Introducción a las redes neuronales:Auto codificadores y auto decodificadores

M. Sc. Saúl Calderón Ramírez
Instituto Tecnológico de Costa Rica,
Escuela de Computación, bachillerato en Ingeniería en Computación,
PAttern Recongition and MAchine Learning Group (PARMA-Group)

10 de mayo de 2019

Resumen

Este material está basado en el capítulo de redes neuronales, del libro *Pattern recognition and Machine Learning* de Cristopher Bishop[1].

1 Autocodificadores

Un auto codificador es una red neuronal $\overrightarrow{r} = m(\overrightarrow{x})$ entrenada para copiar su entrada a la salida, de modo que el modelo tiene en este caso el objetivo de hacer $\overrightarrow{r} \rightarrow \overrightarrow{x}$. Al utilizar por ejemplo un perceptrón multicapa, la salida de la capa oculta $\overrightarrow{h} \in \mathbb{R}^M$ podría pensarse como una *reducción* de la dimensionalidad de la muestra a su entrada $\overrightarrow{x} \in \mathbb{R}^D$, de forma que, en general $D > M$, correspondiendo a una compresión de la entrada, o una forma de aprender las características de la entrada. En general, un auto codificador consiste en dos partes:

1. Una función de codificación $\overrightarrow{h} = f(\overrightarrow{x})$.
2. Una fución de reconstrucción o decodificación $\overrightarrow{r} = g\left(\overrightarrow{h}\right)$.

De esta forma entonces el modelo está dado por:

$$\overrightarrow{r} = m(\overrightarrow{x}) = g(f(\overrightarrow{x}))$$

La Figura 1 ilustra el flujo general de un autocodificador.

Para aplicaciones específicas, el aprender sin ningún error la entrada ($\overrightarrow{r} = \overrightarrow{x}$), no es útil, frecuentemente el autocodificador es entrenado para aprender a aproximar la entrada, como es el caso de los autocodificadores para la eliminación del ruido. El modelo entonces implementa restricciones para permitir la copia aproximada de la entrada.

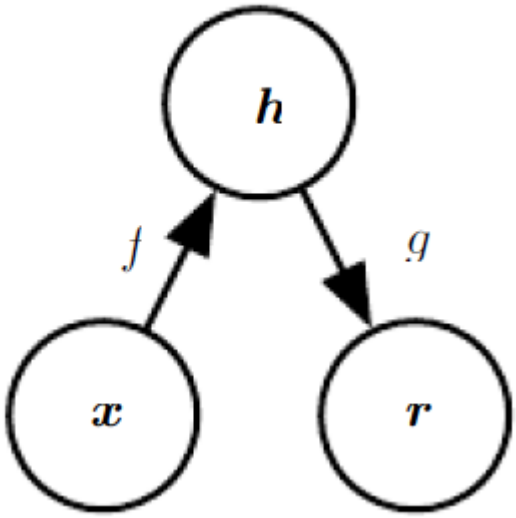

Figura 1: Flujo básico de un autocodificador, tomado de [2].

2 Autocodificadores subcompletos, completos y sobrecompletos

El objetivo de reconstruir la muestra o señal de entrada, tiene sentido cuando utilizamos la **parte de codificación para obtener una compresión o reducción de la dimensionalidad de los datos** $\overrightarrow{h}$. Los autocodificadores diseñados con el objetivo de reducir la dimensionalidad de la entrada $\overrightarrow{x} \in \mathbb{R}^D$ en la capa oculta $\overrightarrow{h} \in \mathbb{R}^M$, con lo que $D > M$, se conocen como **autocodificadores subcompletos** (*undercomplete autoencoders*). El aprendizaje de una representación subcompleta fuerza al modelo a aprender las características mas relevantes a partir de los datos de entrenamiento.

El proceso de aprendizaje de un autocodificador subcompleto consiste simplemente en minimizar la función de pérdida dada por:

$$L\left(\overrightarrow{x}, \overrightarrow{r}\right) = L\left(\overrightarrow{x}, m\left(\overrightarrow{x}\right)\right) = L\left(\overrightarrow{x}, g\left(f\left(\overrightarrow{x}\right)\right)\right)$$

que puede estar definida como por ejemplo como la distancia euclidiana:

$$L\left(\overrightarrow{x}, \overrightarrow{r}\right) = \left\|\overrightarrow{r} - \overrightarrow{x}\right\|^2$$

Una de las técnicas clásicas en la reducción de la dimensionalidad es el análisis de componentes principales (PCA por sus siglas en inglés). **Se ha demostrado**

que un autocodificador con funciones de activación no lineales puede aprender generalizaciones no lineales mucho más poderosas que un PCA.

Otras variantes de autocodificadores son:

1. **Autocodificadores completos**: Las dimensiones de la salida a la capa oculta son equivalentes a la capa de entrada $D = M$
2. **Autocodificadores sobrecompletos**: Las dimensiones de la salida a la capa oculta son mayores a la capa de entrada $D < M$

En tales variantes, la cantidad de parámetros del modelo dificultan la generalización, memorizando las muestras y evitando el aprendizaje de características útiles.

Como se estudió en el problema de regresión, controlar la complejidad del modelo manualmente, de forma proporcional con la cantidad de datos y la dimensionalidad de los mismos es una tarea manual propensa a malas decisiones. La **regularización** de los autocodificadores es necesaria para resolver tal problemática, y además promover la ralidad o *sparsity* del modelo y la robustez al ruido. La regularización de un autocodificador se implementa usualmente a través de la función de pérdida. Un ejemplo de ello son los autocodificadores ralos o *sparse autoencoders*, los cuales agregan un término de penalidad o regularizador para incentivar la *ralidad* del modelo $\Omega\left(\overrightarrow{h}\right)$, modificando la función de pérdida de la siguiente forma*:*

$$L\left(\overrightarrow{x}, g\left(f\left(\overrightarrow{x}\right)\right)\right) + \Omega\left(\overrightarrow{h}\right)$$

2.1 *Upsampling* y *downsampling*

- **Etapa de codificación:** La etapa de codificación implementa múltiples capas, compuestas en primera instancia por un banco de filtros, con una función de activación (ReLU por ejemplo) y una posterior capa de *max-pooling*. La superposición de estas capas reduce la dimensionalidad de los datos.
- **La etapa de decodificación**: La etapa de decodificación aumenta la dimensionalidad de la salida a en cada capa, a través de un muestreo hacia arriba o *upsampling*. El *upsampling* se refiere a estimar una señal de dimensionalidad $\hat{\vec{x}} \in \mathbb{R}^n$ a partir de la señal $\overrightarrow{u} \in \mathbb{R}^m$, con $m < n$. En imágenes, el *upsampling* se refiere a aumentar el tamaño de la imagen, como lo muestra la Figura 2. Existen muchos métodos para estimar la imagen con tamaño aumentado, por ejemplo la interpolación por vecino más cercano, la interpolación bi-linear o la interpolación bi-cúbica. Todos estos métodos están basados en el concepto de interpolación, e implementan distintas variantes con distintos núcleos, con lo cual decidir el método

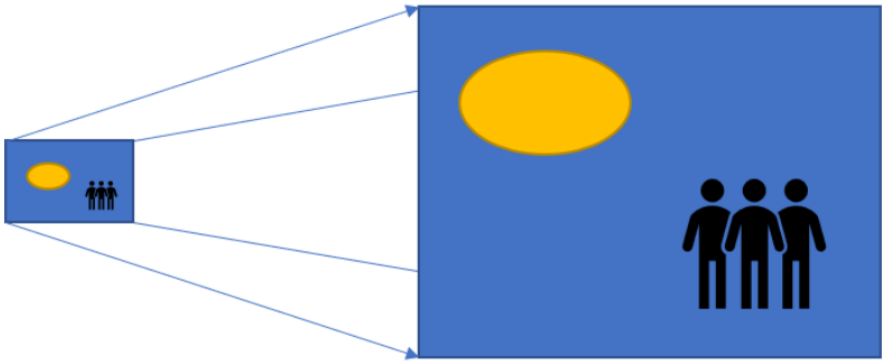

Figura 2: Upsampling *o aumentado de tamaño de una imagen, tomado de* `https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0`.

de interpolación se vuelve un problema de ingeniería manual de características (*manual feature engineering*). Sin embargo, sería idóneo plantear el problema de forma que la **red pueda aprender el tipo de interpolación a realizar**. Para eso se implementa el concepto de convolución transpuesta.

1. **Convolución transpuesta**: La convolución transpuesta permite aumentar el tamaño de la dimensionalidad de una imagen, a través de un conjunto de pesos aprendibles. La convolución transpuesta se le llama **deconvolución** o **convolución fraccional estriad**a (*fractional strided convolution*). Recordando el concepto de convolución, tomando una imagen, por ejemplo $U \in \mathbb{R}^{4\times4}$, y un filtro $F \in \mathbb{R}^{3\times3}$, la convolución sin utilizar *padding*, *y* un stride de *1*, resulta en:

$$U * F = Y \in \mathbb{R}^{2\times2},$$

pues la dimensión de la imagen de salida está dada por la fórmula $\frac{m-k+2c}{p}+1$, donde en este caso $m = 4$, $k = 3$, y el stride o paso es de $4 - 3 + 1 = 2$. De esta forma, la operación de convolución disminuye las dimensiones de la imagen resultante. Tómese entonces

$$U = \begin{bmatrix} 4 & 5 & 8 & 7 \\ 1 & 8 & 8 & 8 \\ 3 & 6 & 6 & 4 \\ 6 & 5 & 7 & 8 \end{bmatrix} \quad F = \begin{bmatrix} 1 & 4 & 1 \\ 1 & 4 & 3 \\ 3 & 3 & 1 \end{bmatrix} \Rightarrow U * F = Y = \begin{bmatrix} 122 & 148 \\ 126 & 134 \end{bmatrix}$$

donde por ejemplo $Y_{0,0} = U_{0:3,0:3} \cdot F$, en términos del producto punto. El computo anterior para todas las entradas $Y_{i,j}$ se aprecia en la Figura 3. La Figura 3, muestra además, cómo al realizar el cálculo de $Y_{0,0}$, se rea-

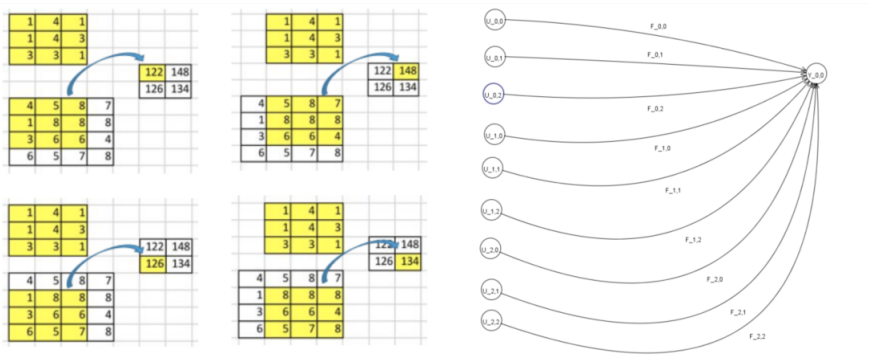

Figura 3: Producto punto para cada una de las entradas $Y_{i,j}$, tomado de `https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0`.

liza una conectividad o **relación de muchos a uno,** lo que implica una reducción de la dimensionalidad de la señal. Para el *upsampling* es necesario entonces pasar una relación de uno a muchos, lo que implica ir en dirección contraria, y asociar en el ejemplo anterior 1 valor en la matriz U a 9 valores de la matriz F, como muestra la Figura 4. Para realizar la operación de deconvolución o en sentido contrario de la convolución, la cual definiremos como

$$\widetilde{F} \circledast Y = \vec{u}$$

es necesario expresar primero la convolución en términos matriciales. Para ello se vectorizará la matriz $U \in \mathbb{R}^{4\times4}$, de forma que $\vec{u} \in \mathbb{R}^{16\times1}$:

$$U = \begin{bmatrix} 4 & 5 & 8 & 7 \\ 1 & 8 & 8 & 8 \\ 3 & 6 & 6 & 4 \\ 6 & 5 & 7 & 8 \end{bmatrix} \Rightarrow \vec{u} = \begin{bmatrix} 4 \\ 5 \\ 8 \\ 7 \\ 1 \\ 8 \\ 8 \\ 8 \\ 3 \\ 6 \\ 6 \\ 4 \\ 6 \\ 5 \\ 7 \\ 8 \end{bmatrix}$$

para entonces utilizar el operador de multiplicación matricial. Ello hace

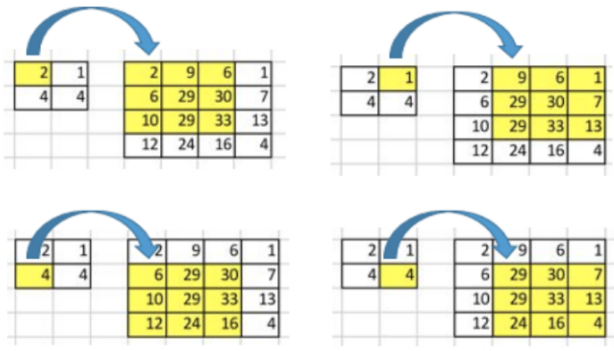

Figura 4: Relación de muchos a uno, operación en sentido contrario de la convolución, tomado de `https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0`.

necesario pasar el filtro $F \in \mathbb{R}^{3\times3}$ a una matriz que al multiplicarse, resulte en la salida vectorizada $\vec{y} \in \mathbb{R}^4$, con lo que el filtro F se reescribe en $\widetilde{F} \in \mathbb{R}^{4\times16}$ como se ilustra en la Figura 5. De esta forma, la multiplicación del filtro redimensionado $\widetilde{F} \in \mathbb{R}^{4\times16}$ con la imagen de entrada vectorizada $\vec{u}$, resulta en la salida vectorizada $\vec{y}$:

$$\widetilde{F}\,\vec{u} = \vec{y}$$

Recordemos que el objetivo consiste en ir del vector $\vec{y} \in \mathbb{R}^4$ al vector $\vec{u} \in \mathbb{R}^{16\times1}$, manteniendo la relación de 1 a 9 en este caso. Para esto, es necesario plantear la ecuación de la siguiente forma:

$$\widetilde{F}^T\,\vec{y} = \vec{u}$$

con lo cual es posible obtener la matriz original U, redimensionando $\vec{u}$. Lo anterior se ilustra en la Figura 6. Nótese entonces como el filtro transpuesto $\widetilde{F}^T$al multiplicarse con la señal $\vec{y}$ resultó en una señal de mayor dimensión $\vec{u}$. Los coeficientes de la matriz $\widetilde{F}$ pueden ser aprendibles para un conjunto de datos, **de forma que la operación de *upsampling* es en en sí misma aprendible**. La convolución transpuesta lleva ese nombre pues preserva la conectividad (pasando de 9 a 1, a de 1 a 9 para el ejemplo). El nombre puede ser engañoso, pues en realidad no implica que es necesario calcular una convolución y transponerla. En la página `https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md` se muestra la operación de la convolución transpuesta animada.

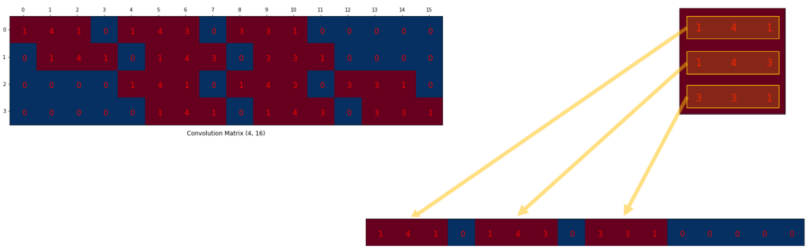

Figura 5: Redimensionamiento del filtro $F^{3\times3}$ al filtro $\widetilde{F} \in \mathbb{R}^{4\times16}$. Observe como los pixeles de la última columna y la última fila están en cero, para la primera fila de la matriz redimensionada $\widetilde{F}$, tomado de `https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0`.

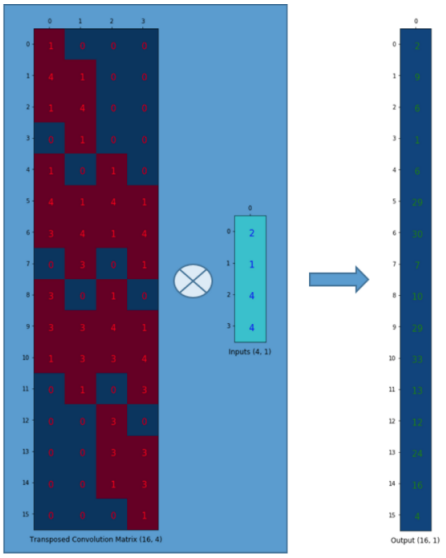

Figura 6: Multiplicación con la matriz del filtro transpuesta, tomado de `https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0`.

2.2 Autocodificador para eliminación del ruido

La función de pérdida de un autocodificador usual, sin regularizar,

$$L\left(\overrightarrow{x}, \overrightarrow{r}\right) = L\left(\overrightarrow{x}, m\left(\overrightarrow{x}\right)\right) = L\left(\overrightarrow{x}, g\left(f\left(\overrightarrow{x}\right)\right)\right)$$

como se comentó anteriormente tiende a aprender la función identidad, cuando se utiliza por ejemplo, una función de pérdida cuadrática $L\left(\overrightarrow{x}, \overrightarrow{r}\right) = \|\overrightarrow{r} - \overrightarrow{x}\|^2$.

Un autocodificador para la eliminación de ruido minimiza en cambio, la asiguiente función de pérdida

$$L\left(\overrightarrow{x}, g\left(f\left(\overrightarrow{\tilde{x}}\right)\right)\right)$$

donde $\overrightarrow{\tilde{x}}$ corresponde a una muestra corrupta por algún tipo de ruido. Este tipo de autocodificadores tienen entonces por objetivo remover o atenuar tal corrupción, en vez de simplemente *copiar* la entrada.

El autocodificador tiene por objetivo aprender la distribución del fenómeno de corrupción de la imagen o adición del ruido, la cual se expresa en la siguiente función de densidad condicional:

$$C\left(\overrightarrow{\tilde{x}} \mid \overrightarrow{x}\right).$$

El flujo de diseño de técnicas de filtrado y eliminación de ruido con un autocodificador es radicalmente diferente al flujo clásico: en vez de diseñar la función de transferencia del filtro, tomando en cuenta las características de la señal deseadas a conservar, eliminar o mejorar, con un enfoque basado en un autocodificador, se generan las muestras de la función de densidad $C\left(\overrightarrow{\tilde{x}} \mid \overrightarrow{x}\right)$ para estimar la función de densidad de la reconstrucción de una imagen, dada por $R\left(\overrightarrow{x} \mid \overrightarrow{\tilde{x}}\right)$. El proceso de entrenamiento consiste entonces en generar un par $\left\langle \overrightarrow{x}, \overrightarrow{\tilde{x}} \right\rangle$ para entrenar al autocodificador, y minimizar su función de pérdida, como lo muestra la Figura 7.

3 Autocodificadores convolucionales: La Arquitectura Completamente convolucional (Fully Convolutional network) y U-net

El uso típico de las redes convolucionales consiste en estimar una etiqueta $t \in \mathbb{Z}$ para una imagen $U \in \mathbb{R}^{a \times b}$, de forma que el modelo es una función $\tilde{t} = f\left(U\right)$. El problema de segmentación, se refiere a asignar una etiqueta a cada pixel específico, produciendo una imagen nueva a la salida, con la etiquetación por pixel, de forma que el modelo debe en este caso producir una matriz de estimaciones de la misma dimensión, o al menos similar, a la muestra de entrada $U \in \mathbb{R}^{a \times b}$, de forma que:

$$\widetilde{Y} = f\left(U\right)$$

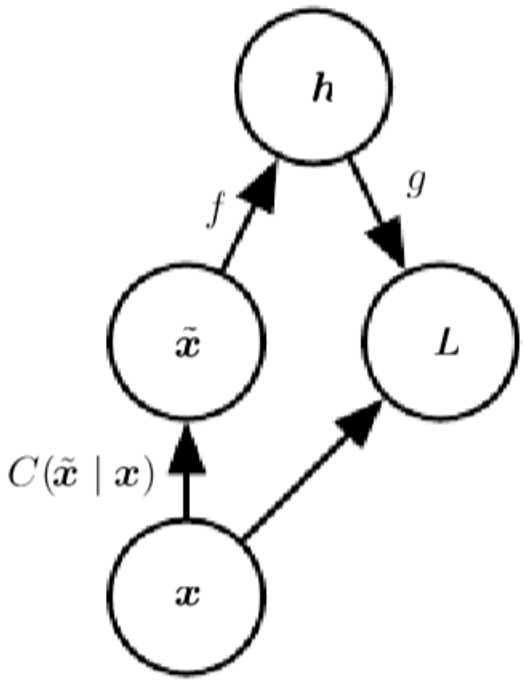

Figura 7: Diagrama de un Autocodificador para eliminar ruido.

Figura 8: Muestra de una imagen de *ground truth.*

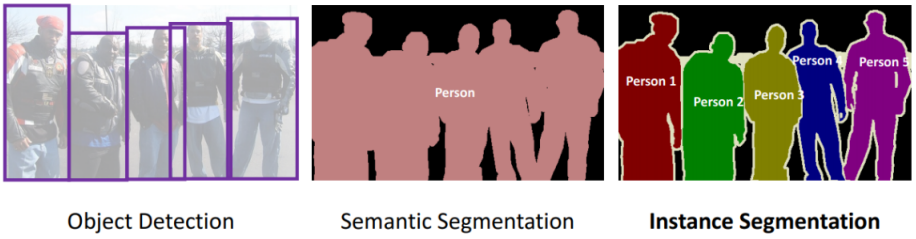

Figura 9: Segmentación de instancias.

con $\widetilde{Y} \in \mathbb{R}^{a \times b}$, donde cada pixel de tal estimación pertenece a un conjunto finito de clases, de forma que $\widetilde{Y}_{i,j} \in \mathbb{Z}$. Existen dos variantes del problema de segmentación, la primera se refiere a la **segmentación semántica** de los pixeles, donde cada etiqueta del pixel corresponde a clases semánticamente diferentes, como por ejemplo, el problema de segmentar los pixeles de personas y gatos en imágenes, sin importar cuántas apariciones de cada clase existan en la imagen de entrada U. La Figura 8 muestra una imagen de *ground truth* Y de ejemplo.

El problema de segmentación de instancias es definido entonces como la distinción de la ocurrencia distinta de cada clase en la imagen. En el ejemplo anterior, se refiere entonces a distinguir, con etiqueta distinta, la aparición de la persona 1, la persona 2, etc.. en la imagen de entrada. La Figura 9 muestra la diferencia entre los problemas de segmentación semántica y segmentación de instancias.

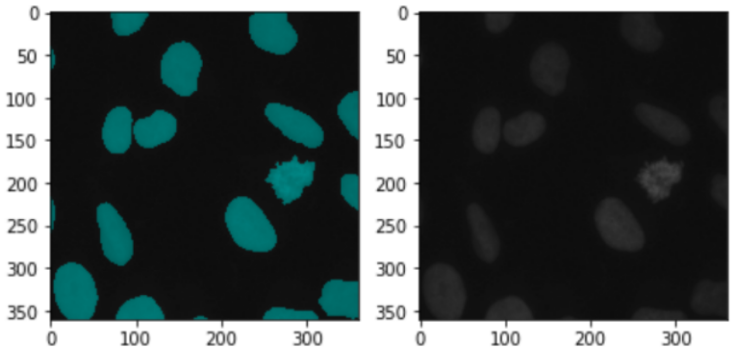

Figura 10: Segmentación de instancias en microscopía por fluorescencia.

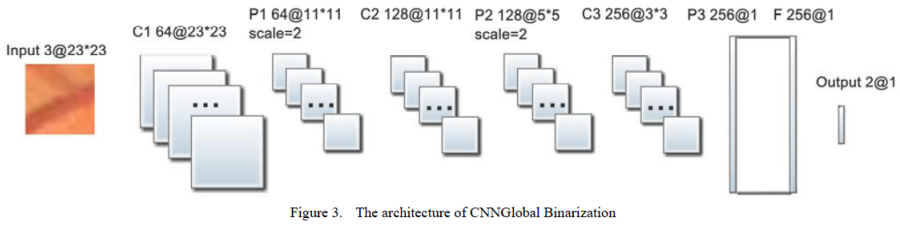

Figure 3. The architecture of CNNGlobal Binarization

Figura 11: Arquitectura de segmentación de vasos sanguíneos oculares propuesta en el artículo *Convolutional Neural Network for Retinal Blood Vessel Segmentation*. En este caso $c = d = 23$

El problema de segmentación de instancias es por mucho más complejo que el de segmentación semántica. En contextos donde la segmentación de instancias es necesaria, como por ejemplo la segmentación y conteo de células en imágenes de microscopía, como lo ilustrado en la Figura 10, se implementan *atajos* como por ejemplo la detección de límites entre las instancias.

Una modificación simple de las arquitecturas convolucionales vistas hasta ahora consiste en construir una red entrenada con múltiples ventanas $\mathcal{V} = \{V_1, V_2, \ldots, V_m\}$, con dimensiones $V_i \in \mathbb{R}^{c \times d}$ de la imagen $U \in \mathbb{R}^{a \times b}$, de forma que $a \gg c$ y $b \gg d$. Las ventanas pueden no son disjuntas, lo que posibilita la generación de $m = a \times b$ ventanas como máximo, con un paso o stride de 1. El modelo tiene una neurona de salida, que indica si el pixel sobre el que está centrado la ventana corresponde al fondo o al objeto de interés, utilizando codificación binaria.

Sin embargo, una arquitectura basada en la estimación de la clase con las ventanas $\mathcal{V}$ de tamaño arbitrario $c \times d$, presenta ciertas desventajas. La primera, se refiere a la gran cantidad de redundancia de los datos, sobre todo si usa un paso pequeño. Otra desventaja es la definición de las dimensiones de la venta-

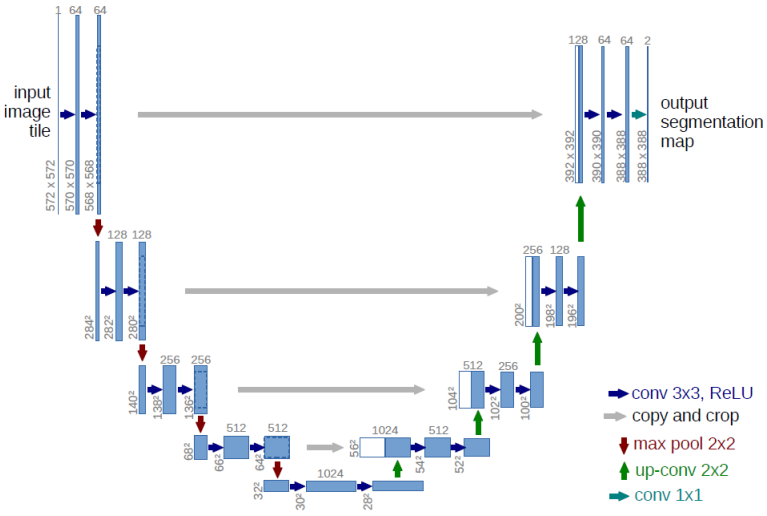

Figura 12: Arquitectura U-net.

na $c \times d$. Dimensiones altas de la ventana hace necesario la implementación de múltiples capas de *maxpooling*, lo que reduce la precisión espacial de la red a la salida, y aumenta el costo computacional total de entrenamiento para conjuntos de datos moderados. Sin embargo, ventanas más grandes toman en cuenta más datos del contexto, lo que mejora la predicción, aunque con menor precisión espacial. El uso de ventanas de menor tamaño imposibilita el uso de datos del contexto para que el modelo tome la decisión correcta.

La arquitectura U-net, basada en la arquitectura completamente convolucional (Fully Convolutional Network), resuelve los problemas de entrenamiento por ventanas y definición de su tamaño (relación precisión espacial/uso de contexto), implementando dos etapas en la red: una etapa de contracción o **codificación**, y otra de expansión o **decodificación**, y la alimentación no secuencial entre tales etapas, lo cual se ilustra en el diagrama de la Figura 12.

La función de pérdida: La función de pérdida puede consistir en el MSE: $\left\| Y - \widetilde{Y} \right\|^2$, o funciones más específicas para comparar señales binarias como el coeficiente de Dice la cual se puede usar para evaluar la similitud estructural entre dos imágenes binarias $U \in \mathbb{R}^{m \times n}$ y $V \in \mathbb{R}^{m \times n}$, y viene dado por:

$$S = \frac{2\left(\sum_j^m \sum_i^n U\left[i,j\right]\ V\left[i,j\right]\right)}{\left(\sum_j^m \sum_i^n U\left[i,j\right]\right) + \left(\sum_j^m \sum_i^n V\left[i,j\right]\right)}$$

donde el numerador equivale conceptualmente a la cantidad de pixeles del objeto de interés intersecados en ambas imágenes U y V : $2\left|U \cap V\right|$(con las barras denotando la cantidad de pixeles con 1's en la matriz) y el denominador

a la suma de los pixeles del objeto de interés en ambas imágenes: $|U| + |V|$, lo que entonces significa que se puede reescribir S como:

$$S = \frac{2\,|U \cap V|}{|U| + |V|}$$

Referencias

[1] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.