Comparación de Capas Convolucionales Normales y Separables Produndas: Impacto en Parámetros, Tiempo de Entrenamiento y Rendimiento.

Moreno Santiago José Miguel **Primavera, 2025**

RESUMEN: Se analiza la comparación entre capas convolucionales normales y separables en redes neuronales. El objetivo es estudiar su impacto en el número de parámetros, tiempo de entrenamiento y resultados obtenidos. Se propone experimentar con arquitecturas que incluyan convoluciones separables modificadas y evaluar su desempeño frente a convoluciones normales y las separables estándar.
Palabras clave: Capas convolucionales, Convoluciones separables, Redes neuronales, Parámetros, Entrenamiento.

ABSTRACT: A comparison between standard and separable convolutional layers in neural networks is analyzed. The goal is to study their impact on parameter count, training time, and performance. Experiments with architectures featuring modified separable convolutions are proposed to evaluate their performance compared to standard and separable convolutions.

Keywords: Convolutional layers, Separable convolutions, Neural networks, Parameters, Training.

1 Introducción

Las redes neuronales convolucionales (CNNs) son un tipo de arquitectura fundamental en el campo de la visión por computadora, especialmente para tareas como la clasificación de imágenes, detección de objetos y segmentación. Su nombre proviene de las capas convolucionales, que son responsables de identificar características clave en las imágenes, como bordes, formas y patrones complejos. Estas redes son muy eficaces debido a su capacidad para aprender características jerárquicas y espaciales de las imágenes de manera automática durante el proceso de entrenamiento.

Una imagen en color, como las que se encuentran en los formatos RGB, consta de tres canales de color: Rojo (R), Verde (G) y Azul (B). Cada canal contiene la información de intensidad de ese color en cada píxel de la imagen. Estos tres canales se combinan para formar la imagen completa que vemos.

Figure 1: Ejemplo de una imagen con sus componentes RGB

En una red convolucional, los filtros (o kernels) se aplican directamente sobre la imagen completa, considerando simultáneamente la información de los tres canales. Esto permite que los filtros aprendan características que dependen de la interacción entre los canales, lo que resulta crucial para capturar patrones complejos presentes en las imágenes.

El proceso de convolución es fundamental para las CNNs. Consiste en aplicar un filtro (una pequeña matriz de valores) sobre la imagen, realizando un producto punto entre el filtro y las regiones de la imagen. Este proceso genera un mapa

de características, donde se destacan los patrones detectados en la imagen original, lo que facilita tareas de clasificación o predicción. Es en la fase de entrenamiento de una CNN, la CNN "aprende" los valores óptimos para cada filtro o kernel, esto valores óptimos son los que permiten extraer atributos significativos (texturas, bordes, formas), y estos atributos conforman el mapa de características de la imagen o patrones, patrones que luego se utilizaran para clasificar los objetos en otras imágenes

A medida que se agregan más capas convolucionales, la red es capaz de identificar patrones más complejos y abstractos, desde bordes simples en las primeras capas hasta representaciones completas de objetos en capas más profundas.

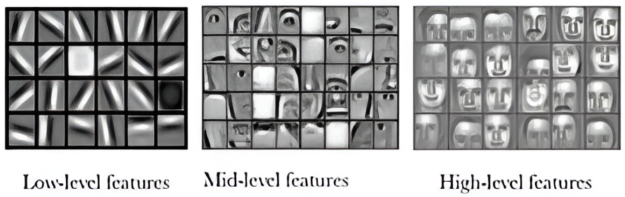

Figure 2: Cada capa aprende niveles de abstracción progresivos para comprender eficientemente conceptos visuales complejos.

1.1 Proceso de Convolución

Es importante entender las diferencias entre la convolución matemática estricta y el proceso que usamos comúnmente en las redes neuronales, conocido como *correlación cruzada*.

1.1.1 La Convolución Matemática: Rotación del Filtro

En su definición matemática más estricta, la **convolución** se refiere a un proceso que involucra *rotar el filtro 180 grados* antes de aplicarlo sobre la imagen. Esto es parte de la formulación matemática original de la convolución, que se utiliza en muchos campos del procesamiento de señales.

Para ilustrarlo, supón que tienes un filtro de tamaño 3x3:

$$F = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Y una pequeña sección de la imagen I:

$$I = \begin{bmatrix} 2 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

En la convolución matemática, primero rotaríamos el filtro F 180 grados:

$$F_{\text{rotado}} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Luego, este filtro rotado se aplicaría a la imagen, realizando un producto punto entre el filtro y las regiones de la imagen para producir un valor en el mapa de características.

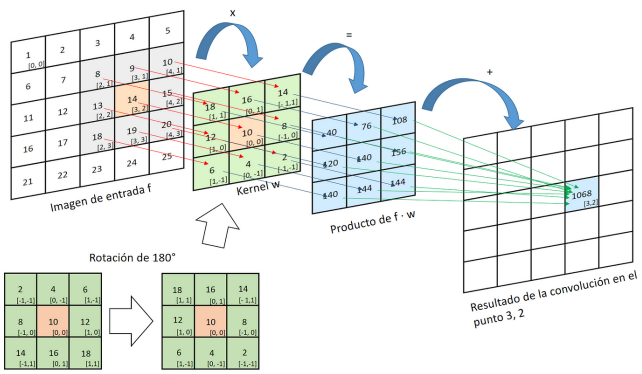

Figure 3: Proceso de convolución, en donde primero se multiplican el filtro w rotado con su región correspondiente de la imagen.

1.1.2 La Correlación Cruzada: Sin Rotación del Filtro

Sin embargo, en el contexto de las redes neuronales, usamos un proceso llamado **correlación cruzada** en lugar de la convolución matemática. En este caso, *no rotamos el filtro* antes de aplicarlo a la imagen. El filtro se utiliza tal cual está, y el proceso consiste en realizar un producto punto entre el filtro y la región de la imagen, generando un valor en el mapa de características.

Por ejemplo, en lugar de rotar el filtro F 180 grados, lo aplicamos directamente a la imagen I sin alterarlo. El filtro se mueve a lo largo de la imagen, generando un valor para cada posición.

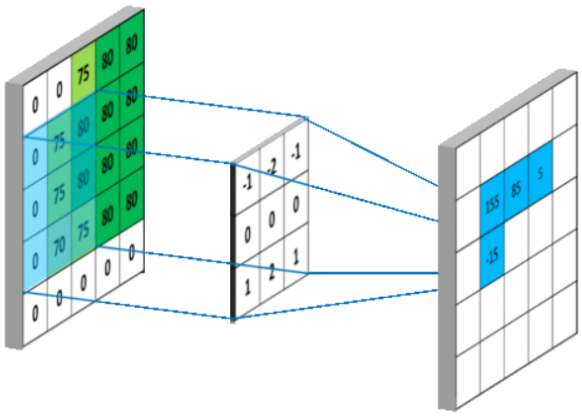

Figure 4: Correlación Cruzada: La Convolución en la Práctica del Deep Learning

1.1.3 ¿Por Qué Usamos Correlación Cruzada en Lugar de Convolución?

El uso de **correlación cruzada** en lugar de la convolución matemática tiene varias ventajas prácticas:

- **Eficiencia Computacional**: La correlación cruzada es más sencilla de implementar y computacionalmente más eficiente que la convolución matemática.
- **Resultados Similares**: Aunque la diferencia entre convolución y correlación cruzada es matemática, en la práctica, los resultados obtenidos son muy similares en tareas como la clasificación de imágenes.

Es por esto que, aunque en teoría se podría realizar una convolución "estricta" con rotación, la correlación cruzada se utiliza habitualmente en las redes neuronales convolucionales debido a su eficiencia.

1.2 Los otros componentes de una red neuronal convolucional

Además de las capas convolucionales, las CNN incluyen varias otras capas y operaciones que mejoran el rendimiento y la capacidad de generalización del modelo.

1.2.1 Convolución + ReLU

Convolución: Ya lo mencionamos, consiste en aplicar un filtro (o kernel) a la imagen o a las características extraídas por las capas anteriores. Esto produce un mapa de características (feature map), el cual es más pequeño que la entrada inicial debido a la reducción de tamaño que se produce en el proceso de convolución (especialmente cuando no se usa padding).

ReLU (Rectified Linear Unit): Después de aplicar la convolución, es común aplicar una función de activación como ReLU. Esta operación transforma las salidas negativas de la convolución a cero, lo cual ayuda a introducir no linealidades en el modelo, mejorando la capacidad de la red para aprender patrones complejos. La fórmula de ReLU es:

$$\text{ReLU}(x) = \max(0, x)$$

Esto ayuda a mejorar la eficiencia de entrenamiento al evitar el problema de desvanecimiento del gradiente, que puede ocurrir en redes profundas con activaciones más suaves.

1.2.2 Max Pooling

El **max pooling** es una operación de reducción de dimensionalidad que se utiliza para reducir el tamaño de las características espaciales, manteniendo las características más relevantes. Se aplica en un conjunto de valores dentro de una ventana de tamaño fijo (por ejemplo, 2×2) y selecciona el valor máximo dentro de esa ventana. El max pooling hace lo siguiente:

- Reduce el tamaño de las representaciones de las características (es decir, reduce las dimensiones de los mapas de características).
- Ayuda a reducir la cantidad de parámetros y el costo computacional.
- Introduce invarianza a pequeñas transformaciones, como traslaciones o escalados.

Por ejemplo, si tienes un mapa de características de tamaño 4×4 y aplicas un max pooling con una ventana de 2×2, el tamaño del mapa de salida será 2×2, y cada valor será el máximo dentro de su respectiva ventana 2×2.

1.2.3 Fully Connected (FC) + ReLU

Después de varias capas de convolución y pooling, la red neuronal convolucional suele tener una o más capas **fully connected** (FC), que son similares a las capas en redes neuronales tradicionales (densas). Las capas FC conectan todas las neuronas de una capa con todas las neuronas de la capa siguiente.

Fully Connected: En estas capas, cada nodo está conectado a todos los nodos de la capa anterior. La idea es combinar todas las características extraídas por las capas convolucionales para hacer predicciones más abstractas. Esta es la etapa donde la red ”decide” a qué clase pertenece la entrada.

ReLU: Al igual que con las capas convolucionales, las capas FC generalmente siguen con una activación ReLU para introducir no linealidades en la red y permitir la modelización de relaciones complejas.

1.2.4 Softmax

La función **softmax** se aplica generalmente en la última capa de la red, en especial cuando estamos trabajando con un problema de clasificación multiclase. La función softmax convierte las salidas de la red (que pueden ser valores continuos no normalizados) en probabilidades, de modo que la suma de todas las probabilidades sea igual a 1. Esto permite que el modelo prediga la clase más probable entre un conjunto de clases posibles.

La fórmula de la función softmax para un conjunto de n valores $z_1, z_2, ..., z_n$ es:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

Aquí, z_i es el valor de la salida de la red para la clase i, y e es la base del logaritmo natural.

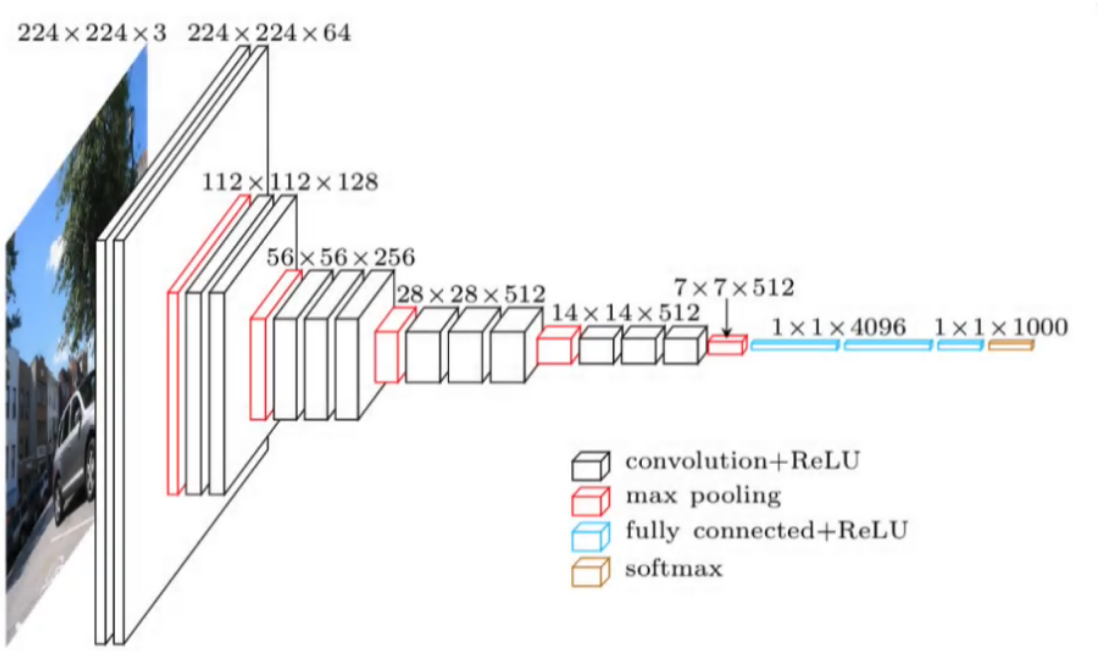

Figure 5: Ejemplo de la estructura de una Red Convolucional

1.2.5 Resumen del flujo en una red convolucional

1. **Entrada:** Una imagen pasa a través de la red.
2. **Capas Convolucionales + ReLU:** Se aplican filtros (kernels) a la imagen y las salidas de estas capas se activan mediante ReLU.
3. **Max Pooling:** Reduce las dimensiones de las características y ayuda a abstraer la información.
4. **Capas Fully Connected + ReLU:** Combinan las características extraídas para tomar decisiones más abstractas.
5. **Softmax:** En la última capa, la función softmax convierte las activaciones en probabilidades de clasificación.

Este proceso permite a la CNN extraer características jerárquicas de la imagen, comenzando con características simples como bordes y texturas en las primeras capas y avanzando hacia características más complejas (formas, objetos) a medida que avanzamos en la red. El uso de **ReLU** y **softmax** asegura que el modelo pueda manejar datos no lineales y proporcionar salidas probabilísticas útiles para tareas como clasificación.

1.3 Convoluciones Espacialmente Separables y Convoluciones Separables por Profundidad

Al observar las redes neuronales convolucionales que son muy grandes, como ResNets, VGG y similares, surge la pregunta de cómo podemos hacer que todas estas redes sean más pequeñas con menos parámetros y al mismo tiempo mantener el mismo nivel de precisión o incluso mejorar la generalización de el modelo usando una menor cantidad de parámetros.

1.3.1 Convoluciones Espacialmente Separables

Las **convoluciones espacialmente separables** son un concepto relativamente sencillo de entender, y representan la idea de separar una convolución en

dos de manera eficiente. Sin embargo, estas tienen algunas limitaciones significativas que hacen que no se usen ampliamente en el aprendizaje profundo.

El término ”espacialmente separable” hace referencia a que se trabaja principalmente con las dimensiones espaciales de una imagen y un kernel, es decir, el ancho y la altura. La otra dimensión, conocida como la *profundidad*, hace referencia al número de canales de la imagen.

En una convolución espacialmente separable, un kernel se divide en dos kernels más pequeños. El caso más común es dividir un kernel 3×3 en dos kernels: uno de 3×1 y otro de 1×3, como se muestra a continuación:

$$\begin{bmatrix} 3 & 6 & 9 \\ 4 & 8 & 12 \\ 5 & 10 & 15 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Figure 6: Separando espacialmente un núcleo de 3x3.

De esta manera, en lugar de realizar una convolución con 9 multiplicaciones, se realizan dos convoluciones con 3 multiplicaciones cada una (6 en total), logrando el mismo efecto. Con menos multiplicaciones, la complejidad computacional disminuye y la red neuronal puede ejecutar más rápido.

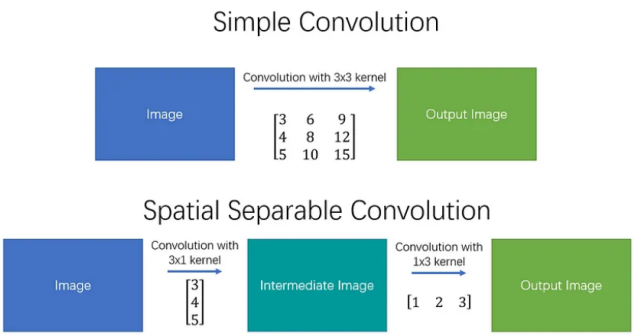

Figure 7: Convolución simple y espacialmente separable.

Uno de los ejemplos más conocidos de un kernel que se puede separar espacialmente es el **kernel Sobel**, utilizado para detectar bordes en las imágenes. Este kernel puede ser separado en dos de la siguiente manera:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Figure 8: Separando el kernel de Sobel.

El principal problema con las convoluciones espacialmente separables es que no todos los kernels pueden ser ”separados” en dos kernels más pequeños. Esto se convierte en un inconveniente significativo durante el entrenamiento, ya que, de todos los posibles kernels que la red neuronal podría adoptar, solo puede terminar utilizando aquellos que pueden separarse en dos kernels más pequeños.

1.3.2 Convoluciones Separables por Profundidad

A diferencia de las convoluciones espacialmente separables, las **convoluciones separables por profundidad** trabajan con kernels que no se pueden ”factorizar” en dos kernels más pequeños. Por esta razón, las convoluciones separables por profundidad son más comúnmente utilizadas. Este tipo de convolución es el que se encuentra en `keras.layers.SeparableConv2D` o `tf.layers.separable_conv2d`.

El término ”separable por profundidad” hace referencia a que no solo se trabajan con las dimensiones espaciales, sino también con la dimensión de la profundidad, es decir, el número de canales. Por ejemplo, una imagen puede tener 3 canales: RGB. Después de varias convoluciones, una imagen puede tener múltiples canales. Podemos imaginar que cada canal representa una interpretación particular de la imagen; por ejemplo, el canal ”rojo” interpreta la ”rojez” de cada píxel, el canal ”azul” interpreta el ”azul” de cada píxel y el canal ”verde” interpreta el ”verde” de cada píxel. Una imagen con

64 canales tendría 64 interpretaciones diferentes de la misma imagen.

Similar a las convoluciones espacialmente separables, una convolución separable por profundidad divide un kernel en dos kernels separados que realizan dos convoluciones: la convolución por profundidad y la convolución puntual.

En una convolución separable por profundidad, el kernel es descompuesto en dos pasos:

1. **Convolución por profundidad (Depthwise Convolution)**: En este paso, se realiza una convolución de cada canal de la imagen de manera independiente, usando un kernel que es de tamaño $k \times k \times 1$. Esto se realiza para cada canal de la imagen de entrada.

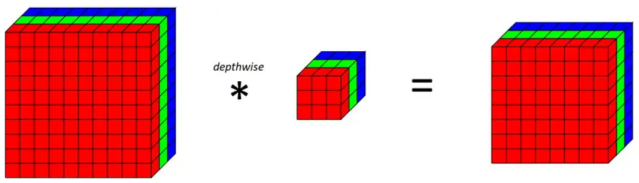

Figure 9: Aplicando un filtro convolucional en profundidad en 10x10x3 salidas de volumen de entrada 8x8x3 volumen.

2. **Convolución puntual (Pointwise Convolution)**: A continuación, se aplica una convolución 1×1, que se encarga de combinar los resultados de la convolución de profundidad en un solo valor por píxel para cada canal de salida. Este paso se aplica en todas las combinaciones posibles de canales.

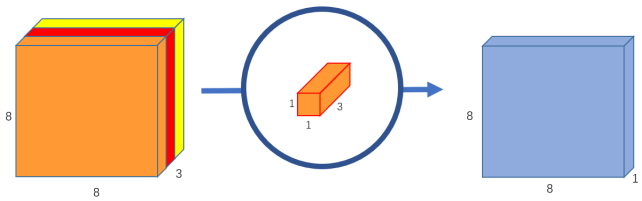

Figure 10: La convolución puntual transforma una imagen de 3 canales en una imagen de 1 canal.

Esta descomposición resulta en una reducción significativa de la cantidad de parámetros y operaciones de cálculo, en comparación con las convoluciones tradicionales que usan un kernel de mayor tamaño. Por ejemplo, si se tiene un filtro 3×3 que se aplica a una imagen con C canales de entrada, en una convolución estándar se requieren $3 \times 3 \times C$ parámetros para el kernel. Sin embargo, en una convolución separable por profundidad, se requieren $3 \times 3 \times 1$ parámetros para la convolución por profundidad y $1 \times 1 \times C$ parámetros para la convolución puntual, lo que reduce el número de parámetros total.

2 Justificación y Objetivos del Experimento

En la introducción se abordaron los fundamentos de las convoluciones normales y convoluciones separables, dos tipos de operaciones fundamentales en redes neuronales convolucionales. Las convoluciones normales permiten combinar información de todos los canales de una imagen en un único mapa de características, mientras que las convoluciones separables procesan cada canal de forma independiente y realizan una operación de combinación en profundidad para reducir los mapas generados a un solo mapa final. Las convoluciones separables se han popularizado debido a su capacidad para reducir el número de parámetros y, en algunos casos, mejorar la eficiencia del modelo sin comprometer el rendimiento.

El objetivo de este trabajo es explorar y comparar el rendimiento de estas dos arquitecturas en el contexto de un problema clásico de clasificación de imágenes. A través de este experimento, se pretende analizar no solo el rendimiento en términos de precisión y tiempo de entrenamiento, sino también entender cómo la reducción de parámetros a través de las convoluciones separables afecta la capacidad de la red para generalizar y aprender patrones en los datos.

A continuación, se presentará la construcción de dos modelos de redes neuronales convolucionales: uno utilizando convoluciones normales y otro con convoluciones separables. Ambos modelos serán entrenados y evaluados utilizando un conjunto de datos de clasificación de imágenes, lo que permitirá establecer comparaciones entre ambos enfoques y profundizar en los efectos de las modificaciones propuestas en la estructura de las redes.

3 Selección y Organización del Conjunto de Datos

En este trabajo, utilizaremos el conjunto de datos **Animals10** disponible en Kaggle. Este conjunto de datos contiene aproximadamente 28,000 imágenes de calidad media, que pertenecen a 10 categorías de animales: perro, gato, caballo, araña, mariposa, gallina, oveja, vaca, ardilla y elefante. El conjunto fue creado y puesto a disposición por **Alessio Corrado** y se encuentra accesible a través del siguiente enlace: Animals10 Dataset.

Este conjunto de datos será utilizado para entrenar y evaluar los modelos de redes neuronales convolucionales y separables, siguiendo la organización y la estructura que detallaremos a continuación.

El primer paso consistió en definir un tamaño fijo para las imágenes de entrada, en este caso, de 64x64 píxeles. A continuación, se organizó el conjunto de datos, donde las imágenes están almacenadas en carpetas separadas por categorías. Cada una de estas carpetas contiene imágenes que pertenecen a una clase específica, como perros, gatos, caballos, entre otras. Al cargar las imágenes, estas se redimensionan y se convierten al formato RGB para garantizar una representación consistente de los colores.

Para facilitar el entrenamiento, se asigna un número a cada categoría (por ejemplo, un índice del 0 al 9) y se utiliza este mapeo para etiquetar las imágenes de acuerdo con su clase. Una vez que las imágenes están listas, se procede a dividir el conjunto de datos en tres subconjuntos: entrenamiento (70%), validación (15%) y prueba (15%). Esta división permite evaluar el rendimiento del modelo durante y después del entrenamiento.

El preprocesamiento de las imágenes incluye la normalización, que ajusta los valores de los píxeles para que estén en el rango de 0 a 1, y la codificación one-hot de las etiquetas, lo que convierte las categorías en vectores binarios adecuados para la clasificación. Además, se utilizó el batching y el prefetching para mejorar la eficiencia durante el entrenamiento, asegurando que los datos sean procesados de manera rápida y optimizada.

4 Red Neuronal Convolucional

4.1 Arquitectura del Modelo

Para la arquitectura de la red neuronal convolucional, utilizaremos una estructura organizada en varias capas secuenciales, diseñadas para procesar imágenes y realizar clasificación multiclase. La red se compone de las siguientes capas:

- **Capa de entrada:** La red recibe imágenes con un tamaño de 64×64 píxeles y 3 canales de color (RGB).
- **Capas convolucionales separables:** La red incluye varias capas de convolución separable, que permiten extraer características espaciales de las imágenes. Estas capas están seguidas por capas de *batch normalization*, que estabilizan y aceleran el proceso de entrenamiento, y por capas de *max pooling*, que reducen las dimensiones espaciales de las características extraídas y optimizan la carga computacional.

- **Capa de aplanamiento:** Después de las capas convolucionales, las características extraídas se aplanan para ser procesadas por las capas densas, lo que permite que la red interprete y clasifique la información de manera estructurada.
- **Capas densas:** La red incluye varias capas densas (*fully connected*), encargadas de la toma de decisiones finales. Estas capas están regularizadas mediante *L2 regularization* y se aplica *dropout* para prevenir el sobreajuste y mejorar la generalización del modelo.
- **Capa de salida:** La red culmina con una capa de salida que consta de 10 unidades, correspondientes a las 10 clases de animales. La activación utilizada en esta capa es *softmax*, que convierte los valores de salida en probabilidades, permitiendo realizar la clasificación multiclase.

4.1.1 Hiperparámetros

Con la estructura base de la arquitectura definida, es necesario buscar los hiperparámetros óptimos para entrenar la red neuronal. Para esto, se utilizó la herramienta Optuna, que permite la optimización de los hiperparámetros mediante la búsqueda de valores que maximizan el rendimiento del modelo.

Optuna es una biblioteca de optimización automática que facilita la búsqueda de hiperparámetros de manera eficiente. A través de un proceso de prueba y error, Optuna ajusta los hiperparámetros como el número de capas, el tamaño de los filtros en las capas convolucionales, el número de unidades en las capas densas, las tasas de aprendizaje, entre otros. Al realizar esta búsqueda, Optuna evalúa el modelo en función de su desempeño, utilizando métricas como la precisión o la pérdida, y optimiza los parámetros para mejorar el rendimiento de la red.

El uso de Optuna permite encontrar combinaciones de hiperparámetros que pueden no ser evidentes a simple vista, mejorando así la capacidad generalizadora del modelo y reduciendo el riesgo de sobreajuste, lo cual es crucial para lograr una red eficiente en tareas de clasificación de imágenes.

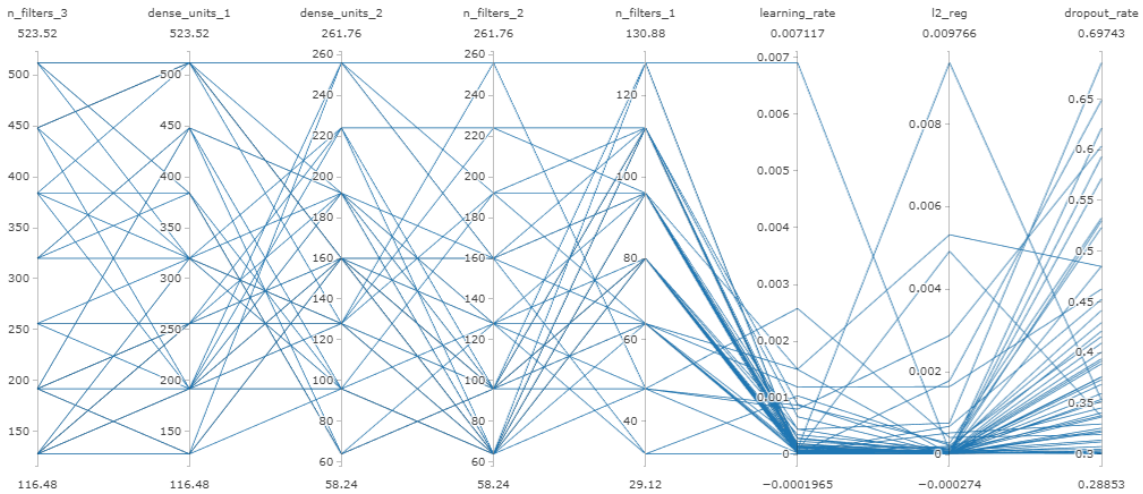

Figure 11: 50 experimentos para optimizar los hiperparámetros de Optuna

Finalmente obteniendo el siguiente modelo:

Layer (type)	Output Shape	Param #
conv2d_141 (Conv2D)	(None, 62, 62, 112)	3,136
batch_normalization_141 (BatchNormalization)	(None, 62, 62, 112)	448
max_pooling2d_141 (MaxPooling2D)	(None, 31, 31, 112)	0
conv2d_142 (Conv2D)	(None, 29, 29, 64)	64,576
batch_normalization_142 (BatchNormalization)	(None, 29, 29, 64)	256
max_pooling2d_142 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_143 (Conv2D)	(None, 12, 12, 448)	258,496
batch_normalization_143 (BatchNormalization)	(None, 12, 12, 448)	1,792
max_pooling2d_143 (MaxPooling2D)	(None, 6, 6, 448)	0
flatten_47 (Flatten)	(None, 16128)	0
dense_141 (Dense)	(None, 512)	8,258,048
dropout_94 (Dropout)	(None, 512)	0
dense_142 (Dense)	(None, 160)	82,080
dropout_95 (Dropout)	(None, 160)	0
dense_143 (Dense)	(None, 10)	1,610

Table 1: Resumen de la arquitectura del modelo ”sequential_47”

Total params: 26,008,832 (99.22 MB)
Trainable params: 8,669,194 (33.07 MB)
Non-trainable params: 1,248 (4.88 KB)
Optimizer params: 17,338,390 (66.14 MB)

4.2 Entrenamiento del modelo

Primero, el modelo se entrenó utilizando la GPU T4 de Google Colab. Se completaron 15 de las 20 épocas previstas, debido a que el generador de datos o el conjunto de datos utilizado para el entrenamiento no tenía suficientes muestras o no estaba configurado correctamente para generar los lotes necesarios para completar todas las épocas. Sin embargo, como veremos más adelante, esto no representó un inconveniente significativo para los resultados obtenidos. El tiempo promedio por época fue de 773.33 segundos, mientras que el tiempo total de entrenamiento fue de 11,600 segundos, lo que equivale a 3.22 horas.

Posteriormente, utilizamos la CPU que ofrece Google Colab con mayor cantidad de memoria RAM, lo que redujo los tiempos de entrenamiento significativamente. En esta configuración, el tiempo promedio por época fue de 202 segundos, con un tiempo total de 4,053 segundos, equivalente aproximadamente a 1.13 horas.

La GPU T4 es más potente que la CPU para operaciones paralelas como el entrenamiento de redes neuronales, pero el menor tiempo en la CPU puede deberse a varios factores:

1. **Tamaño del modelo y conjunto de datos**: Si el modelo o los datos no son lo suficientemente grandes para saturar la GPU, la CPU puede ser más eficiente, evitando el tiempo de transferencia de datos entre memorias. Esto se observó cuando solo se completaron 15 de las 20 épocas previstas por el tamaño del conjunto de datos.

2. **Cola de datos y procesamiento previo**: Si la preparación de datos o el procesamiento

previo no están optimizados, pueden crear un cuello de botella que favorezca el uso de la CPU. Esto se evidenció en la optimización de hiperparámetros con Optuna, un proceso exigente.

Además, aunque el entrenamiento se realizó en Google Colab, la ejecución simultánea de un juego en línea en el sistema local podría haber afectado ligeramente la estabilidad del entorno debido a la latencia en la conexión WiFi.

Los resultados obtenidos para el set de test fueron los siguientes:

- Test loss: 0.1617
- Test accuracy: 0.9957

En cuanto al rendimiento en los conjuntos de entrenamiento y validación:

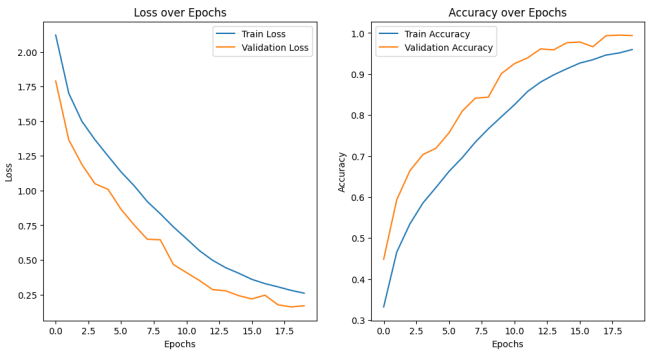

Figure 12: Pérdida y Precisión durante el Entrenamiento

Interpretación de la gráfica de pérdida:

Ambas pérdidas (entrenamiento y validación) disminuyen con el avance de las épocas, lo que indica que el modelo está aprendiendo y generalizando adecuadamente. Al principio, la pérdida de validación es ligeramente mayor que la de entrenamiento, lo cual es normal. Hacia el final del entrenamiento, las líneas se acercan y la pérdida de validación incluso aumenta un poco, lo que sugiere que el modelo podría estar comenzando a sobreajustar (overfitting) ligeramente los datos de entrenamiento. No obstante, la diferencia es mínima, por lo que no es motivo de preocupación en este caso.

Interpretación de la gráfica de precisión:

Tanto la precisión de entrenamiento como la de validación aumentan con las épocas, lo que confirma que el modelo está aprendiendo correctamente. La precisión de validación se estabiliza hacia el final, lo cual es común en los procesos de entrenamiento. Nuevamente, la diferencia entre la precisión de entrenamiento y validación es pequeña, lo que indica una buena capacidad de generalización del modelo.

4.3 Prediciones usando este modelo

Ahora utilizaremos otro Colab para evaluar cómo predice el modelo utilizando únicamente los datos de prueba, es decir, aquellos que el modelo no ha visto antes.

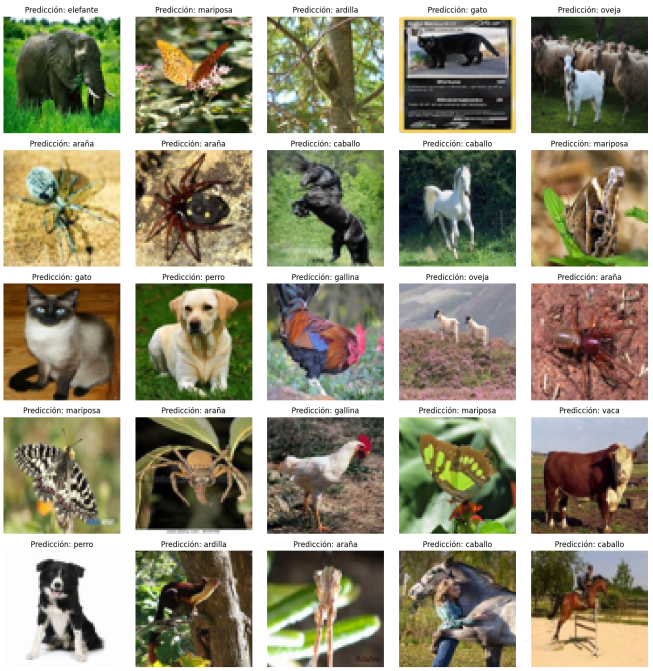

Figure 13: 25 imágenes correctamente clasificadas de 25.

5 Red Neuronal con Convoluciones Separables Profundas

5.1 Arquitectura del Modelo

Para la arquitectura de la red neuronal con convoluciones profundas, se mantendrá la misma estructura que en el caso de las convoluciones convencionales, con el objetivo de poder comparar ambos resultados en etapas posteriores. No obstante, debido a la utilización de un tipo diferente de convoluciones, se han realizado algunos ajustes leves, que se detallan a continuación.

En esta nueva implementación, se emplea la capa `SeparableConv2D` para las tres capas convolucionales. Esta capa divide la operación de convolución en dos pasos: primero, una convolución de profundidad (*depthwise*) y luego una convolución punto a punto (*pointwise*), lo que, en general, mejora la eficiencia computacional sin sacrificar significativamente el rendimiento.

5.1.1 Hiperparámetros

Nuevamente, con la estructura base de la arquitectura definida, es necesario buscar los hiperparámetros óptimos para entrenar la red neuronal. Para ello, se utilizó nuevamente la herramienta Optuna. Sin embargo, para esta prueba, utilizaremos los mismos hiperparámetros que se emplearon en el caso de la red neuronal de convoluciones.

Layer (type)	**Output Shape**	**Param #**
separable_conv2d_1 (SeparableConv2D)	(None, 62, 62, 112)	475
batch_normalization (BatchNormalization)	(None, 62, 62, 112)	448
max_pooling2d (MaxPooling2D)	(None, 31, 31, 112)	0
separable_conv2d_2 (SeparableConv2D)	(None, 29, 29, 64)	8,240
batch_normalization_1 (BatchNormalization)	(None, 29, 29, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	0
separable_conv2d_3 (SeparableConv2D)	(None, 12, 12, 448)	29,696
batch_normalization_2 (BatchNormalization)	(None, 12, 12, 448)	1,792
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 448)	0
flatten (Flatten)	(None, 16128)	0
dense (Dense)	(None, 512)	8,258,048
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 160)	82,080
dropout_1 (Dropout)	(None, 160)	0
dense_2 (Dense)	(None, 10)	1,610

Table 2: Resumen de la arquitectura del modelo ”sequential”

Total params: 25,145,441 (95.92 MB)
Trainable params: 8,381,397 (31.97 MB)
Non-trainable params: 1,248 (4.88 KB)
Optimizer params: 16,762,796 (63.94 MB)

5.2 Entrenamiento del modelo

A diferencia del modelo anterior, utilizando la GPU T4 de Google Colab, este sí pudo entrenarse durante las 20 épocas, con un tiempo promedio de 557.5 segundos por época y un total de 11,150 segundos (aproximadamente 3 horas y 5 minutos).

Sin embargo, realizamos otra prueba con la CPU de alta capacidad de RAM. En esta ocasión, cada época tuvo un tiempo promedio de 155 segundos, lo que resultó en una duración total de 3,109 segundos, equivalente a 0.86 horas o 51.82 minutos. Los motivos de esta diferencia probablemente sean similares a los observados en el caso de la red convolucional.

Los resultados obtenidos en el conjunto de prueba (test) son los siguientes:

- **Test loss:** 0.1498572826385498
- **Test accuracy:** 0.9969450235366821

En cuanto al rendimiento en los conjuntos de entrenamiento y validación:

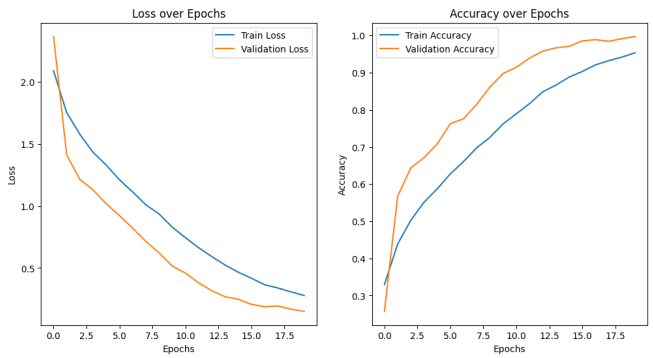

Figure 14: Pérdida y Precisión durante el Entrenamiento

- **Gráfica de pérdida:** Ambas líneas (entrenamiento y validación) presentan una tendencia descendente, lo que indica que el modelo mejora su rendimiento. Inicialmente, la pérdida disminuye rápidamente, luego se estabiliza y, hacia el final, se observa un ligero aumento en la pérdida de validación, lo cual podría ser una señal de sobreajuste.
- **Gráfica de precisión:** Ambas líneas muestran una tendencia ascendente, reflejando una mejora en las predicciones. La precisión aumenta rápidamente al inicio y después se estabiliza. Existe una pequeña brecha entre las precisiones de entrenamiento y validación, lo que también podría indicar un leve sobreajuste.

5.3 Prediciones usando este modelo

A diferencia del modelo con convoluciones, este modelo presentó una gran dificultad al intentar clasificar un conjunto de datos no conocido.

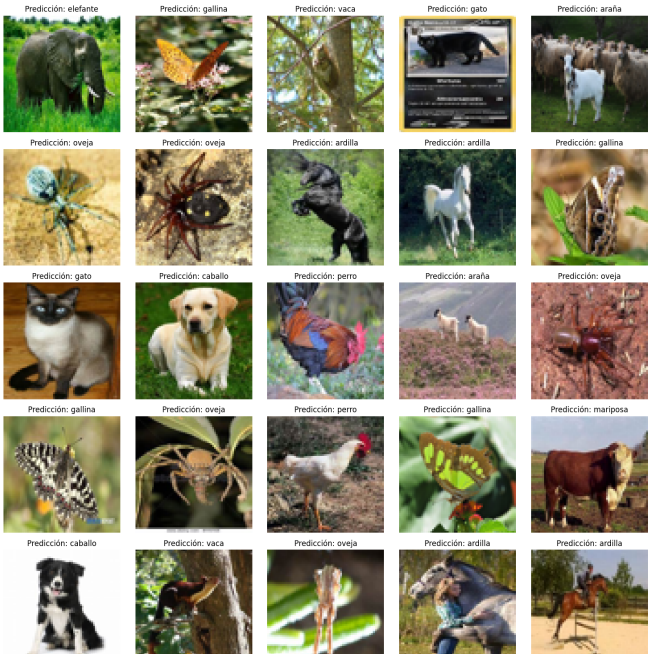

Figure 15: 3 imágenes correctamente clasificadas de 25.

Es importante señalar que, aunque en el entorno de Google Colab donde se entrenó el modelo se logró realizar la predicción, los resultados obtenidos no son completamente confiables. Al repetir varias veces el proceso, los resultados pre-

sentados en la Figura 13 resultaron ser los más frecuentes.

Actualización:

Me llamó la atención por qué, al evaluar directamente el modelo después de entrenarlo, el *test accuracy* daba un valor cercano al 100% de precisión. Sin embargo, cuando pasábamos las imágenes y queríamos visualizar las etiquetas, parecía que no clasificaba correctamente. Para evitar posibles errores relacionados con variables o configuraciones, se ejecutó el proceso en otro cuaderno de Colab.
Al analizar las imágenes, pudimos ver que el modelo clasificaba correctamente. Por ejemplo, las arañas aparecían etiquetadas como ovejas, los caballos como ardillas, y siendo los gatos y los elefantes los únicos en ser clasificados correctamente. Esto indica que el modelo había aprendido correctamente las características de las clases.
Al revisar el código, me di cuenta de que cada vez que generaba el conjunto de datos, las clasificaciones se mapeaban de manera diferente. Esto explicaba por qué las etiquetas aparecían de forma distinta, ya que se asignaron en otro orden. En cambio, en el cálculo de *test accuracy*, las predicciones se evaluaban utilizando los valores de probabilidad (es decir, los números), lo que resultaba en un buen desempeño. Después de realizar un mapeo correcto de esos números, el modelo comenzó a clasificar correctamente.

Nota:

Los notebooks de entrenamiento de modelo y búsqueda de hiperparámetros se encuentran en: `Colab - Entrenamiento de Modelo`.
Los experimentos de Optuna están disponibles en: `DAGsHub - Experimentos Optuna`.
Las predicciones para clasificar se pueden consultar en: `Colab - Predicciones para Clasificación`.

Donde se repitieron los entrenamientos con la CPU de alta capacidad de RAM en: `Colab - CPU Alta Capacidad de RAM`.

6 Convoluciones separables profundas con hiperparámetros optimizados por Optuna

6.1 Arquitectura del Modelo

Para la arquitectura de la red neuronal con convoluciones profundas utilizando esta vez parametros distintos al de convoluciones, dichos para metros fueron optimizados por optuna

6.1.1 Hiperparámetros

Se hicieron 50 experimentos para buscar los hiperparemtros que mejor optimizaran el modelo obteniendo los siguientes resultados:

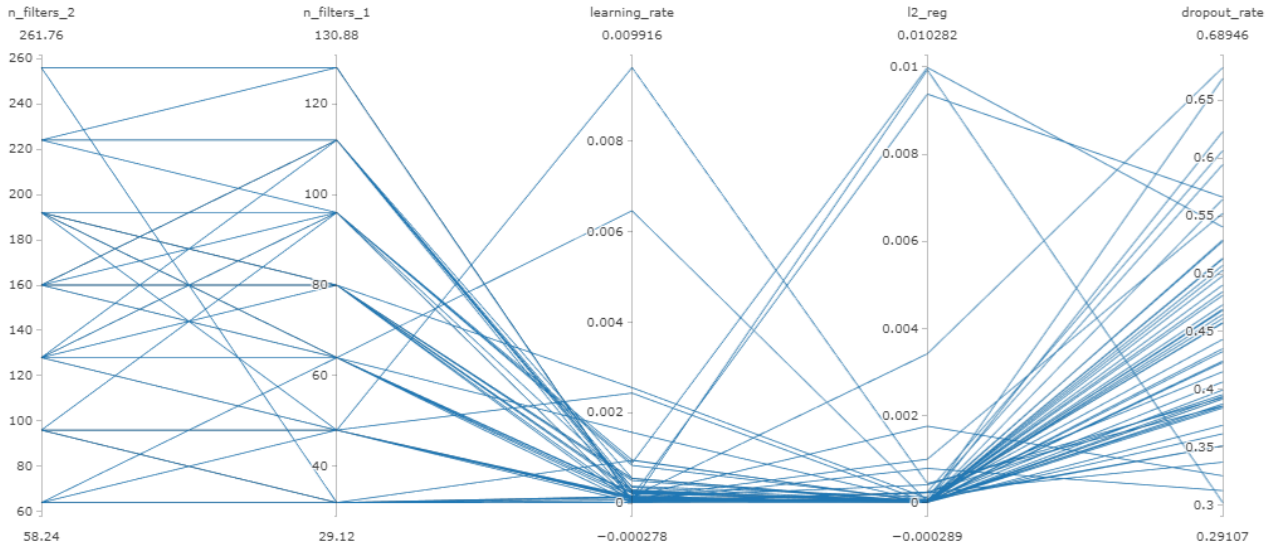

Figure 16: 50 experimentos para optimizar los hiperparámetros de Optuna

Layer (type)	**Output Shape**	**Param #**
separable_conv2d (SeparableConv2D)	(None, 62, 62, 48)	219
batch_normalization (BatchNormalization)	(None, 62, 62, 48)	192
max_pooling2d (MaxPooling2D)	(None, 31, 31, 48)	0
separable_conv2d_1 (SeparableConv2D)	(None, 29, 29, 96)	5,136
batch_normalization_1 (BatchNormalization)	(None, 29, 29, 96)	384
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 96)	0
separable_conv2d_2 (SeparableConv2D)	(None, 12, 12, 512)	50,528
batch_normalization_2 (BatchNormalization)	(None, 12, 12, 512)	2,048
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 512)	0
flatten (Flatten)	(None, 18432)	0
dense (Dense)	(None, 512)	9,437,696
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 160)	82,080
dropout_1 (Dropout)	(None, 160)	0
dense_2 (Dense)	(None, 10)	1,610

Table 3: Resumen de la arquitectura del modelo ”sequential”

Total params: 28,737,057 (109.62 MB)
Trainable params: 9,578,581 (36.54 MB)
Non-trainable params: 1,312 (5.12 KB)
Optimizer params: 19,157,164 (73.08 MB)

6.2 Entrenamiento del modelo

Originalmente, todas las redes se pensaban ejecutar utilizando la GPU T4. Sin embargo, al desarrollar esta red, perdimos acceso a cualquier GPU de Google Colab, lo que nos llevó a continuar sin ella, pero manteniendo el acceso a la alta capacidad de la RAM. Por lo tanto, no se intentó usar la GPU T4 en este caso. A pesar de ello, como se mencionó anteriormente, logramos obtener mejores resultados con esta configuración.

Este modelo requirió 76 segundos por cada época, y en total, en las 20 épocas, tardó 1520 segundos (25 minutos y 20 segundos).

Los resultados obtenidos en el conjunto de prueba (test) son los siguientes:

- **Test loss:** 0.06702090799808502
- **Test accuracy:** 0.9971995949745178

En cuanto al rendimiento en los conjuntos de entrenamiento y validación:

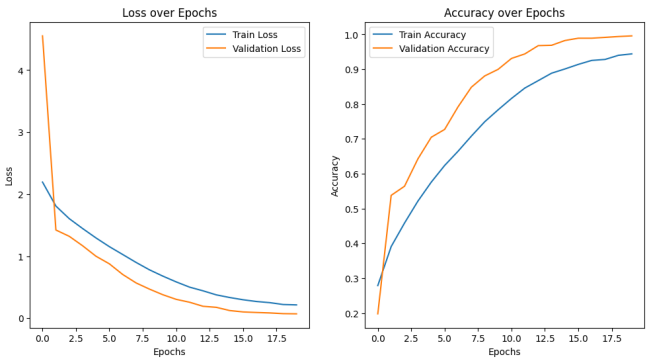

Figure 17: Pérdida y Precisión durante el Entrenamiento

- **Pérdida (Loss) a lo largo de las Épocas:** La pérdida en los conjuntos de entrenamiento y validación disminuye con el aumento de las épocas, lo que indica que el modelo está aprendiendo y mejorando sus predicciones, a pesar de una ligera diferencia entre ambos conjuntos.
- **Precisión a lo largo de las Épocas:** La precisión en los conjuntos de entrenamiento y validación aumenta con el número de épocas, lo que refleja una mejora en las predicciones del modelo, mostrando una tendencia a generalizar bien a nuevos datos.

6.3 Prediciones usando este modelo

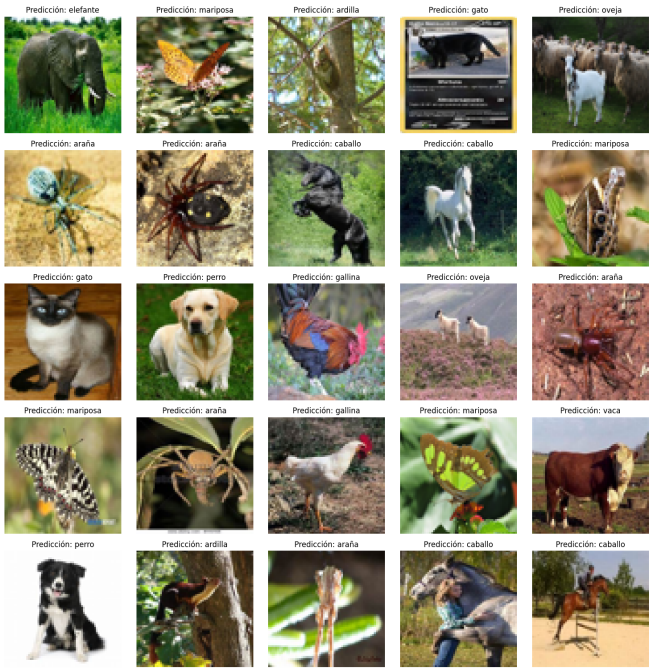

Figure 18: 25 imágenes correctamente clasificadas de 25.

> Nota:
>
> Este modelo, al igual que los anteriores, no presentó problemas para clasificar correctamente a todos los animales. Para más detalles, recomiendo revisar el notebook de predicciones, donde dejé un segmento de código que permite subir imágenes a Google Colab y predecirlas. De esta forma, se puede realizar el mismo procedimiento con cualquier imagen de Google.

7 Comparación entre la red convolucional y la red convolucional separable profunda

A continuación se presentan las arquitecturas de tres modelos de redes neuronales: una red convolucional (Conv2D) y dos versiones de redes convolucionales separables profundas (SeparableConv2D).

Capa (Tipo)	**Modelo 1: Conv2D**	**Modelo 2: SeparableConv2D (Hiperparámetros iguales)**	**Modelo 3: SeparableConv2D (Otros hiperparámetros)**
Capa 1	conv2d (Conv2D)	separable_conv2d_1 (SeparableConv2D)	separable_conv2d (SeparableConv2D)
Forma de salida	(None, 62, 62, 112)	(None, 62, 62, 112)	(None, 62, 62, 48)
Número de parámetros	3,136	475	219
Capa 2	batch_normalization	batch_normalization	batch_normalization
Forma de salida	(None, 62, 62, 112)	(None, 62, 62, 112)	(None, 62, 62, 48)
Número de parámetros	448	448	192
Capa 3	max_pooling2d (MaxPooling2D)	max_pooling2d (MaxPooling2D)	max_pooling2d (MaxPooling2D)
Forma de salida	(None, 31, 31, 112)	(None, 31, 31, 112)	(None, 31, 31, 48)
Número de parámetros	0	0	0
Capa 4	conv2d_1 (Conv2D)	separable_conv2d_2 (SeparableConv2D)	separable_conv2d_1 (SeparableConv2D)
Forma de salida	(None, 29, 29, 64)	(None, 29, 29, 64)	(None, 29, 29, 96)
Número de parámetros	64,576	8,240	5,136
Capa 5	batch_normalization_1	batch_normalization_1	batch_normalization_1
Forma de salida	(None, 29, 29, 64)	(None, 29, 29, 64)	(None, 29, 29, 96)
Número de parámetros	256	256	384
Capa 6	max_pooling2d_1 (MaxPooling2D)	max_pooling2d_1 (MaxPooling2D)	max_pooling2d_1 (MaxPooling2D)
Forma de salida	(None, 14, 14, 64)	(None, 14, 14, 64)	(None, 14, 14, 96)
Número de parámetros	0	0	0
Capa 7	conv2d_2 (Conv2D)	separable_conv2d_3 (SeparableConv2D)	separable_conv2d_2 (SeparableConv2D)
Forma de salida	(None, 12, 12, 448)	(None, 12, 12, 448)	(None, 12, 12, 512)
Número de parámetros	258,496	29,696	50,528
Capa 8	batch_normalization_2	batch_normalization_2	batch_normalization_2
Forma de salida	(None, 12, 12, 448)	(None, 12, 12, 448)	(None, 12, 12, 512)
Número de parámetros	1,792	1,792	2,048
Capa 9	max_pooling2d_2 (MaxPooling2D)	max_pooling2d_2 (MaxPooling2D)	max_pooling2d_2 (MaxPooling2D)
Forma de salida	(None, 6, 6, 448)	(None, 6, 6, 448)	(None, 6, 6, 512)
Número de parámetros	0	0	0
Capa 10	flatten (Flatten)	flatten (Flatten)	flatten (Flatten)
Forma de salida	(None, 16128)	(None, 16128)	(None, 18432)
Número de parámetros	0	0	0
Capa 11	dense (Dense)	dense (Dense)	dense (Dense)

Capa (Tipo)	Modelo 1: Conv2D	Modelo 2: SeparableConv2D (Hiperparámetros iguales)	Modelo 3: SeparableConv2D (Otros hiperparámetros)
Forma de salida	(None, 512)	(None, 512)	(None, 512)
Número de parámetros	8,258,048	8,258,048	9,437,696
Capa 12	dropout (Dropout)	dropout (Dropout)	dropout (Dropout)
Forma de salida	(None, 512)	(None, 512)	(None, 512)
Número de parámetros	0	0	0
Capa 13	dense_1 (Dense)	dense_1 (Dense)	dense_1 (Dense)
Forma de salida	(None, 160)	(None, 160)	(None, 160)
Número de parámetros	82,080	82,080	82,080
Capa 14	dropout_1 (Dropout)	dropout_1 (Dropout)	dropout_1 (Dropout)
Forma de salida	(None, 160)	(None, 160)	(None, 160)
Número de parámetros	0	0	0
Capa 15	dense_2 (Dense)	dense_2 (Dense)	dense_2 (Dense)
Forma de salida	(None, 10)	(None, 10)	(None, 10)
Número de parámetros	1,610	1,610	1,610
Total de parámetros	26,008,832	25,145,441	28,737,057
Parámetros entrenables	8,669,194	8,381,397	9,578,581
Parámetros no entrenables	1248	1248	1312
Parámetros optimizados	17,338,390	16,762,796	19,157,164

Estos modelos se entrenaron a lo largo de 20 épocas y lograron los siguientes resultados.

Modelo	Test Loss	Accuracy	Tiempo de Entrenamiento
Conv2D (Modelo 1)	0.1617	99.57%	1h 13m
SeparableConv2D Profundo (Modelo 2)	0.1499	99.69%	51m 49s
SeparableConv2D con Hiperparámetros (Modelo 3)	0.0670	99.72%	25m 20s

Table 5: Comparativa de rendimiento y tiempo de entrenamiento entre modelos de redes convolucionales.

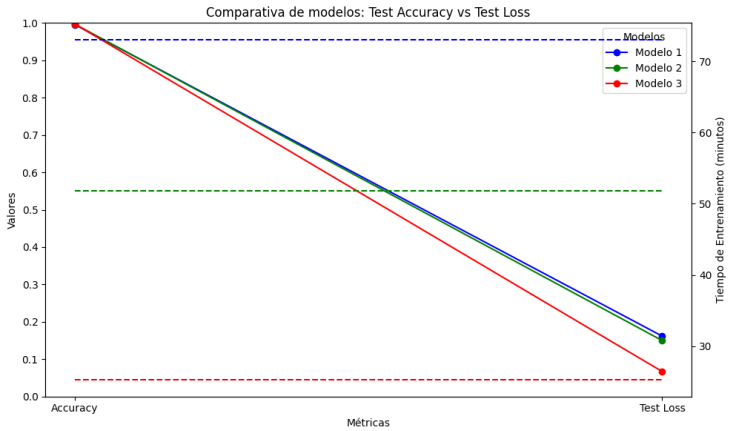

Figure 19: Rendimiento y tiempo de entrenamiento entre modelos de redes convolucionales.

Comparación de Modelos y Optimización de Parámetros

- El **Modelo 2** y el **Modelo 3** utilizan `SeparableConv2D`, que tiene menos parámetros por capa que las capas `Conv2D` estándar del **Modelo 1**. Sin embargo, otros factores como la cantidad de filtros o capas adicionales implican que la reducción total de parámetros no es significativa.
- El **Modelo 3** tiene el mayor número total de parámetros (**28,737,057**), seguido del **Modelo 1** (**26,008,832**) y el **Modelo 2** (**25,145,441**).
- A pesar de tener más parámetros, el **Modelo 3** optimiza la mayor cantidad de parámetros (**19,157,164**), lo que le da una mayor capacidad de adaptación.
- El **Modelo 3** se destaca con el mejor `Test Loss` (**0.0670**) y la mayor `Accuracy` (**99.72%**), lo que demuestra que es el más preciso y eficiente en términos de generalización.
- El **Modelo 3** es el más eficiente en tiempo, con **25 minutos y 20 segundos**, mucho más rápido que el **Modelo 1** (**1h 13m**) y el **Modelo 2** (**51m 49s**).
- El **Modelo 3** presenta más parámetros optimizados, lo que indica que la optimización independiente de su arquitectura, sin replicar los hiperparámetros de la capa de convolución, mejora su capacidad y le otorga una ventaja al lograr un mejor desempeño en comparación con los demás modelos.
- La diferencia entre los `parámetros entrenables` y los `no entrenables` es mínima en los tres modelos, indicando que la mayoría de los parámetros en cada modelo son entrenables, contribuyendo directamente a la optimización.
- El **Modelo 3** muestra la mayor capacidad para aprender patrones complejos, lo que lo hace más adecuado para tareas que requieren un alto grado de precisión.

Conclusión Final:

- El **Modelo 3** se destaca por su precisión y eficiencia, con el mejor `Test Loss` y `Accuracy`, y es el modelo más adecuado para tareas complejas debido a su capacidad de optimización y tiempo de entrenamiento corto.
- El **Modelo 2** ofrece un buen rendimiento, es más rápido que el **Modelo 1**, pero no alcanza la precisión del **Modelo 3**.
- El **Modelo 1** tiene una buena precisión, pero es menos eficiente en tiempo y no supera a los otros modelos en rendimiento.

Como comentario final podemos decir que los **Modelos con SeparableConv2D** tienden a ser más eficientes, y el **Modelo 3**, con sus hiperparámetros ajustados, sobresale tanto en desempeño como en optimización.

8 Capas Convolucionales Separable Profundas sin Pointwise

Anteriormente, mencionamos que las capas convolucionales separables profundas se descomponen en dos pasos: el **depthwise**, que utiliza filtros diferentes para cada canal de entrada, y el **pointwise**, que combina la salida de cada canal. Ahora, proponemos un caso donde la información contenida en cada canal RGB es más importante que la imagen completa. Un ejemplo de esto son las **imágenes hiperespectrales**: en algunos casos de imágenes científicas, como las hiperespectrales (que capturan más de tres bandas de colores), cada banda (que podría corresponder a un canal RGB) contiene una porción única de información. Sin embargo, los métodos tradicionales de combinación de estos tres canales (RGB) no son suficientes para capturar toda la riqueza de la información contenida en cada banda.

Por lo tanto, queremos probar cómo funciona una red sin la capa **pointwise**, utilizando un conjunto de datos que contenga información importante en sus tres canales, y compararlo con otro modelo que sí contenga la capa **pointwise**. Esto nos permitirá observar cómo el modelo sin **pointwise** podría ser capaz de detectar patrones específicos en los canales, mientras que el modelo con **pointwise** podría ser menos eficiente ya que, al combinar esa información, perdería los detalles de cada canal.

8.1 Intento con set de datos: Indian Pines AVIRIS Hyperspectral Dataset

El conjunto de datos *Indian Pines* fue considerado inicialmente debido a que las imágenes hiperespectrales, con 220 bandas espectrales, ofrecen más información que las imágenes RGB tradicionales. Esto permite capturar detalles precisos sobre la superficie terrestre, lo que es crucial para clasificar materiales, vegetación, minerales y otros elementos. Nuestra hipótesis era probar si un modelo sin la capa *pointwise* podría aprender a manejar cada banda de forma independiente, sin combinar la información de los canales, lo que podría ser beneficioso para mantener la especificidad de cada canal.

Sin embargo, al considerar la alta dimensionalidad del conjunto (220 bandas), nos dimos cuenta de que el modelo podría tener dificultades para aprender representaciones útiles ya que la cantidad de datos necesarios para entrenar un modelo adecuado era demasiado grande para los recursos disponibles. Además, la falta de experiencia con este tipo de datos y el hecho de que solo se disponía de tres imágenes limitó aún más la viabilidad del uso de este conjunto. Estos factores nos llevaron a abandonar la idea de utilizar este conjunto de datos para nuestra hipótesis.

8.2 Hipótesis sobre la no combinación de canales en redes neuronales

Hipótesis:

Supongamos que tenemos un conjunto de datos en el cual cada canal contiene información que se pierde al combinar la imagen. Si entrenamos un modelo utilizando los canales de la imagen *por separado*, sin emplear el enfoque **pointwise**, se obtendría un mejor desempeño en comparación con entrenar el modelo utilizando **pointwise**, el cual combina la información de los canales de manera conjunta.

Planteamiento:

1. **Escenario:**
    - Imaginemos una imagen de un animal que se camuflajea en su entorno. Supongamos que el animal es visible en el canal azul, pero se pierde al combinar

los tres canales. La clave para reconocer al animal radica en no combinar estos tres canales, ya que la información relevante está distribuida en solo uno de ellos.

- Al no usar **pointwise** y tratar cada canal por separado, el modelo podrá clasificar correctamente que hay un animal en la imagen.
- Por otro lado, al usar **pointwise**, el modelo puede combinar las características de los tres canales en cada punto, lo que hace que el animal se vuelva invisible para el modelo.

2. **Contradicción:**

   - Si no hay información relevante en los canales y entrenamos el modelo de esta manera, el tiempo de entrenamiento y los parámetros de la red no contribuirán a mejorar el rendimiento. En lugar de optimizar el desempeño, el modelo perdería tiempo procesando los canales por separado sin aprovechar la información global de la imagen, lo cual afectaría negativamente la precisión durante la evaluación.

3. **Conclusión:**

   - Es decir, si usamos una red convolucional profunda separable de la misma manera que antes, pero sin aplicar el enfoque **pointwise**, el rendimiento del modelo será inferior al obtenido cuando se aprovecha la combinación de los canales. Esto nos ayuda a apoyar nuestra hipótesis de que el uso de **pointwise** es más efectivo cuando la imagen no contiene información relevante en cada canal. Sin embargo, en el caso de que cada canal tenga información relevante, no utilizar **pointwise** podría ser una opción válida.

Comentario adicional:

Es importante destacar que, en algunos casos, el conjunto de datos podría contener información relevante incluso cuando los canales se procesan por separado. Aunque en ciertos escenarios el modelo podría clasificar correctamente, el rendimiento general será más lento y menos preciso. Por lo tanto, se espera que el uso de **pointwise** para combinar los canales de manera adecuada permita aprovechar mejor la información complementaria de cada canal y, en última instancia, mejorar la precisión y el rendimiento global del modelo.

8.3 El Modelo sin Pointwise

Actualización:

El problema aquí surge porque, como mencionamos anteriormente, las capas de este tipo de convolución, tales como `keras.layers.SeparableConv2D` o `tf.layers.separable_conv2d`, están compuestas por dos partes: la convolución *depthwise* y la convolución *pointwise*. La convolución *depthwise* aplica un filtro de convolución a cada canal por separado, mientras que la convolución *pointwise* combina los resultados de estos filtros utilizando un filtro de convolución 1x1.
El objetivo es eliminar la parte *pointwise*. Tras investigar, descubrí que la única solución sería recurrir nuevamente a las convoluciones normales, es decir, utilizando `Conv2D`, donde el filtro de convolución se aplica a todos los canales simultáneamente, similar a lo que hace la convolución *depthwise*. Este enfoque permite realizar una convolución en profundidad, donde los filtros afectan a todos los canales de manera conjunta.

Sin embargo la idea de eliminar el pointwise en una red convolucional profunda tiene como ob-

jetivo evitar la fusión de los canales de la imagen. Inestigando, este enfoque no es común ni ideal, debido a que las redes convolucionales están diseñadas para detectar patrones de forma eficiente, capturando interacciones entre los canales de entrada, especialmente en tareas de visión por computadora.

El Rol del Pointwise permite que la red capture interacciones entre los canales, lo cual es esencial para tareas complejas de visión por computadora, donde los patrones de los canales se combinan de manera efectiva.

Si eliminamos el pointwise, cada canal se procesa de manera independiente sin combinarse explícitamente con los demás. Esto podría hacer que la red sea más eficiente computacionalmente, pero pierde la capacidad de aprender interacciones entre los diferentes canales de entrada.

¿Qué pasa con las Imágenes Hiperespectrales? El caso de las imágenes hiperespectrales agrega complejidad, ya que estas imágenes tienen muchos más canales que los tres típicos en el modelo RGB. En este caso, la falta de interacción entre canales podría ser aún más perjudicial, ya que las imágenes hiperespectrales contienen información valiosa en cada canal y aún así las interacciones entre estos canales suelen ser cruciales para tareas de clasificación o detección.

Para obtener los mejores resultados en tareas prácticas, es preferible mantener la interacción entre los canales, como se hace con las convoluciones separables profundas o las CNNs 3D, que integran el pointwise de alguna manera.

Entonces si bien es posible procesar los canales por separado (dividiendo una imagen en su componente y entrenando la red solo con componentes), esto no es lo más recomendable para la mayoría de las aplicaciones de visión por computadora. En el caso de las imágenes hiperespectrales, el uso de técnicas como las convoluciones separables profundas o las CNNs 3D sigue siendo una opción más efectiva, ya que permiten preservar las interacciones entre los canales mientras aprovechan la eficiencia computacional.