Escola Tècnica Superior d'Enginyeria ETSE-UV

MÁSTER UNIVERSITARIO EN CIENCIA DE DATOS

VNIVERSITAT DE VALÈNCIA

TRABAJO DE FIN DE MÁSTER

CLASIFICACIÓN DE IMÁGENES NATURALES Y MULTI-ESPECTRALES MEDIANTE REDES NEURONALES CONVOLUCIONALES Y NORMALIZACIÓN DIVISIVA

AUTOR:
JORDI SILVESTRE LLOPIS

TUTORES:
JORDI MUÑOZ MARÍ
VALERO LAPARRA PÉREZ-MUELAS

JULIO, 2019

MÁSTER UNIVERSITARIO EN CIENCIA DE DATOS

TRABAJO DE FIN DE MÁSTER

CLASIFICACIÓN DE IMÁGENES NATURALES Y MULTI-ESPECTRALES MEDIANTE REDES NEURONALES CONVOLUCIONALES Y NORMALIZACIÓN DIVISIVA

AUTOR:
JORDI SILVESTRE LLOPIS

TUTORES:
JORDI MUÑOZ MARÍ
VALERO LAPARRA PÉREZ-MUELAS

TRIBUNAL:

PRESIDENTE: VOCAL 1:

VOCAL 2: **FECHA DE DEFENSA:**

CALIFICACIÓN:

RESUMEN

El método de “Normalización Divisiva Generalizada” (GDN) aprovecha la potencia de las convoluciones para realizar normalizaciones aplicadas localmente en imágenes. Gracias a esta técnica, se logra maximizar el contraste de forma uniforme en casos de iluminación variante entre áreas. La capacidad de apreciación de formas y patrones se ve incrementada, frente a la que se obtendría aplicando métodos de normalización global convencionales como el *Batch Normalization.*

La comparación de ambas técnicas sobre redes neuronales convolucionales ha permitido determinar que la normalización GDN aporta mejoras significativas aplicada a modelos de clasificación de imágenes naturales. No ha sido posible concluir si esta mejora también se manifiesta de forma positiva en modelos de segmentación semántica con arquitecturas “U-Net”, quedando pendiente de verificación mediante experimentos más exhaustivos que no ha sido posible realizar por falta de recursos.

ÍNDICE DE CONTENIDOS

TABLA DE ILUSTRACIONES

1 Introducción

El *Machine Learning* ha aportado soluciones a multitud problemas hasta ahora inabordables por las ciencias de la computación. En especial, el campo de la visión por computador avanza a un ritmo inimaginable hace tan solo una década. El reconocimiento automático de objetos, personas y espacios ya es una realidad tangible.

En este proyecto se desarrollarán modelos para la clasificación y segmentación de imágenes naturales y de satélite, basados en metodologías *Deep Learning* mediante redes neuronales convolucionales. Se realizará una pequeña investigación con el fin de determinar cuáles son las técnicas y arquitecturas con mejores resultados en el estado del arte actual para la resolución de los problemas planteados.

El objetivo principal es el de determinar si el uso del método de normalización divisiva generalizada (GDN) aporta mejoras significativas al rendimiento de los modelos frente a otras alternativas de normalización más extendidas. La valoración del rendimiento obtenido por las soluciones desarrolladas se realizará aplicando métricas consistentes con la naturaleza de los problemas planteados.

2 Estado del Arte

Dentro de la ciencia de datos, existe un amplio campo de estudio en lo referente a imagen. Dos de los problemas más comunes a resolver en este campo son la clasificación y la segmentación semántica. En la clasificación de imágenes, se trata de identificar los objetos que aparecen en ellas dentro de una o varias categorías determinadas. Por otra parte, la segmentación semántica trata de distinguir objetos o regiones dentro de las imágenes.

En los últimos tiempos, el Deep Learning ha ayudado a conseguir numerosos avances en visión por computador. Gran parte de los recientes éxitos se debe al uso de operaciones de convolución (kernels) como parte fundamental de redes neuronales artificiales. Las convoluciones permiten extraer características de una imagen, fase tras la que se podrán aplicar otras técnicas para solucionar los problemas a que nos enfrentamos.

A continuación se estudiará un poco más en profundidad qué son las convoluciones y cómo pueden ayudar en la visión por computador.

2.1 Redes neuronales Convolucionales

Una convolución es una operación matricial entre unos datos de entrada en forma de matriz, y otra matriz llamada *"kernel"* o filtro. Este filtro se proyecta sucesivamente sobre los datos de entrada, produciendo como salida un único valor producto de la convolución en cada proyección. Cada uno de estos valores se calcula como la suma de los productos *element-wise*[1] de la transpuesta del filtro por la sub-matriz sobre la que se proyecta.

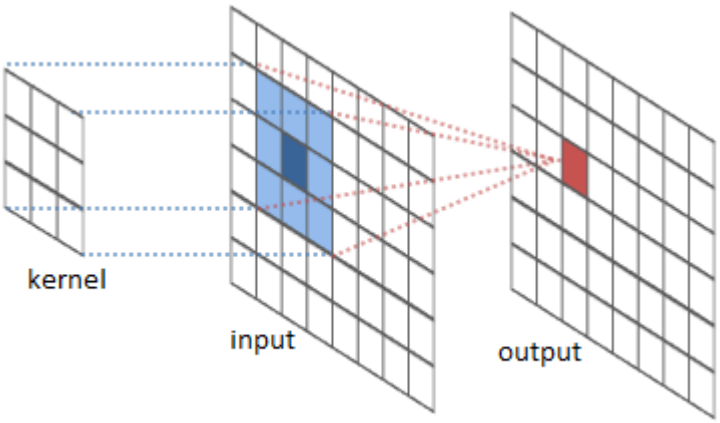


Ilustración 1. Convolución de dos matrices. Fuente: River Trail tutorial. [1]

[1] **Operación Element-Wise**: Se dice de aquellas operaciones que se aplican sobre factores vectoriales elemento a elemento, por lo que requiere que ambos factores sean equidimensionales. En el caso de dos matrices bidimensionales A y B se operaría cada elemento $A_{i,j}$ con el $B_{i,j}$, resultando una matriz de igual tamaño que las originales.

Las convoluciones se aplican en ventanas de un determinado tamaño (el del filtro). Estas ventanas se "desplazan" a lo largo y ancho de las imágenes, operando sobre todas las capas que pudiesen existir. De esta forma, permiten procesar toda la imagen independientemente de su tamaño y profundidad de capas (monocolor, RGB, etc.).

El tamaño de la salida producida dependerá de diversos factores. Uno de los más importantes es la técnica empleada en el procesamiento de los límites de la imagen. Otros dos factores son el tamaño del filtro y el *"stride"* o paso (en píxeles) entre una convolución y otra. [2]

• **Con relleno:** Consiste en rellenar los píxeles más allá de los límites de la imagen con determinados valores, de manera que se pueda aplicar la convolución más allá de los extremos originales. De esta forma, es posible obtener una matriz resultante de igual tamaño a la original (si $stride = 1$). Comúnmente se aplica *"Zero-Padding"*, que consiste hacer un relleno con ceros (por lo que no influirían en el resultado). Existen otras aproximaciones que utilizan el valor del píxel más cercano, la media de los vecinos, etcétera.

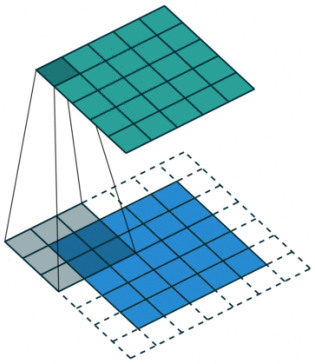


Ilustración 2. Convolución empleando píxeles de relleno. Fuente: Types of convolutions in Deep Learning. [3]

• **Sin relleno:** Consiste en aplicar la convolución haciendo coincidir los límites del filtro con los de los datos de entrada. La imagen resultante será más pequeña que la original, y dependerá tanto del tamaño del filtro y el paso. En la Ilustración 3 podemos observar un ejemplo de convolución sin relleno, filtro (en gris) 3x3 y paso 2 sobre una matriz de entrada (en azul) tamaño 5x5. El resultado (en verde) es una matriz de tamaño 2x2.

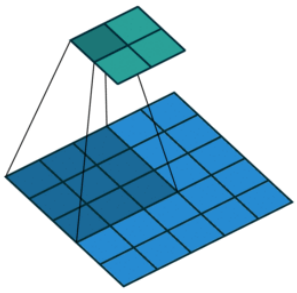


Ilustración 3. Convolución sin píxeles de relleno. Fuente: Types of convolutions in Deep Learning. [3]

• **Otras aproximaciones:** Existen otras aproximaciones para determinar las regiones que intervienen al realizar la convolución, como en el caso de las convoluciones dilatadas (espaciado entre filas y/o columnas intervinientes) o truncadas (selectivamente se ignoran algunas filas y/o columnas)

El número de filtros que se aplique a una imagen determinará el número de capas o canales de salida. Cada filtro actúa independientemente sobre todos los canales existentes en los datos de entrada, produciendo una salida bidimensional. El resultado conjunto de la convolución de los distintos filtros (bidimensionales) sobre los datos (tridimensionales) será, por tanto, un objeto tridimensional de profundidad igual al número de filtros. [4] El tamaño dependerá de los parámetros estudiados anteriormente.

Las convoluciones permiten aprender patrones localizados dentro de las imágenes, por lo que son muy útiles detectando formas y características de los objetos en ellas. Esto otorga a las redes convolucionales ciertas propiedades muy útiles en la visión por computador: [4]

- **Los patrones aprendidos son invariantes a la traslación:** Puesto que la operación de convolución se aplica iterativamente sobre ventanas de la imagen, y no sobre toda ella en su conjunto, las activaciones que se produzcan serán independientes de la región donde los objetos o características estén situados.

- **Son capaces de aprender jerarquías de patrones:** Una primera capa de convolución puede aprender patrones ubicados en regiones más concretas. La combinación de éstos en capas posteriores permitirá detectar sucesivamente características más generales (Ilustración 4).

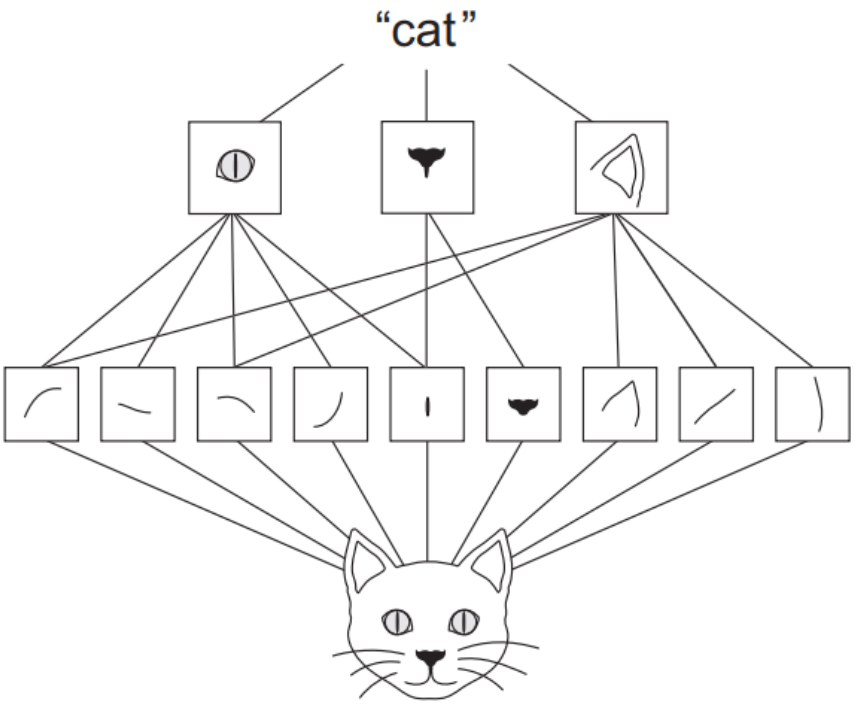


Ilustración 4. Jerarquía espacial aprendida mediante convoluciones. Fuente: Deep Learning with Python [4]

2.1.1 Convolutional Neural Networks (CNN)

Las redes neuronales convolucionales son aquellas que hacen uso de capas convolucionales para extraer características de los datos de entrada. En función de su arquitectura serán útiles para solucionar determinados tipos de problemas, como la clasificación o la segmentación semántica.

Aunque cualquier red neuronal que utilice capas convolucionales se podría denominar CNN, comúnmente se encajan aquí aquellas arquitecturas que tratan de resolver problemas de clasificación. Las arquitecturas utilizadas para realizar segmentación semántica de imágenes, como se verá más adelante, son las llamadas FCNN ("*Fully Convolutional Neural Networks*").

El componente arquitectural que determina el tipo de problema a resolver por la red suele ser la capa o capas posteriores, ya que de ellas depende la salida. Comúnmente, las redes neuronales utilizan capas densas (*"fully connected"*) para realizar tareas de clasificación. También en imágenes son las más apropiadas para implementar la salida.

La arquitectura común en las redes neuronales convolucionales emplea parejas de capas convolucionales (extracción de características) seguidas de capas *"Max-Pooling"* (reducción de la dimensionalidad, *downsampling, ver* Ilustración 5). La capa de *pooling* selecciona las características más importantes de entre un conjunto de características vecinas. Aplicado a imágenes, selecciona el píxel con valor máximo de cada sub-matriz de vecinos. El tamaño de estas sub-matrices se determina a través de un parámetro, de igual forma que el *stride* (o paso) entre un *pooling* y otro. Reduciendo la dimensionalidad, logramos generalizar el problema resumiendo las características más importantes en ella.

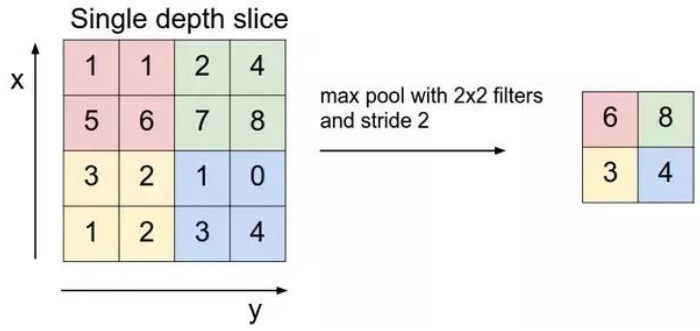


Ilustración 5. Max pooling de tamaño 2x2 y stride 2. Fuente: What is Max Pooling in CNNs. [5]

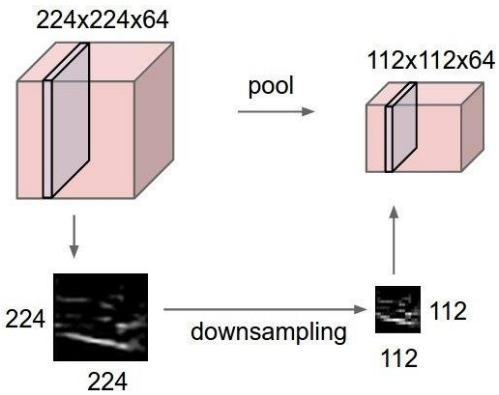


Ilustración 6. Efecto del Max pooling sobre la dimensionalidad. Fuente: What is Max Pooling in CNNs. [5]

Tras extraer las características más importantes y reducir la dimensionalidad de los datos mediante un cierto número de parejas convolucional-pooling, finalmente se emplea un conjunto de capas densas que se ocuparán de realizar la predicción. Puesto que las capas convolucionales trabajan con objetos tridimensionales y las densas con unidimensionales, será necesario aplicar una capa de conversión llamada *"flatten"* entre ambas. Usualmente, en capas intermedias, se utiliza la función de activación *"ReLU"* (*Rectified Linear Unit*) definida como $f(x) = max(0, x)$ tanto en capas convolucionales como densas.

Las peculiaridades vienen en la última capa, donde entra en juego el objetivo que se trate de lograr con el modelo. Para problemas de clasificación y segmentación, dicha capa deberá tener un número de neuronas igual al número de clases a predecir, a modo de probabilidades o distribución de probabilidades. En problemas de clasificación binaria podrá implementarse con una única neurona de salida, viéndose ésta como la probabilidad de la clase positiva. [4] Cabe destacar que, en problemas de clasificación, esta última capa se implementa mediante una capa densa, mientras que en segmentación se hace a través de una convolución de filtro y *stride* unidad.

Consecuentemente, en función de la salida se deberá seleccionar una función de error (*"loss")* apropiada. En la Tabla 1 se muestra un resumen con las configuraciones más comunes de la última capa y bajo qué circunstancias utilizarlas. En la Ilustración 7 se puede observar la arquitectura normal de una CNN.

Tipo de Problema	Activación en última capa	Función de Error
Clasificación Binaria	Sigmoid	Binary_crossentropy
Multiclase, cl. única	Softmax	Categorical_crossentropy
Multiclase, cl. múltiple	Sigmoid	Binary_crossentropy
Regresión Arbitraria	-	MSE
Regresión 0-1	Sigmoid	MSE o Binary_crossentropy

Tabla 1. Elección correcta de Funciones de Activación y Error. Fuente: Deep Learning with Python. [4]

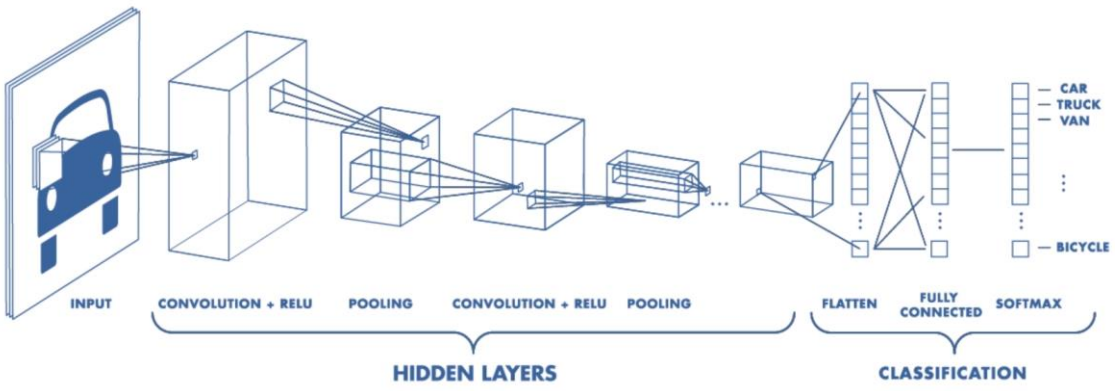


Ilustración 7. Arquitectura CNN. Fuente: What Are Convolutional Neural Networks? [6]

2.1.2 Fully Convolutional Networks (FCNN)

Las *Fully Convolutional Neural Networks* son ampliamente utilizadas en la segmentación de imágenes. Se diferencian de las CNN en que no utilizan capas densas, sino que la salida es producida directamente por capas convolucionales.

La clave consiste en substituir la sección *fully-connected* por convoluciones de tamaño 1x1. De esta manera, la salida producida consistirá en un conjunto de predicciones independientes para cada píxel [7]. De igual manera que ocurría con las CNN, en este caso el número de filtros dependerá del número de clases de nuestro problema.

La problemática añadida a la segmentación con redes convolucionales reside en la reducción de la dimensionalidad. El *downsampling* es un proceso necesario para lograr extraer las características más importantes, pero provoca que el tamaño de los datos se vea reducido. Por ello, sería imposible realizar una predicción para todos los píxeles de una imagen utilizando únicamente capas convolucionales y *max-pooling*.

Para solucionarlo, y tras haber reducido los datos, se hace uso de las llamadas deconvoluciones (o convoluciones transpuestas), que actúan en la dirección inversa a la de las convoluciones. El resultado es una arquitectura similar a la de un autoencoder. En este caso, sin embargo, lo que se trata de reconstruir no es la imagen original sino una representación abstracta la misma. Los píxeles son una clasificación del objeto del que forman parte (ver Ilustración 8 e Ilustración 9)

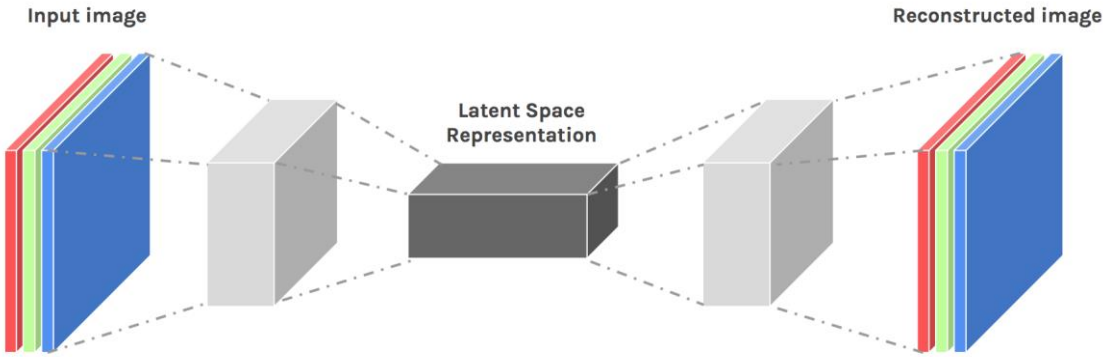


Ilustración 8. Autoencoder Convolucional. Fuente: Autoencoders, Introduction and Implementation in TF. [8]

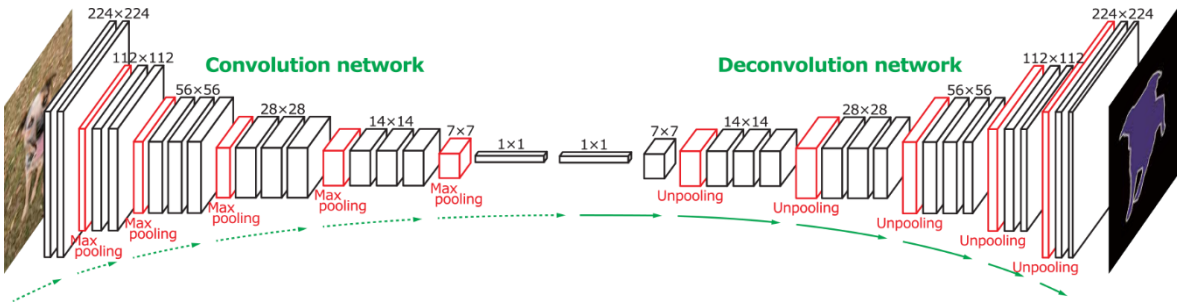


Ilustración 9. Ejemplo de arquitectura FCNN. Fuente: Learning Deconvolution Network for Semantic Segmentation. [9]

2.1.3 U-Net

Las redes U-Net pueden ser vistas como una evolución de las FCNN. Se basan en la misma arquitectura, pero tienen la peculiaridad que hacen uso de la salida de las convoluciones para agregar información a la deconvolución. El resultado es una red quasi simétrica, donde tras cada fase de *upsampling* se concatenan todos los canales previos al *downsampling* correspondiente.

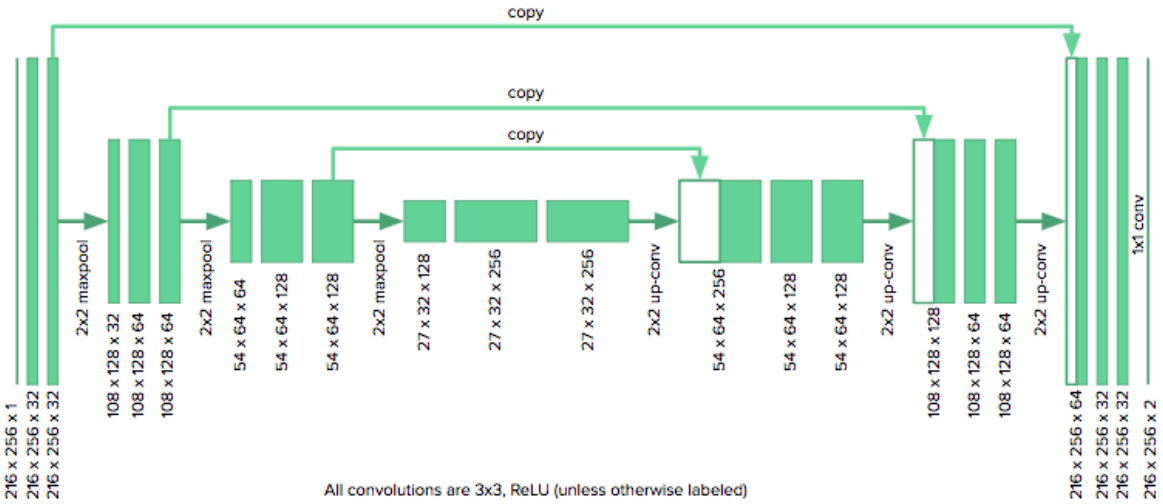


Ilustración 10. Ejemplo de Arquitectura U-Net. Fuente: Cardiac MRI Segmentation. [10]

Como se puede observar en el ejemplo de la Ilustración 10, tras la fase de *upsampling* se obtiene una estructura del mismo tamaño que la imagen original. El número de canales de salida, dos en este caso, de igual manera que las CNN convencionales dependerán del número de clases a predecir.

2.2 Normalización

Uno de los principales problemas del *machine learning* aplicado a imágenes, son las diferencias de energía entre imágenes y entre zonas de la propia imagen. Éstas pueden venir de cambios en la óptica o dispositivo de captura, espacio de captura (interior/exterior), posición de sol, estado del clima, iluminación, etc. Los modelos son muy sensibles a estas diferencias, y si no se tratan adecuadamente es probable que no se logren resultados óptimos.

Para mitigar los efectos de este tipo de problemas, a menudo es necesario aplicar técnicas de normalización. Una de las principales razones de la obtención de malos resultados es la saturación generalizada de las funciones de activación. El objetivo de la normalización consiste en equilibrar los rangos de valores para los píxeles de los datos de entrada. [11] Existe multitud de técnicas, algunas de ellas aplicables durante el preprocesado de los datos, y otras implícitas en los modelos. En cualquier proceso de modelización con imágenes, usualmente es importante trasladar los datos del rango [0,255] a [0,1] para evitar que los pesos internos se vuelvan extremadamente grandes (o pequeños) y por lo tanto inestables.

2.2.1 Batch Normalization

El *"batch normalization"* es una técnica mediante la cual se trata de normalizar los datos utilizando una capa de la propia red. A partir de los datos de cada *batch*[2] en el entrenamiento, se busca modelizar la media y desviación estándar de la distribución, con la que posteriormente se normalizarán todos los datos. El resultado es que se maximiza el contraste de cada imagen normalizando a partir de la distribución en las muestras de entrenamiento.

Esta técnica consta de dos fases, pues se aplica de distinta forma durante el entrenamiento y la inferencia:

- En la primera de ellas, durante el entrenamiento del modelo, la normalización se realizará teniendo en cuenta la media y varianza de los datos del propio *batch*:
    - Se calcula la media y varianza del *batch*

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{Batch mean}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \qquad \text{Batch variance}$$

Ecuación 1. Media y varianza del batch. Fuente: BN theory and how to use in TF. [11]

    - Se normalizan los datos del batch a partir de éstas

$$\overline{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Ecuación 2. Normalización del batch. Fuente: BN theory and how to use in TF. [11]

    - Se actualizan la media y varianza generales teniendo en cuenta las calculadas para el *batch* actual.
- En la segunda fase, durante la inferencia, se debe realizar una normalización equivalente a la del entrenamiento. De esta forma, aunque no se introduzca una gran cantidad de imágenes en *batch*, se normalizará utilizando la media y varianza aprendidas y fijadas durante el entrenamiento.

2.2.2 Normalización Divisiva Generalizada (GDN)

Se trata de una transformación parametrizada no lineal adecuada para normalizar imágenes naturales. A diferencia del método anterior, éste trata de maximizar el contraste de manera local, normalizando por ventanas en lugar de tomar la imagen como unidad de datos sobre la que aplicarlo.

El proceso, básicamente, consiste en aplicar una convolución a la imagen con la que obtener la energía en cada ventana. Los valores en cada píxel se dividen por el resultado de la convolución, que representa la actividad agrupada de sus vecinos. [12] En ventanas

[2] ***Batch*:** Subconjunto de los datos de entrada que se introduce en el modelo en cada iteración.

con baja actividad este valor será pequeño, con lo que el resultado de la división hará incrementar el valor del píxel (y con él el contraste). En ventanas con elevada actividad el valor será grande, con lo que el valor del píxel se decrementará (y de nuevo, se incrementará el contraste).

$$y_i = \frac{z_i}{(\beta_i + \sum_j \gamma_{ij} |z_j|^{\alpha_{ij}})^{\varepsilon_i}} \qquad siendo \qquad z = Hx$$

Ecuación 3. Ecuación de GDN. Fuente: Density Modelling of Images Using a GDN. [12]

Según [12], "Los vectores β, ε, así como las matrices H, γ, α son parámetros del modelo. La complejidad total es de $2N + 3N^2$, donde N es la dimensionalidad del espacio de entrada". H es el filtro de convolución que, en modelos convolucionales, realiza la transformación previa a la normalización de la imagen de entrada (x). La ecuación puede ser vista como una generalización de múltiples métodos de modelización estadística de la distribución. Se puede interpretar una inspiración de un modelo "pseudo" lineal, pues el denominador es la suma de la media ajustada (β_i) con los productos de parámetros (γ_{ij}) también a ajustar y un factor que representa la energía del vecindario en cuestión, calculado como el valor absoluto de los valores del vecindario elevados a un parámetro α_{ij}. Cabe comentar que otro objetivo de β_i es el de evitar que el denominador sea nulo. Finalmente, el resultado se ajusta exponencialmente mediante otro parámetro (ε_i). Con todo ello, obtenemos una estimación de la energía del vecindario en relación con la distribución aprendida de los datos. Tensor Flow implementa una versión simplificada de ésta, en la que $\alpha \equiv 2, \varepsilon \equiv 1/2$ por razones computacionales. [13]

La principal ventaja que presenta es que se trata de un método de normalización no supervisado que normaliza cada imagen por ventanas en sí mismas, sin depender explícitamente del resto de la imagen, batch o dataset. Los resultados de aplicar esta normalización son imágenes muy similares a las naturales, reduciendo especialmente las diferencias más notables de iluminación entre zonas de la propia imagen a la vez que se maximiza el contraste.

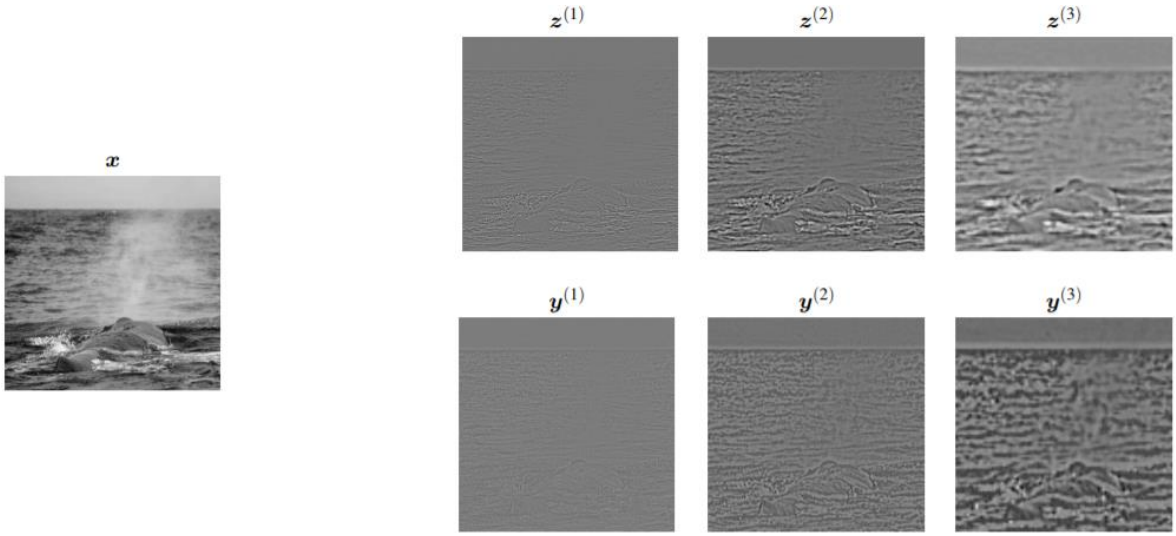


Ilustración 11. Ejemplo de normalización local GDN sobre descomposición por pirámide Laplaciana. Fuente: Perceptual Image QA using a normalized Laplacian Pyramid [14]

En la Ilustración 11 “se pueden observar tres escalas de descomposición mediante pirámide Laplaciana ($z = Hx$) de la imagen original (x). Se ha realizado un *upsample* de las descomposiciones con fines de visualización. y son las mismas imágenes con el contraste normalizado localmente” [14] mediante la técnica GDN. La defínción en los detalles del agua tras la nube de agua pulverizada que se pierden con la descomposición, son recuperados gracias a la normalización local de contraste.

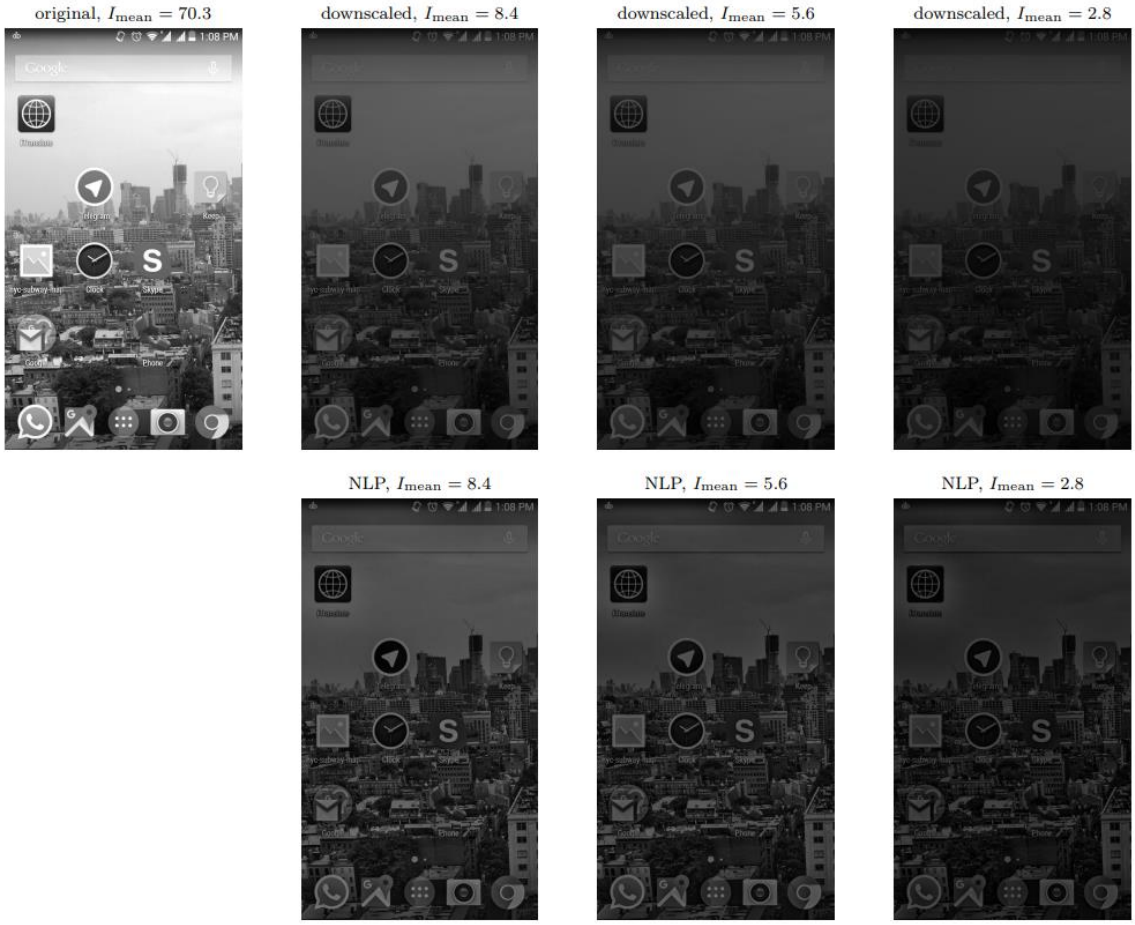


Ilustración 12. GDN sobre distintas intensidades de brillo en pantalla móvil. Fuente: Perceptually Optimized Image Rendering. [15]

En la Ilustración 12 se muestra la imagen renderizada en un dispositivo móvil a distintas intensidades de consumo de energía. La fila superior muestra un escalado lineal en el brillo, mientras que la fila inferior muestra las mismas imágenes procesadas mediante normalización GDN. Como se percibe, aunque las imágenes de la misma columna provocan iguales consumos de energía en la pantalla, la cantidad de detalle perceptible es significativamente diferente. [15]

En esencia, esta transformación aporta unas características muy útiles para la detección de patrones en imagen por convoluciones. Logra acentuar los detalles de zonas más sombrías, donde la resolución y texturas son menos perceptibles. Cabe tener en cuenta en la diferencia en coste computacional de este método frente a otros, donde el tiempo de cálculo necesario es inferior (como el *batch normalization*).

3 Conjunto de Datos

En este apartado se hablará de los conjuntos de datos que se utilizarán para entrenar y validar los modelos de prueba. Puesto que se pretende implementar tanto modelos de clasificación como de segmentación, ha sido necesario obtener datos etiquetados para ambas finalidades. Todos los data sets a emplear son accesibles online de forma libre y gratuita. Constan de una amplia aceptación, dado su uso en múltiples artículos publicados.

3.1 CIFAR-10

Se trata de una colección de 60.000 pequeñas imágenes, donde cada una de ellas contiene un objeto o ser vivo de entre 10 tipos distintos. A cada una de las imágenes, le corresponde una etiqueta que identifica el elemento que aparece en ella de entre las 10 clases. Todas ellas tienen un tamaño de 32x32px y constan 3 capas de color (RGB). [16]

El conjunto se encuentra perfectamente balanceado entre clases, puesto que contiene 6.000 imágenes de cada una. Además, en su propia web oficial los autores aseguran que son mutuamente exclusivas (cada imagen contiene un objeto correspondiente a una sola clase). El total de ellas se proporciona dividido de origen en dos subconjuntos, uno de entrenamiento y otro para test, los cuales contienen 5.000 y 1.000 imágenes de cada tipo, respectivamente. El objetivo del dataset, por tanto, es el de clasificar las imágenes dentro de una de las 10 posibles clases (Ilustración 13).

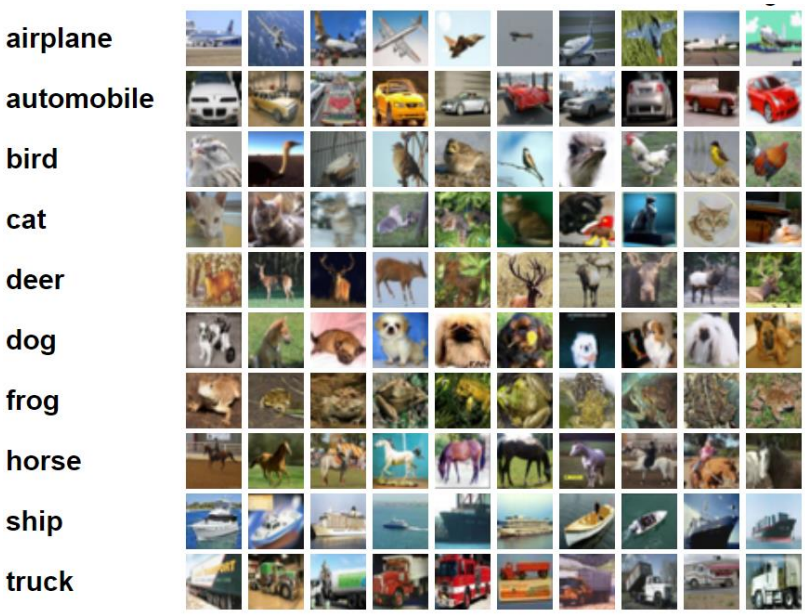


Ilustración 13. Ejemplos de imágenes etiquetadas de CIFAR-10. Fuente: CIFAR-10 dataset [16]

Existe una segunda versión del dataset, llamado CIFAR-100, donde se incluyen imágenes de las mismas características correspondientes a 100 clases distintas. Se proporcionan 600 imágenes por clase, de las que 500 son de entrenamiento y 100 de test.

3.2 INRIA Aerial Image Labeling Dataset

El problema que se plantea resolver con este dataset es el de la segmentación automática de imágenes aéreas a nivel de píxel, con el objetivo de detectar los edificios mostrados en ellas. Se proporcionan dos subconjuntos de imágenes que cubren un total de 810 km^2, uno de ellos para entrenamiento y otro para test (405 km^2 cada uno). Las imágenes tienen un tamaño de 5000x5000 píxeles con una resolución espacial de 0.3m y tres capas de color (RGB). Además, han sido previamente ortorectificadas[3]. [17]

Puesto que se trata de un problema de segmentación semántica binaria (edificio o no-edificio), la segmentación de referencia (*ground truth)* proporcionada poseerá dos clases. Las etiquetas, al igual que las imágenes originales, se obtienen en formato TIFF. Las imágenes originales tienen tres capas 0-255, representando los tres espectros de color RGB. En el caso de las etiquetas se presentan como imagen de una sola capa, siendo 0 el valor para los píxeles de la clase negativa (no-edificio) y 255 para los píxeles de la clase positiva (edificio).

Ilustración 14. Imágenes y ground truth (INRIA Aerial Image Labeling Dataset). Fuente: INRIA-AIL Dataset [17]

[3] **Ortorectificación:** Proceso mediante el que se eliminan los efectos producidos por la perspectiva de captura de las imágenes o por la naturaleza topográfica del terreno, con el fin de conseguir una escala constante de representación donde los objetos se visualicen en su posición y tamaño reales. [20]

El dataset está disponible públicamente en la web oficial del INRIA (*Institut National de Recherche en Informatique et en Automatique*) a modo de concurso, por lo que tan solo se proporciona la segmentación correcta para el conjunto de entrenamiento. Las etiquetas del conjunto de prueba se mantienen privadas con el fin de verificar el funcionamiento de los modelos desarrollados por los participantes. Existe una tabla de clasificación pública a partir de los resultados obtenidos por los concursantes.

La finalidad del trabajo original [17] era la de estudiar si los métodos de segmentación semántica se podían generalizar a cualquier ciudad. Por ello, en ambos subconjuntos se proporcionan imágenes de diferentes ciudades con densidades de población, tomadas en épocas distintas del año. Existen 36 imágenes de cada una de las 5 ciudades, correctamente identificadas. En el sitio web oficial se sugiere utilizar las 5 primeras imágenes de cada ubicación para fines de validación.

4 Metodología de trabajo

Este proyecto ha sido íntegramente desarrollado utilizando el lenguaje de programación Python 3.7. Se trata de un lenguaje interpretado ampliamente extendido en el mundo de la ciencia de datos, gracias a la inmensa cantidad de librerías que facilitan en gran medida el tratamiento de datos y el desarrollo de modelos.

El desarrollo de los modelos se ha realizado mediante la extendida librería "*Tensor Flow*" para Python, en su versión 1.13.1 con aceleración de GPU (apoyada en NVIDIA CUDA 10.0). Esta librería procesa los datos a través de tensores, una unidad de información especialmente diseñada para albergar y operar con datos multidimensionales transparentemente para el usuario.

La versión para GPU de Tensor Flow es compatible con procesadores CUDA desarrollados por NVIDIA, lo que permite realizar la computación en GPUs compatibles. Una GPU moderna media puede tener más de 100 veces el número de procesadores respecto una CPU puntera. Esta ventaja reduce significativamente el tiempo de cálculo necesario para entrenar modelos computacionalmente complejos. El desarrollo y pruebas tempranas de los modelos se han realizado sobre un ordenador portátil MSI GP62MVR-7RF con las siguientes características:

- Procesador Intel i7-7700HQ (4 cores, 8 threads, 3.8 GHz, 6MB cache)
- 8 GB RAM DDR4 2133 MHz y almacenamiento SSD
- NVIDIA GTX1060 3GB (CUDA capability 6.1)

El entrenamiento final de los modelos se ha realizado sobre un servidor de computación (disponibilidad parcial) con las siguientes características:

- Procesador 2 x Intel Xeon E5-2600v4 (14 cores, 3.2GHz, 35MB cache)
- 128GB RAM DDR4 2400 MHz y almacenamiento SSD
- 4 x NVIDIA TESLA M60 (CUDA capability 5.2)

Desarrollo de los modelos

A continuación, se describirán las arquitecturas de los modelos desarrollados. Puesto que el objetivo principal del proyecto es el de utilizar la normalización mediante GDN y compararla con otros métodos, se ha planteado desarrollar tres variantes para cada modelo de clasificación o segmentación semántica:

- Una primera variante, llamada "BASE", no utilizará normalización de ningún tipo en el modelo. Aunque no es recomendable trabajar con imágenes sin aplicar normalización, servirá como línea-base con la que comparar el resto de los modelos.
- La segunda variante, llamada "BN", aplicará normalización mediante *"Batch Normalization"*.
- La tercera variante, llamada "GDN", aplicará normalización de las imágenes mediante el método *"Generalized Divisive Normalization*".

4.1 Clasificación

El modelo desarrollado para la clasificación de los *datasets* CIFAR-10 y CIFAR-100 [16] consiste en una CNN con cuatro capas convolucionales y sus respectivas "*maxpooling*". Tras la capa *"flatten"* se han aplicado tres capas *"fully connected"* que producen una salida del tamaño del número de clases (10 y 100, respectivamente). Se ha aplicado *"ReLU*" como función de activación en las capas intermedias, *"Zero-Padding"* en todas las convoluciones, así como la función *"softmax"* para la capa final. La función de error a optimizar es una *"cross-entropy"*. En los modelos con normalización se han colocado capas de *"batch normalization"* o *"GDN"*, según corresponda, tras las capas convolucionales.

CNN BASE para CIFAR-10

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	3x3/1	30x30x32	896 / float32
maxpool_1	2x2/1	29x29x32	
conv2d_2	3x3/1	27x27x64	18496 / float32
maxpool_2	2x2/1	26x26x64	
conv2d_3	3x3/1	24x24x128	73856 / float32
maxpool_3	2x2/1	23x23x128	
conv2d_4	3x3/1	21x21x256	295168 / float32
maxpool_4	2x2/1	20x20x256	
flatten		102400	
fc_1		256	26214656 / float32
fc_2		512	131584 / float32
fc_out		10	5130 / float32
			TOTAL: 26.739.786 (106.959.144 bytes)

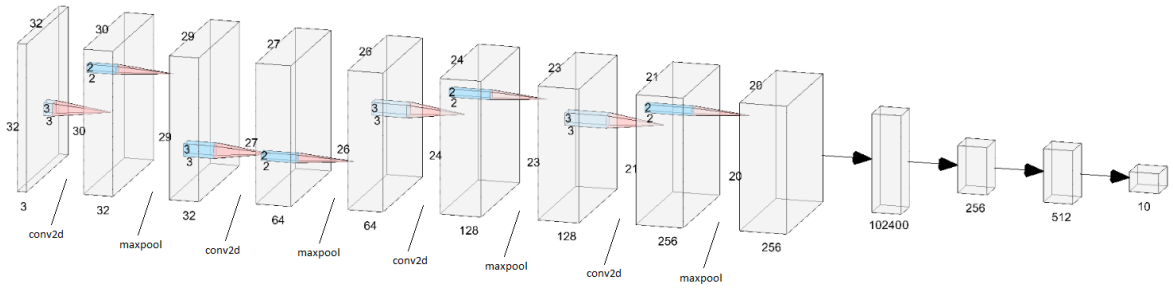


Ilustración 15. CNN BASE CIFAR-10.

CNN BN para CIFAR-10

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	3x3/1	30x30x32	896 / float32
batch_normalization		30x30x32	64 / float32
maxpool_1	2x2/1	29x29x32	
conv2d_2	3x3/1	27x27x64	18496 / float32
batch_normalization		27x27x64	128 / float32
maxpool_2	2x2/1	26x26x64	
conv2d_3	3x3/1	24x24x128	73856 / float32
batch_normalization		24x24x128	256 / float32
maxpool_3	2x2/1	23x23x128	
conv2d_4	3x3/1	21x21x256	295168 / float32
batch_normalization		21x21x256	512 / float32
maxpool_4	2x2/1	20x20x256	
flatten		102400	
fc_1		256	26214656 / float32
fc_2		512	131584 / float32
fc_out		10	5130 / float32

TOTAL: 26.740.746
(106.962.984 bytes)

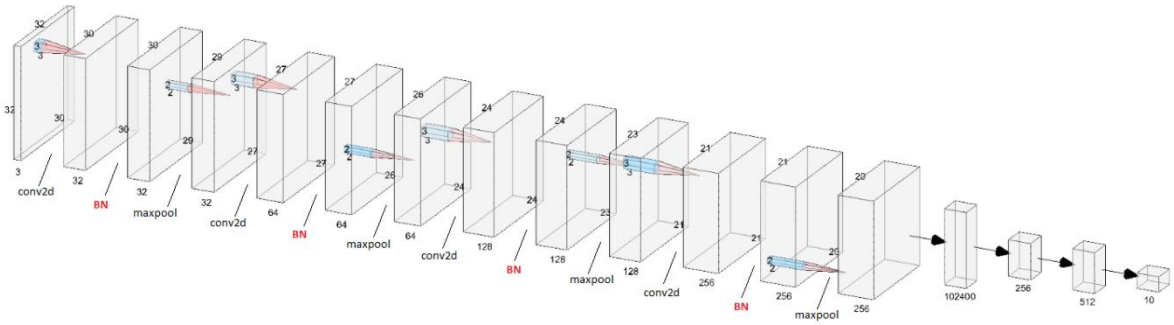


Ilustración 16. CNN BN CIFAR-10.

CNN GDN para CIFAR-10

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	3x3/1	30x30x32	896 / float32
gdn		30x30x32	1056 / float32
maxpool_1	2x2/1	29x29x32	
conv2d_2	3x3/1	27x27x64	18496 / float32
gdn		27x27x64	4160 / float32
maxpool_2	2x2/1	26x26x64	
conv2d_3	3x3/1	24x24x128	73856 / float32
gdn		24x24x128	16512 / float32
maxpool_3	2x2/1	23x23x128	
conv2d_4	3x3/1	21x21x256	295168 / float32
gdn		21x21x256	65792 / float32
maxpool_4	2x2/1	20x20x256	
flatten		102400	
fc_1		256	26214656 / float32
fc_2		512	131584 / float32
fc_out		10	5130 / float32
			TOTAL: 26.827.306 (107.309.224 bytes)

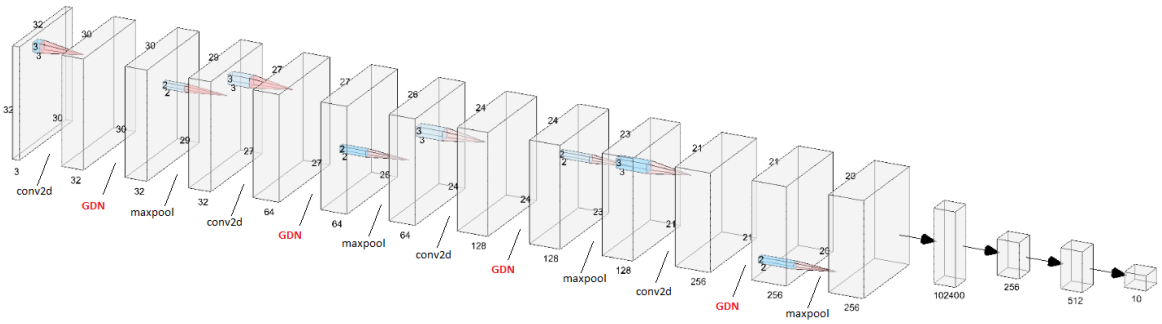


Ilustración 17. CNN GDN CIFAR-10.

CNN BASE para CIFAR-100

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	3x3/1	30x30x32	896 / float32
maxpool_1	2x2/1	29x29x32	
conv2d_2	3x3/1	27x27x64	18496 / float32
maxpool_2	2x2/1	26x26x64	
conv2d_3	3x3/1	24x24x128	73856 / float32
maxpool_3	2x2/1	23x23x128	
conv2d_4	3x3/1	21x21x256	295168 / float32
maxpool_4	2x2/1	20x20x256	
flatten		102400	
fc_1		256	26214656 / float32
fc_2		512	131584 / float32
fc_out		100	51300 / float32

TOTAL: 26.785.956
(107.143.824 bytes)

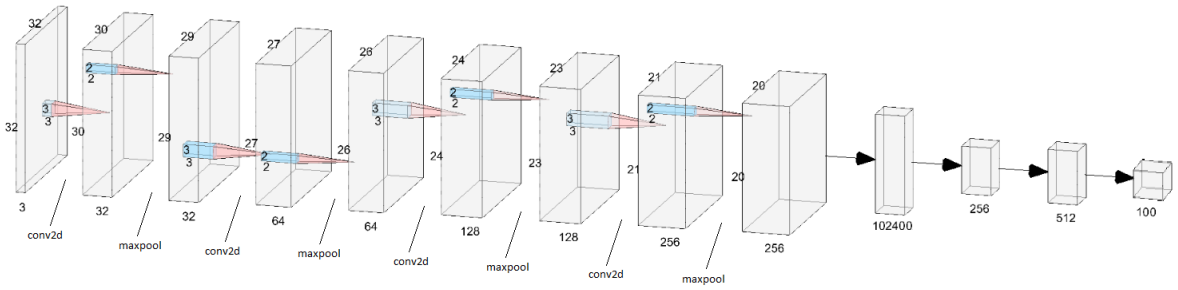


Ilustración 18. CNN BASE CIFAR-100.

CNN BN para CIFAR-100

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	3x3/1	30x30x32	896 / float32
batch_normalization		30x30x32	64 / float32
maxpool_1	2x2/1	29x29x32	
conv2d_2	3x3/1	27x27x64	18496 / float32
batch_normalization		27x27x64	128 / float32
maxpool_2	2x2/1	26x26x64	
conv2d_3	3x3/1	24x24x128	73856 / float32
batch_normalization		24x24x128	256 / float32
maxpool_3	2x2/1	23x23x128	
conv2d_4	3x3/1	21x21x256	295168 / float32
batch_normalization		21x21x256	512 / float32
maxpool_4	2x2/1	20x20x256	
flatten		102400	
fc_1		256	26214656 / float32
fc_2		512	131584 / float32
fc_out		100	51300 / float32
			<u>TOTAL:</u> 26.786.916 (107.147.664 bytes)

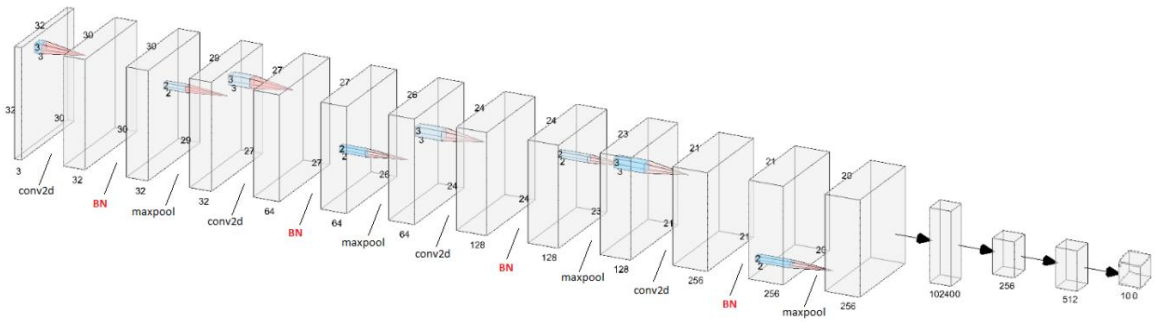


Ilustración 19. CNN BN CIFAR-100.

CNN GDN para CIFAR-100

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	3x3/1	30x30x32	896 / float32
gdn		30x30x32	1056 / float32
maxpool_1	2x2/1	29x29x32	
conv2d_2	3x3/1	27x27x64	18496 / float32
gdn		27x27x64	4160 / float32
maxpool_2	2x2/1	26x26x64	
conv2d_3	3x3/1	24x24x128	73856 / float32
gdn		24x24x128	16512 / float32
maxpool_3	2x2/1	23x23x128	
conv2d_4	3x3/1	21x21x256	295168 / float32
gdn		21x21x256	65792 / float32
maxpool_4	2x2/1	20x20x256	
flatten		102400	
fc_1		256	26214656 / float32
fc_2		512	131584 / float32
fc_out		100	51300 / float32

<u>TOTAL:</u> 26.873.476
(107.493.904 bytes)

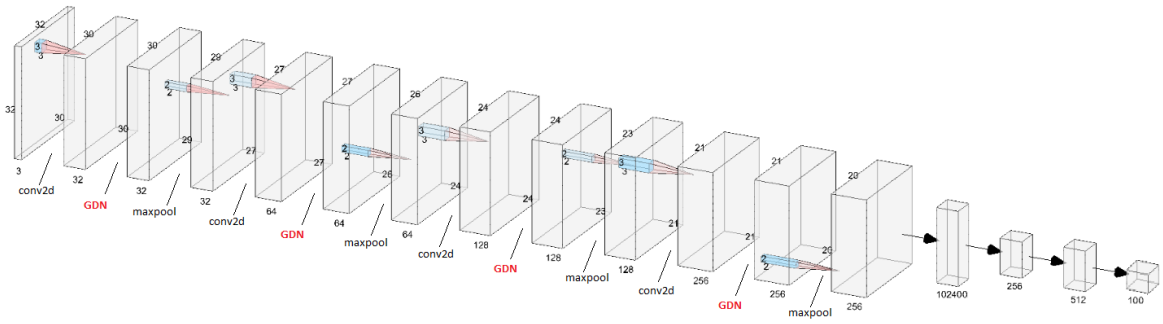


Ilustración 20. CNN GDN CIFAR-100

4.2 Segmentación

El modelo desarrollado para la segmentación del dataset *"INRIA Aerial Image Labeling"* [17] consiste en una FCNN con arquitectura U-Net. Consta de cuatro capas convolucionales (*"Zero-Padding"*) con sus respectivas *maxpooling* para realizar el *downsampling*. Tras éstas, se ha colocado una capa de "*dropout"* para tratar de mitigar el sobreajuste. A continuación, cuatro capas de convolución transpuesta realizarán la tarea de *upsampling*, concatenando a la salida de cada una de ellas la correspondiente salida de las capas convolucionales anteriores. Para acabar, una última convolución con filtro y *stride* unidad, a la que se le aplicará una activación mediante función sigmoide, proporcionará tras el redondeo a la parte entera la predicción de probabilidad de la clase positiva. Las activaciones de las capas intermedias serán *"ReLU"*, y la función de error a optimizar *"binary cross-entropy"*.

Un análisis en los datos de entrada reveló que el número de píxeles de clase positiva se encuentra notablemente desbalanceado frente a la contraria. En los datos de entrenamiento (imágenes 6 a 36 de cada ciudad, como se sugiere [17]) el porcentaje de clases positivas es tan solo del 16.11% del total de píxeles.

Para compensar este desbalanceo se ha utilizado una variante de la función de error que penaliza los errores en la clase positiva, multiplicándolos por un peso parametrizado (pw). Aunque no existe ningún consenso sobre la forma de determinar este factor, atendiendo a la su implementación en *"TensorFlow"* [18] y debates en comunidades online [19] se ha decidido calcular como $pw = numNegativePx/numPositivePx$. Puesto que el parámetro multiplica únicamente los errores en la clase positiva, si el número de muestras positivas es inferior, entonces $pw > 1$ y se penalizarán más éstos mismos. Al contrario, si el número de muestras positivas es superior, entonces $pw < 1$ y se disminuirá la importancia de errores en la clase positiva. De esta forma, ambas clases intervendrán igualmente en el cómputo global del error a minimizar.

Dada la gran cantidad de memoria necesaria para procesar el *dataset*, y puesto que no sería viable cargarlo completamente en RAM, se han implementado generadores que realizan lecturas a disco. Cuando se va a procesar un *batch* se realiza la carga de éste en memoria, eliminándose antes de pasar al siguiente, disminuyendo notablemente el consumo de recursos. Eso sí, el uso de generadores produce un incremento en el tiempo necesario para el entrenamiento y validación, debido a los tiempos de acceso y lectura.

Durante el desarrollo de los modelos se experimentaron problemas de memoria al momento de ejecutar los modelos, debido a las limitaciones de la GPU. Fue necesario reducir el tamaño de las imágenes para poder realizar las primeras pruebas, recortando cada una de 5000x5000 a 20 imágenes de 250x250.

Tras esto, se observó que el tiempo necesario para entrenar una época era superior a 5 horas, lo que hacía el problema impracticable (teniendo en cuenta que se querían entrenar tres variantes de cada uno). Se decidió seleccionar los datos de una de las 5 ciudades con el fin de determinar si (al menos) la arquitectura del modelo era viable. Tras

unas pocas épocas con resultados positivos, se optó por reducir aún más el tamaño de las imágenes, aunque esto empeorase el rendimiento de los modelos.

De esta forma, se seccionaron de nuevo las imágenes originales, esta vez a un tamaño de 100x100 por imagen. Gracias a este último ajuste, el tiempo de entrenamiento por época se redujo a unos 45 minutos por época.

Tras obtener los resultados (ver apartado 0), se procedió a estudiar qué tipo de modificaciones se podían hacer para mejorarlos. Aquí se descubrió que, tras las sucesivas reducciones de tamaño de imagen, el tamaño de los filtros no había sido modificado. Un filtro de 32x32 es demasiado grande para una imagen de 100x100; más aún si se aplican utilizando relleno de tipo *"Zero-Padding"*. Con esta configuración, las convoluciones cercanas a los bordes son calculadas con unos pocos de los valores extremos en los filtros. Además, la última de las convoluciones se realiza con filtro mayor a la propia imagen, lo cual carece de sentido.

Además, se observó que las tres variantes de modelo eran más similares de lo que se cabía esperar, pues tan solo se había aplicado una capa de normalización y el número total de parámetros no variaba en más de 20 entre modelos. Así pues, tras revisar la bibliografía de nuevo, se procedió de nuevo a implementar las tres variantes de la U-Net con una configuración más adecuada (filtros más pequeños y normalización tras cada convolución). Al verse reducido el número de parámetros, el tiempo de entrenamiento se redujo de nuevo hasta alcanzar (aproximadamente) media hora por época.

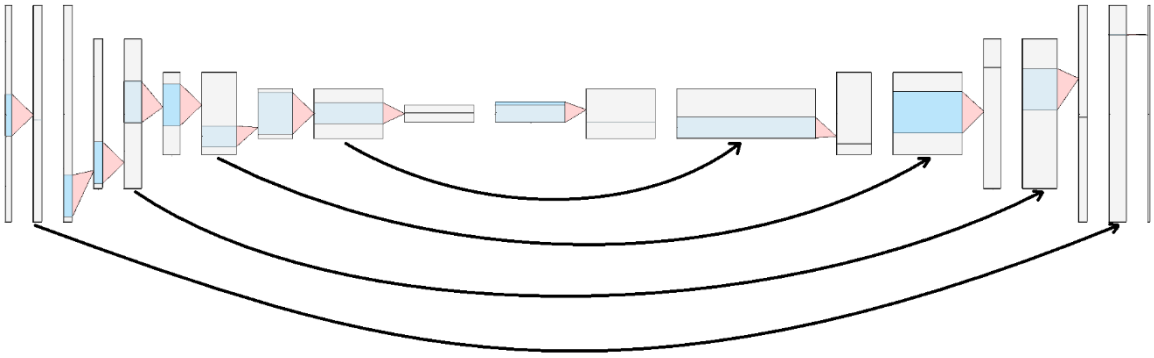


Ilustración 21. Ejemplo de arquitectura U-Net desarrollada para INRIA AIL.

U-Net BASE v1 para INRIA Aerial Image Labelling Dataset

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	32x32/1	100x100x4	12292 / float32
maxpool_1	32x32/1	69x69x4	
conv2d_2	32x32/1	69x69x8	32776 / float32
maxpool_2	32x32/1	38x38x8	
conv2d_3	32x32/1	38x38x16	131088 / float32
maxpool_3	16x16/1	23x23x16	
conv2d_4	32x32/1	23x23x32	524320 / float32
maxpool_4	16x16/1	8x8x32	
dropout	0.5	8x8x32	
conv2d_trans_4	16x16/1	23x23x32	262176 / float32
concat	(conv4)	23x23x64	
conv2d_trans_3	16x16/1	38x38x16	262160 / float32
concat	(conv3)	38x38x32	
conv2d_trans_2	32x32/1	69x69x8	262152 / float32
concat	(conv2)	69x69x16	
conv2d_trans_1	32x32/1	100x100x4	65540 / float32
concat	(conv1)	100x100x8	
conv2d_out	1x1/1	100x100x1	9 / float32
			TOTAL: 1.552.513 (6.210.052 bytes)

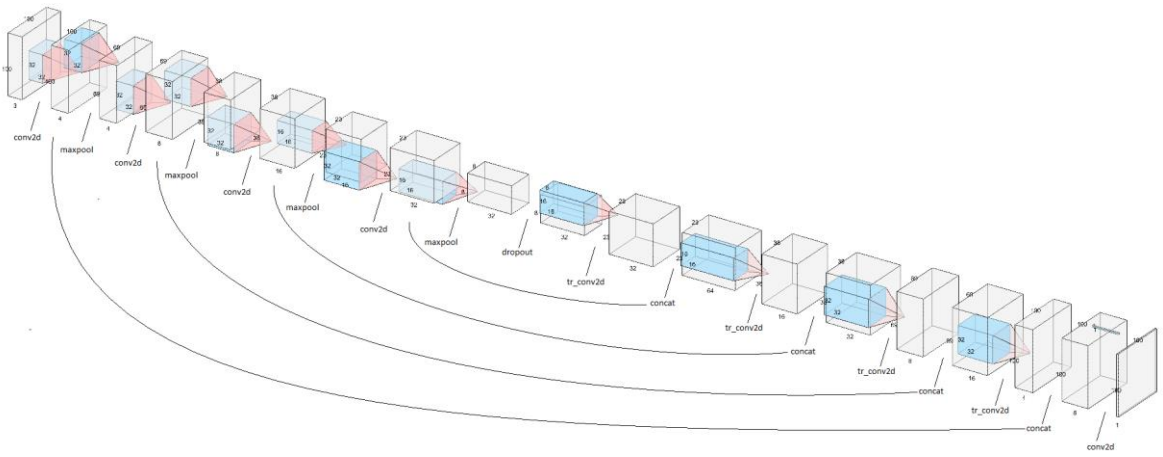


Ilustración 22. U-Net BASE INRIA-AIL.

U-Net BN v1 para INRIA Aerial Image Labelling Dataset

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	32x32/1	100x100x4	12292 / float32
batch_normalization		100x100x4	8 / float32
maxpool_1	32x32/1	69x69x4	
conv2d_2	32x32/1	69x69x8	32776 / float32
maxpool_2	32x32/1	38x38x8	
conv2d_3	32x32/1	38x38x16	131088 / float32
maxpool_3	16x16/1	23x23x16	
conv2d_4	32x32/1	23x23x32	524320 / float32
maxpool_4	16x16/1	8x8x32	
dropout	0.5	8x8x32	
conv2d_trans_4	16x16/1	23x23x32	262176 / float32
concat	(conv4)	23x23x64	
conv2d_trans_3	16x16/1	38x38x16	262160 / float32
concat	(conv3)	38x38x32	
conv2d_trans_2	32x32/1	69x69x8	262152 / float32
concat	(conv2)	69x69x16	
conv2d_trans_1	32x32/1	100x100x4	65540 / float32
concat	(conv1)	100x100x8	
conv2d_out	1x1/1	100x100x1	9 / float32
			TOTAL: 1.552.521 (6.210.084 bytes)

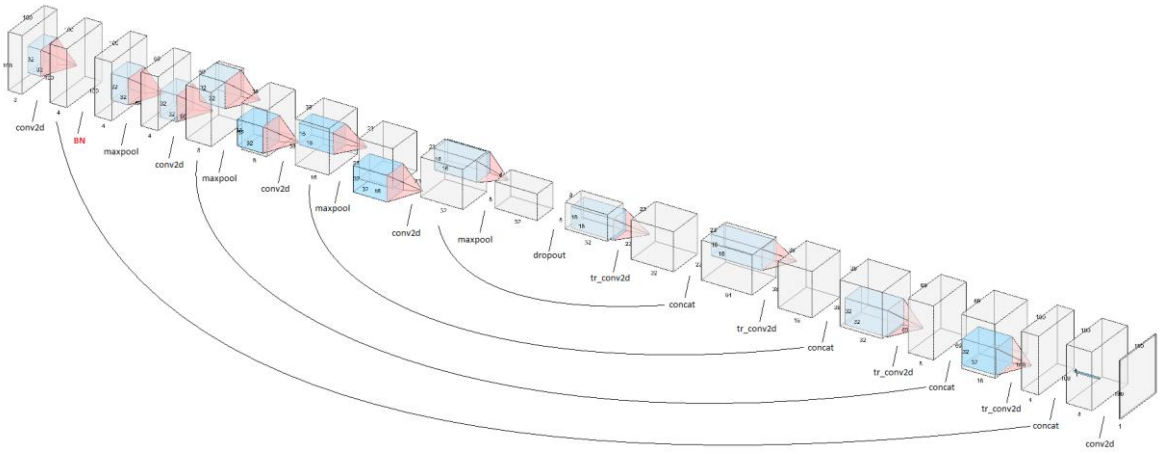


Ilustración 23. U-Net BN INRIA-AIL.

U-Net GDN v1 para INRIA Aerial Image Labelling Dataset

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	32x32/1	100x100x4	12292 / float32
gdn		100x100x4	20 / float32
maxpool_1	32x32/1	69x69x4	
conv2d_2	32x32/1	69x69x8	32776 / float32
maxpool_2	32x32/1	38x38x8	
conv2d_3	32x32/1	38x38x16	131088 / float32
maxpool_3	16x16/1	23x23x16	
conv2d_4	32x32/1	23x23x32	524320 / float32
maxpool_4	16x16/1	8x8x32	
dropout	0.5	8x8x32	
conv2d_trans_4	16x16/1	23x23x32	262176 / float32
concat	(conv4)	23x23x64	
conv2d_trans_3	16x16/1	38x38x16	262160 / float32
concat	(conv3)	38x38x32	
conv2d_trans_2	32x32/1	69x69x8	262152 / float32
concat	(conv2)	69x69x16	
conv2d_trans_1	32x32/1	100x100x4	65540 / float32
concat	(conv1)	100x100x8	
conv2d_out	1x1/1	100x100x1	9 / float32
			TOTAL: 1.552.533 (6.210.132 bytes)

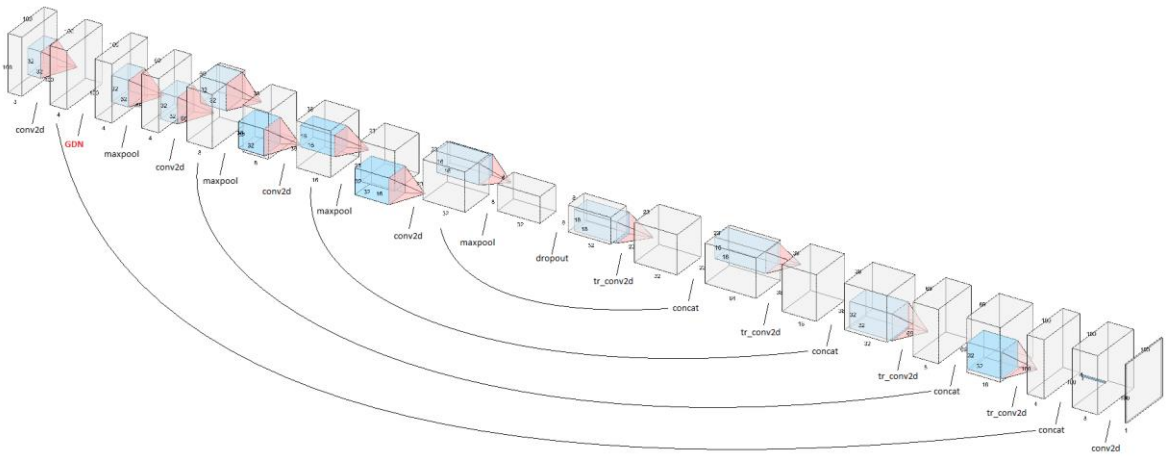


Ilustración 24. U-Net GDN INRIA-AIL.

U-Net BASE v2 para INRIA Aerial Image Labelling Dataset

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	8x8/1	100x100x4	772 / float32
maxpool_1	8x8/1	93x93x4	
conv2d_2	8x8/1	93x93x4	1028 / float32
maxpool_2	8x8/1	86x86x4	
conv2d_3	4x4/1	86x86x8	520 / float32
maxpool_3	4x4/1	83x83x8	
conv2d_4	4x4/1	83x83x16	2064 / float32
maxpool_4	4x4/1	80x80x16	
dropout	0.5	80x80x16	
conv2d_trans_4	4x4/1	83x83x16	4112 / float32
concat	(conv4)	83x83x32	
conv2d_trans_3	4x4/1	86x86x8	4104 / float32
concat	(conv3)	86x86x16	
conv2d_trans_2	8x8/1	93x93x4	4100 / float32
concat	(conv2)	93x93x8	
conv2d_trans_1	8x8/1	100x100x4	2052 / float32
concat	(conv1)	100x100x8	
conv2d_out	1x1/1	100x100x1	9 / float32
			TOTAL: 18.761 (75.044 bytes)

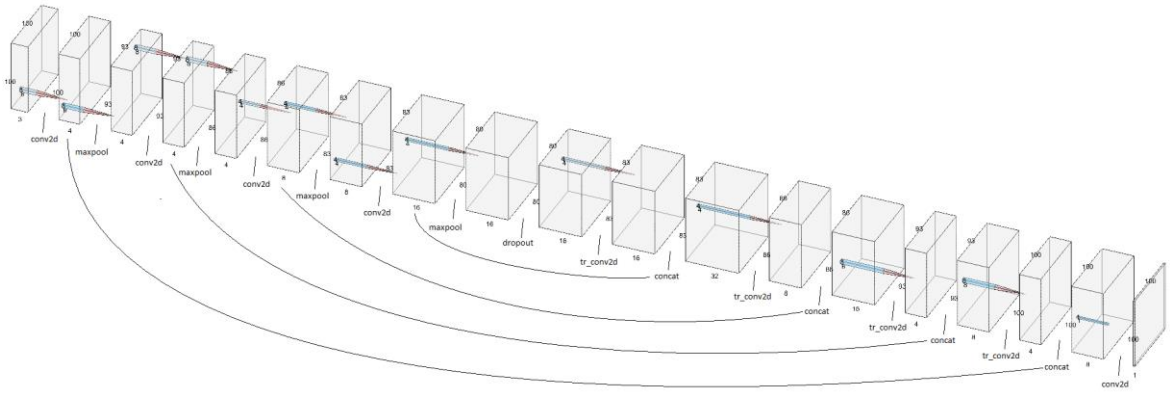


Ilustración 25. U-Net BASEv2 INRIA-AIL.

U-Net BN v2 para INRIA Aerial Image Labelling Dataset

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	8x8/1	100x100x4	772 / float32
batch_normalization_1		100x100x4	8 / float32
maxpool_1	8x8/1	93x93x4	
conv2d_2	8x8/1	93x93x4	1028 / float32
batch_normalization_2		93x93x4	8 / float32
maxpool_2	8x8/1	86x86x4	
conv2d_3	4x4/1	86x86x8	520 / float32
batch_normalization_3		86x86x8	16 / float32
maxpool_3	4x4/1	83x83x8	
conv2d_4	4x4/1	83x83x16	2064 / float32
batch_normalization_4		83x83x16	32 / float32
maxpool_4	4x4/1	80x80x16	
dropout	0.5	80x80x16	
conv2d_trans_4	4x4/1	83x83x16	4112 / float32
concat	(conv4)	83x83x32	
conv2d_trans_3	4x4/1	86x86x8	4104 / float32
concat	(conv3)	86x86x16	
conv2d_trans_2	8x8/1	93x93x4	4100 / float32
concat	(conv2)	93x93x8	
conv2d_trans_1	8x8/1	100x100x4	2052 / float32
concat	(conv1)	100x100x8	
conv2d_out	1x1/1	100x100x1	9 / float32
			TOTAL: 18.825 (75.300 bytes)

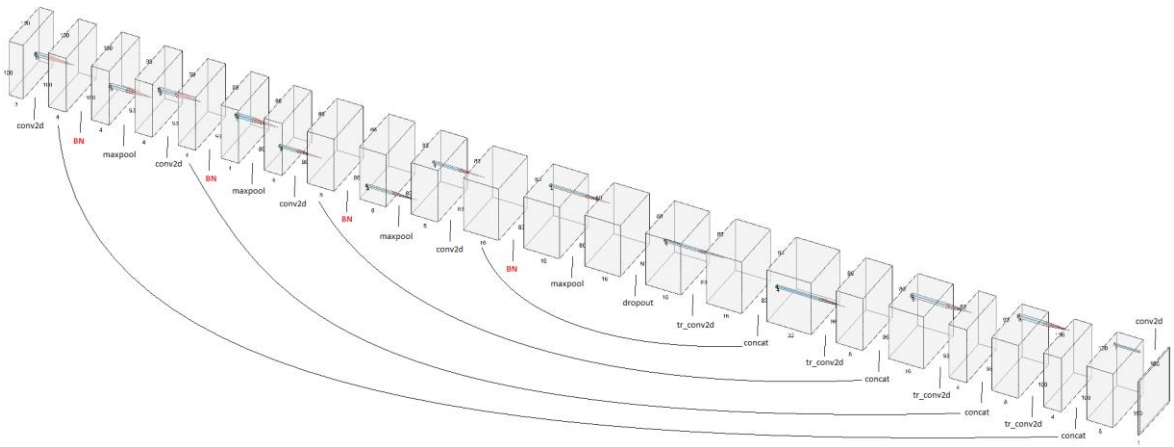


Ilustración 26. U-Net BNv2 INRIA-AIL.

U-Net GDN v2 para INRIA Aerial Image Labelling Dataset

Capa	Filter Size/Stride	Salida WxHxD	Parámetros
conv2d_1	8x8/1	100x100x4	772 / float32
gdn_1		100x100x4	20 / float32
maxpool_1	8x8/1	93x93x4	
conv2d_2	8x8/1	93x93x4	1028 / float32
gdn_2		93x93x4	20 / float32
maxpool_2	8x8/1	86x86x4	
conv2d_3	4x4/1	86x86x8	520 / float32
gdn_3		86x86x8	72 / float32
maxpool_3	4x4/1	83x83x8	
conv2d_4	4x4/1	83x83x16	2064 / float32
gdn_4		83x83x16	272 / float32
maxpool_4	4x4/1	80x80x16	
dropout	0.5	80x80x16	
conv2d_trans_4	4x4/1	83x83x16	4112 / float32
concat	(conv4)	83x83x32	
conv2d_trans_3	4x4/1	86x86x8	4104 / float32
concat	(conv3)	86x86x16	
conv2d_trans_2	8x8/1	93x93x4	4100 / float32
concat	(conv2)	93x93x8	
conv2d_trans_1	8x8/1	100x100x4	2052 / float32
concat	(conv1)	100x100x8	
conv2d_out	1x1/1	100x100x1	9 / float32
			<u>TOTAL:</u> 19.145 (76.580 bytes)

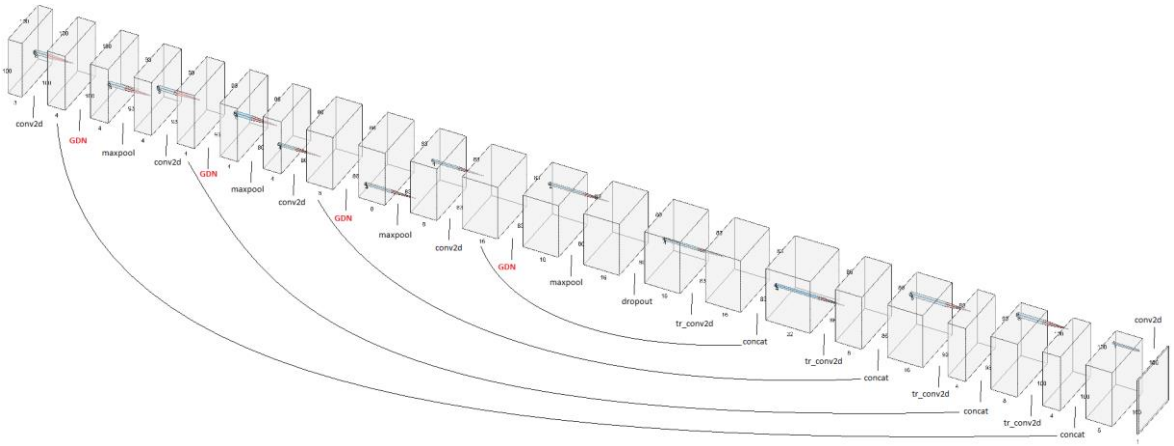


Ilustración 27. U-Net GDNv2 INRIA-AIL.

5 Resultados

En este apartado se presentarán los resultados obtenidos tras el entrenamiento de los modelos. Se comentará el rendimiento obtenido por los modelos, mostrando algunos ejemplos de validación.

Las métricas calculadas para evaluar el rendimiento de los modelos son: *"loss"* (media de la función de error) y *"accuracy"* (tasa de acierto). Adicionalmente, en los modelos de segmentación se ha calculado el "IoU" (*"Intersection Over Union"*). Se trata de una métrica útil en detección de objetos, puesto que tiene en cuenta el efecto del desbalanceo representando la proporción de área de superposición entre los objetos detectados y el *ground truth*, frente a la unión de ambas. Además, esta métrica se utiliza en el leaderboard oficial del dataset [17], por lo que será de utilidad a modo comparativo.

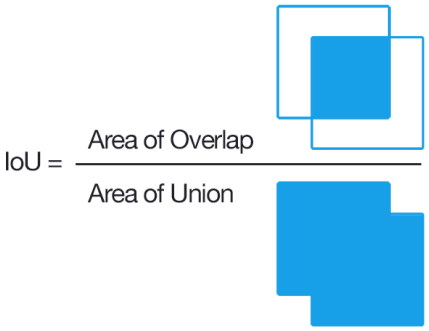


Ilustración 28. Intersection Over Union. Fuente: IoU for object detection. [20]

5.1 Clasificación

A continuación, se presentan los resultados de los modelos de clasificación para el dataset CIFAR-10. Se proporcionan gráficas de evolución de las métricas "*accuracy*" y "*loss*" a lo largo de las épocas y una tabla-resumen con los valores finales de éstas.

CNN BASE CIFAR-10

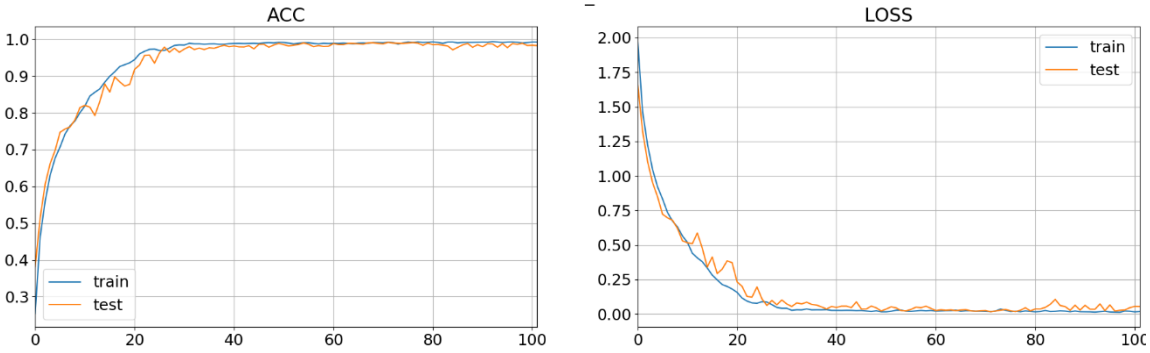


Ilustración 29. Resultados CNN BASE CIFAR-10.

CNN BN CIFAR-10

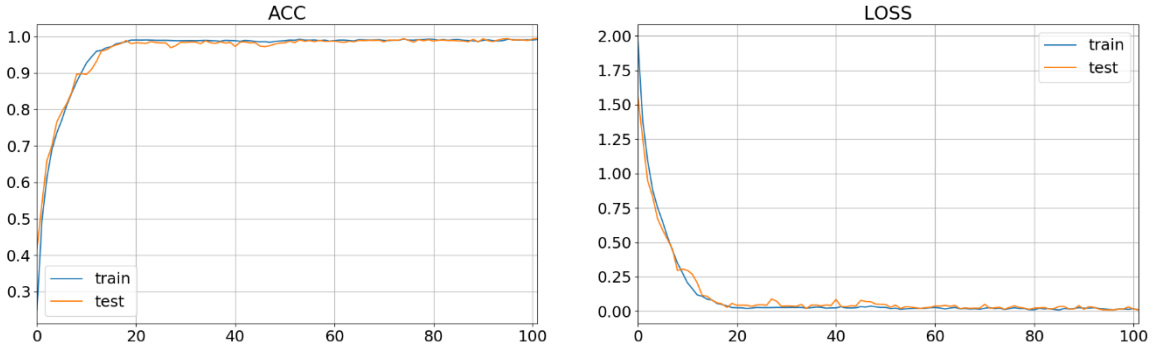


Ilustración 30. Resultados CNN BN CIFAR-10.

CNN GDN CIFAR-10

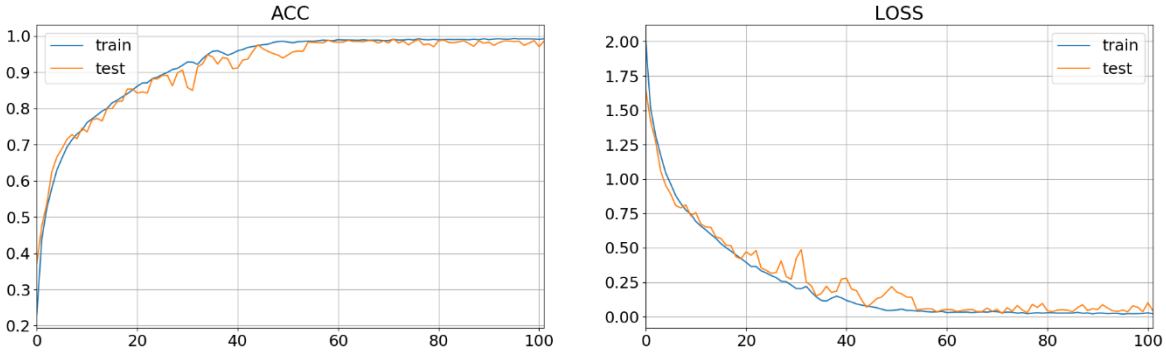


Ilustración 31. Resultados CNN GDN CIFAR-10.

	CNN BASE CIFAR-10		**CNN BN CIFAR-10**		**CNN GDN CIFAR-10**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
ACC	0.99542	0.99759	0.99171	0.99179	0.99248	0.99520
LOSS	0.01036	0.00578	0.01479	0.017391	0.01880	0.07305

Tabla 2. Métricas de resultados de clasificación en CNNs CIFAR-10 (época 100).

Como se puede observar, en los tres casos los modelos han logrado alcanzar un *accuracy* cercano al 100% a la vez que minimizaban casi al máximo el *loss*. Las diferencias se observan más bien en la velocidad de aprendizaje, pues con el mismo learning rate en todos los casos se han necesitado aproximadamente 30, 20 y 45 épocas para alcanzar el máximo rendimiento en los modelos "base", "bn" y "gdn" respectivamente. En este caso, el mejor modelo sería el que aplica normalización mediante "batch normalization", antendiendo meramente al criterio de velocidad de entrenamiento, por no existir diferencias importantes en las métricas alcanzadas.

A continuación, se estudiará el rendimiento de los modelos ante CIFAR-100, el "caso complejo" del *dataset*.

CNN BASE CIFAR-100

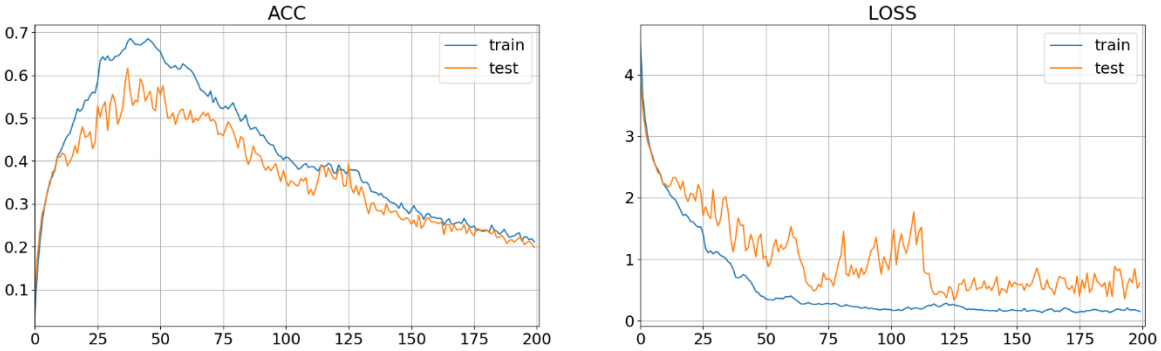


Ilustración 32. Resultados CNN BASE CIFAR-100.

CNN BN CIFAR-100

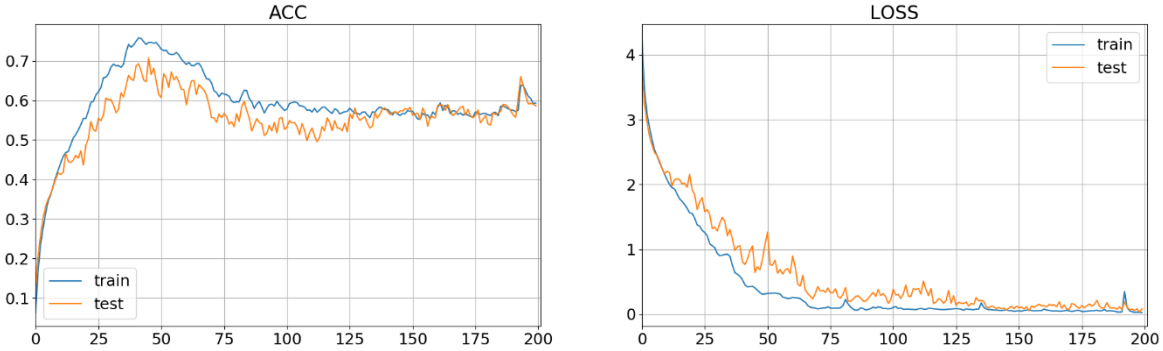


Ilustración 33. Resultados CNN BN CIFAR-100.

CNN GDN CIFAR-100

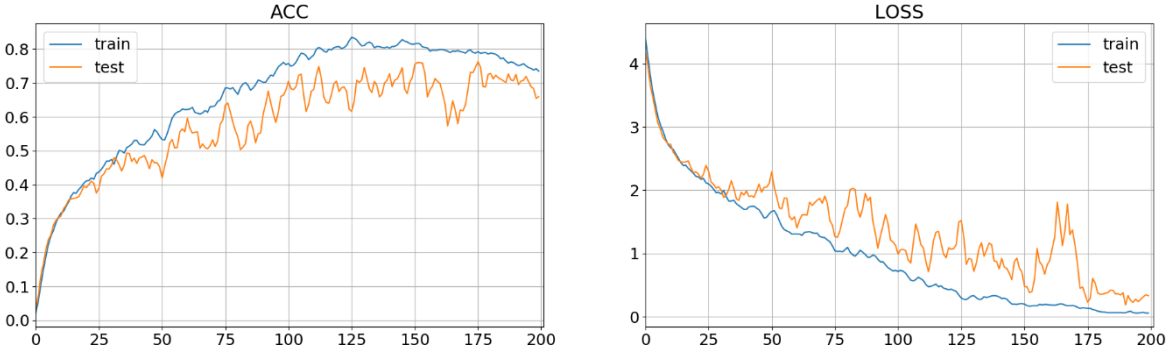


Ilustración 34. Resultados CNN GDN CIFAR-100.

	CNN BASE CIFAR-100		CNN BN CIFAR-100		CNN GDN CIFAR-100	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
ACC	0.21098	0.19829	0.59364	0.58650	0.73488	0.65959
LOSS	0.15027	0.61077	0.02469	0.08343	0.05855	0.332837

Tabla 3. Métricas de resultados de clasificación en CNNs CIFAR-100 (época 200).

	CNN BASE CIFAR-100		CNN BN CIFAR-100		CNN GDN CIFAR-100	
	EPOCH 38		EPOCH 46		EPOCH 176	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
ACC_opt	0.67825	0.61650	0.74701	0.70820	0.79224	0.76249
LOSS_opt	0.83738	1.11437	0.40380	0.64431	0.13189	0.23030

Tabla 4. Métricas de resultados de clasificación, en época de mayor "accuracy" de Test para CNNs CIFAR-100.

En este caso ha sido necesario duplicar el número de épocas de entrenamiento (frente al anterior) para obtener resultados fehacientes suficientes. Se puede observar que, de nuevo, el modelo que ha necesitado más épocas para alcanzar su óptimo ha sido el "gdn" (150 épocas), frente a las otras dos variantes (con aproximadamente 50 épocas en ambos casos).

Sin embargo, para este *dataset* los resultados han sido considerablemente superiores en el caso "gdn", pues ha superado el 79.2% de *accuracy* **en entrenamiento** y el 76.2% **en test**. En el resto de las variantes, apenas se supera el 67.8% y el 74.7% de *accuracy* **en entrenamiento** para los modelos "base" y "bn", respectivamente, para su mejor época.

Un hecho que se extrae de las gráficas es que los tres casos sufren de sobreajuste a partir de cierto punto. El optimizador, en su objetivo de minimizar el error sigue reduciéndolo en los casos con menor entropía causando un sobreajuste en estas muestras, en detrimento de la predicción para aquellas muestras más cercanas a fronteras de decisión. Un ejemplo claro de que esto podría estar ocurriendo lo encontramos en la Tabla 3, si comparamos las métricas de **test** para los casos **BN** y **GDN**. Aunque en el caso GDN el "*loss*" es considerablemente más alto que en el BN, el *"accuracy"* es superior en el modelo GDN. Esto indica que aunque en la GDN el modelo está "menos seguro" con las muestras de test (muestras más cercanas a fronteras de decisión), acierta más que el modelo BN que se encuentra sobreentrenado. Lo mismo ocurre con las métricas del modelo BASE si las comparamos con las presentadas en la Tabla 4 (época 38), donde aunque el *"loss"* sea superior, el *"accuracy"* es mejor por no sufrir sobreentrenamiento.

Si se tiene, por ejemplo, un modelo de clasificación binaria en el que dos muestras producen probabilidades de salida 0.95 y 0.52, para minimizar el error sería posible sobreajustar la primera consiguiendo 0.98 y 0.48, con lo que la entropía cruzada descendería, pero la tasa de acierto también. Esto es lo que parece estar ocurriendo en este caso, por lo que sería necesario seleccionar los pesos del modelo en su punto de entrenamiento óptimo (*early stopping*), o aplicar técnicas de regularización o *data augmentation* para tratar de mejorar su rendimiento.

5.2 Segmentación

A continuación, se presentan los resultados del entrenamiento para los modelos de segmentación. Se proporcionan gráficas de evolución de las métricas "*accuracy*", "*loss*" e "IoU" a lo largo de las épocas y una tabla-resumen con los valores finales de éstas.

U-Net BASE INRIA v1

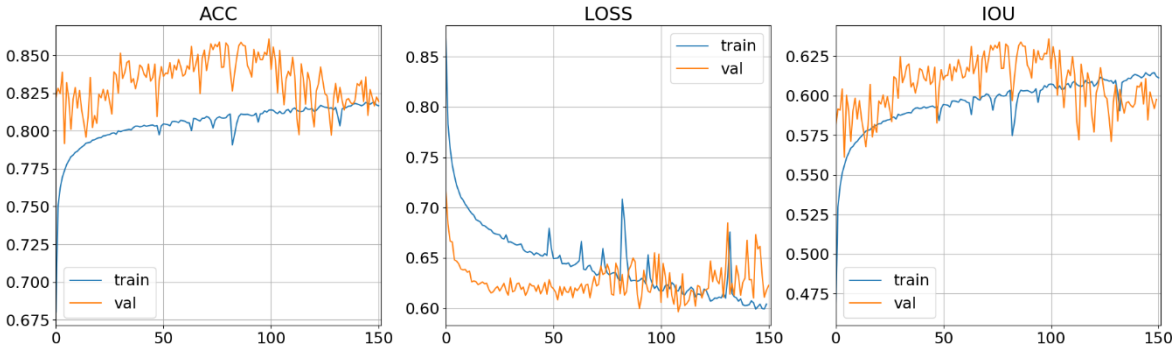


Ilustración 35. Resultados U-Net BASE INRIA v1.

U-Net BN INRIA v1

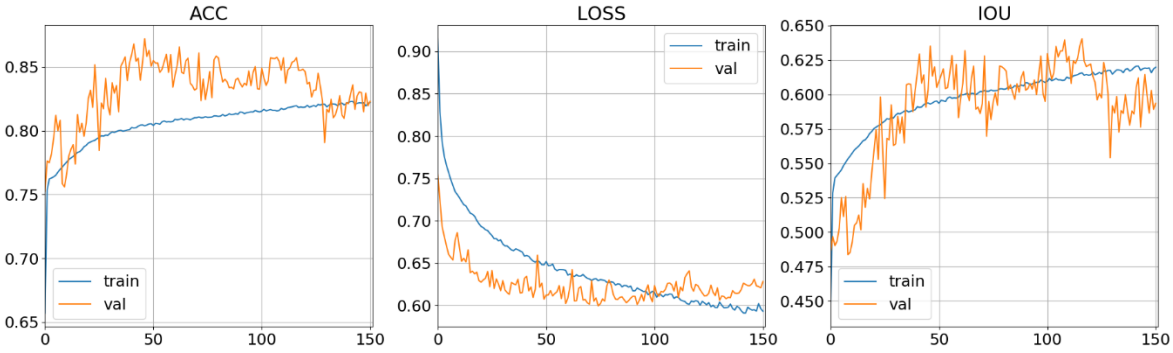


Ilustración 36. Resultados U-Net BN INRIA v1.

U-Net GDN INRIA v1

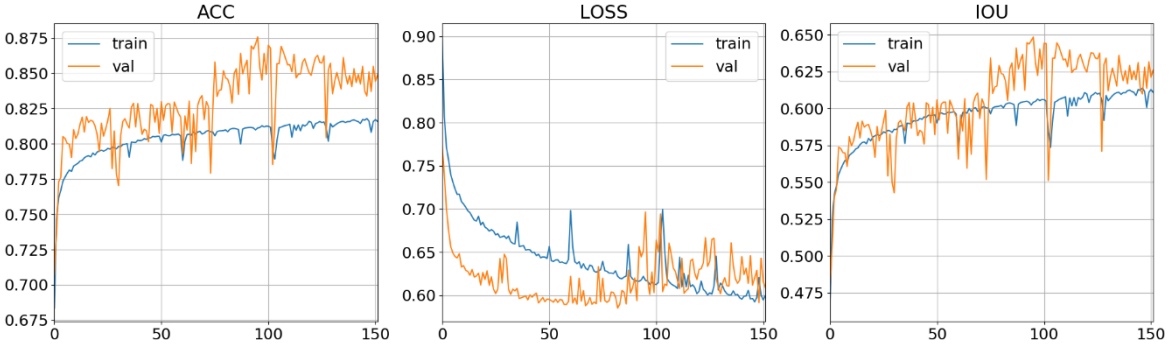


Ilustración 37. Resultados U-Net GDN INRIA v1.

	U-Net BASE v1		U-Net BN v1		U-Net GDN v1	
	Train	**Val**	**Train**	**Val**	**Train**	**Val**
ACC	0.82785	0.86373	0.83472	0.86132	0.82674	0.84462
LOSS	0.63168	0.56277	0.61151	0.54881	0.62682	0.56141
IoU	0.62027	0.64897	0.63042	0.64759	0.61943	0.62650

Tabla 5. Métricas de resultados en modelos v1 (época 150).

	U-Net BASE v1		U-Net BN v1		U-Net GDN v1	
	EPOCH 100		EPOCH 117		EPOCH 96	
	Train	**Val**	**Train**	**Val**	**Train**	**Val**
ACC_opt	0.81376	0.86087	0.81960	0.85726	0.81120	0.87573
LOSS_opt	0.61788	0.65383	0.60145	0.64083	0.61712	0.69657
IoU_opt	0.60715	0.63586	0.61570	0.64044	0.60432	0.64857

Tabla 6. Métricas de resultados de segmentación, en época de mayor "accuracy" de Validación para U-Net v1

Como se puede observar, aunque las métricas son un poco ruidosas conservan una forma correcta para un modelo que entrena y generaliza. Los resultados no parecen ser demasiado satisfactorios, pues teniendo en cuenta el gran desbalanceo entre clases (16% para la clase positiva) el *accuracy* es bastante bajo. Observando el *leaderboard* oficial [21], vemos que se ha llegado a alcanzar un *accuracy* de 97.14% junto con IoU de 80.32% para el conjunto de test (ciudades distintas a las del dataset de entrenamiento), lejos del 87.57% de acc y 64.86% de IoU logrados en el pico más elevado de validación (época 96 para GDN).

Se procede a continuación a estudiar la segunda arquitectura desarrollada.

U-Net BASE INRIA v2

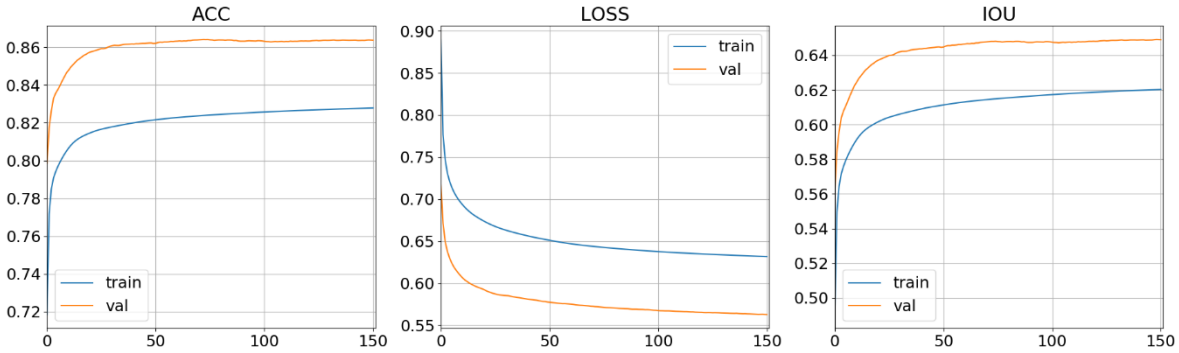


Ilustración 38. Resultados U-Net BASE INRIA v2.

U-NET BN INRIA v2

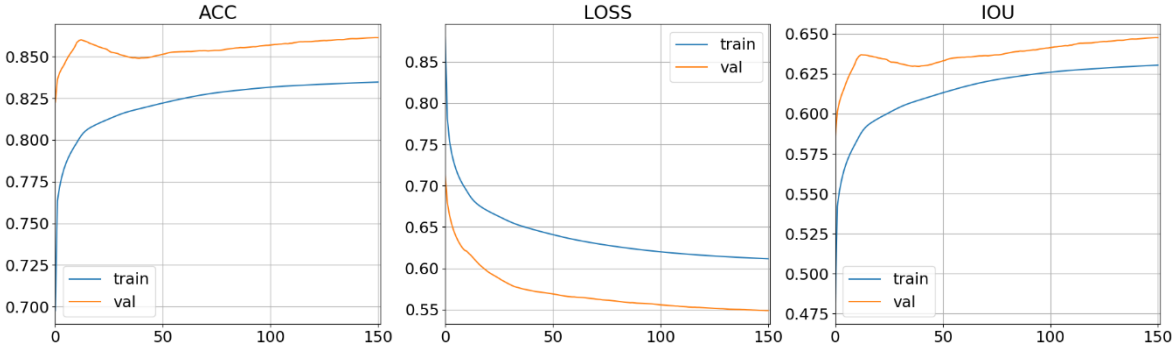


Ilustración 39. Resultados U-Net BN INRIA v2.

U-Net GDN INRIA v2

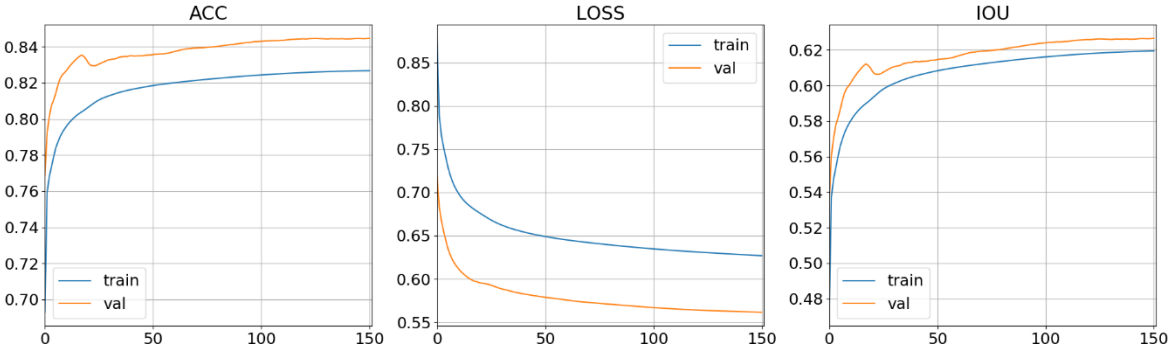


Ilustración 40. Resultados U-Net GDN INRIA v2.

	U-Net BASE v2		U-Net BN v2		U-Net GDN v2	
	Train	**Val**	**Train**	**Val**	**Train**	**Val**
ACC	0.81663	0.81950	0.82226	0.82302	0.81811	0.83652
LOSS	0.60713	0.62258	0.59326	0.62825	0.59125	0.60542
IoU	0.61133	0.59451	0.61978	0.59603	0.61438	0.61327

Tabla 7. Métricas de resultados en modelos v2 (época 150).

En este caso nos encontramos ante métricas mucho menos ruidosas. Los resultados no están suavizados, por lo que parece que la reducción del tamaño en los filtros ha estabilizado el proceso de optimización de los pesos. Aunque el caso "base" no es el que peores resultados proporciona a simple vista, sus métricas tampoco destacan demasiado frente al caso "bn". En cuanto al modelo con normalización GDN, el mejor en este punto, es probable que necesite de más épocas para perfeccionar sus resultados (aunque tampoco presente ventaja excesiva frente al resto hasta el momento). Aun así, se confirma que de nuevo los resultados quedan lejos del *leaderboard* oficial.

Tras 150 épocas, observamos que ninguno de los modelos ha convergido o presenta claros signos de sobreentrenamiento. No es posible postular conclusiones firmes acerca de cuál sería el mejor modelo para este problema. Tampoco se podría determinar, con los datos hasta el momento si, al menos, la arquitectura e hiperparámetros son aceptables. Por todo ello será necesario entrenar un número mayor de épocas, tarea inviable con los recursos y el tiempo disponibles.

El hecho de que las métricas de entrenamiento sean inferiores a las de validación responde a cuestiones de implementación de la capa *dropout*, que al encontrarse activa durante el entrenamiento hace empeorar las métricas. Cuando se inicia la validación la capa de dropout se "desactiva" (asignando 1.0 a la probabilidad de propagación de las neuronas), lo que repercute en un modelo más robusto y unas mejores métricas de rendimiento ante los datos.

Se presentan a continuación las matrices de confusión y algunas predicciones hechas por los modelos "v2", tras 150 épocas de entrenamiento, para la fase de validación:

Matrices de Confusión en Validación para modelos U-Net v2					
BASE		**BN**		**GDN**	
4.69757e+08	6.90423e+07	4.66915e+08	7.18845e+07	4.54956e+08	8.38434e+07
1.61201e+07	7.00801e+07	1.47853e+07	7.14150e+07	1.32663e+07	7.29339e+07

Tabla 8. Matrices de Confusión de validación para modelos U-Net v2 (época 150).

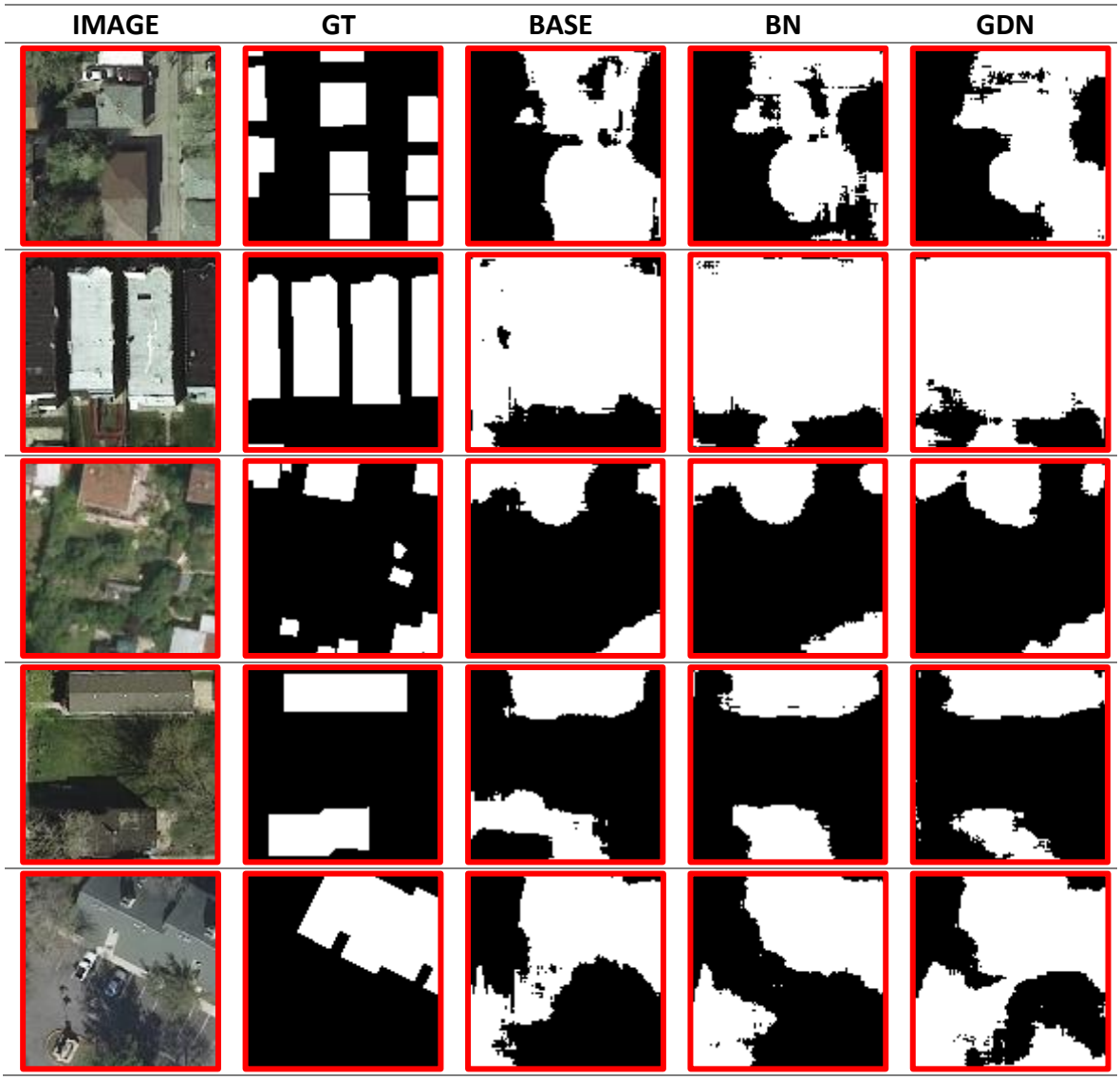


Tabla 9. Predicciones de validación para modelos U-Net v2 (época 150).

Aunque (con un poco de imaginación) se pueden intuir algunas de las formas del *ground truth*, no se observan buenos resultados ni en las predicciones ni en las métricas. Las matrices de confusión son claras respecto el desbalanceo existente en los datos. Las predicciones mostradas no son más que una muestra de la tónica existente en validación.

Por ello, se plantea un experimento para tratar de determinar si el modelo e hiperparámetros actuales son capaces de abordar el problema en cuestión. Tomando como base el caso más sencillo de los modelos ("U-Net BASE INRIA v2"), se tratará de sobreajustarlo con un "pequeño *dataset*" de 25 imágenes. Con esto, se determinará si el número de parámetros del modelo es capaz de aprender un problema de esta complejidad, además de descartar posibles errores en la programación de los modelos y su proceso de entrenamiento en Python.

A continuación, se presentan tres predicciones de ejemplo realizadas por el experimento para este *subset* de entrenamiento a lo largo de las épocas:

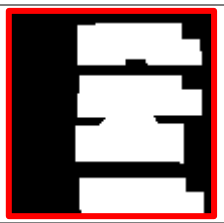
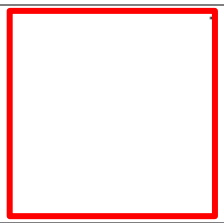
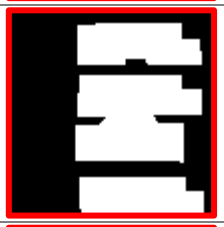
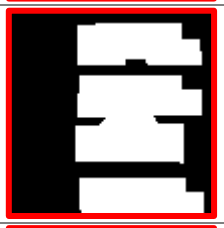
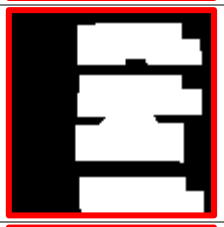
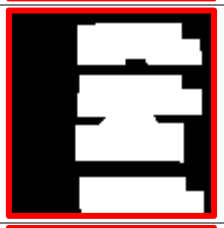
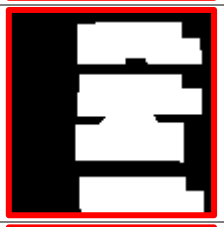
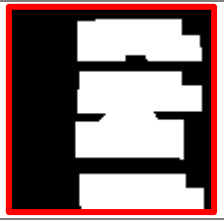
EPOCH	IMG	GT	PREDICTION	ACC_train	IoU_train
80				0.50898	0.26772
160				0.87517	0.73368
240				0.94376	0.86181
320				0.95383	0.88485
400				0.97387	0.92853
480				0.98031	0.94642
560				0.98392	0.95533
600				0.98686	0.96476

Tabla 10. Ejemplo de sobreajuste 1 (U-Net BASE v2)

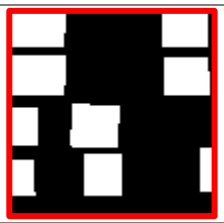
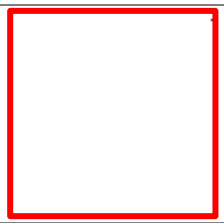
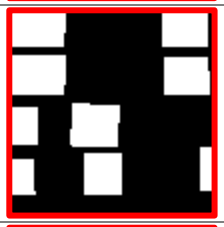
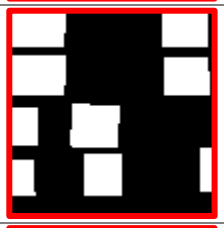
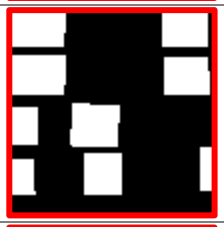
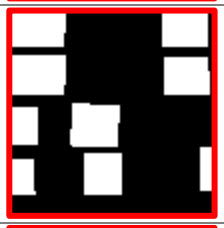
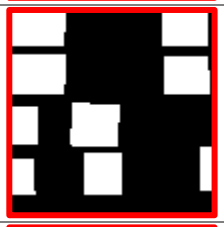
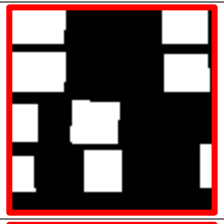
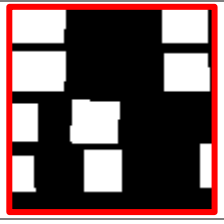
EPOCH	IMG	GT	PREDICTION	ACC_train	IoU_train
80				0.50898	0.26772
160				0.87517	0.73368
240				0.94376	0.86181
320				0.95383	0.88485
400				0.97387	0.92853
480				0.98031	0.94642
560				0.98392	0.95533
600				0.98686	0.96476

Tabla 11. Ejemplo de sobreajuste 2 (U-Net BASE v2).

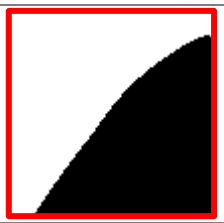
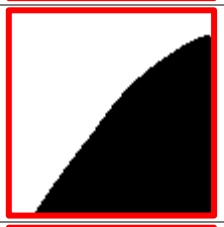
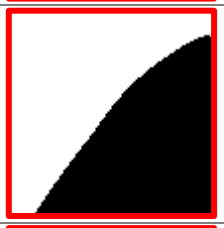
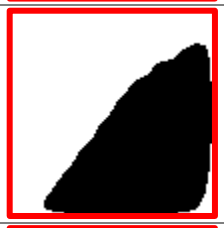
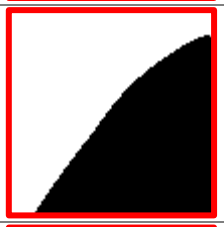
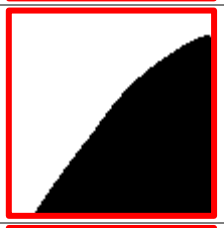
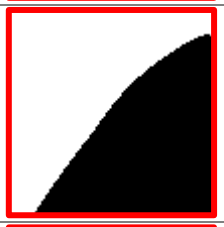
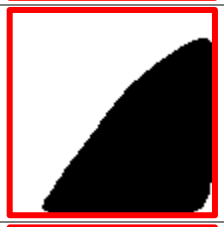
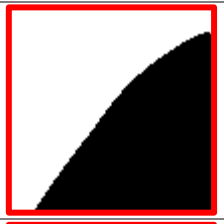
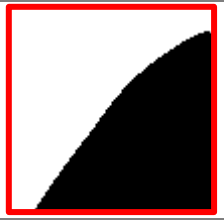
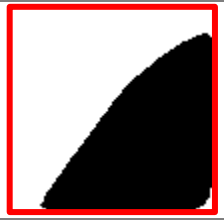
EPOCH	IMG	GT	PREDICTION	ACC_train	IoU_train
80				0.50898	0.26772
160				0.87517	0.73368
240				0.94376	0.86181
320				0.95383	0.88485
400				0.97387	0.92853
480				0.98031	0.94642
560				0.98392	0.95533
600				0.98686	0.96476

Tabla 12. Ejemplo de sobreajuste 3 (U-Net BASE v2).

Tras los ejemplos anteriores, se puede confirmar que el modelo cuenta con suficientes parámetros para alcanzar un sobreentrenamiento en muestras del propio dataset. Ha sido necesario un número de iteraciones superior al realizado con el dataset completo, donde se lograron entrenar 150 épocas. Se ha requerido de 600 épocas hasta que se han logrado predicciones aceptables para las 25 muestras incluidas en el propio subconjunto entrenamiento.

Se puede afirmar que, antes de poder extraer conclusiones firmes sobre si la GDN proporciona mejoras en modelos de segmentación, será necesario continuar con el entrenamiento. Esta tarea es inviable con el tiempo y recursos disponibles actualmente, aunque sería interesante marcarlo como trabajo futuro.

6 Conclusiones

Se han desarrollado modelos de clasificación y segmentación de imágenes naturales utilizando redes neuronales convolucionales con metodologías *"Deep Learning"*. Se han implementado arquitecturas referencia en el estado del arte actual, como lo es la U-Net para segmentación, mediante el framework Tensor Flow con computación en GPU. Se han diseñado técnicas para tratar de mitigar los problemas de recursos que impedían la ejecución de los modelos con los datos originales. Han sido calculadas métricas de rendimiento ajustadas a las necesidades específicas de cada problema. Además, se han aportado ejemplos gráficos de la predicción obtenida en modelos de segmentación.

Presentados los resultados, se ha estudiado y comparado el rendimiento obtenido en los modelos que aplican normalización "GDN" frente a otros que no normalicen o empleen el extendido método del *Batch Normalization*. Tras su análisis, podemos confirmar que existen evidencias que apuntan que, el uso de normalización mediante *Generalized Divisive Normalization* (*"GDN"*), proporcionaría mejores resultados en **modelos de clasificación** de imágenes naturales.

En cuanto a segmentación se refiere, los resultados obtenidos no permiten decantar firmemente las conclusiones hacia ninguno de los métodos en concreto. Aunque el modelo "GDN" parece presentar ligeramente mejores resultados, no supera claramente a ninguna de las alternativas. Como se ha tratado de demostrar mediante el experimento de sobreajuste, serían necesarias muchas más épocas de entrenamiento para poder formular conclusiones válidas.

Es interesante comentar que se ha observado un crecimiento generalizado en el coste de entrenamiento con el uso de esta técnica de normalización. La causa es tanto el aumento en el número de parámetros como la necesidad de ajustar la distribución en los datos antes de proporcionar normalizaciones correctas. Aun así, las mejoras de rendimiento que se extraen de los problemas de clasificación justificarían su aplicación en otros problemas de desarrollo de modelos con imágenes.

Como trabajo futuro, sería interesante realizar entrenamientos más profundos de los modelos de segmentación, con el fin de buscar resultados realmente concluyentes. En cuanto al modelo de clasificación de CIFAR-100, cabría probar si el uso de métodos de regularización y/o *data augmentation* ayuda a mejorar los resultados obtenidos.

7 Referencias

[1] Intel Labs, «Bringing Parallelism to the Web with River Trail,» [En línea]. Available: http://intellabs.github.io/RiverTrail/tutorial/. [Último acceso: 12 06 2019].

[2] V. Dumoulin y F. Visin, «A guide to convolution arithmetic for deep learning,» 2018. arXiv:1603.07285v2. [En línea].

[3] P.-L. Pröve, «An Introduction to different Types of Convolutions in Deep Learning,» 22 7 2017. [En línea]. Available: https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d. [Último acceso: 12 6 2019].

[4] F. Chollet, Deep Learning with Python, Shelter Island, NY: Manning, 2018.

[5] J. Ricco, «What is max pooling in convolutional neural networks? - Quora,» 16 6 2017. [En línea]. Available: https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks. [Último acceso: 13 6 2019].

[6] S. Patel y J. Pingel, «MathWorks. Introduction to Deep Learning: What Are Convolutional Neural Networks?,» [En línea]. Available: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html.

[7] K. Kai y W. Xiaogang, *Fully Convolutional Neural Networks for Crowd Segmentation,* The Chinese University of Hong Kong, 2014.

[8] C. Manish, «Autoencoders - Introduction and Implementation in TF,» 26 6 2017. [En línea]. Available: https://towardsdatascience.com/autoencoders-introduction-and-implementation-3f40483b0a85. [Último acceso: 16 6 2019].

[9] H. Noh, S. Hong y B. Han, «Learning Deconvolution Network for Semantic Segmentation. CVLab, Pohang University of Science and Technology.,» [En línea]. Available: http://cvlab.postech.ac.kr/research/deconvnet/. [Último acceso: 16 6 2019].

[10] C.-H. Yee, «Cardiac MRI Segmentation,» [En línea]. Available: https://chuckyee.github.io/cardiac-segmentation/. [Último acceso: 16 6 2019].

[11] F. Peccia, «Batch normalization: theory and how to use it with Tensorflow,» 16 9 2018. [En línea]. Available: https://towardsdatascience.com/batch-normalization-theory-and-how-to-use-it-with-tensorflow-1892ca0173ad. [Último acceso: 16 6 2019].

[12] J. Ballé, V. Laparra y E. P. Simoncelli, *Density Modeling of Images Using a Generalized Normalization Transformation,* New York: ICLR, 2016.

[13] J. Ballé, «tf.contrib.layers.gdn,» [En línea]. Available: https://www.tensorflow.org/versions/r1.13/api_docs/python/tf/contrib/layers/gdn. [Último acceso: 26 6 2019].

[14] V. Laparra, J. Ballé, A. Berardino y E. P. Simoncelli, «Perceptual image quality assessment using a normalized Laplacian pyramid,» de *Human Vision and Electronic Imaging (HVEI)*, San Francisco, California, USA, 2016.

[15] V. Laparra, A. Berardino, J. Ballé y E. P. Simoncelli, *Perceptually Optimized Image Rendering,* Universitat de València: Image Processing Lab; New York University: Center for Neural Science, Courant Institute of Mathematical Sciences, 2017.

[16] A. Krizhevsky, «Learning Multiple Layers of Features from Tiny Images,» April 8, 2009.

[17] E. Maggiori, Y. Tarabalka, G. Charpiat y P. Alliez, «Can Semantic Labeling Methods Generalize to Any City? The Inria Aerial Image Labeling Benchmark,» de *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Fort Worth, Texas, USA, 2017.

[18] TensorFlow Community, «Weighted Cross Entropy with Logits,» [En línea]. Available: https://www.tensorflow.org/versions/r1.13/api_docs/python/tf/nn/weighted_cross_entropy_with_logits. [Último acceso: 20 6 2019].

[19] Sergei, «StackOverflow Community: How to Correctly Calculate tf.nn.weighted_cross_entropy_with_logits pos_weight variable,» 20 2 2018. [En línea]. Available: https://stackoverflow.com/questions/43564490. [Último acceso: 20 6 2019].

[20] A. Rosebrock, «Intersection Over Union (IoU) for object detection,» 7 11 2016. [En línea]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. [Último acceso: 22 6 2019].

[21] INRIA, «Leaderboard - Aerial Image Labeling Dataset,» [En línea]. Available: https://project.inria.fr/aerialimagelabeling/leaderboard/. [Último acceso: 24 6 2019].

[22] Open Source Geospatial Foundation (OSGeo), «Orthorectification - OSSIM,» 2 7 2014. [En línea]. Available: https://trac.osgeo.org/ossim/wiki/orthorectification. [Último acceso: 3 6 2019].