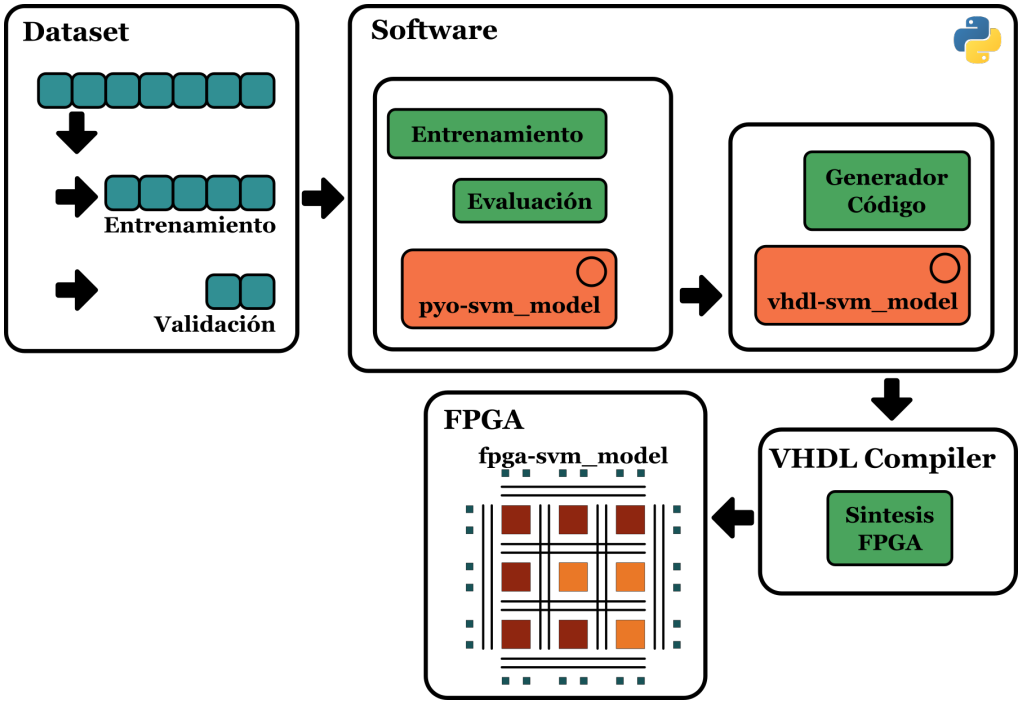

Figura 1: Diagrama de los bloques de los que estará compuesto el generador de código.

11. Cronograma de actividades y de entregables

En la Figura 1 se esboza la metodología propuesta:

Dataset. Un conjunto de datos se partirá para entrenamiento y validación con datos no vistos

Software (punto flotante). Se entrenarán las SVM lineales o RBF según corresponda, y se optimizarán los hiperparámetros. Se obtendrá el modelo de la SVM (pyo-svm_model). Un algoritmo generará y optimizará el código VHDL que describa dicho modelo.

Software (punto fijo). El resultado será una descripción de hardware en punto fijo (vhdl-svm_model), que luego podrá ser sintenizada para una arquitectura FPGA (archivo BIT).

Hardware (punto fijo). La FPGA configurada (fpga-svm_model) con el archivo BIT para que esta implementela la SVM del modelo pyo-svm_model.

Por otra parte, se realizará un contraste estadístico Friedman con post hoc Hommel para evaluar si existe una diferencia estadísticamente significativa entre las implementaciones a pesar de la diferencia de precisiones punto flotante/punto fijo. En la Figura 2 se ejemplifica dicho proceso, tomando en cuenta la clasificación del conjunto de datos no vistos.

De la breve descripción metodológica se desglosan las tareas y entregables que se muestran en la Figura 3. Estas actividades, correspondientes a la primera etapa del proyecto a

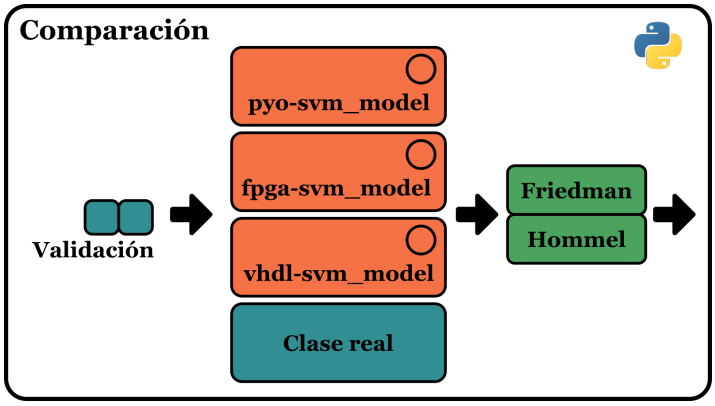

Figura 2: Esquema de validación de la implementación en hardware (fpga-svm_model) de una SVM dada.

realizarse durante 2020, se detallan a continuación:

1. Búsqueda sistemática de métodos de implementación de kernel no lineal en VHDL-FPGA y en genenar de métodos de clasificación y extracción de rasgos en FPGA excluyendo el entrenamiento.
2. Búsqueda sistemática de sistemas para generar código a partir de un modelo genérico de la implementación.
3. Modelado UML del framework para generar código para la conceptualización teórica, el análisis del alcance y las limitaciones del generador de código y sus aplicaciones una vez implementado y evaluado en el FPGA.
4. Implementación de entrenamiento y evaluación de SVM de kernel lineal. Etapa fundamental para la contrucción del modelo general de SVM de kernel lineal ya que permitirá el análisis de los elementos constantes y variables entre los distintos conjuntos de datos.
5. Creación de un modelo genérico de SVM de kernel lineal en VHDL.
6. Desarrollo del algoritmo de generación de código para SVM de kernel lineal; tomando en cuenta que las SVM se implementan siguiendo una misma función, los parámetros variables son: los vectores de soporte, con sus correspondientes α's y y's, el algoritmo que se propondrá deberá adaptar un modelo genérico de SVM de kernel lineal al modelo que se requiera implementar, así mismo se deberá hacer un análisis del detrimento del desempeño de la clasificación debido al punto fijo para así establecer la representación adecuada en cada caso.
7. Desarrollo de la herramienta de comparación entre la implementación de la SVM en punto flotante (computadora) y la de punto fijo (FPGA). Esto permitirá realizar pruebas que permitan corroborar la no diferencia estadística entre las implementaciones.
8. Realización de las pruebas de comparación y reporte de avances. Lo anterior implica la evaluación integral de la implementación tomando en cuenta las bases de datos de

Fase/ 1 de 3	Año: 2020												Productos entregables
Actividad/Mes	1	2	3	4	5	6	7	8	9	10	11	12	
Búsqueda sistemática de métodos de implementación de kernel no lineal en VHDL-FPGA													Reporte de revisión y análisis de referencias
Búsqueda sistemática de sistemas para generar código													Reporte de revisión y análisis de referencias
Modelado UML del framework para generar código													Modelos UML del sistema (StarUML Project)
Implementación de entrenamiento y evaluación de SVM de kernel lineal													Scripts python evaluados y documentados en repositorio
Creación de un modelo genérico de SVM de kernel lineal en VHDL													Modelo genérico VHDL de SVM de kernel lineal
Desarrollo del algoritmo de generación de código para SVM de kernel lineal													Scripts en python que transforman modelos SVM de kernel lineal a su descripción VHDL
Desarrollo de la herramienta de comparación entre la implementación de la SVM en punto flotante (computadora) y la de punto fijo (FPGA)													Reporte de métricas de comparación
Realización de las pruebas de comparación y reporte de avances													Un artículo sometido a un congreso internacional especializado en el tema
Implementación de entrenamiento y evaluación de SVM de kernel RBF													Scripts python evaluados y documentados en repositorio
Desarrollo del modelo genérico de la SVM de kernel RBF en VHDL													Modelo genérico VHDL de SVM de kernel RBF
Desarrollo del algoritmo de generación de código para SVM de kernel RBF													Scripts en python que transforman modelos SVM de kernel RBF a su descripción VHDL
Realización de las pruebas de comparación y reporte de avances													Reporte de métricas de comparación
Someter artículo a revista especializada													Artículo sometido a revista JCR
Reporte anual													- Reporte anual de actividades - Publicación de repositorios y bases de datos

Figura 3: Cronograma de actividades para la primera fase del proyecto a llevarse a cabo durante el año 2020.

la UCI *Machine Learning* para las cuales las SVM de kernel lineal discriminen por lo menos con el 85 % de *accuracy* en la representación de punto flotante.

9. Implementación de entrenamiento y evaluación de SVM de kernel RBF. Similar a la actividad 4, pero considerando el kernel no lineal RBF.
10. Desarrollo del modelo genérico de la SVM de kernel RBF en VHDL. Similar a la actividad 5, pero con kernel RBF
11. Desarrollo del algoritmo de generación de código para SVM de kernel RBF. Similar a la actividad, pero con kernel RBF en el cual se incluye el parametro γ y la evaluación de una función exponencial.
12. Realización de las pruebas de comparación y reporte de avances. Evaluación de las SVM con kernel RBF implementadas en FPGA y comparar las métricas de desempeño con respecto a las obtenidas en punto flotante.
13. Someter artículo a revista especializada.
14. Reporte anual.

12. Propiedad intelectual de los entregables y productos generados en el proyecto.

Para esta etapa del proyecto no hay propiedad intelectual, ya que es el desarrollo conceptual del prototipo así como su validación. De esta etapa se desprenderán: a) Artículo JCR especializada en FPGA y embebidos, b) Artículo para congreso, c) Notas de aplicación y d) Liberación de código.

13. Referencias

[1] Shereen Afifi, Hamid GholamHosseini y Roopak Sinha. "A system on chip for melanoma detection using FPGA-based SVM classifier". En: *Microprocessors and Microsystems* 65 (mar. de 2019), págs. 57-68. DOI: 10.1016/j.micpro.2018.12.005.

[2] Ons Boujelben y Mohammed Bahoura. "Efficient FPGA-based architecture of an automatic wheeze detector using a combination of MFCC and SVM algorithms". En: *Journal of Systems Architecture* 88 (ago. de 2018), págs. 54-64. DOI: 10.1016/j.sysarc.2018.05.010.

[3] E. Canto-Navarro, M. Lopez-Garcia y R. Ramos-Lara. "Floating-point accelerator for biometric recognition on FPGA embedded systems". En: *Journal of Parallel and Distributed Computing* 112 (feb. de 2018), págs. 20-34. DOI: 10.1016/j.jpdc.2017.09.010.

[4] H.K. Chatterjee, R. Gupta y M. Mitra. "Real Time P and T Wave Detection from Ecg using FPGA". En: *Procedia Technology* 4 (2012), págs. 840-844. DOI: 10.1016/j.protcy.2012.05.138.

[5] K. Divya Krishna y col. "Computer Aided Abnormality Detection for Kidney on FPGA Based IoT Enabled Portable Ultrasound Imaging System". En: *IRBM* 37.4 (ago. de 2016), págs. 189-197. DOI: 10.1016/j.irbm.2016.05.001.

[6] Bernd Lesser, Manfred Mücke y Wilfried N. Gansterer. "Effects of Reduced Precision on Floating-Point SVM Classification Accuracy". En: *Procedia Computer Science* 4 (2011), págs. 508-517. DOI: 10.1016/j.procs.2011.04.053.

[7] Hongxing Peng y col. "Research on Multi-class Fruits Recognition Based on Machine Vision and SVM". En: *IFAC-PapersOnLine* 51.17 (2018), págs. 817-821. DOI: 10.1016/j.ifacol.2018.08.094.

[8] Omar Pina-Ramirez, Raquel Valdes-Cristerna y Oscar Yanez-Suarez. "An FPGA Implementation of Linear Kernel Support Vector Machines". En: *2006 IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig 2006)*. IEEE, sep. de 2006. DOI: 10.1109/reconf.2006.307784.

[9] Jens Rettkowski, Andrew Boutros y Diana Göhringer. "HW/SW Co-Design of the HOG algorithm on a Xilinx Zynq SoC". En: *Journal of Parallel and Distributed Computing* 109 (nov. de 2017), págs. 50-62. DOI: 10.1016/j.jpdc.2017.05.005.

[10] Shady Soliman y col. "FPGA implementation of dynamically reconfigurable IoT security module using algorithm hopping". En: *Integration* 68 (sep. de 2019), págs. 108-121. DOI: 10.1016/j.vlsi.2019.06.004.

[11] Xuhui Yang y col. "FPGA-based approximate calculation system of General Vector Machine". En: *Microelectronics Journal* 86 (abr. de 2019), págs. 87-96. DOI: 10.1016/j.mejo.2019.02.018.