



Informe de Servicio Social:
**Aplicación de una Red Neuronal
a Videos para Identificar el Plano
de Enfoque de Movimientos de
Partículas**

José Samuel Rodríguez Olguín

Realizado en el Taller de Hidrodinámica y Turbulencia, Facultad de Ciencias
Universidad Nacional Autónoma de México
En el programa de Apoyo a la investigación

con clave: 2019-12/12-1813

Fecha de inicio: 05 de noviembre de 2019

Fecha de termino: 15 de julio de 2021

Resumen

Mi trabajo de servicio social consistió en la investigación y síntesis de información sobre redes neuronales como recurso para posteriores trabajos realizados en el Taller de Hidrodinámica y Turbulencia. Con esto realicé el desarrollo de una red neuronal que recibiera como datos de entrada videos con partículas usadas como trazadores para medir el flujo de un fluido y como resultado generara una máscara para cada fotograma que incluyera sólo las partículas enfocadas y descartara el fondo y las partículas desenfocadas. Para esto, primero generé manualmente dos conjuntos de cien máscaras a partir de cien imágenes con partículas enfocadas y desenfocadas. Luego, desarrollé un algoritmo de preprocesamiento para seccionar las imágenes a tamaños menores para poder ser procesadas por la red neuronal. Posteriormente, comencé a modificar la arquitectura de la U-net, con la librería Keras, hasta obtener predicciones acertadas. Finalmente, realicé un algoritmo para poder visualizar los resultados de procesar las imágenes con la red neuronal. La ventaja de aplicar redes neuronales radica en que en lugar de proponer algoritmos y parámetros como otros métodos, el algoritmo de retropropagación de las redes neuronales selecciona los parámetros óptimos a partir del conjunto de entrenamiento.

1. Introducción

1.1. Enfoque y Nitidez

Bajo la aproximación paraxial, para cierta distancia focal dada; idealmente las imágenes más nítidas formadas en el sensor de la cámara (que se encuentra en un plano) provendrán del *plano de enfoque*. Idealmente, todo punto localizado en este plano será proyectado como un punto en el sensor de la cámara.

Los puntos que no se encuentren en el plano de enfoque, en el sensor en lugar de un punto se proyectará una imagen inscrita en un círculo de radio c , llamado **círculo de confusión**, por lo que la imagen pierde nitidez.

Siempre que el radio del círculo de confusión no exceda un valor preescrito, las imágenes se mantendrán nítidas o *enfocadas*. Por lo que entonces existen puntos en planos delante y detrás del plano de enfoque que forman círculos en el sensor cuyo radio no excede dicho valor; estos planos forman la **profundidad de campo**. Las distancias hacia el frente y detrás de el plano de enfoque no son simétricas. Las ecuaciones de las distancias de la profundidad de campo se pueden encontrar en Rowlands (2017).

Podemos encontrar más información sobre el enfoque en Hetch (2017), Giusfredi (2019), Bass et al. (2009), Rowlands (2017), Teubner & Brückner (2019), Schwartz (2019).

1.2. Planteamiento del Problema

Cuando se quiere realizar Velocimetría por Seguimiento de Sombras de Partículas (PSTV por su siglas en inglés: Particle Shadow Tracking Velocimetry) grabadas por una sola cámara, se presenta un gran problema: El video obtenido proyecta la posición en tres dimensiones de estas partículas, a imágenes en dos dimensiones. Por lo que la información del video se vuelve inexacta por diversas razones, entre las cuales están:

1. La escala y nitidez de las imágenes formadas depende de la distancia de las partículas a la cámara
2. Si el fenómeno no es invariante respecto a esta distancia, entonces tenemos apilada en la imagen información de varios planos diferentes.

Se ha logrado obtener la información 3D de la posición de partículas mediante arreglos que involucran el uso de dos cámaras. En microfluídica se han desarrollado métodos para obtener las posiciones 3D de partículas esféricas o esferoides a partir de su proyección con distinto enfoque en el sensor digital. Por ejemplo: Barnkob et al. (2015) usaron un conjunto experimental de calibración, posteriormente Rossi (2020) generó un conjunto computacional de calibración; y en ambos casos se compara cada partícula del experimento a estudiar con el conjunto de calibración para determinar la distancia a los objetivos de cada partícula, considerándolas como fuentes emisoras de luz. Sin embargo el éxito de estos métodos radica en que los instrumentos usados en microfluídica generan una relación aproximada de 1 : 10 entre la profundidad de campo respecto a la profundidad del volumen de interés, por lo que las partículas generan un gran espectro de formas diferentes cuando se alejan del plano de enfoque. En el Taller de Hidrodinámica y Turbulencia, el problema que se plantea a resolver, la velocimetría por seguimiento de sombras, la profundidad del volumen de interés es del orden de la profundidad de campo, donde las sombras fuera de la profundidad de campo no distorsionan mucho sus proyecciones en el sensor, ha sido abordado por Echeverría et al. (2020) con algoritmos de filtrado de imágenes para obtener sólo las partículas enfocadas. Esto permite acotar el volumen estudiado para fenómenos donde las velocidades en 3D son relevantes. El presente proyecto realiza la velocimetría por seguimiento de sombras usando redes neuronales; creando máscaras que contengan sólo partículas enfocadas.

1.3. Redes Neuronales

Las redes neuronales artificiales son una regresión no lineal para aproximar la relación de dependencia entre variables de entrada y salida. A partir de esta relación podremos predecir resultados con nuevos datos de entrada. Las redes neuronales están constituidas por los siguientes elementos:

Neuronas Llamamos neurona a la estructura que efectúa una función de activación a la suma pesada de un conjunto de datos de entrada:

Los datos de entrada pueden provenir de otras neuronas.

Capas Una capa es el conjunto de neuronas que recibe el mismo conjunto de datos de entrada:

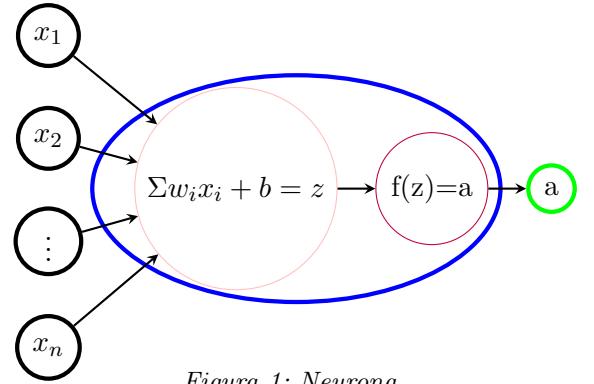


Figura 1: Neurona.

Este conjunto de datos puede provenir de otra capa; y no es necesario que los datos sean completamente conexos a las neuronas (como, por ejemplo en la segunda neurona de la primera capa oculta de la figura 2).

Capa de Entrada Se le llama capa de entrada al conjunto de datos iniciales que recibe la red neuronal. Por esta razón, la capa de entrada no es contada con el resto de las capas. Es decir, cuando nos referimos a una red de 1 capa; nos referimos a que la red neuronal consta de una sola capa (y sus datos de salida) además de la capa de entrada.

Capa de Salida La capa de salida es aquella que procesa y otorga el conjunto de datos de salida.

Capas Ocultas Todas las capas entre la capa de entrada y la capa de salida son capas ocultas. (Vasilev et al. (2019))

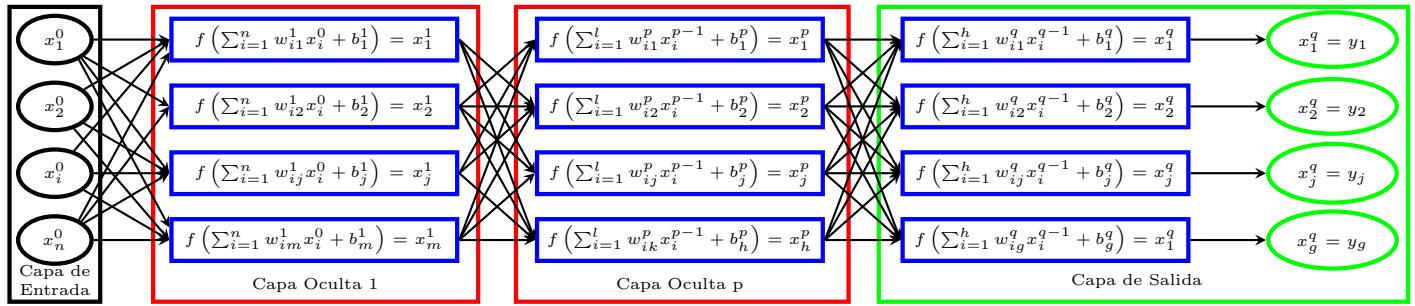


Figura 2: Red neuronal de q -capas.

1.3.1. Funciones de Activación

Las funciones de activación son una parte medular de las redes neuronales, pues sin ellas no importaría cuántas capas tenga la red, sería un modelo lineal y todas las capas colapsarían en una sola capa equivalente; pues sólo estaríamos multiplicando matrices. El objetivo de las funciones de activación es volver no lineales las capas para así volver compleja la red y crecer el espacio de funciones que podemos aproximar. Las funciones de activación más empleadas en las redes neuronales son:

ReLU La función lineal rectificada [*Rectified Linear Units*]

$$f_{ReLU}(z) = \begin{cases} z & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$$

es la función más empleada en las redes neuronales, en las capas ocultas, pues es una función no lineal computacionalmente eficiente. Su derivada es: (Berzal (2018))

$$\frac{d}{dx} f_{ReLU}(z) = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$$

Sigmoide La función sigmoide:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

se puede considerar como una función escalón suavizada, por lo tanto continua y diferenciable. Se emplea con frecuencia en la capa de salida para clasificaciones binarias; pues en su rango $[0, 1]$ la función envía a los valores muy negativos asintóticamente al 0 y a los valores muy positivos asintóticamente al 1. (Shanmugamani (2020), Berzal (2018)) Su derivada es:

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$$

1.3.2. El Perceptrón

El perceptrón es la arquitectura mostrada en la figura 1; es la arquitectura más simple de red neuronal y se puede emplear para problemas de clasificación de dos conjuntos separados por un plano. Conectándose a otros perceptrones o neuronas, se pueden resolver problemas más complejos.

1.3.3. Retropropagación

Cuando los datos de entrada son procesados, es decir, que pasan a través de las neuronas, se dice que las entradas se han *propagado hacia adelante* para darnos los valores de salida.

Ahora supongamos que conocemos el valor de salida. El objetivo de la red neuronal es reducir la diferencia entre el valor predicho y el valor de salida. La retropropagación es el algoritmo en el que los pesos w se actualizan hacia atrás a partir de el error entre el valor predicho y el valor real. (Sing & Manure (2020), Shanmugamani (2020))

1.3.4. Descenso del Gradiente

Pensando el error $J = J(\theta_1, \theta_2, \dots, \theta_i, \dots, \theta_n)$ como una hipersuperficie dependiente de n variables, al buscar minimizar el error (la función de Costo) estamos buscando entonces el mínimo de J . Para esto, vamos a actualizar nuestros parámetros θ_i en sentido contrario a la pendiente $\nabla_{\theta_1, \theta_2, \dots, \theta_i, \dots, \theta_n} J$, que es la dirección hacia la cual J va reduciendo su valor. Por lo que, entonces actualizaremos nuestros parámetros de la siguiente forma:

$$\theta_i = \theta_i - \eta \nabla_{\theta_i} J(\theta_1, \theta_2, \dots, \theta_i, \dots, \theta_n) = \theta_i - \eta \frac{\partial J(\theta_1, \theta_2, \dots, \theta_i, \dots, \theta_n)}{\partial \theta_i} \quad (1)$$

donde definimos a η como la *taza de aprendizaje*. η es un parámetro crucial, pues nos indica qué tanto vamos a avanzar en la dirección que indica el gradiente. Un valor bajo de η nos llevaría muy lentamente al mínimo, lo cual sería ineficiente. Sin embargo, un η muy grande podría no converger a un punto y mantenernos alejados del mínimo (Rebala et al. (2019), Carlos Santana Vega (2018a), Berzal (2018)).

Aunque no existe una prueba de que el mínimo sea un mínimo global, conforme aumenta el número de dimensiones de la hipersuperficie (los parámetros que actualizamos para el aprendizaje de la red neuronal) la probabilidad de encontrarnos con un mínimo local disminuye exponencialmente; puede ser que encontremos mínimos locales respecto a unas cuantas de las variables, pero es improbable que todas nos lleven a un mínimo local, por lo que el gradiente difícilmente será cero. Sin embargo, el problema con el que podemos llegar a encontrarnos es con *mesetas*, donde la función parece aplanarse, y el gradiente tiene un valor pequeño (Ng 2015 (2015)). Es importante hacer una nota en cómo se van obteniendo las derivadas para los parámetros; pues cada capa de una red neuronal es una función sobre el resultado anterior; o sea que cada capa es anidar la primer; por ejemplo, una red de tres capas se podría

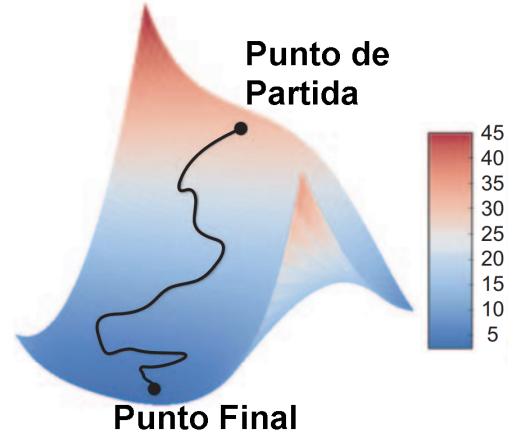


Figura 3: Descenso del Gradiente. Adaptado de Chollet (2018)

ver como: $f(g(h(\theta, x), \psi), \phi)$, por lo que, querer actualizar el parámetro θ , implica calcular las derivadas de las funciones que anidan $h(\theta, x)$ por regla de la cadena:

$$\frac{\partial J(\theta, \phi, \psi)}{\partial \theta} = \frac{\partial J}{\partial f} \frac{\partial f}{\partial g} \frac{\partial g}{\partial h} \frac{\partial h}{\partial \theta} \quad (2)$$

De ahí la denominación *retropropagación*, pues vamos calculando las derivadas de las últimas capas para conocer las derivadas de las primeras capas. Podemos pensar entonces en la retropropagación como otra forma de nombrar la regla de la cadena.

1.3.5. Funciones de Costo

Es importante entonces preguntarnos cómo ponderaremos el error de nuestro modelo respecto a valores observados; pues el valor especulado puede ser mayor o menor que el observado, por lo que, por ejemplo, al sumar todas las diferencias perdemos información sobre qué tanto difiere el modelo de los valores observados en conjunto. Para obtener información del error usamos las *funciones de costo* (Krohn et al. (2020)). Mencionaremos dos ejemplos, el primero por heurística y el segundo es al que recurriremos para esta red neuronal.

Error Cuadrático Medio Una forma de ponderar la distancia absoluta de los valores predichos con los observados es:

$$J(y) = \frac{1}{n} \sum_{j=0}^n (y_j - \hat{y}_j)^2 \quad (3)$$

Pues elevar al cuadrado nos da valores del mismo signo(Purkait (2019)). Su derivada es:

$$\frac{\partial J}{\partial y_j} = \frac{2}{n} (y_j - \hat{y}_j) \quad (4)$$

Entropía Cruzada Para los problemas de clasificación y probabilidades la última capa suele ser la función sigmoide, cuyo rango es $[0, 1]$. La derivada de la función sigmoide puede ser problemática, pues es siempre < 1 ; lo cual puede afectar el aprendizaje por retropropagación. Por lo tanto, en lugar de emplearse el error cuadrático medio como función de coste, se usa la entropía cruzada:

$$J(y) = -\frac{1}{n} \sum_{j=0}^n [\hat{y}_j \log(y_j) + (1 - \hat{y}_j) \log(1 - y_j)] \quad (5)$$

Cuya derivada:

$$\frac{\partial J}{\partial y_j} = \frac{y - \hat{y}}{y(1 - y)} \quad (6)$$

Podemos notar que el denominador entonces, elimina el factor por la derivada de la función sigmoide en la retropropagación. (Berzal (2018), Vasilev et al. (2019))

		Predicción	
		Positivo	Negativo
Clase Real	Positivo	Verdaderos Positivos	Falsos Negativos
	Negativo	Falsos Positivos	Verdaderos Negativos

1.3.6. Métricas de evaluación

Para los problemas de clasificación binaria nuestro modelo sólo puede darnos cuatro tipos de resultados: Verdaderos Positivos (VP), Verdaderos Negativos (VN), Falsos Positivos (FP) y Falsos Negativos (FN) representados en la matriz de contingencia de la figura 4. La metrica más utilizada es la precisión, definida como: (Berzal (2018))

$$\text{precisión} = \frac{VP + VN}{n} \quad (7)$$

con n el número total de datos evaluados, por tanto el error es su complemento:

$$\text{error} = 1 - \text{precisión} \quad (8)$$

Las métricas de evaluación, a diferencia de las funciones de coste; no tienen impacto alguno en el algoritmo de aprendizaje de las redes neuronales.(Purkait (2019))

1.3.7. Sobreajuste: Métodos de Regularización

El sobreajuste surge cuando al entrenar la red neuronal con los datos muestra, la función resultante predice muy bien los resultados observados en este conjunto, pero al evaluar la red neuronal, o predecir a partir de datos nuevos, la red tiene un mal desempeño y grandes errores. Entre los métodos empleados para la regularización se encuentran:

- **Ajustar la capacidad del modelo:** Usualmente una red neuronal más somera, con menos parámetros de aprendizaje, tendrá menor capacidad para ajustarse a un conjunto de datos específico. Sin embargo, esto puede afectar también al desempeño con datos nuevos.
- **Regularización de la función de coste:** Esto implica cambiar la función de coste, agregándole términos por ejemplo.
- **Aplicar un enfoque bayesiano:** El *Dropout* es la manera más usada. Se puede especificar una probabilidad a algunas neuronas o parámetros para no actualizarse cada entrenamiento.
- **Generar más datos:** *Data augmentation* en inglés. Es, copiar y modificar los datos de entrenamientos; mediante, por ejemplo, rotaciones o distorsiones. Es importante que los datos modificados sigan cumpliendo las características de la categorías que se pretende estudiar.
- **Obtener más datos:** Esta es la mejor forma para regularizar una red neuronal. Sin embargo es la más difícil, pues no siempre se puede disponer de más datos. Para la aplicación de una red neuronal para identificar partículas en el plano de enfoque, sí es posible y de hecho sólo se usa este método. Este método de regularización es el éxito del *BigData* (Berzal (2018)).

1.3.8. Descenso del Gradiente Estocástico

Idealmente los parámetros se actualizan bajo la suposición de que conocemos el gradiente exacto de la función que deseamos optimizar. Sin embargo, sólo disponemos de una muestra de datos a partir de la cual estimamos el gradiente. Pero, mientras este conjunto sea representativo de la distribución real de datos, podemos asumir que obtendremos una buena estimación del gradiente. Aún así, mientras el conjunto sea más grande, mejor será la estimación. Tener un error en la estimación del gradiente puede ser benéfico, pues este error funge como un método de regularización. Cuando los conjuntos de entrenamiento son muy grandes, como es el caso de esta aplicación de redes neuronales, el conjunto de entrenamiento se parte en lotes. (Berzal (2018))

1.4. Visión de Computadora

La visión de computadora es la ciencia de entender o manipular imágenes y videos digitales. El uso de redes neuronales en la visión de computadora puede clasificarse en: clasificación, detección, segmentación y generación; de imágenes y/o videos (Shanmugamani

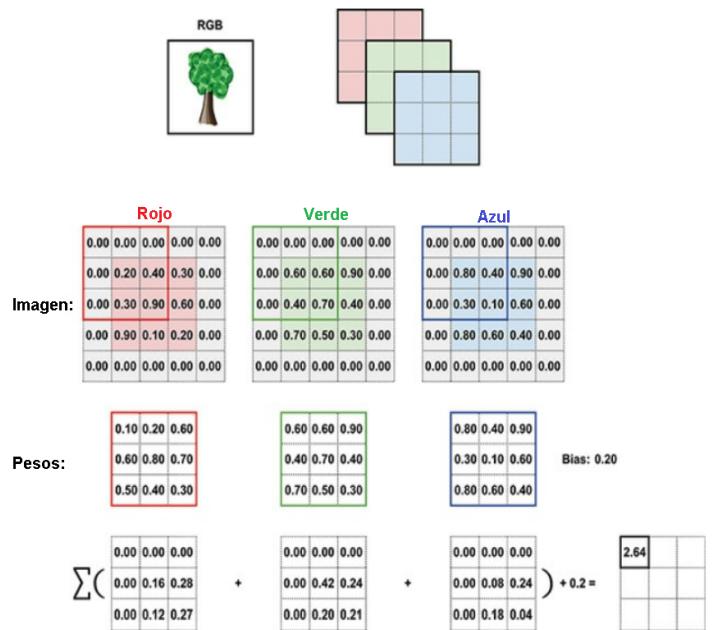


Figura 5: Esquema de una neurona convolucional. Adaptado de Krohn et al. (2020)

(2020)). El problema que se desea resolver está en la categoría de:

Segmentación Semántica La segmentación es el proceso de dividir una imagen en estructuras con significado (Scherer (2020)). La segmentación semántica es una tarea de clasificación pixel a pixel (Shanmugamani (2020)).

Segmentación de Instancias La segmentación por instancias es una forma más específica de segmentación semántica; pues ahora dividimos las estructuras con significado en entidades particulares que pueden pertenecer a la misma estructura.

Un ejemplo para aclarar la diferencia entre estos dos tipos de segmentación es el siguiente: Detectar los pixeles que corresponden a personas en una imagen es una tarea de segmentación semántica. Clasificar los pixeles pertenecientes a cada persona, es una tarea de segmentación de instancias.

1.5. Redes Neuronales Convolucionales

Referidas también por el acrónimo CNN [*Convolutional Neural Network*], se basan en el uso de convoluciones. Se utilizan habitualmente para resolver problemas que requieren procesar imágenes, aunque su uso no se limita a eso. Las CNN reducen el número de parámetros que usaría una red neuronal completamente conexa, explotando la correlación dimensional de los datos mediante la convolución. La convolución sustituye a la multiplicación matricial de las redes neuronales completamente conexas. (Berzal (2018), Gad (2019), p 183-198))

Convolución En las redes neuronales, la convolución se define como la correlación cruzada. Definimos la convolución para una señal x de dimensiones $N_1 \times N_2 \times N_3$ en la posición n_1, n_2, n_3 y un filtro h de tamaño $K_1 \times K_2 \times K_3$ como:

$$(x * h)[n_1, n_2, n_3] = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} \sum_{k_3=1}^{K_3} h[k_1, k_2, k_3] x[n_1 + k_1, n_2 + k_2, n_3] \quad (9)$$

En el procesamiento de imágenes, la primer dimensión corresponde al eje vertical, la segunda al eje horizontal, y la tercera a los canales (por ejemplo, RGB) (Berzal (2018)). La convolución reduce el tamaño de la imagen a:

$$\begin{aligned} N_1 &= N_1 - K_1 + 1 \\ N_2 &= N_2 - K_2 + 1 \\ N_3 &= N_3 - K_3 + 1 \end{aligned} \quad (10)$$

Convencionalmente, los filtros de las neuronas convolucionales deben coincidir en número de canales con la imagen que procesará, para que la salida tenga sólo un canal, pues las salidas de las neuronas de una misma capa convolucional serán los canales de la salida de la capa.

1.5.1. U-Net

La U-net es una arquitectura de red neuronal desarrollada para la segmentación semántica de imágenes biomédicas. Como se muestra en la figura 6 esta red neuronal va codificando la imagen mediante capas convolucionales, aumentando el número de canales y reduciendo su resolución tomando el valor máximo de cada cuatro pixeles cuadrados. Posteriormente, la imagen se decodifica concatenando con codificaciones anteriores y aplicando capas convolucionales. El esquema que representa esta codificación y decodificación es lo que le da el nombre de U-net [Red-U](Ronneberger et al. (2015))(Ayyadevara (2019)).

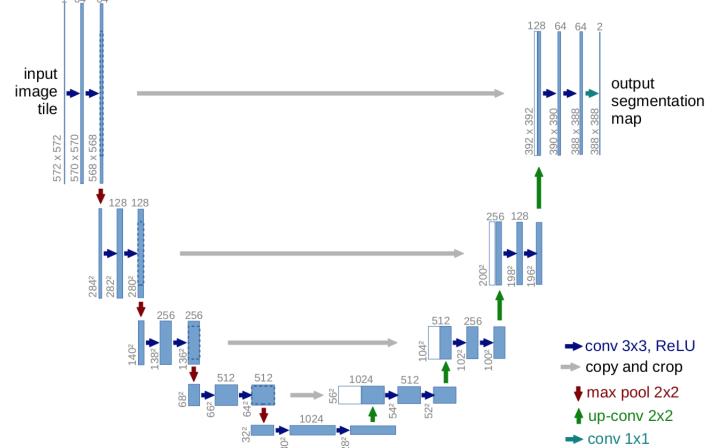


Figura 6: Esquema de la arquitectura U-net, de Ronneberger et al. (2015)

1.6. Plataformas Auxiliares y Bibliotecas de Redes Neuronales

1.6.1. CUDA

Computer Unified Device Architecture es una librería desarrollada por NVIDIA que popularizó el uso de cómputo basado en procesadores gráficos. (Joshi (2020)) La arquitectura de las GPUs, especializada en operaciones matriciales y vectoriales y procesos en paralelo; nos permite ejecutar de forma más eficiente que las CPUs las redes neuronales. Aunque la librería fue desarrollada por NVIDIA, es compatible con un amplio rango de tarjetas gráficas. Usualmente, para la mayor parte de aplicaciones de redes neuronales, no es necesario conocer la sintaxis del código, basta con instalar la plataforma desde el sitio <https://developer.nvidia.com/cuda-zone>, pues la mayor parte de interfaces de programación de aplicaciones llama automáticamente las funciones de CUDA.

1.6.2. Tensorflow

Como su nombre lo indica, es una librería que toma como eje los *tensores*. Como se ha explicado, las redes neuronales trabajan con arreglos de rango n. En el contexto de redes neuronales, los *tensores* son una generalización de los vectores y matrices a dimensiones superiores.

El otro componente fundamental de Tensorflow es el *flow*, el flujo; pues Tensorflow provee de un marco de trabajo para crear grafos computacionales; que empatan con la estructura de las redes neuronales, por lo que permite el flujo de información en forma de tensores hacia adelante, para obtener los resultados, y hacia atrás para la retropropagación de errores(Sing & Manure (2020))

La librería de Tensorflow está escrita en Python, C++ y CUDA; por las extenciones operacionales de éste último lenguaje, Tensorflow puede ejecutarse paralelamente en el procesador gráfico.

La documentación oficial de Tensorflow se puede encontrar en su sitio web: www.tensorflow.org.

1.6.3. Keras

Keras es una librería para redes neuronales escrita en Python. Provee una interfaz sencilla para usar Tensorflow como motor. Keras está diseñado para una experimentación fácil y sencilla enfocándose en la modularidad, extensibilidad y amigabilidad para el usuario. (Shanmugamani (2020)) la documentación oficial de Keras se encuentra en: <https://keras.io/>. En el apéndice se explican los métodos usados para la aplicación de redes neuronales para la identificación de planos de enfoque.

2. Aplicación y Arquitectura de la Red Neuronal

El objetivo principal de la red neuronal es seleccionar las partículas que se encuentren en el plano de enfoque. Primero generé tres conjuntos de cien imágenes. A partir de estos conjuntos, rústicamente mediante el software de edición de imagen *Adobe Photoshop* © eliminé el fondo, seleccionando puntos estratégicos y todos los pixeles adyacentes con una intencidada ± 15 . Para el primer conjunto eliminé todas las partículas indistinguibles. Para el segundo y tercer conjunto, eliminé todas las partículas que no estuvieran estrictamente enfocadas. Posteriormente seleccioné las partículas, y eliminé cualquier otro elemento restante. Transformé la imagen a color monotono: blanco, y finalmente rellené el fondo con negro. De esta forma generé las máscaras de entrenamiento para la red neuronal.

Las redes neuronales son computacionalmente exhaustivas, por lo que las imágenes de resolución original $1280px \times 800px$ fueron recortadas en *subsecciones* de $256px \times 200px$. Generar subsecciones a partir de imágenes permite analizar imágenes de tamaños relativamente arbitrarios; siempre que la dimensiones de las subsecciones sean múltiplos de 8 para la arquitectura *Shallow U-net* y múltiplos de 4 para la arquitectura *W-net* y que sus dimensiones sean $\gg 2px$. Con estos conjuntos se entrenaron las redes neuronales. A las redes neuronales desarrolladas no se les incluyó ningún algoritmo de regularización; pues se pueden obtener más datos de forma sencilla.

2.1. Shallow U-net

Adapté U-net desarrollada por Ronneberger et al. (2015) a una versión con menos capas; de ahí el nombre *Shallow* (Somera). Todas las convoluciones de la U-net se realizan por pasos de un pixel, y salvo la última capa, todas las capas convolucionales tienen función ReLU de activación. La Shallow U-net procesa la imagen de la siguiente forma:

1. Se aplican 64 filtros convolucionales de $3 \times 3 \times 1$ a la imagen; dándonos como resultado una imagen de 64 canales.
2. Se aplican 64 filtros convolucionales de $3 \times 3 \times 64$ a la imagen de 64 canales, devolviéndonos una imagen de 64 canales.
3. Cada canal se contrae seleccionando el pixel de mayor intensidad en ventanas de 2×2 . Su anchura y altura por tanto se reduce a la mitad.
4. Se aplican 128 filtros convolucionales de $3 \times 3 \times 64$, para obtener una imagen de 128 canales.
5. Se aplican 128 filtros convolucionales de $3 \times 3 \times 128$, para obtener una imagen de 128 canales.
6. Cada canal se contrae seleccionando el pixel de mayor intensidad en ventanas de 2×2 . Su anchura y altura por tanto se reduce a un cuarto de las originales.
7. Se aplican 256 filtros convolucionales de $3 \times 3 \times 128$, para obtener una imagen de 256 canales.
8. Se aplican 256 filtros convolucionales de $3 \times 3 \times 256$, para obtener una imagen de 256 canales.
9. Cada canal se contrae seleccionando el pixel de mayor intensidad en ventanas de 2×2 . Su anchura y altura por tanto se reduce a un octavo de las originales
10. Se aplican 512 filtros convolucionales de $3 \times 3 \times 256$, para obtener una imagen de 512 canales.
11. Se aplican 512 filtros convolucionales de $3 \times 3 \times 512$, para obtener una imagen de 512 canales.
12. Se agranda la imagen, copiando cada pixel en ventanas de 2×2 . La imagen vuelve a un cuarto de su altura y anchura originales.
13. Se aplican 256 filtros convolucionales de $3 \times 3 \times 512$, para obtener una imagen de 256 canales.
14. Se concatena la imagen con la obtenida en el paso , resultando en una imagen de 512 canales.
15. Se aplican 256 filtros convolucionales de $3 \times 3 \times 512$, para obtener una imagen de 256 canales.
16. Se aplican 256 filtros convolucionales de $3 \times 3 \times 256$, para obtener una imagen de 256 canales.
17. Se agranda la imagen, copiando cada pixel en ventanas de 2×2 . La imagen vuelve a la mitad de su altura y anchura originales.
18. Se aplican 128 filtros convolucionales de $3 \times 3 \times 256$, para obtener una imagen de 128 canales.
19. Se concatena la imagen con la obtenida en el paso , resultando en una imagen de 256 canales.
20. Se aplican 128 filtros convolucionales de $3 \times 3 \times 256$, para obtener una imagen de 128 canales.
21. Se aplican 128 filtros convolucionales de $3 \times 3 \times 128$, para obtener una imagen de 128 canales.
22. Se agranda la imagen, copiando cada pixel en ventanas de 2×2 . La imagen vuelve a su altura y anchura originales.
23. Se aplican 64 filtros convolucionales de $3 \times 3 \times 128$, para obtener una imagen de 64 canales.
24. Se concatena la imagen con la obtenida en el paso , resultando en una imagen de 128 canales.
25. Se aplican 64 filtros convolucionales de $3 \times 3 \times 64$, para obtener una imagen de 64 canales.
26. Se aplican 64 filtros convolucionales de $3 \times 3 \times 64$, para obtener una imagen de 64 canales.
27. Se aplican 2 filtros convolucionales de $3 \times 3 \times 64$, para obtener una imagen de 2 canales.
28. Se aplica 1 filtro convolucional de $1 \times 1 \times 2$ con función de activación sigmoide. El resultado es la máscara predicha.

Esta arquitectura no tiene algoritmos de regularización.

2.2. W-net

Inspirado en la U-net, desarrollé la *W-net*. Su nombre, al igual que la U-net, viene por el diagrama que representa la codificación y decodificación de la imagen. Todas las convoluciones de la W-net se realizan por pasos de un pixel, y salvo la última capa, todas las capas convolucionales tienen función ReLU de activación. La W-net procesa la imagen de la siguiente forma:

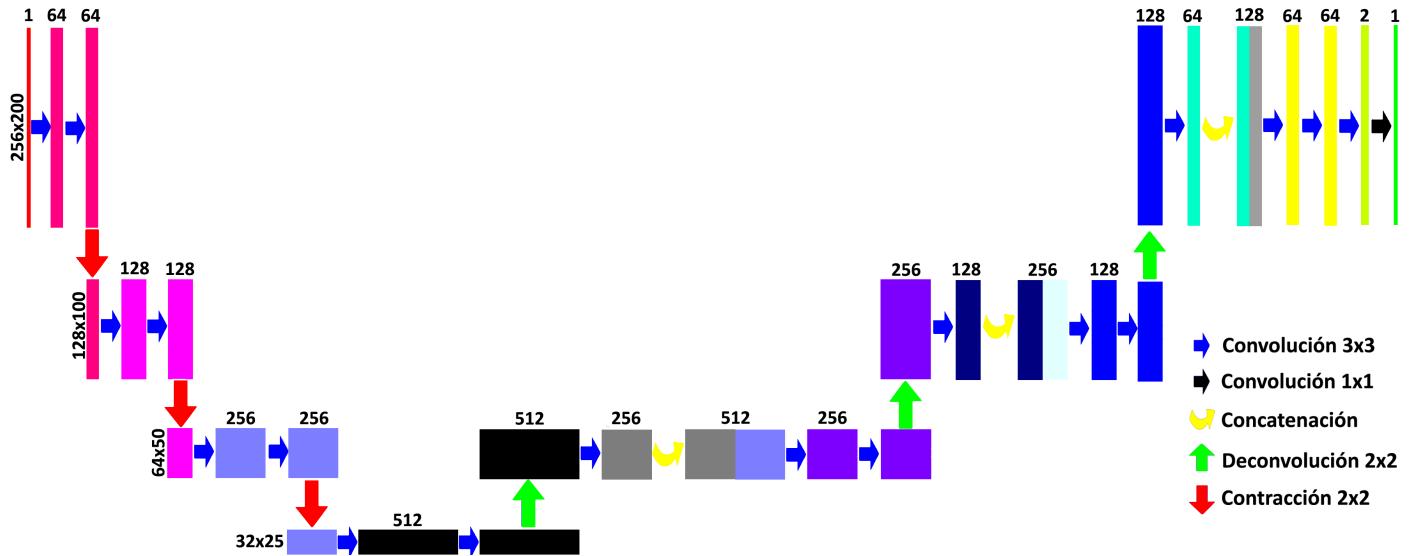


Figura 7: Shallow U-net: Adaptación de la U-net

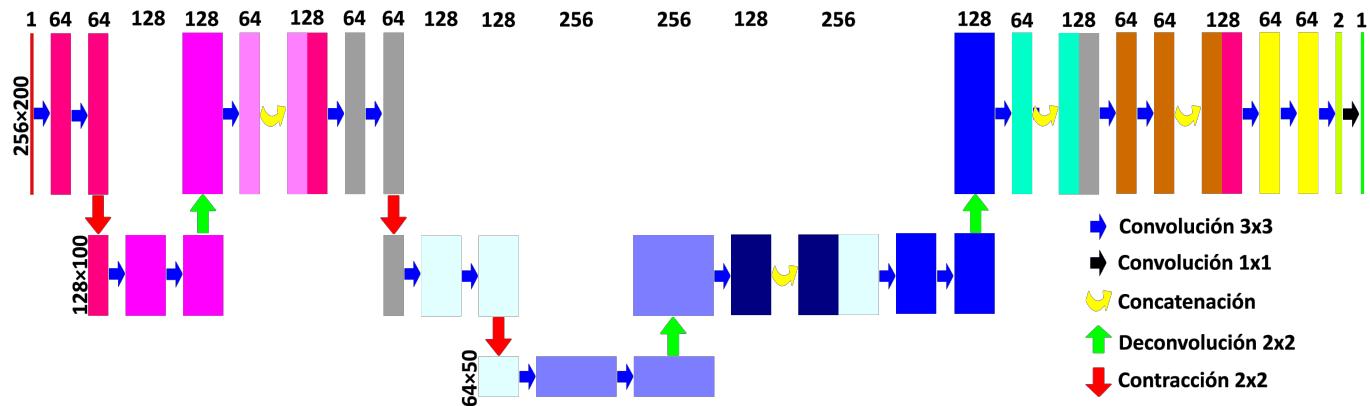


Figura 8: W-net: Arquitectura de la red neuronal implementada

1. Se aplican 64 filtros convolucionales de 3x3x1 a la imagen; dándonos como resultado una imagen de 64 canales.
2. Se aplican 64 filtros convolucionales de 3x3x64 a la imagen de 64 canales, devolviéndonos una imagen de 64 canales.
3. Cada canal se contrae seleccionando el pixel de mayor intensidad en ventanas de 2x2. Su anchura y altura por tanto se reduce a la mitad.
4. Se aplican 128 filtros convolucionales de 3x3x64, para obtener una imagen de 128 canales.
5. Se aplican 128 filtros convolucionales de 3x3x128, para obtener una imagen de 128 canales.
6. Se agranda la imagen, copiando cada pixel en ventanas de 2x2. La imagen vuelve a su altura y anchuras originales.
7. Se aplican 64 filtros convolucionales de 3x3x128, para obtener una imagen de 64 canales.
8. Se concatena la imagen con la obtenida en el paso 2, resultando en una imagen de 128 canales.
9. Se aplican 64 filtros convolucionales de 3x3x128 para obtener una imagen de 64 canales.
10. Se aplican 64 filtros convolucionales de 3x3x64, para obtener una imagen de 64 canales.
11. Se contrae la imagen seleccionando el pixel de mayor intensidad en ventanas de 2x2. Se reducen a la mitad la altura y la anchura.
12. Se aplican 128 filtros convolucionales de 3x3x64, para obtener una imagen de 128 canales.
13. Se aplican 128 filtros convolucionales de 3x3x128, para obtener una imagen de 128 canales.

14. Se contrae la imagen seleccionando el pixel de mayor intensidad en ventanas de 2x2. La altura y anchura pasan a ser un cuarto de las originales.
15. Se aplican 256 filtros convolucionales de 3x3x128, para obtener una imagen de 256 canales.
16. Se aplican 256 filtros convolucionales de 3x3x256, para obtener una imagen de 256 canales.
17. Se agranda la imagen, copiando cada pixel en ventanas de 2x2. La imagen vuelve a la mitad de la altura y anchura originales.
18. Se aplican 128 filtros convolucionales de 3x3x256, para obtener una imagen de 128 canales.
19. Se concatena la imagen con la obtenida en el paso 13, resultando una imagen de 256 canales.
20. Se aplican 128 filtros convolucionales de 3x3x256, para obtener una imagen de 128 canales.
21. Se aplican 128 filtros convolucionales de 3x3x128, para obtener una imagen de 128 canales.
22. Se agranda la imagen, copiando cada pixel en ventanas de 2x2. La imagen vuelve a su altura y anchura original.
23. Se aplican 64 filtros convolucionales de 3x3x128, para obtener una imagen de 64 canales.
24. Se concatena la imagen con la obtenida en el paso 10, resultando una imagen de 128 canales.
25. Se aplican 64 filtros convolucionales de 3x3x128, para obtener una imagen de 64 canales.
26. Se aplican 64 filtros convolucionales de 3x3x64, para obtener una imagen de 64 canales.
27. Se concatena la imagen con la obtenida en el paso 2, resultando una imagen de 128 canales.
28. Se aplican 64 filtros convolucionales de 3x3x128, para obtener una imagen de 64 canales.
29. Se aplican 64 filtros convolucionales de 3x3x64, para obtener una imagen de 64 canales.
30. Se aplican 2 filtros convolucionales de 3x3x64, para obtener una imagen de 2 canales.
31. Se aplica 1 filtro convolucional de 1x1x2 con función de activación sigmoide. El resultado es la máscara predicha.

Esta arquitectura no tiene algoritmos de regularización.

2.3. Diferencias

Las diferencias entre las arquitecturas anteriores con la U-net son mínimas; por lo que podemos enlistarlas.

	U-net	Shallow U-net	W-net
Número de capas convolucionales	23	19	21
Número de parámetros entrenables		8,557,445	2,584,261
Reducción Máxima	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$
Número máximo de canales	1024	512	256
Regularización	50 % Dropout	Ninguna	Ninguna
Arquitectura	Codificación - Decodificación	Codificación - Decodificación	Codificación - Decodificación - Codificación - Decodificación

3. Resultados

El código que desarrollé puede encontrarse en el siguiente enlace: <https://github.com/rodriguezsamuel/Plano-de-enfoque>.

El código fue ejecutado en una computadora con un CPU Intel Core i7-7700HQ a 3.6GHz, 16GB de RAM y un GPU NVIDIA GeForce GTX 1050 con 4GB de RAM dedicada. Las redes neuronales fueron entrenadas con un lote de 90 imágenes 5 veces. Para la Shallow U-net el tiempo promedio por imagen para el entrenamiento fue de 2.416s. El tiempo promedio para la predicción fue de 0.793s por imagen.

Para la W-net el tiempo promedio por imagen para el entrenamiento fue de 3.06s, lo cual es un 26.6 % más de tiempo que la Shallow U-net. El tiempo promedio para la predicción fue de 0.976s por imagen, un 23.1 % más de tiempo que la Shallow U-net. La figura 9 es un ejemplo de los resultados.

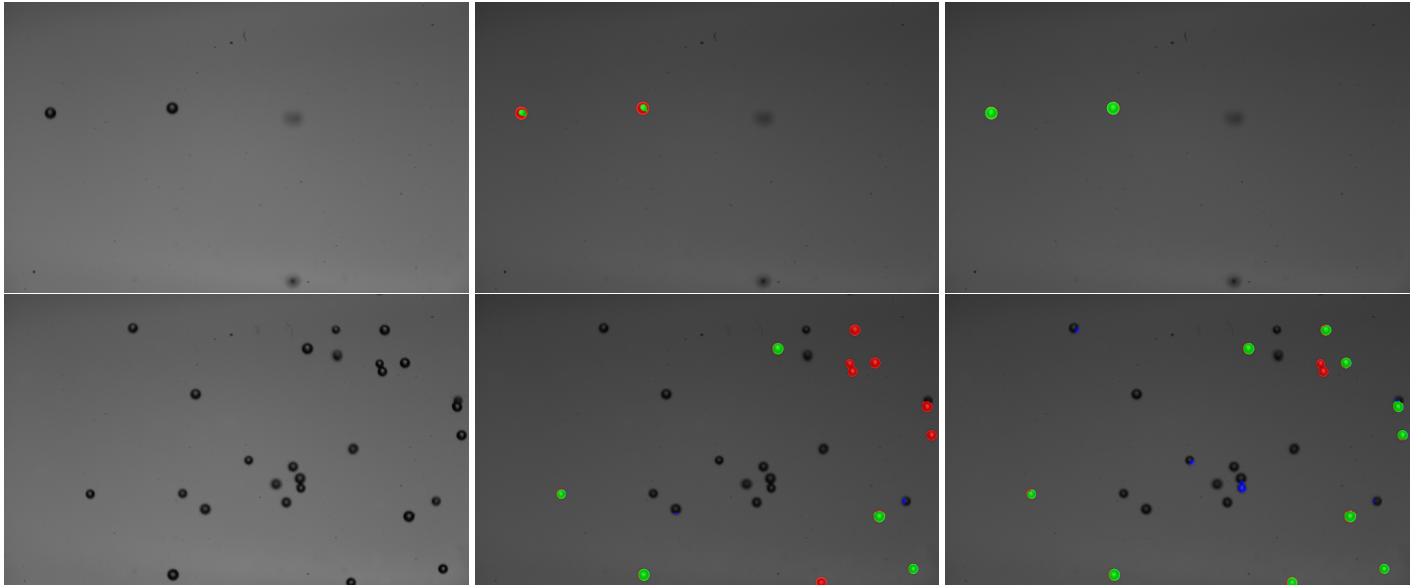


Figura 9: De izquierda a derecha: La imagen, la predicción de partículas enfocadas por la Shallow U-net y la predicción de partículas enfocadas por la W-net (verde: verdaderos positivos, azul: falsos positivos, rojo: falsos negativos).

		Predicción Shallow U-net		Predicción W-net	
		Positivo	Negativo	Positivo	Negativo
Clase Real	Positivo	Verdaderos Positivos: 16	Falsos Negativos: 18	Verdaderos Positivos: 31	Falsos Negativos: 3
	Negativo	Falsos Positivos: 6	Verdaderos Negativos: 34	Falsos Positivos: 7	Verdaderos Negativos: 33

Figura 10: Matrices de contingencia de los resultados de la arquitecturas aplicadas

Para evaluar estos modelos podemos hacer las matrices de contingencia tomando todos los casos para las partículas.

Así obtenemos que la **precisión de la arquitectura Shallow U-net: 67.5 %** y la **precisión de la arquitectura W-net: 86.5 %**; por lo que podemos concluir que la W-net tuvo un desempeño mucho mayor a la Shallow U-net. Hay que aclarar que la evaluación estuvo sujeta a mi subjetividad para considerar las partículas enfocadas o no. Otro resultado que podemos notar es que las máscaras predichas usualmente tienen un área menor que las máscaras para evaluar.

4. Conclusiones

La aplicación de esta arquitectura de red neuronal nos permite generar, con una buena precisión imágenes binarias que contengan sólo partículas enfocadas. A partir de estas imágenes binarias, se pueden aplicar métodos como PIV *Particle Image Velocimetry* o PTV *Particle Image Velocimetry*.

La precisión de estas arquitecturas puede mejorar definiendo nuevos conjuntos de entrenamiento con criterios más objetivos o al menos homogéneos para decidir si las partículas están enfocadas o no. Las partículas en las

que la red generó una máscara parcial pueden indicar que quizá entrenando las redes neuronales más épocas los resultados podrían mejorar.

La mayor ventaja de las redes neuronales es que a diferencia de algoritmos que podríamos catalogar como *rudimentarios* es la cantidad de parámetros que debemos definir. Por ejemplo, en este caso específico de detección de partículas, sólo debemos definir criterios como el umbral de binarización o la convexidad del área de las regiones, mientras que otros algoritmos deben incluir parámetros como el tamaño de las ventanas de interrogación o buscar los promedios de las intensidades de los pixeles para definir el fondo.

Como trabajo a futuro se propone mejorar estas arquitecturas o realizar otra nuevas para implementar mediante redes neuronales un algoritmo de PTV. Para este caso se aplicó la arquitectura U-net por sencillez, pero existen otras arquitecturas de redes neuronales usadas para la segmentación semántica. La más destacada además de la U-net, es *Mask R-CNN* (2017) de He et al. que más compleja que la U-net, pero también más específica, pues está desarrollada para ejecutar la segmentación de instancias.

Otra posible vertiente propuesta como trabajo a futuro sería mejorar la red neuronal para que en lugar de descartar las partículas fuera del plano de enfoque, pueda dar información acerca de la distancia en el eje z a la que se encuentran. Las imágenes nunca forman puntos, ni siquiera en el plano de enfoque, debido a las aberraciones (i.e. esférica, coma, astigmatismo, curvatura del plano de enfoque, aberración cromática) (Yacoubian (2015)) y a la difracción misma de la luz (Schwartz (2019)). Pero, estas aberraciones pueden entonces tomarse como información extra, que si bien es muy complicado proponer modelos analíticos, la retropropagación de las redes neuronales podría ajustarse para desencriptar la información de la distancia a partir de estas aberraciones.

Referencias

- Ayyadevara, V. K. (2019). *Neural Networks with Keras Cookbook*, page sec 7.3. Packt Publishing.
- Barnkob, R., Kähler, C. J., and Rossi, M. (2015). General defocusing particle tracking. *Lab on a Chip*, 15(17):3556–3560.
- Bass, M., DeCusatis, C., Enoch, J. M., Lakshminarayanan, V., Li, G., MacDonald, C., Mahajan, V., and Van Stryland, E. (2009). *Handbook of Optics, Volume 1.*, pages sec 1.74–75. McGraw-Hill Professional Publishing. OCLC: 958557660.
- Berzal, F. (2018). *Redes Neuronales & Deep Learning*, pages 60–61,197,201,260–291,427–470,487,558–559,599–601,606–607. Edición Independiente, Granada, España.
- Chollet, F. (2018). *Deep Learning with Python*, page 50. Manning Publications, USA.
- Echeverría, C., Porta, D., Stern, C., and Guzmán, J. E. V. (2020). A method to determine the measurement volume for particle shadow tracking velocimetry (PSTV). *Journal of Visualization*, 23(4):577–590.
- Gad, A. F. (2019). *Practical Computer Vision Applications Using Deep Learning with CNNs*, pages 183–198. Apress, Menoufa, Egypt.
- Giusfredi, G. (2019). *Physical Optics: Concepts, Optical Elements, and Techniques*, pages 203–204. UNITEXT for Physics. Springer International Publishing.
- He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015 Part III*, 2017 IEEE International Conference on Computer Vision:2980–2988.
- Hecht, E. (2017). *Optics*, page 187. Pearson Education, Inc, 5 ed edition.
- Joshi, A. V. (2020). *Machine Learning and Artificial Intelligence*, page 118. Springer, Redmond, WA, USA.
- Krohn, J with Beyleveld, G. and Bassens, A. (2020). *Deep Learning Illustrated A Visual, Interactive Guide to Artificial Intelligence*, pages sec III.8.1, III.10.1. Addison-Wesley.

- Ng, A. (2015). Deep learning specialization. master deep learning, and break into ai. <https://www.coursera.org/specializations/deep-learning>.
- Purkait, N. (2019). *Hands-On Neural Networks with Keras*, pages 1.2.3.1,2.4.18. Packt Publishing.
- Rebala, G., Ravi, A., and Churiwala, S. (2019). *An Introduction to Machine Learning*, pages 31–32. Springer.
- Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015 Part III*, Lecture Notes in Computer Science, vol 9351:234–241.
- Rossi, M. (2020). Synthetic image generator for defocusing and astigmatic PIV/PTV. *Measurement Science and Technology*, 31(1):017003.
- Rowlands, A. (2017). *Physics of Digital Photography*, pages sec 1.2.1,1.3.1–1.3.2. IOP Publishing.
- Santana Vega, C. (2018a). ¿qué es el descenso del gradiente? https://www.youtube.com/watch?v=A6FiCDoz8_4.
- Santana Vega, C. (2018b). ¿qué es una red neuronal? <https://www.youtube.com/channel/UCy5znSnfMsDwaLlR0nZ7Qbg>.
- Scherer, R. (2020). *Computer Vision Methods for Fast Image Classification and Retrieval*, volume 821 of *Studies in Computational Intelligence*, page 22. Springer, Czestochowa, Poland.
- Schwartz, S. H. (2019). *Geometrical and visual optics a clinical introduction*, pages 210–212. McGraw Hill Education, 3rd edition. OCLC: 1100899784.
- Shanmugamani, R. (2020). *Deep Learning for Computer Vision*, pages sec 1.1.2.1,1.1.4.1,1.2,1.3.4,5. Packt Publishing.
- Singh, P. and Manure, A. (2020). *Learn Tensorflow 2.0: Implement Machine Learning and Deep Learning Models with Python*, pages 2–3,58–61. Apress, Bangalore, Karnataka, India.
- Teubner, U. and Brückner, H. J. (2019). *Optical Imaging and Photography: Introduction to Science and Technology of Optics, Sensors and Systems*, pages 138–149. De Gruyter.
- Vasilev, I., Slater, D., Spacagna, G., Roeleants, P., and Zocca, V. (2019). *Python Deep Learning*, pages 37–42,87. Packt Publishing, Birmingham, UK, 2nd edition.
- Yacoubian, A. (2015). *Optics Essentials: An Interdisciplinary Guide*, pages 77–82. CRC Press, Carlsbad, California, USA.

Atentamente

Vo. Bo.