

Capítulo 2

Marco Téorico

En este capítulo se presentan los conceptos fundamentales necesarios para comprender los modelos utilizados en la detección de criaderos de mosquitos a partir de imágenes fotográficas. Asimismo, se expone una descripción general de la biología del mosquito vector, incluyendo su ciclo de vida y comportamiento, así como los factores ambientales que influyen en su reproducción y proliferación.

2.1. Biología de los Mosquitos

Los mosquitos son insectos que pertenecen al grupo de los artrópodos, al igual que los arácnidos, crustáceos y miriápodos. Una de sus principales características es la presencia de un exoesqueleto y apéndices articulados, como las patas y las antenas. Su esqueleto no crece una vez alcanzada la etapa adulta; por ello, los mosquitos deben mudar su exoesqueleto para poder desarrollarse durante las etapas juveniles. Desde el punto de vista anatómico, el cuerpo del mosquito se divide en tres regiones principales: cabeza, tórax y abdomen. Cada una de estas partes alberga órganos específicos (véase la Figura 2.1). A nivel mundial, se han identificado más de 3700 especies de mosquitos. Aunque en su mayoría habitan en ambientes exteriores, algunas especies pueden adaptarse a espacios interiores, y su actividad de alimentación puede ocurrir tanto de día como de noche. Las hembras suelen tener una mayor longevidad que los machos, con una esperanza de vida que puede oscilar entre dos y cuatro semanas. Si bien existe una gran diversidad de especies, en este trabajo nos enfocaremos en el mosquito *Aedes aegypti*, dada su relevancia epidemiológica en México y América Latina, especialmente en relación con la transmisión de

enfermedades como el dengue, zika y chikungunya [10].

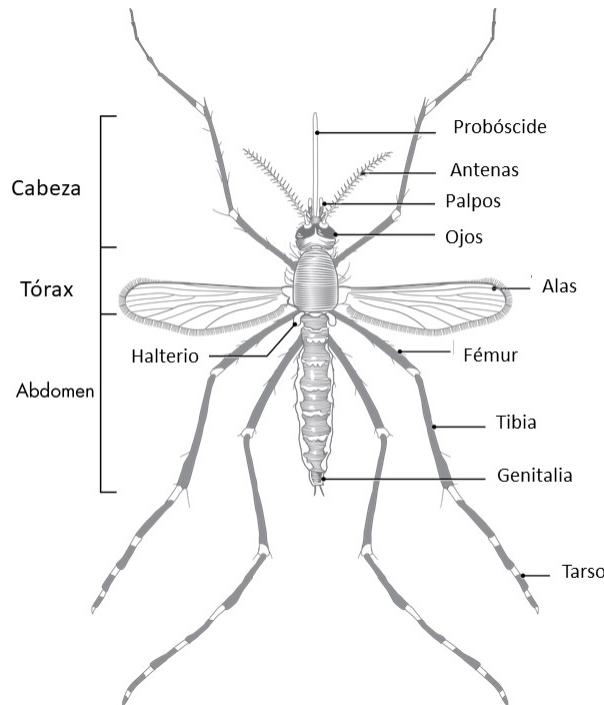


Figura 2.1: Anatomía de los mosquitos. El cuerpo del mosquito se divide principalmente en tres partes: cabeza, tórax, abdomen. Figura tomada de [11].

2.1.1. Ciclo de vida de los mosquitos

Tanto el *Aedes aegypti* como el resto de los mosquitos pasan por un ciclo de cuatro etapas en su desarrollo biológico o ciclo de vida, este ciclo esta conformado por huevos, larvas, pupas y adultos, ver Figura 2.2. Las primeras tres etapas son de carácter acuático, y la última correspondiente a la etapa de adultez, el cual es terrestre. Este ciclo puede tomar la duración de 12 a 15 días, dependiendo de las características ambientales en el que se desarrolle.

Los mosquitos hembra depositan sus huevos en cúmulos de agua o sobre objetos que contienen pequeñas cantidades de agua. Estos huevos, que pueden adherirse a las paredes de los recipientes o asentarse en el fondo, suelen pasar desapercibidos al ojo humano debido a su tamaño reducido. Presentan una forma alargada y elíptica; inicialmente tienen un color claro, pero con el paso de las horas se oscurecen. Los huevos son altamente resistentes y pueden sobrevivir en condiciones extremas, incluyendo

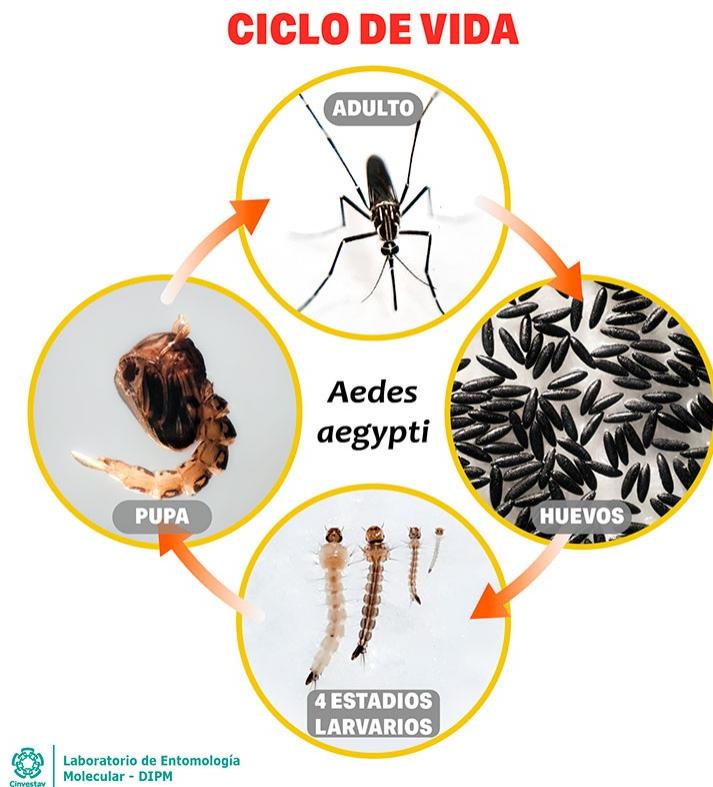


Figura 2.2: Ciclo de vida del mosquito *Aedes aegypti*. Este ciclo consta de cuatro etapas: huevos, 4 estadios larvarios, pupa y adulto. Figura tomada de [12].

sequías y variaciones de temperatura. Pueden permanecer viables entre 7 y 12 meses, manteniéndose en estado latente hasta que la humedad activa su eclosión. Esta etapa tiene una duración aproximada de 2 a 3 días. Una vez eclosionados, emergen las larvas, las cuales son altamente activas y se desplazan constantemente en busca de alimento, alimentándose principalmente de microorganismos como bacterias y hongos. Las larvas deben ascender periódicamente a la superficie del agua para respirar, y en esta etapa son visibles al ojo humano. A lo largo de su desarrollo, atraviesan cuatro estadios larvarios, en un periodo que suele durar entre 8 y 10 días. Posteriormente, las larvas se transforman en pupas, etapa en la que ocurren importantes procesos de metamorfosis para dar paso a la forma adulta. Aunque las pupas pueden moverse, no se alimentan, ya que utilizan la energía acumulada durante la fase larval. Esta fase tiene una duración aproximada de 2 días, bajo condiciones ambientales favorables. Una vez completada la metamorfosis, emergen los mosquitos adultos, liberándose de la cubierta pupal. En esta nueva etapa,

los mosquitos buscan fuentes de energía ricas en carbohidratos, como el néctar de flores o la pulpa de frutas [13].

2.1.2. Hábitats y factores ambientales que influyen en su reproducción

El desarrollo de los mosquitos depende en gran medida de la presencia de cúmulos de agua, ya que este elemento es fundamental para las primeras etapas de su ciclo de vida, por tanto todo ambiente acuático en el que se puedan desarrollar las etapas inmaduras de los mosquitos son llamados *criaderos de mosquitos*. Los criaderos de mosquitos naturalmente suelen ser charcos de agua, lagunas, estanques, pantanos, inclusive huecos de árboles u hojas de plantas acuáticas, sin embargo, debido al crecimiento en la producción de plásticos y otros productos se han originado criaderos de mosquitos artificiales como son botellas de plásticos, latas, cacharros, envases de vidrio, neumáticos, macetas, etc... estos objetos del uso cotidiano que son desechados integrándose al ambiente se han convertido en un hábitat para los mosquitos, puesto que pueden llegar a contener agua y pueden brindar protección de posibles depredadores.

También la temperatura es crucial para los huevos, un ambiente húmedo y una temperatura de entre 25 a 30°C suelen ser condiciones óptimas. Temperaturas por debajo de los 10°C ralentizan o detienen el desarrollo de los huevos y larvas. Los climas cálidos reducen el ciclo de vida de los mosquitos por lo que su reproducción es acelerada. También los ambientes con humedad alta favorecen a la supervivencia de los mosquitos adultos y la eclosión de sus huevos. Una vez alcanzada la etapa adulta, los mosquitos prefieren habitar en lugares frescos y húmedos, tales como zonas boscosas, selvas y cuevas. Algunas especies también se han adaptado a entornos urbanos y suburbanos, incluyendo jardines, parques, interiores de viviendas y edificaciones [14].

Transmisión de enfermedades a través de la picadura de mosquitos

La transmisión de enfermedades por mosquitos es una consecuencia natural de su proceso de alimentación y reproducción. Comprender los ciclos biológicos mediante los cuales los mosquitos infectan a los humanos es fundamental para establecer estrategias efectivas de prevención y control de las enfermedades transmitidas por vectores. En la Figura 2.3 se ilustran dichos ciclos biológicos para el caso del *Aedes*

aegypti, aunque es importante destacar que estos procesos son similares en la mayoría de las especies de mosquitos vectores.

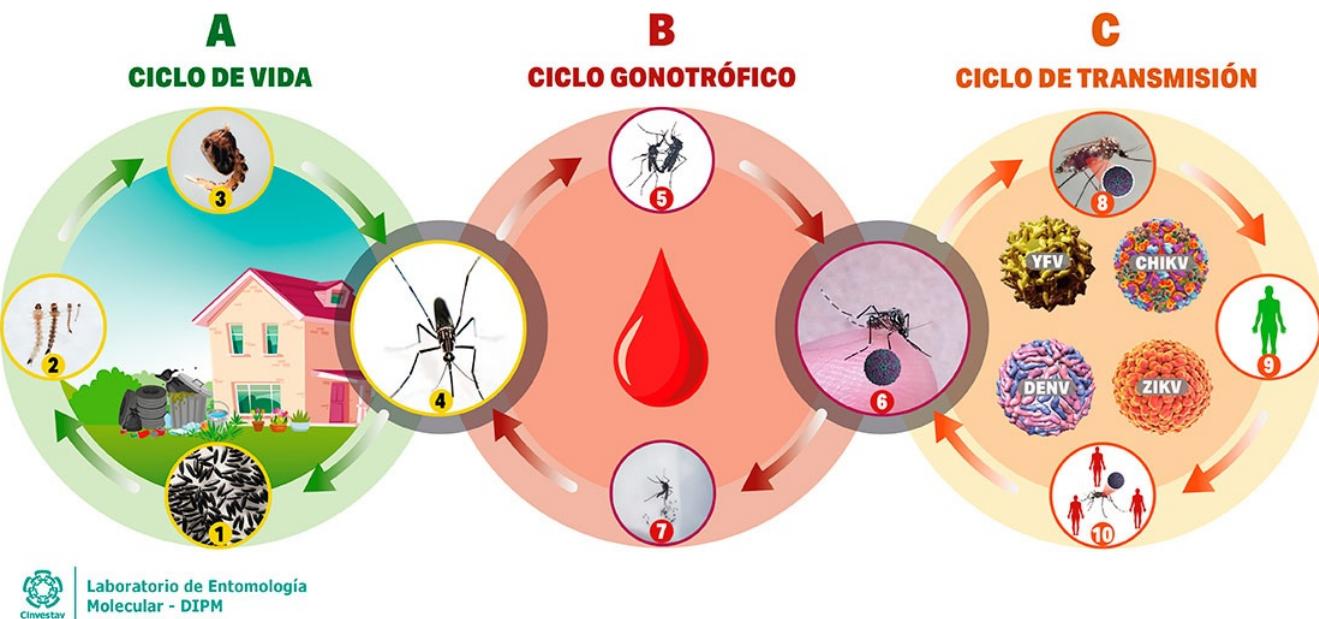


Figura 2.3: Ciclos biológicos que encadenan la transmisión de enfermedades a través de *Ae. aegypti*. Estos ciclos comprenden tres componentes principales: el ciclo de vida, el ciclo gonotrófico y el ciclo de transmisión. El primero describe el proceso mediante el cual el mosquito se desarrolla desde el huevo hasta alcanzar la fase adulta y reproducirse. El ciclo gonotrófico se refiere al intervalo entre una toma de sangre y la siguiente, es decir, el tiempo que transcurre desde que una hembra se alimenta de un hospedero hasta que repite este comportamiento. Finalmente, el ciclo de transmisión involucra el proceso por el cual un mosquito infectado transmite un virus a través de su picadura a personas susceptibles, contribuyendo así a la propagación de enfermedades en la población. Figura tomada de [12].

El ciclo de vida (A), junto con las etapas (1), (2), (3) y (4), ha sido descrito previamente en la Subsección 2.1.1. A partir de la etapa (4), el mosquito alcanza la fase adulta, momento en el cual comienza la búsqueda de fuentes de energía y se encuentra en disposición de reproducirse. En esta fase se inicia el ciclo gonotrófico (B). Cuando un macho y una hembra copulan, el macho transfiere espermatozoides junto con hormonas que inducen cambios fisiológicos y conductuales en la hembra, preparando su organismo para la formación de huevos (5). Para completar este proceso, la hembra busca fuentes de energía necesarias para la maduración de los huevos. Dado que las hembras de *Aedes aegypti* son antropófilas (es decir, muestran preferencia por alimentarse de humanos), son únicamente ellas las que se alimentan de sangre (6). Tras la ingestión de sangre, las hembras buscan un sitio de

oviposición adecuado para depositar los huevos (7). Durante la etapa de búsqueda de alimento (6), tiene lugar el ciclo de transmisión (C). Si el hospedero del cual se alimenta el mosquito está infectado con algún arbovirus, el virus es ingerido junto con la sangre, infectando al mosquito de por vida (8). En el siguiente ciclo gonotrófico, la hembra volverá a alimentarse, y si el nuevo hospedero es susceptible, el virus será transmitido a través de la saliva del mosquito durante la picadura (9). Así, los mosquitos infectados continúan propagando el virus en cada nuevo ciclo gonotrófico, contribuyendo a la diseminación de la enfermedad en la población humana susceptible (10). Los virus transmitidos comúnmente por *Aedes aegypti* incluyen: Virus de la Fiebre Amarilla (YFV), Virus del Dengue (DENV), Virus del Zika (ZIKV) y Virus del Chikungunya (CHIKV).

2.2. Aprendizaje Automático

La inteligencia artificial (IA) es un campo de la informática que tiene como objetivo desarrollar sistemas capaces de realizar tareas complejas que, tradicionalmente, requieren de la inteligencia humana, como el reconocimiento de patrones, la identificación de objetos o la toma de decisiones. Este enfoque dota a las máquinas de habilidades cognitivas similares a las humanas. A diferencia de los enfoques clásicos basados en reglas explícitas, en IA se busca que las computadoras aprendan a partir de datos, los cuales se convierten en su principal fuente de información y les permiten mejorar su rendimiento conforme reciben nuevos ejemplos. En general, los datos se dividen en tres conjuntos: los datos de entrenamiento, que se utilizan para que el modelo aprenda; los datos de validación, que sirven para ajustar los parámetros del modelo; y los datos de prueba, que permiten evaluar el desempeño del modelo una vez entrenado. Dentro de la IA, se encuentra un subcampo fundamental denominado *aprendizaje automático* o *Machine Learning* (ML). Este subcampo se enfoca en el diseño de algoritmos capaces de aprender a partir de los datos, y se clasifica comúnmente en tres categorías principales: *aprendizaje supervisado*, *aprendizaje no supervisado* y *aprendizaje por refuerzo*. En el aprendizaje supervisado, los modelos se entrena con datos etiquetados, es decir, cada entrada se asocia con una salida esperada. Algunos de los algoritmos más representativos en esta categoría son: la regresión lineal, la regresión logística, los árboles de decisión, las máquinas de vectores de soporte y las redes neuronales. Por otro lado, en el aprendizaje no supervisado no se cuenta con etiquetas, y los algoritmos buscan descubrir patrones o estructuras subyacentes dentro

de los datos. Entre los modelos más utilizados en esta categoría se encuentran: *k-means*, *kernel PCA*, y *Stochastic Neighbor Embedding*. Finalmente, el aprendizaje por refuerzo se basa en que un agente tome decisiones dentro de un entorno con el objetivo de maximizar una señal de recompensa acumulada. Este enfoque es ampliamente utilizado en videojuegos y robótica. Ejemplos de algoritmos en esta categoría incluyen el *Q-learning* y las *Deep Q-Networks*.

2.2.1. Redes Neuronales

El aprendizaje profundo, o *deep learning* en inglés, es un área del *machine learning* que se basa en modelos compuestos por unidades fundamentales denominadas perceptrones (véase la Figura 2.4) [15].

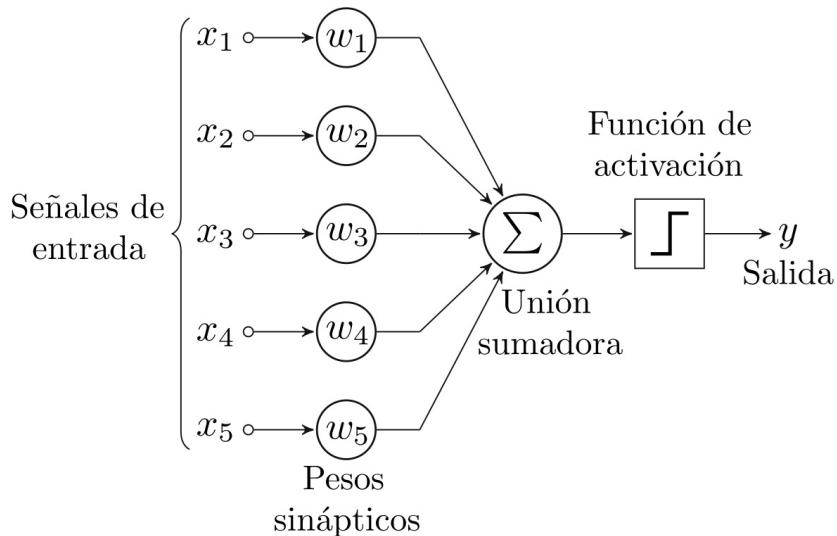


Figura 2.4: El Perceptrón. El perceptrón es una red con una sola neurona que recibe varios elementos de entrada (x_1, x_2, \dots), los pondera linealmente mediante un conjunto de pesos (w_1, w_2, \dots), y les suma un término adicional llamado *sesgo*. Esta combinación lineal se introduce en una función denominada *función de activación*, la cual produce un valor de salida (y), conocido como la activación de la neurona [16].

Cuando se agrupan múltiples neuronas en una misma estructura, se forma una capa. Estas capas suelen organizarse en forma de cadena, donde cada nivel representa una función específica. La extensión de estas estructuras da lugar al denominado *perceptrón multicapa* [17]. La arquitectura de un perceptrón multicapa se caracteriza por tener capas compuestas por neuronas interconectadas, y se distinguen principalmente tres tipos: la capa de entrada, las capas ocultas y la capa de salida. Las neuronas de la capa de entrada no realizan procesamiento como tal, sino que se encargan de recibir y almacenar

los valores característicos o señales x de entrada, que luego transmiten hacia la siguiente capa. Las capas ocultas llevan a cabo un procesamiento no lineal de las señales recibidas, transformándolas y enviándolas a la siguiente capa. Por último, la capa de salida genera la respuesta final de la red, a partir de los patrones de entrada procesados. Cada neurona dentro de la red recibe información, la procesa y transmite su salida a las neuronas de la capa siguiente. Las conexiones entre capas están orientadas en una única dirección y se propagan hacia adelante, por lo que este tipo de redes se denominan *redes neuronales profundas feedforward*. El número de capas ocultas determina la profundidad del modelo, lo cual da origen al término *deep learning* (véase la Figura 2.5).

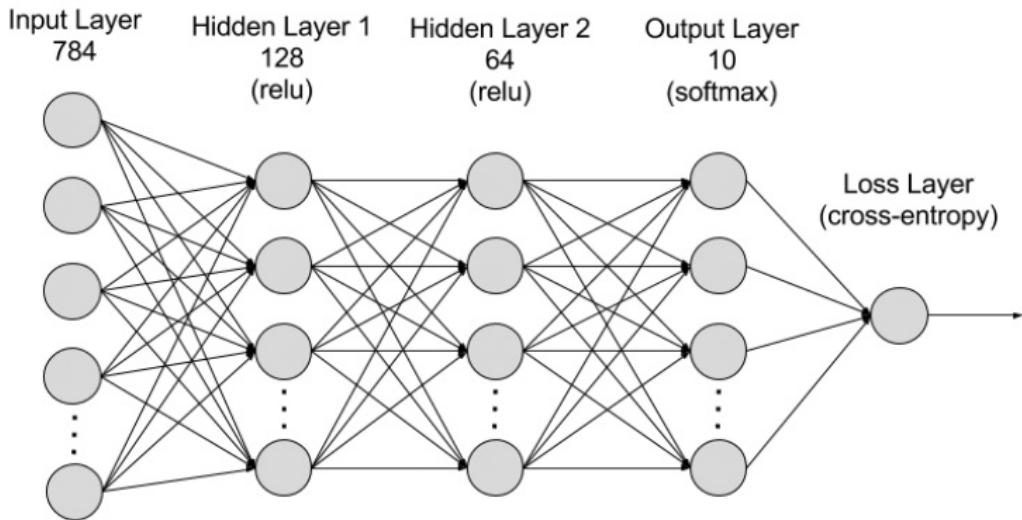


Figura 2.5: Ejemplo de perceptrón multicapa. En este ejemplo se muestra un perceptrón multicapa con 784 características de entrada, con dos capas ocultas, la primera con 128 neuronas y la segunda con 64, y una capa de salida de 10 neuronas, indicando que la clasificación lo realiza en 10 categorías. Figura tomada de [18].

Cada neurona dentro de una capa posee un conjunto específico de pesos, uno por cada neurona de la capa anterior, con los cuales pondera las activaciones recibidas. Además, a cada neurona se le suma un término denominado *sesgo*. Tanto los pesos como los sesgos constituyen los parámetros de la red y se ajustan de manera iterativa durante el proceso de entrenamiento. Este procedimiento de entrenamiento se denomina *backpropagation*, y será discutido más adelante.

Para formalizar matemáticamente estos conceptos, consideremos un perceptrón multicapa con C capas en total, de las cuales $C - 2$ son capas ocultas. Denotemos con n_c el número de neuronas en la capa c , donde $c = 1, 2, \dots, C$. Sean además $W^c = (w_{ji}^c)$ las matrices de pesos correspondientes a la capa

c , donde w_{ji}^c representa el peso que conecta la neurona j de la capa $(c - 1)$ con la neurona i de la capa c y u_i^c representa el sesgo asociado a la neurona i en la capa c .

Podemos calcular las activaciones de la capa c de la siguiente manera:

1. Para la capa de entrada, la activación se define como:

$$a_i^1 = x_i, \quad \text{para } i = 1, 2, \dots, n_1. \quad (2.1)$$

donde $X = (x_1, x_2, \dots, x_{n_1})$ representa el vector de entrada de la red.

2. Para las neuronas en las capas ocultas, la activación de la neurona i en la capa c se define como:

$$a_i^c = f \left(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} a_j^{c-1} + u_i^c \right), \quad \text{para } i = 1, 2, \dots, n_c \text{ y } c = 2, 3, \dots, C-1. \quad (2.2)$$

3. Para la capa de salida, la activación de la neurona i se define como:

$$y_i = a_i^C = f \left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C \right), \quad \text{para } i = 1, 2, \dots, n_C, \quad (2.3)$$

donde $Y = (y_1, y_2, \dots, y_{n_C})$ es el vector de salida de la red.

En las expresiones anteriores, la función f representa la función de activación, la cual suele elegirse entre funciones como la sigmoide, la tangente hiperbólica o la ReLU (véase Tabla 2.1).

Sigmoide	Tangente hiperbólica	ReLU
$\sigma(x) = \frac{1}{1 + e^{-x}}$	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f(x) = \max(0, x)$

Tabla 2.1: Funciones de activación más comunes en redes neuronales.

El diseño de una red neuronal implica seleccionar el número de capas, el número de neuronas en cada una y la función de activación. Estos hiperparámetros dependen del problema específico. Las capas de entrada y salida están determinadas por el número de características de entrada y el número de clases

o variables objetivo. Sin embargo, la elección del número de capas ocultas y neuronas en ellas no tiene una regla exacta y constituye un área activa de investigación. En la práctica, se prueban diferentes arquitecturas y se elige aquella que ofrezca el mejor desempeño, ya que varias configuraciones pueden resolver eficientemente un mismo problema.

El aprendizaje de las redes neuronales se realiza mediante el algoritmo *backpropagation*, un método supervisado donde la red ajusta sus parámetros (pesos y sesgos) para minimizar una función de error que mide la discrepancia entre la salida generada por la red y la salida deseada. El entrenamiento se formula como un problema de optimización:

$$\min_W E \quad (2.4)$$

donde W es el conjunto de parámetros de la red y E es la función de error, por ejemplo, el error cuadrático medio:

$$E = \frac{1}{N} \sum_{n=1}^N e(n), \quad e(n) = \frac{1}{n_C} \sum_{i=1}^{n_C} (s_i(n) - y_i(n))^2, \quad (2.5)$$

siendo $Y(n) = (y_1(n), \dots, y_{n_C}(n))$ la salida de la red y $S(n) = (s_1(n), \dots, s_{n_C}(n))$ la salida deseada para el dato n . El entrenamiento busca encontrar W^* tal que la función de error se minimice, logrando que la red produzca salidas cercanas a las respuestas verdaderas.

El aprendizaje en redes neuronales multicapa consiste en minimizar una función de costos. Debido a la no linealidad introducida por las funciones de activación, esta función de costos no es convexa y puede contener múltiples mínimos locales. Por ello, el entrenamiento se plantea como un problema de optimización no lineal, abordado usualmente mediante métodos basados en el gradiente. El algoritmo más común es el *descenso de gradiente estocástico*, en el cual cada parámetro w se actualiza según

$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w}, \quad (2.6)$$

donde α es la tasa de aprendizaje y $e(n)$ el error para el dato n . Este procedimiento da lugar al algoritmo de *backpropagation* o *regla delta generalizada*, que ajusta los pesos y sesgos propagando el error hacia atrás a través de la red.

En el caso de la última capa (capa C), los pesos w_{ji}^{C-1} entre la neurona j en la capa $C - 1$ y la neurona i en la salida se actualizan como:

$$w_{ji}^{C-1}(n) = w_{ji}^{C-1}(n-1) - \alpha \frac{\partial e(n)}{\partial w_{ji}^{C-1}}, \quad (2.7)$$

donde la derivada se desarrolla como:

$$\frac{\partial e(n)}{\partial w_{ji}^{C-1}} = -(s_i(n) - y_i(n)) \frac{\partial y_i(n)}{\partial w_{ji}^{C-1}}. \quad (2.8)$$

Dado que la salida de la red se ve afectada por dicho peso, su derivada es:

$$\frac{\partial y_i(n)}{\partial w_{ji}^{C-1}} = f' \left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C \right) a_j^{C-1}(n). \quad (2.9)$$

Definimos entonces:

$$\delta_i^C(n) = -(s_i(n) - y_i(n)) f' \left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C \right), \quad (2.10)$$

y finalmente la regla de actualización para los pesos es:

$$w_{ji}^{C-1}(n) = w_{ji}^{C-1}(n-1) + \alpha \delta_i^C(n) a_j^{C-1}, \quad (2.11)$$

para $j = 1, 2, \dots, n_{C-1}$, $i = 1, 2, \dots, n_C$. Esta expresión muestra que el cambio en el peso depende del producto entre la activación de la neurona de la capa anterior y el valor δ , que cuantifica el error en la neurona de salida.

Un procedimiento similar para obtener la ecuación 2.11 se realiza para modificar el sesgo de una neurona en la capa $C - 1$, por lo que se llega a la expresión

$$u_i^C(n) = u_i^C(n-1) + \alpha \delta_i^C(n), \text{ para } i = 1, 2, \dots, n_C. \quad (2.12)$$

Para desarrollar las expresiones del segundo caso (el resto de las capas de la red neuronal) podemos basarnos en el peso de una neurona k en la capa $C - 2$ en la de otra neurona j en la capa $C - 1$ y llegar a

las expresiones de w_{kj}^{C-2} y u_j^{C-1} , sin embargo esto se puede generalizar para el peso y sesgo de cualquier neurona en la capa c a la capa $c+1$ para $c = 1, 2, \dots, C-1$. Mediante un proceso similar al mostrado se pueden llegar a la expresión del peso

$$w_{kj}^c(n) = w_{kj}^c(n-1) + \alpha \delta_j^{c+1}(n) a_k^c(n) \quad (2.13)$$

para $k = 1, 2, \dots, n_c$, $j = 1, 2, \dots, n_{c+1}$ y $c = 1, 2, \dots, C-2$, donde la activación de una neurona k en la capa c para el dato de entrenamiento n es $a_k^c(n)$ y $\delta_j^{c+1}(n)$, que viene dado por la siguiente expresión

$$\delta_j^{c+1}(n) = f' \left(\sum_{k=1}^{n_c} w_{kj}^c a_k^c + u_j^c \right) \sum_{i=1}^{n_{c+1}} \delta_i^{c+2}(n) w_{ji}^c, \quad (2.14)$$

de igual forma la expresión para los sesgos se generalizan, llegando a la siguiente expresión

$$u_j^{c+1}(n) = u_j^{c+1}(n-1) + \alpha \delta_j^{c+1}(n) \quad (2.15)$$

para $j = 1, 2, \dots, n_{c+1}$ y $c = 1, 2, \dots, C-2$.

En este desarrollo se consideró f como la función de activación sin especificar, en este sentido dependiendo de la selección de esta función (sigmoide, tangente hiperbólica, ReLU, etc.) basta con obtener su derivada para obtener un término más específico de δ .

Un aspecto fundamental para obtener buenos resultados es la regularización, la cual previene el sobreajuste y mejora la capacidad del modelo para generalizar. Entre los métodos de regularización más utilizados se encuentra el *dropout*, que consiste en desactivar aleatoriamente ciertas neuronas con una probabilidad p . Esta técnica se aplica comúnmente a las capas completamente conectadas, aunque también puede extenderse a las capas convolucionales. Otro método importante es la *batch normalization*, que normaliza las activaciones intermedias para evitar que los valores de salida de una capa sean excesivamente grandes o pequeños, estabilizando así el proceso de entrenamiento.

Aunque las redes neuronales fueron inspiradas vagamente por la neurociencia, éstas no intentan modelar el comportamiento del cerebro humano, las redes neuronales modernas suelen ser más bien dirigidas por las matemáticas que buscan aproximar una función f^* donde un clasificador es una función tal que $y = f^*(x)$ donde x son los datos de entrada y y la clase, de esta forma los datos de entrada no

son más que aproximación a $f^*(x)$ con etiquetas $y \approx f^*(x)$. De acuerdo con el *teorema de aproximación universal* cualquier función continua definida en un subconjunto compacto de \mathbb{R}^d puede ser aproximada uniformemente, con una red neuronal, usando la función lineal en la capa de salida y al menos una capa oculta con la función sigmoide.

2.2.2. Redes Neuronales Convolucionales

Después de las redes neuronales multicapa surgieron algunas variantes, entre ellas las redes neuronales convolucionales, o *convolutional neural networks* (CNNs) en inglés [19]. Estas redes se desarrollaron para resolver un problema importante que presentaba el perceptrón multicapa al procesar y clasificar imágenes: estos modelos no eran capaces de detectar patrones espaciales, ya que consideraban cada píxel de manera independiente, perdiendo así la correlación espacial entre ellos. Además, cuando se trataba de imágenes de gran dimensión, el entrenamiento del perceptrón multicapa se volvía computacionalmente muy costoso.

Las redes neuronales convolucionales solucionaron estos problemas al introducir capas convolucionales, de *pooling* y completamente conectadas, las cuales permitieron preservar la información espacial y, al mismo tiempo, reducir la dimensionalidad de las imágenes, mejorando significativamente la eficiencia del modelo. Las CNNs fueron diseñadas especialmente para procesar datos con estructura cuadricular, como es el caso de las imágenes, que pueden representarse como una cuadrícula bidimensional de píxeles.

La principal característica de las CNNs es su capacidad para identificar patrones en las imágenes, tales como bordes, esquinas y texturas. A medida que se concatenan múltiples capas convolucionales, las redes son capaces de capturar estructuras cada vez más complejas, lo que las hace especialmente útiles en tareas de clasificación de imágenes. Hoy en día, las CNNs han revolucionado el campo del aprendizaje automático en lo que respecta al procesamiento de imágenes y videos, siendo la base de modelos utilizados para tareas como clasificación, segmentación y detección de objetos.

La arquitectura de una red neuronal convolucional está compuesta principalmente por la entrada (*input*), capas convolucionales y una red neuronal completamente conectada (es decir, un perceptrón multicapa). En la entrada de la red se reciben datos con estructura cuadricular; en el caso de imágenes, se representa una cuadrícula de dimensiones ancho b y alto h , junto con su profundidad, que comúnmente

es tres debido a los canales de color RGB.

Una vez recibida la información de entrada, esta es procesada por las capas convolucionales mediante una operación llamada convolución, que se define de la siguiente forma:

$$S(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n), \quad (2.16)$$

donde $I(i, j)$ representa el valor del píxel de la imagen en la posición (i, j) , $K(m, n)$ es un filtro (también llamado *kernel*) o matriz de pesos de tamaño (m, n) —usualmente cuadrada—, y $S(i, j)$ es el resultado de la convolución en la posición (i, j) , que da como salida un único valor.

Esta operación se aplica desplazándose por toda la cuadrícula de entrada. El paso con el que se mueve el filtro se denomina *stride*; por ejemplo, un *stride* de 1 indica que el filtro se moverá un píxel a la vez tanto en dirección horizontal como vertical. Este parámetro constituye un hiperparámetro de la red (véase la Figura 2.6).

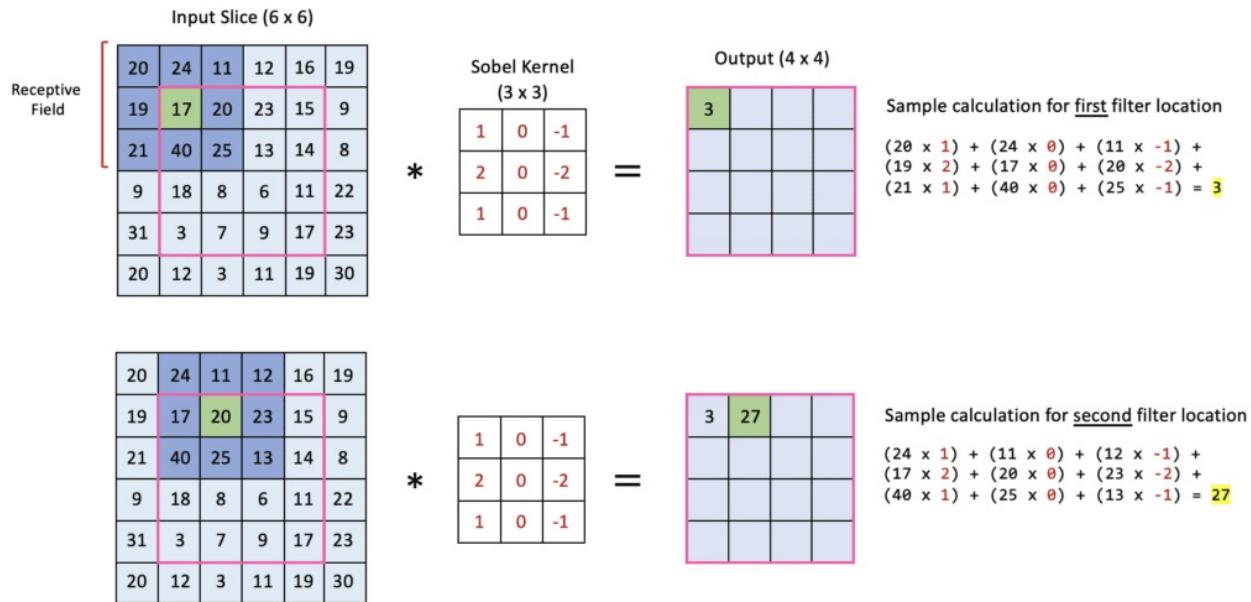


Figura 2.6: Ejemplo de convolución. Se muestra una matriz de entrada de tamaño 6×6 y un kernel de tamaño 3×3 . La operación de convolución se ilustra para las posiciones $(0, 0)$ y $(0, 1)$, donde se realiza la multiplicación elemento a elemento entre la matriz de entrada y el kernel. La suma de estos productos da como resultado un único valor, correspondiente a la posición (i, j) en la matriz de salida, la cual tiene tamaño 4×4 (véase la ecuación 2.16). La convolución se completa desplazando el kernel sobre toda la matriz de entrada hasta llenar la matriz de salida. Figura tomada de [20].

En las capas convolucionales se suelen aplicar múltiples filtros. El número de filtros o *kernels* en cada capa también es un hiperparámetro importante. El objetivo es que cada filtro aprenda a detectar un patrón específico dentro de la imagen.

Un punto a considerar es que las dimensiones de la matriz de salida suelen ser menores que las de la matriz de entrada, como se muestra en el ejemplo de la Figura 2.6, lo cual no siempre es deseable. Lo ideal es preservar las dimensiones originales de la entrada. Para solucionar este problema, se define otro hiperparámetro en las CNN: el *padding*. El padding permite conservar las dimensiones originales de la imagen al agregar bordes adicionales, usualmente con ceros.

Existen diversas técnicas de *padding*, como el *zero padding*, *same padding* o *valid padding*. La más utilizada es el *zero padding*, que consiste en llenar simétricamente con ceros la matriz de entrada. De esta forma, al aplicar la operación de convolución sobre la nueva matriz extendida, se preservan las dimensiones espaciales originales.

Una vez realizada la convolución sobre toda la cuadrícula de entrada, se obtiene una matriz resultante que posteriormente se pasa por una función de activación. Esta función transforma cada elemento de la matriz, generando así lo que se conoce como un *mapa de activación*. El número de filtros o *kernels* determina la profundidad del mapa de activación, mientras que su altura y anchura suelen coincidir con las de la cuadrícula de entrada (en caso de aplicar padding adecuado).

Posteriormente, el mapa de activación es procesado por una capa de *pooling*, que también utiliza una ventana deslizante, pero sin pesos. En lugar de aplicar una operación de convolución, esta capa aplica una operación como el máximo (*max*) o el promedio (*mean*) sobre los valores contenidos en la ventana. La operación más común es el *max pooling*, donde se toma el valor máximo dentro de cada ventana. Este proceso reduce las dimensiones espaciales (alto y ancho), mientras que la profundidad se mantiene (véase figura 2.7).

El objetivo del *pooling* es reducir la resolución espacial del mapa de activación, conservando la información relevante. Se puede pensar en este proceso como un resumen de la información del mapa de activación, y la matriz resultante se utiliza como entrada para la siguiente capa convolucional.

El proceso de un bloque convolucional se ejemplifica en la Figura 2.8. Un bloque convolucional puede diseñarse con distintas capas convolucionales y diferentes filtros. A medida que se encadenan varios bloques convolucionales, las dimensiones del mapa de activación se reducen en ancho y alto, pero

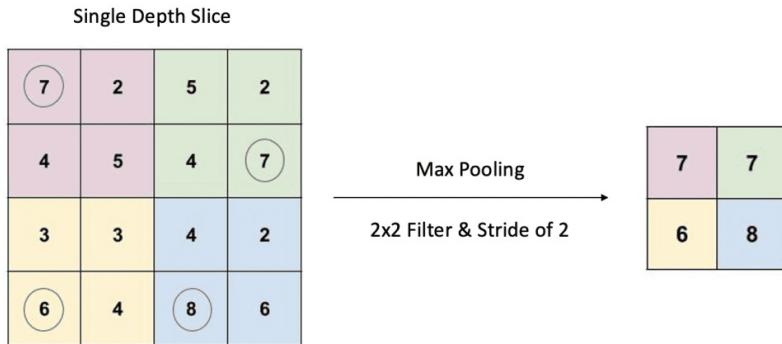


Figura 2.7: Ejemplo de *pooling*. En este ejemplo, la matriz de entrada tiene dimensiones 4×4 , y el filtro de *pooling* es de tamaño 2×2 . Se aplica la operación de *max pooling*, que consiste en seleccionar el valor máximo dentro de cada ventana de la matriz de entrada. Al superponer el filtro sobre la matriz y desplazarse por toda ella, se obtiene una matriz de salida de tamaño 2×2 . Figura tomada de [20].

aumentan en profundidad. En cierto punto del diseño de la arquitectura, este mapa de activación, de dimensiones $b \times h \times p$, se vectoriza como un vector columna de dimensión $(b \cdot h \cdot p) \times 1$. Este nuevo vector es el que se proporciona a un perceptrón multicapa completamente conectado (*fully connected*), encargado de realizar la clasificación y determinar la salida final del modelo.

Es importante mencionar que los pesos y parámetros de una red convolucional se ajustan de manera iterativa mediante el algoritmo de retropropagación (*backpropagation*); los detalles del entrenamiento pueden consultarse en [21].

En términos generales, se ha presentado una descripción del funcionamiento básico de las redes neuronales convolucionales. No obstante, existen otros aspectos relevantes en el diseño de una CNN que pueden consultarse en diversos libros y artículos especializados.

2.3. Detección de objetos

2.3.1. Conceptos clave y métricas

En visión computacional existen varias tareas fundamentales relacionadas con la identificación de objetos en una imagen. La primera es la *clasificación*, que se enfoca en asignar una categoría a una imagen completa según un conjunto de clases predefinidas. Otra tarea es la *detección de objetos*, que no

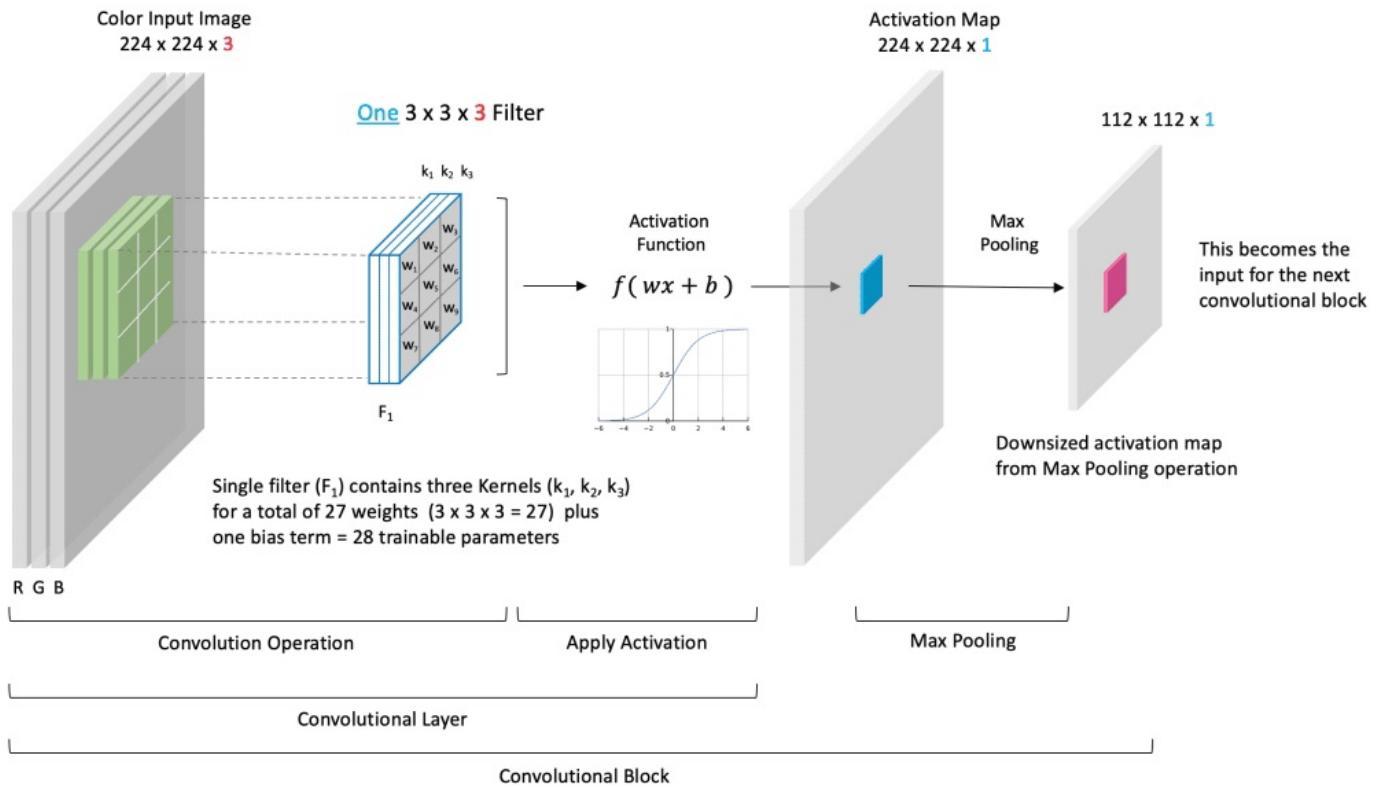


Figura 2.8: Ejemplo de bloque convolucional. Se muestra una matriz de entrada tridimensional de tamaño $224 \times 224 \times 3$, que representa una imagen a color con canales R, G y B. Se aplica un kernel de tamaño $3 \times 3 \times 3$, realizando la operación de convolución, que se extiende naturalmente a tres dimensiones. Cada operación de convolución produce un valor escalar, el cual es transformado mediante una función de activación y almacenado en el mapa de activación correspondiente. En este ejemplo, al emplear un solo filtro, se obtiene un mapa de activación de dimensión $224 \times 224 \times 1$. Posteriormente, se aplica *pooling* para reducir las dimensiones espaciales, resultando en una matriz de salida de $112 \times 112 \times 1$, que sirve como entrada para la siguiente capa convolucional. Figura tomada de [20].

solo identifica la presencia de un objeto, sino que también determina su ubicación dentro de la imagen mediante el uso de cajas delimitadoras, lo cual la convierte en una tarea más compleja. Finalmente, la *segmentación de imágenes* representa un nivel de precisión aún mayor: en lugar de encuadrar los objetos con rectángulos, esta técnica asigna una etiqueta a cada píxel, delimitando con exactitud las regiones que pertenecen a cada objeto (véase la Figura 2.9).

Existen distintas técnicas desarrolladas para abordar cada una de las tareas mencionadas anteriormente, lo que ha dado lugar a modelos especializados en cada una de ellas. Sin embargo, la tarea de interés en esta tesis es la **detección de objetos**. Para comprender los modelos utilizados en esta área,

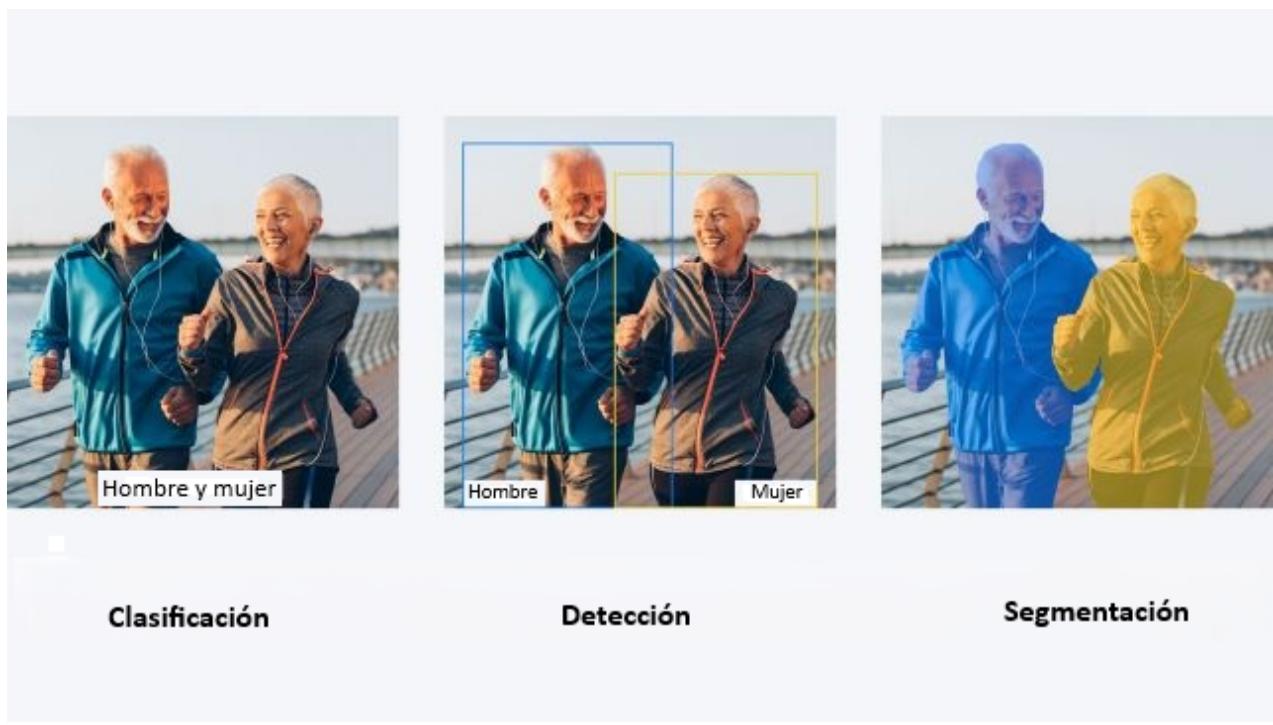


Figura 2.9: Tareas de visión por computadora. Clasificación, detección y segmentación. Figura modificada de [22].

es fundamental tener claros algunos conceptos clave.

Cuando se entrena un modelo de detección, este devuelve predicciones sobre la ubicación y la categoría de los objetos presentes en una imagen. Comúnmente, dichas predicciones se representan mediante recuadros denominados *bounding boxes*. Existen dos formas comunes de describir las coordenadas de una *bounding box*:

1. Mediante las coordenadas de la esquina superior izquierda (x_{tl}, y_{tl}) y la esquina inferior derecha (x_{br}, y_{br}), representándose como $(x_{tl}, y_{tl}, x_{br}, y_{br})$.
2. Mediante las coordenadas del centro del recuadro (x_c, y_c), junto con su ancho w y alto h , representándose como (x_c, y_c, w, h) .

No obstante, los modelos pueden cometer errores en sus predicciones, o bien pueden realizarlas de forma correcta. En el contexto de la detección de objetos, las predicciones se clasifican principalmente en tres categorías:

- **True Positive (TP)**: Son las predicciones correctas realizadas por el modelo, en las que el objeto ha sido detectado y clasificado correctamente.
- **False Positive (FP)**: Son predicciones incorrectas en las que el modelo detecta un objeto que no está presente o lo clasifica erróneamente.
- **False Negative (FN)**: Corresponden a los objetos presentes en la imagen que el modelo no logra detectar.

Para identificar a qué categoría pertenecen las predicciones realizadas por un modelo de detección de objetos, es necesario definir una métrica conocida como *Intersection over Union (IoU)*, la cual mide el grado de superposición entre la caja predicha y la caja real (también llamada *ground truth*). Esta métrica se calcula mediante la siguiente expresión:

$$IoU = \frac{\text{Área de intersección}}{\text{Área de unión}}, \quad (2.17)$$

donde la intersección corresponde al área común entre ambas cajas y la unión representa el área total combinada de ambas cajas.

Para determinar si una predicción se considera un *True Positive (TP)*, se define un umbral de IoU. Por ejemplo, si se establece un umbral de 0.5, se exige que al menos haya una superposición del 50 % entre la caja predicha y la verdadera (*IoU threshold = 50 %*).

Además del umbral de IoU, los modelos suelen utilizar otro parámetro conocido como umbral de confianza (*confidence threshold* o simplemente *thresh*), que define qué predicciones se consideran válidas según su probabilidad asignada. En la mayoría de los modelos, este umbral se fija en 0.5, lo que implica que las predicciones con una probabilidad inferior al 50 % no serán tomadas en cuenta.

En algunos casos, un modelo puede generar múltiples predicciones para un mismo objeto. Para resolver esta situación se emplea una técnica llamada *Non-Maximum Suppression (NMS)*, la cual elimina las cajas redundantes conservando únicamente aquella con la mayor puntuación de confianza.

Entre las métricas más utilizadas en la detección de objetos se encuentra la *precision*, que se define como la proporción entre el número de predicciones correctas (TP) y el total de predicciones realizadas (TP + FP). Matemáticamente, se expresa como:

$$\text{precision} = \frac{TP}{TP + FP}. \quad (2.18)$$

La segunda métrica es la sensibilidad (*recall* en inglés), es la proporción de predicciones correctamente predichas y el total de objetos reales que debió predecir, se calcula de la siguiente manera

$$\text{recall} = \frac{TP}{TP + FN}. \quad (2.19)$$

Puede presentarse el caso en que un modelo realice muchas predicciones generando un gran número de cajas delimitadoras, lo cual puede conducir a un buen *recall* pero a una mala *precision*. Por el contrario, si el modelo realiza pocas predicciones, es posible que tenga una alta *precision*, pero que no detecte todos los objetos reales, lo que resultaría en un bajo *recall*. Para encontrar un equilibrio entre ambas métricas, se define una tercera métrica denominada *F1 score*, la cual se calcula mediante la siguiente expresión:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (2.20)$$

Cuando los valores de *precision* y *recall* son similares y elevados, el F1-score también será alto. En cambio, si una de las dos métricas es baja, el F1-score disminuirá, reflejando el desequilibrio entre ambas.

Además, es posible calcular valores de *precision* y *recall* para cada predicción y representar gráficamente la curva *precision-recall*. El área bajo esta curva se conoce como *precisión promedio* (*Average Precision*, o AP), y permite evaluar el desempeño del modelo para una categoría específica. Si se calcula el AP para cada una de las clases del conjunto de datos y luego se promedian estos valores, se obtiene la *media de la precisión promedio* (*mean Average Precision*, o mAP), que proporciona una medida global del rendimiento del modelo. A mayor valor de mAP, mejor será el desempeño general del modelo en la tarea de detección de objetos.

Por último, cabe mencionar el concepto de *anclas* (*anchors*), que son cajas predefinidas con diferentes tamaños y proporciones. Estas cajas ayudan al modelo a detectar objetos de distintas formas y escalas en una imagen. Durante el entrenamiento, las anclas se ajustan de manera automática para adaptarse mejor a los objetos presentes en las categorías del conjunto de datos.

2.3.2. Métodos de detección

R-CNN

Las redes reuronales convolucionales basadas en regiones (R-CNN por sus siglas en inglés) marcaron un precedente en la tarea de detección de objetos dentro de la visión por computadora [23]. Fue propuesta en 2014 por Ross Girshick y colaboradores. El procedimiento general de este método se observa en 2.10.

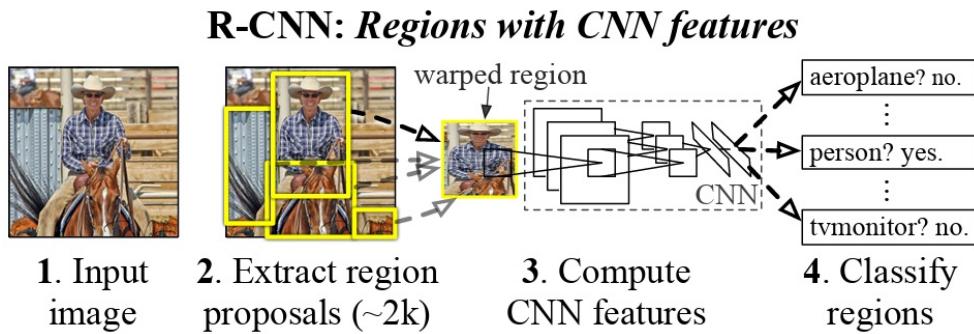


Figura 2.10: **Regiones con características a partir de un CNN.** 1. En el paso uno se ingresa la imagen que contiene los objetos de interés a detectar. 2. Se extraen regiones de la imagen, en donde probablemente hay un objeto. 3. Se procesan las regiones para obtener las características a través de una CNN. 4. Se clasifican las regiones de acuerdo a las categorías. Figura tomada de [23].

Detallemos cada uno de los pasos.

1. Se ingresa la imagen de acuerdo a sus dimensiones.
2. De la imagen ingresada se extraen propuestas de regiones donde es posible encontrar los objetos de interés pertenecientes a las categorías que se desean detectar. Para este paso, se utiliza un método independiente para extraer las regiones, el método utilizado es la búsqueda selectiva (*Selective Search*). Este algoritmo comienza segmentando pequeñas regiones de la imagen basándose en la similitud de colores, texturas, bordes y tamaños. A medida que se avanza en las iteraciones, mediante clustering se van agrupando regiones de tamaños pequeños para formar regiones más grandes (véase la Figura 2.11). Este proceso continua hasta obtener alrededor de 2000 regiones de propuesta.

Cada una de estas regiones extraídas se utilizan para entrenar la CNN, sin embargo es necesario proporcionarles una etiqueta correspondiente, para esto se utiliza el IoU con las cajas verdaderas,

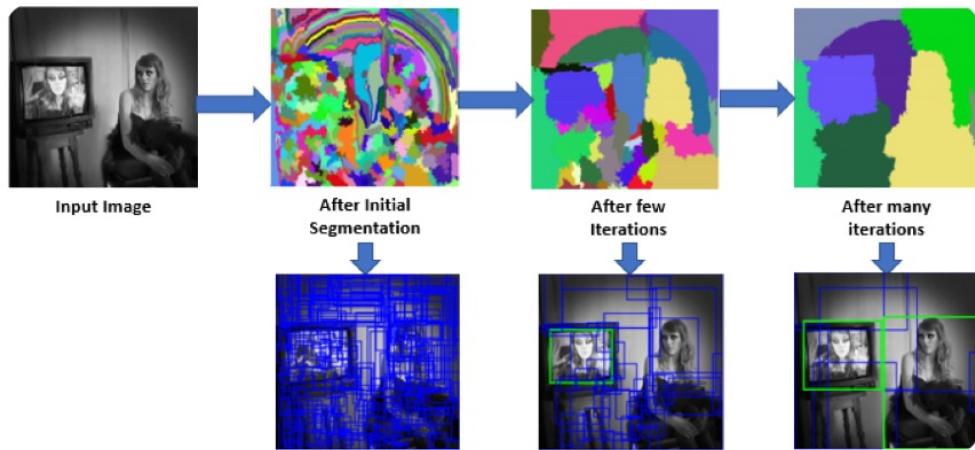


Figura 2.11: Ejemplo de búsqueda selectiva. En este ejemplo se observa que el conjunto de regiones de propuestas disminuye debido al agrupamiento entre estas en cada iteración. Cada propuesta de región se delimita con una caja, en este ejemplo el objeto a detectar es la categoría mujer, y las cajas verdaderas se muestran en verde. Podemos apreciar que este método aproxima muy bien algunas regiones de propuestas a las verdaderas. Figura tomada de [24].

las que tengan una superposición mayor a 0.5 con una caja verdadera de alguna categoría se le asigna su misma categoría, las que tengan un valor menor o cero de superposición son asignadas a una nueva categoría llamada fondo (*background* en inglés) (para este método siempre añadimos la categoría de fondo a las categorías ya establecidas).

3. Una vez identificadas las propuestas de regiones con su correspondiente etiqueta, estas son usadas para entrenar una CNN, en este caso los autores emplearon un CNN AlexNet, la arquitectura de esta red neuronal convolucional consiste en cinco bloques convolucionales cuyas salidas son normalizadas por lotes que vienen seguidas de dos capas densas de 4096 neuronas cada una y una última capa densa de 1000 neuronas encargadas de la clasificación. Para el R-CNN se remueve la última capa densa y se sustituye por una capa con el número de neuronas igual a $C + 1$, donde C es el número de categorías, y donde se considera una categoría más, la categoría de fondo. Antes de entrenarse la red, consideramos que las propuestas de regiones son de distintos tamaños, y dado la entrada de la red de 227×227 , entonces cada propuesta de región es deformada a estas dimensiones. Una vez entrenada la red, tenemos una red capaz de clasificar propuestas de regiones en las categorías que se consideran y la categoría de fondo.
4. En esta etapa se remueve la última capa de la CNN entrenada en el paso 3, y se utiliza la segunda

capa de 4096 neuronas para entrenar un SVM lineal para cada categoría, para clasificarlas como positivo o negativo, en este paso los parámetros del CNN no se modifican y permanecen constantes. Sin embargo para entrenar los SVMs es necesario etiquetar cada imagen como positivo o negativo, para cada clase, se consideran positivo únicamente las regiones verdaderas y negativos aquellas regiones con un $IoU < 0.3$ de intersección con las cajas verdaderas de la categoría considerada.

Una vez finalizadas cada una de las etapas anteriores únicamente prosigue una etapa de regresión de cajas delimitadoras para mejorar la localización de las cajas predichas. Y finalmente se realiza una supresión no máxima de las cajas predichas para mostrar las detecciones por el modelo.

Fast R-CNN

Fast R-CNN, fue propuesto por Ross Girshick en 2015, surge como una mejora directa de R-CNN [25]. Uno de los principales problemas de R-CNN es su lentitud, ya que procesa unas 2000 regiones por imagen de forma individual, lo que además implica un alto costo computacional y tres etapas de entrenamiento secuencial. Fast R-CNN resuelve estas limitaciones (véase la Figura 2.12).

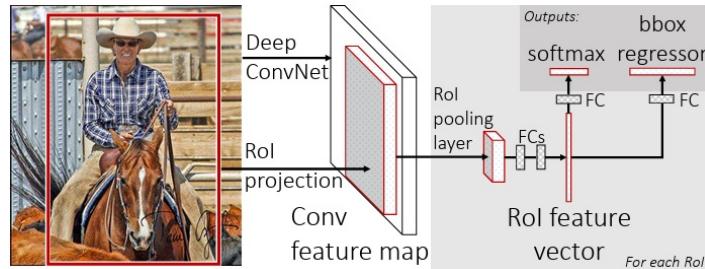


Figura 2.12: Arquitectura de Fast R-CNN. Reduce etapas del modelo R-CNN para lograr un procesamiento más rápido. Figura tomada de [25].

La arquitectura parte de R-CNN y emplea búsqueda selectiva para generar propuestas de regiones. Se seleccionan 128 regiones por imagen: aquellas con superposición $IoU \geq 0.5$ se consideran positivas, las que cumplen $0.1 \leq IoU < 0.5$ se asignan a la clase fondo, y las que tienen $IoU < 0.1$ se descartan. Se busca mantener una proporción de 25 % regiones positivas y 75 % de fondo.

A diferencia de R-CNN, Fast R-CNN procesa toda la imagen en la CNN, y luego redimensiona las coordenadas de las regiones propuestas a la escala de la última capa convolucional. Para resolver la inconsistencia de tamaños de las regiones (al ingresar a capas totalmente conectadas de tamaño fijo

4096), se introduce la capa *RoI Pooling*, que ajusta todas las regiones a un tamaño uniforme (véase la Figura 2.13).

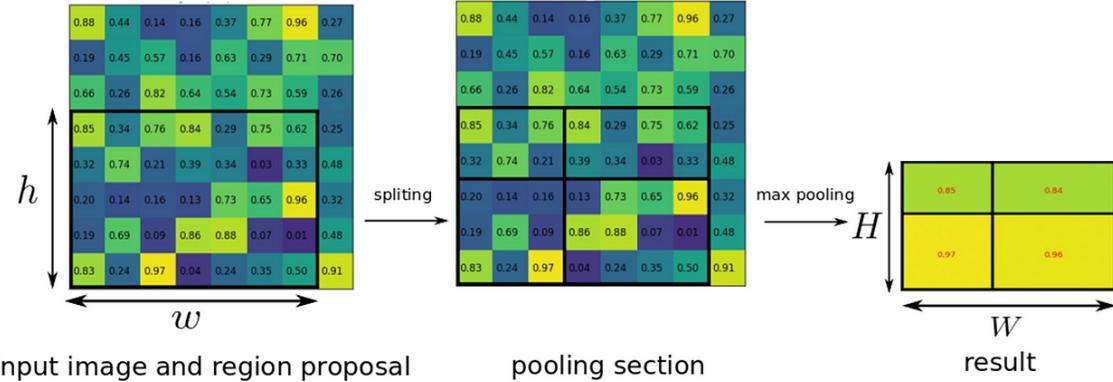


Figura 2.13: Funcionamiento del *RoI Pooling*. El mapa de activación correspondiente a una región propuesta es dividido en secciones pudiendo no ser del mismo tamaño y a cada sección se le aplica *max pooling* para generar una representación de tamaño fijo $H \times W$. Este proceso permite adaptar regiones de distintos tamaños a una dimensión uniforme para ser procesadas por capas densas. Figura tomada de [26].

Después del *RoI Pooling*, las regiones se procesan por capas densas, y su salida se utiliza para dos tareas: (1) una capa de clasificación con $C + 1$ salidas (incluye la clase fondo), con activación *softmax*, y (2) una capa de regresión con $4 \times C$ salidas, correspondiente a los parámetros de las cajas delimitadoras para cada clase.

Ambas tareas se entrena conjuntamente usando la siguiente función de pérdida:

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda \cdot [u \geq 1] \cdot L_{\text{reg}}(t^u, v) \quad (2.21)$$

Donde:

- p : vector de probabilidades predichas por clase.
- u : clase verdadera.
- t^u : caja predicha para la clase u .
- v : caja verdadera.
- L_{cls} y L_{reg} : pérdida de clasificación (entropía cruzada) y pérdida de regresión (Smooth L1).

- $[u \geq 1]$: indica que solo se aplica regresión si la región no es fondo.
- λ : peso para la pérdida de regresión (usualmente se fija en 1).

Fast R-CNN logra mejorar significativamente el tiempo de procesamiento y la precisión en comparación con R-CNN.

Faster R-CNN

Fue publicado por Shaoqing Ren, Kaiming He, Ross B. Girshick y Jian Sun en el año 2015 [27]. Este modelo surgió para incrementar la velocidad en la detección. El modelo Fast R-CNN obtiene sus regiones de propuestas mediante búsqueda selectiva, proceso que le demora la mayor parte del tiempo en la detección de objetos, Faster R-CNN remplaza la búsqueda selectiva por una Red de Propuestas de Regiones o *Region Proposal Network*, *RPN* en inglés, que se encarga de generar las regiones de propuestas y además aprende a generarlas. La arquitectura de la red Faster R-CNN se observa en la Figura 2.14.

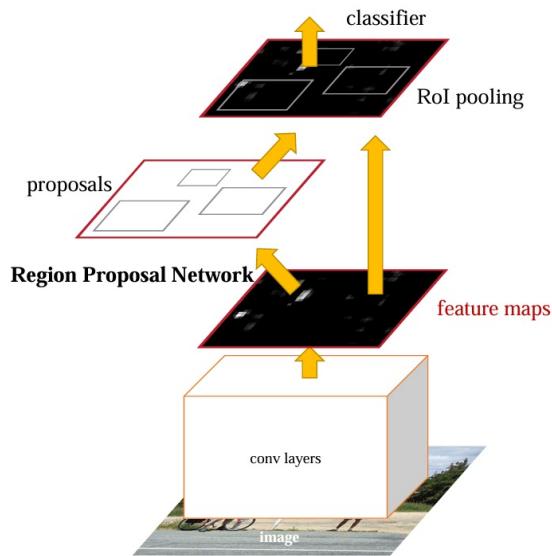


Figura 2.14: Arquitectura de Faster R-CNN. La imagen pasa por una CNN, se aprovecha el último mapa de activación para generar las propuestas de regiones mediante RPN, estas propuestas se extraen del mapa de características y se procesa para su clasificación. Figura tomada de [28].

Detalles de la arquitectura Faster R-CNN:

1. La imagen se procesa en una CNN pudiendo ser ResNet o VGG, estas capas extraen características importantes de la imagen y en el último bloque convolucional genera un mapa de activación.
2. Para generar las regiones de propuestas se considera una red pequeña que tiene como entrada una ventana espacial de $n \times n$, que se desplaza por todo el mapa de activación, los autores utilizan $n = 3$, en cada coordenada central de la ventana deslizante se generan cajas de anclaje o *Anchor Boxes* en inglés, estos son cuadros de distintas dimensiones y relación de aspecto especificados para extraer regiones de propuestas en la imagen, los autores utilizan 3 cajas con 3 relaciones de aspectos para cada una, siendo un total de $K = 9$ cajas de anclaje (véase la Figura 2.15). Aplicando un kernel cada región del mapa de activación $n \times n$ se obtiene una salida de $1 \times 1 \times C$, donde C es el número de canales de la salida para la convolución, al final se obtiene un volumen de $C \times W \times H$, donde cada celda es la representación de las características para cada ventana $n \times n$, después de pasar por una función de activación. Posteriormente, de este volumen se derivan dos capas convolucionales hermanas de 1×1 para la clasificación y la regresión de las cajas. Al final de cada una de estas capas se tienen dos mapas de activación de $2K \times W \times H$ y $4K \times W \times H$ respectivamente. Para cada coordenada en $W \times H$ se tienen dos vectores, uno de $2K$ que indica la clasificación binaria de si hay un objeto o no en cada caja de anclaje y el segundo de $4K$ que nos indica las coordenadas de la caja delimitadora.
3. Una vez que se tienen las propuestas de regiones que se extraen del último mapa de activación de CNN utilizado y se muestran 256 regiones de propuestas, 128 cajas de anclajes positivos, cuadros con una superposición $IoU \geq 0.7$ con las cajas verdaderas y 128 cajas de anclajes negativas, cuadros con una superposición $IoU < 0.3$ con las cajas verdaderas, las cajas con una superposición entre $0.3 \leq IoU < 0.7$ son ignorados. Luego, se ajustan a un mismo tamaño mediante RoI Pooling como en Fast R-CNN.
4. A partir de este punto se continua como un Fast R-CNN (se comparte el mismo CNN), vectorizando estas regiones del mismo tamaño para ser procesadas por una capa *fully connected* y realizando la clasificación y el ajuste de las cajas delimitadoras.
5. Finalmente, se reducen las predicciones con supresión no máxima.

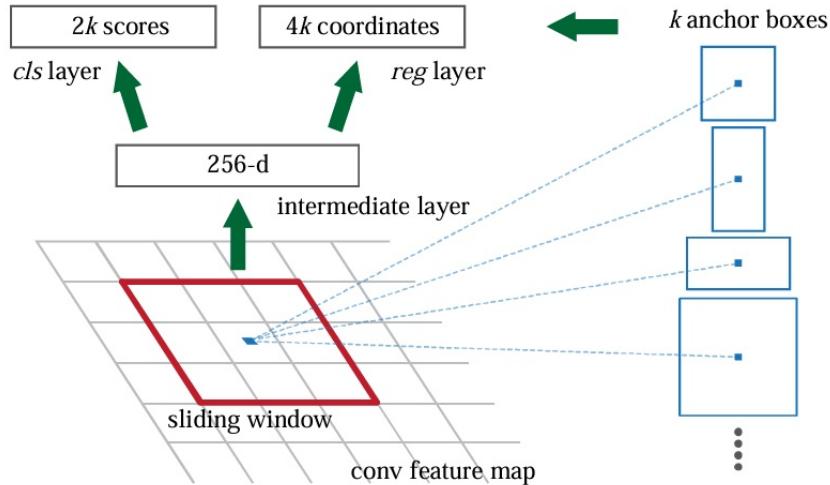


Figura 2.15: Region Proposal Network (RPN). Ejemplo de ventana espacial de $n = 3$ que se desplaza por todo el mapa de activación o de características, en la coordenada central de cada ventana se establecen las cajas de anclajes establecidas. Figura tomada de [28].

Para el entrenamiento del RPN, se utiliza una función de perdida que combina la clasificación y la regresión en una sola función, con esta función de perdida en RPN y la función de perdida utilizada en la parte de Fast R-CNN se modifican los pesos de la CNN compartida.

You Only Look Once (YOLO)

YOLO fue propuesto en 2016 por Joseph Redmon, Santosh Divvala, Ross Girshick y Ali Farhadi, marcando un cambio significativo en la tarea de detección de objetos [29]. A diferencia de los métodos tradicionales, que abordaban la detección en dos etapas (propuesta de regiones y clasificación), YOLO unifica todo el proceso en una sola red neuronal convolucional. Esta red es capaz de predecir simultáneamente múltiples cuadros delimitadores y sus respectivas probabilidades de clase, lo que permite realizar detecciones en tiempo real con alta eficiencia.

El modelo comienza dividiendo la imagen de entrada en una cuadrícula de $S \times S$ celdas. Cada celda es responsable de predecir B cajas delimitadoras y una puntuación de confianza asociada. Esta puntuación refleja la certeza del modelo de que la caja contiene un objeto, y qué tan bien se ajusta la predicción a la realidad. Formalmente, la confianza se define como:

$$Pr(\text{Objeto}) \times \text{IoU}_{\text{pred}}^{\text{verd}}.$$

Cada caja delimitadora realiza cinco predicciones: x, y, w, h y confianza, donde (x, y) representan el centro de la caja relativo a la celda, y w, h son el ancho y alto relativos al tamaño total de la imagen. La confianza se calcula como la intersección sobre la unión (IoU) entre la caja predicha y la caja real. Si no hay objeto en la celda, la confianza debe ser cero. Para que una celda sea responsable de detectar un objeto, el centro de dicho objeto debe ubicarse dentro de la celda.

Además, cada celda predice C probabilidades de clase condicionales $Pr(\text{Clase}_i \mid \text{Objeto})$, que indican la probabilidad de que el objeto pertenezca a la clase i , dado que existe un objeto en esa celda, independientemente del número de cajas B (véase la Figura 2.16).

Durante la fase de prueba, se multiplica la probabilidad condicional por la confianza de cada caja para obtener una puntuación específica por clase, de la siguiente forma:

$$Pr(\text{Clase}_i \mid \text{Objeto}) \times Pr(\text{Objeto}) \times \text{IoU}_{\text{pred}}^{\text{verd}} = Pr(\text{Clase}_i) \times \text{IoU}_{\text{pred}}^{\text{verd}}. \quad (2.22)$$

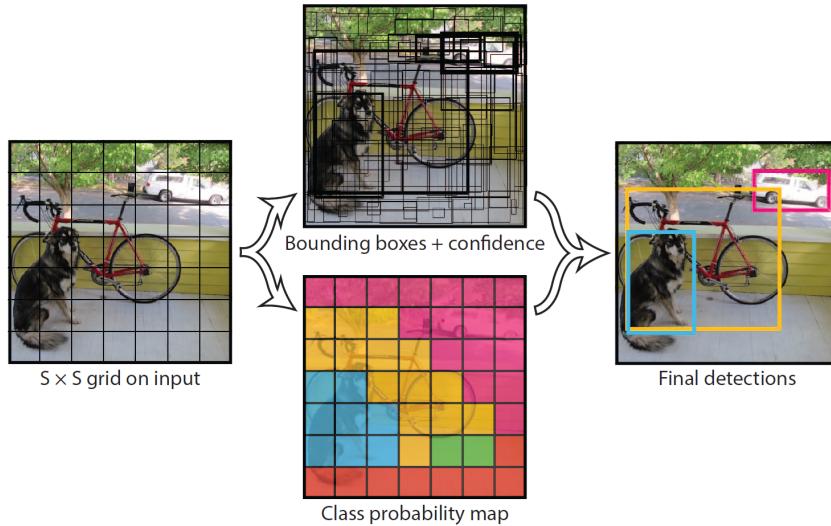


Figura 2.16: Modelo YOLO. La imagen se divide en $S \times S$ celdas. Cada celda predice B cajas delimitadoras y su confianza, así como C probabilidades de clase. Las predicciones se codifican en un tensor de dimensiones $S \times S \times (5B + C)$. Figura tomada de [29].

La red neuronal convolucional utilizada en YOLO está inspirada en el modelo *GoogLeNet* para clasificación de imágenes. La arquitectura consta de 24 capas convolucionales seguidas por 2 capas completamente conectadas. La salida final es un tensor de predicción de dimensión $S \times S \times (5B + C)$.

Durante el entrenamiento la función de pérdida que se optimiza es la siguiente:

$$\begin{aligned}
\mathcal{L} = & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbf{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbf{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbf{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbf{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} \mathbf{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2.
\end{aligned} \tag{2.23}$$

Donde:

- $\mathbf{1}_{ij}^{\text{obj}}$ y $\mathbf{1}_{ij}^{\text{noobj}}$ indican si el predictor j de la celda i es responsable (o no) de un objeto.
- x_i, y_i, w_i, h_i y sus versiones con gorro (\hat{x}_i , etc.) son las coordenadas y dimensiones reales y predichas de la caja delimitadora.
- C_i y \hat{C}_i son las puntuaciones de confianza reales y predichas.
- $p_i(c)$ y $\hat{p}_i(c)$ representan las probabilidades reales y predichas de que la celda i contenga un objeto de la clase c .
- λ_{coord} y λ_{noobj} son hiperparámetros que ponderan la pérdida de localización y de confianza en celdas vacías.

A pesar de que YOLO es un modelo sólido que supera a otros enfoques en tareas de detección de objetos tanto en precisión como en velocidad, representó un punto de partida para el desarrollo de versiones posteriores que abordaron sus principales limitaciones, como la baja eficacia en la detección de objetos pequeños. No obstante, el modelo original tiene la capacidad de aprender representaciones generales de los objetos, lo que le permite generalizar bien en distintos dominios.

YOLOv4

Después de tres versiones de YOLO pùblicados en el que cada versión presenta mejoras en la arquitectura, implementación de nuevas técnicas, y mejoran el desempeño del modelo YOLO, YOLOv4 fue pùblicado el 23 de abril de 2023 por Alexey Bochkovskiy, Chien-Yao Wang y Hong-Yuan Mark Liao [30]. El objetivo de este modelo era tener un modelo de detección de objetos eficiente y potente en el que cualquier persona con una GPU convencional pudiera entrenar y tener resultados en tiempo real y de calidad.

Al igual que muchos detectores modernos, el modelo YOLOv4 cuenta con una base *backbone* que es una red neuronal convolucional encargada de extraer las características de la imagen para la detección de objetos, seguido de un *neck* que se encarga de refinar las características extraídas y combinar mapas de características provenientes de distintas etapas del backbone, el propósito del neck es mejorar la información semántica y de localización para distintas escalas. Finalmente YOLOv4 está compuesto por un *head* que toma la información resultante del neck y se encarga de realizar la detección de los objetos y su clasificación.

Para seleccionar la arquitectura de YOLOv4 los autores consideraron encontrar un balance óptimo entre la resolución de entrada de la red, el número de capas convolucionales, el número de filtros y el número de salida de capa. En base a los criterios a considerar, la arquitectura de YOLOv4 se estructura de la siguiente forma:

- **Backbone:** *CSPDarknet53*. Es una arquitectura de red neuronal convolucional basada en Darknet-53 (usado en YOLOv3), que incorpora la estrategia *Cross Stage Partial* (CSP). Esta técnica divide el flujo de características en dos partes: una se procesa mediante convoluciones y la otra se mantiene intacta para fusionarse posteriormente. Esto mejora la eficiencia computacional sin comprometer la precisión.
- **Módulo adicional:** *Spatial Pyramid Pooling* (SPP). Es un módulo que amplía el campo receptor aplicando *max pooling* con distintos tamaños de kernel al último mapa de características del *backbone*, concatenando luego los resultados con el mapa original. Esto permite captar información a múltiples escalas sin perder resolución.

- **Neck:** *Path Aggregation Network* (PAN). Es una arquitectura que complementa a la *Feature Pyramid Network* (FPN), la cual combina mapas de características de alto nivel (semánticamente ricos) con los de bajo nivel (más detallados) en una estructura de arriba hacia abajo. PAN añade una vía de flujo inversa (de abajo hacia arriba), enriqueciendo aún más la información multiescala y mejorando la detección de objetos de distintos tamaños.
 - **Head:** Utiliza la misma estructura de predicción que YOLOv3, basada en cajas de anclaje y predicción a tres escalas.

La arquitectura con mayor detalle y visualización se puede observar en la figura 2.17.

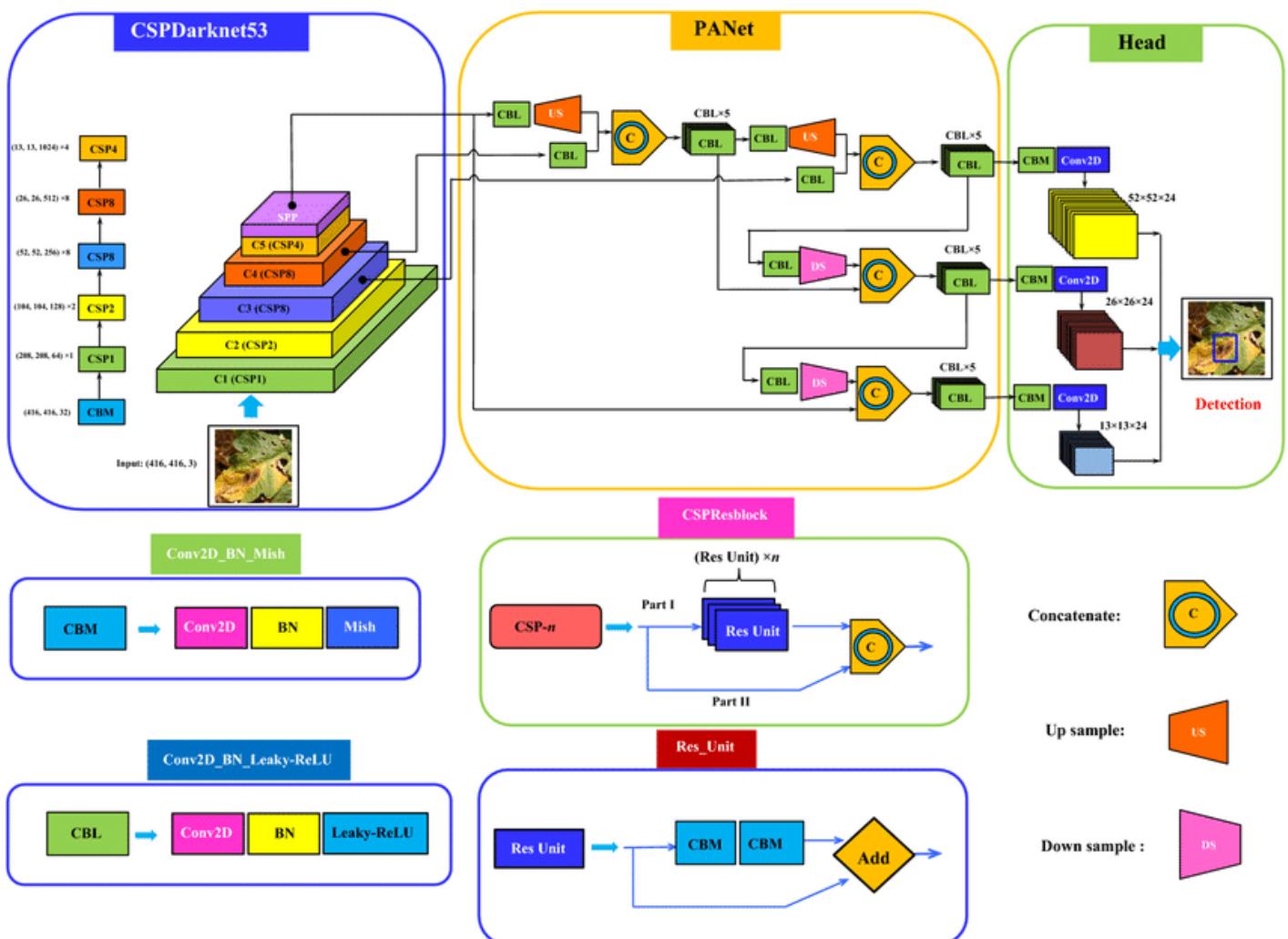


Figura 2.17: Arquitectura del modelo YOLOv4. Figura tomada de [31].

También los autores integraron algunos métodos extras que ayudan el proceso de entrenamiento y de inferencia, a los métodos que modifican la estrategia de entrenamiento o incrementan únicamente el costo durante el entrenamiento, los llamaron “*bag of freebie*” (bolsa de beneficios gratuitos). A los métodos complementarios (plugins) y métodos de postprocesamiento que solo aumentan ligeramente el costo de inferencia pero que pueden mejorar significativamente la precisión de la detección de objetos les llamaron “*bag of specials*” (bolsa de mejoras especiales). Tras realizar diferentes experimentos finalmente se incluyeron los siguientes en el modelo de YOLOv4.

Bag of Freebies	
Backbone	Detector
CutMix	CIoU-loss
Mosaic data augmentation	CmBN
DropBlock regularization	DropBlock regularization
Class label smoothing	Mosaic data augmentation
—	Self-Adversarial Training
—	Eliminate grid sensitivity
—	Multiple anchors per ground truth
—	Cosine annealing scheduler
—	Optimal hyperparameters
—	Random training shapes

Bag of Specials	
3 Backbone	Detector
Mish activation	Mish activation
Cross-stage partial connections (CSP)	SPP-block
Multi-input weighted residual connections (MiWRC)	SAM-block
—	PAN path-aggregation block
—	DIoU-NMS

Tabla 2.2: Técnicas utilizadas como Bag of Freebies y Bag of Specials en YOLOv4

Bag of Freebies

CutMix combina imágenes cortando una región de una imagen y superponiéndolo en otra, también mezcla etiquetas, esto ayuda a aumentar la robustez. **Mosaic data augmentation** une 4 imágenes en una sola imagen de entrenamiento. Esto ayuda a aumentar la diversidad en las imágenes mejorando la

detección. **DropBlock regularization** es una técnica de regularización que desactiva bloques enteros del mapa de características en vez de hacerlo de forma distribuida, este método resulta ser eficiente en el contexto de redes neuronales convolucionales. **Class label smoothing** evita que el modelo se vuelva “confiado” asignando una probabilidad ligeramente menor a uno al vector one hot correspondiente a la categoría correcta y una probabilidad muy pequeña en donde el vector es cero. **CIoU-loss** es una función de pérdida para boxes que mejora la penalización de las cajas considerando la distancia entre ellas, la superposición y el aspecto de las cajas predichas reales. **CmBN (Cross mini-Batch Normalization)** es una variante del BatchNorm donde se comparten estadísticos entre los distintos mini batches haciéndolo más estable para lotes pequeños. **Self-Adversarial Training (SAT)**. El modelo altera la imagen para engañarse a sí mismo y luego la entrena con esa imagen manipulada, mejorando su resistencia a ataques adversarios. **Eliminate grid sensitivity** cambia el alcance de las predicciones de las coordenadas para ir más allá de las limitaciones fijas de los recuadros del grid, resulta ser más estable y eficaz. **Multiple anchors per ground truth** permite que distintas cajas de anclaje se asigne a un mismo objetos, esto ayuda a mejorar la detección. **Cosine annealing scheduler** es un plan de reducción de tasa de apredizaje con forma de coseno que ayuda a una convergencia más estable y fina. **Optimal hyperparameters** usa los hiperparámetros óptimos al conjunto de datos tras una búsqueda exahustiva en el espacio de hiperparámetros mediante algoritmos genéticos. **Random training shapes** permite entrenar la red con tamaños aleatorios de la entrada.

Bag of Freebies

Mish activation es una función de activación suave y no lienal que mejora la convergencia frente otras funciones. **Cross-stage partial connections (CSP)** ya fue explicado previamente en la arquitectura. **Multi-input weighted residual connections (MiWRC)** combina múltiples entradas usando conexiones residuales ponderadas. **SAM-block (Spatial Attention Module)** es un modulo de atención que resalta regiones espaciales de la imagen de mayor importancia para la detección de los objetos de interés. **DIoU-NMS** es un método de supresión de no-máximos (NMS) que, además de la superposición (IoU), considera la distancia entre los centros de las cajas para mantener las más relevantes.

YOLOv4 resultó ser un modelo consolidado con una arquitectura práctica y de alto rendimiento, siendo más accesible para el uso convencional al público. Este modeló mostró que una buena combinación de métodos pueden dar resultados solidos y escalables.

YOLOv7

YOLOv7 fue publicado el 6 de junio de 2022 por Chien-Yao Wang¹, Alexey Bochkovskiy y Hong-Yuan Mark Liao¹ [32]. YOLOv7 sigue usando una estructura de backbone, neck y head. En el backbone utilizan Extended Efficient Layer Aggregation Network (E-ELAN en inglés), es una mejora al backbone ELAN del YOLOv6. E-ELAN resulta ser un backbone estable para un gran número de bloques computacionales apilados y es independiente de la longitud de la ruta del gradiente. Su estrategia consiste en convoluciones agrupadas (group convolutions en inglés) para expandir los canales y la cardinalidad de los bloques computacionales (véase la figura 2.18).

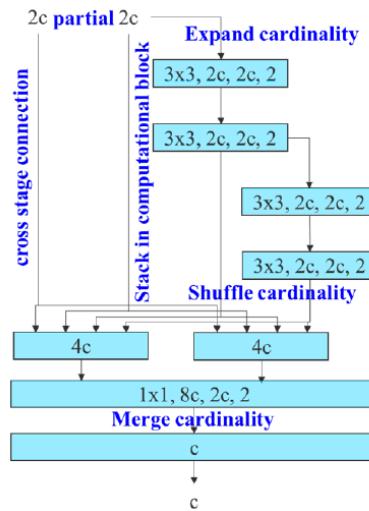


Figura 2.18: Arquitectura del bloque E-ELAN. El mapa de características con c canales pasa por dos procedimientos, uno pasa por procesamientos convolucionales y el otro permanece intacto. En la rama de Expand cardinality (el input se reparte entre varios caminos que aprenden cosas distintas), se crean multiples grupos o bloques convolucionales y todas se procesan en paralelo. Una vez procesados cada grupo pasan a Shuffle cardinality donde los canales son reorganizados o barajeados aleatoriamente por grupo para que en el Merge cardinality el output de todos los grupos se concatenen. Gracias a esto el mapa de características es más rico en información listo para pasar al siguiente bloque del backbone o al neck. Figura tomada de [32].

Para el neck YOLOv7 usa SPPCSPC que es un Spatial Pyramid Pooling (SPP) que amplía el campo receptivo de la red aplicando max-pooling con diferentes tamaños de kernel para capturar contexto a

múltiples escalas, sin afectar el tamaño de salida. Posteriormente le sigue un CSPC (CSP-Convolution) que es una variante de CSP, que involucra una estructura de tipo ELAN. Finalmente el head está compuesto por capas convolucionales (véase figura 2.19).

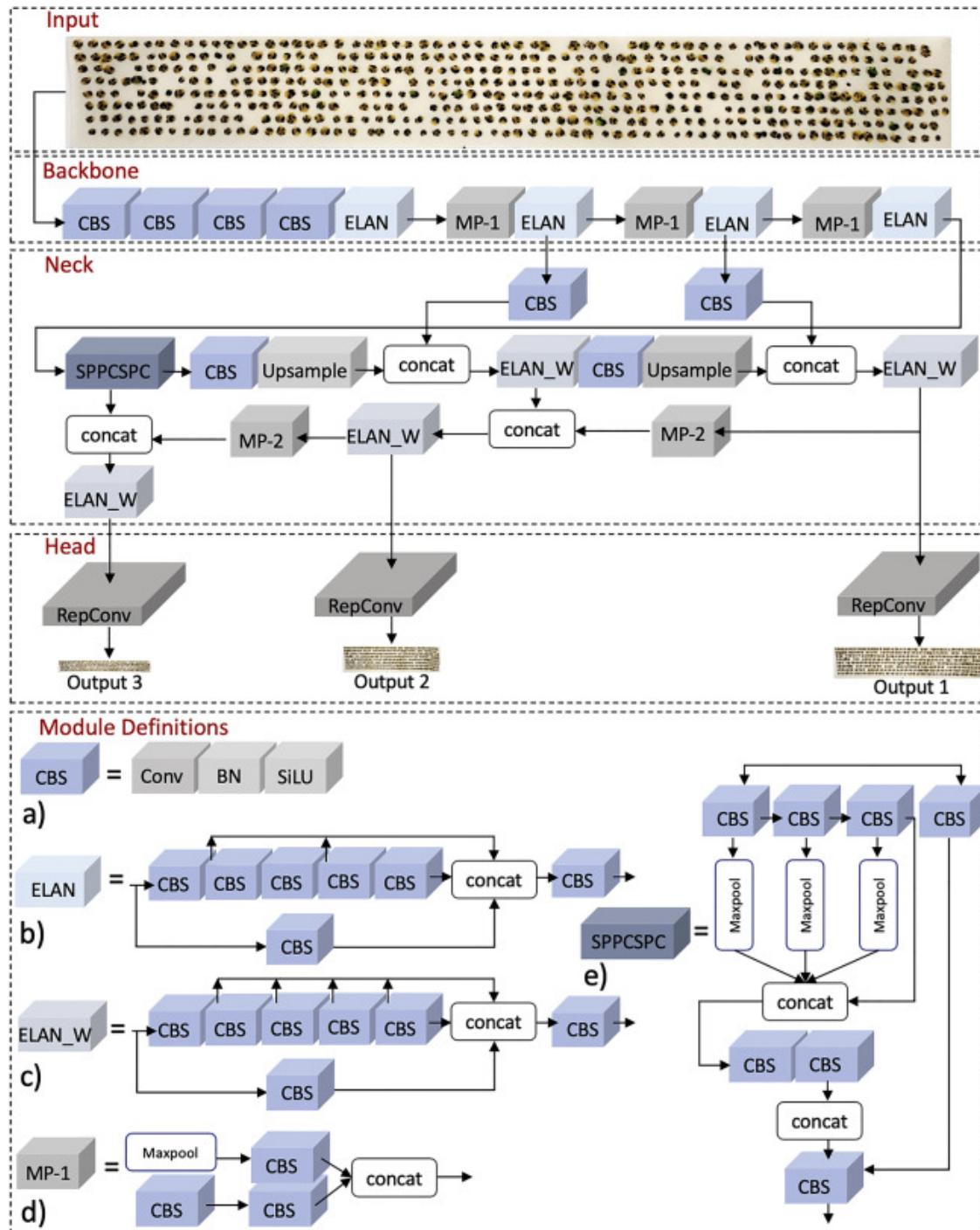


Figura 2.19: Arquitectura general de YOLOv7. Figura tomada de [33].

Bolsa de regalos entrenables (Trainable Bag-of-Freebies)

Las técnicas que implementaron los autores para mejorar el entrenamiento del modelo sin comprometer el costo de la inferencia fueron los siguientes:

- **Convolución reparametrizada planificada (Planned Reparameterized Convolution):** RepConv convina capas convolucionales de 3x3, 1x1 y conexión identidad. Los autores realizan una modificación en este bloque RepConvN donde eliminan la conexión identidad del bloque original.
- **Supervisión profunda y asignación de etiquetas (Deep Supervision y Label Assignment):** La supervisión profunda consiste en utilizar cabezales auxiliares para ayudar en el aprendizaje de la red. Los autores llaman “cabeza principal” (lead head) a la cabeza responsable de la salida final y la cabeza asistida para mejorar la predicción “cabeza auxiliar” (auxiliary head), véase la figura 2.20.

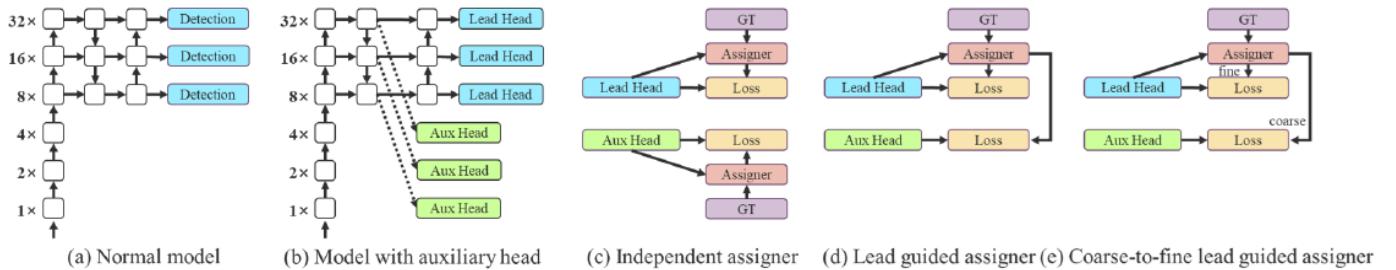


Figura 2.20: Asignador de etiquetas. En (a) se muestra el modelo sin cabezales auxiliares y en (b) con cabezales auxiliares, sin embargo al uso de cabezales auxiliares, se habían estado tratando por separado para ejecutar la asignación de etiquetas como se muestra en (c). En YOLOv7 se propone un nuevo método de asignación de etiquetas en el que se utiliza la predicción de la cabeza principal para guiar a la auxiliar (d), pero para tener una mejor representación de la distribución se generan etiquetas blandas (soft labels) en la cabeza principal, especialmente de dos tipos, la etiqueta fina (fine label) es la misma que la etiqueta blanda generada por la cabeza principal, la etiqueta gruesa se genera permitiendo que más celdas (grids) sean tratadas como objetivos positivos relajando las restricciones del proceso de asignación de muestras positivas (e). Figura tomada de [32].

- **Normalización por lotes (batch normalization):** Integra la media y varianza de la normalización por lotes en el sesgo y pesos de la capa de convolución durante la etapa de inferencia.
- **Conocimiento implícito en YOLOR combinado con mapa de características por suma y multiplicación:** Consiste en una técnica implementada en YOLOR donde se integran presen-

taciones implícitas con los mapas de características explícitos de la red, utilizando operaciones de suma y multiplicación para enriquecer la información aprendida.

- **EMA (Exponential Moving Average) o Media Móvil Exponencial:** Es una técnica utilizada para suavizar los parámetros del modelo y obtener los mejores resultados en la inferencia.

Escalado y reparametrización

La reparametrización es la simplificación de distintos módulos computacionales en uno solo en la etapa de inferencia. Los autores mencionan que han desarrollado un nuevo módulo de reparametrización y han implementado estrategias de aplicación para distintas arquitecturas.

Por otro lado el escalamiento es una forma de ampliar o reducir un modelo ya diseñado para poder ser implementado en distintos equipos de cómputo de acuerdo a los intereses de presición y tiempo de inferencia, para esto se relizan variaciones en los componentes de la red como es el número de canales, capas, etapas y resolución de entrada. Cuando se trata de escalamiento en modelos de concatenación, es necesario tratar el escalamiento en conjunto considerando la profundidad (agregar más bloques) y el ancho (número de canales). Suponiendo que se realiza un escalado hacia arriba, incrementando el modelo, al aumentar la profundidad en el modelo de concatenación, en automático se incrementa el número de canales, por lo que se debe aumentar el ancho del modelo, de lo contrario las siguientes capas reciben más datos de lo esperado y puede perjudicar el entrenamiento o romper la arquitectura, por esta razón no se pueden modificar estas características de los modelos por separado en los modelos de concatenación, para esto los autores proponen un escalado compuesto (compound scaling), donde prestan atención al número de canales después de modificar la profundidad para tener un ajuste equilibrado entre los canales (véase la figura 2.21).

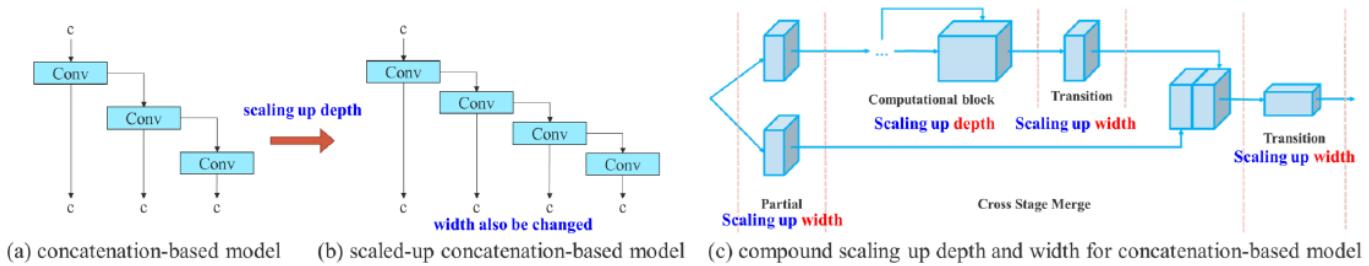


Figura 2.21: Escalamiento de modelos basados en concatenación. En(a) se muestra una parte de modelo basdo en concatenación, al incrementar la profundidad y se agrega una capa convolucional se incrementan las capas del mapa de características de $3c$ a $4c$ como se observa en (b), por tanto los autores presentan el nuevo escalamiento considerando la modificación de los canales (c). Figura tomada de [32].

YOLOv8

En enero de 2023 fue lanzado YOLOv8, esta versión fue lanzada nuevamente por la empresa Ultralytics, el mismo equipo encargado de desarrollar YOLOv5. Esta nueva versión trajo nuevas mejoras en la arquitectura y en la implementación de los modelos mejorando la experiencia de usuario, sin embargo no existe un artículo oficial en el que se detallen características de la arquitectura y mejoras en el modelo, en cambio, Ultralytics ha mantenido un repositorio GitHub actualizado donde se encuentra todo el código disponible del modelo, permitiendo al publico y a la comunidad documentar la arquitectura y ver sus mejoras [34].

YOLOv8 se basa en la arquitectura de YOLOv5 y mantiene la estructura de backbone, neck y head.

- **Backbone:** YOLOv8 mantiene una arquitectura similar al de YOLOv5 basado en CSPDarknet que captura características de bajo con información semántica de alto nivel.
- **Neck:** Utiliza SPPF (por sus siglas en inglés, Spatial Pyramid Pooling - Fast) es una versión optimizada del módulo SPP (Spatial Pyramid Pooling) que se utiliza en YOLOv5, cumple el mismo desempeño que SPP pero mejorando la velocidad y eficiencia con menos operaciones. También se incorpora un módulo PAN que ayudan e mejorar la información multiescala del backbone, es crucial para detectar diferentes objetos con distintas proporciones de tamaño. con estos cambios en el neck ofrecen un mejor uso de memoria y eficiencia computacional.
- **Head:** El head se encarga de generar las predicciones, los cuadros delimitadores y las etiquetas

de las clases. YOLOv8 incorpora una metodología sin uso de anclas predeterminadas, alejándose de las versiones anteriores de YOLO en el que las predicciones se basaban en cajas de anclajes ya definidas. Al integrar esta nueva metodología, se simplifica el costo computacional, reduce el número de hiperparámetros y mejora la adaptabilidad del modelo a objetos con diferentes relaciones de aspecto y escalas.

Algunos cambios de la arquitectura de YOLOv8 con respecto a YOLOv5 son los siguientes puntos:

1. Reemplaza el módulo $C3$ con el módulo $C2f$.
2. Reemplaza la primera convolución de $6x6$ con una convolución $3x3$ en el Backbone.
3. Elimina dos capas convolucionales (No.10 y No.14).
4. Reemplaza la primera capa convolucional con kernel $1x1$ con una convolución $3x3$ en el Bottleneck.
5. Utiliza la cabeza desacoplada.

El rendimiento de YOLOv8 no solo se debe a los cambios en la arquitectura, sino también en las metodologías implementadas en el entrenamiento:

- **Aumento de datos:** Implementan CutMix y Mosaic para el aumento y generalización del modelo ante los datos.
- **Función de pérdida focal:** Se utiliza una función de pérdida focal para dar mayor importancia a las categorías más difíciles de clasificar, ayuda a mitigar el desequilibrio en las clases.
- **Cálculo de perdidas:** Se utilizan las siguientes funciones de pérdida para mejorar la detección. **CIoU + DFL y BCL**, CIoU Loss es una mejora de la pérdida IoU tradicional que se usa para ajustar las cajas delimitadoras; DFL (Distribution Focal Loss) es una técnica que ayuda en la regresión de las coordenadas de las cajas delimitadoras, se utiliza junto con CIoU para una mayor presición; BCL (Binary Cross-Entropy Loss), es una función de pérdida binaria que puede ser utilizada para detección de objetos multiclas de manera independiente.

- **Entrenamiento de precisión mixto:** Es una técnica que se utiliza para combinar distintas presiciones numéricas durante el entrenamiento para reducir el uso de memoria y acelerar el entrenamiento sin afectar la presición del modelo.

Para más detalles de la arquitectura véase la figura 2.22.

YOLOv8

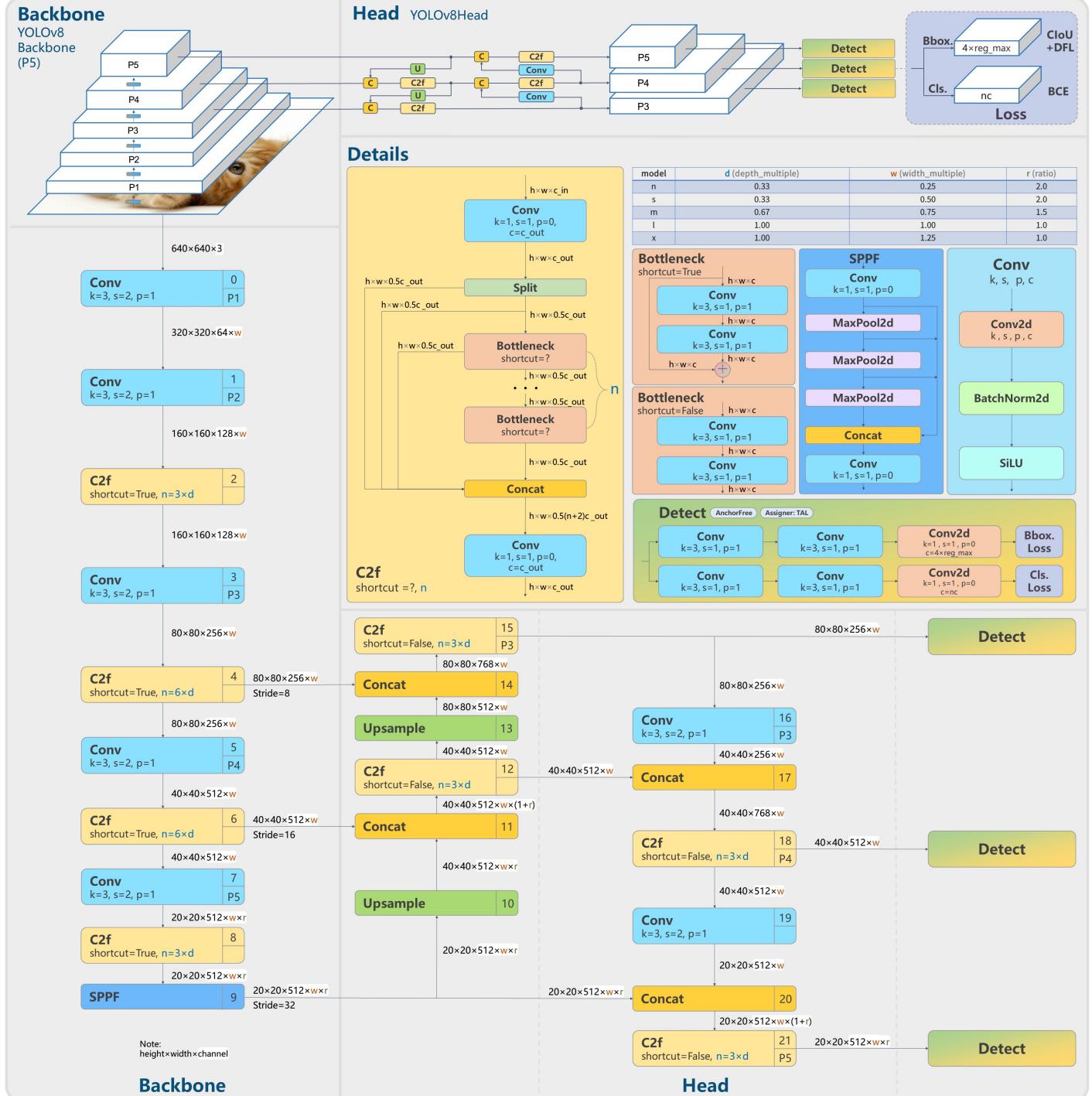


Figura 2.22: Arquitectura de YOLOv8. Figura tomada de [35].

YOLOv12

YOLOv12 es el modelo más reciente de la familia YOLO publicado en febrero de 2025 por Yunjie Tian, Qixiang Ye y David Doermann [36]. YOLOv12 es un modelo importante porque muestra un nuevo paradigma para la detección de objetos basados en la atención. En la actualidad es el modelo con mayor presión y velocidad en comparación con las versiones anteriores de YOLO. Sigue teniendo una estructura de backbone, neck y head que se detallan en las siguientes características:

- **Backbone: Red de Agregación de Capas Residuales Eficientes (Residual Efficient Layer Aggregation Network o R-ELAN).** YOLOv12 incorpora bloques de R-ELAN estratégicamente con capas convolucionales en el backbone, este bloque introducido permite una conectividad residual con un factor de escala predeterminado en 0.01, que ayuda al entrenamiento y la eficiencia del gradiente, este nuevo bloque intenta solucionar los problemas con los que ELAN cuenta, una módulo que introduce inestabilidad y carece de conexión residual (véase la figura 2.23). También incorpora bloques un nuevo tipo de bloques convolucionales en el que priorizan las operaciones ligeras y una mayor paralelización. Además usan la técnica de 7×7 convoluciones separables para reducir la carga computacional. Esta técnica reemplaza las convoluciones mediante kernels manteniendo la percepción espacial con menos parámetros.

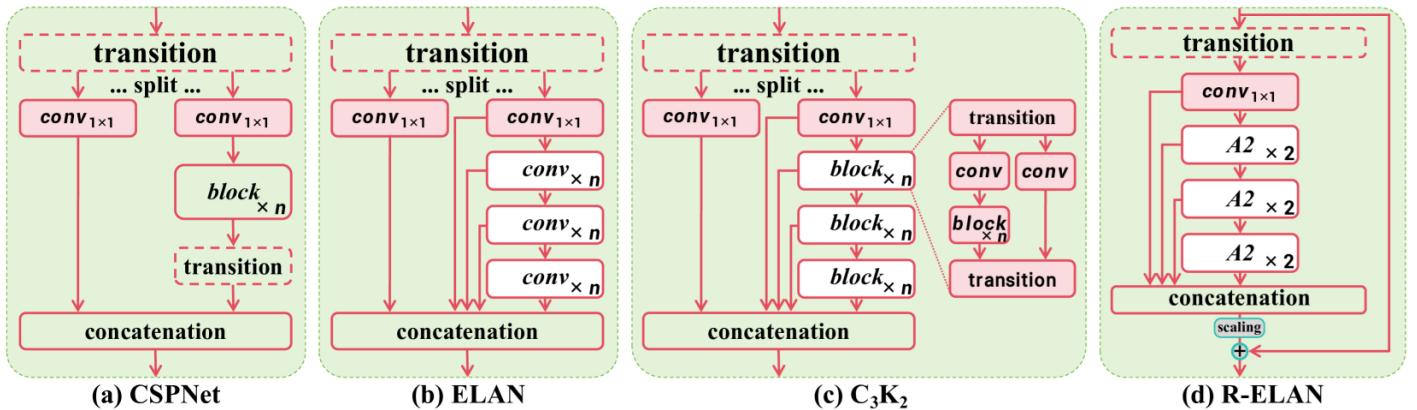


Figura 2.23: Arquitectura de bloque R-ELAN frente a modulos populares.(a) CSPNet, (b) ELAN, (c) C_3K_2 , (d) R-ELAN (propuesto en YOLOv12). Figura tomada de [36].

- **Neck: Atención de área (Area Attention en inglés):** La innovación clave de YOLOv12 es la atención de área (Area Attention en inglés) acelerado con FlashAttention. La atención por área es

un modulo que divide cada mapa de características en partes (en YOLOv12 típicamente en 4) y aplica atención local en cada área, esta estrategia ayuda a reducir la complejidad computacional pero mantiene un alto campo receptivo. Este módulo lo aceleran con FlashAttention permitiendo minimizar el uso de memoria conservando la velocidad. FlashAttention es un algoritmo de atención escrito en CUDA que permite optimizar los mecanismos de atención aprovechando el hardware reduciendo la memoria y manteniendo la velocidad.



Figura 2.24: Comparación de los mecanismos representativos de atención local con los propuestos en YOLOv12. Aunque las mecanismos de atención suelen ser muy eficaces para capturar dependencias globales resulta ser más lenta que las redes neuronales convolucionales, también el proceso de cálculo de la atención, los patrones de acceso a la memoria son menos eficientes en comparación con las CNN. Para poder solucionar estos dos factores se realiza un enfoque de atención local para reducir el costo computacional. Existen distintas maneras de realizar la partición del mapa de características, el empleado en YOLOv12 consiste en dividir l áreas verticales u horizontales (valor predeterminado de 4). Esto evita operaciones complejas y garantiza un alto campo receptivo lo que permite una alta eficiencia. Figura tomada de [36]

- **Head:** YOLOv12 es similar al de YOLOv11 que incorpora multi-escala y con predicciones separadas (como decoupled head), sin anclas explícitas. Transforma los mapas de características que se obtienen del neck en coordenadas de cuadro delimitador y puntuaciones de clasificación.

Para más detalles de la arquitectura véase la figura 2.25.

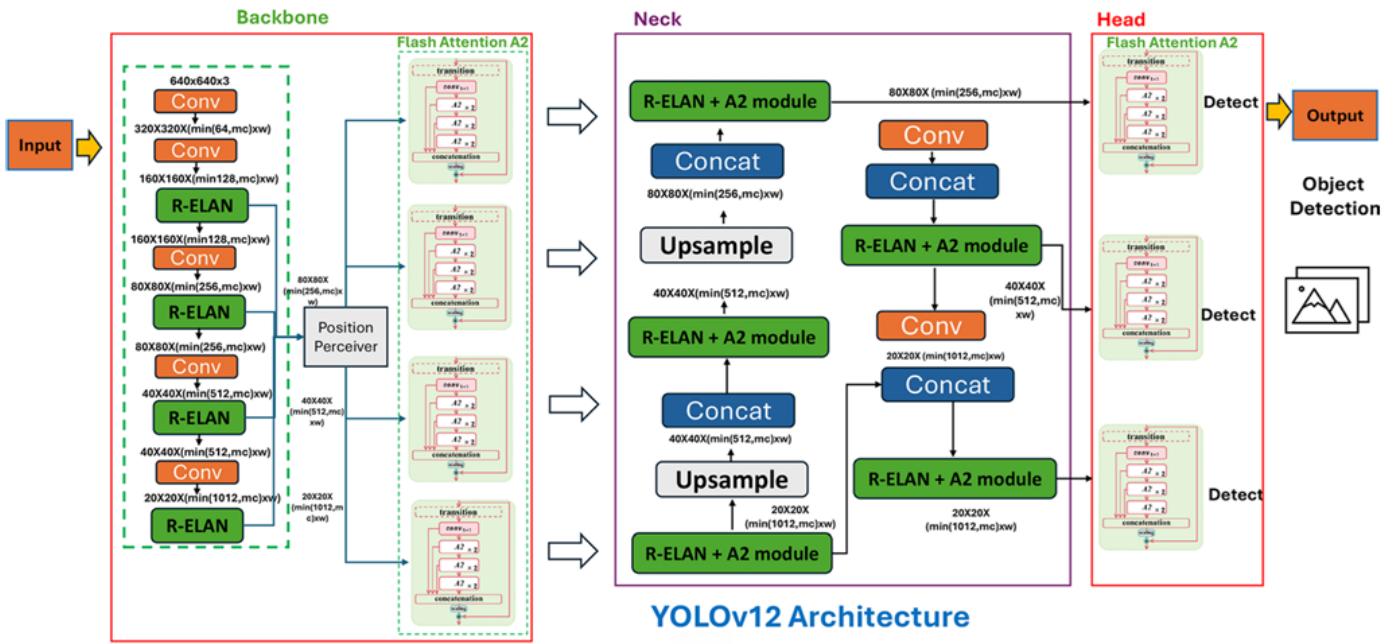


Figura 2.25: Arquitectura de YOLOv12. Figura tomada de [37].

Evolución y aportes de la familia YOLO

Versión	Año	Tareas	Contribuciones	Framework
YOLO [1]	2015	Detección de objetos, clasificación básica	Detector de objetos de una sola etapa	Darknet
YOLOv2 [6]	2016	Detección de objetos, clasificación mejorada	Entrenamiento multiescala, agrupamiento dimensional	Darknet
YOLOv3 [13]	2018	Detección de objetos, multiescala	Bloque SPP, red troncal Darknet-53	Darknet
YOLOv4 [12]	2020	Detección de objetos, seguimiento básico	Activación de Mish, columna vertebral CSPDarknet-53	Darknet
YOLOv5 [14]	2020	Detección de objetos, segmentación de instancias	Detección sin ancla, activación SWISH, PANet	PyTorch
YOLOv6 [15]	2022	Detección de objetos, segmentación de instancias	Autoatención, OD sin anclas	PyTorch
YOLOv7 [16]	2022	Detección, seguimiento y segmentación de objetos	Transformadores, reparametrización E-ELAN	PyTorch
YOLOv8 [17]	2023	Detección de objetos, instancias y segmentación panóptica	GAN, detección sin anclaje	PyTorch
YOLOv9 [18]	2024	Detección de objetos, segmentación de instancias	IGP y GELAN	PyTorch
YOLOv10 [19]	2024	Detección de objetos	Asignaciones duales consistentes para una formación sin NMS	PyTorch
YOLOv11 [10]	2024	Detección de objetos, segmentación de instancias	Capacidades ampliadas, eficiencia mejorada	PyTorch
YOLOv12 [11]	2025	Detección de objetos, segmentación de instancias	Diseño avanzado centrado en la atención (FlashAttention, R-ELAN), convoluciones separables 7×7 , eficiencia computacional mejorada	PyTorch

Tabla 2.3: Evolución de la familia YOLO.

2.4. Trabajos relacionados

A través de algoritmos de aprendizaje profundo, como redes neuronales convolucionales y modelos de segmentación semántica, es posible analizar imágenes aéreas y detectar cuerpos de agua estancada u otros entornos favorables para la reproducción de mosquitos.

Estas técnicas pueden integrarse con sistemas de teledetección y sistemas de información geográfica (GIS), permitiendo generar mapas de riesgo y optimizar las estrategias de control vectorial.

Diversos estudios han implementado estas técnicas con resultados prometedores. En [38], se propusieron enfoques para detectar criaderos de mosquitos a partir de imágenes aéreas obtenidas por drones. Utilizaron el modelo de red neuronal convolucional YOLOv3 para identificar tanques de agua, logrando una precisión media (mAP) de 0.9650. También emplearon la técnica de *Bag of Visual Words* (BoVW) junto con máquinas de vectores de soporte (SVM) para detectar escenarios de riesgo.

En Shanghái, investigadores desarrollaron un modelo YOLOv7 para identificar posibles hábitats de mosquitos, alcanzando puntuaciones F1 y mAP superiores a 0.99 [39].

En Nueva York (EE.UU.), se emplearon drones con imágenes de alta resolución y redes neuronales convolucionales para monitorear hábitats de *Ae. albopictus* en zonas suburbanas, demostrando una mejora significativa respecto a los métodos terrestres de vigilancia vectorial [40].

En Río de Janeiro (Brasil), se utilizó la arquitectura ResNet-50-FPN como base para detectar criaderos de *Aedes aegypti*. Además, se publicó una base de datos de imágenes aéreas tomadas por drones. Se obtuvieron puntuaciones F1 de 0.65 y 0.77 para la detección de llantas y tinacos, respectivamente, lo cual demuestra la capacidad del sistema para localizar objetos que representan criaderos potenciales [41].