Técnicas de Machine Learning para Conducción

Álvaro De La Cruz Casado
Manuel Oreja Valverde
Alejandro Pascua Piña
Marcos Robles Palencia

Facultad de Informática
Departamento de Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Trabajo Fin de Grado en Ingeniería Informática
Curso 2018-2019

Directores
Carlos García Sánchez
Guillermo Botella Juan

Agradecimientos

A nuestros directores, Dr. Carlos García Sánchez y Dr. Guillermo Botella Juan, por dejarnos la libertad de elegir por nosotros mismos las directrices el proyecto que queríamos realizar,y depositar la confianza necesaria en nosotros para lograrlo, así como su ayuda y dedicación durante estos meses.

ÍNDICE

Agradecimientos 3

ÍNDICE 4

1.- Introducción 6

1.- Introduction 7

2.- Lista de palabras clave 8

3.- Antecedentes 9

3.1.- Inteligencia Artificial 9

3.2.- Deep Learning 11

3.3.- Redes Neuronales 13

3.4.- Conducción autónoma 19

4.- Objetivos y plan de trabajo 21

4.1.- Familiarización con redes neuronales 24

4.2.- Redes neuronales de clasificación y detección de objetos 26

4.2.1.- AlexNet 26

4.2.2.- VGG16 28

4.2.3.- GoogleNet 28

4.2.4.- Inception-v3 29

4.2.5.- ResNet 30

4.2.6.- ResNeXt 31

4.2.7.- RCNN 31

4.2.8.- YOLO 32

4.2.9.- Entrenamiento e inferencia con redes neuronales para clasificación de objetos 32

4.3.- KITTI 36

4.4.- Redes de predicción de disparidad 38

4.4.1. - SegStereo y CRL 40

4.4.2.- PSMNet 43

4.4.3.- MADNet 56

4.5.- Portabilidad y optimización del tiempo de inferencia 62

4.5.1- PSMNET 64

4.5.2- MADNet 65

4.5.3- SegStereo y CRL 66

4.5.4- Redes pre-entrenadas de clasificación de objetos 67

4.5.5- Red de clasificación de objetos con fine-tune 71

5.- Conclusiones y trabajo futuro 73

5.- Conclusions and future work **75**

6.- Trabajo realizado por cada integrante del grupo **77**
Álvaro de la Cruz Casado 77
Manuel Oreja Valverde 79
Alejandro Pascua Piña 81
Marcos Robles Palencia 83

7.- Anexos I Tecnologías utilizadas **85**
7.1.- Python 85
7.2.- Caffe 86
7.3.- Keras 87
7.4.- Pytorch 88
7.5.- TensorFlow 89
7.6.- TensorRT 90
7.7.- CUDA 92
7.8.- OpenCL 94

8.- Bibliografía **95**

1.- Introducción

Este trabajo tiene como objetivo principal trabajar con un tema tan de moda como la inteligencia artificial aplicada a la conducción autónoma. Este campo es bastante amplio y existen multitud de problemas que se pueden resolver a través de la inteligencia artificial. En nuestro caso nos hemos centrado concretamente en el problema del cálculo de la profundidad de los elementos existentes en un campo visual. Para resolver dicho problema existen multitud de arquitecturas de redes neuronales convolucionales, todas ellas basadas en el procesamiento de un par de imágenes en estéreo. De este procesamiento se obtiene un mapa de disparidad que se puede traducir en una imagen que representa la distancia existente del coche a cada uno de los píxeles que la representa.

Además, hemos tenido como objetivo utilizar OpenVINO, una herramienta que acelera la velocidad con la que se puede realizar inferencia sobre una red neuronal. Esto implica que todos los problemas que requieran de inteligencia artificial puedan tener una respuesta lo más rápida posible, uno de los puntos más críticos en este campo. Por otro lado, utilizar esta tecnología junto a elementos de cómputo vectorial de bajo consumo energético, como puede ser Movidius, puede ser una buena combinación para solventar el problema del consumo de energía por los elementos de cómputo hardware a bordo del vehículo.

Teniendo en cuenta estos objetivos, nuestro trabajo primeramente trata sobre el estado del arte en cuanto a inteligencia artificial y sus subramas, conducción autónoma y tecnologías software que se utilizan para afrontar estos temas. Posteriormente presentamos el trabajo previo con redes neuronales de clasificación como primer acercamiento a nuestro objetivo principal. A continuación se expone el trabajo realizado con redes neuronales para cálculo de disparidad, sus resultados y uso con OpenVINO. Finalmente se realiza una conclusión acerca de los resultados obtenidos.

1.- Introduction

The main objective of this work is to work with a topic as fashionable as artificial intelligence applied to autonomous driving. This field is quite broad and there are many problems that can be solved through artificial intelligence. In our case, we have focused specifically on the problem of calculating the depth of the existing elements in a visual field. To solve this problem there are a multitude of convolutional neural network architectures, all based on the processing of a pair of images in stereo. From this processing a disparity map is obtained that can be translated into an image that represents the existing distance of the car to each of the pixels that represents it.

In addition, we have aimed to use OpenVINO, a tool that accelerates the speed with which inference can be made over a neural network. This implies that all the problems that require artificial intelligence can have an answer as quickly as possible, one of the most critical points in this field. On the other hand, using this technology together with elements of vector computing of low energy consumption, such as Movidius, can be a good combination to solve the problem of energy consumption by the elements of hardware computation on board the vehicle.

In consideration of these objectives, our work will first talk about the state of the art in terms of artificial intelligence and its sub-branches, autonomous driving and software technologies that are used to face these issues. Later we present the previous work with neural networks of classification as the first approach to our main objective. Next, the work done with neural networks for disparity calculation, its results and use with OpenVINO is exposed and finally a final conclusion about the obtained results is made.

2.- Lista de palabras clave

- Aprendizaje Profundo - Deep Learning
- Red Neuronal - Neural Network
- KITTI (Karlsruhe Institute of Technology and Toyota Technological Institute)
- Conducción Autónoma - Autonomous Driving
- Disparidad - Disparity
- Inferencia - Inference
- OpenVINO - OpenVINO
- Redes Convolucionales - Convolutional Neural Networks
- Clasificación - Classification
- Conjunto de Datos de Entrenamiento - Training Dataset

3.- Antecedentes

3.1.- Inteligencia Artificial

La Inteligencia Artificial (IA en español, AI en Inglés de sus siglas Artificial Intelligence) es uno de los grandes avances que tenemos actualmente. Citando textualmente a la Wikipedia, *"el término inteligencia artificial se aplica cuando una máquina imita las funciones «cognitivas» que los humanos asocian con otras mentes humanas, como por ejemplo: «aprender» y «resolver problemas»" (https://es.wikipedia.org/wiki/Inteligencia_artificial)*. Sin embargo, este concepto no es algo que se haya inventado tan recientemente como puede parecer. En 1956 ya se estableció formalmente el término por medio de John McCarthy, Marvin Minsky y Claude Shannon, basándose en las características que debía tener una máquina para ser considerada "inteligente" según el test de Turing.
Anteriormente ya se había discutido acerca del concepto, aunque nunca se había establecido como tal. Aristóteles había descrito el conjunto de reglas que describen una parte del funcionamiento de la mente para obtener conclusiones racionales; Ctesibio de Alejandría construyó un regulador de flujo de agua autocontrolado en el 250 A.C.; Ramon Llull expuso en su libro Ars Magna que el razonamiento podía llegar a ser realizado de manera artificial; Alan Turing diseñó la máquina que lleva su nombre que demostraba la viabilidad de un dispositivo físico para implementar cualquier cómputo formalmente definido; en 1943 Warren McCulloch y Walter Pitts elaboraron un modelo de neuronas artificiales, considerado el primero del campo; y Hebert Simon, Allen Newell y J.C. Shaw desarrollaron en 1955 el primer lenguaje de programación orientado a resolver problemas, el IPL-11, así como el LogicTheorist, que era capaz de demostrar teoremas matemáticos.
Posteriormente a 1956, Hebert Simon y Allen Newell continuaron desarrollando el General Problem Solver (GPS) orientado también a la resolución de problemas; John McCarthy desarrolló el LISP, el primer lenguaje para procesamiento simbólico; entre finales de los 50 y principios de los 60 Robert K. Lindsay desarrolló Sad Sam, que interpretaba oraciones en inglés y sacaba conclusiones; en 1963 Quillian desarrolló las redes semánticas; a mediados de los 60 aparecen los "sistemas expertos", predictores de una probabilidad bajo un set de condiciones; en 1973 se crea PROLOG, un lenguaje de programación muy usado en IA; IBM desarrolló una supercomputadora que era capaz de jugar al ajedrez, y fue capaz de derrotar al vigente campeón del mundo de ajedrez en 1997; en 2009 había ya desarrollándose sistemas inteligentes terapéuticos para detectar emociones que les permitieran interactuar con niños autistas; y en 2014 un ordenador fue capaz de pasar el Test de Turing, haciendo creer a un interrogador que era una persona quien respondía a sus preguntas.

A medida que ha ido pasando el tiempo, la definición que se tenía puede haber ido cambiando, pero actualmente podemos establecer una diferenciación entre las distintas IA:
Máquinas Reactivas: Perciben el mundo y actúan sobre lo que ven. No acumulan recuerdos ni los usan para tomar decisiones. Un ejemplo de esto es Deep Blue, que jugaba al ajedrez

y era capaz de identificar cada ficha y qué movimientos podía hacer, para poder realizar predicciones sobre los mejores movimientos a realizar.
Máquinas con Memoria Limitada: Tienen una serie de recuerdos a los que añaden detalles del momento presente, como los coches autónomos, que almacenan datos del aspecto de autovías o semáforos, pero añaden los peatones o coches de alrededor de ese momento.
Máquinas con Teoría de la Mente: Son capaces de entender las ideas del resto así como tener ideas propias. De esta forma, son capaces de tener una interacción social.
Máquinas con Conciencia Propia: Es algo a lo que se pretende aspirar. Son máquinas que serían capaces de verse a sí mismas con perspectiva en su entorno, internamente y siendo capaces de predecir comportamientos y sentimientos ajenos. Serían una extensión en cierto modo de las máquinas con Teoría de la Mente.

3.2.- Deep Learning

El Deep Learning es el compendio de algoritmos de aprendizaje automático que intenta transformar abstracciones de alto nivel en datos mediante arquitecturas compuestas de transformaciones no lineales múltiples. Es parte del Machine Learning. En palabras generales, enseña a los ordenadores a aprender mediante ejemplos, algo natural en los seres humanos. Es un concepto clave presente por ejemplo en los vehículos sin conductor, permitiéndoles distinguir entre las distintas señales existentes, o entre un peatón y una farola por ejemplo. Mediante el Deep Learning, un modelo informático aprende a clasificar imágenes, texto o sonido por ejemplo. Estos modelos pueden obtener una precisión tan exacta que se han dado casos de que superan el rendimiento humano. Se entrenan con una serie de datos etiquetados denominados Datasets, y arquitecturas de redes neuronales con muchas capas, de las cuales hablaremos más adelante.

Aunque no existe una única definición de Deep Learning, las distintas publicaciones que hablan del mismo se centran en distintas características como:

- Usar una serie de capas con unidades de procesamiento no lineal para extraer y transformar variables, cada una usando la salida de la capa anterior como entrada. Los algoritmos empleados pueden emplear o bien aprendizaje supervisado o aprendizaje no supervisado. El aprendizaje supervisado es una forma de deducir una respuesta en base a datos de entrenamiento. Estos datos de entrenamiento tienen una serie de datos de entrada, y una serie de datos de salida deseados. La finalidad del aprendizaje supervisado es crear una función que pueda predecir el valor correspondiente a cualquier entrada válida tras haber sido entrenada. Esto se consigue generalizando a partir de los datos que se le proporcionan para poder trabajar sobre situaciones no vistas previamente. El aprendizaje no supervisado se ajusta a las observaciones. A priori no existe un conocimiento. Los datos de entrada son tratados como un conjunto de variables aleatorias que construyen un modelo de densidad para el conjunto de datos.
- Estar basado en aprendizaje de múltiples niveles de características o representaciones de datos. De estas características, las de más alto nivel se derivan de las de niveles menores para formar así una representación jerárquica.
- Aprender múltiples niveles de representación correspondientes a niveles de abstracción distintos. De esta forma se conforma una jerarquía de conceptos.

Aunque el inicio de las teorías sobre el Deep Learning se desarrollaron en la década de los 80, no ha sido hasta hace poco cuando ha empezado a resultar útil realmente. Hay dos razones principales para esto:
Requiere de grandes cantidades de datos etiquetados (datasets). Como ejemplo podemos hablar de los vehículos sin conductor, para los cuales se necesitan millones de imágenes de entrenamiento y miles y miles de horas de vídeo analizado.
Requiere también de una gran potencia de cálculo. Las GPU actuales tienen paralelización en su arquitectura, lo que contribuye enormemente en mejorar la eficiencia del Deep

Learning. Si se combina con clusters o incluso con la realización de cálculos en la nube, resulta en una reducción muy significativa del tiempo de entrenamiento de una red a horas, o incluso menos.

El término “Deep” (profundo) de Deep Learning hace referencia al número de capas ocultas en la red neuronal. Las tradicionales contienen solamente dos o tres capas, mientras que estas “profundas” pueden tener hasta 150.

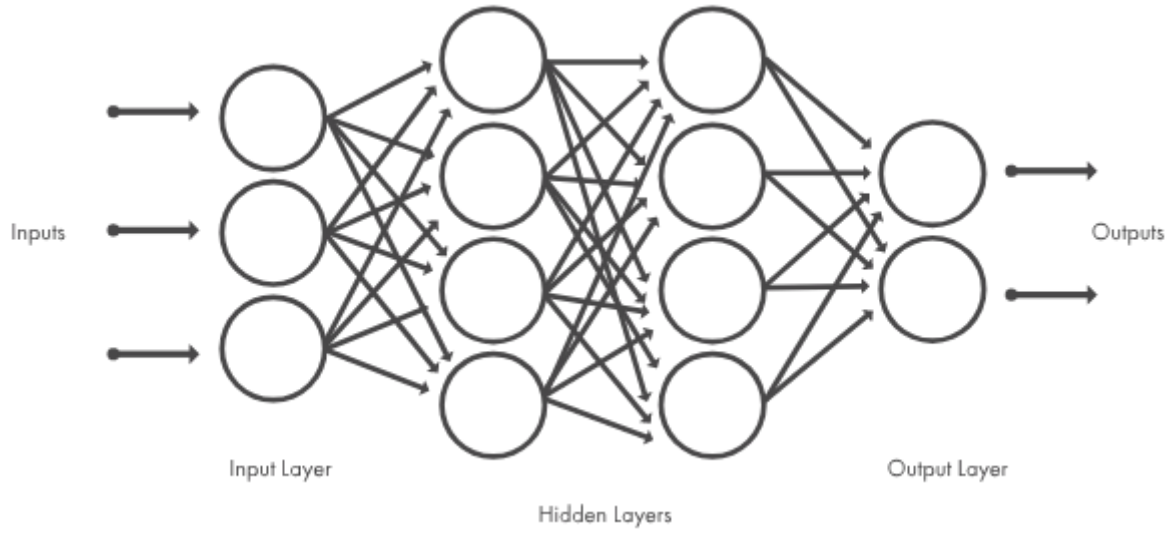


Imagen de una red neuronal[1]

Uno de los tipos más populares son las redes neuronales convolucionales, de las cuales hablaremos más adelante.

Hablando de la diferencia entre Machine Learning y Deep Learning, aparte de que el segundo es un subgrupo del primero, es que en el Machine Learning, el flujo de trabajo comienza con la extracción manual de las características relevantes de las imágenes, mientras que en el Deep Learning, estas características se extraen directamente de las imágenes. Además, el Deep Learning es capaz de trabajar con datos sin procesar, aprendiendo a realizar una clasificación automáticamente.
Otra de las diferencias, que a la vez es también una ventaja, es que, por regla general, mejoran a medida que aumenta el tamaño de los datos.

[1] *https://es.mathworks.com/discovery/neural-network.html*

3.3.- Redes Neuronales

Las redes neuronales son entornos de trabajo en los que conviven diferentes algoritmos de machine learning que trabajan conjuntamente para para procesar datos complejos. Las redes neuronales aprenden a realizar tareas mediante los ejemplos, y una vez entrenadas, intentar aplicar lo aprendido mediante estos ejemplos a datos nuevos. Las redes están formadas por neuronas, cada neurona tiene una o varias salidas, una o varias entradas, sobre las que realiza operaciones, y un peso, que multiplica los valores obtenidos antes de ser enviados a las salidas. Cuando se dice que la red neuronal aprende, es que van ajustando los pesos de las neuronas para satisfacer todos los ejemplos que se le van dando.

La primera neurona creada fue el perceptrón, su funcionamiento era muy simple, como se ha explicado anteriormente, tenía varias entradas, las multiplicaba por unos pesos (que había que ir ajustando de forma manual), sumaba esos valores, y si el total de la suma era superior a un determinado valor, el umbral, la neurona se activaba y la salida era 1, si no llegaba al umbral, la salida era 0. Gracias a esto se pudieron automatizar funciones lógicas “sencillas”. Un perceptrón, sólo es capaz de producir una salida, así que si se quieren evaluar varias características sobre unas mismas entradas, basta con conectar todas las entradas a diferentes neuronas, una por cada característica que se pretenda averiguar, por ejemplo, si queremos averiguar si la entrada es un círculo o un cuadrado, se necesita una neurona que te diga si es un cuadrado o no, y otra que te diga si es un círculo o no.

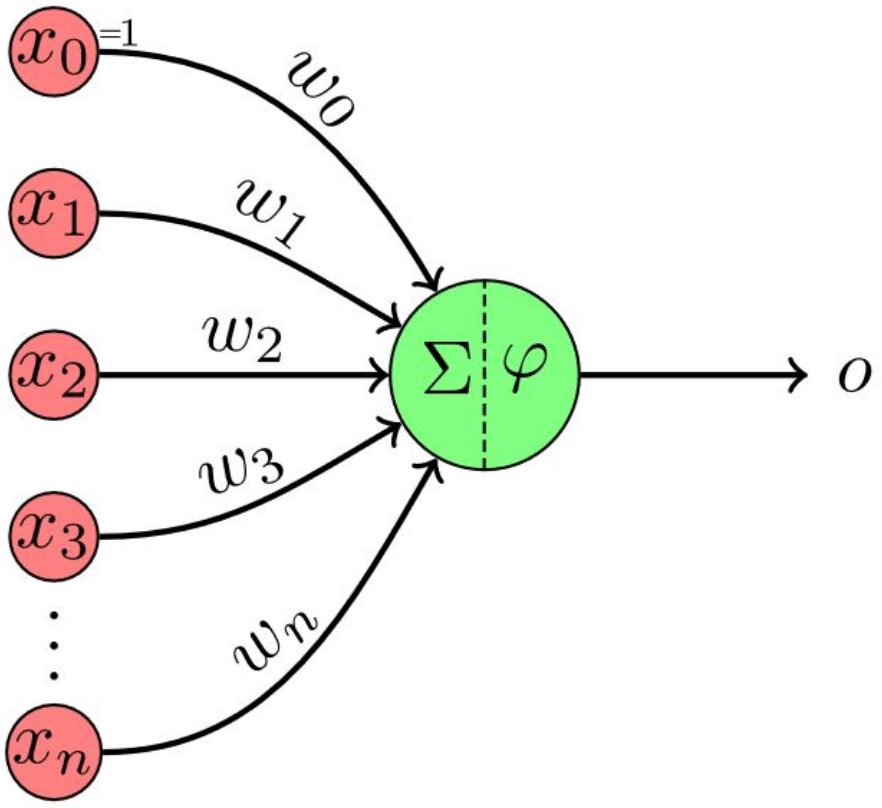


Imagen de un perceptrón[2]

El siguiente paso fue el perceptrón multicapa, que consta de 3 tipos de capas, una de entrada, que recibe las entradas y tiene sus salidas conectadas a las entradas de la siguiente capa, una o más capas ocultas que tienen tanto la entrada como la salida conectadas a otras capas, y una última de salida, que determina el resultado, ésta última capa tendrá tantas neuronas como clasificaciones se quieran hacer. La importancia de las capas ocultas, es que permiten obtener características más complejas y precisas que si dispusieramos de una sola capa.

[2] *https://commons.wikimedia.org/wiki/File:Perceptron-unit.svg*

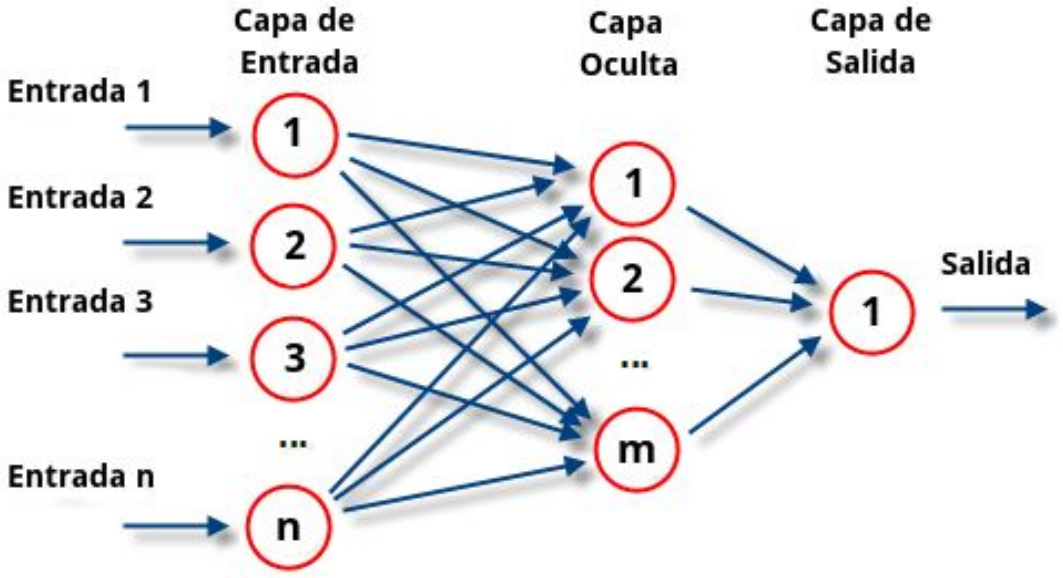
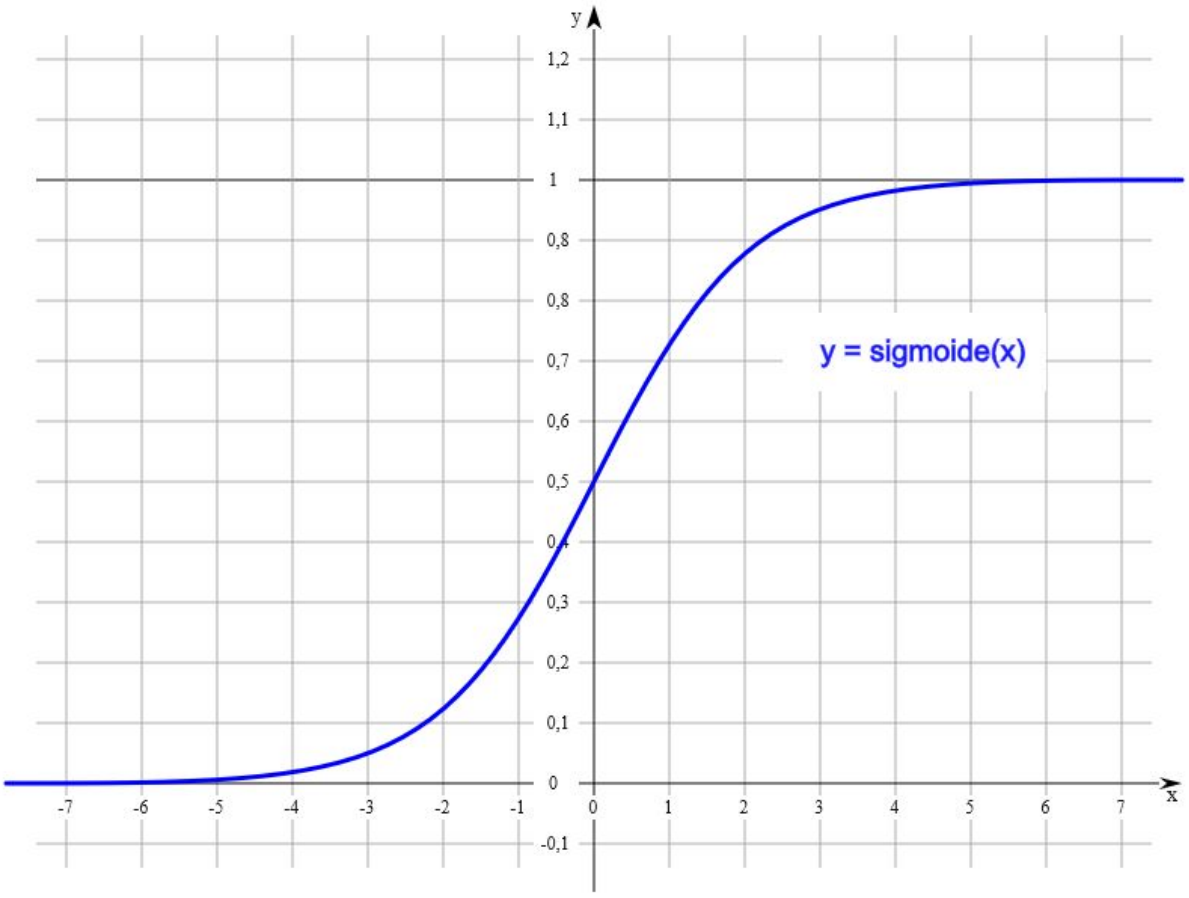


Imagen de un perceptrón multicapa[3]

Aun con los perceptrones multicapa, era necesario ir ajustando los pesos de cada neurona a mano, lo cual era bastante complicado. Para poder conseguir que los pesos se fueran ajustando automáticamente y la red aprendiera por sí misma, fue necesario el empleo del algoritmo de propagación hacia atrás. En él, cuando se produce un fallo al entrenar (no se ha detectado como persona a una imagen de una persona), te basas en las salidas obtenidas y se realizan cambios en los pesos de las neuronas de esta capa para que se ajuste a la salida esperada, con estos nuevos pesos, se ajustan los pesos de la capa anterior, y así sucesivamente.

Aun con el algoritmo de propagación hacia atrás, resultaba muy costoso ajustar los pesos de las neuronas, pues hay que recordar, que los perceptrones tienen salidas binarias y se activan a partir de un determinado umbral, por lo que los posibles valores de los pesos para que se obtenga la salida deseada, son muy numerosos. Esto se facilitó en gran medida gracias a un tipo de neurona, las neuronas sigmoide, que en vez de producir salidas binarias, producían salidas reales en el intervalo [0, 1], siendo menos complejo el ir ajustando los pesos.

[3] https://es.wikipedia.org/wiki/Archivo:RedNeuronalArtificial.png



Gráfica de un sigmoide[4]

Las redes neuronales convolucionales son un tipo de redes neuronales empleadas principalmente en la visión por computador. Se distinguen del resto por que son capaces de buscar características comunes en conjuntos pequeños de entradas, lo que permite entre otras cosas, reducir el número de neuronas necesarias.

En las redes neuronales convolucionales hay distintas capas:

- Capa de convolución: Se encarga de extraer características de la imagen, respeta la relación espacial entre los píxeles. Para ello se aplica un filtro sobre un conjunto de píxeles y se extraen las características.

[4] *https://es.wikipedia.org/wiki/Archivo:Funci%C3%B3n_sigmoide_01.svg*

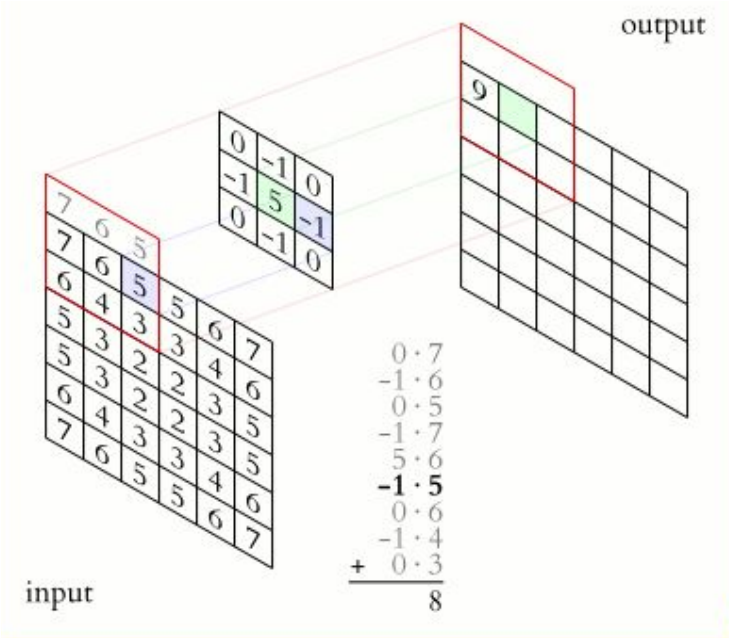


Imagen capa de convolución[5]

- Rectificador (ReLU): Esta capa, que se suele colocar después de las capas de convoluciones, sirve para rectificar algunos valores, se suele emplear la función f(x) = max (0, x), que sustituye los valores negativos por cero. Con ello se consigue mejorar el funcionamiento de la red. Aunque ésta es la función ReLU más empleada por ser la más rápida, se pueden emplear otras funciones como la tangente hiperbólica o la sigmoide.

- Capa de reducción: En este paso se reduce el número de “píxeles”. Se puede hacer reducción de varias maneras, como el máximo, la media, suma… Siendo el máximo el más empleado, ya que ha demostrado ser el que mejor rendimiento tiene. La reducción empleando el máximo consiste en dividir la imagen en secciones más pequeñas, y quedarse con el elemento máximo (se haría de igual modo si se emplean los métodos de la media o suma).

[5] https://commons.wikimedia.org/wiki/File:3D_Convolution_Animation.gif

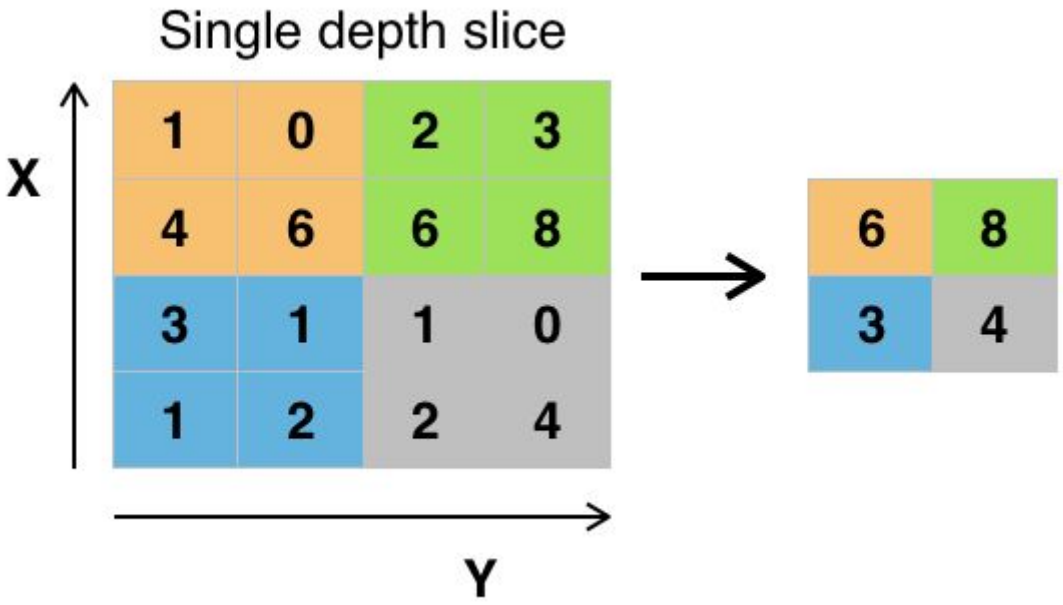


Imagen capa de reducción empleando el máximo[6]

- Capa de clasificación: En este paso se emplea una completamente conectada, es decir todas las salidas de la capa anterior se conectan con todas las neuronas de esta capa. En esta capa habrá una neurona por cada posible clase que pueda identificar la red.

[6] *https://commons.wikimedia.org/wiki/File:Max_pooling.png*

3.4.- Conducción autónoma

La conducción autónoma es un tema muy en auge actualmente. Es difícil no haber oído hablar de coches autónomos, aquellos que son capaces de conducirse sin interacción humana. Para conseguir esto, hacen falta diferentes sensores y herramientas capaces de procesar toda esa información.

Cabe destacar que los vehículos a control remoto, no se consideran autónomos, ya que necesitan de la interacción de un humano (aunque este no se encuentre dentro del vehículo) para realizar cualquier acción.

Es importante saber que hay diferentes niveles de autonomía y qué caracteriza a cada uno, ya que cuando piensas en un coche autónomo, seguramente le venga a la cabeza uno de nivel 4 o 5, pero hay más variedad.

Según la SAE (Sociedad de Ingenieros de Automoción), una sociedad que se encarga de crear estándares para todo tipo de vehículos, pueden existir 6 niveles de autonomía en los vehículos.

- Nivel 0, Sin automatización: El conductor se encarga de manejar el vehículo en todo momento, aunque haya sistemas de alerta.

- Nivel 1, Asistencia al conductor: El conductor maneja el vehículo, pero hay sistemas que pueden controlar la dirección o la velocidad en algunas situaciones (Por ejemplo un sistema que mantiene la velocidad del vehículo a 120 km/h).

- Nivel 2, Automatización parcial: El conductor maneja el vehículo, pero hay sistemas que pueden controlar la dirección y la velocidad en algunas situaciones (Por ejemplo un coche capaz de aparcar solo).

- Nivel 3, Automatización condicional: El vehículo es capaz de conducirse automáticamente en ciertas condiciones, pero el conductor ha de estar alerta por si las condiciones cambian y ha de tomar el control (Por ejemplo el vehículo podrá conducirse solo si se circula por una autopista con poco tráfico, pero el conductor tendrá que tomar el control si está circulando por una calle en obras).

- Nivel 4, Alta automatización: El vehículo es capaz de conducirse autónomamente en prácticamente cualquier situación, siendo capaz de detectar lo que le rodea, como peatones, y actuar en consecuencia, aunque puede necesitar la intervención del conductor, es capaz de continuar incluso si el conductor no reacciona.

- Nivel 5, Automatización completa: El vehículo es capaz de ser conducido automáticamente en todos los aspectos como si lo hiciera un conductor, y no se requerirá su intervención en ningún momento.

Podría decirse que el precursor de los coches autónomos tuvo lugar sobre la década de 1920, cuando en Estados Unidos, la empresa Houdina Radio Control hizo circular un vehículo sin conductor. Aunque esto todavía no se considera conducción autónoma, ya que el vehículo estaba controlado por control remoto mediante radio por el coche que iba detrás, despertó aún más el interés por los vehículos autonomos.

En 1939, Norman Bel Geddes presentó en la Exposición Universal de Nueva York una maqueta con su idea sobre cómo sería una ciudad en el futuro, en la que había multitud de coches eléctricos que circulaban de forma autónoma por las carreteras. A pesar de ser solo una ida, hizo que muchas empresas se interesaran e invirtieran en el avance de la conducción autónoma.

Ya en 1986, Ernst Dickmanns, un ingeniero aeroespacial y profesor universitario en Munich, que había dedicado los últimos años a intentar conseguir que los vehículos pudieran “ver”, diseñó y construyó el primer vehículo autónomo. Se trataba de una furgoneta Mercedes a la que había instalado cámaras, sensores y ordenadores para realizar dicha tarea. Tan solo un año más tarde, y tras las exitosas pruebas, consiguió el permiso para que su furgoneta autónoma circulara por una carretera vacía en Alemania.

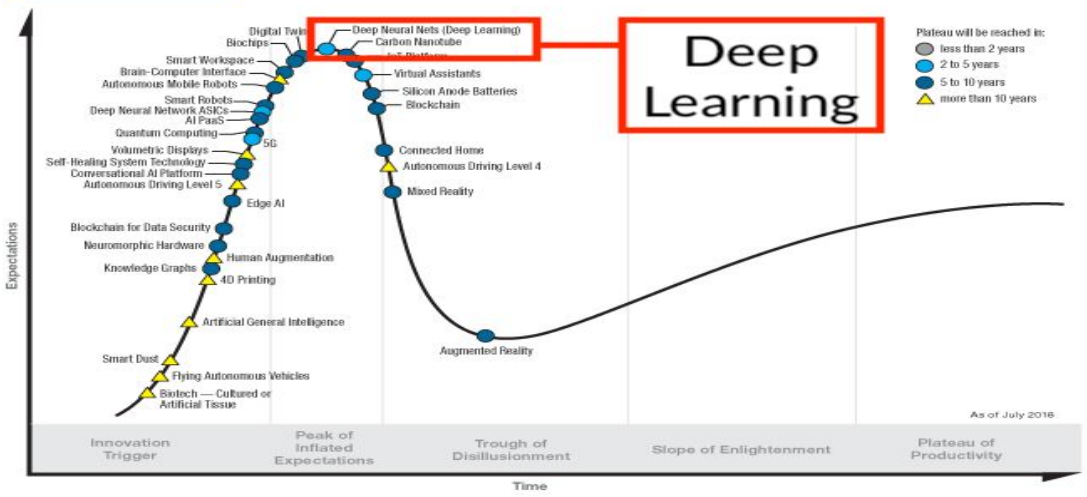
Ocho años más tarde, en 1994 y gracias a los avances de Ernst en este campo, consiguió el permiso de las autoridades de Francia para realizar una demostración de sus avances, esta vez realizando las modificaciones sobre un Mercedes 500 SEL, el cual círculo más de 1000 kilómetros por una carretera de circunvalación gala llegando a alcanzar los 130 km/h, consiguió que el vehículo circulara por una carretera transitada e incluso adelante a otros coches.

Desde entonces, multitud de empresas han avanzado en este campo y han probado su mejoras, sin embargo, aún queda mucho que avanzar y mucho por hacer, tanto a nivel técnico, para que los vehículos sean capaces de realizar un mayor número de tareas o realicen de manera más refinada aquellas que ya hacen, como a nivel legislativo.

En 2018 tuvo lugar el primer atropello mortal producido por un coche autónomo, cuando una persona cruzaba por mitad de una carretera de noche (haciéndolo de manera inadecuada, no por un paso de cebra). Esto no solo hizo que se retiraran permisos para pruebas de vehículos autónomos en diversos lugares, sino que reavivó numerosas cuestiones legales que aún no están controladas, como cuál es el responsable en caso de atropello, ¿es el ocupante del vehículo?, ¿lo es la empresa que lo fabricó?, o ¿cómo debería reaccionar el vehículo si ha de tomar una opción que pone en riesgo vidas humanas?, si por ejemplo circula en sentido contrario otro vehículo con 4 pasajeros y la única forma de evitarlo es invadir el carril contiguo en el que hay un vehículo con 1 solo pasajero, ¿debe el coche autónomo buscar evitar el mayor número de muertes?, ¿debe chocar contra el vehículo que circula en sentido contrario (pues es el que ha cometido la infracción, a diferencia del vehículo de carril contiguo que circula de manera adecuada)?, o ¿debe tomar la decisión en la que más probabilidades tenga de sobrevivir su ocupante?.

4.- Objetivos y plan de trabajo

La conducción autónoma y la inteligencia artificial son unos de los mayores hot topics de los últimos años. Poco a poco están pasando a ser las tecnologías del presente, y lo que no nos cabe duda es que serán las tecnologías del futuro. Como se puede ver en la curva de Gartner, actualmente está en el máximo de las expectativas. Por ello nos ha parecido muy interesante orientar nuestro trabajo sobre estos dos temas al mismo tiempo, teniendo en cuenta que la conducción autónoma no se podría entender sin inteligencia artificial y los grandes avances realizados el la última década.



Curva de de Gartner[7]

La inteligencia artificial aplicada a la conducción autónoma se utiliza para resolver múltiples problemas, como por ejemplo, la detección de peatones. En nuestro caso concreto, nos hemos centrado en la clasificación de objetos, necesaria para que el vehículo sea capaz de asociar lo que percibe con objetos concretos, y en el problema del cálculo de la profundidad de los elementos que aparecen en las imágenes capturadas por un par de cámaras estéreo. Esto nos permite conocer la distancia a la que se encuentran del coche los objetos captados por las cámaras, información necesaria para poder realizar tomas de decisiones por parte del vehículo. Para resolver este problema existen numerosas redes neuronales que a partir de un par de imágenes obtenidas por un par de cámaras en estéreo predicen la disparidad, que es la distancia existente entre un elemento que aparece en la imágen izquierda respecto la ubicación en la que se encuentra en la imágen de la derecha, o viceversa. De esta distancia podemos calcular la profundidad y así obtener la distancia a la que se encuentra del coche. Por tanto nuestro primer objetivo es familiarizarnos con las redes

7 https://medium.com/machine-learning-in-practice/deep-learnings-permanent-peak-on-gartner-s-hype-cycle-96157a1736e

neuronales de clasificación de objetos y las redes neuronales de disparidad de imágenes en estéreo, como base para poder desarrollar el resto del trabajo.

Se plantean 2 líneas de trabajo:

KITTI es la plataforma referencia utilizada por los desarrolladores de inteligencia artificial aplicada a la conducción autónoma. En ella podemos encontrar numerosas redes neuronales que resuelven los diversos problemas anteriormente comentados, entre ellos la predicción de la disparidad. Además de servir de repositorio, esta plataforma sirve como benchmark, proporcionando diversos rankings divididos en diferentes categorías, siendo una de ellas la de disparidad en estéreo, en el que las mejores redes se ordenan conforme al error que realizan al calcular la disparidad. Se proporcionan las herramientas necesarias para realizar su benchmark y aparecer en el ranking con una posición según los resultados obtenidos en él. Teniendo en cuenta esto, uno de los objetivos es estudiar y reproducir el trabajo realizado sobre las diferentes redes neuronales para predicción de disparidad más vanguardistas, las cuales están en esta plataforma. Incluyendo el entrenamiento sobre estas y la ejecución del benchmark del KITTI.

Ya que en los vehículos autónomos es necesario clasificar los objetos que se detectan para poder tomar decisiones en consecuencia, nos interesa ver cómo funciona esta parte, por lo que vamos a reproducir el funcionamiento de alguna red neuronal de clasificación de objetos y comparar entre sí tanto el tiempo empleado como los resultados a la hora de decidir a qué clase pertenece un objeto en una imagen para poder tomar las decisiones oportunas.

Por otro lado, debido a que es un campo relativamente nuevo, hay varios frameworks en los que se desarrollan las redes, y todavía no hay estándares, resulta complicado poder usar cualquier red desde tu dispositivo, ya que cada framework tiene sus necesidades y dependencias, por tanto necesitas entornos dispares dependiendo del framework. Por otro lado, la utilización de redes neuronales aplicadas a la conducción autónoma requieren de una gran capacidad de cálculo, la cual no es posible tener en un coche, pues su consumo de energía es muy elevado. Sin embargo, sin una capacidad de cálculo elevada es difícil conseguir obtener la profundidad de los elementos de una imagen en tiempos de inferencia muy reducidos, uno de los requisitos necesarios para que el vehículo sea capaz de reaccionar a tiempo frente a cualquier situación. Vamos a utilizar OpenVINO, que cubre ambos aspectos mencionados: Por un lado, pasa las redes a un formato estándar en el que se pueden portar a distintos tipos de hardware de Intel como CPU, GPU, FPGA y dispositivos de bajo consumo como Movidius, indistintamente de en qué framework se hayan creado; y por otro lado optimiza los modelos para reducir el tiempo de inferencia. Esto último es bastante interesante debido al requisito anteriormente explicado acerca de la necesidad de una velocidad de inferencia muy alta. Por ello, otro de nuestros objetivos es utilizar las redes neuronales para clasificación de objetos y para cálculo de disparidad publicadas en el KITTI sobre esta tecnología, lo cual, para estas últimas, aún nadie ha realizado, y así medir su mejora de velocidad de inferencia con OpenVINO. Además se utilizará Movidius junto con OpenVINO, permitiéndonos también enfrentarnos al requisito del limitado uso de energía disponible para cómputo, gracias a que Movidius permite realizar

cálculos vectoriales con una potencia menor que una GPU pero con un menor consumo de energía.

A continuación se enumera de una forma resumida los objetivos previamente explicados:

1. Toma de contacto con Deep Learning y redes neuronales a través de la realización de cursos y documentación online.

2. Inferencia y entrenamiento con redes neuronales para clasificación de imágenes, con el objetivo de familiarizarnos con el proceso de entrenamiento y obtención de resultados tras la clasificación de una imagen.

3. Reproducción del trabajo realizado por los desarrolladores de las redes neuronales para predicción de disparidad publicadas en KITTI, incluyendo entrenamiento de las mismas y realización de la inferencia en cada una de ellas para obtener resultados comparables según las métricas del KITTI.

4. Utilización de OpenVINO sobre redes neuronales para clasificación de objetos y predicción de disparidad.

4.1.- Familiarización con redes neuronales

Se realizan cursos de deep learning para clasificación y detección de objetos para comprender mejor en qué se basan las tecnologías que vamos a utilizar. En estos cursos se cubren aspectos fundamentales como: descriptores, clasificadores, búsqueda y refinamiento, y la evaluación del rendimiento.

El procedimiento de detectar un objeto en una imagen consiste en buscar en la imagen aquellas áreas que pueden ser el objeto que se busca, y compararlas con un modelo de ese objeto que compruebe si se parece lo suficiente a él o no. Para ello, hay que crear un modelo que sea capaz de distinguir entre diferentes tipos de objetos. Para que este modelo funcione correctamente, ha de ser capaz de detectar objetos que pertenecen a una misma clase, aunque sean un poco distintos entre sí, por ejemplo, si se están detectando personas, debería ser capaz de hacerlo independientemente de su tamaño, color de piel, tipo de ropa… Esto tiene que hacerse independientemente de las características del entorno en el que está el objeto. Por ejemplo, si hay más o menos luz, una persona sigue siendo una persona. Una vez tenemos ese modelo, hay que buscar todas las posibles apariciones en la imagen. Este puede ser un proceso largo, por lo que hay que tratar de mejorar la eficiencia de la búsqueda todo lo posible. Durante la búsqueda, se extraen características y se generan posibles candidatos del objeto que estamos buscando, se clasifican esos candidatos para descartar aquellos que verdaderamente no se adaptan al modelo, y posteriormente se refina esta decisión para obtener sólo aquellos que lo hacen con una alta probabilidad.

Como estamos trabajando con imágenes, es importante saber cómo están compuestas, por píxeles. En una imagen a color, un píxel tiene 3 canales (RGB), el rojo, el verde y el azul, por lo tanto tiene asociados 3 valores (uno para cada canal) que van entre 0 y 255. Estos valores, y por tanto el color del píxel, pueden variar de acuerdo con el color de la luz, que varía dependiendo de las longitudes de onda que la componen, de las superficies que aparecen en la imagen, que absorben y reflejan en distintas proporciones las longitudes de onda de la luz, y de la sensibilidad de la cámara, la cual tiene 3 sensores, uno para cada canal RGB y dependiendo de cómo estén calibrados el color puede variar.

Para extraer las características se emplean descriptores, que nos indican si han encontrado o no las características propias del modelo, buscadas en la imagen. Es importante que estos descriptores funcionen bien en todos los casos posibles. Para ello hay que tener en cuenta que la intensidad puede cambiar, y si no se trata adecuadamente, puede hacer que nuestro descriptor no funcione correctamente. Una forma sencilla de hacer esto, es usar las coordenadas cromáticas, que consiste en dividir cada canal RGB, por la intensidad (la suma de los canales). Otra posibilidad es que el haya cambios de color producidos por la luz o la cámara, esto se puede minimizar haciendo que la imagen sea una imagen canónica, que simula que la imagen se ha tomado con una luz controlada. Para ello hay que aplicar un filtro la imagen antes analizarla. Uno de estos algoritmos es el white patch, que asume que el color máximo en cada canal se corresponde con el blanco, y adapta el resto en

consecuencia, para ello multiplica cada canal por 255 y lo divide por el máximo de ese canal en la imagen.

Cuando ya se han clasificado los píxeles y se sabe cuáles pertenecen a la clase buscada, se generan las ventanas para poder localizar a los objetos de dicha clase en la imagen. Para ello se asignan etiquetas a los píxeles de dicha clase, teniendo la misma etiqueta aquellos que se encuentran conectados, por lo que al final, cada objeto distinto tendrá una etiqueta distinta. A continuación se crean las ventanas, para ello se buscan los extremos superior izquierdo e inferior derecho de cada objeto, a partir de las menores y mayores coordenadas x e y de los píxeles de cada etiqueta.

4.2.- Redes neuronales de clasificación y detección de objetos

Como sabemos, las redes de clasificación de objetos tienen como finalidad identificar la clase de objeto a la que pertenece la representación de una imagen. Como salida tendremos la probabilidad de que la imagen pertenezca a cada una de las clases que es capaz de clasificar, entendiendo que pertenece a la que mayor probabilidad tenga. No se debe confundir con otro tipo de red muy similar llamada red de detección de objetos. Estas últimas nos proporcionan un conjunto de objetos que ha podido identificar junto a la clase a la que pertenecen y la región de la imagen en la que aparece el objeto.

Conocer las principales redes de clasificación y detección de objetos forman parte de la toma de contacto con las redes neuronales.

En el siguiente enlace se pueden conocer las principales redes neuronales destinadas a la clasificación y detección de objetos:

https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/.

A continuación se explicarán de una forma introductoria dichas redes, de las cuales, algunas han sido utilizadas en el punto 4.2.8 como una previa toma de contacto con las redes neuronales utilizadas sobre frameworks de desarrollo para realizar entrenamientos e inferencia.

4.2.1.- AlexNet

Sobre la primera red, AlexNet, cabe destacar que es una arquitectura de red muy simple pero muy potente.
Si nos fijamos en su estructura, se observa que es una arquitectura muy sencilla con capas convolucionales y de pooling. La diferencia principal es el uso de la GPU para entrenar la red, ya que inicialmente se empleaba la CPU solamente para dicha tarea. Esta red implementa el uso de las GPUs lo cual acelera el entrenamiento hasta 10 veces más rápido.

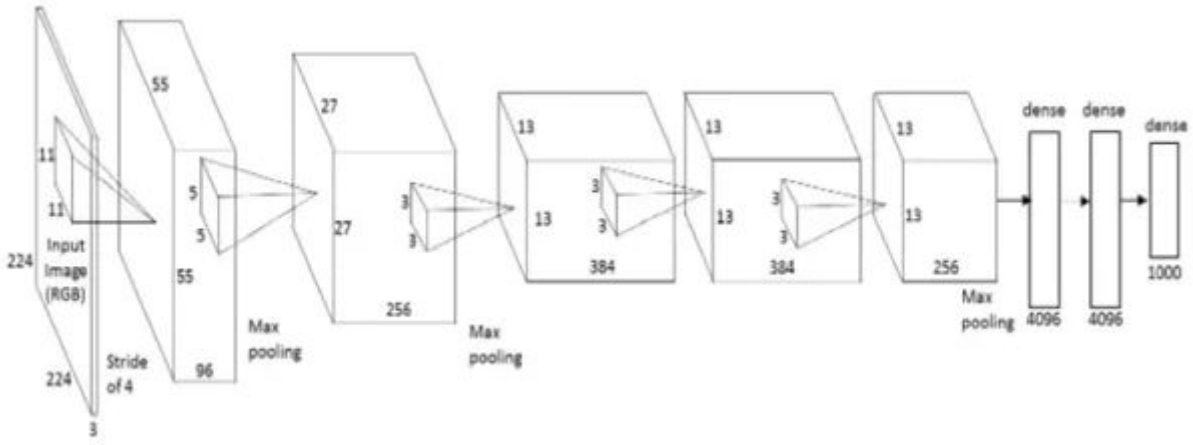


Imagen de la topología de la red[8]

```
#AlexNet with batch normalization in Keras
#input image is 224x224

model = Sequential()
model.add(Convolution2D(64, 3, 11, 11, border_mode='full'))
model.add(BatchNormalization((64,226,226)))
model.add(Activation('relu'))
model.add(MaxPooling2D(poolsize=(3, 3)))

model.add(Convolution2D(128, 64, 7, 7, border_mode='full'))
model.add(BatchNormalization((128,115,115)))
model.add(Activation('relu'))
model.add(MaxPooling2D(poolsize=(3, 3)))

model.add(Convolution2D(192, 128, 3, 3, border_mode='full'))
model.add(BatchNormalization((128,112,112)))
model.add(Activation('relu'))
model.add(MaxPooling2D(poolsize=(3, 3)))

model.add(Convolution2D(256, 192, 3, 3, border_mode='full'))
model.add(BatchNormalization((128,108,108)))
model.add(Activation('relu'))
model.add(MaxPooling2D(poolsize=(3, 3)))

model.add(Flatten())
model.add(Dense(12*12*256, 4096, init='normal'))
model.add(BatchNormalization(4096))
model.add(Activation('relu'))
model.add(Dense(4096, 4096, init='normal'))
model.add(BatchNormalization(4096))
model.add(Activation('relu'))
model.add(Dense(4096, 1000, init='normal'))
model.add(BatchNormalization(1000))
model.add(Activation('softmax'))
```

Código de implementación de la red[9]

8 https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/

9 https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/

4.2.2.- VGG16

La segunda red, VGG, se caracteriza por su forma piramidal. Las capas van estrechándose y profundizándose a medida que se avanza en la misma. Hay capas convolucionales seguidas de capas de pooling. Estas últimas son las responsables de ir estrechando las capas.

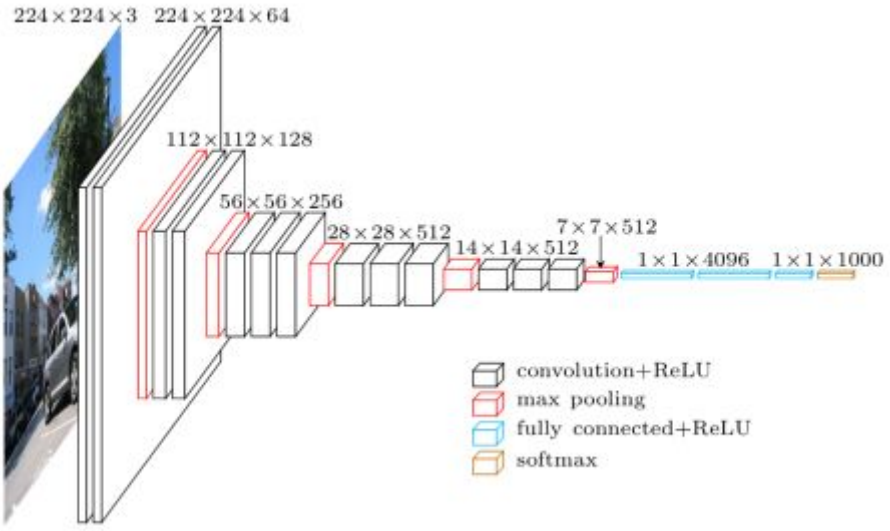


Imagen de la topología de la red[10]

4.2.3.- GoogleNet

La siguiente red es GoogleNet. Fue la ganadora de la competición ImageNet 2014 de reconocimiento visual, por lo cual es una muy buena opción para su estudio. En comparación con la anterior red que empleamos, tiene 22 capas, mientras que VGG tiene solamente 19. La principal diferencia que observamos es que incorpora en una única capa la opción de elegir cómo seguir con el trabajo realizado. En la siguiente imagen se puede apreciar esta capa descrita:

10 https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/

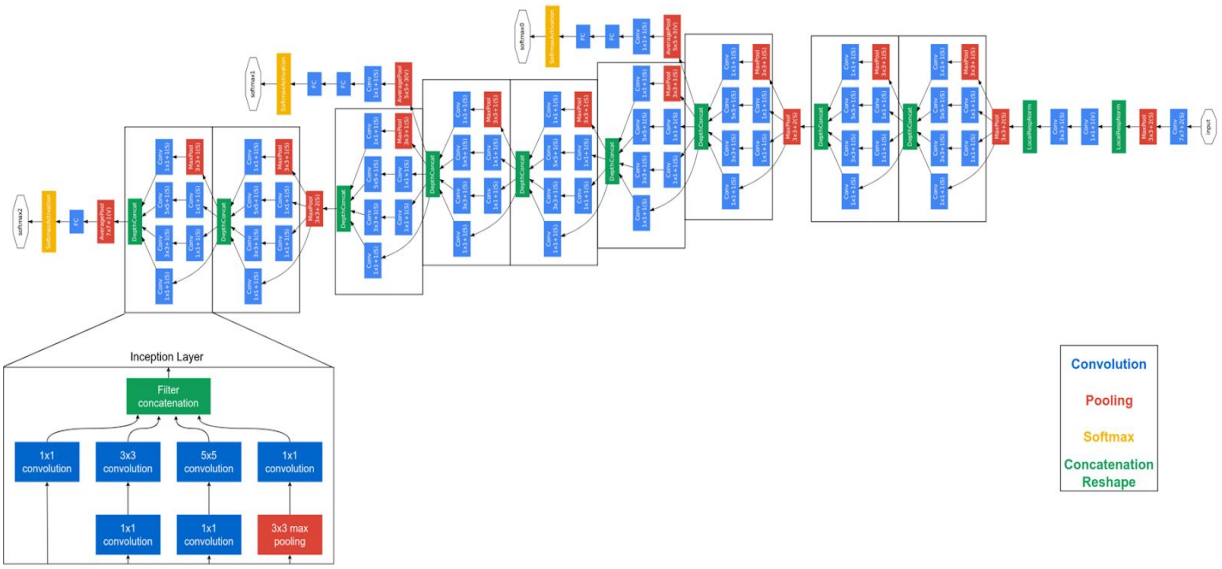


Imagen de la topología de la red[11]

Como se puede observar, se da la opción de aplicar una convolución o realizar un pooling de lo que se reciba de la capa anterior.

4.2.4.- Inception-v3

Inception-v3 tiene 42 capas, mientras que GoogleNet tiene solamente 22, además el coste computacional es solamente dos veces y media superior a GoogleNet. La principal diferencia que observamos es que incorpora la factorización en las capas de convolución que ayuda a reducir el problema del sobreajuste. En la siguiente imagen se puede apreciar esta topología descrita:

11 *https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/*

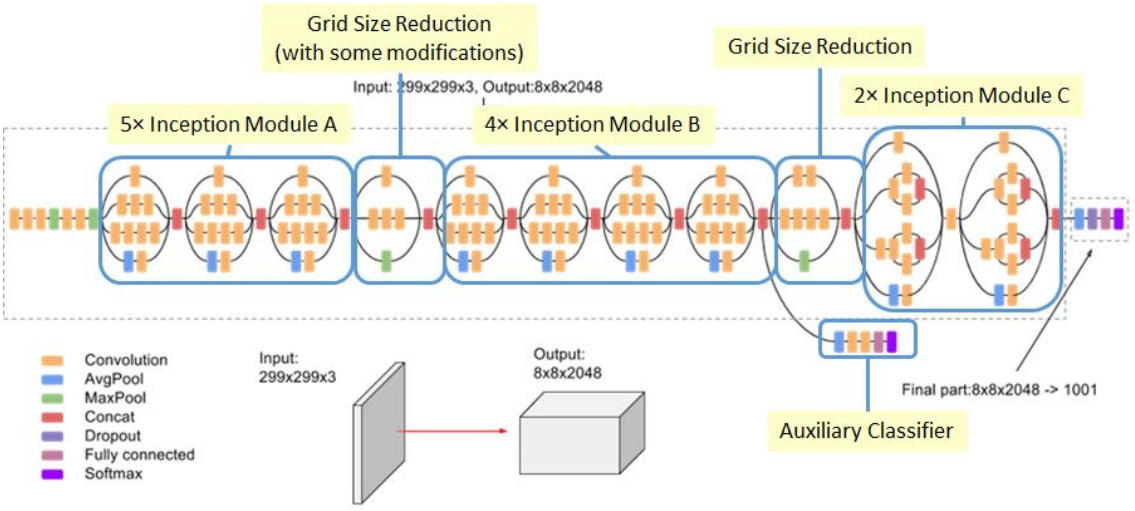


Imagen de la topología de la red[12]

4.2.5.- ResNet

ResNet implementa una serie de módulos residuales (de ahí su nombre ResNet, Residual Networks). Estos módulos dan la opción de realizar una serie de acciones sobre la entrada recibida, o por el contrario pasar de largo y redirigir la salida tal cual entró.

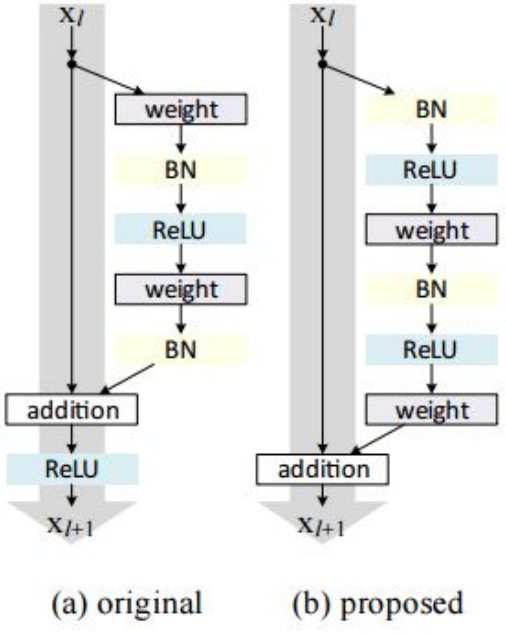


Imagen de la topología de la red[13]

12 https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

13 *https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/*

4.2.6.- ResNeXt

ResNeXt se considera el estado del arte en cuanto a reconocimiento de objetos. En la siguiente imagen se observa la comparativa entre ResNet, la red anteriormente comentada, y ResNeXt (izquierda y derecha respectivamente):

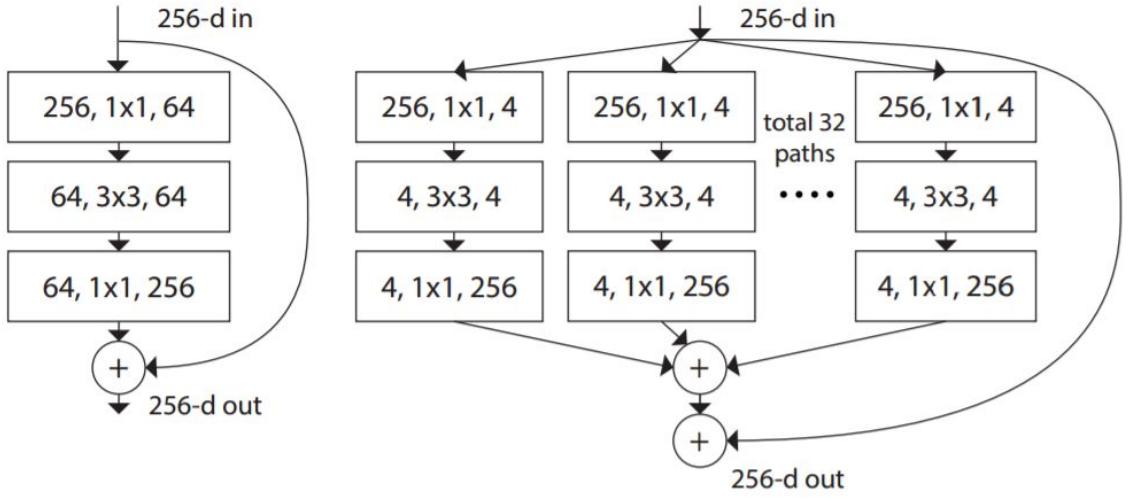
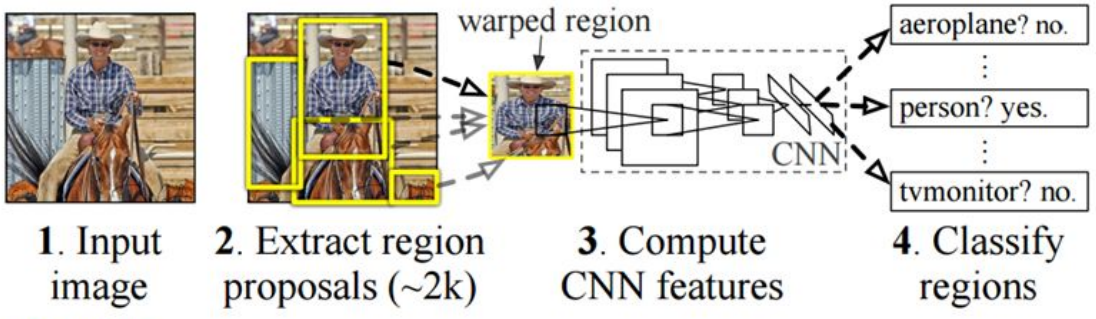


Imagen de comparación entre ResNet y ResNeXt[14]

4.2.7.- RCNN

RCNN (Region Based CNN) realiza la detección de objetos de la siguiente manera: traza una serie de áreas dentro de la imagen, y después intenta reconocer los objetos que hay en cada una de las mismas. En la siguiente imagen se puede observar su funcionamiento:



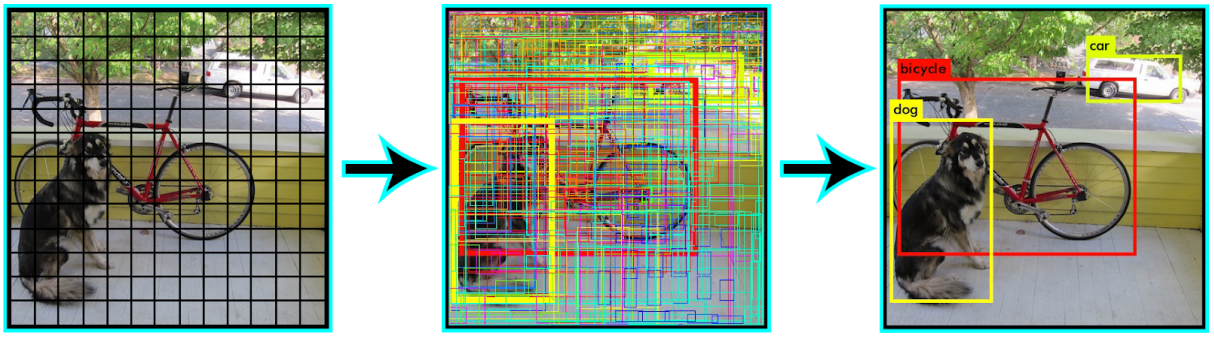
Funcionamiento red RCNN[15]

14 https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/

15 https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/

4.2.8.- YOLO

YOLO divide la imagen en celdas, y después analiza cada celda para identificar a qué pertenece. Una vez ha clasificado las celdas, las va fusionando para formar celdas más grandes que representen los objetos finales identificados. Su funcionamiento es el siguiente:



Funcionamiento red YOLO[16]

4.2.9.- Entrenamiento e inferencia con redes neuronales para clasificación de objetos

El trabajo con redes neuronales para clasificación de objetos consiste en entrenar e inferir diferentes redes.

Este trabajo es realizado sobre una máquina Debian, 8 cores AMD Opteron y 16GB de RAM, la cual no dispone de GPU. En ella hay configurada un ambiente proveído principalmente de Python y Keras, que utiliza con backend TensorFlow. La versión de Tensorflow que se instala por defecto emplea el conjunto de instrucciones SSE4.1. Debido a esta limitación, puesto que el procesador empleado no soporta ese repertorio de instrucciones, como solución se utiliza una versión de TensorFlow que no utilice dichas instrucciones, como es la 1.5.

Por un lado, en cuanto en el entrenamiento, se utiliza una implementación de AlexNet sobre Keras utilizando 100 épocas. Las épocas son el número de veces que se realiza el entrenamiento de una red utilizando una única vez cada imagen del dataset. A mayor número de épocas, se obtiene una red entrenada con una precisión superior, debido a que se vuelve a realizar un ajuste de los pesos que hacen que la red se adapte mejor a la hora de inferir sobre el dataset empleado. Empleando estas 100 épocas con un tamaño de lote o batch size de 128, siendo el tiempo de finalización excesivo, implica conseguir el 10% del total en dos días. Para solventar esto, el proceso de entrenamiento consta de 10 épocas y

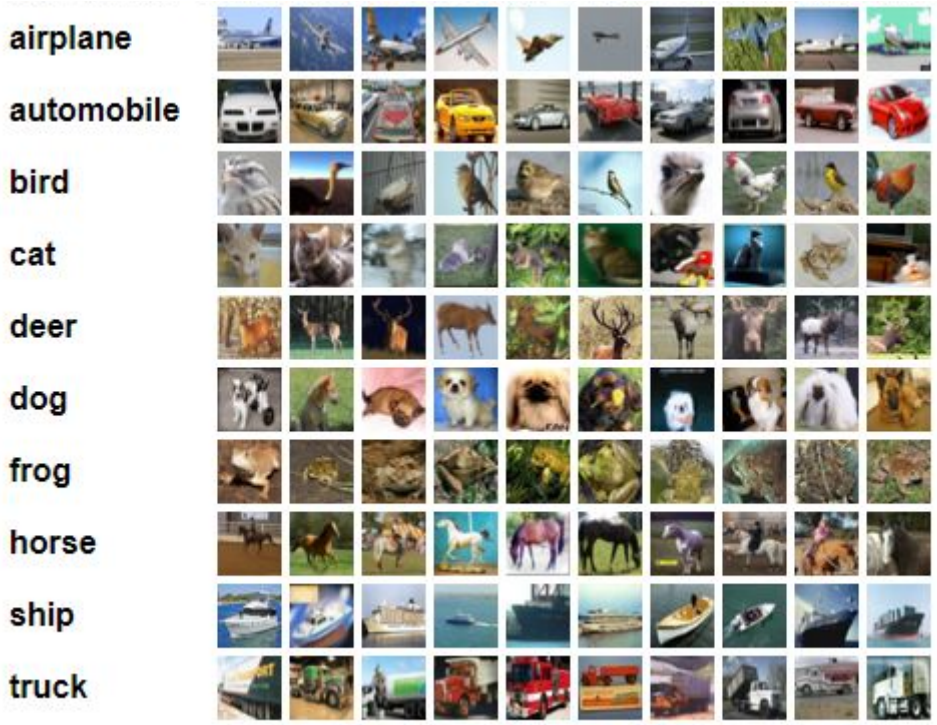
16 https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/

un batch-size de 16, reduciendo así considerablemente el tiempo requerido a dos días. Tras finalizar el proceso de entrenamiento, se lleva a cabo una evaluación de la red y el guardado de la topología y pesos, permitiéndonos volver a cargar y usar la red sin necesidad de volver a entrenar. Además, el modelo utilizado es una modificación en la que su última capa tiene 10 salidas, debido a que el dataset que va a ser utilizado consta únicamente de 10 clases.

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
Train on 50000 samples, validate on 10000 samples
Epoch 1/5
 3392/50000 [=>............................] - ETA: 4:13:58 - loss: 1.9031 - acc: 0.3057
```

Ejemplo de una secuencia de entrenamiento para 5 épocas

Para el entrenamiento se utiliza el dataset CIFAR-10. Esto es debido a que en el ambiente de desarrollo utilizar otros datasets tan grandes como Imagenet supondría un aumento en el tiempo necesario y una mayor ocupación de disco duro no disponible. Siendo 150 GB el tamaño requerido para ImageNet, mientras que CIFAR-10 requiere de 170 MB. Dicho dataset consta de 60000 imágenes de tamaño 32x32 a color, divididas en 10 clases, con un total de 6000 imágenes por clase. De entre todas estas imágenes, hay 50000 para entrenar las redes neuronales, y 10000 para realizar el posterior testeo de la eficacia de las mismas. Las 50000 fotos de entrenamiento se dividen en 5 lotes de 10000. Hay 10 clases de imágenes: Aviones, Automóviles, Pájaros, Gatos, Ciervos, Perros, Ranas, Caballos, Barcos y Camiones. No todas las categorías son aplicables a la conducción autónoma, que es el campo que nos compete, pero para una primera toma de contacto nos pareció un dataset muy completo. El lote de imágenes para testear las redes contiene 1000 elegidas aleatoriamente de cada clase, mientras que los lotes de entrenamiento contienen las imágenes no seleccionadas, en orden aleatorio.



Captura de la web de CIFAR10 con las categorías existentes y 10 imágenes aleatorias de cada categoría[17]

Por otro lado, en cuanto a la inferencia se utilizan redes neuronales pre-entrenadas sobre Imagenet y proporcionadas por Keras, como es el caso de VGG-16. Además de estas redes pre-entrenadas, re infiere adicionalmente sobre la red entrenada anteriormente, utilizando la topología y pesos previamente obtenidos. Para la realización de inferencia se utilizan imágenes aleatorias, no pertenecientes a ningún dataset en concreto.

En cuanto a los resultados, al haber estado entrenados sobre dataset diferentes, debido a la limitación de tiempo y espacio previamente explicada, no podemos comparar la precisión con la que clasifican, pero sí la velocidad a la que lo hacen y la especificidad con la que lo hacen.

Los resultados en cuanto a tiempos de inferencia son los siguientes:

Red	Tiempo de inferencia sobre una imagen (en segundos)
VGG-16	1.330
AlexNet	0.670

[17] https://alexisbcook.github.io/2017/using-transfer-learning-to-classify-images-with-keras/

Podemos observar que, a pesar de que VGG-16 es una red que podría estar optimizada para Keras, debido a que está implementada por sus propios desarrolladores, tiene unos tiempos de inferencia muy inferiores respecto a AlexNet, la cual ha sido desarrollada a partir de una implementación no oficial. Esto es producido por la propia naturaleza y arquitectura de la red. VGG-16 está compuesta por un número de capas igual a 16 y más complejas que AlexNet, que dispone de la mitad de capas y siendo estas más simples. Esto da lugar a que AlexNet sea más rápida.

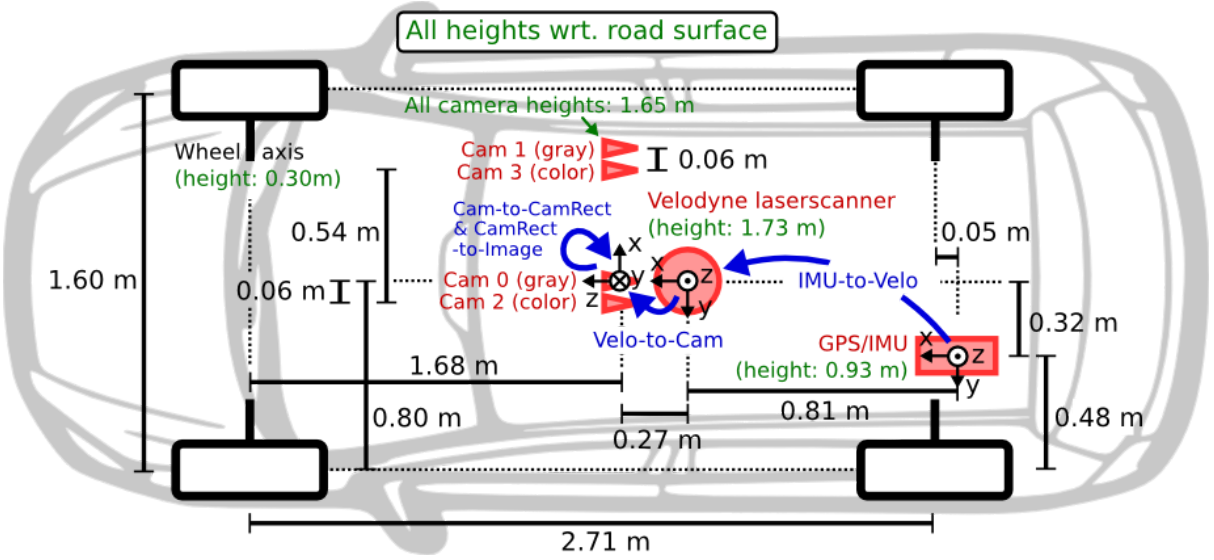
En cuanto a la especificidad de la red, VGG-16 es superior a AlexNet, debido a que ha sido entrenada sobre ImageNet, que consta de 1000 clases, mientras que CIFAR-10 sólo de 10. Para la siguiente imagen, VGG-16 es capaz de detectar que es un coche deportivo, mientras que AlexNet sólo detecta que se trata de un automóvil.

Imagen utilizada para el testeo de las redes[18]

[18] https://www.elmundo.es/motor/2018/07/30/5b5ef0ee22601d5b158b459e.html

4.3.- KITTI[19]

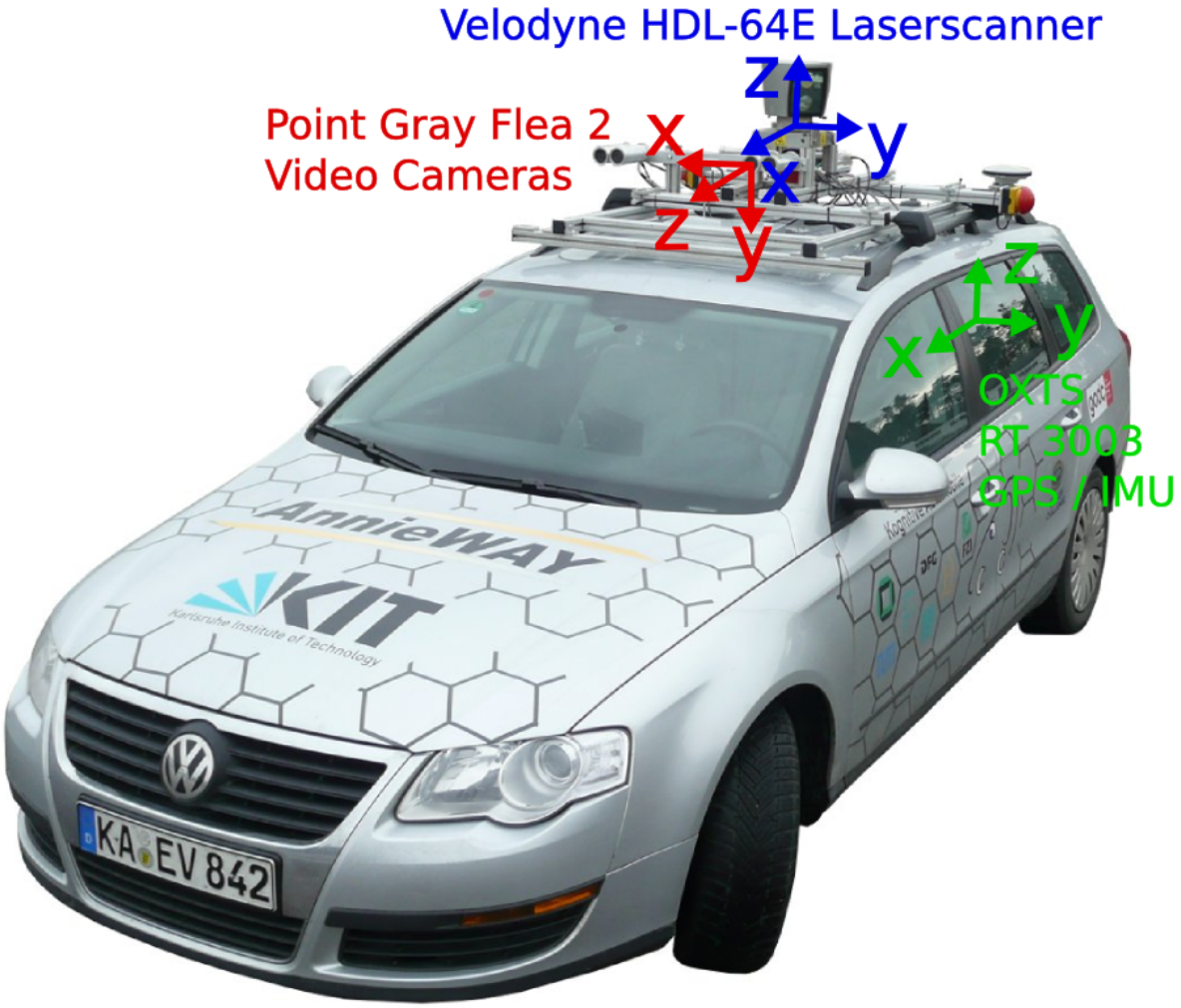
KITTI es una suite de datasets recopilados por una plataforma de conducción autónoma llamada Annieway, un proyecto del Instituto de Tecnología Karlsruhe y el Instituto Tecnológico de Chicago. Dichos datasets se pueden emplear en visión en stereo, optical flow, visual odometry, 3D object detection and 3D tracking (empleamos los términos en inglés ya que estos términos son usados en español también). Por otro lado, las imágenes son tomadas desde un coche equipado con dos cámaras de vídeo de alta resolución en color y en escala de grises. Para proporcionar una mayor veracidad en cuanto a la distancia del suelo, se emplea un láser Velodyne y un sistema GPS de localización. Todas las imágenes se toman en la ciudad de Karlsruhe, en áreas rurales y en autopistas. En cada imagen se pueden encontrar hasta un máximo de 15 coches y 30 peatones. Aparte de proporcionar todas las imágenes en bruto, también se proporcionan benchmarks específicos para cada tarea, añadiendo también métricas de evaluación y una web para evaluar lo obtenido.



Plano del coche utilizado en la recolección de imágenes del dataset[20]

[19] http://www.cvlibs.net/datasets/kitti/

[20] *http://www.cvlibs.net/datasets/kitti/setup.php*



Imágen del coche utilizado para la recolección de imágenes del dataset[21]

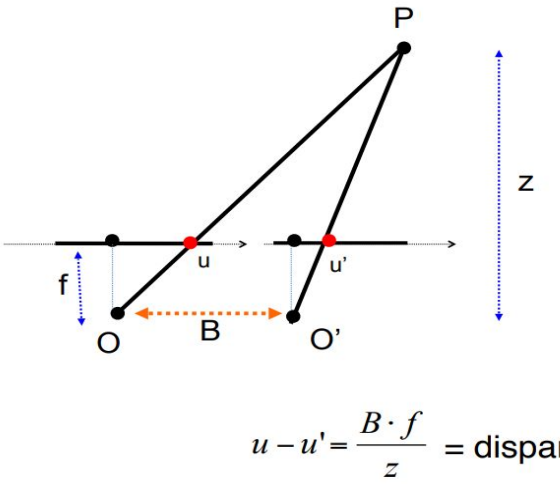
Decidimos profundizar también en el concepto de la disparidad antes de meternos en materia.

[21] http://www.cvlibs.net/datasets/kitti/setup.php

4.4.- Redes de predicción de disparidad

Los seres humanos tenemos un sistema visual capaz de ver en tres dimensiones. Esto es posible gracias a la visión binocular. Dicha visión es generada debido a que nuestros ojos miran a un mismo objeto desde ángulos ligeramente distintos (debido a la pequeña separación que existe entre ellos). Como resultado, se obtienen dos imágenes muy parecidas, pero no iguales. La disparidad binocular es la diferencia entre los dos puntos de vista de ambos ojos. Dicha diferencia es la que permite crear imágenes 3D en superficies llanas. Hablando de disparidad, podemos encontrar dos tipos: cruzada y no cruzada: En la disparidad cruzada el objeto que ve el ojo está más cerca que el punto de fijación, mientras que en la no cruzada el objeto está más alejado que dicho punto.

De estos conceptos, nace la estimación de la profundidad de objetos en una escena, partiendo de un par de imágenes (estéreo). De estas dos imágenes podemos calcular la disparidad para así obtener la profundidad, utilizando la fórmula descrita en la imágen inferior. Esta disparidad calculada proporciona la profundidad de cada píxel en cada una de las imágenes. Dicho cálculo se realiza detectando qué píxeles de la imagen de la izquierda corresponde con los pixeles de la derecha, o viceversa, para así calcular la distancia entre ambos, la llamada disparidad. A este problema se le conoce como Stereo Matching. A partir de la disparidad y otros parámetros cuyo valores ya conocemos, como la distancia focal o la línea base, podemos estimar la profundidad. Los algoritmos que calculan dicha disparidad, asumen que las imágenes están rectificadas, lo que significa que los planos de ambas imágenes son paralelos entre sí y a la dirección en la que existe el desplazamiento de las imágenes. Esto es garantizado en la forma de obtener las imágenes. Pero puede darse el caso de que existan oclusiones, puntos o regiones que se ven en una imagen y no en la otra por estar tapadas por un objeto. Estas oclusiones son fuente de errores en muchos algoritmos, pero a su vez pueden ser usadas para recuperar la estructura de la escena y darnos mucha información esencial sobre la misma.



Cálculo de la disparidad y relación con línea base (B), distancia focal (f) y profundidad (z)[22]

Una vez realizado todo esto, pasamos a realizar uno de los objetivos de este trabajo: realizar el entrenamiento e inferencia de distintas redes con este propósito.

En el apartado de Stereo Evaluation 2015 de la web de KITTI (http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo) encontramos las redes neuronales más actuales en cuanto a visión en estéreo se refiere. De ellas se han descartado aquellas sin ninguna referencia a páginas del desarrollador, y aquellas cuyo código no está accesible en un repositorio público, puesto que nuestro objetivo es evaluar las mismas. Nos quedamos con 30 redes un 15% de las 200 redes que aparecen, de las cuales no todas publican el código para entrenar la red (algunas emplean capas desarrolladas por empresas privadas y no quieren hacer público el código) o no tienen una implementación para realizar la inferencia. A continuación se detallan las redes con las que pudimos trabajar.

22 *https://answers.opencv.org/question/187734/derivation-for-perspective-transformation-matrix-q/*

4.4.1. - SegStereo y CRL

La primera red objeto de estudio es SegStereo. Puesto 37 KITTI-Stereo (durante el desarrollo del trabajo).

Dicha red emplea características semánticas de segmentación para que su implementación mejore la precisión de la predicción en mapas de disparidad. Según se puede observar en su paper, es una red que alcanza resultados buenos dentro del estado del arte con el benchmark Stereo del KITTI, y también lo han probado con los datasets de CityScapes y FlyingThings3D obteniendo resultados decentes.

Utilizan ResNet (de la cual hablamos anteriormente) con operación de correlación como codificador, y varios bloques deconvolucionales como decodificador para volver a un mapa de disparidad de toda la imagen. La operación de correlación está diseñada para calcular los pesos de coincidencias basándose en pares de mapas de características. Emplean además una subred de segmentación para extraer características semánticas de la disparidad. Tanto la evaluación semántica como la disparidad semántica son totalmente convolucionales.

Su modelo permite tanto entrenamiento supervisado como no supervisado. En el aprendizaje no supervisado, la pérdida de consistencia fotométrica y la semántica se calculan y se propagan hacia atrás en la red.

En el aprendizaje supervisado, emplean la pérdida de regresión supervisada en lugar de la pérdida de consistencia fotométrica no supervisada, consiguiendo así los resultados en el KITTI antes comentados. Emplean un finetune (entrenamiento de una red partiendo de pesos ya existentes) para realizar los tests después con los datasets de CityScapes y FlyingThings3D.

Su implementación está hecha en Caffe, y de entre sus capas, no todas son originales suyas, ya que emplean dos capas de la FlowNet 2.0 (la de correlation y correlation1d) y una de PSPNet (la de interpolación). Así mismo, la herramienta para disparidad es del Toolkit de OpticalFlow de *liruoteng* (https://github.com/liruoteng/OpticalFlowToolkit)

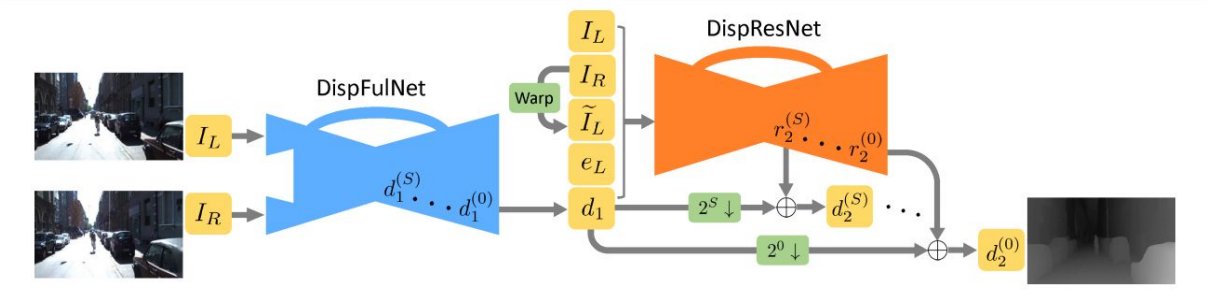
El proyecto de SegStereo se puede encontrar en Github. Dicho proyecto dispone de un script en Python para evaluar la disparidad de la red. A dicho script se le pasan como parámetro los pesos de la red entrenada, los cuales podemos encontrar en varios enlaces a Google Drive que aparecen en el repositorio; el modelo de la red, que podemos encontrar dentro de la carpeta *models/* y en el directorio de la red correspondiente; el directorio con las imágenes a evaluar (las cuales no se proporcionan directamente en el repositorio); el directorio para volcar la salida; y otros parámetros adicionales.

Para poder entrenar la red y llegar a realizar la inferencia, es necesario tener Caffe instalado en su versión GPU, puesto que varias de sus capas no están soportadas en la versión del

framework para CPU. Debido a esto, se utiliza una máquina con dos GPUs NVIDIA GTX 980 y una GPU NVIDIA GTX 1080.

A continuación se trabaja paralelamente con la red CRL puesto que está implementada en Caffe también.

CRL es una red con una estructura en cascada con dos etapas, como se puede observar en la foto que aparece debajo. La primera es una DispNet con módulos adicionales para realizar la convolución, proporcionando así una mayor exactitud en la métrica de la disparidad. La segunda etapa es una DispResNet que rectifica la disparidad de la primera etapa, generando señales residuales. El conjunto de las salidas de ambas etapas forman la disparidad final. Normalmente se obtendría la disparidad directamente después de la segunda etapa, pero han demostrado que este aprendizaje residual es más efectivo a la hora de obtener una mayor precisión. Este modelo, al igual que la anterior SegStereo, ofrece unos resultados dentro del estado del arte en cuanto a rendimiento con imágenes en estéreo.



Arquitectura de CRL[23]

En el repositorio de CRL no se proporciona el código necesario para entrenar un modelo propio, porque una de las capas es propiedad de una empresa privada para la que trabajan los desarrolladores, por lo cual únicamente se realiza la inferencia sobre los modelos proporcionados.

Para poder entrenar SegStereo, que de ambas redes es la única que proporciona código de entrenamiento, hay que emplear el comando *caffe train*. Del mismo modo, para poder realizar la inferencia, hay que obtener un archivo *deploy* a partir del archivo *solver* que tiene la topología de la red. Para ello, se eliminan las capas de datos de entrenamiento del solver, ya que no son necesarias a partir de ahora, se añade una única capa "data" con las proporciones de las imágenes del dataset a testear.

A la hora de realizar la inferencia, debido a un error producido al procesar las capas, se detectó que el proyecto de SegStereo y el de CRL emplean un Caffe modificado según sus necesidades, que tiene sus propias capas personalizadas, distintas a las de la rama

[23] https://arxiv.org/pdf/1708.09204.pdf

principal de Caffe mantenida por la comunidad. Estas modificaciones de Caffe, que incluyen las capas necesarias para poder trabajar con estas redes, presenta numerosas dependencias a nivel de librerías, por lo que no ha sido posible compilarlo.

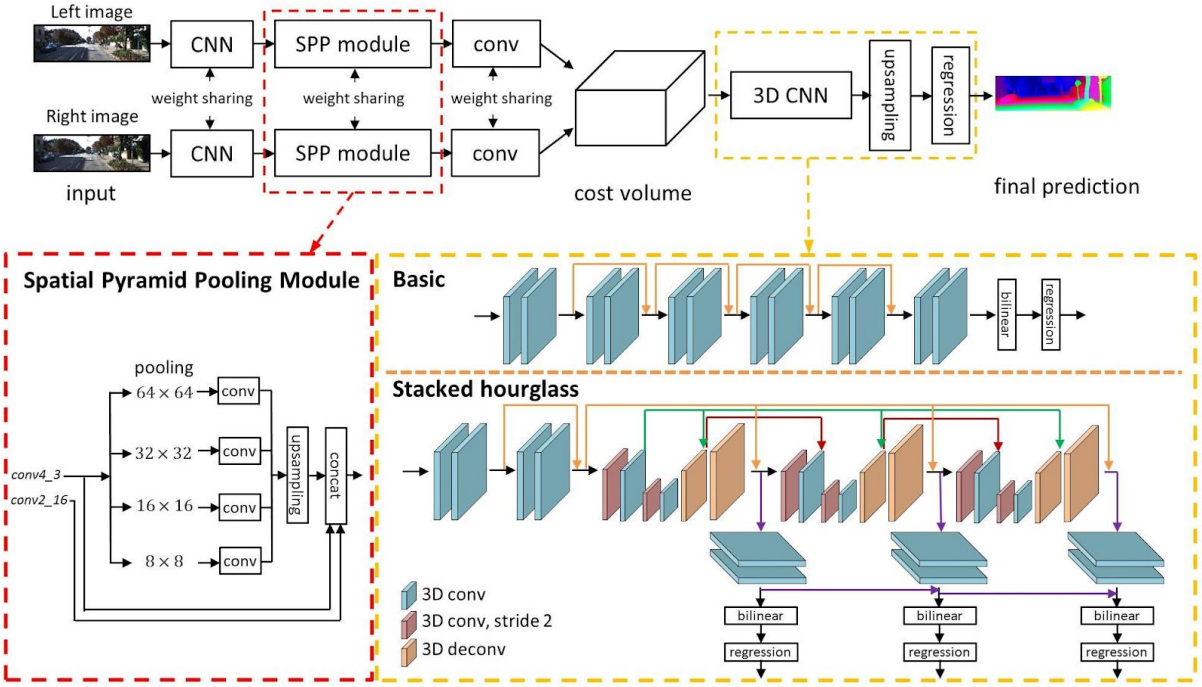
Debido a esto, no es posible realizar la inferencia con estas dos redes, por tanto no podemos sacar unas estadísticas de rendimiento ni compararlas con las otras redes de las que se han sacado resultados.

4.4.2.- PSMNet

Puesto 49 KITTI-Stereo (durante el desarrollo del trabajo).

PSMNet es una red utilizada para el cálculo de disparidad implementada en PyTorch. Las arquitecturas más comunes de redes están basadas en el procesamiento de un par de imágenes (izquierda y derecha) a través de dos redes neuronales siamesas. Éstas dividen la imágen en porciones que posteriormente alimentan a redes convolucionales, sin embargo, esta técnica presenta problemas a la hora de no perder información de las zonas de la imagen que separan unas porciones de otras. Para resolver esto, PSMNet propone una arquitectura de red basada en un pooling piramidal para no perder dicha información. El pooling piramidal resumen la imagen en diferentes tamaños de porciones de imagen y extrae información de todas ellas. Por otro lado utiliza redes convolucionales 3D. Estas últimas presentadas en dos versiones, utilizando el un modelo básico o stacked hourglass.

En el caso de la primera, el modelo básico consta de bloques residuales de 12 capas convencionales 3D (3x3x3). Por otro lado el modelo StackHourglass es más complejo, utilizando en su arquitectura 3 redes StackHourglass conectadas entre sí y que calculan cada una de ellas su mapa de disparidad, utilizándose la salida de las tres para recalcular los pesos en el entrenamiento y la última red para realizar la inferencia. Tiene la misma función que el pooling piramidal, no perder la información del contexto entre las particiones de la imagen tras ser separadas y evaluadas independientemente.



Topología de la red[24]

[24] https://github.com/JiaRenChang/PSMNet

En el KITTI y en su repositorio se puede encontrar resultados y redes pre-entrenadas de PSMNet con su modelo StackHourglass. Sin embargo, no hay nada de esto con su modelo básico. Como más adelante se indica, debido a nuestras limitaciones para utilizar el modelo StackHourglass y la falta de resultados con su modelo básico, nuestro objetivo con esta red es entrenar la red con el modelo básico, aportando una red actualmente no disponible entrenada, además de los resultados de inferencia obtenidos, tampoco publicados en el KITTI. Por otro lado, se utilizarán las redes pre-entrenadas, las cuales usan el modelo StackHourglass, para comparar los resultados bajo el mismo ambiente de trabajo.

La red está disponible para poder ser entrenada en el dataset SceneFlow. Dicho dataset está compuesto por 3 datasets sintéticos. Estos datasets contienen imágenes estéreo y sus respectivos groundtruths de disparidad sobre las temáticas de vuelo, conducción y un personaje animado con aspecto de mono llamado Monkaa. Además, también tiene disponible scripts para inferir disparidades y la realización de fine-tune sobre redes ya entrenadas a partir de scripts.

Para el proceso de reproducción de entrenamiento, inferencia y benchmark de PSMNet se ha utilizado un entorno sobre PyTorch con dos GPUs NVIDIA GTX 980 y una GPU NVIDIA GTX 1080. En dicho proceso con este entorno existen las siguientes limitaciones:

- Limitaciones en almacenamiento de disco duro: el dataset SceneFlow completo ocupa en torno a 210 GB, mientras que la capacidad disponible es de alrededor de 100 GB, teniendo en cuenta que no sólo almacenamos datasets.

- Limitaciones de memoria en GPU: las GPUs disponibles no tienen la memoria suficiente para soportar un entrenamiento sobre PSMNet con redes convolucionales 3D stacked-hourglass. En el caso de la inferencia no ocurre dicho problema, por ello que se ha podido reproducir la inferencia y benchmarks con los modelos ya pre-entrenados disponibles en su repositorio.

- Limitaciones de tiempo: realizar un entrenamiento de un número elevado de épocas implica un coste de tiempo demasiado alto, obligándose a no poder trabajar con otras redes o realizar otros procesos hasta que no hubiese terminado el entrenamiento.

Para solventar las anteriores limitaciones se toman las siguientes medidas:

- Medida para solventar limitación de almacenamiento: utilizar solamente parte del dataset SceneFlow. En concreto el subdataset Driving,el cual ocupa 25 GB y está relacionado con la temática de conducción, que es el tema sobre el que se basa nuestro trabajo.

- Medidas para solventar la memoria de GPU limitada: utilizar redes convolucionales 3D-Básicas para entrenamiento. La cual es menos precisa pero más rápida y

requiere menos memoria de GPU, capacitando el entrenamiento. Además de utilizar un batch-size menor disminuya el uso de memoria de GPU aún más.

- Medidas para solventar el tiempo limitado: realizar entrenamientos de 200 épocas sobre SceneFlow como máximo y utilización de varias GPUs heterogéneas a la vez durante el proceso de entrenamiento para disminuir el tiempo de ejecución del mismo.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.73       Driver Version: 410.73       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 980     Off  | 00000000:02:00.0 Off |                  N/A |
| 38%   70C    P2   166W / 180W |   2722MiB /  4043MiB |    100%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 1080    Off  | 00000000:03:00.0 Off |                  N/A |
| 29%   57C    P2   102W / 200W |   2849MiB /  8119MiB |     98%      Default |
+-------------------------------+----------------------+----------------------+
|   2  GeForce GTX 980     Off  | 00000000:82:00.0 Off |                  N/A |
| 28%   44C    P8    13W / 180W |     11MiB /  4043MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     28280      C   python                                     2711MiB |
|    1     28280      C   python                                     2839MiB |
+-----------------------------------------------------------------------------+
```

Utilización de GPUs heterogéneas durante el entrenamiento.

La reproducción del trabajo realizado sobre PSMNet por los desarrolladores consta de 4 etapas realizadas según nuestras limitaciones. El objetivo es obtener y comparar los resultados obtenidos, teniendo en cuenta las limitaciones anteriores, con los resultados publicados en el KITTI por sus desarrolladores, tanto en resultados de error promedio de inferencia como en tiempos de inferencia. Además de comparar el modelo básico, del cual no hay resultados de ningún tipo relacionados con él, y StackHourglass sobre un mismo ambiente, como se ha comentado anteriormente.

La primera etapa es el entrenamiento. PSMNet es entrenada con su modelo básico cuatro veces con diferentes números de épocas sobre el subdataset SceneFlow llamado Driving. Estos entrenamientos corresponden con 50, 100, 150 y 200 épocas respectivamente. La realización de un número mayor de épocas supone un coste de tiempo demasiado elevado, aproximadamente de una semana por cada 200 épocas e inhabilitando el uso de la máquina para otros fines durante este periodo.

La segunda etapa es la realización de tres fine-tunes sobre la red entrenada con 200 épocas, debido a que es la red que tiene mayor precisión. El objetivo de los fine-tunes es

mejorar aún más la precisión a la hora de inferir con el dataset del KITTI. Debido a esto, el dataset utilizado es un subconjunto del mismo, como han realizado los desarrolladores. El número de épocas utilizadas es de 20, 60 y 100 épocas respectivamente.

La tercera etapa es la ejecución de la inferencia sobre el dataset KITTI. A partir de todas las redes entrenadas disponibles se realiza la inferencia. Estas redes entrenadas corresponden con las 4 primeras redes entrenadas por nosotros en la primera etapa con el modelo básico, las 3 redes entrenadas con fine-tune sobre el modelo de 200 épocas realizado por nosotros en la tercera etapa y dos redes pre-entrenadas disponibles en su repositorio. Estas dos últimas redes constan de un modelo Staked-Hourglass, sin embargo una de ellas es entrenada sobre SceneFlow completa durante 10 épocas y la restante es una red con un fine-tune de 1000 épocas sobre el dataset KITTI utilizando como base la red StackHourglass entrenada durante 10 épocas, anteriormente citada.

La cuarta y última etapa es la ejecución del benchmark de KITTI, disponible en la plataforma como “development kit”. Esta etapa se realiza con todas las redes utilizadas en la etapa anterior. La ejecución del benchmark es sobre una herramienta desarrollada en C++ y Matlab.

A continuación se adjuntan dos tablas con los resultados de la reproducción sobre error promedio de disparidad cometido a la hora de inferir el dataset de KITTI, utilizando la herramienta de benchmark anteriormente descrita. La primera tabla corresponde al error cometido si tenemos en cuenta la disparidad y la segunda si no la tenemos en cuenta:

Leyenda	
FG	Porcentaje de valores atípicos promediados solo en regiones de primer plano
BG	Porcentaje de valores atípicos promediados solo en regiones de segundo plano
ALL	Porcentaje de valores atípicos promediados sobre todos los píxeles del groundtruth (precisión de la clasificación del conjunto de entrenamiento para técnicas de machine learning).

Error promedio de disparidad sobre KITTI teniendo en cuenta oclusión					
Modelo	N.º de épocas utilizadas para entrenamiento	N.º de épocas utilizadas para finetune	FG	BG	ALL

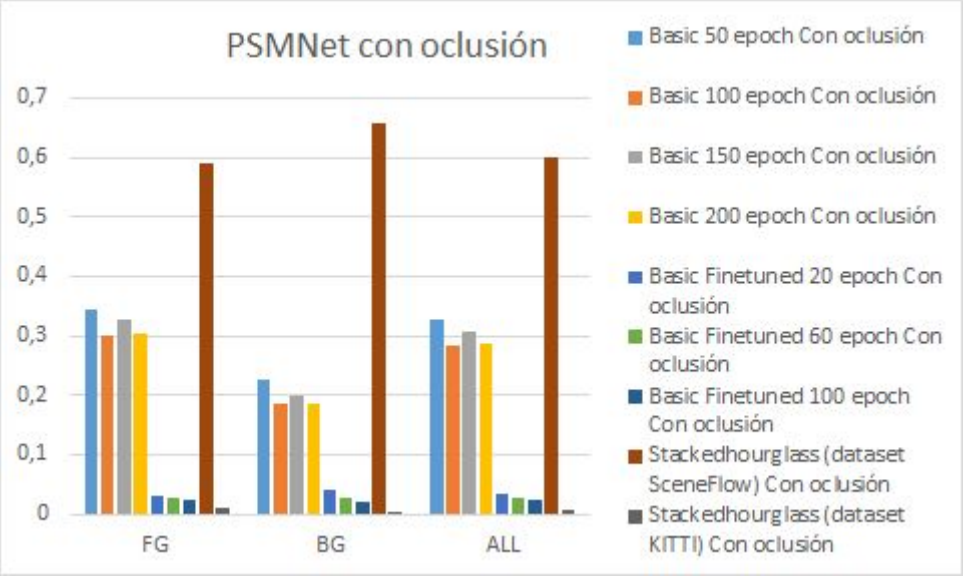
Basic	50	0	0.344629	0.226702	0.326991
Basic	100	0	0.300079	0.185429	0.282932
Basic	150	0	0.327358	0.200405	0.308371
Basic	200	0	0.304922	0.186698	0.287240
Basic	200	20	0.030800	0.042024	0.032479
Basic	200	60	0.027524	0.026481	0.027368
Basic	200	100	0.023689	0.021512	0.023364
Stackedhourglass	10*	0	0.588933	0.658463	0.599332
Stackedhourglass	10*	1000	0.009653	0.001704	0.008464

Error promedio de disparidad sobre KITTI sin tener en cuenta oclusión					
Modelo	N.º de épocas utilizadas para entrenamiento	N.º de épocas utilizadas para finetune	FG	BG	ALL
Basic	50	0	0.340618	0.220339	0.322851
Basic	100	0	0.296492	0.176746	0.278804
Basic	150	0	0.322555	0.189452	0.302894
Basic	200	0	0.300126	0.176827	0.281913
Basic	200	20	0.029239	0.036111	0.030254
Basic	200	60	0.026123	0.023112	0.025678
Basic	200	100	0.022499	0.017337	0.021736
Stackedhourglass	10*	0	0.586312	0.650923	0.595856
Stackedhourglass	10*	1000	0.008916	0.001490	0.007819

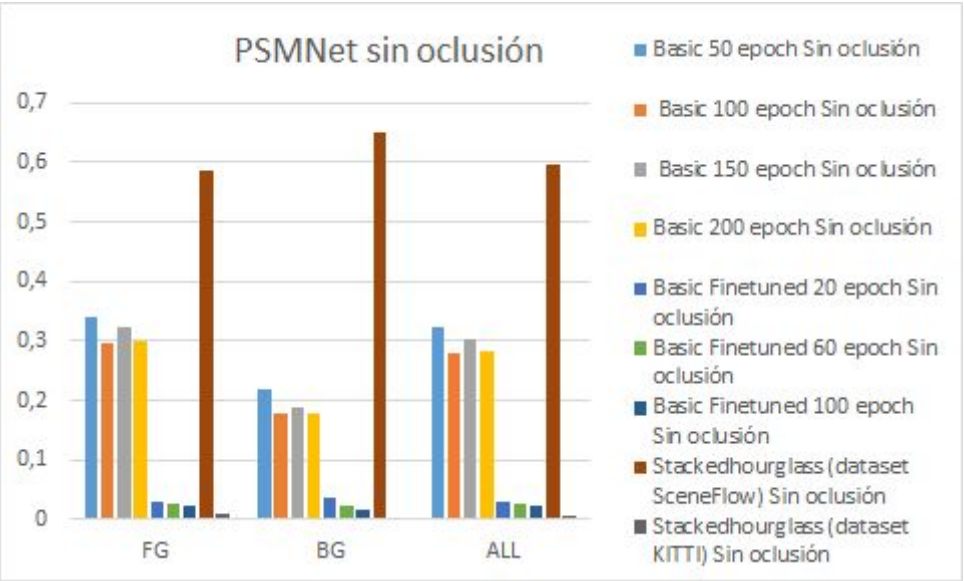
*El número de épocas realizadas en estas redes es sobre SceneFlow completo, no sobre el subdataset Driving de SceneFlow como hemos realizado con el modelo básico.

A continuación se muestra una gráfica de la comparación para PSMNet de los resultados obtenidos con oclusión y sin oclusión.

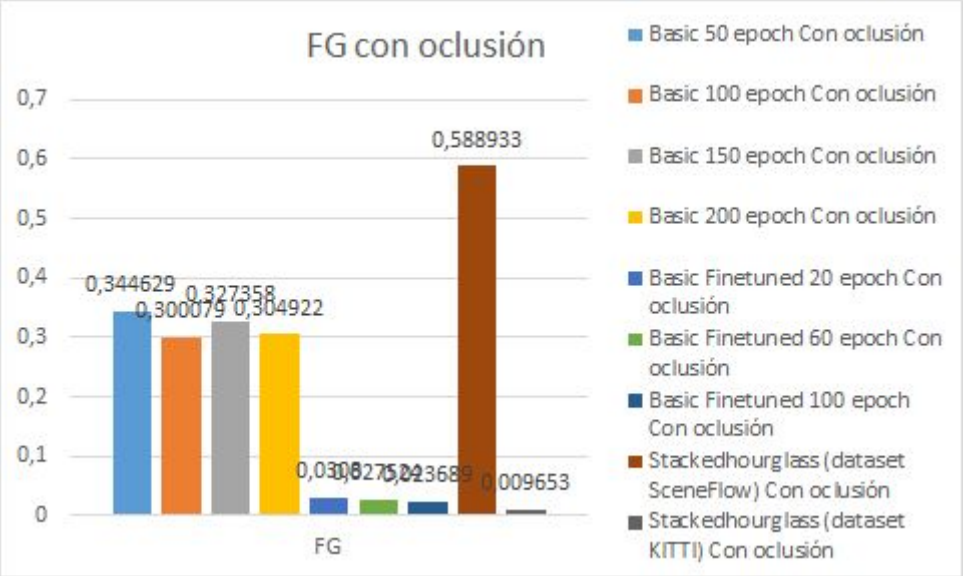
PSMNet	
Leyenda	Significado
Basic 50 epoch	PSMNet entrenada con su modelo básico con 50 épocas sobre el subdataset SceneFlow llamado Driving.
Basic 100 epoch	PSMNet entrenada con su modelo básico con 100 épocas sobre el subdataset SceneFlow llamado Driving.
Basic 150 epoch	PSMNet entrenada con su modelo básico con 150 épocas sobre el subdataset SceneFlow llamado Driving.
Basic 200 epoch	PSMNet entrenada con su modelo básico con 200 épocas sobre el subdataset SceneFlow llamado Driving.
Basic finetuned 20 epoch	PSMNet tras hacer fine-tune con 20 épocas sobre la red entrenada con 200 épocas sobre el modelo básico utilizando el dataset del KITTI.
Basic finetuned 60 epoch	PSMNet tras hacer fine-tune con 60 épocas sobre la red entrenada con 200 épocas sobre el modelo básico utilizando el dataset del KITTI.
Basic finetuned 100 epoch	PSMNet tras hacer fine-tune con 100 épocas sobre la red entrenada con 200 épocas sobre el modelo básico utilizando el dataset del KITTI.
Stackedhourglass (dataset SceneFlow)	PSMNet entrenada con su modelo Stacked-Hourglass con 10 épocas sobre el dataset SceneFlow completo.
Stackedhourglass (dataset KITTI)	PSMNet tras hacer fine-tune con 1000 épocas sobre la red entrenada con 10 épocas sobre el modelo Stacked-Hourglass utilizando el dataset del KITTI.



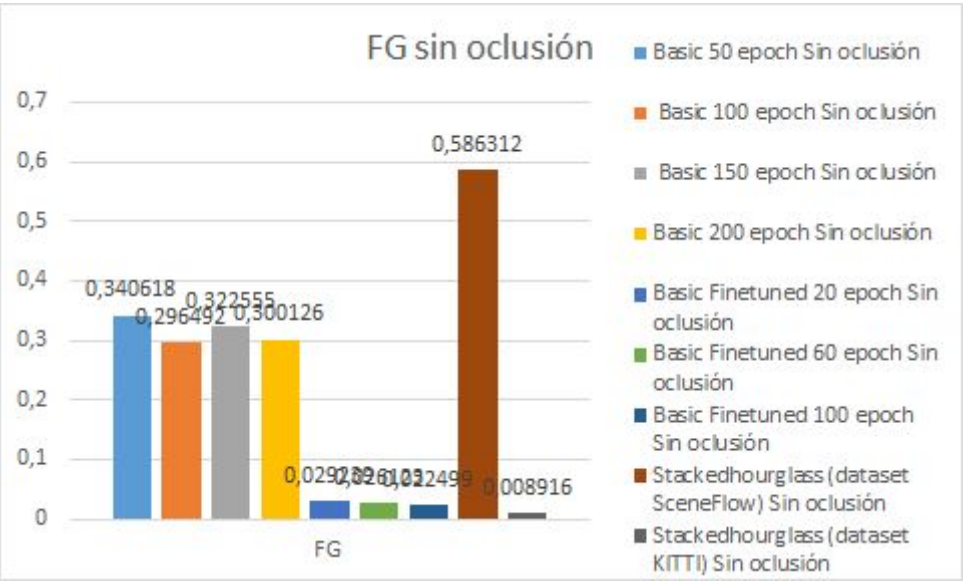
Resultados generales con oclusión. En el eje x se disponen las diferentes redes utilizadas y en el eje el error medio de predicción de disparidad con oclusión.



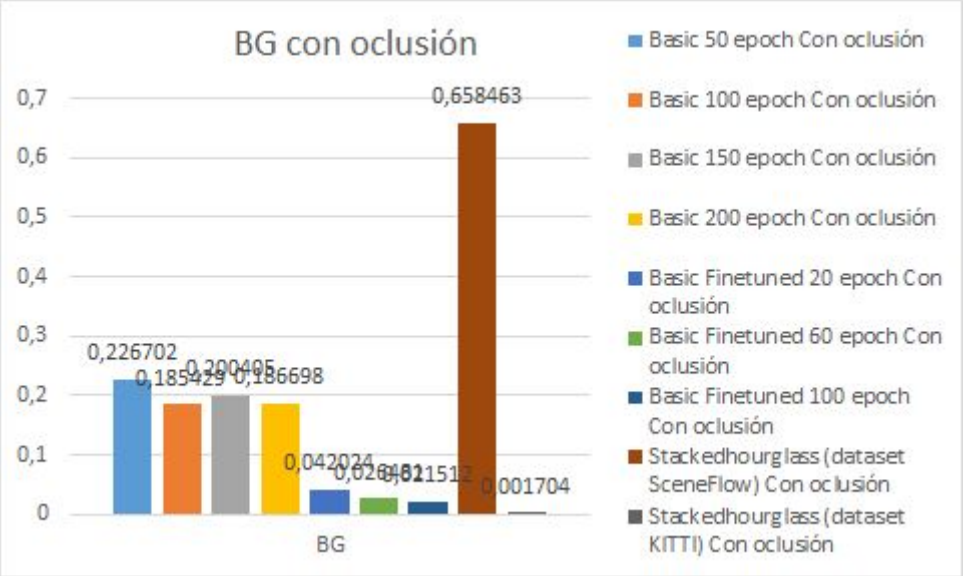
Resultados generales sin oclusión. En el eje x se disponen las diferentes redes utilizadas y en el eje el error medio de predicción de disparidad sin oclusión.



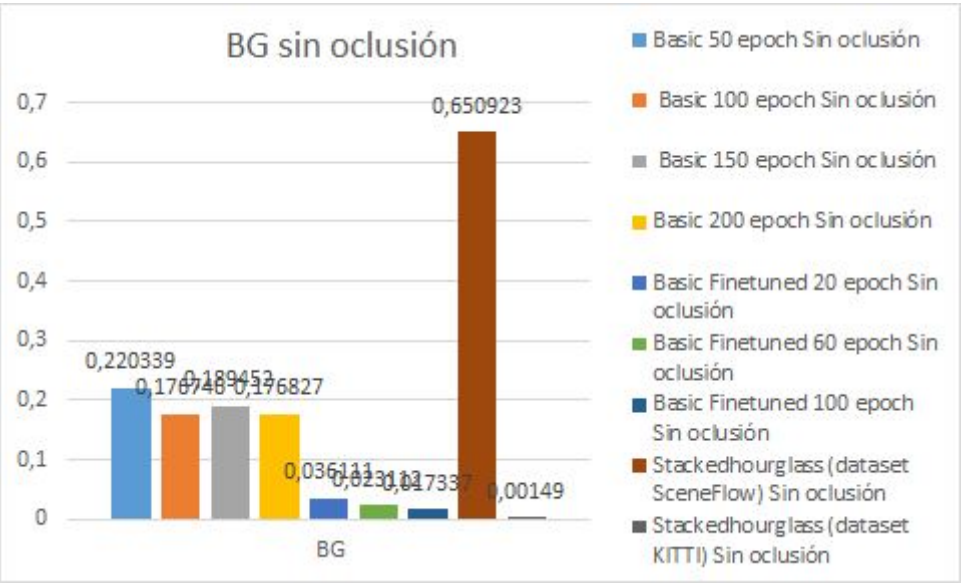
Resultados para foreground con oclusión. En el eje x se disponen las diferentes redes utilizadas y en el eje el error medio de predicción de disparidad con oclusión en primer plano de la imagen.



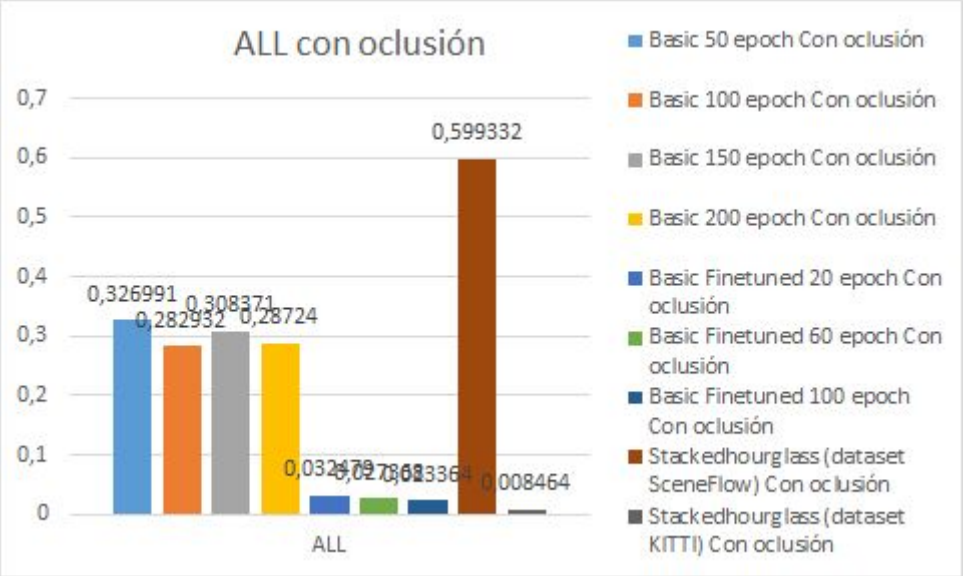
Resultados para foreground sin oclusión. En el eje x se disponen las diferentes redes utilizadas y en el eje el error medio de predicción de disparidad sin oclusión en primer plano de la imagen.



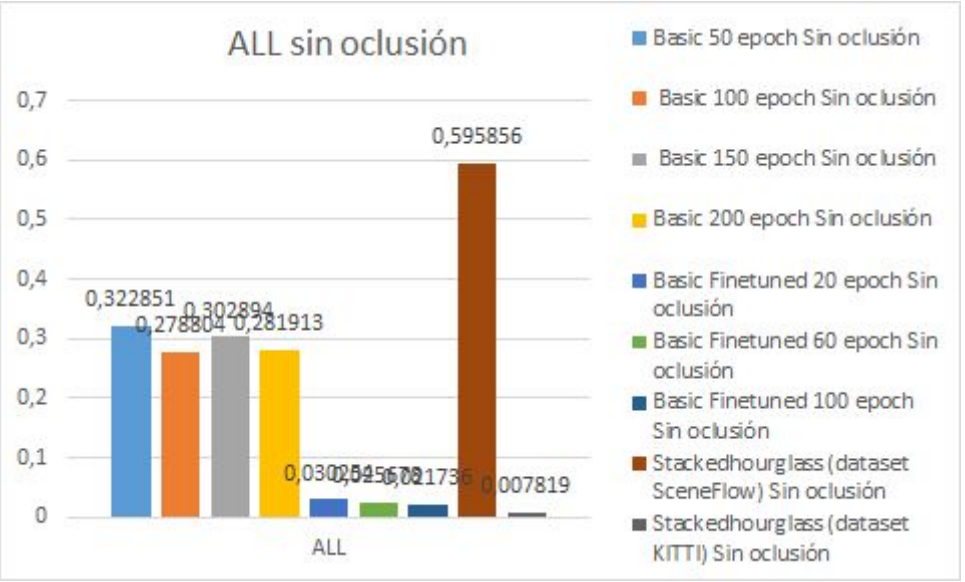
Resultados para background con oclusión. En el eje x se disponen las diferentes redes utilizadas y en el eje el error medio de predicción de disparidad con oclusión en el fondo de la imagen.



Resultados para background sin oclusión. En el eje x se disponen las diferentes redes utilizadas y en el eje el error medio de predicción de disparidad sin oclusión en el fondo de la imagen.



Resultados para toda la imagen con oclusión. En el eje x se disponen las diferentes redes utilizadas y en el eje el error medio de predicción de disparidad con oclusión toda la imagen.



Resultados para toda la imagen sin oclusión. En el eje x se disponen las diferentes redes utilizadas y en el eje el error medio de predicción de disparidad sin oclusión toda la imagen.

Teniendo en cuenta los resultados obtenidos con las redes PSMNet, tanto para el modelo StackHourglass como para el modelo Básico, se puede observar cómo hay una gran diferencia entre las redes sobre las que se ha realizado un fine-tune con el dataset del KITTI, mejorando los resultados sobre el mismo significativamente. Por otro lado, dado las limitaciones surgidas, no ha sido posible entrenar en igualdad de condiciones la red entrenada Básica y la red pre-entrenada StackHourglass. Por ello, comparar ambas redes

de una forma imparcial es imposible, sin embargo, se puede comprobar que la red Básica presenta un error promedio de disparidad algo superior a los de StackHourglass, debido a que no se ha realizado tantas épocas sobre el dataset KITTI y probablemente también a que la red es más simple. A pesar de haber realizado un mayor número de épocas, como el dataset de entrenamiento sobre SceneFlow no ha sido el mismo, los resultados no habrían sido comparables de igual modo. Sin embargo, se aporta algo nuevo, que son resultados de la red Básica con los diversos entrenamientos y fine-tune sobre KITTI teniendo en cuenta nuestras limitaciones. Además se proporcionan redes PSMNet con modelo Básico ya entrenadas. Estas redes están entrenadas con un mayor número de épocas en SceneFlow, en comparación con la red StackHourglass pre-entrenada y publicada en su repositorio, por lo que se tiene una red no entrenada sobre el KITTI con mejores resultados cuando posteriormente se infiere sobre un dataset relacionado con la conducción, como es el KITTI.

Teniendo en cuenta los resultados con y sin oclusión, se puede comprobar que sin oclusión el error disminuye, algo totalmente lógico, ya que no tiene en cuenta las zonas en las que una cámara del par estéreo no es capaz de visualizar y la otra sí. En cuanto a la diferencia entre el error entre el fondo de la imágen y el primer plano, en este último es menor en todos los casos. Ambos comportamientos por todas las redes publicadas en el KITTI.

Por otro lado, los tiempos de inferencia en cada modelo son diferentes. Analizar estos tiempos puede ser interesante para comprobar cómo de rápido son y si es viable utilizarlos en un caso real. Los resultados son:

- Con el modelo StackedHourglass se consigue un tiempo total de 111.69 segundos segundos, lo que quiere decir que ha tardado una media de 0.5545 segundos en cada imagen. Implicando una tasa de frames de 1.8034 FPS.

- Con el modelo Básico se consigue un tiempo total de 89.72 segundos segundos, lo que quiere decir que ha tardado una media de 0.4486 segundos en cada imagen. Implicando una tasa de frames de 2,2291 FPS.

Con estos resultados se puede concluir que el modelo básico mejora respecto del modelo Stacked Hourglass, siendo un 1,2448 más lento que el modelo básico, es decir, el modelo Básico es un 24 % más rápido. Por esto, se puede concluir que el modelo Básico es superior en cuanto a velocidad de inferencia.

En cuanto a la tasa de frames, como se puede observar, ninguno de los dos modelos son viables sobre nuestro entorno de pruebas para casos reales, necesitando un mínimo de 16 FPS. 1.8034 y 2.2291 están muy distanciados del mínimo necesario.

A continuación se adjunta una tabla resumen con los resultados de los tiempo obtenidos durante la inferencia con los modelos de la reproducción.

Modelo	Tiempo total (segundos)	Tiempo por imágen	Frames por segundo (FPS)
Stacked-Hourglass	111.69	0.5545	1.8034
Basic	89.72	0.4486	2,2291

Finalmente se presenta los resultados publicados por el KITTI y se compara con los resultados obtenidos previamente en nuestra reproducción. En este caso sólo se tiene en cuenta la red Básica y StackHourglass reproducidas con mejor los resultados. Estas corresponden a las que fueron aplicadas un fine-tune de mayor épocas.

Resultados publicados en KITTI sobre PSMNet y reproducidos			
Modelo	Tiempo de inferencia por imagen (segundos)	Ambiente	Error promedio de inferencia sobre la imágen completa (con oclusión)
Stacked-Hourglass (Publicado en KITTI)	0.41	Nvidia GTX Titan Xp	0.0232
Stacked-Hourglass (Reproducido)	0.5545	Nvidia GTX 1080	0.008464
Basic (Reproducido)	0.4486	Nvidia GTX 1080	0.023364

Como se puede observar, los resultados de disparidad reproducidos sobre StackHourglass son mucho mejores que los publicados en el KITTI, igualando prácticamente la red Básica reproducida a la red StackHourglass publicada. Superar sus resultados sólo sería posible si la red pre-entrenada usada en la reproducción y la usada por KITTI para el benchmark son las mismas en cuanto arquitectura pero no han sido entrenadas de igual forma. Por ello el benchmark realizado por KITTI debe tener alguna limitación durante el entrenamiento. Teniendo en cuenta los resultados obtenidos tras realizar fine-tune durante la reproducción y la gran mejora que esto produce, se tiene la hipótesis de que para publicar resultados en el KITTI, las redes no pueden estar ajustadas o entrenadas sobre el mismo dataset del benchmark, lo cual parece coherente, a pesar que se utilice para estos casos un subconjunto del conjunto total de imágenes. Utilizar el mismo dataset para entrenamiento y testeo no es aconsejable, pues los resultados no reflejan la realidad. Limitando el entrenamiento de esta forma los benchmarks deberían ser más objetivos.

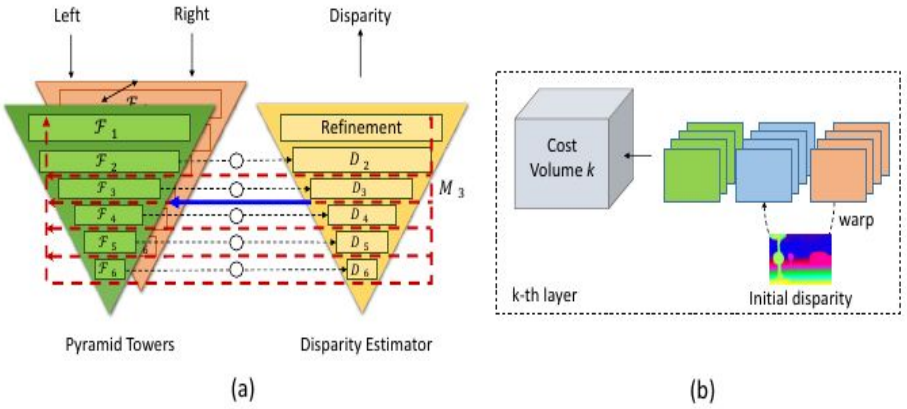
En cuanto a los tiempos de inferencia, los tiempos publicados en el KITTI son muy superiores debido al ambiente utilizado, el cual es mucho más potente por su mayor

capacidad de cómputo. Atendiendo a la velocidad de inferencia, se obtiene una tasa de frames de aproximadamente 2.44 FPS, no pudiendo ser utilizada en un caso real, considerando 16 FPS una tasa aceptable.

4.4.3.- MADNet

Puesto 135 KITTI-Stereo (durante el desarrollo del trabajo).

MADnet es otra red orientada al cálculo de la disparidad, en este caso, implementada sobre TensorFlow. Como característica fundamental, es una red preparada para el aprendizaje on-line, es decir, es capaz de aprender dinámicamente a través de un Stream de pares de imágenes continuo teniendo en cuenta lo aprendido anteriormente, teóricamente proporcionando un mejor resultado que entrenamientos y fine tunes batch clásicos cuando se enfrentan a dataset muy diferentes a los que han sido entrenados. Esto requiere que la red aprenda de forma no supervisada e infiera rápidamente un par de imágenes para que pueda volver a realizarlo con el siguiente par de imágenes a una tasa de aproximadamente 15 FPS. A pesar de ello, esta característica no era importante para nosotros, pues el objetivo principal era poder utilizarla a través de un batch de imágenes y no a través de las imágenes estéreo obtenidas por un coche en la vida real. Otra característica de la red es la capacidad de entrenar por módulos, es decir, por porciones red red, de ahí su nombre MADnet (Modularly ADaptive Network).



Descripción esquemática de la arquitectura MADnet propuesta (a), cada círculo entre una F_K y la D_K correspondiente representa una deformación y una capa de correlación (b). Cada par (F_l, D_l) compone un módulo M_l, adaptable de forma independiente por medio de la MAD (flecha azul) mucho más rápido en comparación con la realización de la parte posterior completa (flecha roja)[25]

Una vez reproducido el entrenamiento, inferencia y benchmarks con PSMNet, el objetivo es trabajar con MADNet. En este caso, este trabajo consiste en reproducir el trabajo realizado por sus desarrolladores de forma similar a PSMNet. Sin embargo en este caso no se realiza el entrenamiento para ahorrar tiempo, visto que es un proceso bastante largo y que ya hemos sido capaces de realizar con PSMNet.

[25] https://github.com/CVLAB-Unibo/Real-time-self-adaptive-deep-stereo

MADNet es una red que tiene la capacidad de ser entrenada de forma no supervisada cuando infiere. En nuestro caso este no es nuestro objetivo, por lo que la etapa de inferencia está configurada y ejecutada para que funcione como el resto de redes utilizadas.

El entrenamiento es realizado con un red pre-entrenada sobre el KITTI, con un número épocas no indicadas en su paper ni en su repositorio, lugar donde está disponible.

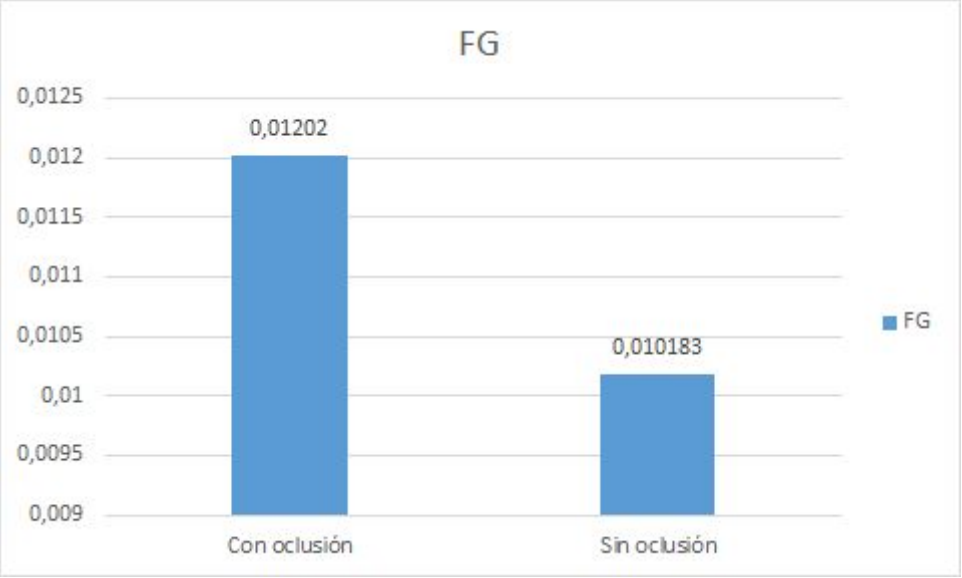
Los resultados de error promedio de disparidad de la inferencia tras ejecutar la herramienta development kit de KITTI para realizar el benchmark son:

Leyenda	
FG	Porcentaje de valores atípicos promediados solo en regiones de primer plano
BG	Porcentaje de valores atípicos promediados solo en regiones de segundo plano
ALL	Porcentaje de valores atípicos promediados sobre todos los píxeles del groundtruth (precisión de la clasificación del conjunto de entrenamiento para técnicas de machine learning).

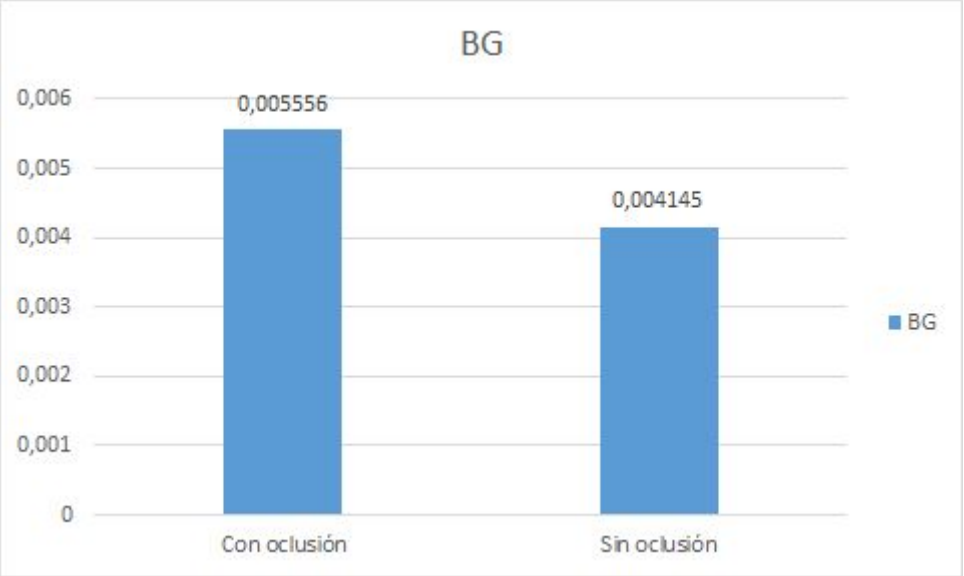
Se tiene en cuenta la oclusión	FG	BG	ALL
Sí	0.012020	0.005556	0.011053
No	0.010183	0.004145	0.009291

A continuación se muestra una gráfica de la comparación para MADnet de los resultados obtenidos con oclusión y sin oclusión.

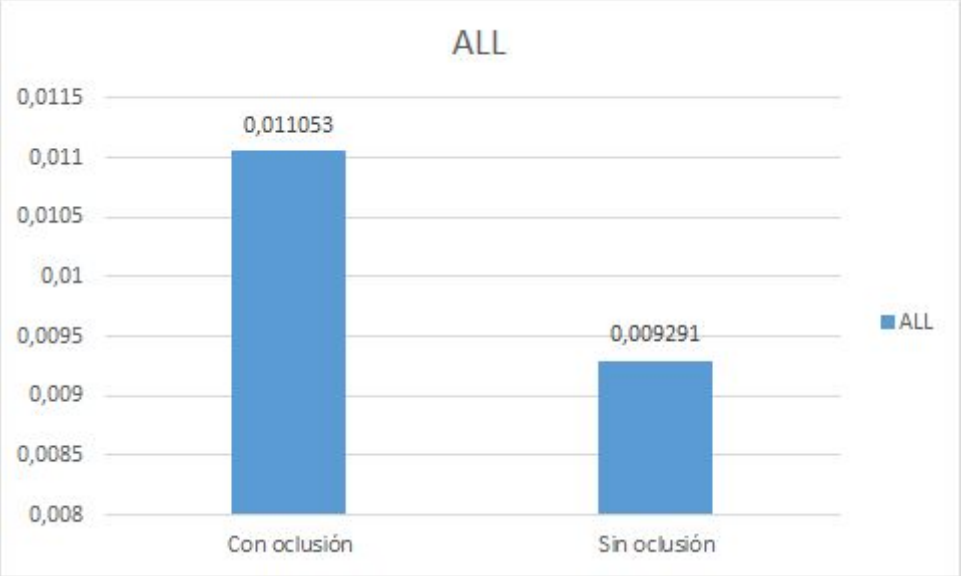
Resultados generales. En el eje x se disponen los resultados según la zona de la imagen a la que correspondan, ya sea foreground (FG), background (BG) o la imagen completa (ALL). En el eje y se dispone el error medio de predicción de disparidad.



Resultados para foreground con oclusión y sin oclusión. En el eje x se disponen la red con oclusión y sin oclusión y en el eje y el error medio de predicción de disparidad en primer plano de la imagen.



Resultados para background con oclusión y sin oclusión. En el eje x se disponen la red con oclusión y sin oclusión y en el eje y el error medio de predicción de disparidad en el fondo de la imagen.



Resultados para toda la imagen con oclusión y sin oclusión. En el eje x se disponen la red con oclusión y sin oclusión y en el eje y el error medio de predicción de disparidad sin oclusión toda la imagen.

Como ocurre con PSMNet, el error promedio de disparidad sigue el mismo patrón para todas las redes cuando analizamos la inferencia con y sin oclusión, siendo menor sin oclusión. También ocurre lo mismo si analizamos el error en el fondo de la imagen, primer plano o imagen completa, siendo mayor el error en el fondo.

En cuanto a los tiempos inferencia los resultados fueron los siguientes:

- El tiempo total que ha tardado en la inferencia de las doscientas imágenes del dataset de evaluación del KITTI ha sido de 9.46 segundos lo que quiere decir que ha tardado una media de 0.04745 segundos en cada imagen. Dicho esto, tiene una tasa de frames por segundo de 21.035 FPS.

Modelo	Tiempo total (segundos)	Tiempo por imágen (segundos)	Frames por segundo (FPS)
MADNet	9.46	0.04754	21,035

Esto demuestra que es una red muy rápida en cuanto a tiempos de inferencia se refiere. Esto es debido a que como requisito debe ser capaz de inferir y entrenar por cada imagen de entrada. Si nos centramos en la tasa de frames por segundo, se puede comprobar que sí es una red que podría ser utilizada en condiciones reales, pues supera los 16 FPS que consideramos mínimos para que el vehículo pueda reaccionar con antelación a esta información.

A continuación se comparan todos los resultados reproducidos con los resultados publicados en el KITTI.

Resultados publicados en KITTI sobre MADNet y reproducidos			
Modelo	Tiempo de inferencia por imagen (segundos)	Ambiente	Error promedio de inferencia sobre la imágen completa (con oclusión)
MADNet (Publicado)	0.02	GPU (no especificada) - 2.5 GHz	0.0466
MADNet (Reproducido)	0.04754	Nvidia GTX 1080	0.011053

De nuevo, teniendo en cuenta el error promedio de disparidad, los datos reproducidos son mejores que los publicados en el KITTI. Por esto mantenemos la hipótesis de que la red pre-entrenada no es la misma que ha sido utilizada por KITTI para el benchmark, en cuanto a parámetros de entrenamiento, como puede ser no poder entrenar sobre el KITTI.

En cuanto a los tiempos, la GPU utilizada para inferir por sus desarrolladores no está especificada, pero si observamos que sus tiempos son mejores a los reproducidos. También podría usarse en un caso real, llegando a los 50 FPS, una tasa de frames por segundo bastante aceptable.

4.5.- Portabilidad y optimización del tiempo de inferencia

Ya que el objetivo final de estas redes es ser usadas en un vehículo autónomo, nos parece interesante tanto reducir las dependencias de los frameworks para que las redes puedan ser empleadas en distintos elementos hardware sin tener que adaptar el entorno para cada framework que se utilice,como tratar de optimizarlas con OpenVINO para reducir los tiempos de inferencia en detección o clasificación, ya que el tiempo es un factor crítico en este ambiente y cuanto más rápido se detecten y clasifiquen los objetos, más tiempo de reacción tendrá el vehículo y mejor se comportará.

Para hacer esto, hay que generar un modelo optimizado a través de una serie de pasos que dependen del framework en el que fueron implementada la red. Esto es porque cada framework de desarrollo tiene una interpretación interna del modelo distinta, por lo que los modelos exportados no son compatibles entre ellos. Este modelo final siempre constará de un archivo XML que define la arquitectura de la red y de un binario que define los pesos de la red. A partir de dicho modelo podemos ya utilizar los plugins de OpenVINO para añadirlo al motor de inferencia y obtener una velocidad de inferencia más rápida.

Logo de OpenVINO[26]

El kit de herramientas OpenVINO[27] está diseñado para inferencia visual abierta y optimización de la red neuronal.

Este kit es gratuito, se puede descargar y ayuda a los desarrolladores a acelerar el desarrollo de la visión informática de alto rendimiento y el deep learning en las aplicaciones de visión. Permite el aprendizaje profundo en los aceleradores de hardware y una ejecución heterogénea y sencilla en varios tipos de plataformas Intel. Incluye el Intel Deep Learning Deployment Toolkit con un optimizador de modelos y un motor de inferencia, así como bibliotecas de visión informática optimizadas y funciones para OpenCV y OpenVX.

OpenVINO es un conjunto de herramientas pensado para diseñar soluciones de visión por computador con las mejores prestaciones y menores tiempos de desarrollo posibles.

[26] *https://01.org/openvinotoolkit*
[27] https://software.intel.com/en-us/openvino-toolkit

Permite un acceso muy sencillo a todo el conjunto de opciones hardware de Intel para mejorar el rendimiento, reducir el consumo de energía y maximizar la utilización del hardware, con la intención de poder hacer más con menos recursos y abrir nuevas posibilidades de diseño:

- Inferencia de última generación de Deep Learning basada en Redes Neuronales
- Ejecución distribuida con APIS compartidas sobre sistemas heterogéneos de Intel como:
    - CPUs, GPUs, NCSs (Movidius Neural Compute Stick 1 y 2) y FPGAs.
    - Aceleración de los tiempos de desarrollo hasta el despliegue final gracias a las bibliotecas de funciones y núcleos pre-optimizados.
    - Llamadas optimizadas a OpenCV y OpenVX.
    - Utilización del conjunto de herramientas Intel® Deep Learning Deployment Toolkit.
    - Aceleración de los resultados de la inferencia del aprendizaje profundo.

El conjunto de herramientas de Intel Deep Learning Deployment está formado por:

- Motor de inferencia con plugins para distintas plataformas hardware de manera individual (CPU, GPU, VPU, y FPGA), pero también formando sistemas heterogéneos con cualquier combinación de ellas.
- Optimizador de modelos capaz de tomar como entrada descripciones de entornos de deep learning como Caffe y TensorFlow y generar representaciones intermedias (IR) sobre las que poder trabajar.

Como OpenVINO permite adquirir la portabilidad y la optimización en los tiempos de inferencia que se buscan en este trabajo, se aspira a realizarlo con las cuatro redes disponibles en el KITTI con las que trabajamos. Aunque todavía OpenVINO sólo es capaz de optimizar directamente modelos de algunos frameworks como Caffe y TensorFlow, es capaz de solucionar esto y poder aceptar redes provenientes de otros entornos a través de ONNX.

ONNX es un formato abierto que pretende ser un estándar para permitir, con algunas limitaciones, exportar redes de unos frameworks de desarrollo a otros, permitiendo a los desarrolladores de Inteligencia Artificial poder exportar e importar sus modelos más fácilmente entre herramientas de Deep Learning.

4.5.1- PSMNET

El primer trabajo con OpenVINO se realiza con PSMNet. Para poder utilizarlo en OpenVINO se necesita pasar la red entrenada a un formato de red llamado ONNX, como se ha explicado anteriormente. Esto es necesario porque OpenVINO no es compatible directamente con redes entrenadas en PyTorch, pero sí lo es con redes ONNX. A la hora de realizar la transformación de la red a ONNX para poder ser optimizada posteriormente en OpenVINO, no es posible. Tras investigar el uso de este formato de red, la hipótesis inicial es que se debe a la incompatibilidad en alguna de las capas de PSMNet, que no estaba soportada por ONNX.

```
model = basic(192)
model = nn.DataParallel(model, device_ids=[0])
model.cuda()
checkpoint = torch.load('checkpoint200.tar')
model.load_state_dict(checkpoint['state_dict'])
epoch = checkpoint['epoch']
loss = checkpoint['train_loss']

print('Number of model parameters: {}'.format(sum([p.data.nelement() for p in model.parameters()])))

input = torch.randn(2, 3, 384, 1248, device='cuda')
#384, 1248
torch.onnx.export(model, input, "tmp/onnx_basic.onnx", export_params=True, verbose=True)
```

Script creado para intentar generar la red ONNX de PSMNet

Para asegurar que el script estaba correctamente desarrollado y que el error no residía en el código creado, se desarrolla un script similar para exportar una red de clasificación pre-entrenada en formato ONNX. En este caso se usa una red VGG-16, con vertida con éxito del formato PyTorch al formato ONNX.

```
model = models.vgg16(pretrained=True)
torch.save(model.state_dict(), "tmp/trainedVGG16.pt")


model =  models.vgg16(pretrained=False)
model.load_state_dict(torch.load( "tmp/trainedVGG16.pt"))
model.eval()

#print('Number of model parameters: {}'.format(sum([p.data.nelement() for p in model.parameters()])))

input = torch.randn(10, 3, 224, 244)
torch.onnx.export(model, input, "tmp/VGG16.onnx", export_params=True, verbose=True)
```

Script creado para generar la red ONNX de una red de clasificación de objetos ya entrenada (VGG16), el cual sí funcionó.

Debido a esto, se puede asegurar que el fallo al pasar la red a ONNX se debe a que algunas de las capas empleadas por PSMNET aún no están soportadas en ONNX.

4.5.2- MADNet

A continuación se trabaja con MADNet. En este caso la red está implementada con Tensorflow, el cual se puede pasar a OpenVINO directamente sin necesidad de pasarlo a ONNX. Esto implica que no hay que realizar ninguna transformación a un formato de red intermedia. Para este proceso hay que utilizar una utilidad que OpenVINO proporciona en Python o en C, la cual ejecutándola y añadiendo como parámetros la red exportada de Tensorflow en formato meta-grafo sin congelar, junto con algunos otros parámetros, nos genera la red optimizada lista para usar en OpenVINO. Sin embargo, utilizar esta utilidad con MADNet no es posible.

Dependiendo de el formato de la red introducido en la utilidad para transformar la red, las causas que lo imposibilitan son diferentes. En el caso de que el formato sea meta-grafo, sin congelar la red, OpenVINO no es capaz de detectar ciertas variables de TensorFlow que sí están inicializadas.

En el módulo de entrenamiento *train.py* se puede observar cómo se inicializan las variables que posteriormente OpenVINO no detecta.

```
with tf.Session(config=tf.ConfigProto(gpu_options=gpu_options)) as sess:
        #init stuff
        sess.run([tf.global_variables_initializer(),tf.local_variables_initializer()])

        #restore disparity inference weights
        restored,step_eval = weights_utils.check_for_weights_or_restore_them(args.output,sess,initial_weights=args.weights)
```

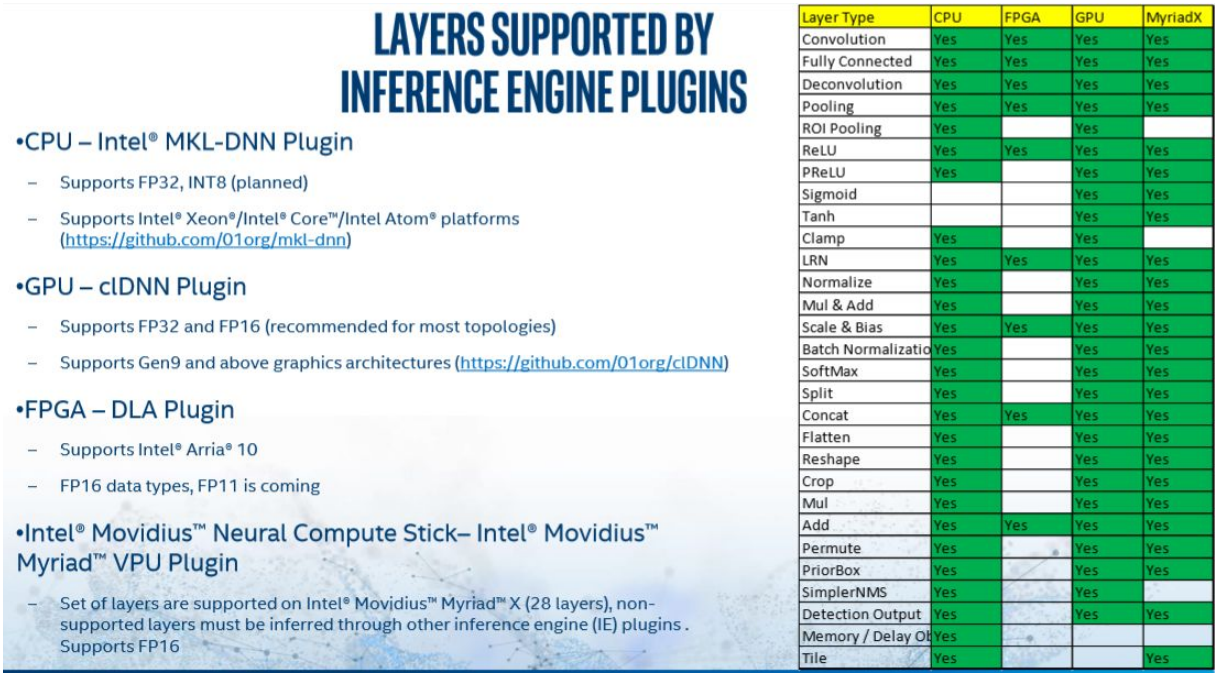
Código perteneciente al script de entrenamiento de MADNet

Por otro lado, en el caso de que el formato sea una red congelada, congelando únicamente el nodo de salida de la red denotado como *“model/resize_image_with_crop_or_pad/control_dependency_3”*, no es posible debido a que no tiene definido placeholders y OpenVINO no es capaz de detectarlos. Esta causa es debida a la propia naturaleza live-inference de la arquitectura de la red explicada previamente.

Para poder visualizar los nodos de la red y obtener el nombre de los nodos salida se utiliza la herramienta web TensorBoard, la cual muestra en el navegador la representación del grafo de la red.

4.5.3- SegStereo y CRL

SegStereo y CRL están en Caffe, que es soportado directamente por OpenVINO, sin necesidad de tener que hacer pasos intermedios. Sin embargo, a pesar de seguir los pasos para transformar redes de Caffe a OpenVINO, con el fin de poder ver la mejora que se obtenía, no es posible realizar la conversión debido a un error en las capas soportadas. OpenVINO solo soporta 27 capas definidas en Caffe, la falta de compatibilidad de algunas de las capas empleadas en la descripción de la topología de las redes SegStereo y CRL, como por ejemplo las capas de tipo: *correlation_1d*, *constant*, o *silence*, hacen que no sea posible el portarlas a OpenVINO.



Layer Type	CPU	FPGA	GPU	MyriadX
Convolution	Yes	Yes	Yes	Yes
Fully Connected	Yes	Yes	Yes	Yes
Deconvolution	Yes	Yes	Yes	Yes
Pooling	Yes	Yes	Yes	Yes
ROI Pooling	Yes		Yes	
ReLU	Yes	Yes	Yes	Yes
PReLU	Yes		Yes	Yes
Sigmoid			Yes	Yes
Tanh			Yes	Yes
Clamp	Yes		Yes	
LRN	Yes	Yes	Yes	Yes
Normalize	Yes		Yes	Yes
Mul & Add	Yes		Yes	Yes
Scale & Bias	Yes	Yes	Yes	Yes
Batch Normalizatio	Yes		Yes	Yes
SoftMax	Yes		Yes	Yes
Split	Yes		Yes	Yes
Concat	Yes	Yes	Yes	Yes
Flatten	Yes		Yes	Yes
Reshape	Yes		Yes	Yes
Crop	Yes		Yes	Yes
Mul	Yes		Yes	Yes
Add	Yes	Yes	Yes	Yes
Permute	Yes		Yes	Yes
PriorBox	Yes		Yes	Yes
SimplerNMS	Yes		Yes	
Detection Output	Yes		Yes	Yes
Memory / Delay Ob	Yes			
Tile	Yes			Yes

Imagen obtenida de una presentación de Intel. En ella se muestra cuáles son las capas soportadas para la transformación a OpenVINO dependiendo del hardware al que vaya a estar destinada la red.

4.5.4- Redes pre-entrenadas de clasificación de objetos

El objetivo de este apartado es comprobar que OpenVINO es una solución real para la portabilidad y mejora en el tiempo de inferencia de redes de clasificación de objetos definidas sobre diferentes frameworks.

Para ello se convierten las distintas redes a formato OpenVINO, obteniendo el XML y el binario. Este paso se realiza con una red AlexNet y otra VGG-16 sobre dos frameworks distintos, Caffe, que puede ser transformada directamente a OpenVINO, y Keras, que tiene que ser transformada primero a ONNX y posteriormente realizar la conversión a OpenVINO.

La transformación se realiza para cada red dos veces, una para CPU y otra para Movidius, un dispositivo de bajo consumo de Intel, creado especialmente para realizar una gran cantidad de cálculos en un tiempo bajo, lo cual es ideal para un vehículo autónomo. Para poder realizar la conversión a CPU, es necesario que se realice una optimización del modelo para que trabaje con operaciones de FP32. En cambio, para poder ser empleado en Movidius, se ha de realizar la optimización del modelo para que trabaje con operaciones de FP16. Esto se indica en un argumento al transformarlo.

Mostramos como ejemplo la transformación a OpenVINO de una red AlexNet que estaba en Keras y ha sido pasada a ONNX.

El comando para la transformación a CPU con FP32 es el siguiente: *python3 /opt/intel/compputer_vision_sdk/deplyment_tools/model_optimizer/mo.py --input_model alexnet.onnx*

Generando la salida siguiente:

```
[ SUCCESS ] Generated IR model.
[ SUCCESS ] XML file: /home/usuario_test/ONNX/./alexnet.xml
[ SUCCESS ] BIN file: /home/usuario_test/ONNX/./alexnet.bin
[ SUCCESS ] Total execution time: 1.73 seconds.
```

Conversión de AlexNet en Keras a OpenVINO FP32

EL Comando para la transformación para el uso de Movidius con FP16: *python3 /opt/intel/computer_vision_sdk/deployment_tools/model_optimizer/mo.py --input_model alexnet.onnx --data_type FP16*

Generando la salida siguiente:

```
[ SUCCESS ] Generated IR model.
[ SUCCESS ] XML file: /home/usuario_test/ONNX/./alexnet.xml
[ SUCCESS ] BIN file: /home/usuario_test/ONNX/./alexnet.bin
[ SUCCESS ] Total execution time: 2.08 seconds.
```

Conversión de AlexNet en Keras a OpenVINO FP16

Una vez obtenido el modelo optimizado y dado que la intención es ver si se producen mejoras a la hora de hacer inferencia en la clasificación de imágenes, todas las redes han clasificado la misma imagen de un coche.

Imagen utilizada para el testeo de las redes con OpenVINO[28]

A continuación se muestran los resultados de los tiempos de inferencia al clasificar esta imagen en las redes de AlexNet y VGG-16 en los framework de Caffe y Keras, usando solo la CPU sin la optimización que te produce OpenVINO, y usando las optimizaciones de OpenVINO tanto para CPU, como para el dispositivo Movidius.

[28] https://www.elmundo.es/motor/2018/07/30/5b5ef0ee22601d5b158b459e.html

Tiempos de inferencia(ms)				
Framework	Caffe		Keras	
Red	AlexNet	VGG-16	AlexNet	VGG-16
Para CPU sin Optimizar	350	2970	670	1330
Optimizada para CPU	31,5	212,9	37,8	211
Optimizada para Movidius	86,2	717,3	79	776,7

Los valores de los tiempos están en milisegundos.

En esta tabla se puede observar los siguientes resultados para una misma red implementada en un framework concreto:

- Para la red AlexNet para Caffe, se consigue entorno a un 1100% de mejora al optimizar el modelo para CPU, y una mejora de aproximadamente un 400% al realizar la optimización para Movidius. Aunque con la optimización para CPU se consigue un 270% de mejora con respecto a la de Movidius.

- Para la red VGG-16 para Caffe se consigue entorno a un 1400% de mejora al optimizar el modelo para CPU, y una mejora de aproximadamente un 415% al realizar la optimización para Movidius. Aunque con la optimización para CPU se consigue un 335% de mejora con respecto a la de Movidius.

- Para la red AlexNet para Keras, se consigue entorno a un 1770% de mejora al optimizar el modelo para CPU, y una mejora de aproximadamente un 850% al realizar la optimización para Movidius. Aunque con la optimización para CPU se consigue un 210% de mejora con respecto a la de Movidius.

- Para la red VGG-16 para Keras se consigue más de un 630% de mejora al optimizar el modelo para CPU, y una mejora de aproximadamente un 170% al realizar la optimización para Movidius. Aunque con la optimización para CPU se consigue un 370% de mejora con respecto a la de Movidius.

Hay que tener en cuenta que Movidius es un dispositivo de bajo consumo y que aun así ha conseguido reducir en gran medida los tiempos originales, a pesar de no reducir los tiempos más que la optimización para CPU.

Por otro lado, se pueden observar los siguientes resultados atendiendo a las distintas optimizaciones:

- Para CPU sin optimizar, los tiempos en las redes AlexNet son bastante menores que los tiempos de las redes VGG-16, ya que la topología de la VGG-16 es mucho más compleja. Por otro lado llama la atención que en Caffe la red AlexNet tarda aproximadamente la mitad de tiempo en clasificar que su homóloga en Keras, sin embargo, la red VGG-16 tarda aproximadamente la mitad de tiempo en clasificar en Keras que en Caffe.

- Para la optimización con OpenVINO, tanto para CPU como para Movidius es muy llamativo que aunque las diferencias en inferencia de las redes en los distintos frameworks eran diferentes, al ser optimizadas, cada red pasa a tener unos tiempos de inferencia muy similares, independientemente del framework en el que hubiesen sido implementadas.

Cabe destacar que al medir los tiempos de inferencia para comprobar si hay una mejora, se ha comprobado que al realizar la optimización, no solo mejora el tiempo, sino la precisión de la clasificación, como puede observarse en este ejemplo de la VGG-16:

Probabilidades ordenadas de mayor a menor para VGG-16 para CPU sin optimizar:

1. *0.2419953 label 436: 'beach wagon, station wagon, wagon, estate car, beach waggon, station waggon, waggon',*
2. *0.1655973 label 817: 'sports car, sport car',*
3. *0.1615566 label 751: 'racer, race car, racing car',*
4. *0.1043116 label 717: 'pickup, pickup truck',*
5. *0.1040226 label 468: 'cab, hack, taxi, taxicab',*

Probabilidades ordenadas de mayor a menor para VGG-16 para CPU optimizada:

1. *0.3889190 label 751: ‘racer, race car, racing car’,*
2. *0.1899234 label 817: ‘sports car, sport car’,*
3. *0.1435201 label 436: 'beach wagon, station wagon, wagon, estate car, beach waggon, station waggon, waggon',*
4. *0.0781002 label 468: ‘cab, hack, taxi, taxicab’,*
5. *0.0765963 label 717: 'pickup, pickup truck',*

Se observa una mejora significativa de la precisión al clasificar la imagen, ya que la probabilidad de que sea un coche de carreras, o deportivo al 58% y una furgoneta al 14% describen mucho mejor la imagen del coche empleada que el ser una furgoneta al 24% y un coche de carreras o deportivo al 32%.

4.5.5- Red de clasificación de objetos con fine-tune

Previamente se han realizado pruebas con redes entrenadas sobre ImageNet, un dataset que distingue 1000 clases y no está focalizado exclusivamente en la conducción. Para acercar más el trabajo al mundo de la conducción, en este apartado se usa una red Inception-v3 (explicada anteriormente) la cual se adapta para clasificar los coches más robados en Estados Unidos. El proceso del entrenamiento y congelación se realiza empleando el framework Keras, para finalmente transformar la red a formato OpenVINO y poder realizar la clasificación y comparar los resultados. Dicho proceso es realizado bajo el mismo ambiente con OpenVINO utilizado con el resto de redes.

Aunque el primer paso es entrenar la red, en nuestro caso se ha decidido utilizar una red pre-entrenada sobre ImageNet para agilizar el proceso de entrenamiento, sobre la cual se realiza un fine-tune con un dataset propio, un conjunto de imágenes con los 9 modelos de los coches más robados en Estados Unidos. Con esto se consigue adaptar la red de propósito general a nuestro objetivo mucho más concreto.

Honda Civic (1997): coche más robado en Estados Unidos.
Imágenes pertenecientes al dataset para fine-tune.

El proceso para conseguir esto consiste en cargar la red con los pesos de ImageNet sin añadir las capas finales, ya que el número de clases que se van a inferir no son 1000 como ocurre en ImageNet, sino que el número de clases está definido por el nuevo dataset empleado para el fine-tune, en este caso 9 clases. Por lo tanto hay que añadir manualmente las últimas capas, con el número de salidas igual al nuevo número de clases que se van a identificar.

Tras adaptar la red para clasificar las nuevas clases en las que nos vamos a centrar, es necesario compilar la red antes de realizar el fine-tune y guardar la red con los nuevos pesos. Se ha decidido realizar este mismo proceso de fine-tune una vez más sobre toda la

red generada, pero con un ratio de aprendizaje menor, para no modificar drásticamente el resultado anterior y que mantenga un gran porcentaje de ajustes de pesos sobre ImageNet.

Antes de poder convertir la red a otro formato es necesario congelarla, creando un grafo cuyas variables de los nodos de salida se transforman en constantes. Dado que Keras utiliza como backend TensorFlow, es posible exportar la red a un formato congelado de red congelada TensorFlow, permitiéndose así transformarlo directamente al formato optimizado de OpenVINO sin realizar transformaciones previas como sucede con PyTorch, explicado anteriormente.

python3 /opt/intel/compputer_vision_sdk/deplyment_tools/model_optimizer/mo.py --input_model=students-workpsace/tf_model/top_layers.iv3.pb --input_shape=[1,299,299,3].

```
[ SUCCESS ] Generated IR model.
[ SUCCESS ] XML file: /home/alumno_local/INTEL-LABS/LAB-2/vmmr-car-theft-trainin
g-material-master/students-workpsace/transfor/top_layers.iv3.xml
[ SUCCESS ] BIN file: /home/alumno_local/INTEL-LABS/LAB-2/vmmr-car-theft-trainin
g-material-master/students-workpsace/transfor/top_layers.iv3.bin
[ SUCCESS ] Total execution time: 11.74 seconds.
```

Resultado exitoso de la transformación de Inception al modelo optimizado para OpenVINO

A partir del modelo optimizado para OpenVINO y el modelo sin optimizar se realiza el proceso de inferencia. Este proceso se realiza en CPU con y sin OpenVINO y además se utiliza el dispositivo Movidius de bajo consumo energético con OpenVINO.

Tiempos de inferencia(ms)			
Red	Para CPU sin Optimizar	Optimizada para CPU	Optimizada para Movidius
Inception-v3	1084	127	338

Los valores de los tiempos están en milisegundos.

En comparación con la inferencia sobre la misma red sin OpenVINO y con OpenVINO, se observa una mejora de tiempo de inferencia bastante grande. Mientras que sin OpenVINO en CPU obtenemos alrededor de 1,08 segundos, con OpenVINO obtenemos 0.127 segundos. Esto implica una mejora del 750% mejor, es decir, casi 9 veces más rápida. Incluso en Movidius la mejora ha sido de un 220%, es decir, más de 3 veces más rápida.

5.- Conclusiones y trabajo futuro

Tras la realización del trabajo fin de grado podemos concluir que:

El campo de las redes neuronales y su aplicación para la conducción es un campo bastante novedoso y actual. Hemos comprobado que todavía no hay estándares para todos los frameworks ni una compatibilidad entre ellos. La mayoría de las mejoras son realizadas por parte de la comunidad que mantiene cada uno de los frameworks, así como por los desarrolladores de cada red, y tardan un tiempo en estar disponibles. Este es el caso de algunas redes de Caffe, que necesitaban compilar una rama especial con más librerías y archivos, ya que la principal no es suficiente, o de OpenVINO, que solo soporta un número limitado de capas que permite transformar en cada framework.

Las redes de predicción de la disparidad son por ahora un tipo de red menos explotada, pero muy complejas. Es por ello que tanto el proceso de entrenamiento como el portarlas a OpenVINO para realizar la inferencia no es una tarea sencilla. El principal problema es el reducido número de implementaciones disponibles y utilizables a día de hoy. A pesar de ello, ha sido posible trabajar con ellas. Tras reproducir el trabajo de los desarrolladores, que consta del entrenamiento, inferencia y benchmarks de error promedio de disparidad y tiempos de la red, PSMNet StackHourglass ha resultado ser la que mejores resultados ha dado en cuanto a error promedio de disparidad, comparado con el resto. En cuanto al modelo Basic de PSMNet, hemos conseguido obtener una red entrenada que utiliza dicho modelo, la cual no está disponible en ningún repositorio. Dicha red mejora los resultados de error de disparidad cuando está entrenada únicamente sobre SceneFlow, si la comparamos con la red PSMnet StackHourglass entrenada únicamente con el mismo dataset. Además, la utilización de un modelo más simple, como es el caso de la misma, implica tener una red más rápida, como se ha podido comprobar en este caso, lo cual es bastante beneficioso cuando el dispositivo donde vamos a realizar el cómputo de la inferencia no tiene una potencia presumiblemente elevada. Esto es algo muy relevante en este campo, puesto que las limitaciones a la hora de llevar a la práctica este análisis de imágenes en estéreo para la conducción son muy claras. Tanto el tiempo de ejecución como la energía que requiere realizar los cómputos deben ser lo mínimo posibles. A pesar que el modelo Basic de PSMNet mejora al modelo StackHourglass, ninguno de los dos permiten, ni utilizando GPUs que no son de bajo consumo, como es la Nvidia GTX 1080, inferir a una tasa de frames por segundo aceptable, no llegando a 3 FPS con dicha GPU, por lo que usarlas en un vehículo para un caso real no es viable.

En lo referente a MADNet, cabe destacar la velocidad de inferencia, la cual es muy superior en comparación a las otras redes de disparidad como PSMNet debido a que es una red pensada para inferencia online, es decir, tiene que ser capaz de procesar un Stream de imágenes para inferir y entrenar de forma no supervisada a una tasa de frames por segundo que permita al vehículo obtener información sobre la profundidad de los elementos de las imágenes al instante para poder reaccionar a tiempo ante ellas. En este caso, MADNet sí puede utilizarse en un caso real, pues es capaz de inferir a más de 16 FPS. Sin embargo,

poder utilizarlo con una unidad de cómputo de bajo consumo para la inferencia sería lo ideal.

En cuanto a OpenVINO, se puede concluir que funciona bastante bien sobre redes de clasificación, pero sin embargo, aún no soporta la portabilidad de redes de disparidad para realizar inferencia con las mismas. Por una parte, está la razón de que no puede usarse directamente sobre redes implementadas en PyTorch, requiriendo así previamente transformarla a una red ONNX. Por otra parte, se ha comprobado que las redes de disparidad utilizadas no pueden ser transformadas a OpenVINO, ya que algunas de las capas empleadas en este tipo de redes aún no están definidas en esta tecnología, o por el simple hecho de no tener placeholders o porque OpenVINO no detecta la inicialización de ciertas variables, como ha ocurrido con MADNet para ambos casos.

Para finalizar, en cuanto a redes de clasificación, se ha comprobado que las redes AlexNet y VGG-16, tanto en Caffe como en Keras, al ser convertidas a OpenVINO producen resultados muy similares, concluyendo que no importa desde cuál de estos dos frameworks se adapte a la hora de portarlas a dicho framework. En ambos casos se reduce el tiempo de inferencia al ser pasadas a OpenVINO para CPU de manera muy notable, especialmente en la VGG-16, que es más compleja. Por otro lado, se ha visto que al transformarlas para poder ser utilizadas en un dispositivo de bajo consumo como el Movidius, la VGG-16, al ser más compleja y tener un mayor tiempo de inferencia, reduce en gran medida ese tiempo. Sin embargo, en la AlexNet, una red menos compleja y que tarda menos inicialmente, el tiempo de inferencia ha aumentado. En cualquier caso, los tiempos con Movidius son siempre superiores a los de CPU. Y en cuanto al trabajo con Inception-v3, tras haber sido posible realizar a partir de la red pre-entrenada un fine-tune adaptado a nuestras necesidades, haberse podido congelar la red y utilizarla en OpenVINO, se ha podido observar una mejora en cuanto velocidad de inferencia bastante notable. Por tanto se concluye que en todos los casos OpenVINO mejora la velocidad de inferencia en gran medida.

Como trabajo futuro se podría continuar trabajando con las nuevas redes que se vayan subiendo a la página de KITTI y mirando las mejoras de los frameworks y de herramientas como OpenVINO para ver si finalmente este tipo de redes son soportadas para poderlas implementar en dispositivos de bajo consumo como Movidius, para que puedan ser introducidos dentro de los elementos de un automóvil. En este trabajo hemos estado centrados en utilizar redes ya creadas por otras personas, reentrenarlas, hacer inferencia y compararlas, por lo que nos resultaría muy interesante llegar a definir nuestra propia red. Por otro lado, ya que el trabajo está orientado a vehículos autónomos, nos haría mucha ilusión probar algunas de estas redes en un automóvil preparado para este tipo de pruebas.

5.- Conclusions and future work

After completing the final degree project we can conclude that:

Neural networks field and its application for autonomous driving is still such a new and actual topic. We have checked that that there are no standards for all the frameworks or a compatibility between them yet. Most of the improvements are developed by the community which takes care of the frameworks, and the developers of the network themselves, so advances take some time to be available. This is the case of some Caffe networks, which need to compile a specific branch with more libraries and files because the main one is not enough, or OpenVINO, which only supports a limited number of layers from each framework.

Disparity prediction neural networks are a less used neural network type, however they are so complex. Because of that, training process and portability to OpenVINO to make the inference are not simple tasks. The main problem is the little number of available and usable implementations these days. Despite this, it has been able to work with them. After reproducing developers work, which is includes training, inference, average disparity error from the benchmarks and the times from the net, PSMNet StackHourglass has been the one with best results about the average disparity error compared with the others. Talking about the Basic PSMNet model, we have been able to obtain a trained net which uses this model, which is not available at any repository. This network improves the disparity results, if we only train with the SceneFlow dataset, in comparison with PSMNet StackHourglass only trained with SceneFlow. Furthemore, using a more simple model, as we used with PSMNet Basic, implies getting a faster network, as we have proved, which is really beneficial when the computational unit used is not really powerful. This fact is very relevant for this field due to the limitations during the inference for the pair of stereo images. The time and energy needed should be as reduced as possible. Even though the PSMNet Basic model is faster than his StackHourglass model, they cannot infer, not using a energy-saving GPU as the Nvidia GTX 1080, getting an acceptable rate of frames per second, being this rate lower than 3 FPS with this GPU. This implies that it could not be used in a real case.

Talking about MADNet, highlights the speed of inference, which is much bigger than other disparity networks like PSMNet, as it is an online inference network. This means it is able to process an image stream to infer, and train not being supervised with a FPS rate that makes the vehicle able to get information about image elements depth in a very short time to be able to react to them. In this case, MADNet can be used in a real case, as it can infer with more than 16 FPS. However, it would be desirable to use it with a low consumption device.

Talking about OpenVINO, it can be concluded that it works quite well on classification networks, but nevertheless, itt still does not support the portability of disparity networks to make inference with them. On the one hand, there is the reason that it can not be used directly on networks implemented in PyTorch, requiring previously transforming it into an ONNX networks. On the other hand, it has been verified that the disparity networks used can not be transformed to OpenVINO, since some of the layers used in this type of networks are

not yet defined in this technology, or simply because they do not have placeholders or because OpenVINO does not detect the initialization of certain variables, as has happened with MADNet in both cases.

To sum up, talking about classification networks, it has been proved that AlexNet and VGG-16, as well as in Caffe as in Keras, when converted to OpenVINO, produce very similar results, which means it does not matter from which framework we start to convert them to OpenVINO. In both cases, inference time gets decreased when converted to OpenVINO CPU in a very remarkable way, especially on VGG-16, which is more complex. On the other hand, it has been seen that converting them to be able to use them on a low consumption device like Movidius, VGG-16, as it is more complex and has a bigger inference time, reduces it in a very remarkable way. However, AlexNet, a less complex network and which uses less time to make the inference at the beginning, now has increased inference time. Anyway, Movidius timestamps are always bigger than CPU ones. Talking about our work with Inception-v3, after having been possible to make an adapted fine-tune from the pre-trained neural network, having been able to freeze the network, and use it in OpenVINO, it has been possible to notice an upgrade in inference speed in a very remarkable way. Finally, we can conclude that in all cases, OpenVINO increases inference speed to a large extent.

As future work, we could keep working with new networks that manage to appear in the KITTI ranking and looking for frameworks and tools improvements such as OpenVINO, in case disparity networks are finally supported, so we could test them at the low power device Movidius, so they can be used inside vehicles. In this work we have been focused in the use of networks which have been created by other persons, retrain them, make inference and compare them, so on the one hand we find really interesting to be able to create our own network. On the other hand, and considering that our works is focused in self driving vehicles, we would be really happy about testing these networks on a vehicle ready to perform such a task.

6.- Trabajo realizado por cada integrante del grupo

Álvaro de la Cruz Casado

Durante este trabajo he realizado las siguientes tareas:

1. Aprendizaje sobre inteligencia artificial, clasificación y detección de objetos a través de la documentación y vídeos disponibles en los cursos de inteligencia artificial de la Universidad Autónoma de Barcelona (UAB).

2. Preparación del entorno en las máquinas empleadas durante el desarrollo del TFG, con la instalación de las dependencias requeridas de forma que cualquier red descargada se pudiera ejecutar correctamente.

3. Trabajo con redes de detección de objetos simples:
    a. Configuración de las redes, leyendo los papers que se proporcionaban de cada una de ellas para observar su funcionamiento.
    b. Prueba de las redes, resolviendo cualquier error que pudiera surgir para poder entrenar y ejecutar la red y obtener resultados analizables.
    c. Entrenamiento de las redes con el CIFAR10.
    d. Obtención y comparativa de resultados en base a la imagen que se le pasaba a la red neuronal para ser analizada.

4. Preparación de una máquina virtual para pruebas en local sobre redes neuronales tanto para detección y clasificación como para inferencia.

5. Estudio de la estructura de las redes que emplean el framework Caffe, y los distintos archivos requeridos, como pueden ser el *solver*, el *deploy* o el *caffemodel*.

6. Pruebas con redes que emplean Caffe, tanto entrenamiento como posterior testeo:
    a. Compilación de Caffe-CPU para su uso con las redes de SegStereo y CRL, aunque al querer entrenar y ejecutar las redes propuestas mediante el comando *caffe train* y el comando *caffe test*, en cada una de ellas saltó un error de que una de las capas no se encontraba disponible, puesto que había sido diseñada para Caffe-GPU.
    b. Investigué también cómo convertir un *solver* de la red de SegStereo en un *deploy* que poder ejecutar para realizar el testeo de la red, modificando las capas del archivo *.prototxt*.
    c. Así mismo, empleé un script del repositorio de SegStereo para obtener la disparidad con las redes elegidas, pero el mismo error de la capa preparada para Caffe-GPU salió al ejecutar dicho script.

7. Empleo de Docker para usar entornos preparados para cada red
    a. Configuración de la imagen de Caffe en Docker para su posterior empleo en entrenamiento y ejecución de modelos de red de SegStereo y CLR.
    b. Creación de un Workspace desde 0 para poder usar las redes anteriores, al no haber podido en un principio usar la imagen proporcionada

8. Estudio del paso de redes en PyTorch a ONNX para solventar problemas surgidos a nuestros compañeros con la red de PSMNet.
    a. Mediante un script que creaba un modelo de una red en Pytorch genérico, lo entrenaba y lo exportaba en formato ONNX.

9. Conversión de redes en Caffe a OpenVINO para observar su (en teoría) mejora en rendimiento en la clasificación de imágenes.
    a. Clasificación de una misma imagen de un coche empleando para ello primeramente una red VGG-16 que venía con OpenVINO para entender el funcionamiento del mismo.
    b. Seguidamente clasificación empleando una VGG-16 nativa en Caffe, y una AlexNet nativa en Caffe también.
    c. Conversión de ambas redes a OpenVINO para poder realizar una comparativa de tiempos y poder evaluar la mejora o no de las mismas.
    d. Clasificación de la imagen con la VGG-16 obteniendo un tiempo muchísimo más reducido que en su ejecución en Caffe nativo.
    e. Clasificación también de la imagen con la AlexNet convertida a OpenVINO, obteniendo un tiempo de la mitad que en su ejecución en Caffe nativo.

10. Trabajo con Movidius para observar su (en teoría) mejora en rendimiento
    a. Optimización de los modelos de AlexNet y VGG-16 a FP16 para poder emplearlos con Movidius.
    b. Clasificación de la misma imagen con el modelo de VGG-16 optimizado en FP16, obteniendo un tiempo menor que en la clasificación en Caffe nativo, pero tres veces mayor que con OpenVINO.
    c. Clasificación de la imagen con el modelo de AlexNet optimizado en FP16, obteniendo un tiempo mayor tanto a Caffe nativo como a OpenVINO.

11. Documentación del trabajo:
    a. Antecedentes del trabajo
    b. Redes de detección de objetos
    c. KITTI
    d. Redes de disparidad:
        i. SegStereo
        ii. CRL

Manuel Oreja Valverde

Durante este trabajo he realizado las siguientes tareas en el siguiente orden cronológico:

1. Aprendizaje sobre inteligencia artificial, clasificación y detección de objetos a través de la documentación y vídeos disponibles en los cursos de inteligencia artificial de la Universidad Autónoma de Barcelona (UAB).

2. Trabajo con Keras sobre redes neuronales convolucionales. En este trabajo comencé con la simple utilización de redes neuronales para clasificación de objetos que Keras disponía, con el fin de realizar inferencia sobre diferentes imágenes y comprobar sus resultados. Posteriormente trabajé con redes neuronales no disponibles en Keras, como AlexNet, y que se podían definir a través de la definición de sus capas mediante este mismo framework. Esto implica, además de definir su arquitectura, entrenarlas con datasets como CIFAR10. Además se evalué, guardé y cargué los modelos entrenados generados.

3. Trabajo con redes neuronales orientadas al cálculo de la disparidad. Esta fase ha sido la más larga debido a que tuvimos que encontrar y probar redes disponibles en el KITTI que tuviesen publicado su código en repositorios y que además diesen la oportunidad de entrenar e inferir, para posteriormente ser evaluadas con las utilidades disponibles en el Development Kit de KITTI. En mi caso trabajé con varias redes:

    a. Primeramente con PSMNet sobre PyTorch. Durante el primer intento no la pude probar, ya que no teníamos una máquina con GPU. La causa era que, a pesar de que la red viene preparada para ser usada sobre una CPU, no funcionó, a pesar de que realicé pequeñas modificaciones para forzar a que el modelo fuese compatible con CPU. Durante el segundo intento, ya con una máquina con GPUs al red funcionó, pero no sin haber solventado anteriormente las limitaciones de memoria de GPUs, almacenamiento en disco y tiempo, explicadas anteriormente.

    Una vez la red dejó de entrenar y generó todos los checkpoints de las 200 épocas ejecuté el código de inferencia sobre el dataset de KITTI. Para mejorar aún más las inferencia, realicé 100 épocas de fine-tune sobre el dataset de KITTI con sus respectivos checkpoints.Tras comprobar que las imágenes parecían estar bien generadas utilicé el Development Kit del KITTI. Posteriormente lo ejecuté para realizar las evaluaciones sobre varios conjuntos de imágenes inferidas de diferentes checkpoints de nuestro entrenamiento. Finalmente también repetí este proceso con las redes pre-entrenadas disponibles en el repositorio para comparar resultados con las que entrenamos.

Una vez obtenido los resultados intente generar el modelo optimizado de PSMNet para realizar la inferencia sobre OpenVINO y así medir la mejora de velocidad que proporciona. Como OpenVino no soporta redes PyTroch, si queremos usar modelos PyTorch tenemos que convertirlos a ONNX previamente. Primeramente hice pruebas con redes simples como AlexNet a través de redes ya pre-entrenadas proporcionadas por PyTorch. Una vez conseguido, realicé lo mismo conPSMNet pero no fue posible.

b. Tras haber obtenido el anterior resultado quisimos utilizar otra red en otro framework que pudiese ser inferida con y sin OpenVINO para medir la mejora de velocidad de inferencia. MADNet fue la escogida, implementada sobre TensorFlow.

   Para comenzar su inferencia antes generé un .csv con las rutas a las imágenes del dataset de KITTI a partir de un script de bash. Tras esto generé las imágenes inferidas y las evalué con el Development Kit de KITTI, previamente adaptado.

   Finalmente se exportó la red a la máquina 7Picos que disponía de OpenVINO. Allí ejecuté el script de conversión a modelo optimizado para OpenVINO sin éxito, y tras no encontrar solución terminamos de trabajar con la red.

c. Por último trabajé con la red Inception-v3. La finalidad era realizar sobre la red un entrenamiento/fine-tune para posteriormente congelarla y transformarla al formato optimizado para OpenVINO con objetivo de inferir sobre la misma.

4. Documentación del trabajo, especialmente enfocándose en la documentación de resultados y conclusiones de las redes con las que trabajé.

Alejandro Pascua Piña

A lo largo de este trabajo he realizado las siguientes actividades:

1. Visualización de vídeos y lectura de documentos de la Universidad Autónoma de Barcelona (UAB) sobre el aprendizaje con inteligencia artificial y la clasificación y detección de objetos.

2. Toma de contacto con redes neuronales sencillas como AlexNet, realizar entrenamiento y clasificación con ellas.

3. Estudio de Caffe sobre el funcionamiento de cómo entrenar redes neuronales y cómo usarlas posteriormente para realizar clasificación de objetos.

    a. Siendo necesarios el entrenamiento:

        i. La arquitectura de la red, un train.prototxt
        ii. Un solver para poder entrenarlo, un solver.prototxt

        Durante el entrenamiento se van produciendo distintos archivos que van guardando los avances producidos en el entrenamiento, los .solverstate, así si, si paras el entrenamiento o éste acaba repentinamente por alguna razón (corte de luz por ejemplo), puedes volver a poner la red a entrenar desde el último “punto de guardado”. Finalmente obtienes un archivo con los pesos para la red, necesario para utilizarla, los .caffemodel.

    b. Siendo necesarios para la clasificación:

        i. Un deploy del modelo, un deploy.prototxt
        ii. Los pesos anteriormente obtenidos, el .caffemodel.

        Para obtener el deploy, es necesario quitar las capas de datos del modelo y añadir una capa de entrada. Una vez tengas esto realizado, ya puedes realizar inferencia sobre otras fotos con tu red.

4. En lo referente al KITTI, me centre en las redes SegStereo y CRL, ya que estaban implementadas con Caffe. Con ellas realicé:

    a. Estudio de las redes y del código proporcionado para poder usarlas.

    b. Tratar de compilar una rama de Caffe proporcionada por las redes (cada una la suya), ya que no estaban soportadas por la rama principal de Caffe. Esto llevó a solucionar diferentes problemas sobre librerías, y lidiar con otros que no fuimos capaces de resolver.

c. Tratar de emplear Docker para evitar los problemas relacionados con la compilación de las variantes de Caffe proporcionadas por las redes, aunque sin éxito.

5. OpenVino:

    a. Pasar redes que habían sido previamente probadas en Caffe a OpenVino para comprobar si se producía una mejora en el tiempo de inferencia.

        i. Pasar el modelo de Caffe a OpenVINO con GPU, pero sin posibilidad de realizar la inferencia al no disponer de una GPU Intel.

        ii. Pasar el modelo de Caffe a OpenVINO con CPU para las redes VGG-16 y AlexNet, probando la inferencia y comprobando que los tiempos obtenidos al realizar la inferencia con él, son mucho menores.

        iii. Pasar el modelo de Caffe de las redes anteriores a OpenVINO con FP16, para poder realizar la inferencia en el Movidius, un dispositivo centrado en el bajo consumo.

        iv. Tratar de comprobar los tiempos de inferencia con un modelo de FP16 sobre CPU sin éxito, ya que al parecer, solo acepta FP32.

6. En la memoria:

    a. Redes Neuronales

    b. Conducción Autónoma

    c. Primeros Pasos

    d. OpenVINO desde Caffe

Marcos Robles Palencia

Durante este trabajo he realizado las siguientes actividades:

1. Aprendizaje a través de la visualización de videos y lectura de documentos de la Universidad Autónoma de Barcelona (UAB), sobre inteligencia artificial, clasificación y detección de objetos. Esto fue de gran utilidad para iniciarme en este mundo tan en auge, puesto que me ayudó a trabajar con modelos, realizar medidas de rendimiento, conocer el funcionamiento del tratamiento de los píxeles de la imagen para poder clasificarlos y saber a qué clase pertenecen.
   Aprendizaje sobre machine learning, redes neuronales y su aplicación en la conducción autónoma.

2. Trabajo con redes de clasificación de objetos:

    a. Obtener información de este tipo de redes, a través de sus papers, para conocer su funcionamiento.

    b. Preparación del entorno para ejecutar las distintas redes neuronales convolucionales, buscando información de cada framework en el cual estaban desarrolladas. Muchas me dieron errores, y al final conseguí entrenar y testear las siguientes:

        i. Una implementación de AlexNet en Keras, para la cual tuve que informarme sobre el dataset de CIFAR10 y cómo conseguir que esta red pudiese tener este dataset de entrada, para entrenar y testearla posteriormente.

        ii. Una implementación de una red preentrenada de VGG-16 de Keras para su inferencia con el dataset de CIFAR10.

        iii. Finalmente, almacene los pesos y la topología de las redes para posteriormente usarlas para la inferencia en OpenVINO.

3. Trabajo con redes orientadas al cálculo de disparidad:

    a. Busque información en la página de KITTI, para usar su dataset, sobre todo el de imágenes de stereo sobre conducción y de las redes de disparidad que se proporcionaban, para ello investigue las redes de predicción de disparidad. En este paso tuve especial dificultad, por las pocas redes con código disponible y funcional. Me centré en trabajar sobre las siguientes redes:

i. Trabajo con la red de disparidad PSMNet.
   Lo primero fue informarme sobre el dataset SceneFlow, para finalmente usar de los 3 dataset existentes el de conducción, por los problemas de espacio que tuve en la máquina, explicados en la memoria. Una vez resueltos estos problemas, me dispuse a entrenarla con diferentes épocas, para posteriormente evaluar los cambios de rendimiento. Conseguí ejecutar este entrenamiento en varias GPUs.
   Realice pruebas de inferencia sobre el dataset del KITTI2015 y ejecuté el estudio para calcular el error de disparidad, que te provee la página, y de esta forma obtener los resultados mostrados anteriormente.

ii. Trabajo con la red de disparidad MADNet.
   En esta red conseguí realizar la inferencia sobre el dataset del KITTI2015 y ejecuté el estudio para calcular el error de disparidad, que te provee la página, y de esta forma obtener los resultados mostrados anteriormente.

4. Trabajo con OpenVINO:

   a. Intenté pasar las redes PSMNet y MADNet para su ejecución en las herramientas de OpenVINO, con los resultados negativos, al producirse los errores de conversión mostrados anteriormente.

   b. Sobre las redes de clasificación de objetos con las que trabajé al principio (AlexNet y VGG-16):

      i. Las transformé a formato ONNX y ejecuté los scripts para poder conseguir los archivos necesarios para la inferencia.

      ii. Realicé la inferencia en CPU utilizando OpenVINO y de ahí saqué los resultados de tiempos y clases, con mayor probabilidad obtenida, de cada red.

      iii. Optimicé el modelos de las redes con FP16, para realizar la inferencia en el dispositivo Movidius, y de ahí saqué los resultados de tiempos y clases, con mayor probabilidad obtenida de cada red.

5. Documentación del trabajo:

   El trabajo que he realizado en la memoria ha sido: portada normalizada, introducción, tecnologías utilizadas, redes de clasificación de objetos, explicación de las tareas realizadas con PSMNet y MADNet, incluyendo gráficas, trabajo con OpenVINO y conclusiones.

7.- Anexos I Tecnologías utilizadas

7.1.- Python[29]

Logo de Python[30]

Python es un lenguaje de programación multiparadigma, soporta orientación a objetos, programación imperativa y programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1.

Existen múltiples librerías de código abierto enfocadas a su uso para Deep Learning en lenguaje Python. Las más importantes son: TensorFlow, Theano, Keras, Caffe, Lasagne, DSSTNE, PrettyTensor, Torch, mxnet, DL4J, y Microsoft Cognitive Toolkit.

Python ofrece librerías de herramientas científicas, numéricas, herramientas de análisis y estructuras de datos, y algoritmos de Machine Learning como NumPy, SciPy, Matplotlib, Pandas o PyBrain. También ofrece entornos interactivos de programación orientados a Data Science.

[29] https://www.python.org/
[30] *https://es.wikipedia.org/wiki/Archivo:Python.svg*

7.2.- Caffe[31]

Logo de Caffe[32]

Caffe es uno de los frameworks más antiguos, fue desarrollado por Berkeley Al Research (BAIR) y posee una licencia de código abierto BSD 2-Clause license.

Entre las principales características se encuentran: arquitectura que fomenta la innovación, código expandible que fomenta el desarrollo activo, velocidad para la realización de investigaciones y una comunidad interesada.

Caffe soporta diferentes tipos de arquitectura de deep learning para clasificación de imágenes y segmentación de imágenes. Soporta CNN, RCNN, LSTM y redes neuronales totalmente conectadas. Caffe soporta aceleración de cálculos basados en GPU.

No es un framework de propósito general. Se centra únicamente en la visión artificial. Los inconvenientes que vimos es que no era nada flexible y la documentación es bastante pobre. Uno de los puntos flacos que tiene es la dificultad de instalación. Tiene muchas dependencias que resolver.

Proporciona una API de programación para Python y Matlab para cambios sencillos, aunque si se quiere introducir cambios significativos deben de ser programados en C++ o en CUDA.

[31] https://caffe.berkeleyvision.org/
[32] *https://maxchama.blogspot.com/2019/03/caffe-deep-learning-framework.html*

7.3.- Keras[33]

Logo de Keras[34]

Keras es una biblioteca de Redes Neuronales de Código Abierto escrita en Python. Es capaz de ejecutarse sobre TensorFlow.

Está diseñada sobre todo para posibilitar la experimentación en más o menos poco tiempo con redes de deep learning. Sus principales características se centran en ser amigable para el usuario, modular y extensible.

Su autor principal y mantenedor ha sido el ingeniero de Google François Chollet.

Chollet expone que Keras ha sido concebido para actuar como una interfaz en lugar de ser una framework de machine learning standalone. Ofrece un conjunto de abstracciones más intuitivas y de alto nivel haciendo más sencillo el desarrollo de modelos de deep learning independientemente del backend computacional utilizado.

Keras contiene varias implementaciones de los bloques para la construcción de las redes neuronales como por ejemplo los layers, funciones objetivo, funciones de activación y optimizadores matemáticos.

Se caracteriza por un modelo de programación sin adornos, para maximizar la legibilidad y minimizar las líneas de código,además dispone de una documentación muy completa.

[33] https://keras.io/

[34] *https://commons.wikimedia.org/wiki/File:Keras_Logo.jpg*

7.4.- Pytorch[35]

Logo de Pytorch[36]

PyTorch es una librería de código abierto de Python basado en Torch está diseñada para realizar cálculos numéricos haciendo uso de la programación de tensores. Además permite su ejecución en GPU para acelerar los cálculos.

PyTorch se suele usar para sustituir numpy y procesar los cálculos en GPU y para la investigación y desarrollo en el campo del machine learning, centrado principalmente en el desarrollo de redes neuronales.

PyTorch es una librería muy reciente, sin embargo, dispone de gran cantidad de manuales y tutoriales donde encontrar ejemplos. Además de una comunidad que está creciendo rápidamente.

PyTorch dispone una interfaz muy sencilla para la creación de redes neuronales pese a trabajar de forma directa con tensores sin la necesidad de una librería a un nivel superior.

PyTorch trabaja con grafos dinámicos en vez de estáticos, es decir, en tiempo de ejecución se pueden ir modificando las funciones y el cálculo del gradiente variará con ellas.

PyTorch dispone de soporte para su ejecución en tarjetas gráficas (GPU), utiliza internamente CUDA.

[35] https://pytorch.org/
[36] *https://commons.wikimedia.org/wiki/File:Pytorch_logo.png*

7.5.- TensorFlow[37]

Logo de Tensorflow[38]

TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, fue desarrollado por Google para deep learning, se la considera de bajo nivel lo que proporciona mayores posibilidades de adaptación en detrimento de la simplicidad de uso.

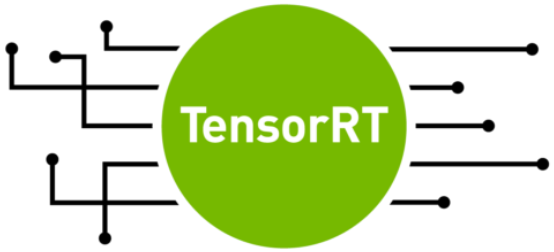
TensorFlow para el cálculo numérico utiliza grafos de flujo de datos. Los nodos en el grafo representan operaciones matemáticas, mientras que los bordes del grafo representan los conjuntos de datos multidimensionales (tensores) comunicados entre ellos.

En su web lo definen como: “An open-source software library for Machine Intelligence” (una librería de código libre para Machine Intelligence), pero es más precisa esta definición que aparece justo debajo: “ TensorFlow™ is an open source software library for numerical computation using data flow graphs” (Tensorflow es una librería de código libre para computación numérica usando grafos de flujo de datos).

Tensorflow soporta Python y C++, además de permitirte distribuir los cálculos en CPU y GPU (varias de forma simultánea) e incluso el escalado horizontal usando gRPC.

[37] https://www.tensorflow.org/
[38] *https://es.wikipedia.org/wiki/Archivo:Tensorflow_logo.svg*

7.6.- TensorRT[39]



Logo de TensorRT[40]

NVIDIA TensorRT es una plataforma de alto rendimiento para inferencia en deep learning.

Incluye un optimizador de inferencia de deep learning y un tiempo de ejecución que ofrece baja latencia y alto rendimiento para aplicaciones de inferencia de deep learning.

Las aplicaciones basadas en TensorRT funcionan hasta 40 veces más rápido que las plataformas solo para CPU durante la inferencia. Con TensorRT, puedes optimizar los modelos de redes neuronales entrenados en todos los frameworks principales, calibrar para obtener una precisión más baja con alto acierto.

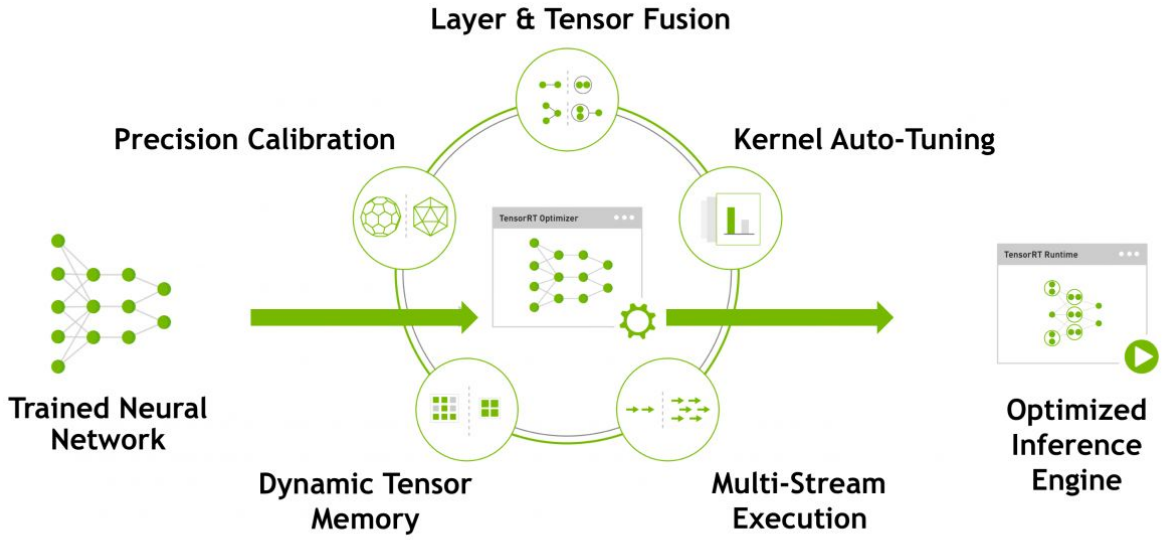
TensorRT se basa en CUDA, el modelo de programación paralelo de NVIDIA, y le permite optimizar la inferencia para todos los frameworks de deep learning aprovechando bibliotecas, herramientas de desarrollo y tecnologías en CUDA-X AI para inteligencia artificial, máquinas autónomas, computación de alto rendimiento y gráficos.

TensorRT proporciona optimizaciones INT8 y FP16 para implementaciones de producción de aplicaciones de inferencia de deep learning, como transmisión de video, reconocimiento de voz, recomendaciones y procesamiento de lenguaje natural. La inferencia de precisión reducida reduce significativamente la latencia de la aplicación, que es un requisito para muchos servicios en tiempo real, aplicaciones automáticas e integradas.

Puedes importar modelos entrenados desde cada framework de deep learning a TensorRT.

Con TensorRT, los desarrolladores pueden concentrarse en crear aplicaciones novedosas basadas en inteligencia artificial en lugar de ajustar el rendimiento para la implementación de inferencia.

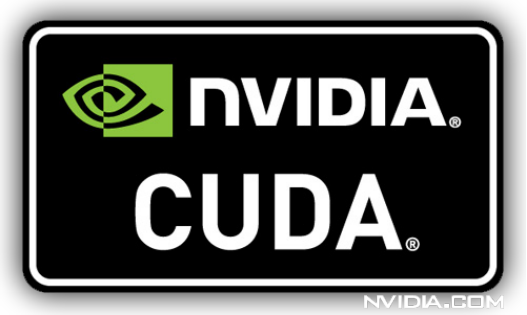
[39] https://developer.nvidia.com/tensorrt
[40] *https://devblogs.nvidia.com/speed-up-inference-tensorrt/*



Funcionamiento de TensorRT[41]

[41] https://developer.nvidia.com/tensorrt

7.7.- CUDA[42]



Logo de CUDA[43]

CUDA son las siglas de Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo) que hace referencia a una arquitectura de cálculo en paralelo incluyendo un compilador y un conjunto de herramientas de desarrollo creadas por NVidia que permiten a los programadores usar una variación del lenguaje de programación C para codificar algoritmos en GPU de NVidia aprovechando la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema.

CUDA intenta explotar las ventajas de las GPU frente a las CPU de propósito general utilizando el paralelismo que ofrecen sus múltiples núcleos, que permiten el lanzamiento de un alto número de hilos simultáneos.

CUDA presenta ciertas ventajas sobre otros tipos de computación sobre GPU utilizando APIs gráficas:

1. Lecturas dispersas: se puede consultar cualquier posición de memoria.
2. Memoria compartida: CUDA pone a disposición del programador un área de memoria que se compartirá entre hilos.
3. Lecturas más rápidas de y hacia la GPU.
4. Soporte para enteros y operadores a nivel de bit.

CUDA presenta ciertas limitaciones:

1. No se puede utilizar recursividad, punteros a funciones, variables estáticas dentro de funciones o funciones con número de parámetros variable
2. No está soportado el renderizado de texturas
3. En precisión simple no soporta números desnormalizados o NaNs

[42] https://developer.nvidia.com/cuda-zone
[43] *https://es.wikipedia.org/wiki/Archivo:CUDA.png*

4. Puede existir un Cuello de botella entre la CPU y la GPU por los anchos de banda de los buses y sus latencias.
5. Los threads o Hilo de ejecución, por razones de eficiencia, deben lanzarse en grupos de al menos 32, con miles de hilos en total.

7.8.- OpenCL[44]

Logo de OpenCL[45]

OpenCL (Open Computing Language, en español lenguaje de computación abierto) consta de una interfaz de programación de aplicaciones y de un lenguaje de programación. Juntos permiten crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse tanto en CPU como en GPU. El lenguaje está basado en C99, eliminando cierta funcionalidad y extendiéndose con operaciones vectoriales.
Apple creó la especificación original y fue desarrollada en conjunto con AMD, IBM, Intel y NVIDIA.

OpenCL forma parte de Mac OS X v10.6, mientras que AMD decidió apoyar OpenCL en lugar de su antigua API Close to Metal. Intel también dispone de su propio entorno de desarrollo y NVIDIA además de tener su propia API para chips gráficos llamada CUDA, también admite OpenCL.

[44] https://www.khronos.org/opencl/
[45] *https://es.m.wikipedia.org/wiki/Archivo:OpenCL.jpg*

8.- Bibliografía

https://github.com/yangguorun/SegStereo

```
@inproceedings{yang2018SegStereo,
  author    = {Yang, Guorun and
               Zhao, Hengshuang and
               Shi, Jianping and
               Deng, Zhidong and
               Jia, Jiaya},
   title        = {SegStereo: Exploiting Semantic Information for Disparity
Estimation},
  booktitle = ECCV,
  year      = {2018}
}
```

https://github.com/Artifineuro/crl

```
@inproceedings{pang2017cascade,
       title={Cascade residual learning: A two-stage convolutional neural
network for stereo matching},
     author={Pang, Jiahao and Sun, Wenxiu and Ren, Jimmy SJ and Yang, Chengxi
and Yan, Qiong},
    booktitle = {ICCV Workshop on Geometry Meets Deep Learning},
    month = {Oct},
    year = {2017}
}
```

https://github.com/JiaRenChang/PSMNet

```
@inproceedings{chang2018pyramid,
  title={Pyramid Stereo Matching Network},
  author={Chang, Jia-Ren and Chen, Yong-Sheng},
    booktitle={Proceedings of the IEEE Conference on Computer Vision and
Pattern Recognition},
  pages={5410--5418},
  year={2018}
}
```

https://github.com/CVLAB-Unibo/Real-time-self-adaptive-deep-stereo

```
@InProceedings{Tonioni_2019_CVPR,
    author = {Tonioni, Alessio and Tosi, Fabio and Poggi, Matteo and
Mattoccia, Stefano and Di Stefano, Luigi},
    title = {Real-time self-adaptive deep stereo},
    booktitle = {The IEEE Conference on Computer Vision and Pattern
Recognition (CVPR)},
    month = {June},
    year = {2019}
```

}

KITTI

```
@INPROCEEDINGS{Menze2015CVPR,
  author = {Moritz Menze and Andreas Geiger},
  title = {Object Scene Flow for Autonomous Vehicles},
  booktitle = {Conference on Computer Vision and Pattern     Recognition (CVPR)},
  year = {2015}
}
```


[1]https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/: Información sobre redes neuronales convolucionales.
[2] https://radiopaedia.org/articles/batch-size-machine-learning: Información sobre el batch size.
[3] https://github.com/BIGBALLON/cifar-10-cnn: Distintas redes con CIFAR10.
[4] https://github.com/NVIDIA/DIGITS/issues/156: Información sobre un error aparecido en la compilación de Caffe.
[5]https://askubuntu.com/questions/960238/nvcc-fatal-unsupported-gpu-architecture-compute-20: Información sobre un error de la GPU.
[6] https://software.intel.com/en-us/articles/OpenVINO-IE-Samples#image-classification: Información de OpenVINO Toolkit.
[7] https://github.com/BVLC/caffe/issues/3028: Solucionar error OpenBLAS al compilar Caffe.
[8] http://www.cvlibs.net/datasets/kitti/: Página principal de KITTI.
[9] https://github.com/yangguorun/SegStereo: Repositorio de la red SegStereo.
[10] https://github.com/movidius/ncappzoo/blob/master/caffe/AlexNet/run.py: Script de ejecución de AlexNet para MOVIDIUS.
[11] https://gist.github.com/huyng/34b0b5e6af6e623f331f: Ejemplo de inferencia con Caffe.
[12]https://picodotdev.github.io/blog-bitix/2014/11/como-crear-una-imagen-para-docker-usando-un-dockerfile/: Información para crear un docker.
[13]https://caffe.berkeleyvision.org/: Página principal de Caffe
[14] https://github.com/BVLC/caffe/wiki/Using-a-Trained-Network:-Deploy: Creación de un archivo Deploy a partir de una definición de red.
[15] https://github.com/nvidia/digits/issues/1107: Solucionar error no se encuentra Caffe.
[16] https://docs.docker.com/engine/reference/commandline/image_build/: Información para crear la imagen de un docker.
[17] https://github.com/intel/caffe: Repositorio de INTEL del framework Caffe.
[18] https://github.com/BVLC/caffe: Repositorio principal del framework Caffe.
[19] https://iie.fing.edu.uy/publicaciones/2005/Lec05a/Lec05a.pdf: Paper sobre el cálculo de la disparidad en imágenes estéreo.
[20]https://stackoverflow.com/questions/30830987/how-to-create-an-caffemodel-file-from-training-image-and-its-labeled: Cómo obtener un caffemodel.
[21] http://shengshuyang.github.io/A-step-by-step-guide-to-Caffe.html: Información sobre la instalación y utilización de Caffe.
[22] https://github.com/Artifineuro/crl/: Repositorio principal de la red CRL.
[23] https://es.wikipedia.org/wiki/Inteligencia_artificial: Información de inteligencia artificial.

[24] https://es.wikipedia.org/wiki/Test_de_Turing: Información del Test de Turing.
[25] https://hipertextual.com/2016/12/tipos-de-inteligencia-artificial: Artículo sobre la Inteligencia Artificial.
[26] https://omicrono.elespanol.com/2018/10/que-es-la-inteligencia-artificial/: Información sobre inteligencia artificial.
[27] https://www.apd.es/tipos-de-inteligencia-artificial/: Artículo sobre los típos de IA.
[28]https://www.blog.andaluciaesdigital.es/deep-learning-inteligencia-artificial-y-machine-learning/: Artículo sobre la diferencia entre Inteligencia Artificial, Deep Learning y Machine Learning.
[29] https://es.wikipedia.org/wiki/Aprendizaje_profundo: Información sobre el aprendizaje profundo o deep learning.
[30] https://es.mathworks.com/discovery/deep-learning.html: Artículo sobre la importancia del Deep Learning.
[31] https://es.wikipedia.org/wiki/Aprendizaje_supervisado: Información sobre el aprendizaje supervisado.
[32] https://es.wikipedia.org/wiki/Aprendizaje_no_supervisado: Información sobre el aprendizaje no supervisado.
[33] https://es.wikipedia.org/wiki/Disparidad_binocular: Información sobre disparidad binocular.
[34] https://computervisiononline.com/dataset/1105138650: Artículo sobre el benchmark de KITTI.
[35]https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/: artículo sobre arquitecturas de deep learning.
[36] https://www.cs.toronto.edu/~kriz/cifar.html: Información y archivos del dataset CIFAR10.
[37]https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo: Artículo sobre las Redes Neuronales.
[38] https://en.wikipedia.org/wiki/Artificial_neural_network: Información sobre las redes neuronales.
[39]http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/: Artículo sobre la historia de las redes neuronales.
[40] https://en.wikipedia.org/wiki/Convolutional_neural_network#Distinguishing_features: Artículo sobre las Redes Convolucionales.
[41] https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/: Información sobre redes neuronales convolucionales.
[42] https://en.wikipedia.org/wiki/Self-driving_car: Información sobre la conducción autónoma.
[43] https://www.bmw.com/en/automotive-life/autonomous-driving.html: Información sobre conducción autonoma.
[44]https://www.nationalgeographic.com.es/promociones/pasado-presente-futuro-conduccion-autonoma_12014: Artículo sobre la Conducción Autónoma.
[45] https://www.autonocion.com/historia-coche-autonomo/: Artículo sobre los coches autónomos.
[46] https://es.wikipedia.org/wiki/Python: Artículo sobre Python.
[47] https://data-speaks.luca-d3.com/2018/05/deep-learning-con-python-introduccion.html: Serie de artículos sobre Deep Learning con Python.

[48] https://data-speaks.luca-d3.com/2018/03/atrevete-con-el-python-un-experimento.html: Artículo sobre Python.
[49]https://medium.com/@ricardo.guerrero/frameworks-de-deep-learning-un-repaso-antes-de-acabar-el-2016-5b9bf5b9f9af: Información sobre frameworks de deep learning.
[50]http://www.diegocalvo.es/implementacion-de-redes-neuronales-comparativa-frameworks/: Información y comparación sobre frameworks de deep learning.
[51] https://en.wikipedia.org/wiki/Caffe_(software): Artículo sobre el framework Caffe.
[52] https://es.wikipedia.org/wiki/Keras: Información sobre el framework de Keras.
[53]http://www.diegocalvo.es/implementacion-de-redes-neuronales-comparativa-frameworks/: Artículo sobre distintos frameworks para la implementación de Redes Neuronales.
[54]https://medium.com/@ricardo.guerrero/frameworks-de-deep-learning-un-repaso-antes-de-acabar-el-2016-5b9bf5b9f9af: Artículo sobre fremeworks de deep learning.
[55]https://planetachatbot.com/deep-learning-f%C3%A1cil-con-deepcognition-9af43b2319ba: Información sobre deep learning y frameworks de deep learning.
[56] https://cleverpy.com/que-es-pytorch-y-como-se-instala/: Artículo explicativo sobre PyTorch.
[57] https://en.wikipedia.org/wiki/PyTorch: Información sobre PyTorch.
[58] https://es.wikipedia.org/wiki/TensorFlow: Información sobre TensorFlow.
[59]https://medium.com/@ricardo.guerrero/frameworks-de-deep-learning-un-repaso-antes-de-acabar-el-2016-5b9bf5b9f9af: Artículo sobre frameworks de Deep Learning.
[60]http://www.diegocalvo.es/implementacion-de-redes-neuronales-comparativa-frameworks/: Comparación de frameworks de Deep Learing.
[61]https://newsroom.intel.la/news-releases/la-inteligencia-de-vision-de-intel-transforma-la-industria-del-internet-de-las-cosas/#gs.7xuae0: Información de intel sobre herramientas de
[62]https://www.intel.es/content/www/es/es/internet-of-things/solution-briefs/openvino-toolkit-product-brief.html: Información sobre OpenVINO.
[63]https://vhdl.es/aplicacion-de-la-inteligencia-artificial-en-los-sistemas-de-vision-por-computador-usando-openvino/: Artículo sobre Inteligencia Artificial con OpenVINO
[64] https://www.nvidia.es/object/cuda-parallel-computing-es.html: Artículo de NVIDIA sobre CUDA.
[65] https://es.wikipedia.org/wiki/CUDA: Información sobre CUDA.
[66] https://es.wikipedia.org/wiki/OpenCL: Artículo explicativo sobre OpenCL.
[67]https://www.applesfera.com/aplicaciones-os-x-1/opencl-la-alternativa-libre-a-cuda-y-el-futuro-de-la-computacion-gpgpu: Artículo comparativo entre CUDA y OpenCL
[68] https://software.intel.com/en-us/openvino-toolkit/documentation/code-samples: Códigos de ejemplo para trabajar sobre OpenVINO.
[69]http://docs.openvinotoolkit.org/latest/_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html: Guía para optimizar modelos con OpenVINO.
[70]http://docs.openvinotoolkit.org/latest/_docs_MO_DG_prepare_model_convert_model_Convert_Model_From_Caffe.html: Guía para optimizar modelos en Caffe con OpenVINO.
[71]http://docs.openvinotoolkit.org/latest/_inference_engine_samples_classification_sample_README.html: Ejemplo de clasificación con OpenVINO.