UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

ESTUDIO DE LA VIABILIDAD DE UN SISTEMA DE DETECCIÓN DE BUITRES EMBARCADO BASADO EN EL ANÁLISIS DE IMÁGENES EN ALTA RESOLUCIÓN.

Trabajo final del
Grado en Ingeniería Aeroespacial

Autor
Humberto Casañez Gaspar

Tutor
Ignacio Despujol Zabala

CURSO ACADÉMICO: 2019/2020

Agradecimientos

En esta sección me gustaría dar las gracias a todas aquellas personas que han estado a mi lado durante estos cuatro largos años de carrera o me han dedicado un momento, a todas aquellas que me han ayudado desinteresadamente o me han pedido ayuda, a aquellos que me han visto crecer y han crecido conmigo o los que me han sujetado al caer... En definitiva, a todos aquellos que de una forma u otra me han ayudado a llegar a ser quien hoy en día soy.

En primer lugar, me gustaría agradecer a D. Ignacio Despujol Zabala, la confianza que ha depositado en mi, dándome la oportunidad de realizar este trabajo de fin de grado y todo el tiempo que me ha dedicado para que esto saliese adelante. Ha sido un gran tutor durante estos cuatro largos meses, así como un gran profesor, capaz de motivar e interesar a sus alumnos.

También me gustaría agradecer a mi familia todo el apoyo que siempre me han brindado, por su comprensión y su cariño y por haberme ayudado a llegar hasta esta meta que hace cuatro años parecía tan difícil.

Por último me gustaría agradecerles a mis compañeros y amigos, por darme algo más valioso que un título universitario.

A Marta, por estar siempre a mi lado riendo y llorando conmigo.

A Mario, Nacho, Dani, Miguel y el resto de mis compañeros de clase, por hacerme más amenas, las infinitas horas sentado tras un pupitre.

A Enrique, Pepe, Matute, Tony, Manolo y Kevin por hacerme sentir en cualquier pequeña habitación de residencia como en mi casa.

Gracias a todos.

Resumen

En el presente documento se pretende dar solucionar al problema de las colisiones entre aeronaves y aves, mediante la utilización de un sistema de detección de aves basado en el análisis de imágenes, utilizando para ello una red neuronal clasificadora.

Con este objetivo en mente, se ha llevado a cabo un estudio bibliográfico de las posibles soluciones existentes, así como una recopilación de información, de forma que fuese posible sentar un marco teórico básico sobre las redes neuronales y la inteligencia artificial, del que partir. Consiguiendo de esta forma, un modelo de red neuronal clasificadora a partir de uno de los proyectos utilizados.

Para estudiar la viabilidad de este proyecto se han realizado una serie de cálculos preliminares, basados en las velocidades, tanto de la aeronave como del ave, el campo de visión de la cámara y el tamaño de los buitres, con la finalidad de obtener factores relevantes para el estudio que se ha propuesto, como son la máxima distancia de detección del ave por parte de la red clasificadora o el mínimo tiempo de colisión a dicha distancia. A continuación se han utilizado estos datos para, teniendo en cuenta la definición de las cámaras disponibles en el mercado, determinar los tamaños en píxels de las imágenes con las que debe trabajar la red neuronal. Se ha adquirido una FPGA encontrando la necesidad de un entorno de desarrollo basado en este tipo de hardware de bajo coste del fabricante Xilix, denominado PYNQ-Z2 que nos permite implementar redes neuronales aceleradas con hardware de bajo nivel utilizando un lenguaje de propósito general como es Python.

Posteriormente, se ha trabajado con la FPGA. Para ello, se ha seguido un proyecto que implementa una red neuronal similar a la que se ha considerado óptima para la detección de aves, en este tipo de sistema. El problema es que la implementación descrita en la bibliografía para llevar a cabo la aceleración está hecha con un framework y unas librerías que utilizan una versión de Python anterior a la disponible para el sistema adquirido y que no disponen todavía de actualización para la versión actual, por lo que ha sido necesario dedicar mucho tiempo a modificar la instalación de este framework y las librerías dedicadas a la utilización de las redes neuronales y el acceso al software de aceleración de la FPGA. Durante este proceso se han encontrado una gran cantidad de errores, que han requerido una gran parte del tiempo para ser solventados. Al final se ha conseguido acelerar la red neuronal de detección de imágenes que la literatura indica como más adecuada sin llegar a implementar la clasificación de aves debido a la falta de tiempo. El estudio ha permitido también detectar qué algoritmos adicionales será necesario implementar para que la red neuronal pueda trabajar y las limitaciones del hardware del entorno de desarrollo de bajo coste utilizado para las pruebas, lo que permitirá avanzar el proyecto en fases posteriores.

De esta forma, aunque todavía queda trabajo para completar el estudio de viabilidad de este tipo de sistemas embarcados, mediante este proyecto se han hallado indicios de que estos pueden ser viables, puesto que se ha sido posible su aceleración así como estudiar la viabilidad de varios algoritmos de clasificación, detección y tracking. Creando así, una base teórica y práctica sobre la que se podrán apoyar futuros proyectos.

Resum

En el present document es pretén donar solució al problema de les col·lisions entre aeronaus i aus, per mitjà de la utilització d'un sistema de detecció d'aus basat en l'anàlisi d'imatges, utilitzant per a això una xarxa neuronal classificadora.

Amb aquest objectiu en ment, s'ha dut a terme un estudi bibliogràfic de les possibles solucions existents, així com una recopilació d'informació, de manera que fóra possible assentar un marc teòric bàsic sobre les xarxes neuronals i la intel·ligència artificial, del que partir. Aconseguint d'aquesta manera, un model de xarxa neuronal classificadora a partir d'un dels projectes utilitzats.

Per a estudiar la viabilitat d'aquest projecte, s'han realitzat una sèrie de càlculs preliminars, basats en les velocitats, tant de l'aeronau com de l'au, el camp de visió de la càmera i la grandària dels voltors, amb la finalitat d'obtindre factors rellevants per a l'estudi que s'ha proposat, com són la màxima distància de detecció de l'au per part de la xarxa classificadora o el mínim temps de col·lisió a la distància anomenada. A continuació s'han utilitzat aquestes dades per a, tenint en compte la definició de les camares disponibles en el mercat, determinar les grandàries en píxels de les imatges amb què ha de treballar la xarxa neuronal. S'ha adquirit una FPGA trobant la necessitat d'un entorn de desenvolupament basat en aquest tipus de hardware de baix cost del fabricant Xilix, denominat PYNQ-Z2 que ens permet implementar xarxes neuronals accelerades amb hardware de baix nivell utilitzant un llenguatge de propòsit general com és Python.

Posteriorment, s'ha treballat amb la FPGA. Per a això, s'ha seguit un projecte que implementa una xarxa neuronal semblant a què s'ha considerat òptima per a la detecció d'aus, en este tipus de sistema. El problema és que la implementació descrita en la bibliografia per a dur a terme l'acceleració està feta amb un framework i unes llibreries que utilitzen una versió de Python anterior a la disponible per al sistema adquirit i que no disposen encara d'actualització per a la versió actual, per la qual cosa ha sigut necessari dedicar molt de temps a modificar la instal·lació d'aquest framework i les llibreries dedicades a la utilització de les xarxes neuronals i l'accés al programari d'acceleració de la FPGA. Durant este procés s'han trobat una gran quantitat d'errors, que han requerit una gran part del temps per a ser resolts. Al final s'ha aconseguit accelerar la xarxa neuronal de detecció d'imatges que la literatura indica com més adequada sense arribar a implementar la classificació d'aus degut a la falta de temps. L'estudi ha permés també detectar quins algoritmes addicionals serà necessari implementar perquè la xarxa neuronal puga treballar i les limitacions del hardware de l'entorn de desenvolupament de baix cost utilitzat per a les proves, la qual cosa permetrà avançar el projecte en fases posteriors.

Així, tot i que encara queda treball per a completar l'estudi de viabilitat d'aquest tipus de sistemes embarcats, per mitjà d'este projecte s'han trobat indicis de que aquest sistema pot ser viable, ja que s'ha aconseguit accelerar este tipus de sistemes així com estudiar la viabilitat de diversos algoritmes de classificació, detecció i tracking. Creant així, una base teòrica i pràctica sobre la qual es podran recolzar futurs projectes.

Abstract

The aim of this document is to solve the problem of collisions between aircraft and birds, through the use of a bird detection system based on image analysis, using a classifier neural network.

With this objective in mind, a bibliographic study of the possible existing solutions as well a deep research have been carried out, allowing to establish a basic theoretical framework on neural networks and artificial intelligence. Obtaining in this way, a classifier neural network model from one of the projects used.

To study the viability of this project, a series of preliminary calculations have been made, based on the speeds of both the aircraft and the bird, the camera's field of view and the size of the vultures, in order to obtain relevant factors. for the study that has been proposed, such as the maximum detection distance of the bird by the sorting network or the minimum collision time at said distance. These data have then been used to determine the sizes in pixels of the images with which the neural network must work, taking into account the definition of the cameras available on the market. An FPGA has been acquired finding the need for a development environment based on this type of low-cost hardware from the manufacturer Xilix, called PYNQ-Z2 that allows us to implement accelerated neural networks with low-level hardware using a general-purpose language such as Python.

After that, it has been necessary to use the FPGA. For this, a project that implements a neural network similar to the one considered optimal for the detection of birds in this type of system, has been followed. The problem is that the implementation described in the bibliography to carry out the acceleration is made with a framework and some libraries that use a Python version older than the one available for the acquired system and that do not yet have an update for the current version, therefore, it has been necessary to spend a lot of time modifying the installation of this framework and the libraries dedicated to the use of neural networks and the access to the FPGA acceleration software. During this process, a large number of errors have been found, which have required a large part of the time to be solved. In the end, it has been possible to accelerate the neural network for image detection that the literature indicates as the most appropriate without actually implementing the bird classification one due to lack of time. The study has also made it possible to detect which additional algorithms will need to be implemented so that the neural network can work and the hardware limitations of the low-cost development environment used for testing, which will allow the project to advance in later phases.

In this way, although there is still work to complete the feasibility study of this type of on-board systems, through this project it has been found that this system may be feasible, since it has been possible to accelerate this type of systems and to study the feasibility of various classification, detection and tracking algorithms. Creating thus, a theoretical and practical base on which future projects can be supported.

Índice

Lista de figuras III

Glosario IV

I. Memoria 1

1. Introducción 2
1.1. Contexto . . . 2
1.2. Objetivos . . . 5
1.3. Estado del arte . . . 5
1.3.1. Clasificación y seguimiento de aves mediante redes neuronales . . . 5
1.3.2. Implementación en una FPGA . . . 8

2. Antecedentes 11
2.1. Redes neuronales . . . 11
2.1.1. Redes neuronales convolucionales . . . 12
2.1.2. LeNet y ResNet . . . 15
2.2. Fiel-Programmable Gate Arrays (FPGAs) . . . 16
2.2.1. PYNQ-Z2 . . . 16
2.3. Métodos para el seguimiento de objetos . . . 18
2.3.1. Sustracción de fondo dinámico . . . 18
2.3.2. Sliding window . . . 19

3. Desarrollo del proyecto 21
3.1. Cálculos preliminares . . . 21
3.2. Preparación de la PYNQ-Z2 . . . 24
3.2.1. Instalación y configuraciones previas . . . 24
3.2.2. Instalación del framework . . . 25
3.3. Tests de funcionamiento y aceleración . . . 27
3.4. Transferencia del modelo . . . 30
3.5. Errores encontrados durante el proyecto . . . 30
3.5.1. Errores durante la instalación del framework . . . 30
3.5.2. Errores encontrados durante los test . . . 32
3.6. Conclusiones . . . 35

II. Pliego de condiciones 37

III. Presupuesto 42

IV. Anexo de código 46

V. Anexo de planos **57**

Bibliografía **60**

Índice de figuras

1. Número de casos de “bird-strike” reportados por región [1] . . . 3
2. Concentración de casos de “bird-strike” reportados por fase de vuelo [1] . . . 3
3. Esquema simplificado de una CNN [6] . . . 5
4. Fragmento del dataset del Prof. Yoshihashi y su equipo [7] . . . 6
5. Técnica de “data augmentation” de variación de la temperatura del color [10] . . . 7
6. Resultados de la detección utilizando la sustracción de fondo dinámico [14] . . . 10
7. Estructura interna de una neurona artificial . . . 11
8. Estructura de una red neuronal con una única capa oculta . . . 12
9. Función ReLU . . . 13
10. Estructura de una red neuronal convolucional . . . 14
11. Clasificación de objetos mediante CNN . . . 14
12. Clasificación de una aeronave y un ave en vuelo mediante CNN . . . 15
13. Estructura de una LeNet [20] . . . 15
14. Estructura de una ResNet [21] . . . 16
15. Componentes de la PYNQ-Z2 . . . 17
16. Seguimiento de un helicóptero mediante sustracción de fondo dinámico [24] . . . 19
17. Creación de un algoritmo de sliding window [25] . . . 20
18. Detección y tracking de coches mediante sliding window [26] . . . 20
19. Tiempos de colisión a $250\ m$ . . . 22
20. Tiempos de colisión a $500\ m$ . . . 22
21. Tiempos de colisión a $1000\ m$ . . . 22
22. Campo de visión de la cámara en función de la distancia al ave . . . 23
23. Tamaño en píxeles del ave en función de la distancia y la resolución . . . 23
24. Software balenaEtcher . . . 24
25. Página principal de Jupyter Notebooks para la PYNQ-Z2 . . . 25
26. Celda 3 modificada sin aceleración . . . 28
27. Celda 6 modificada sin aceleración . . . 28
28. Celda 17 sin aceleración . . . 29
29. Predicciones LeNet sin aceleración . . . 29
30. Tiempos de procesameinto . . . 29
31. Código original para Python 3.4 . . . 31
32. Código modificado para Python 3.6 . . . 31
33. Modificación de código para solucionar el error de OpenBlas . . . 32
34. Celda 3 con aceleración . . . 32

III-1. Subtotal de coste del material utilizado . . . 43
III-2. Subtotal de coste del personal . . . 44
III-3. Subtotal de costes derivados . . . 44
III-4. Total de costes del proyecto . . . 45
III-5. Total de emisiones de CO_2 atribuidas al desarrollo del proyecto . . . 45

IV-1. Código de Error:Unable to locate package (Comando: sudo apt-get install) . . . 54
IV-2. Código de Error: pyconfig.h: No such file or directory (Comando: make all) . . . 54
IV-3. Código de Error: invalid syntax (Comando: import caffe) . . . 55
IV-4. Código de Error: Module not found (Comando: import im2col_lasagne_cython.pyx) . . . 55
IV-5. Código de Error: Module not found (Comando: import acc8.pyx) . . . 55
IV-6. Código de Error: Import error: undefined symbol (Comando: import acc8.pyx) . . . 55
IV-7. Código de Error: Module not found (Comando: from pynq.drivers import DMA) . . . 56
IV-8. Código de Error: KeyError *SEG_axi_switch_0_Reg* (Comando: import conv_fpga) . . . 56

Glosario

AI Artificial Intelligence.

CNN Convolutional Neural Network.

CO2 Carbon Dioxide.

CPU Central Processing Unit.

DAG Directed Acyclic Graph.

FOV Field of View.

FPGA Field-Programmable Gate Array.

HDMI High Definition Multimedia Interface.

HRSC High Resolution Stereo Camera.

IC Integrated Circuit.

IVA Impuesto sobre el Valor Añadido.

LDS Liner Dynamic System.

MNIST dataset Modified National Institute of Standards and Technology dataset.

NN Neural Network.

OACI Organización de Aviación Civil Internacional.

ReLU Rectifier Linear Unit.

ResNet Deep Residual Network.

SD Secure Digital.

SG Stauffer and Grimson.

TFG Trabajo de Fin de Grado.

USB Universal Serial Bus.

WFOV Wide Field of View.

I MEMORIA

1. Introducción

1.1. Contexto

La aviación es un medio de transporte rápido y cada vez más económico, pero principalmente destaca por su seguridad, pues presenta los requerimientos más extrictos de entre todos los medios de transporte con respecto a este campo, puesto que los accidentes tienen un alto coste de vidas y bienes materiales.

Como se ha mencionado, las medidas de seguridad en las aeronaves son extremadamente altas, sin embargo hay factores imprevisibles que pueden ocasionar accidentes aumentando, de esta forma, su riesgo. Un problema recurrente en el sector de la aviación son las aves y los accidentes que se producen a causa de estas son conocidos como “bird-strike”, las consecuencias de los mismos pueden ir desde un aterrizaje forzoso con daños materiales, hasta la pérdida de vidas humanas.

El caso más sonado recientemente es el del vuelo 1549 de American Airlines (15 de enero de 2009), procedente de Nueva York y con destino Seattle. La aeronave era un Airbus A320 que transportaba a 5 tripulantes y 150 pasajeros y que se encontró con una bandada de barnaclas canadienses (Branta canadensis) poco después de despegar, las aves se colaron en los motores de la aeronave provocando el fallo de ambos motores y el amerizaje forzado de la aeronave en el río Hudson (Nueva York). Afortunadamente, las pérdidas fueron solo de carácter material.

Debido a la importancia de este tipo de altercados, la OACI ha recopilado los informes, desde 2008 hasta 2015, procedentes de un amplio conjunto de países para crear el sistema IBIS [1]. En la Figura 1, se muestra el número de casos de “bird-strike” recogidos, como se puede observar durante esos siete años el número de incidentes asciende hasta más de 70000, concentrándose en las regiones de Europa y Norteamérica.

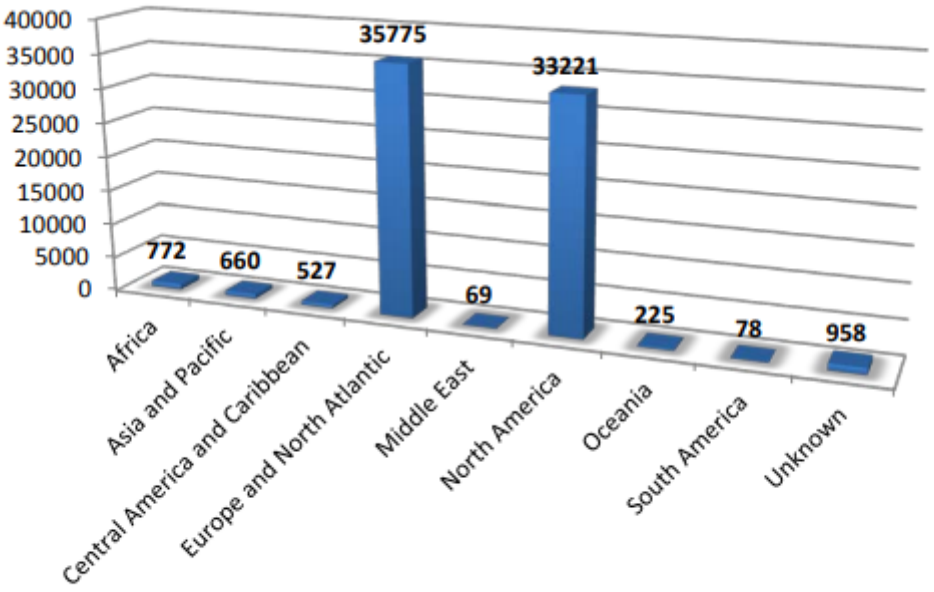


Figura 1: Número de casos de “bird-strike” reportados por región [1]

En la Figura 2, se muestra la distribución de las colisiones para cada una de las fases de vuelo, diferenciando entre el período de 2001-2007 y el de 2008-2015.

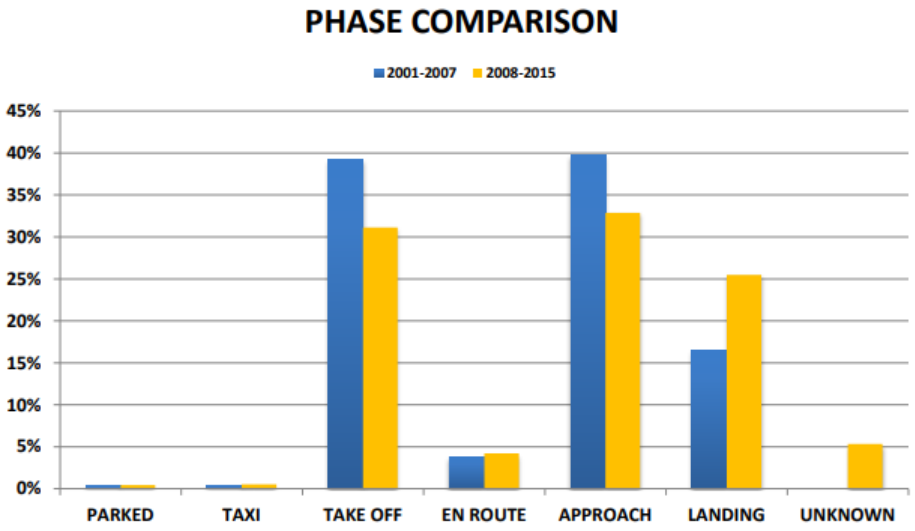


Figura 2: Concentración de casos de “bird-strike” reportados por fase de vuelo [1]

Se observa que la mayor concentración de incidentes ocurre durante las fases de despegue, aproximación y aterrizaje, fases del vuelo que se llevan a cabo dentro del entorno aeroportuario, motivo por el que los aeropuertos se han servido de la cetrería, utilizando halcones adiestrados para mantener a las aves alejadas de las vías de despegue y aterrizaje de las aeronaves. Además, la Figura 2 resulta muy ilustrativa a la hora de enteder que el riesgo de “bird-strike”depende en gran medida de la altitud a la que se encuentre la aeronave, puesto que la mayoría de incidentes

se concentran en fases de vuelo en el que la aeronave se encuentra a una cota baja.

Habiendo relacionado la altitud de vuelo con el riesgo de colisión, se observa que mientras que la aviación comercial alcanza altas cotas de vuelo en ruta (por lo general demasiado altas para las aves), la aviación general suele realiza operaciones a una altitud menor aumentando así el riesgo de accidentes. Este hecho sumado a una menor diferencia de tamaño entre las aeronaves más pequeñas y las más grandes aves, hace que los "bird-strikes" se conviertan en un fenómeno de gran peligro, tanto para la vida de los pasajeros como para la de las aves.

En España, las mayores aves y causantes de un gran número de casos de "bird-strike" son los buitres con el buitre leonado (Gyps fulvus) y el buitre negro (Aegypius monachus) encabezando la lista. Estos animales pesan entre 6 y 12 kilogramos cuando son adultos y cuentan con una envergadura de entre 2,5 y 3 metros. En la última década encontramos casos como el del Airbus A340 de Iberia que en 2012 chocó frontalmente contra un buitre leonado a 2000 metros de altura, causando graves daños estructurales en el morro del avión [2] y diversos accidentes de aviación general con consecuencias normalmente fatales, como la muerte de tres jóvenes en Perales de Tajuña 2016 al colisionar su Cessna172 contra un buitre, haciendo que una de las alas se desprendiese de la aeronave [3], la muerte de una familia en Cuenca ese mismo año [4] o la muerte de 3 personas en la localidad navarra de Arbizu también en 2016 [5]. Por supuesto, ambos casos se cobraron también la vida de los buitres.

Por ello resultaría un gran avance en la seguridad de los vuelos de aviación general si fuera posible desarrollar un dispositivo de bajo coste que permita detectar a los buitres en vuelo antes de que se produzca una colisión y alertar al piloto de la zona en la que se detectan para que pueda esquivarlos (desviándose en dirección opuesta y hacia arriba).

Algunas de las soluciones propuestas han sido la localización de los buitres mediante chips gps o el estudio de sus conductas y sus zonas de vuelo para intentar prevenir y evitar, en la medida de lo posible, los impactos con estos animales. Pero estas aproximaciones no sirven para el caso de la aviación general.

En el caso de la aviación general, en el que los aviones vuelan a velocidades relativamente bajas y los impactos se producen en la fase de crucero, lejos de los aeropuertos (con lo que no se puede utilizar un radar con base en tierra), se hace necesaria la utilización de un dispositivo embarcado de bajo peso y consumo que permita detectar los pájaros en vuelo.

Una forma de abordar el problema es utilizar el reconocimiento de imágenes utilizando la inteligencia artificial, AI por sus siglas en inglés, más concretamente aprovechando las características de las redes neuronales, NN, a la hora de realizar la tarea del procesamiento de imágenes. Las NN se basan en una arquitectura que intenta emular el funcionamiento del cerebro humano a través de nodos interconectados entre sí, llamados neuronas. Estas han supuesto una revolución para la tecnología, ya que permiten a la computación resolver problemas cada vez más complejos.

En los últimos años la investigación en las redes neuronales convolucionales, CNN, ha aumentado rápidamente. Las CNN se especializan en el procesado de imágenes a través de un gran número de capas ocultas con diferentes especializaciones que permiten reconocer desde líneas y formas hasta rostros humanos, consiguiendo así resolver problemas de clasificación partiendo de grandes volúmenes de imágenes "datasets".

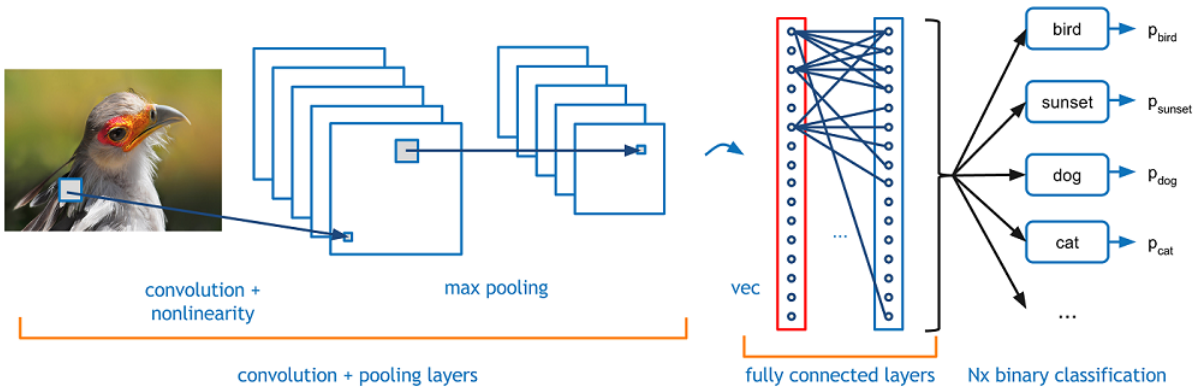


Figura 3: Esquema simplificado de una CNN [6]

En un sistema embarcado, las CNN permiten clasificar imágenes en tiempo real. Sin embargo, también comportan un problema y es la capacidad de procesamiento de la computadora de abordo, ya que este tipo de redes neuronales requieren de un elevado número de operaciones por segundo. Este es el motivo de la creación de los “hardware accelerators”para CNN. Entre ellos las FPGAs son uno de los más utilizados. Estas estan basadas en un chip integrado que permite la reconfiguración de puertas a nivel de hardware para acelerar el procesamiento, cuenta con un gran número de pequeños elementos lógicos que se pueden programar en diferentes módulos para conseguir un amplio rango de aplicaciones diferentes.

1.2. Objetivos

El objetivo de este trabajo final de grado es el de establecer las bases y realizar un estudio de la viabilidad tecnológica referente al diseño de un sistema de detección de buitres basado en el procesamiento de imágenes mediante una CNN entrenada y descargada en una FPGA, de forma que en un futuro se pueda embarcar en una aeronave para alertar al piloto de la presencia de aves en las proximidades y que este sea capaz de reaccionar con la suficiente antelación como para poder evitar la colisión. Para ello, La CNN obtendrá las imágenes a través de una cámara 4K, también embarcada en el avión.

1.3. Estado del arte

En esta sección se recopilan los estudios previos relacionados con el objetivo de este trabajo de fin de grado. Por una parte se han buscado proyectos referentes al estudio de las aves aplicando “deep learning” en busca de tanto de imágenes de estos animales para el entrenamiento de la futura CNN, como de los tipos de redes neuronales que mejor funcionan a la hora de clasificar estos animales. Por otra parte se recopila información a cerca de la programación de la FPGA a través de trabajos en los que se implemente una red neuronal en ella.

1.3.1. Clasificación y seguimiento de aves mediante redes neuronales

Analizando los proyectos basados en el análisis y la diferenciación de aves mediante la utilización de una red neuronal clasificatoria, se encuentra el estudio del profesor R. Yoshihashi y su equipo [7]. En él se analizan las imágenes obtenidas durante tres días en un campo eólico,

con más de 30000 imágenes en total. Las fotografías obtenidas son tanto de aves como de otros objetos que pudieran confundirse con las mismas, como insectos, aviones, hojas o las propias aspas de los aerogeneradores.

De esta forma y utilizando la ayuda de expertos ornitólogos para el correcto etiquetado de cada una de las aves se obtiene un gran "dataset" de imágenes de aves [8]. En la Figura 4 se puede ver una pequeña muestra del mismo con las diferentes categorías de estudio.

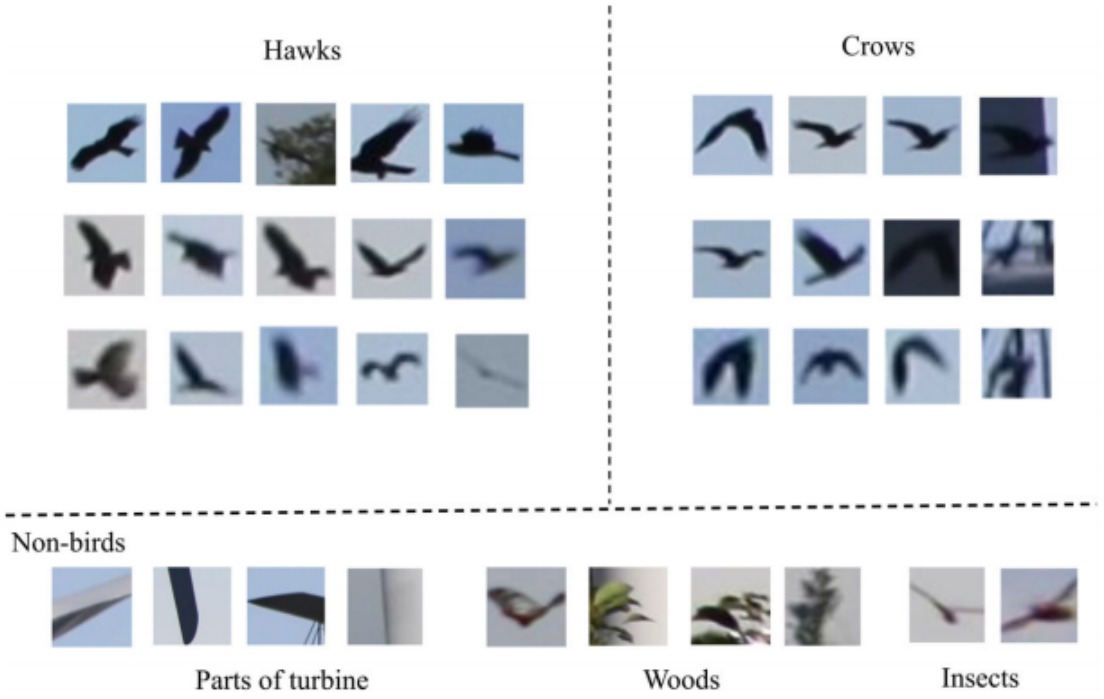


Figura 4: Fragmento del dataset del Prof. Yoshihashi y su equipo [7]

Posteriormente se utiliza dicho "dataset" para probar diferentes métodos de clasificación basados en "deep learning" entre ellos el entrenamiento de dos redes neuronales (LeNet y ResNet).

- AdaBoost27 combinado con detectores de características de imagen como Haar-like22, Histogram of Orientated Gradients HOG^{23} o RGB
- LeNet
- ResNet

Por lo que a las redes neuronales se refiere, tanto la LeNet como la ResNet están basadas en la estructura CNN, especializándose en el procesamiento de imágenes. Según el estudio realizado, con ambas se obtienen resultados más que satisfactorios a la hora de detectar y clasificar las distintas aves.

Cabe mencionar que en este estudio, debido a que la posición de la cámara es fija, se utiliza un algoritmo de sustracción del fondo, para obtener las zonas de la imagen candidatas a albergar un ave.Lo que simplifica mucho el problema frente al que se plantea en el TFG, ya que en las

imágenes a utilizar la cámara está continuamente en movimiento.

Después de evaluar los diferentes métodos, previamente mencionados, se llega a la conclusión de que aquellos con mejor rendimiento para la detección son AdaBoost27+RGB y ResNet, obteniendo resultados similares, siguiéndolos de cerca LeNet con resultados bastante parejos. El estudio comenta que los métodos aplicados más sencillos que obtienen buenos resultados, son menos flexibles ante condiciones cambiantes, algo muy habitual en las imágenes obtenidas en un aeronave de aviación general, con lo que optaremos por el uso de las redes neuronales.

Además, junto con el "dataset" previamente mencionado se pueden descargar los modelos caffe [9] ya entrenados, utilizados para la clasificación de aves.

Por otro lado, el estudio de Juha Niemi y Juha T. Tanttu [10] parte de una premisa similar al del Prof. Yoshihashi, la de monitorizar un campo eólico con la finalidad de entrenar una red neuronal para la clasificación de las diferentes especies de aves. En este caso, sin embargo, se aplica la técnica conocida como "data augmentation" (Figura 5) con la que se consigue aumentar el rendimiento del clasificador sin que sea necesario tener una gran "dataset" de imágenes.

Figura 5: Técnica de "data augmentation" de variación de la temperatura del color [10]

Para realizar el proceso de "data augmentation" Juha Niemi y Juha T. Tanttu llevan a cabo una serie de transformaciones en las imágenes que forman el "dataset" . La primera de estas transformaciones se da en la temperatura del color de las imágenes y responde a la necesidad de entrenar a la CNN para que sea capaz de realizar su labor con diferentes condiciones climáticas o en momentos diferentes del día, lo cual afecta a la temperatur del color de las imágenes tomadas. Para llevar esto a cabo se realizan variaciones en la temperatura del color de las imágenes entre 2000 K y 15,000 K con intervalos de 50, 75, 100, 150, 200, 250, 300 y 1000. Utilizando este método con el intervalo más pequeño (50), se consigue para una única imágen un total de 260 imágenes, además de la original.

Posteriormente se rotan las imágenes un ángulo aleatorio comprendido entre 20º y 30º. El hecho de escoger estos ángulos responde a la necesidad de mantener a las aves en ángulos que puedan ser verosímiles, evitando situaciones en las que el ave vuele boca abajo. Mediante este proceso se puede aumentar el número de imágenes contenidas en el "dataset" puesto que las CNN son invariables ante pequeñas translaciones, sin embargo, no lo son ante rotaciones.

A diferencia del caso anterior, en este estudio el modelo de clasificación no se lleva exclusi-

vamente a través de una CNN preentrenada en esa función, si no que se utiliza una arquitectura más sólida con un mayor número de variables. En el sistema propuesto, la imagen, además de transferirse directamente a la CNN para su clasificación, se segmenta para obtener el tamaño del objetivo en pixels. Esta información junto con la distancia a la que se encuentra este objeto proporcionada por el radar hacen posible una estimación del tamaño aproximado del ave, lo que ayuda a determinar la especie de la misma.La información anterior se utiliza en una de las últimas capas de la red neuronal (denominada capa Softmax) en la que se mezclan los vectores de salida de la CNN junto con la información adicional del tamaño del ave y la velocidad de la misma, proporcionada por el radar, devolviendo como resultado la clasificación final del animal.

Otro de los documentos que se han considerado de interés para el estudio que se llevará a cabo es el artículo publicado por Christopher J.W. McClure, Luke Martinson y Taber D. Allison [11] en el que se habla del sistema IdentiFlight. Este es un sistema que se utiliza para identificar y monitorizar águilas en los alrededores de un campo eólico de forma que se pueda determinar si alguna de las turbinas debe de ser apagada por la seguridad del animal.

En este sistema se utilizan cámaras con campo de visión amplio, WFOV, encargadas de detectar objetos en movimiento en el área circundante al campo eólico para después seguirlos con una cámara estéreo de alta resolución,HRSC, que estima la distancia al objeto y toma imágenes del mismo. A partir de ese mopmento se utiliza un algoritmo para clasificar las imágenes obtenidas del objeto. En conjunto el sistema es capaz de detectar y clasificar objetos a una distancia aproximada de unos 1000 m.

A partir del Apéndice Online que se puede encontrar en el mismo artículo es posible extraer más información a cerca del algoritmo utilizado por el sistema IdentiFlight para la clasificación de las aves. Por ejemplo el cálculo del tamaño del ave a partir de la distancia obtenida a través de las imágenes en estéreo cada 125 ms de la cámara HRSC, y el grado angular de la cámara con respecto al ave a partir de la cámara WFOV. Mediante procesos similares el algoritmo de detección es capaz de identificar más de 200 atributos de la imágen entre los que se encuentran la estimación de la longitud del cuerpo, de la emvergadura, de la postura alar y del color.

Cabe mencionar, que en este caso se cuenta con cuatro cámaras fija sobre la superficie de la Tierra, capaces de obtener imágenes de una misma ave desde diferentes ángulos permitiéndo así una detección y clasificación más rápidas. Además, debido a que la posición es estática, el cálculo del tamaño del objeto volador se lleva a cabo con mayor facilidad pudiendo discriminar rápidamente entre objetos lejanos como aviones, demasiado grandes como helicópteros o demasiado pequeños como insectos o drones.

1.3.2. Implementación en una FPGA

Dentro de este campo se encuentra el proyecto de fin de grado de Erwei Wang, tutorizado por el Prof. P.Y.K. Cheung y el Prof. G.A. Constantinides [12]. Este proyecto se basa en la creación de un prototipo de “framework” para FPGA que permita la implementación de una CNN de alto rendimiento en la plataforma PYNQ.

Para ello lleva a cabo un estudio sobre los diferentes “frameworks”disponibles para la plataforma eligiendo tres de los más comunes, como son Tensorflow, Caffe y Theano. Una vez analizados puede destacar las ventajas y desventajas de cada uno de ellos atendiendo a tres parámetros.

- Dificultad de la instalación en la plataforma PYNQ
- Soporte para las capas personalizadas
- Simplicidad en el diseño de la interfaz

En lo referente a la dificultad de instalación se considera Theano como el más sencillo de los tres, puesto que no es necesario llevar a cabo una larga compilación para instalarlo. Caffe, por su parte es considerado de dificultad moderada debido a la larga compilación que requiere, mientras que Tensorflow aparece como fallo en la instalción debido a un fallo con el comando usado para ejecutar la CNN (*session.run()*).

En lo referente al soporte encontramos a Theano como el mejor de los tres, con un elevado soporte y opciones de personalización bien diseñadas. Seguido por Tensorflow y finalmente Caffe.

Por lo que a simplicidad de diseño de la interfaz respecta Theano vuelve a destacar debido a su simplicidad, mencionando el Gráfico Acíclico Dirigido, DAG, contruido a partir de funciones de Python. Por su parte Caffe presenta una simplicidad moderada.

Tras el análisis de cada uno de los "frameworks" Erwei Wang se decide por Theano debido, principalmente, a su simplicidad y la facilidad de su instalación en la plataforma PYNQ. Además Theano es capaz de ejecutar modelos preentrenados en Caffe por lo que instala primero Caffe y después Theano con Lasagne (una librería de redes neuronales para Theano).

Junto a todo lo anteriormente mencionado, el proyecto tiene un repositorio de github [13] donde se adjunta un manual de instalación de Caffe y Theano con Lasagne junto con dos redes neuronales con estructura CNN preentrenadas, para su ejecución en la FPGA, una LeNet y una CIFAR_10.

Por último, se ha creido importante mencionar el estudio de Jaechan Cho y su equipo [14]. En este trabajo se ha llevado a cabo un estudio sobre la implementación de un algoritmo de extracción de fondo dinámico en una FPGA. De esta forma se puede aplicar la técnica de sustracción de fondo a una cámara en movimiento, que en el caso del estudio se aplica a la conducción de un vehículo autónomo. Este requisito hace que además de la detección, clasificación y seguimiento de los objetos la red neuronal necesite una velocidad de procesamiento de imágenes elevada, puesto que el algoritmo se va a utilizar en un sistema embarcado, de forma similar a lo que se pretende realizar en este proyecto, como se observa en la Figura 6.

Figura 6: Resultados de la detección utilizando la sustracción de fondo dinámico [14]

Por desgracia, aunque se menciona la correcta implementación y los buenos resultados del algoritmo clasificador y de seguimiento, en el estudio de Jaechan Cho y su equipo no aparece ningún código que permita implementar un algoritmo de sustracción de fondo dinámico en una FPGA, para su posterior utilización en este proyecto.

Afortunadamente hay varias publicaciones sobre optical flow con código que podría ser adaptado a nuestro proyecto para estimar el movimiento del fondo en fases posteriores del proyecto. Una de ellas, con código en Python a partir del que trabajar para implementar una solución, es [15]

2. Antecedentes

El propósito de esta sección es el de dotar al lector de un conocimiento previo tanto en redes neuronales, como en las FPGA y los métodos de seguimiento de objetos en vídeo, con el fin de crear un marco teórico que facilite la lectura y comprensión del proyecto realizado.

2.1. Redes neuronales

En este apartado se proporciona información básica a cerca de las redes neuronales, haciendo especial hincapié en su funcionamiento y estructura.

Una red neuronal es un modelo de computación con el que se pueden modelar comportamientos inteligentes. Cuentan con una arquitectura basada en capas que albergan un gran número de neuronas interconectadas entre sí, intentando emular la estructura del propio cerebro humano. El comportamiento de una red neuronal, viene determinado tanto por la forma de conexión de cada uno de los elementos de la misma, como por la ponderación de estas conexiones [16].

La neurona es la unidad básica de procesamiento de una red neuronal, cada una de ellas tiene conexiones de entrada, a través de las cuales recibe valores de las neuronas anteriores y conexiones de salida que utiliza para mandar los resultados de su procesamiento a las posteriores. Internamente, la neurona utiliza los valores de entrada para realizar una suma ponderada de los mismos, esta ponderación viene dada por el peso que se le asigna a cada una de las conexiones de entrada, ajustándose de está forma la intensidad con el que cada uno de los valores de entrada afecta a la neurona como se puede observar en la Figura 7.

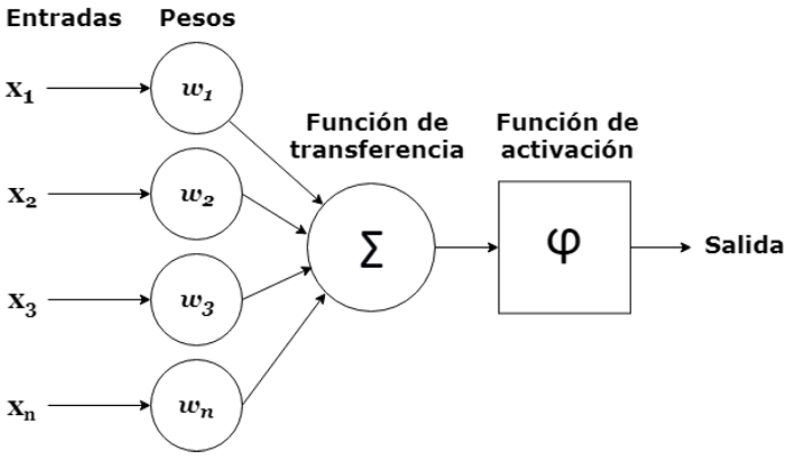


Figura 7: Estructura interna de una neurona artificial

Además, al valor de salida de una neurona se le suele aplicar una función limitadora, o de activación, que modifica este valor añadiéndole deformaciones no lineales con la finalidad de que se puedan encadenar de forma efectiva la computación de varias neuronas [17].

Las neuronas se agrupan en capas que dan forma a la red neuronal. La primera de estas capas se denomina capa de entrada y contiene unidades que representan los campos de entrada, mientras que la última es la capa de salida con unidades que hacen lo propio con los campos de

destino. El resto de capas, entre la de entrada y la de salida, son conocidas como capas ocultas y cada una de ellas utiliza la salida de la capa anterior como su propia entrada. Los datos iniciales se presentan en la capa de entrada y desde esta se propagan de una neurona a otra hasta la capa de salida, de la que se obtiene el resultado, como se puede apreciar de forma simplificada en la Figura 8.

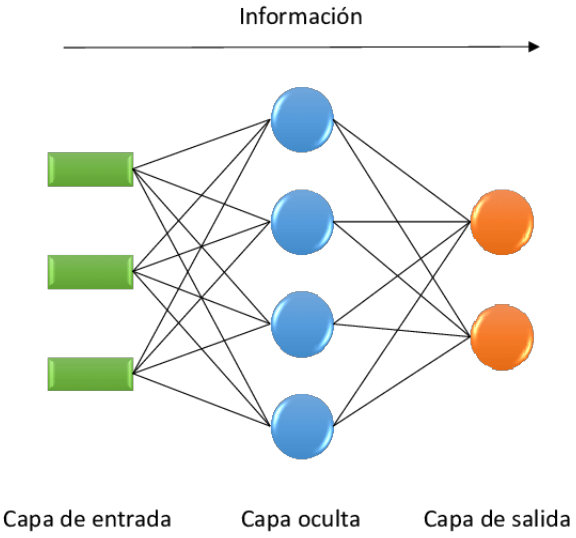


Figura 8: Estructura de una red neuronal con una única capa oculta

El aprendizaje de una red neuronal se basa en los registros individuales. Para su entrenamiento, se introduce un gran número ejemplos cuyos resultados son conocidos de antemano. La propia red genera una predicción para cada uno de ellos y en caso de error pasa la información hacia atrás (fenómeno conocido como “back propagation”), realizando reajustes en las ponderaciones previamente mencionadas. A través de este proceso la red se va haciendo cada vez más precisa hasta que finalmente se la puede utilizar para realizar predicciones de casos en los que el resultado es desconocido [18].

2.1.1. Redes neuronales convolucionales

Las CNN, ofrecen grandes resultados en lo referente a la clasificación y el procesamiento de imágenes. En esta sección se explica brevemente qué diferencia a este tipo de redes neuronales del resto, estudiando su estructura y algunas de sus aplicaciones más comunes.

Una red neuronal convolucional es muy parecida a una red neuronal convencional puesto que ambas tienen la misma estructura formada por capas de neuronas interconectadas entre sí y tienen una función similar, la de aprender mediante ejemplos cuya solución es conocida para ser capaces de obtener predicciones para nuevos casos. La principal ventaja que presentan las CNN frente a las NN es que cada parte de la CNN es entrenada para realizar un tipo de tarea, reduciendo así el número de capas ocultas comparado con una red neuronal convencional y por tanto el tiempo de entrenamiento de la propia red.

Estas redes son capaces de aprender de forma directa las características de las imágenes a analizar por lo que no es necesaria la extracción de estas características de forma manual,

además sus resultados en lo referente al conocimiento son excepcionales y una misma CNN puede volver a entrenarse para abarcar un campo de acción diferente al original.

La estructura de estas redes está formada, principalmente, por tres tipos distintos de capas. cada una con una función específica dentro de la red neuronal.

- Capa de convolución (“Convolutional layer”)
- Capa de reducción (“Pooling layer”)
- Capa de conexión total (“Fully-connected layer”)

La estructura de una CNN se basa en la alternancia entre capas de convolución y reducción para finalmente llegar a las capas de conexión total. Las primeras hacen pasar las imágenes a través de un conjunto de filtros convolucionales, donde cada uno de ellos activa ciertas características de la imágenes. Las capas convolucionales suelen tener una unidad linear rectificadora, más conocida como ReLU (Figura 9), que convierte los valores negativos a cero mientras que mantiene los valores positivos agilizando así el entrenamiento.

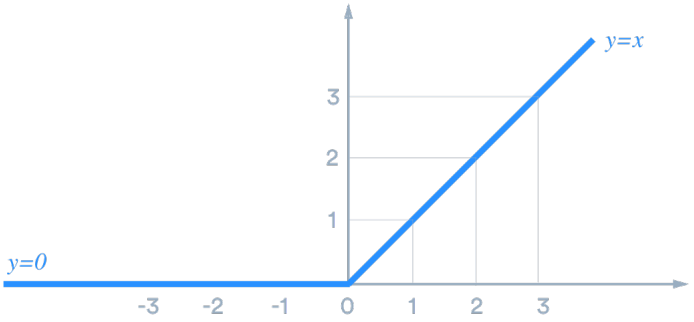


Figura 9: Función ReLU

Las segundas utilizan como entrada la salida de las capas convolucionales y la simplifican mediante una disminución no lineal de la tasa de muestreo, reduciendo así el número de parámetros necesarios para el aprendizaje de la red.

A continuación se encuentran las capas de conexión total, que generan un vector de K dimensiones que contiene las probabilidades de cada una de las clases para la imagen que se está analizando, siendo K el número de clases que la CNN es capaz de predecir. Por último las CNN tienen una capa de clasificación que tendrá K salidas, siendo cada una de estas la clase final predicha por la red neuronal [19]. En conjunto la estructura de este tipo de redes se asemeja a la mostrada en la Figura 10.

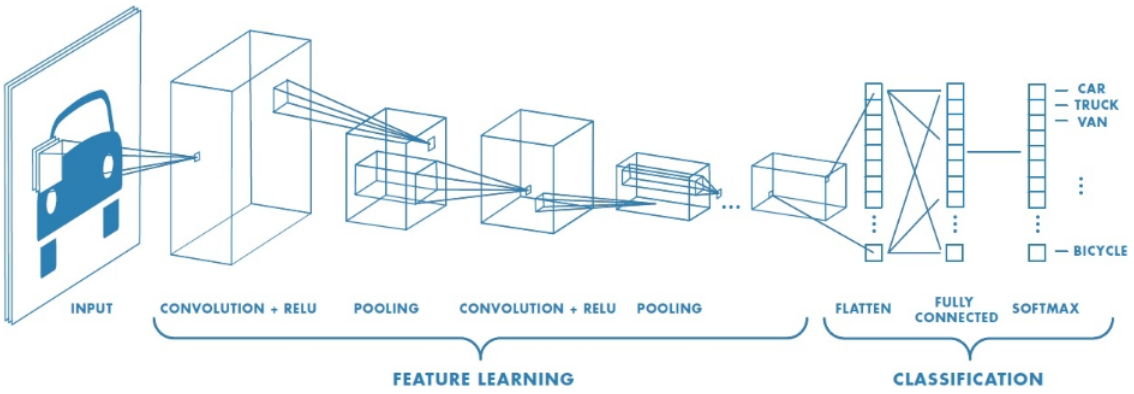


Figura 10: Estructura de una red neuronal convolucional

Gracias a su particular arquitectura las CNN presentan grandes resultados en el campo de la visión artificial destacando en reconocimiento de imágenes y la detección de patrones. Esto les permite tener un elevado número de aplicaciones entre las que encontramos, reconocimiento facial, sistemas de vigilancia, control de tráfico, automatización de la agricultura, reconocimiento de errores en una cadena de montaje automatizada... En la Figura 11 se muestra la clasificación de objetos realizada por una CNN a partir de una imágen dada.

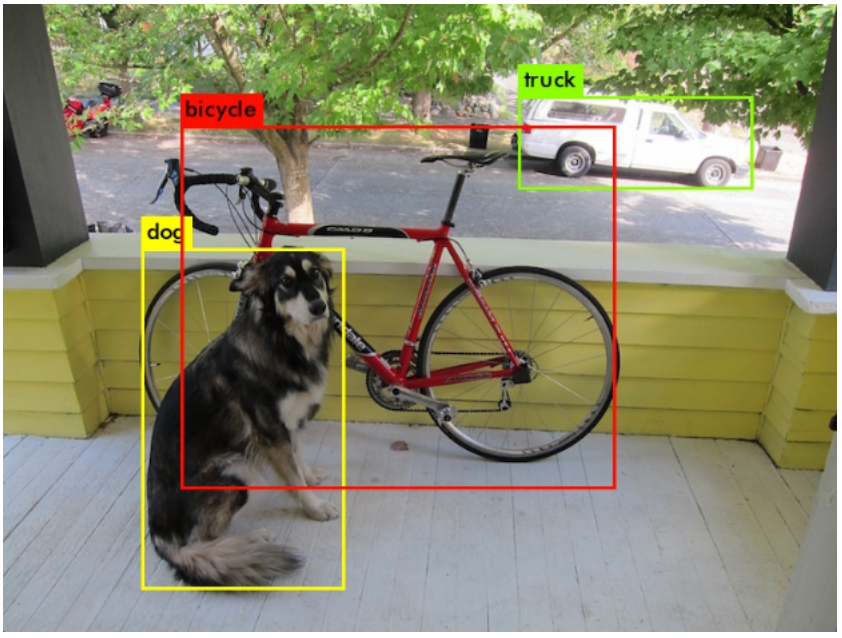
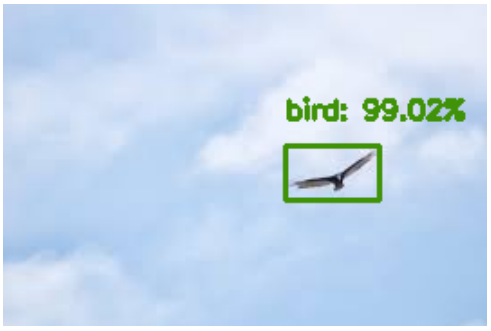


Figura 11: Clasificación de objetos mediante CNN

Este mismo procedimiento se puede aplicar al ámbito de estudio de este TFG, discriminando a partir de una imágen entre un ave u otro objeto que se encuentre en el cielo, como se puede apreciar en la Figura 12.

(a) Aeronave



(b) Ave

Figura 12: Clasificación de una aeronave y un ave en vuelo mediante CNN

2.1.2. LeNet y ResNet

Como se ha mencionado en la sección correspondiente al estado del arte, el estudio del profesor Yoshihashi ha probado la viabilidad tanto de la LeNet como de la ResNet para el estudio y clasificación de aves a partir del procesamiento de imágenes. En esta sección se analizan ambas redes explicando sus características.

La LeNet o lenet-5 es una red neuronal convolucional propuesta por Yann LeCun en 1998 para el reconocimiento de escritura. Se trata de una de las CNN más simples, debido a que fue una de las primeras en proponerse y está basada en la arquitectura clásica de las CNN (explicada anteriormente). En concreto este tipo de red cuenta con dos capas convolucionales, dos capas de reducción y tres capas de conexión total que actúan como clasificador, apreciables en la Figura 13.

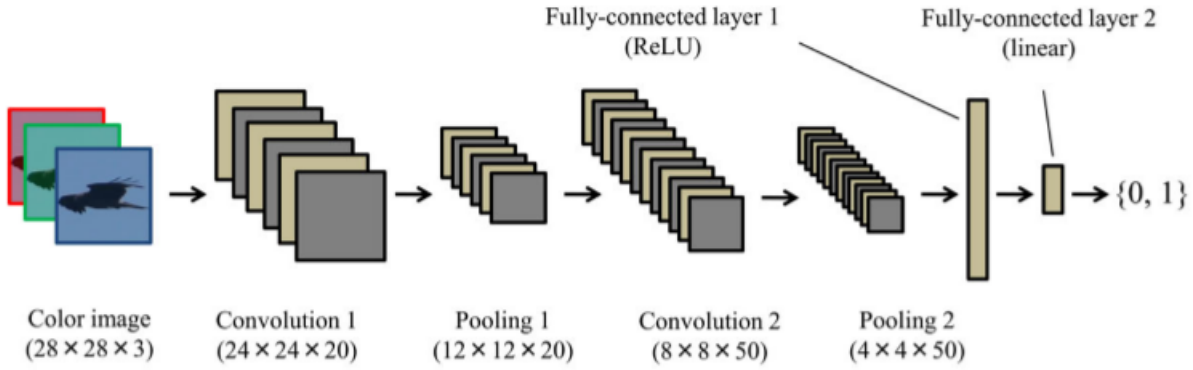


Figura 13: Estructura de una LeNet [20]

Por su parte, la “deep residual network” más conocida como ResNet es un tipo especializado de CNN que ofrece grandes resultados en lo que a la clasificación de imágenes se refiere. La principal diferencia entre este tipo de red y las NN convencionales son las conexiones en forma de atajos entre diferentes capas convolucionales como se muestra en la Figura 14.

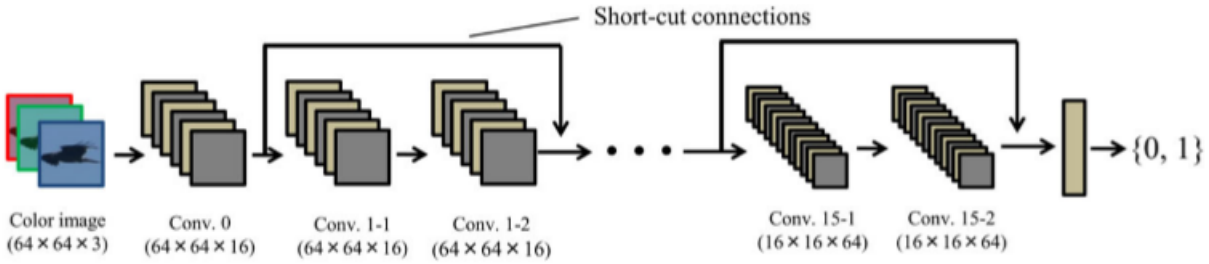


Figura 14: Estructura de una ResNet [21]

Gracias a la arquitectura de la ResNet la información puede propagarse sin ser necesario que atraviese todas las capas de la CNN, lo que hace posible aumentar el número de capas ocultas de la red sin caer en el problema de la desaparición del gradiente. Este es un problema que afecta a las capas más bajas (las más cercanas a la entrada) y ocurre debido a que a medida que el modelo realiza el proceso de “back propagation”durante el entrenamiento, el gradiente se hace cada vez más pequeño haciendo que el entrenamiento de estas primeras capas sea muy lento o directamente imposible.

2.2. Fiel-Programmable Gate Arrays (FPGAs)

En este proyecto se ha decidido utilizar hardware acelerador de bajo coste de forma que este permita la aplicación de una CNN en un sistema embarcado. El hardware elegido para este proyecto es la FPGA, en esta sección se expandirán los conocimientos básicos a cerca de estos aceleradores.

Una FPGA es un circuito integrado, IC, el cual se puede programar para diferentes algoritmnos después de su fabricación. Las FPGAs modernas han llegado a contener hasta dos millones de celdas lógicas que pueden ser reconfiguradas de forma que puedan cubrir la implementación de una gran variedad de algoritmos de software.

Aunque el diseño tradicional de las FPGA tiene más similitudes con un IC normal que con un procesador, estas presentas grandes ventajas en comparación con los IC en lo referente al coste de desarrollo a la vez que ofrecen un nivel de rendimiento similar en la mayoría de los casos.

Otra ventaja de las FPGA en comparación con los IC es su capacidad de reconfigurarse dinámicamente, que és la capacidad de reconfigurar sus puertas lógicas a nivel de hardware para acelerar el procesamiento. Este proceso, es similar al de cargar un programa en un procesador y puede afectar parte o a la totalidad de los recursos disponibles de la propia FPGA [22].

2.2.1. PYNQ-Z2

PYNQ-Z2 es una placa de desarrollo perteneciente a $Xilinx^{TM}$. La placa cuenta con una FPGA Zynq XC7Z020 con 512MB DDR3/Flash y Dual-Core ARMCortex-A9 CPU. La placa está equipada con el sistema operativo Linux Ubuntu 15.10 que incluye el compilador de Python [12]. Como inconveniente para el presente proyecto encontramos que esta placa es solo capaz de procesar vídeo con una resolución de 1080p. En la sección de cálculos se estudiará la relevancia

de este hecho para el desarrollo del proyecto.

En la Figura 15 se observa la placa con sus diferentes componentes con la finalidad de facilitar la localización de los mismos durante las explicaciones que se realizarán más adelante.

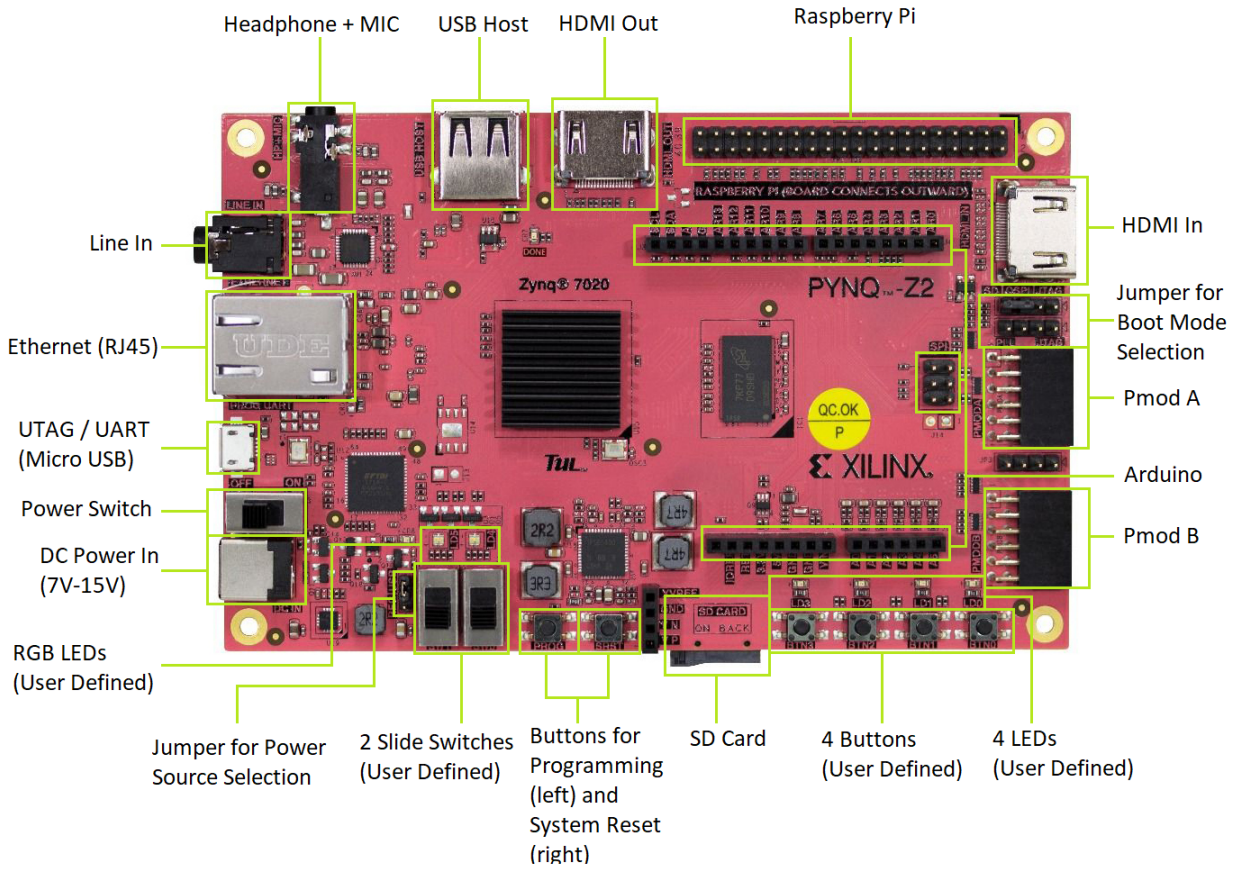


Figura 15: Componentes de la PYNQ-Z2

La PYNQ-Z2 cuenta con la plataforma PYNQ para FPGA que facilita la interacción con entre el usuario y el dispositivo, facilitando así su uso. Más concretamente, el proyecto PYNQ [23] publicado por $Xilinx^{TM}$ permite a los usuarios iniciar los IPs de las FPGA presintetizados en Python, sin tener que profundizar a nivel de hardware.

Gracias a esta plataformas ya no es necesario para el usuario tener un avanzado conocimien-

to sobre la implementación de hardware en la FPGA, esto a su vez ha hecho que en la actualidad las FPGAs sean más accesibles para un mayor número de usuarios explicando así la popularidad que han empezado a obtener.

Además, esta plataforma en concreto es perfecta para la implementación de CNN en sistemas computacionales. Esto es debido a que la propia CPU de la PYNQ contiene el sistema operativo de Linux Ubuntu 15.10 OS con lo que se consigue que la PYNQ sea capaz de ejecutar los ”frameworks”para CNN completamente en su CPU.

Para facilitar el montaje de una red neuronal en la FPGA, esta se ha de realizar utilizando el lenguaje de programación integrado por defecto en la placa, que como se ha mencionado anteriormente, es Python. Además se debe instalar uno de los diferentes frameworks disponibles para Python que contenga librerías de ”deep learning”para la implementación de la CNN. Entre los frameworks más utilizados encontramos Caffe, Theano y Tensorflow.

Para el proyecto se ha pensado utilizar modelos Caffe puesto que el estudio del profesor Yoshihashi utiliza estos mismos modelos, con lo que se entiende que Caffe es un framework válido para esta FPGA. Además la compatibilidad de los modelos Caffe con Theano hace que ambos se conviertan en las opciones más adecuadas.

Theano puede ejecutar modelos preentrenados de Caffe y es bastante fácil de instalar en la FPGA (apenas un par de líneas de código). Además, se puede montar sobre él Lasagne, una librería pensada para crear y entrenar redes neuronales en Theano.

Por último Tensorflow, aunque una de las librerías más conocidas ha dado muchos problemas a la hora de instalarse debido a las incopatibilidades con la nueva versión de python (3.6) instalada por defecto en la FPGA, por lo que se ha dejado de lado.

2.3. Métodos para el seguimiento de objetos

Una vez se tiene la red neuronal lista para su funcionamiento, es importante pensar en la forma de detección y clasificiación de las aves, en el sistema que se ha propuesto al comienzo de este trabajo. Es por eso que está sección esta dedicada a mostrar los métodos de seguimiento de objetos a partir de un vídeo mediante IA, que se han considerado viables para la realización de este proyecto.

2.3.1. Sustracción de fondo dinámico

Como bien se ha mencionado en la sección dedicada al estado del arte, el estudio del Prof. Yoshihashi utiliza la sustracción de fondo estático para reducir las imágenes detectando las zonas de las mismas que pueden albergar el ave. En el caso que se plantea en este proyecto, esa técnica no es posible, puesto que la cámara irá a bordo del avión y por tanto el fondo del vídeo creado presentará un comportamiento dinámico.

La sustracción de fondo dinámico o consiste en separar los objetos del fondo de imágen, utilizando, por lo general, los patrones de movimiento (en este caso un ave se mueve de forma diferente al cielo de fondo). Este procedimiento facilita el seguimiento o la identificación de objetos como se muestra en la Figura 16.

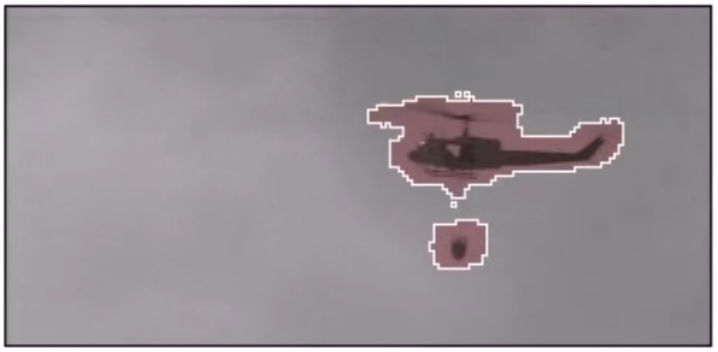


Figura 16: Seguimiento de un helicóptero mediante sustracción de fondo dinámico [24]

Uno de los modelos más populares es el de Stauffer y Grimson, SG, el cual se basa en la distribución temporal de los colores de los pixels como una mezcla de Gaussianos. Este modelo está basado en la premisa de que, en una escena, un objeto en movimiento cambiará, con el tiempo, el color de los píxeles a medida que la recorre, mientras que si se detiene el objeto se volverá a considerar parte del fondo, lo cual no es un problema porque ni el ave ni la aeronave se pueden detener completamente en el cielo.

Sin embargo, aunque es uno de los más utilizados, este modelo presenta una desventaja a tener en cuenta, sobretodo para este proyecto en concreto y es la asunción de que el fondo de la imágen es estático a corto plazo, lo cual es una limitación para escenas con un comportamiento dinámico, como el agua o el cielo. Uno de los enfoques que más prometedores se ha mostrado a la hora de modelar este tipo de patrones de movimiento de fondo es la “dynamic texture“que modela un volumen espacio-temporal como una muestra de un sistema dinámico lineal, o por sus siglas en inglés, LDS, y ha demostrado una robustez sorprendente para la síntesis de vídeo, segmentación y registro de imagen.

2.3.2. Sliding window

En el contexto de la visión por computadora, se utiliza el término “sliding window“ para referirse a una región rectangular del espacio, de medidas fijas, que se desliza sobre el vídeo de fondo. Este tipo de técnicas tienen una gran relevancia en el área de clasificación de objetos, puesto que permite localizar exactamente en que lugar de la imagen reside el objetivo. Para ello utiliza una CNN clasificadora previamente entrenada, a la que se le añade una nueva categoría, “background“. Una vez se tiene la CNN, se divide la imágen en un número definido de ventanas más pequeñas y la ventana empieza a recorrerlas junto al clasificador.

Las ventanas que se utilizan para la técnica de “sliding window“ deben de tener un tamaño similar al objeto de interés, de forma que cuando este se detecte, se pueda crear una caja que lo contenga en su mayor parte. Sin embargo, debido a que los objetos pueden ser de diferentes tamaños, dependiendo de la distancia a la que se encuentren se pueden crear varios tamaños de ventana.

Figura 17: Creación de un algoritmo de sliding window [25]

En la Figura 17 se muestra el proceso de creación de una sliding window para la clasificación y detección de vehículos. Como se puede comprobar, se han creado diferentes tamaños de ventana para cada uno de los escenarios, dependiendo de la distancia de la cámara con el vehículo objetivo. Además, utilizan una superposición del 50 % de la ventana tanto en vertical como en horizontal para conseguir una mayor cobertura de la imágen. En azul se muestra la ventana deslizante, mientras que en negro se muestra la malla en la que se ha dividido la imagen.

Utilizando esta técnica se pueden detectar y clasificar objetos en una imágen estática o realizar un seguimiento o “tracking“ del objeto, creando una caja alrededor del mismo que lo siga durante el transcurso del vídeo, como se muestra en la Figura 18. Por supuesto, esta caja irá variando de tamaño a medida que lo haga el objetivo pero siempre manteniendo a este en su interior.

(a) Escena 1

(b) Escena 2

Figura 18: Detección y tracking de coches mediante sliding window [26]

3. Desarrollo del proyecto

En este apartado se recopilará toda la información recogida así como los procedimientos llevados a cabo para la realización del proyecto.

3.1. Cálculos preliminares

En la presente sección se muestran los cálculos realizados previamente a la utilización de la FPGA. Estos son necesarios para obtener una estimación de los márgenes de tiempo de los que se dispone antes de la colisión y de los tamaños de píxeles correspondientes a la envergadura y distancia del ave.

El primer paso consiste en recoger las características relacionadas con el tamaño y la velocidad de los buitres, así como a la velocidad de la aeronave en las distintas fases de vuelo, siendo estas: vuelo nivelado, vuelo ascendente y vuelo descendente. Para este estudio se ha considerado el modelo de cámara Firefly 8S 4K con una apertura lineal de 90^o [27].

Buitre:

- Envergadura: 3 m
- Altura: 1 m
- Velocidad: 48 km/h

Aeronave:

- Velocidad vuelo ascendente (km/h): 150 km/h
- Velocidad vuelo nivelado/descendente: 200 km/h

Cámara:

- Ángulo de apertura de la lente (α): 90^o

A partir de estos datos se calcula el tiempo que tarda la aeronave en recorrer 1 km y 500 m tanto ascendiendo como en vuelo nivelado o descendente.

$$t_{A1000} = 24\ s$$
$$t_{A500} = 12\ s$$
$$t_{D1000} = 18\ s$$
$$t_{D500} = 9\ s$$

Para conseguir los tiempos más restrictivos de impacto, se debe tener en cuenta la velocidad de vuelo del buitre, puesto que el mínimo tiempo de impacto, se produce cuando el buitre y la aeronave vuelan uno en dirección al otro, produciendose así un impacto frontal. Además, es necesario tener en cuenta el tiempo medio de reacción del ser humano antes de pisar el freno de un vehículo, aproximadamente $0,75\ s$, considerado en el cálculo. En las Figuras 19, 20 y 21 se extraen los tiempos mínimos para cada uno de los casos anteriores, dependiendo de la distancia de detección y de la fase de vuelo.

Tiempos de colisión (250 m)						
Fase de vuelo de la aeronave	Velocidad de la aeronave (km/h)	Velocidad del ave (km/h)	Tipo de impacto	Distancia de detección (m)	Tiempo de reacción del piloto (s)	Tiempo de colisión (s)
Ascenso	150	48	Frontal	250	0,75	3,80
Nivelado/ Descenso	200		Frontal	250		2,88

Figura 19: Tiempos de colisión a $250\ m$

Tiempos de colisión (500 m)						
Fase de vuelo de la aeronave	Velocidad de la aeronave (km/h)	Velocidad del ave (km/h)	Tipo de impacto	Distancia de detección (m)	Tiempo de reacción del piloto (s)	Tiempo de colisión (s)
Ascenso	150	48	Frontal	500	0,75	8,34
Nivelado/ Descenso	200		Frontal	500		6,51

Figura 20: Tiempos de colisión a $500\ m$

Tiempos de colisión (1000 m)						
Fase de vuelo de la aeronave	Velocidad de la aeronave (km/h)	Velocidad del ave (km/h)	Tipo de impacto	Distancia de detección (m)	Tiempo de reacción del piloto (s)	Tiempo de colisión (s)
Ascenso	150	48	Frontal	1000	0,75	17,43
Nivelado/ Descenso	200		Frontal	1000		13,77

Figura 21: Tiempos de colisión a $1000\ m$

Estas velocidades ayudan a estimar el tiempo de procesamiento máximo de la FPGA y la distancia a la que la red neuronal debería ser capaz de detectar una posible ave, a partir de las imágenes de la cámara.

A continuación, se utilizan los datos obtenidos de la cámara Firefly para calcular su campo de visión, o FOV, por sus siglas en inglés. A partir de la apertura de la cámara se puede obtener el campo de visión dependiendo de la distancia utilizando la fórmula.

$$\frac{FoV}{2} = tan\left(\frac{\alpha}{2}\right) \cdot d$$

Donde α es el ángulo de abertura de la lente, en este caso 90^{o} y d es la distancia al objetivo. Teniendo esto en cuenta se puede calcular el FoV en km, dependiendo de la distancia a la que se encuentre el objeto. Los resultados se muestran en la Figura 22.

Campo de Visión (FoV)		
Ángulo de la lente (º)	Distancia al objetivo (m)	FoV (km)
90	1000	2 x 1,13
	500	1 x 0,56
	250	0,5 x 0,28

Figura 22: Campo de visión de la cámara en función de la distancia al ave

A partir de estos datos, conociendo los diferentes tamaños de imágen en píxeles atendiendo a la resolución de las cámaras y su campo de visión, así como el tamaño real del buitre y la distancia del mismo, es posible llevar a cabo un cálculo aproximado del número de píxeles que este ocupará en la imagen, utilizando la expresión:

$$Tam_{Ave}(px) = \frac{X_{Img}(px) \cdot Tam_{Ave}(m)}{X_{Img}(m)}$$

Los resultados se muestran en la Figura 23.

Tamaño del Ave				
Resolución de la cámara	Tamaño de la imagen (pixels)	Tamaño del ave (m)	Distancia al objetivo (m)	Tamaño del ave (píxeles)
4K	3840 x 2160	3	1000	5,8
			500	11,5
			250	23,0
HD	1920 x 1080	3	1000	2,9
			500	5,8
			250	11,5

Figura 23: Tamaño en píxeles del ave en función de la distancia y la resolución

De esta forma y a partir del desarrollo de la red neuronal clasificadora se puede realizar una estimación del número de píxeles aproximado que esta necesita para la detección del buitre, observando así la viabilidad del sistema. Será viable si la red neuronal es capaz de detectar el ave con un margen temporal lo suficientemente grande como para que el piloto pueda evitarla sin realizar movimientos bruscos, que puedan poner en peligro la estabilidad y el manejo de la aeronave.

Tomando estos resultados se puede extraer que una mayor resolución de imagen (4K) permite detectar el buitre más fácilmente a una mayor distancia, obteniendo por tanto un mayor margen de maniobras. Sin embargo, con una resolución tan alta, un dispositivo de bajo coste puede tener dificultades para procesar la información contenida en la imagen en el tiempo deseado.

En este caso, el dispositivo de bajo coste elegido, la FPGA, solo llega al procesamiento de video en HD (1080p). Como se puede comprobar, con esta resolución de vídeo el tamaño en píxeles del objetivo lleva a detectar las aves más cerca que mediante la tecnología 4K, dejando un margen de tiempo menor para la clasificación y la maniobra del piloto. En las siguientes secciones

se estudiará si esta resolución de vídeo, junto con la aceleración de la FPGA, es suficiente para establecer un tiempo de respuesta razonable.

3.2. Preparación de la PYNQ-Z2

En esta sección se recopila el trabajo realizado con la FPGA, previo a cargar la CNN, explicando brevemente la configuración de la misma, pasando por la instalación de Caffe, Theano y Lasagne y detallando los problemas encontrados en cada paso del trabajo, para finalmente tener la placa lista para la carga y el entrenamiento de la CNN.

3.2.1. Instalación y configuraciones previas

La FPGA, PYNQ-Z2 de Xilinx consta de varias opciones de conexión y comunicación con el equipo desde el que se va a llevar a cabo el desarrollo del proyecto. La primera de ellas es tener la placa directamente conectada al equipo, para ello la PYNQ-Z2 cuenta con una conexión UART que permite el acceso a la misma mediante un cable USB-UART y Putty o algún otro programa similar como Tera Term. Además, cambiando el "jumper" encargado de seleccionar la fuente de alimentación se puede modificar la placa para recibir alimentación a través del puerto UART con lo que no es necesario tenerla conectada a la luz, eso sí, debe de estar conectada a Ethernet en la misma red que lo está el ordenador.

La segunda opción es la de conectar la placa directamente al router por Ethernet y comunicarse con ella a partir de la terminal de Jupyter Notebooks, de esta forma no es necesaria la conexión directa entre la placa y el ordenador por lo que se cambia el "jumper" anteriormente mencionado para que la placa reciba la alimentación a través de la entrada de alimentación (esquina inferior izquierda) que se conecta directamente a la red eléctrica.

Con la placa instalada y conectada tanto a Ethernet como a la fuente de alimentación, es necesario grabar el archivo "PYNQ-Z2 PYNQ image" [28] en la tarjeta SD de 16 GB que se incluye con la placa. Para ello se utiliza el programa balenaEtcher, un utilitario libre y de código abierto de diseño simple e intuitivo que permite grabar archivos de imagen en tarjetas de memoria SD y dispositivos Flash USB (Figura 24).

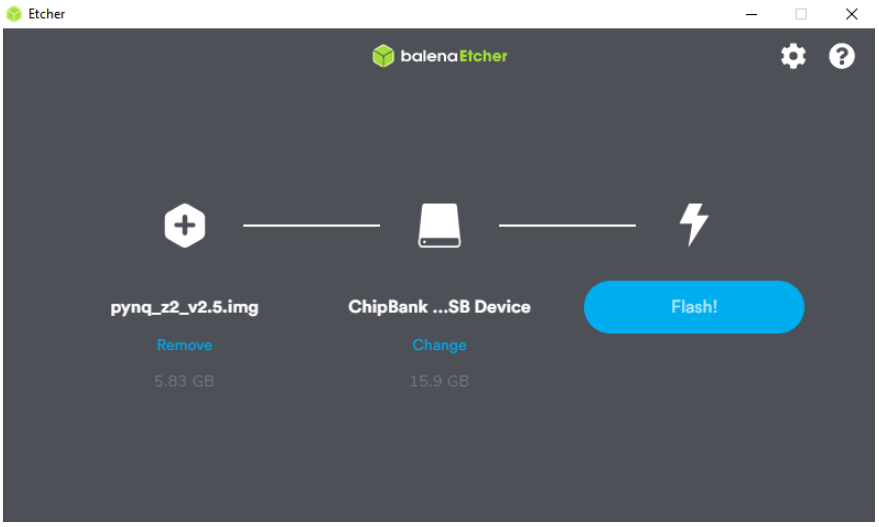


Figura 24: Software balenaEtcher

Una vez se ha grabado la imagen en la tarjeta SD, esta se introduce en la placa y se enciende el interruptor. En este proyecto se ha decidido, por comodidad, conectar la placa mediante Ethernet directamente al router y programar desde Jupyter Notebooks, para ello se escribe la siguiente dirección en el navegador “pynq:9090” y con la contraseña “xilinx” se accede a la página principal de Jupyter Notebooks como se muestra en la Figura 25.

Figura 25: Página principal de Jupyter Notebooks para la PYNQ-Z2

Desde esta, podemos abrir una terminal pulsando en el boton “new” (situado en las esquina superior derecha) y seleccionando la opción “terminal” del desplegable que aparecerá, una vez en este punto se procederá a la Instalación del framework de programación en Python.

3.2.2. Instalación del framework

Para la instalación del framework necesario para el desarrollo de la CNN se ha utilizado el manual de instalación (“MANUAL_INSTALL.md”) que se puede encontrar en el repositorio de github del proyecto de Erwei Wang [13], previamente mencionado en el apartado dedicado al estado del arte.

Cabe mencionar que las imágenes que se pueden encontrar tanto en esta sección como en las siguientes, no han sido extraidas directamente de los archivos de jupyter notebooks, ni de la terminal que esta plataforma contiene, si no que se han escrito los comandos en un editor de texto independiente como es *Sublime Text*, con la finalidad de facilitar la visualización de las imágenes y conseguir un aspecto más homogéneo. Además, se ha utilizado un código de colores concreto donde los comandos escritos en la terminal aparecen en fondo negro, mientras que las líneas de código de los archivos .ipynb (formato de las jupyter notebooks), aparecen en fondo blanco. El código original, tanto el utilizado en la terminal como el de los archivos .ipynb se puede encontrar en el Anexo IV.

En este proyecto se ha dividido la tarea de instalación en dos fases:

- Instalación de Caffe en la PYNQ
- Instalación de Theano con Lasagne

Instalación de Caffe en la PYNQ: Para la correcta instalación de Caffe, el primer paso es el de instalar los requisitos previos, para ello se ha utilizado el comando *sudo apt-get install* seguido del nombre de las librerías que se desea a instalar:

```
sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev
sudo apt-get install libopencv-dev libhdf5-serial-dev protobuf-compiler
sudo apt-get install --no-install-recommends libboost-all-dev
sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev
```

Cabe mencionar que en el caso de *libboost-all-dev*, adicionalmente al comando previamente mencionado, se ha añadido la especificación *–no-install-recommends*. Esto es debido a que Ubuntu, por defecto, instala los paquetes recomendados aparte de las dependencias y de esta forma se consigue que solo instale estas últimas.

A continuación se ha instalado el protobuf 3, utilizando python3, mediante el comando:

```
pip3 install protobuf
```

Seguidamente, debido a la falta de memoria RAM de la placa para instalar Caffe ha sido necesaria la creación de un espacio “swap“ utilizando:

```
mkswap /dev/sda
swapon /dev/sda
```

El comando *mkswap /dev/sda* se utiliza para crear una partición de memoria en el dispositivo seleccionado (gracias a la orden dev), en este caso se ha usado un USB de 30 GB de memoria. Con el espacio “swap” creado, el siguiente paso es la activación del mismo, para la que se ha utilizado el comando *swapon /dev/sda*.

```
sudo swapon --show
sudo swapoff -v /dev/swap
sudo rm /dev/swap
```

Además, se puede comprobar si el espacio de memoria está o no activado utilizando la orden *sudo swapon - -show*. Como respuesta a esta línea, se muestra una tabla con el nombre del espacio, su tipo (“file” tratándose de un archivo o “dev” si es un dispositivo), su tamaño, el espacio utilizado del mismo y su prioridad. También se pueden desactivar los espacios de memoria con el comando *sudo swapoff -v /dev/sda* y eliminarlos con *sudo rm /swapfile*.

Una vez activada la memoria adicional, desde el directorio */home/xilinx* se ha escrito la orden:

```
git clone https://github.com/BVLC/caffe.git
```

De esta forma ha sido posible clonar la página de github con los archivos necesarios y se ha instalado caffe, creándose una carpeta con cada uno de estos archivos en el directorio correspondiente.

Seguidamente, se ha introducido el archivo makefile para la configuración de Caffe (*Makefile.config*) dentro de la carpeta que acababa de crear. Para ello, se ha accedido a la carpeta

utilizando *cd Caffe*. Este archivo se puede encontrar en el respositorio de github de Erwei Wang, en el apartado de “tools” desde ahí es posible acceder al archivo en formato “raw“. A continuación se ha utilizado el comando *wget* con la dirección del archivo para descargarlo en la FPGA. También es posible copiar el archivo *Makefile.config.example* y llamarlo como el archivo deseado, utilizando el comando:

```
cp makefile.config.example makefile.config
```

En ambos casos es necesario modificar el archivo *Makefile.config*, para ello se ha utilizado un editor de texto como *Vi* [29], que permite editar el archivo desde la propia terminal, utilizando el comando:

```
vi Makefile.config
```

Al abrir el archivo se ha podido observar que este está pensado para la utilización de Python 3.4 mientras que la versión de Python que viene por defecto con la placa en la actualidad es el Python 3.6 por lo que ha sido necesario actualizar los directorios. El proceso de actualización de los directorios se explica en el apartado de errores encontrados durante el proyecto.

Una vez se tenía el fichero *Makefile.config* configurado correctamente, el siguiente paso realizado ha sido la instalación de Caffe, mediante los comandos:

```
make all
make test
make runtest
```

Además, como se explica en el apartado de errores encontrados se ha añadido el comando *make pycaffe*.

Instalación de Theano con Lasagne: La instalación de Theano por una parte y Lasagne por la otra han resultado ser mucho más sencillas que la de Caffe. El código del respositorio instalaba una versión de Theano no actualizada para python3.6 por lo que se ha instalado de forma manual la versión correcta [30] mediante la siguiente orden:

```
pip3 install Theano
```

Por último se ha procedido a la instalación de Lasagne utilizando los comandos:

```
pip3 install -r https://raw.githubusercontent.com/Lasagne/Lasagne/v0.1/requirements.txt
pip3 install Lasagne==0.1
```

3.3. Tests de funcionamiento y aceleración

En esta sección se comprobueba el funcionamiento de la red neuronal de tipo LeNet comprendida en el proyecto de PYNQ-Classification de Erwei Wang, entrenada para el reconocimiento de escritura, utilizando para ello la base de datos escritura a mano: MNIST dataset database. El archivo “Using a Caffe Pretrained Network - LeNet5.ipynb“ en el que se haya la red neuronal previamente mencionada se puede encontrar en el AnexoIV, en la sección “Código Original LeNet“.

Este archivo se ha utilizado de base para realizar los diferentes pasos descritos en esta sección.

El primero de los objetivos buscados durante esta parte del proyecto ha sido el de **ejecutar la red neuronal sin la aceleración de la FPGA**, para comprobar de esta forma que todo el framework previamente mencionado ha sido instalado correctamente. Para ello se ha modificado ligeramente el código de ejecución del archivo "Using a Caffe Pretrained Network - LeNet5.ipynb" comentando los comandos que requerían del acceso a la aceleración de la FPGA y comprobando así que la red neuronal lleva a cabo las predicciones correctamente. A continuación se muestra una pequeña explicación de los cambios que se han realizado para ejecutar el archivo sin necesidad de aceleración.

La primera de las modificaciones la encontramos en la celda 3 (Figura 26) donde, a diferencia del código original, las líneas 7, 8, 9 y 11 relacionadas con la activación de la aceleración de la FPGA han sido comentadas.

```
1  import lasagne
2  from lasagne.layers import InputLayer, DropoutLayer, DenseLayer, NonlinearityLayer
3  #from lasagne.layers.dnn import Conv2DDNNLayer as ConvLayer
4  from lasagne.layers import Conv2DLayer as ConvLayer
5  from lasagne.layers import Pool2DLayer as PoolLayer
6  from lasagne.nonlinearities import softmax, rectify, linear
7  #import conv_fpga
8  #from conv_fpga import FPGA_LENET
9  #from conv_fpga import FPGAQuickTest
10 #from conv_fpga import Conv2DLayer as ConvLayer
11 #from conv_fpga import FPGAWeightLoader as FPGALoadW
12 from lasagne.utils import floatX
```

Figura 26: Celda 3 modificada sin aceleración

Esto se vuelve a repetir en la celda 6 (Figura 27), en este caso se han comentado los pesos correspondientes a la FPGA (líneas 3, 5, 9 y 13) manteniendo el resto.

```
1  #FPGALoadW(weight, status, IFDim, OFDim, PadDim)
2  weight = net['conv1'].W.get_value()
3  #FPGALoadW(weight, 1, 28, 24, 0)
4  weight = net['conv2'].W.get_value()
5  #FPGALoadW(weight, 2, 12, 8, 0)
6  weight = net['ip1'].W.get_value()
7  weight = np.transpose(weight)
8  weight = weight.reshape(500, 50, 4, 4)
9  #FPGALoadW(weight, 3, 4, 1, 0, flip_filters=False)
10 weight = net['ip2'].W.get_value()
11 weight = np.transpose(weight)
12 weight = weight.reshape(10, 500, 1, 1)
13 #FPGALoadW(weight, 4, 1, 1, 0, flip_filters=False)
```

Figura 27: Celda 6 modificada sin aceleración

Además de estas modificaciones, se han eliminado las celdas de la 9 a la 14, saltando directamente a la 15 y 16. Y se ha creado una nueva celda 17 modificando la anterior celda 14 pero sin utilizar la aceleración de la FPGA, para ello se ha sustituido la línea 12 cambiando la orden $pred = FPGA\_predicted[i]$ por $pred = predicted[i]$ como se muestra en la Figura 28.

```python
def make_image(X):
    im = np.swapaxes(X.T, 0, 1)
    im = im - im.min()
    im = im * 1.0 / im.max()
    return im

plt.figure(figsize=(16, 5))
for i in range(0, 10):
    plt.subplot(1, 10, i+1)
    plt.imshow(make_image(test_data[i][0]), interpolation='nearest', cmap=plt.get_cmap('gray'))
    true = test_label[0][i]
    pred = predicted[i]
    color = 'green' if true == pred else 'red'
    plt.text(0, 0, true, color='black', bbox=dict(facecolor='white', alpha=1))
    plt.text(0, 32, pred, color=color, bbox=dict(facecolor='white', alpha=1))

    plt.axis('off')
```

Figura 28: Celda 17 sin aceleración

Con las modificaciones realizadas se ha ejecutado el código a modo de comprobación, obteniendo así las predicciones correctas tal y como se puede apreciar en la Figura 29.

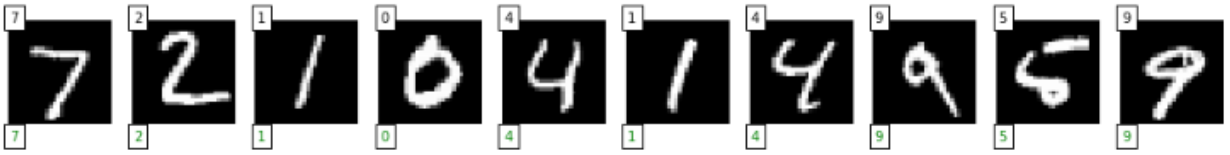


Figura 29: Predicciones LeNet sin aceleración

Una vez se ha comprobado que la red funciona correctamente sin aceleración, se han introducido de nuevo los comandos previamente comentados para comprobar si la red neuronal funciona correctamente con la aceleración de la FPGA. Durante este proceso ha sido necesario solucionar diferentes mensajes de error, relacionados con el fichero “conv_fpga.py“, que se recogen en la sección de “Errores encontrados durante los test de funcionamiento“. Además, debido al cambio de versión de python realizado aparecen varios mensajes de advertencia que para el estudio que se está realizando no tienen mayor importancia.

Con los errores y obviando los mensajes de advertencia previamente mencionados, se comprueba que la LeNet realiza las predicciones de forma satisfactoria y lo último que resta es la comparación de los tiempos de procesado en ambos casos, para comprobar que la FPGA es capaz de acelerar el tiempo de ejecución de la LeNet. En la Figura 30 se muestran el tiempo de procesamiento de la LeNet, sin acelerar y acelerada respectivamente.

Tiempos de procesamiento			
	Tiempo CPU (s)	Tiempo Sistema (s)	Total (s)
Tiempo sin aceleración	11,88	0,32	12,20
Tiempo con aceleración	0,89	0,04	0,93

Figura 30: Tiempos de procesameinto

Como se puede observar la diferencia entre ambos periodos es notable, puesto que se pasa

de $12,2\ s$ a $0,93\ s$, suponiendo esto una aceleración de aproximadamente el $92\,\%$.

Hasta este punto, se ha ejecutado correctamente el modelo preentrenado del proyecto de PYNQ-Classification a la FPGA resolviendo todos los problemas que se han encontrado por el camino y se ha conseguido acelerar con éxito este modelo utilizando la FPGA.

3.4. Transferencia del modelo

Después de la ejecución del modelo base, el siguiente consiste en la transferencia de uno de los modelos que se encuentran en el repositorio del proyecto del profesor R.Yoshihashi [7].

Los modelos se han transferido con éxito utilizando los archivos *.prototxt* y *.caffemodel* cuyo código se puede encontrar en el Anexo IV. En general, el fichero *.prototxt* contiene la descripción de la red neuronal diferenciando entre cada una de las capas de la misma, mientras que el fichero *.caffemodel* almacena los pesos derivados del entrenamiento.

Como ya se ha mencionado en la sección dedicada al estado del arte, en el artículo del profesor R.Yoshihashi, se trabaja con imágenes de 28x28 píxeles debido a que previamente a la clasificación de la CNN se aplica un algoritmo de sustracción de fondo, obteniendo así las zonas en las que es más probable que haya un ave. Como se ha explicado en la sección de antecedentes, en el apartado de sustracción de fondo dinámico, en este estudio en concreto un algoritmo de sustracción de fondo estático no es viable, puesto que la cámara está en movimiento y por tanto el fondo del vídeo tendrá un comportamiento dinámico.

En la sección de estado del arte, se menciona la utilización de un algoritmo de fondo dinámico en el estudio de Jaechan Cho y su equipo [14]. En este artículo se explica como implementar un algoritmo de fondo dinámico utilizando una FPGA, pero no incluye el código necesario para hacerlo. Este proceso se podría implementar pero se escapa del horizonte del proyecto debido a la cantidad de tiempo necesaria para hacerlo.

3.5. Errores encontrados durante el proyecto

En esta sección se recogen todas las incidencias encontradas durante la realización del proyecto, así como las soluciones que se les dieron durante el mismo, con la finalidad de evitar posibles eventualidades en futuros trabajos que pretendan realizar un estudio similar al presente. Además, cada una de los errores que se mencionan a continuación, constan de una referencia al Anexo de Código, en el que se muestra el mensaje de error obtenido durante la ejecución del código ya sea en el terminal o en el propio script de jupyter notebooks.

3.5.1. Errores durante la instalación del framework

La primera de las dificultades la encontramos en la instalación de los requisitos previos, ya que al utilizar el comando *sudo apt-get install* junto al nombre de las librerías seleccionadas, como se ha mencionado con anterioridad, aparecía un error de tipo *Unable to locate package* (Anexo IV Figura **IV**-1) . Para solucionar este error se utilizaron los siguientes comandos de forma consecutiva.

```
sudo apt-get update
sudo apt-get upgrade
```

Durante la ejecución del comando *make all* se registró un error de tipo *No such file or directory* (Anexo IV Figura **IV**-2) debido a la versión de python utilizada, ya que, como se ha mencionado con anterioridad el código original del que se ha partido para realizar este trabajo estaba pensado para python3.4 mientras que en este proyecto se ha utilizado la versión 3.6, lo que ha provocado una elevada cantidad de contratiempos a lo largo del desarrollo del mismo. En este caso el fichero pyconfig.h no se podía encontrar debido a que las direcciones de los directorios de python en el fichero "Makefile.config" estaban configuradas para la versión 3.4. Es por esto que se llevó a cabo una actualización de las mismas utilizando el editor de textos *vi*. En las Figuras 31 y 32 se muestran las líneas editadas dentro del fichero "Makefile.config".

```
# Uncomment to use Python 3 (default is Python 2)
PYTHON_LIBRARIES := boost_python3 python3.4
PYTHON_INCLUDE := /usr/include/python3.4 \
        /usr/include \
                /usr/local/lib/python3.4/dist-packages/numpy/core/include

# We need to be able to find libpythonX.X.so or .dylib.
PYTHON_LIB := /usr/lib/python3.4/config-3.4m-arm-linux-gnueabihf
# PYTHON_LIB := $(ANACONDA_HOME)/lib
```

Figura 31: Código original para Python 3.4

```
# Uncomment to use Python 3 (default is Python 2)
PYTHON_LIBRARIES := boost_python3 python3.6
PYTHON_INCLUDE := /usr/include/python3.6 \
        /usr/include \
                /usr/local/lib/python3.6/dist-packages/numpy/core/include

# We need to be able to find libpythonX.X.so or .dylib.
PYTHON_LIB := /usr/lib/python3.6/config-3.6m-arm-linux-gnueabihf
# PYTHON_LIB := $(ANACONDA_HOME)/lib
```

Figura 32: Código modificado para Python 3.6

Como se puede observar, una vez dentro del editor de texto basta con sustituir python3.4 por python3.6.

Otro de los errores registrados durante la configuración del "Makefile.confi" tuvo relación con la selección de OpenBlas como la librería Blas por defecto, en este caso el error vino acompañado del anterior y se solucionó antes si quiera de haberse detectado, sin embargo se ha creído interesante mencionarlo.

```
# BLAS choice:
# atlas for ATLAS (default)
# mkl for MKL
# open for OpenBlas
BLAS := open
```

(a) Código original

```
# BLAS choice:
# atlas for ATLAS (default)
# mkl for MKL
# open for OpenBlas
BLAS := Open
```

(b) Código modificado

Figura 33: Modificación de código para solucionar el error de OpenBlas

En la Figura 33 se muestra la modificación realizada, siendo esta tan simple como transformar en mayúscula la “o“ de open, la solución de este error viene descrita en el enlace correspondiente de la bibliografía [31].

Una vez instalado Caffe, se comprobó el correcto funcionamiento mediante el comando:

```
import Caffe
```

Encontrando que este devolvía errores relacionados con la compilación, conduciendo a un mensaje de error de tipo *invalid syntax* (Anexo IV Figura **IV**-3). Para solucionar dicho error se utilizaron las siguientes ordenes:

```
make clean
make all
make test
make runtest
make pycaffe
```

El comando *make clean* se utilizó para limpiar la configuración previamente establecida, para realizar de nuevo el mismo proceso explicado en la sección anterior, añadiendo esta vez el comando *make pycaffe* como se explica en el siguiente enlace [32].

3.5.2. Errores encontrados durante los test

Por lo que a los test se refieren, cabe mencionar que todos los errores se dieron durante la prueba de la LeNet utilizando la aceleración de la FPGA. El primero de estos errores lo encontramos en el comando *import conv_fpga*, Figura 34, línea 7.

```
import lasagne
from lasagne.layers import InputLayer, DropoutLayer, DenseLayer, NonlinearityLayer
#from lasagne.layers.dnn import Conv2DDNNLayer as ConvLayer
from lasagne.layers import Conv2DLayer as ConvLayer
from lasagne.layers import Pool2DLayer as PoolLayer
from lasagne.nonlinearities import softmax, rectify, linear
import conv_fpga
from conv_fpga import FPGA_LENET
from conv_fpga import FPGAQuickTest
#from conv_fpga import Conv2DLayer as ConvLayer
from conv_fpga import FPGAWeightLoader as FPGALoadW
from lasagne.utils import floatX
```

Figura 34: Celda 3 con aceleración

El primero de los errores resultantes de la ejecución de esta línea de código estaba relacionado con el archivo *im2col_lasagne_cython.pyx* y era de tipo *Module not found* (Anexo IV Figura **IV**-4). Este archivo era llamado directamente desde el fichero *conv_fpga* mediante la orden:

```
1   import im2col_lasagne_cython
```

Para solucionar este problema primero se solventaron primero los errores encontrados en el modulo Cython desde la terminal mediante la orden:

```
1   python3.6 -m pip install Cython --install-option="--no-cython-compile"
```

Y viendo que el comando *import im2col_lasagne_cython* seguía dando problemas, se ejecutaron los siguientes comandos de forma consecutiva.

```
1   python3.6 setup.py build
2   python3.6 setup.py install
3   python3.6 -m pip install UNKNOWN
```

De esta forma se construyó e instaló el fichero *setup.py* para cargar posteriormente el archivo *im2col_lasagne_cython*.

El siguiente error era del mismo tipo que el anterior *Module not found*. Sin embargo, en este caso estaba relacionado con el archivo *acc8.pyx* (Anexo IV Figura **IV**-5). El cual también se llamaba desde *conv_fpga* con el comando:

```
1   import acc8
```

Para solucionar este problema, primero se creó un fichero *setup2.py* similar al ya utilizado en el error anterior y se usaron los comandos necesarios para crear e instalar el fichero, además se desisntaló y volvió a instalar “Unknown“ como se observa en las órdenes utilizadas:

```
1   python3.6 setup2.py build
2   python3.6 setup2.py install
3   python3.6 -m pip3 uninstall UNKNOWN
4   python3.6 -m pip3 install UNKNOWN
```

Esto condujo a un error de tipo *Undefined symbol* (Anexo IV Figura **IV**-6). Para solucionarlo se utilizaron las siguientes órdenes:

```
1   python3.6 setup2.py build_ext --inplace
2   rm acc8.cpython-36m-arm-linux-gnueabihf.so
3   ls -la /usr/local/lib/python3.6/dist-packages/acc8.cpython-36m-arm-linux-gnueabihf.so
```

Como se puede observar, en este caso se concretó el uso del comando build, añadiendole la opción “- -inplace“ que mueve los recursos compilados a la ruta de búsqueda de python. El comando rm se utilizó para borrar el archivo *acc8.cpython-36m-arm-linux-gnueabihf.so*.

El problema persistió en este caso y eso es debido a que cython intentaba compilar la versión de python3.6 pero ya existía una versión para python3.4. Para solucionar este problema se creó

un enlace simbólico al fichero *acc8.cpython-34m.so* para que la versión de python3.6 lo usara con el nombre *acc8.cpython-36m.so*. A continuación se le cambió el nombre al fichero acc8 que había en la carpeta de librerías dev python3.6 a .bak para poder hacer el siguiente paso. Por último se creó un enlace simbólico en la carpeta de librerías de Python3.6 para que apuntara al fichero acc8 que hay en Theano solucionando así el problema. Los comandos utilizados para realizar las acciones mencionadas se muestran a continuación:

```
ln -s acc8.cpython-34m.so acc8.cpython-36m.so
mv /usr/local/lib/python3.6/dist-packages/acc8.cpython-36m-arm-linux-gnueabihf.so
    /usr/local/lib/python3.6/dist-packages/acc8.cpython-36m-arm-linux-gnueabihf.so.bak
ln -s /home/xilinx/jupyter_notebooks/PYNQ-Classification/PYNQ_SIDE/Theano/Lenet/acc8.cpython-34m.so
    /usr/local/lib/python3.6/dist-packages/acc8.cpython-36m-arm-linux-gnueabihf.so
```

Seguidamente, surgieron dos errores más, el primero de ellos era de tipo *Module not found* (Anexo IV Figura **IV**-7) y estaba relacionado con el módulo pynq.drivers el cual es utilizado en el archivo *conv_fpga.py* en la línea 135, con el comando:

```
from pynq.drivers import DMA
```

El segundo de estos errores era de tipo *Key error* (Anexo IV Figura **IV**-8). El problema se debía a que la función para la versión 3.6 de python esperaba una entrada del diccionario como parámetro de la forma

```
base_addr=PL.ip_dict[name]["phys_addr"]
```

Mientras que en la versión 3.4 se le daba una dirección de memoria.

```
base_addr=int(PL.ip_dict["SEG_{0}_Reg".format(name)][0],16)
```

Para solucionar este problema se actualió el fichero *conv_fpga.py* sustituyendo las líneas necesarias para hacer funcionar el código en python3.6, como se ha mostrado en las figuras anteriores. Además también se realizaron más cambios atendiendo a un nuevo repositorio de github citado en la bibliografía [33], con lo que fue posible corregir posibles errores antges de que llegaran a producirse, razón por la que no aparecen recopilados en esta sección.

Por último, cabe mencionar que todas las soluciones a los errores que se han descrito en el subapartado correspondiente a los test de funcionamiento han sido recopiladas a partir de las fuentes citadas en la bibliografía [33] y [34] .

3.6. Conclusiones

En esta última sección de la memoria se recopilan los resultados y las valoraciones obtenidas a partir del desarrollo del proyecto de estudio y de la redacción de la memoria correspondiente. Además, se plantearán las principales vías de continuación de este proyecto con la finalidad de que, en un futuro, estas puedan ser retomadas, llevando al mismo a un escalón más elevado de complejidad.

En primer lugar, ha sido necesario llevar a cabo una rigurosa recopilación bibliográfica, en la que se ha reunido información de la situación actual en lo que a ”bird-strikes”se refiere. La información obtenida a partir de esta, así como los diferentes casos expuestos hacen posible el entendimiento de los principales factores que dotan de sentido y necesidad al presente trabajo. Factores como la gravedad de los accidentes provocados por estos o la frecuencia de los mismos, resultan determinantes a la hora de plantear el problema para el que este proyecto busca solución. Conjuntamente con esto, encontramos un gran número de trabajos dedicados a la clasificación y el seguimiento de las aves, que aportan un gran variedad de enfoques diferentes y formas de abordar el problema previamente presentado, posibilitando la adquisición de nuevos conocimientos en el campo del procesamiento de imágenes y aumentando el abanico de posibles estrategias a seguir para su resolución.

Además, con el objetivo de ilustrar al lector, se ha llevado a cabo un estudio del marco teórico en el que se fundamente este trabajo de fin de grado. En él se plantean conceptos básicos relacionados con la inteligencia artificial, el análisis de datos y los diferentes tipos de redes neuronales capaces de resolver el problema planteado con anterioridad. Se presentan nuevos problemas, derivados de las condiciones iniciales del mismo, como la necesidad de aumentar la velocidad de procesamiento y soluciones a estos, como la utilización de una FPGA, siguiendo una estructura lógica en la exposición de los conocimientos y mostrando los principales algoritmos con los que se ha trabajado.

Con los elementos esenciales para la contextualización y la comprensión del trabajo realizado ya presentados se ha pasado a describir las diferentes fases de este proyecto, comenzando por los cálculos previos. Estos cálculos, se han utilizado de modo orientativo, dándo una idea aproximada sobre las maginutdes de tiempo y tamaño con los que se va a tratar en el proyecto. Con la finalidad de dotar de peso y realidad el problema teórico expuesto con anterioridad.

La siguiente de las fases tratadas es la de la configuración de la FPGA, en ella se han explicado de forma fácil y concisa los diferentes pasos realizados en este proyecto, con el objetivo de facilitar el seguimiento de los mismos. Para ello se han adjuntado un gran número de imagenes con las líneas de código, así como un código de colores para diferenciar los comandos utilizados en la terminal, del código de las libretas de jupyternotebooks. Posteriormente, se han expuesto brevemente los test realizados, para comprobar el correcto funcionamiento del framework instalado durante la fase anterior, mostrando cada uno de los cambios realizados, así como las respuestas obtenidas para finalmente realizar una valoración de la aceleración de la red neuronal en la FPGA.

En lo referente a las dificultadas encontradas, se ha llevado a cabo una clasificación de los diferentes errores encontrados, dependiendo de la fase en la que estos hayan aparecido, describiéndolos ampliamente y aportando soluciones suficientemente detalladas para cada uno de ellos, de forma que puedan ser corregidos con facilidad durante la ejecución del código. Además, se ha creado un Anexo específico en el que se han incluido los fragmentos de código más rele-

vantes para la compresión del trabajo, además de las respuestas de la terminal y los códigos de los errores recopilados durante la realización del proyecto.

Cabe mencionar que debido a la complejidad del proyecto y al gran número de errores encontrados durante el mismo, éste no se ha podido llevar hasta el estado de desarrollo deseado en un principio, dejando varias vías libres para la continuación y el desarrollo del mismo en futuros proyectos. Entre los temas más interesantes a realizar encontramos la transferencia de un modelo de clasificación de aves en la plataforma que este proyecto ha conseguido crear y acelerar, la implementación de un algoritmo de sustracción de fondo dinámico o en su defecto, un algoritmo de sliding window para aplicar el clasificador de aves ya acelerado, consiguiendo así un programa capaz de detectar, clasificar y seguir aves en un vídeo, aplicando de esta forma la tecnología de la cámara 4k, que en este TFG solo ha dado tiempo a mencionar. Para ello se ha pensado en enviar las imágenes de la cámara a través del puerto HDMI, a la FPGA [35], donde las procesaría el algoritmo de análisis basado en una CNN. Para ello haría falta determinar el número de imágenes por segundo que podrá procesar tanto la sustracción dinámica de fondo, como la red neuronal.

No se ha tratado en este TFG la parte de la captura de imágenes que van a ser analizadas por la red neuronal. Con el hardware disponible hay dos opciones: capturar imágenes individuales a través del puerto USB o capturar el vídeo enviado por la cámara a través del puerto HDMI.

En la primera solución se puede utilizar el puerto USB de la PYNQ-Z2 para recibir las imágenes de la cámara y pasárselas a la FPGA. La cámara utilizada como referencia, la Hawkeye Firefly 8SE 4K 90º, tiene puerto USB con salida A/V. Existen varias librerías que permiten capturar imágenes estáticas a través del puerto USB, como opencv o fswebcam. Hay que valorar cuál de ellas permite realizar la captura de forma más rápida y probar si la PYNQ es capaz de tratar las imágenes a velocidad suficiente para realizar la detección. Esta opción supone sacrificar el número de imágenes por segundo con respecto al vídeo en tiempo real, pero puede permitir trabajar a una resolución de 4K e incluso superior, ya que la cámara Firefly puede trabajar con imágenes estáticas de hasta 16 Mpixels [37]
y para detectar la presencia de buitres no es necesario procesar 25 o 30 imágenes por segundo.

Para la segunda solución se puede utilizar el puerto HDMI de la PYNQ-Z2, pero trabajando solo hasta resolución Full-HD, pues esta es la máxima resolución a la que puede trabajar este hardware. Para trabajar con vídeo 4K a tiempo real se requeriría de una FPGA de mayor capacidad. Una buena opción sería la ZCU104 de Xilinx, de la que se puede obtener un entorno de desarrollo por 1250 euros [36].

Además sería necesario implementar la parte de hardware de alerta al usuario, para la que se ha pensado en una matriz de leds que informará de la localización del buitre, iluminando los leds correspondientes a la posición del mismo con respecto a la aeronave y un aviso sonoro que informará de la proximidad del buitre cambiando su frecuencia. Así como la parte de software para la que se deben tener en cuenta las especificaciones requeridas por este tipo de sistemas.

II PLIEGO DE CONDICIONES

Introducción

En este Anexo se recogerán todos aquellos aspectos y requerimientos necesarios para realizar el proyecto explicado en la memoria, manteniendo unos estándares de seguridad, higiene y salud propias durante las diferentes facetas del mismo. Estos se dividirán atendiendo a su condición en, **requerimientos generales** y **requerimientos específicos**.

Requerimientos generales

Los requerimientos generales hacen referencia a aspectos estrictamente relacionados con el desarrollo de un proyecto, independientemente del campo de estudio del mismo. Estos certifican la seguridad del proyecto así como del entorno en el que se ha realizado. Estos requerimientos se satisfarán de acuerdo a los estándares estipulados en el Real Decreto 486/1997, por el que se establecen las disposiciones mínimas de seguridad y salud en el entorno laboral.

Entorno de trabajo

Cabe mencionar que, debido a la naturaleza de este trabajo, así como a las medidas extraordinarias acaecidas a partir de la crisis del COVID-19, el entorno de trabajo de este proyecto se ha limitado a una vivienda en la localidad de Valencia, la cual cumple con todos los requerimientos establecidos para una vivienda habitual. Además, debido al carácter mayoritariamente computacional de este TFG, ha sido más que suficiente para asegurar la correcta realización del mismo, de forma segura e higiénica.

El entorno de trabajo escogido para realizar este proyecto cuenta además con conexión a internet de forma estable. Esta es necesaria, puesto que la comunicación entre el equipo y la FPGA se ha llevado a cabo de forma remota mediante la red local. Además, también se ha utilizado la conexión a internet para la comunicación con el tutor, así como la propia redacción de la memoria del proyecto, puesto que para ello se ha utilizado la plataforma online LATEX.

Requerimientos específicos

Los requerimientos específicos recogen las especificaciones relacionadas con el propio proyecto. En este caso se incluyen los conocimientos necesarios, tanto por parte del supervisor, como por lo que se refiere al ingeniero ejecutor del proyecto. Además, adicionalmente, se ha creido conveniente crear una sección dedicada a la especificaciones del material utilizado en este proyecto, como el equipo informático o la FPGA.

Supervisión y conocimientos

La realizacion del presente proyecto se debe llevar a cabo bajo la supervisión del profesor tutor, con amplios conocimientos en el campo de la navegación aérea y el análisis de datos. Capaz de guiar al ingeniero encargado de realizar el trabajo en las diferentes fases del proyecto aportando consejos y valoraciones.

Como bien se ha mencionado, debido al confinamiento sufrido durante parte del tiempo de duración del proyecto, esta supervisión no ha podido realizarse de forma presencial por lo que el

profesor ha realizado el seguimiento del proyecto mediante correspondencia por e-mail.

Por su parte el ingeniero encargado de realizar el proyecto debe ser capaz de interpretar código y estar familiarizado con el lenguaje de programación python. Además, debe ser capaz de procesar e interpretar los datos y tener una gran capacidad de solventar errores o funcionamientos irregulares que pueden ser reducidos mediante las regulaciones pertinentes.

Especificaciones de material

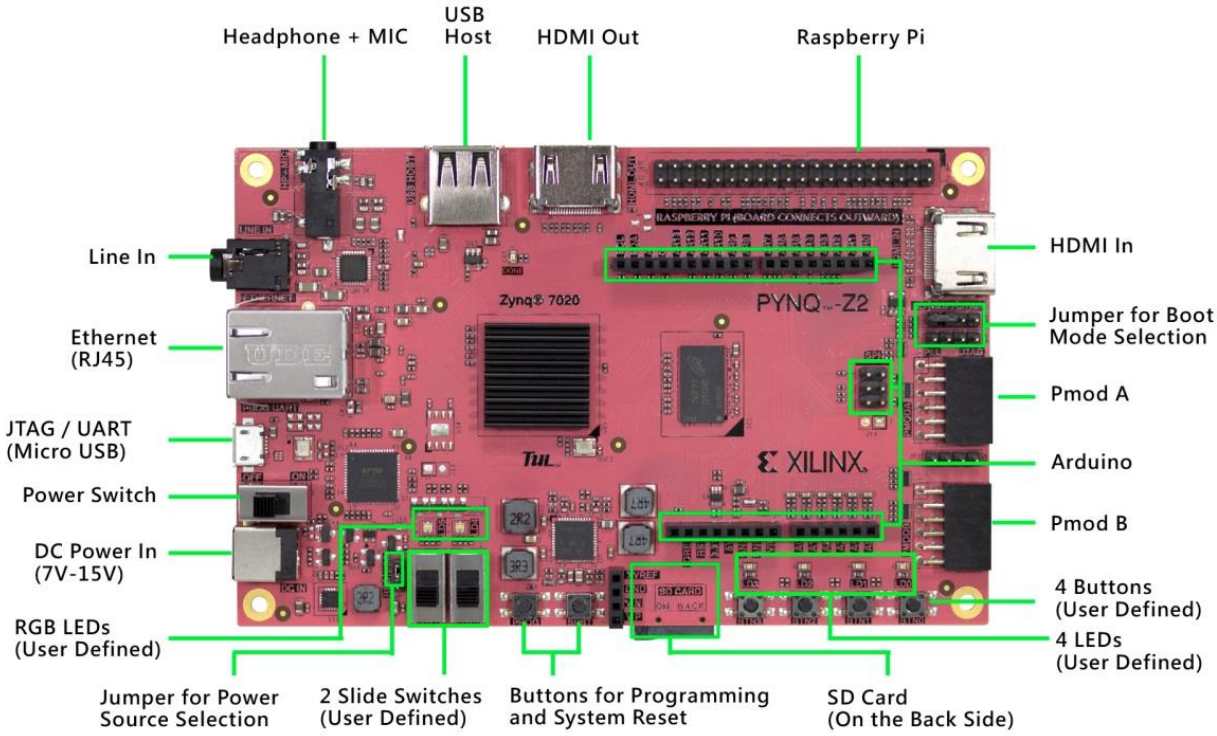
Por úlrimo se ha creido importante realizar una recopilación de las características de hardware de los diferentes equipos que se han utilizado para el desarrollo de este proyecto. Siendo estos el equipo informático y la FPGA.

En lo referente al equipo informático, se adjuntan sus especificaciones de hardware a continuación de forma orientativa.

- Procesador: Intel Core i5-7600K 3.8GHz
- Tarjeta gráfica: Gigabyte GeForce GTX 1050Ti OC WindForce 4GB GDDR5
- RAM: G.Skill Aegis DDR4 2133 PC4-17000 8GB CL15
- Disco duro: Seagate BarraCuda 3.5”1TB SATA3

Como se ha venido mencionando a lo largo de la memoria realizada, en este proyecto se ha utilizado una FPGA para la aceleración del proceso de clasificación de la red neuronal. Es por esto que se ha creído necesario adjuntar las especificaciones técnicas de la misma, de forma que estas puedan ser consultadas con facilidad.

Introducing TUL PYNQ™-Z2 Advance kit



ZYNQ XC7Z020-1CLG400C
- 650MHz dual-core Cortex-A9 processor
- DDR3 memory controller with 8 DMA channels and 4 High Performance AXI3 Slave ports
- High-bandwidth peripheral controllers: 1G Ethernet, USB 2.0, SDIO
- Low-bandwidth peripheral controller: SPI, UART, CAN, I2C
- Programmable from JTAG, Quad-SPI flash, and microSD card
- Programmable logic equivalent to Artix-7 FPGA
  - 13,300 logic slices, each with four 6-input LUTs and 8 flip-flops
  - 630 KB of fast block RAM
  - 4 clock management tiles, each with a phase-locked loop (PLL) and mixed-mode clock manager (MMCM)
  - 220 DSP slices
  - On-chip analog-to-digital converter (XADC)

Memory
- 512MB DDR3 with 16-bit bus @ 1050Mbps
- 16MB Quad-SPI Flash with factory programmed 48-bit globally unique EUI-48/64™ compatible identifier
- microSD slot

Power
- Powered from USB or 7V-15V external power source

USB and Ethernet
- Gigabit Ethernet PHY
- Micro USB-JTAG Programming circuitry
- Micro USB-UART bridge
- USB 2.0 OTG PHY (supports host only)

Audio and Video
- HDMI sink port (input)
- HDMI source port (output)
- I2S interface with 24bit DAC with 3.5mm TRRS jack
- Line-in with 3.5mm jack

Switches, Push-buttons and LEDs
- 4 push-buttons
- 2 slide switches
- 4 LEDs
- 2 RGB LEDs

Expansion Connectors
- Two standard Pmod ports
  - 16 Total FPGA I/O (8 shared pins with Raspberry Pi connector)
- Arduino Shield connector
  - 24 Total FPGA I/O
  - 6 Single-ended 0-3.3V Analog inputs to XADC
- Raspberry Pi connector
  - 28 Total FPGA I/O (8 shared pins with Pmod A port)

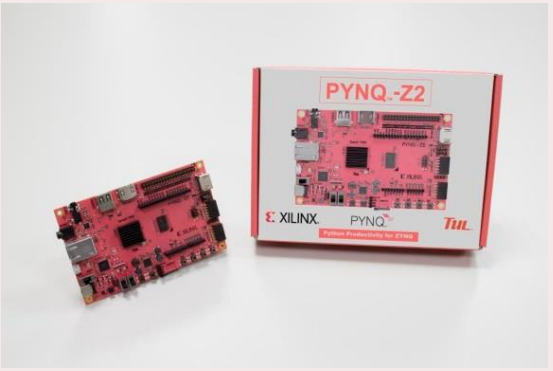
Technology Un-Limited

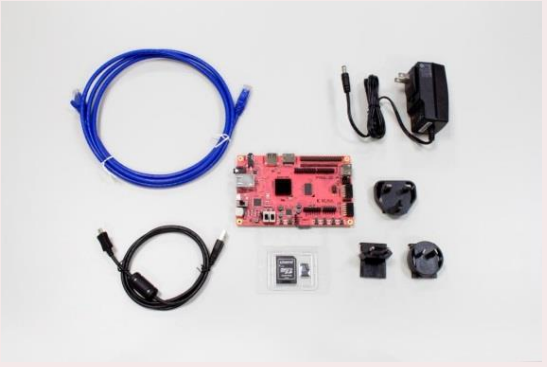
TUL PYNQ™-Z2 Advance kit

Product Specification	
Part number	1M1-M000127DVA
EAN Code:	TUL PYNQ-Z2 Advance Kit | 4713436170846
Processor:	Dual-Core ARM Cortex-A9
FPGA:	1.3M reconfigurable gates
Memory:	512MB DDR3 / 128Mbit FLASH
Storage:	Micro SD card slot
Video:	HDMI In / HDMI Out
Audio:	HP+Mic, Line in, ADAU1761 AUDIO codec
Network:	10/100/1000 Ethernet
Expansion:	USB Host connected to ARM PS
Interfaces GPIO:	Arduino Header, Raspberry Pi, Pmods x2
Other I/O:	6x User LEDs, 4x Pushbuttons, 2x Switches
Dimensions:	87mm x 140mm

Accessories	
1	Micro SD card 8G (image preload)
2	AC/DC adapter
3	USB type-A to Micro USB Cable
4	Ethernet cable
5	Protection acrylic case
6	Protection zipper case

Photos of Product





Technology Un-Limited

III PRESUPUESTO

Introducción

En este Anexo se realizará una estimación del coste monetario que supone este Trabajo de Fin de Grado, aportando una cifra orientativa sobre el presupuesto total necesario para llevarlo a cabo. Para ello, se considerará tanto el coste humano como otros costes derivados del proyecto e igualmente necesarios para la realización del mismo. Puesto que el proyecto ha durado aproximadamente unos 4 meses, se considerará está cifra a la hora de estimar las amortizaciones correspondientes.

Así pues, cabe mencionar que en este presupuesto se utilizará el euro (moneda conjunta de la Unión Europea) como unidad monetaria. Finalmente al valor obtenido se le añadirá el 21 % de IVA.

Desglose de gastos

En esta sección se expone el deglose de los gastos realizados durante el proyecto. Estos se organizan atendiendo a la naturaleza de los mismos, diferenciando entre **coste del material**, **coste de humano** y **coste derivados**. Cabe mencionar que los costes vienen referidos al periodo de trabajo de este proyecto, que en conjunto ha sido considerado de 4 meses.

Coste del material

En este apartado se considera el coste de los equipos empleados para llevar a cabo el proyecto. Debido a que los programas informáticos utilizados en este proyecto eran de software libre, únicamente se tendrá en cuenta el precio del equipo y de la FPGA. En la Figura III-1, se muestra el coste de cada uno de los componentes.

Coste de Material				
Recurso	Coste total (€)	Periodo máximo de amortización (meses)	Periodo de amortización del proyecto (meses)	Importe final (€)
PC	850,00	60	4	56,67
PYNQ-Z2	175,00	48	4	14,58
Material adicional	15,00	6	4	10,00
Cursos edx	83,00	-	-	83
Subtotal Material				**164,25 €**

Figura **III**-1: Subtotal de coste del material utilizado

Coste personal

El coste personal hace referencia a la mano de obra durante el tiempo en el que se ha llevado a cabo este proyecto. Para ello se tendrá en cuenta que a parte del Ingeniero de Grado, se ha de añadir al profesor tutor. En este caso se debe de tener cuenta que el Ingeniero de Grado aún no es ingeniero titulado por lo que se le ha establecido una aproximación de unos 8 euros por hora, mientras que al profesor tutor se le han asignado 20 euros por hora.

De esta forma y suponiendo un trabajo semanal de unas 25 horas, durante estos cuatro meses, se obtiene el importe total correspondiente al coste de personal.Además, debido a que el

campo de trabajo de este proyecto, la inteligencia artificial y el análisis de datos, no es uno de los campos más trabajados durante el grado, ha sido necesaria la realización de complementos formativos. Para ello se han llevado a cabo dos cursos de unas 25 horas cada uno aproximadamente, de la plataforma edx, cuyos nombres son:

- Machine Learning (aprendizaje automático) con Python: una introducción práctica. (IBM)
- Aprendizaje automático y ciencia de datos (UPV-Valencia)

Finalmente en la Figura III-2 se recopila toda la información, obteniendo así el coste final correspondiente a la labor humana.

Coste de Personal			
Recurso	**Coste por hora (€/h)**	**Tiempo empleado (h)**	**Importe final (€)**
Ingeniero de Grado	8	450	3600,00
Prof. Asociado Ingeniero	20	80	1600,00
Subtotal Personal			**5.200,00 €**

Figura **III**-2: Subtotal de coste del personal

Costes derivados

En esta sección se añaden los costes surgidos a partir de la realización del proyecto pero que no tienen una relación directa con el mismo. Entre estos costes encontramos, el alquiler, el consumo eléctrico y de agua agrupado en gastos de oficina, el transporte... Estos gastos se recopilan en la Figura III-3.

Costes Derivados			
Recurso	**Coste mensual (€/mes)**	**Período de amortización del proyecto (meses)**	**Importe final (€)**
Alquiler Oficina	225,00	4	900,00
Gastos Oficina (Agua, Luz...)	60,00	4	240,00
Transporte	23,20	4	92,80
Subtotal Derivados			**1.232,80 €**

Figura **III**-3: Subtotal de costes derivados

Presupuesto final

En esta sección se presenta de forma simplificada los gastos de cada una de las fases del proyecto, previamente mencionadas, para obtener el presupuesto final del mismo. Este resumen se observa en la Figura III-4.

Presupuesto Final	
Concepto	Importe (€)
Material	164,25 €
Humano	5.200,00 €
Derivados	1.232,80 €
Total bruto	6.597,05 €
IVA (21%)	1.385,38 €
Final	**7.982,43 €**

Figura **III**-4: Total de costes del proyecto

Este proyecto pues, supone un coste de $7982,43$. **SIETE MIL NOVECIENTOS OCHENTA Y DOS EUROS CON CUARENTA Y TRES CÉNTIMOS**

Impacto medioambiental

Debido a la tremenda importancia del medio ambiente y la necesidad de preservarlo y cuidarlo, esta última sección del presupuesto se reserva para el cálculo del impacto ambiental de este TFG. Para el cálculo del mismo se ha utilizado la página web [38],que estima la huella de carbono producida, es decir la cantidad de CO2 emitido por las actividades derivadas de este trabajo. Para este cálculo se tendrán en cuenta:

- Consumo eléctrico:
  Para calcular el consumo eléctrico se ha utilizado una de las facturas de Iberdrola del local en el que se ha llevado a cabo el proyecto. En esta se estipula un consumo medio de $6,6\ kW$ el día, a tener en cuenta que esta era utilizada por dos personas, con lo que se ha aproximado el consumo a $3,3\ kW$ el día o unos $101,3\ kW$ al mes.

- Transporte:
  Las emisiones dedicadas al transporte, se han establecido considerando dos viajes de ida y vuelta al mes, en tren de corta distancia ($60\ km$). Lo que suponen un total de $240\ km$ al mes.

En la Figura III-5 se recogen las emisiones de cada una de las actividades que se acaban de describir, obteniendo así el total de emisiones producidas por este proyecto.

Emisiones CO2					
Recurso	Consumo al mes (kW)	Distancia recorrida al mes (km)	Emisiones por mes (kg)	Período del proyecto (meses)	Emisiones totales (kg)
Electricidad	102,3	-	41,16	4	164,64
Transporte	-	240	10,16	4	40,64
Emisiones Finales					**205,28**

Figura **III**-5: Total de emisiones de CO_2 atribuidas al desarrollo del proyecto

IV ANEXO DE CÓDIGO

Introducción

El presente Anexo contiene una recopilación de todas las imágenes del código utilizado en el proyecto, tanto en los archivos de jupyter notebook como los obtenidos en el terminal a la hora de programar directamente en la FPGA, con el proposito de ilustrar las diferentes partes del mismo, así como los errores encontrados durante el desarrollo.

En este Anexo se mostrará primero el Código Original, extraído directamente del repositorio del proyecto PYNQ-Classification [13] con la finalidad de facilitar el seguimiento de los cambios realizados durante este proyecto.

Además, como bien se ha mencionado, se creará un apartado de errores en el que aparecerá el código correspondiente a cada uno de los errores presentados en la sección homónima del propio documento. De esta forma se pretende facilitar la detección de errores similares en proyectos relacionados con el presente campo de estudio, acelerando así la solución de los mismos.

Código Original LeNet en JupyterNotebooks

Introduction

This example demonstrates how to convert a network from Caffe's Model Zoo for use with Lasagne. We will be using the Lenet trained for MNIST.

We will create a set of Lasagne layers corresponding to the Caffe model specification (prototxt), then copy the parameters from the caffemodel file into our model.

Converting from Caffe to Lasagne

Download the required files

First we download `cifar10_nin.caffemodel` and `model.prototxt`. The supplied `train_val.prototxt` was modified to replace the data layers with an input specification, and remove the unneeded loss/accuracy layers.

Import Caffe

To load the saved parameters, we'll need to have Caffe's Python bindings installed.

In [1]:
```
import sys
caffe_root = '/home/xilinx/caffe/'  # this file should be run from {caffe_root}/examples (other
wise change this line)
sys.path.insert(0, caffe_root + 'python')
import caffe
```

Load the pretrained Caffe network

In [2]:
```
net_caffe = caffe.Net('lenet.prototxt', 'lenet_iter_10000.caffemodel', caffe.TEST)
```

Import Lasagne

In [3]:
```
import lasagne
from lasagne.layers import InputLayer, DropoutLayer, DenseLayer, NonlinearityLayer
#from lasagne.layers.dnn import Conv2DDNNLayer as ConvLayer
from lasagne.layers import Conv2DLayer as ConvLayer
from lasagne.layers import Pool2DLayer as PoolLayer
from lasagne.nonlinearities import softmax, rectify, linear
import conv_fpga
from conv_fpga import FPGA_LENET
from conv_fpga import FPGAQuickTest
#from conv_fpga import Conv2DLayer as ConvLayer
from conv_fpga import FPGAWeightLoader as FPGALoadW
from lasagne.utils import floatX
```

Create a Lasagne network

Layer names match those in `model.prototxt`

In [4]:
```
net = {}
net['input'] = InputLayer((None, 1, 28, 28))
net['conv1'] = ConvLayer(net['input'], num_filters=20, filter_size=5, nonlinearity=linear)
net['pool1'] = PoolLayer(net['conv1'], pool_size=2, stride=2, mode='max', ignore_border=False)
net['conv2'] = ConvLayer(net['pool1'], num_filters=50, filter_size=5, nonlinearity=linear)
net['pool2'] = PoolLayer(net['conv2'], pool_size=2, stride=2, mode='max', ignore_border=False)
net['ip1'] = DenseLayer(net['pool2'], num_units=500, nonlinearity = rectify)
net['ip2'] = DenseLayer(net['ip1'], num_units=10, nonlinearity = None)
net['prob'] = NonlinearityLayer(net['ip2'], softmax)
```

Copy the parameters from Caffe to Lasagne

In [5]:
```
import numpy as np

layers_caffe = dict(zip(list(net_caffe._layer_names), net_caffe.layers))

for name, layer in net.items():
    try:
        if name=='ip1'or name=='ip2':
            layer.W.set_value(np.transpose(layers_caffe[name].blobs[0].data))
            layer.b.set_value(layers_caffe[name].blobs[1].data)
        else:
            layer.W.set_value(layers_caffe[name].blobs[0].data[:,:,::-1,::-1])
            layer.b.set_value(layers_caffe[name].blobs[1].data)

    except AttributeError:
        continue
```

Copy the parameters from CPU to FPGA OnChip Memory

In [6]:
```
#FPGALoadW(weight, status, IFDim, OFDim, PadDim)
weight = net['conv1'].W.get_value()
FPGALoadW(weight, 1, 28, 24, 0)
weight = net['conv2'].W.get_value()
FPGALoadW(weight, 2, 12, 8, 0)
weight = net['ip1'].W.get_value()
weight = np.transpose(weight)
weight = weight.reshape(500, 50, 4, 4)
FPGALoadW(weight, 3, 4, 1, 0, flip_filters=False)
weight = net['ip2'].W.get_value()
weight = np.transpose(weight)
weight = weight.reshape(10, 500, 1, 1)
FPGALoadW(weight, 4, 1, 1, 0, flip_filters=False)
```

```
weight shape (20, 1, 5, 5)
Loading Started for Layer  1
Elapsed Test Time:  0.002136251999999672
Loading Finished for Layer  1
weight shape (50, 20, 5, 5)
Loading Started for Layer  2
Elapsed Test Time:  0.0007430980000009413
Loading Finished for Layer  2
weight shape (500, 50, 4, 4)
Loading Started for Layer  3
Elapsed Test Time:  0.009178858000002066
Loading Finished for Layer  3
weight shape (10, 500, 1, 1)
Loading Started for Layer  4
Elapsed Test Time:  0.00025717900000188365
Loading Finished for Layer  4
```

Trying it out

Let's see if that worked.

Import numpy and set up plotting

Import time

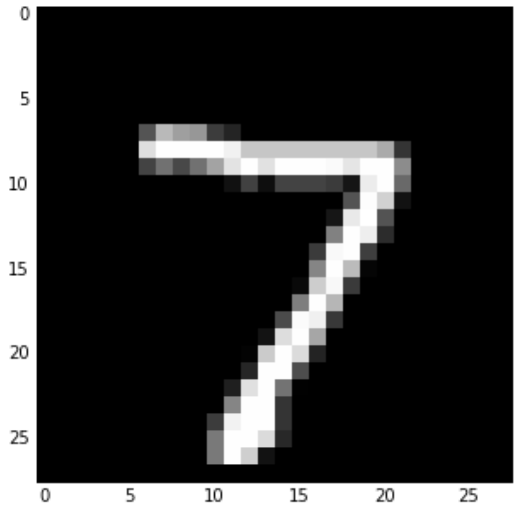
In [7]:
```
import gzip
import _pickle as cPickle
import matplotlib.pyplot as plt
import time

%matplotlib inline
```

Download some test data

Load test mnist handwritting data.

In [8]:
```
data = np.load('test_cpr.npz')
test_data = data['data'].reshape(10000, 1, 28, 28)
test_label = data['label']
##
plt.figure(figsize=(5, 5))
plt.imshow(test_data[0][0], interpolation='nearest', cmap=plt.get_cmap('gray'))
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x2a105d50>
```



FPGA Deployment (Lasagne Layer)

In [9]:
```
FPGA_net = {}
FPGA_net['input'] = InputLayer((None, 1, 28, 28))
FPGA_net['lenet'] = FPGA_LENET(FPGA_net['input'])
FPGA_net['prob'] = NonlinearityLayer(FPGA_net['lenet'], softmax)
```

In [10]:
```
batch_size = 500

%time prob = lasagne.layers.get_output(FPGA_net['prob'], floatX(test_data[0:batch_size]), deter
ministic=True).eval()
FPGA_predicted = np.argmax(prob, 1)
```

```
Elapsed Test Time:  0.9378025709999989
CPU times: user 3.34 s, sys: 1.49 s, total: 4.83 s
Wall time: 13.1 s
```

In [11]:
```
FPGA_accuracy = np.mean(FPGA_predicted == test_label[0][0:batch_size])
#print(FPGA_predicted)
#print(test_label[0][0:600])
print(FPGA_accuracy)
```

```
0.984
```

FPGA Deployment (QuickTest Function)

In [12]:
```
batch_size = 600

OFMDim = 1
OFMCH = 10
%time FPGA_output = FPGAQuickTest(test_data, batch_size, OFMDim, OFMCH)
FPGA_predicted = np.argmax(FPGA_output.reshape(batch_size, -1), 1)
```

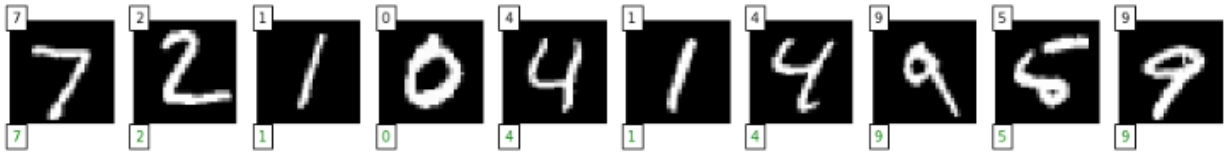
```
Elapsed Test Time:  1.125952114999997
CPU times: user 1.19 s, sys: 50 ms, total: 1.24 s
Wall time: 1.24 s
```

In [13]:
```
FPGA_accuracy = np.mean(FPGA_predicted == test_label[0][0:batch_size])
print(FPGA_accuracy)
```

```
0.983333333333
```

Graph some images and predictions

In [14]:
```
def make_image(X):
    im = np.swapaxes(X.T, 0, 1)
    im = im - im.min()
    im = im * 1.0 / im.max()
    return im

plt.figure(figsize=(16, 5))
for i in range(0, 10):
    plt.subplot(1, 10, i+1)
    plt.imshow(make_image(test_data[i][0]), interpolation='nearest', cmap=plt.get_cmap('gray'))
    true = test_label[0][i]
    pred = FPGA_predicted[i]
    color = 'green' if true == pred else 'red'
    plt.text(0, 0, true, color='black', bbox=dict(facecolor='white', alpha=1))
    plt.text(0, 32, pred, color=color, bbox=dict(facecolor='white', alpha=1))

    plt.axis('off')
```



ARM CPU Deployment

In [15]:
```
batch_size = 600
%time prob = np.array(lasagne.layers.get_output(net['prob'], floatX(test_data[0:batch_size]), d
eterministic=True).eval())
predicted = np.argmax(prob, 1)
```

```
CPU times: user 18.4 s, sys: 1.21 s, total: 19.6 s
Wall time: 1min
```

Check our accuracy

We expect around 90%

In [16]:
```
accuracy = np.mean(predicted == test_label[0][0:batch_size])
# print(predicted)
# print(test_label[0][0:batch_size])
print(accuracy)
```

```
0.985
```

Códigos de Error

```
root@pynq:/home/xilinx# sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libhdf5-serial-dev protobuf-compiler
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package libprotobuf-dev
E: Unable to locate package libleveldb-dev
E: Unable to locate package libsnappy-dev
E: Unable to locate package libhdf5-serial-dev
E: Unable to locate package protobuf-compiler
root@pynq:/home/xilinx# sudo apt-get install --no-install-recommends libboost-all-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package libboost-all-dev
root@pynq:/home/xilinx# sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package libgflags-dev
E: Unable to locate package libgoogle-glog-dev
E: Unable to locate package liblmdb-dev
root@pynq:/home/xilinx#
```

Figura **IV**-1: Código de Error:Unable to locate package (Comando: sudo apt-get install)

```
root@pynq:/home/xilinx/caffe# make all
CXX src/caffe/layer_factory.cpp
In file included from /usr/include/boost/python/detail/prefix.hpp:13:0,
                 from /usr/include/boost/python/args.hpp:8,
                 from /usr/include/boost/python.hpp:11,
                 from src/caffe/layer_factory.cpp:4:
/usr/include/boost/python/detail/wrap_python.hpp:50:11: fatal error: pyconfig.h: No such file or directory
 # include <pyconfig.h>
           ^~~~~~~~~~~~
compilation terminated.
Makefile:591: recipe for target '.build_release/src/caffe/layer_factory.o' failed
make: *** [.build_release/src/caffe/layer_factory.o] Error 1
root@pynq:/home/xilinx/caffe#
```

Figura **IV**-2: Código de Error: pyconfig.h: No such file or directory (Comando: make all)

```
Type "help", "copyright", "credits" or "license" for more information.
>>> import caffe
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/xilinx/caffe/python/caffe/__init__.py", line 1, in <module>
    from .pycaffe import Net, SGDSolver, NesterovSolver, AdaGradSolver, RMSPropSolver, AdaDeltaSolver, AdamSolver, NC
L, Timer
  File "/home/xilinx/caffe/python/caffe/pycaffe.py", line 15, in <module>
    import caffe.io
  File "/home/xilinx/caffe/python/caffe/io.py", line 2, in <module>
    import skimage.io
  File "/usr/lib/python3/dist-packages/skimage/io/__init__.py", line 15, in <module>
    reset_plugins()
  File "/usr/lib/python3/dist-packages/skimage/io/manage_plugins.py", line 93, in reset_plugins
    _load_preferred_plugins()
  File "/usr/lib/python3/dist-packages/skimage/io/manage_plugins.py", line 73, in _load_preferred_plugins
    _set_plugin(p_type, preferred_plugins['all'])
  File "/usr/lib/python3/dist-packages/skimage/io/manage_plugins.py", line 85, in _set_plugin
    use_plugin(plugin, kind=plugin_type)
  File "/usr/lib/python3/dist-packages/skimage/io/manage_plugins.py", line 255, in use_plugin
    _load(name)
  File "/usr/lib/python3/dist-packages/skimage/io/manage_plugins.py", line 299, in _load
    fromlist=[modname])
  File "/usr/lib/python3/dist-packages/skimage/io/_plugins/matplotlib_plugin.py", line 3, in <module>
    import matplotlib.pyplot as plt
  File "/usr/lib/python3/dist-packages/matplotlib/pyplot.py", line 40, in <module>
    from matplotlib.figure import Figure, figaspect
  File "/usr/lib/python3/dist-packages/matplotlib/figure.py", line 39, in <module>
    from matplotlib.axes import Axes, SubplotBase, subplot_class_factory
  File "/usr/lib/python3/dist-packages/matplotlib/axes/__init__.py", line 4, in <module>
    from ._subplots import *
  File "/usr/lib/python3/dist-packages/matplotlib/axes/_subplots.py", line 10, in <module>
    from matplotlib.axes._axes import Axes
  File "/usr/lib/python3/dist-packages/matplotlib/axes/_axes.py", line 23, in <module>
    import matplotlib.dates as _   # <-registers a date unit converter
  File "/usr/lib/python3/dist-packages/matplotlib/dates.py", line 129, in <module>
    from dateutil.rrule import (rrule, MO, TU, WE, TH, FR, SA, SU, YEARLY,
  File "/usr/local/lib/python3.6/dist-packages/dateutil/rrule.py", line 55
    raise ValueError, "Can't create weekday with n == 0"
                    ^

SyntaxError: invalid syntax
>>>
```

Figura **IV**-3: Código de Error: invalid syntax (Comando: import caffe)

```
>>> import im2col_lasagne_cython.pyx
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'im2col_lasagne_cython'
>>>
```

Figura **IV**-4: Código de Error: Module not found (Comando: import im2col_lasagne_cython.pyx)

```
>>> import acc8.pyx
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'acc8'
>>>
```

Figura **IV**-5: Código de Error: Module not found (Comando: import acc8.pyx)

```
root@pynq:/home/xilinx/jupyter_notebooks/PYNQ-Classification/python_notebooks/Theano/Lenet# python3
Python 3.6.5 (default, Apr  1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import acc8
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: /usr/local/lib/python3.6/dist-packages/acc8.cpython-36m-arm-linux-gnueabihf.so: undefined symbol: acc8
>>> exit()
```

Figura **IV**-6: Código de Error: Import error: undefined symbol (Comando: import acc8.pyx)

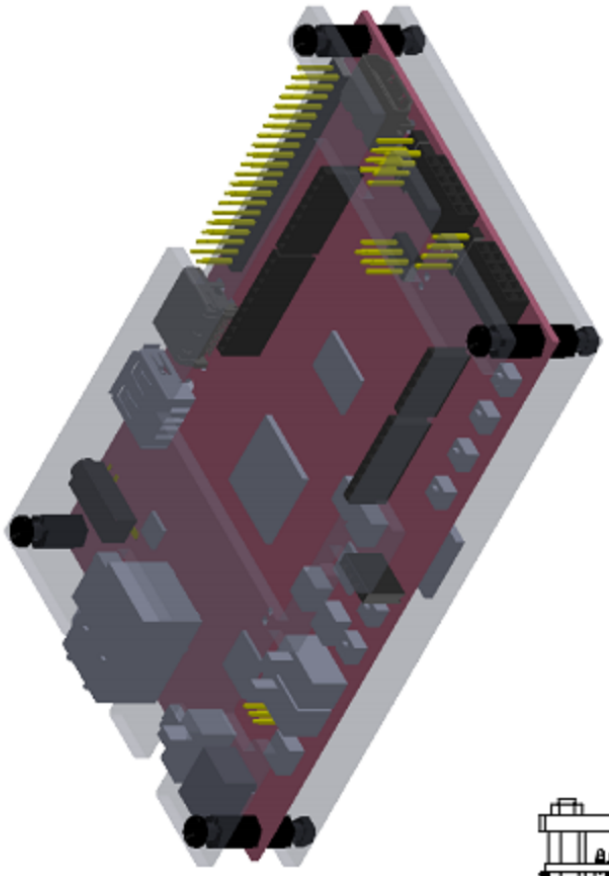
```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-15-adea9fddad99> in <module>()
      5 from lasagne.layers import Pool2DLayer as PoolLayer
      6 from lasagne.nonlinearities import softmax, rectify, linear
----> 7 import conv_fpga
      8 #from conv_fpga import FPGA_LENET
      9 #from conv_fpga import FPGAQuickTest

/home/xilinx/jupyter_notebooks/PYNQ-Classification/python_notebooks/Theano/Lenet/conv_fpga.py in <mod
ule>()
    133 from pynq import Overlay
    134 Overlay('base.bit').download()
--> 135 from pynq.drivers import DMA
    136 import pynq.drivers.dma
    137 Overlay('/home/xilinx/jupyter_notebooks/PYNQ_CNN/Theano/Lenet/Bitstream/lenet.bit').download
()

ModuleNotFoundError: No module named 'pynq.drivers'
```

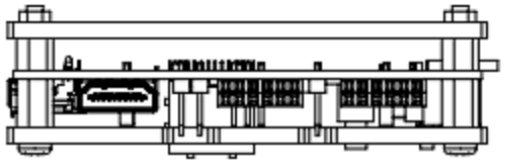
Figura **IV**-7: Código de Error: Module not found (Comando: from pynq.drivers import DMA)

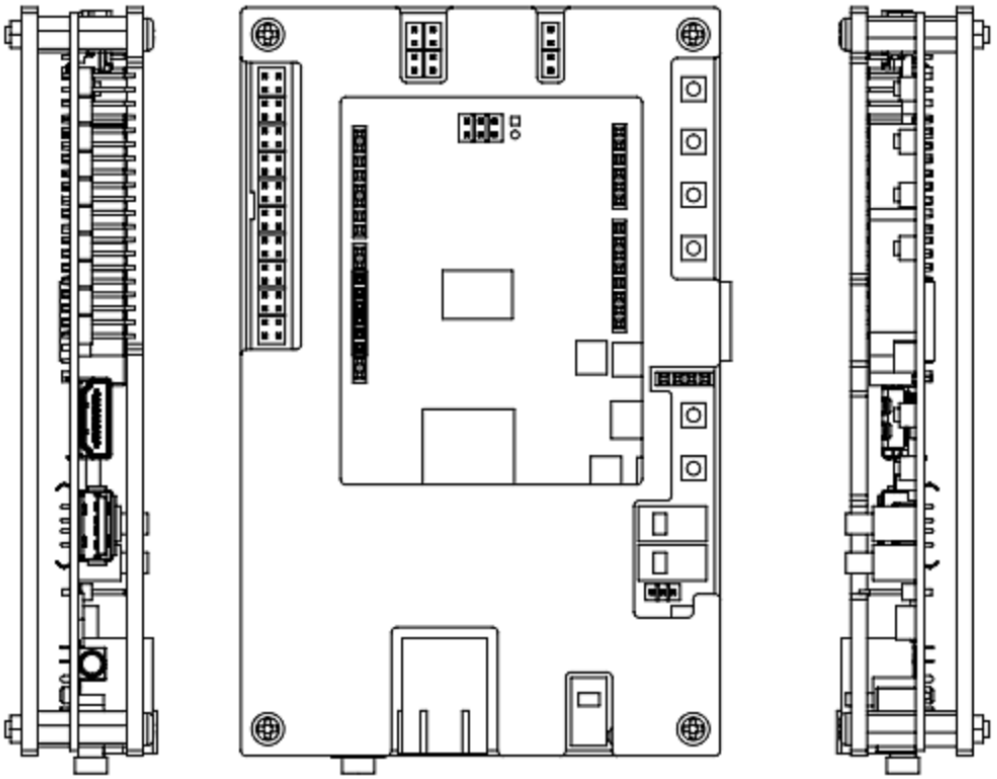
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-6-2a2508b56ba4> in <module>()
      5 from lasagne.layers import Pool2DLayer as PoolLayer
      6 from lasagne.nonlinearities import softmax, rectify, linear
----> 7 import conv_fpga
      8 from conv_fpga import FPGA_LENET
      9 from conv_fpga import FPGAQuickTest

/home/xilinx/jupyter_notebooks/PYNQ-Classification/python_notebooks/Theano/Lenet/conv_fpga.py in <mod
ule>()
    159     print("Unknown plan type: "+repr(plan))
    160
--> 161 hw_switch=StreamingSwitch('axi_switch_0')
    162 def execute_hardware(plan):
    163   dma=[]

/home/xilinx/jupyter_notebooks/PYNQ-Classification/python_notebooks/Theano/Lenet/conv_fpga.py in __in
it__(self, name)
     63 class StreamingSwitch:
     64   def __init__(self,name):
---> 65     base_addr=int(PL.ip_dict["SEG_{0}_Reg".format(name)][0],16)
     66     self.mmio=MMIO(base_addr,256)
     67     self.reset()

KeyError: 'SEG_axi_switch_0_Reg'
```

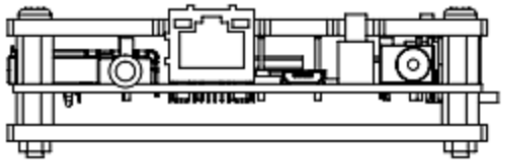
Figura **IV**-8: Código de Error: KeyError *SEG_axi_switch_0_Reg* (Comando: import conv_fpga)

V ANEXO DE PLANOS









UNIT			
SCALE	1:1		
		2018/3/19	
			6/8

Referencias

[1] IBIS. 2008-2015 WILDLIFE STRIKE ANALYSES, ICAO.
Recuperado:https://www.icao.int/safety/IBIS/2008%20-%202015%20Wildlife%20Strike%20Analyses%20(IBIS)%20-%20EN.pdf

[2] Un Airbus de Iberia colisiona en vuelo con un buitre leonado y regresa de urgencia a Barajas, El Confidencial.
Recuperado:https://www.elconfidencial.com/espana/2012-08-31/un-airbus-de-iberia-colisiona-en-vuelo-con-un-buitre-leonado-y-regresa-de-urgencia-a-barajas_220372/

[3] Un buitre, posible causa del accidente de la avioneta de Perales, El País
Recuperado:https://elpais.com/ccaa/2016/03/31/madrid/1459440538_864599.html

[4] La avioneta se estrelló por el impacto de un buitre en un ala, Levante
Recuperado:https://www.levante-emv.com/sucesos/2016/01/19/avioneta-estrello-impacto-buitre-ala/1367536.html

[5] Peligro: buitres volando, el Periódico
Recuperado:https://www.elperiodico.com/es/medio-ambiente/20160821/buitres-causan-accidentes-aereos-espana-5333601

[6] FPGA Acceleration of Convolutional Neural Networks
Recuperado:https://www.nallatech.com/wp-content/uploads/Nalllatech-Whitepaper-FPGA-Accelerated-CNN-003TR.pdf

[7] Bird detection and species classification with time-lapse images around a wind farm: Dataset construction and evaluation, R. Yoshihashi, R. Kawakami,M. Iida, T. Naemura. School of Engineering. The University of Tokyo.
Recuperado: http://bird.nae-lab.org/dataset/yoshihashi-we2017.pdf

[8] Wild Birds in a Wind Farm: Image Dataset for Bird Detection, R. Yoshihashi, R. Kawakami,M. Iida, T. Naemura. The University of Tokyo.
Recuperado: http://bird.nae-lab.org/dataset/

[9] Caffe Deep Learning Framework.
Recuperado: https://caffe.berkeleyvision.org/

[10] Deep Learning Case Study for Automatic Bird Identification, Juha Niemi, Juha T. Tanttu.
Recuperado:https://www.researchgate.net/publication/328587822_Deep_Learning_Case_Study_for_Automatic_Bird_Identification

[11] Automated monitoring for birds in flight: Proof of concept with eagles at a wind power facility, Christopher J.W. McClure, Luke Martinson y Taber D. Allison.
Recuperado: https://www.sciencedirect.com/science/article/pii/S0006320717319407

[12] PYNQ Classification - Python on Zynq FPGA for Neural Networks, ErweiWang, Prof P.Y.K. Cheung, Prof G.A. Constantinides.Imperial College London.
Recuperado: https://github.com/awai54st/PYNQ-Classification/blob/master/PYNQ_CLASSIFICATION.pdf

[13] PYNQ Classification (github repository)
Recuperado:https://github.com/awai54st/PYNQ-Classification

[14] Moving Object Detection Based on Optical Flow Estimation and a Gaussian Mixture Model for Advanced Driver Assistance Systems, Jaechan Cho, Yongchul Jung, Dong-Sun Kim, Seongjoo Lee y Yunho Jung.

[15] Introduction to Motion Estimation with Optical Flow, Nanonets. Recuperado:https://nanonets.com/blog/optical-flow/

[16] ¿Qué es una red neuronal?, MathWorks.
Recuperado: https://es.mathworks.com/discovery/neural-network.html

[17] ¿Qué es una Red Neuronal? Parte 2: La Red, DotCSV.
Recuperado:https://www.youtube.com/watch?v=uwbHOpp9xkc

[18] El modelo de redes neuronales, IBM.
Recuperado:https://www.ibm.com/support/knowledgecenter/es/SS3RA7_sub/modeler_mainhelp_client_ddita/components/neuralnet/neuralnet_model.html

[19] Redes Neuronales Convolucionales, MathWorks.
Recuperado:https://es.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

[20] Gradient-based learning applied to document recognition, LeCun Y, Bottou L, Bengio Y, Haffner P.

[21] Deep resiudal learning for image recognition. Computer vision and pattern recognition, He K, Zhang X, Ren S, Sun J.

[22] Introduction to FPGA Design with Vivado High-Level Synthesis
Recuperado:https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf

[23] Xilinx. Pynq.
Recuperado:https://www.xilinx.com/support/university/boards-portfolio/xup-boards/XUPPYNQ.html

[24] Background Subtraction in Dynamic Scenes, VISAL (Video, Image, and Sound Analysis Lab), Departament of Computer Science, City UNiversity of Hong Kong.
Recuperado: https://visal.cs.cityu.edu.hk/research/dytexbkgnd/

[25] Vehicle Detection and Tracking (github repository)
Recuperado: https://github.com/parilo/carnd-vehicle-detection-and-tracking

[26] Vehicle Detection (github repository)
Recuperado: https://github.com/tatsuyah/vehicle-detection

[27] Cámara Firefly 8s 4K.
Recuperado: https://micamaradeportiva.com/firefly-8s-4k-review-precio-opiniones/

[28] PYNQ: PYTHON PRODUCTIVITY.
Recuperado: http://www.pynq.io/board.html

[29] Editor de texto vi.
Recuperado: https://www.dc.fi.udc.es/ afyanez/info-vi/index.html

[30] Installing Theano, Theano.
Recuperado:http://deeplearning.net/software/theano_versions/0.8.X/install.html

[31] GitHub Issue OpenBlas.
Recuperado:https://github.com/BVLC/caffe/issues/3028

[32] Caffe Installation on Ubuntu 18.04 LTS (Python 3.6)
Recuperado:https://medium.com/@atinesh/caffe-installation-on-ubuntu-18-04-lts-python-3-6-e76375f0d353

[33] PYNQ Classification Modified by Siudya (github repository)
Recuperado:https://github.com/Siudya/PYNQ-Classification

[34] AI-Blog 11
Recuperado: https://www.firstcatch.me/ai-blog-11.html

[35] Video using the Base Overlay, Python productivity for Zynq.
Recuperado: https://pynq.readthedocs.io/en/v1.3/9b_base_overlay_video.html

[36] Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit, Xilinx.
Recuperado:https://www.xilinx.com/products/boards-and-kits/zcu104.html

[37] Firefly 8S – real 4K, 90 or 170 FOV – Review, elProducente.com
Recuperado:http://elproducente.com/firefly-8s-real-4k-action-camera-review-manual/

[38] Calculadora de emisiones de CO2.
Recuperado:https://www.ceroco2.org/calculadoras/