DuocUC | ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

INFORME DE PROYECTO
FORMA C
"ETIQUETADO DE IMÁGENES PARA GOBIERNO DE CHILE"

PROFESOR: MARCO ANTONIO JAPKE ADRIASOLA
DEEP LEARNING
DLY0100_003V

Integrantes:

Cristóbal Cabezas
Jorge López soto

SANTIAGO, 9 DE JUNIO DE 2024

Índice

Índice 2
Resumen ejecutivo 3
Descripción del problema a resolver 4
Descripción técnica de la solución: 4
Fundamentación técnica de los modelos escogidos 6
Detalle de ajustes en las redes 7
Visualizaciones: 7
Justificación de la solución 19
Conclusiones respecto del trabajo realizado 19
Propuesta de mejora al proyecto utilizando arquitecturas especializadas para una versión mejorada 20
4 ANEXO: Implementación Red Convolucional 28
4.1. Introducción de Implementación Red Convolucional 28
4.1. Red Convolucional: Conv2D y MaxPool2D 28
4.1.1. Red Convolucional Base 28
4.1.2. Red Convolucional Mejorado 29
4.2. Red Convolucional: ResNet50 30
4.2.1. ResNet50 Base 30
4.2.2. ResNet50 Mejorado 31
4.3. Red Convolucional: Transfer Learning 32
4.3.1. VGG16 Base 32
4.3.2. VGG16 Mejorado 33
4.4. Evaluación 33
4.4.1. Resultados Red Convolucional 34
4.4.2. Resultados ResNet50 34
4.4.3. Resultados VGG16 34
4.5. Conclusiones 34

Resumen ejecutivo

Este proyecto se enfoca en el desarrollo de un sistema de etiquetado de imágenes utilizando técnicas de Deep Learning, específicamente el conjunto de datos CIFAR-10. El objetivo principal es construir un modelo de aprendizaje automático capaz de clasificar imágenes en 10 clases distintas con alta precisión.

CIFAR-10 es un conjunto de datos que consta de 60,000 imágenes en color de tamaño 32x32, distribuidas en 10 categorías diferentes, con 6,000 imágenes por clase. Estas imágenes se dividen en 50,000 para entrenamiento y 10,000 para pruebas.

El gobierno de Chile busca implementar inteligencia artificial en su página web para etiquetar imágenes de diversos dominios. La necesidad de este proyecto radica en la creciente importancia de automatizar tareas que involucran grandes volúmenes de datos, en particular imágenes, y en la capacidad de las máquinas para tomar decisiones con mayor exactitud y escalabilidad que los humanos.

Para abordar este desafío, se utilizará la metodología CRISP-DM (Cross-Industry Standard Process for Data Mining), que consta de seis fases: comprensión del negocio, comprensión de los datos, preparación de los datos, modelado, evaluación y despliegue.

El éxito de este proyecto no solo radica en la creación de un modelo de clasificación de imágenes preciso, sino también en la implementación de un sistema escalable y eficiente que pueda manejar el etiquetado de imágenes en tiempo real para satisfacer las necesidades del gobierno de Chile en su página web.

Descripción del problema a resolver

El Gobierno de Chile busca implementar inteligencia artificial en su página web para automatizar el etiquetado de imágenes de diversos dominios. Actualmente, la clasificación manual de imágenes es un proceso lento y costoso, que impide una gestión eficiente de grandes volúmenes de datos visuales.

El problema es la necesidad de clasificar imágenes de manera precisa y eficiente en 10 categorías diferentes, crucial para mejorar la experiencia del usuario y optimizar la navegación en la página web gubernamental. Esto implica:

Clasificación precisa de imágenes: El sistema debe reconocer con precisión los objetos o elementos de las imágenes, asignándoles una de las 10 clases predefinidas.

Eficiencia y escalabilidad: El sistema debe ser capaz de manejar grandes volúmenes de imágenes de manera eficiente y escalable, sin comprometer la velocidad de procesamiento.

Adaptabilidad a diversos dominios: Las imágenes pueden provenir de diferentes dominios, como naturaleza, personas, objetos, etc. El sistema debe ser capaz de adaptarse y clasificar imágenes de cualquier dominio con precisión.

Automatización del proceso: La clasificación manual de imágenes es laboriosa y propensa a errores. El sistema debe automatizar este proceso para reducir el tiempo y los costos asociados con la clasificación manual.

Desarrollo de un modelo robusto: Se requiere la construcción de un modelo de aprendizaje profundo robusto y generalizable que pueda manejar la complejidad de las imágenes y generalizar patrones a partir de un conjunto de datos de entrenamiento limitado.

El desafío es desarrollar un sistema de etiquetado de imágenes basado en inteligencia artificial preciso, eficiente, escalable y adaptable a diferentes dominios, para mejorar la gestión de datos visuales en la página web del Gobierno de Chile.

Descripción técnica de la solución:

Para abordar el problema planteado, se propone una solución basada en técnicas de Deep Learning, utilizando el conjunto de datos CIFAR-10 y siguiendo la metodología CRISP-DM.

1. Comprensión del Negocio:

    1.1. Identificar las necesidades y objetivos del Gobierno de Chile en cuanto al etiquetado de imágenes en su página web.
    1.2. Definir las clases de imágenes relevantes para la clasificación.

1.3. Establecer métricas de evaluación, como precisión, recall y F1-score.

2. Comprensión de los Datos:

   2.1. Explorar el conjunto de datos CIFAR-10 para comprender su estructura y características.
   2.2. Visualizar ejemplos de imágenes de cada clase para entender la complejidad y variabilidad de los datos.
   2.3. Realizar un análisis estadístico de los datos para identificar posibles desafíos, como desequilibrios de clase.

3. Preparación de los Datos:

   3.1. Normalizar las imágenes para asegurar que tengan la misma escala y rango de valores.
   3.2. Dividir el conjunto de datos en conjuntos de entrenamiento, validación y prueba.
   3.3. Aplicar técnicas de aumento de datos, como rotaciones, desplazamientos y reflejos, para aumentar la variabilidad del conjunto de datos de entrenamiento.

4. Modelado:

   4.1. Seleccionar una arquitectura de red neuronal convolucional (CNN) adecuada para el problema, como ResNet, VGG o DenseNet.
   4.2. Diseñar y entrenar el modelo utilizando el conjunto de datos de entrenamiento y validación.
   4.3. Utilizar técnicas como transferencia de aprendizaje para inicializar los pesos del modelo con una red preentrenada en un conjunto de datos similar.
   4.4. Ajustar los hiperparámetros del modelo, como la tasa de aprendizaje, el tamaño del lote y la función de pérdida, mediante validación cruzada.

5. Evaluación:

   5.1. Evaluar el rendimiento del modelo utilizando el conjunto de datos de prueba.
   5.2. Analizar las métricas de evaluación para comprender la precisión y el rendimiento del modelo en la clasificación de imágenes.
   5.3. Realizar pruebas adicionales con imágenes nuevas para evaluar la capacidad de generalización del modelo.

6. Despliegue:

   6.1. Implementar el modelo entrenado en un entorno de producción, preferiblemente en la infraestructura en la nube.
   6.2. Configurar una interfaz de usuario amigable para que los usuarios puedan cargar imágenes y recibir las etiquetas predichas.

6.3. Realizar pruebas exhaustivas del sistema desplegado para garantizar su funcionamiento correcto y eficiente en tiempo real.

La solución propuesta busca desarrollar un sistema de etiquetado de imágenes preciso y eficiente, capaz de automatizar el proceso de clasificación de imágenes en la página web del Gobierno de Chile, mejorando así la gestión de datos visuales y la experiencia del usuario.

Fundamentación técnica de los modelos escogidos

Para el desarrollo del presente trabajo, se utilizará el siguiente flujo:

1) Se utilizarán 3 optimizadores:
    a. SGD: El optimizador SGD (Stochastic Gradient Descent) es un algoritmo de optimización utilizado en el entrenamiento de modelos de aprendizaje profundo. Funciona actualizando iterativamente los pesos de la red neuronal en la dirección opuesta al gradiente de la función de pérdida con respecto a los pesos. El gradiente se calcula utilizando un subconjunto aleatorio de ejemplos de entrenamiento en cada iteración, lo que lo hace "estocástico". SGD utiliza una tasa de aprendizaje para controlar el tamaño de los pasos de actualización de los pesos. Es uno de los optimizadores más utilizados debido a su simplicidad y eficacia.
    b. Adam: (Adaptive Moment Estimation) es un algoritmo de optimización que combina los conceptos de Momentum y RMSprop. Es un optimizador popular y eficiente para entrenar redes neuronales.
    c. RMSProp: Adam también utiliza un término similar al RMSprop, que adapta la tasa de aprendizaje de manera individual para cada parámetro en función de la magnitud de sus gradientes. Esto ayuda a controlar la velocidad de aprendizaje para cada parámetro de manera más adaptativa, lo que puede ser beneficioso en problemas con gradientes de magnitudes variables.
2) A cada de uno de estos optimizadores se les aplicarán 3 funciones de activación:
    a. ReLu Rectified Linear Unit): ReLU es una función de activación no lineal que asigna cualquier valor negativo a cero y deja los valores positivos intactos. Es una de las funciones de activación más utilizadas debido a su eficiencia computacional y a su capacidad para mitigar el problema de desvanecimiento del gradiente. ReLU permite que la red neuronal aprenda de manera más rápida y efectiva, especialmente en problemas de clasificación y regresión.
    b. Tanh (Tangente Hiperbólica): Tanh es una función de activación no lineal que asigna valores reales a un rango entre -1 y 1. A diferencia de ReLU, Tanh produce salidas centradas en cero, lo que puede ayudar a mitigar el problema del desvanecimiento del gradiente en redes neuronales profundas. Tanh es útil en capas ocultas para capturar relaciones no lineales y es comúnmente utilizada en arquitecturas de redes neuronales.

c. Sigmoid (Función Sigmoide): La función de activación Sigmoid es una función no lineal que transforma los valores de entrada a un rango entre 0 y 1. Se define como f(x) = 1 / (1 + exp(-x)). La salida de la función Sigmoid se interpreta como una probabilidad, donde valores cercanos a 1 indican alta probabilidad y valores cercanos a 0 indican baja probabilidad. La función Sigmoid es comúnmente utilizada en capas ocultas de redes neuronales para introducir no linealidades y en capas de salida para problemas de clasificación binaria.

Detalle de ajustes en las redes

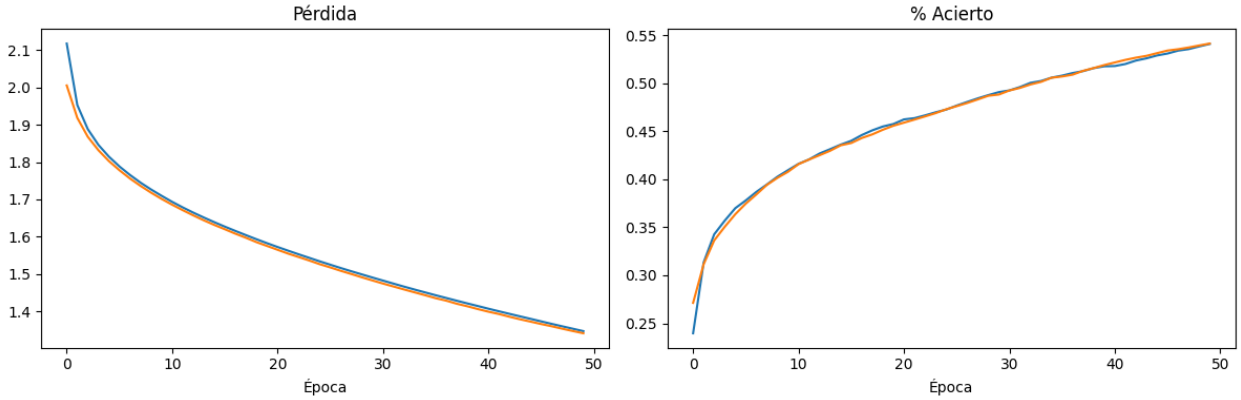
- Cada de unas iteraciones se realizarán utilizando 1, 2 y 3 capas ocultas.
  - La primera capa utilizará 512 neuronas.
  - La segunda, 256.
  - La tercera, 256
- Todos los modelos a entrenar se iterarán en 50 épocas (epochs)
- Salvo los casos claramente señalados, los optimzadores utilizarán sus valores por defecto.
- Asimismo, a todos los modelos se les medirán los resultados de los datos entrenamiento (accuracy, loss), los cuales se contrastarán con los resultados de los datos de prueba (val_accuracy, val_loss).

Visualizaciones:

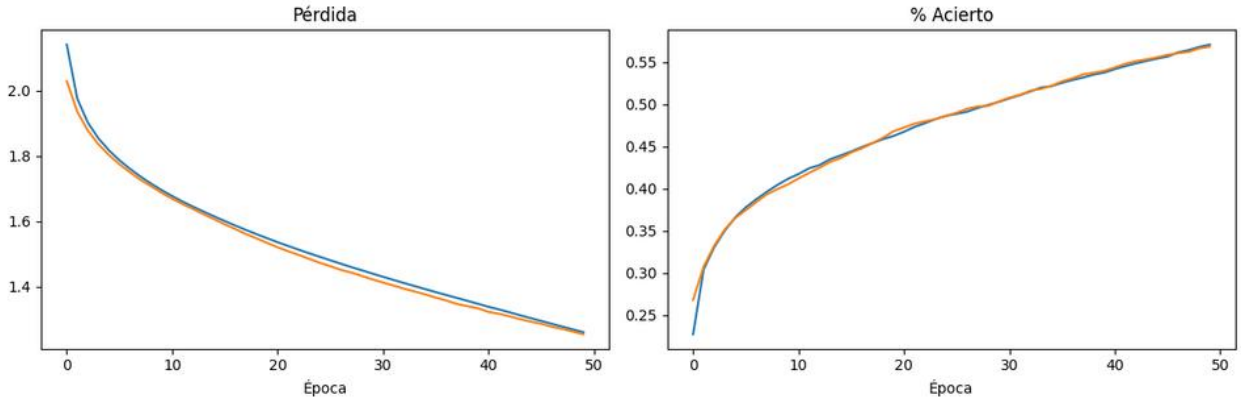
1. SGD

1.1. SGD + ReLu

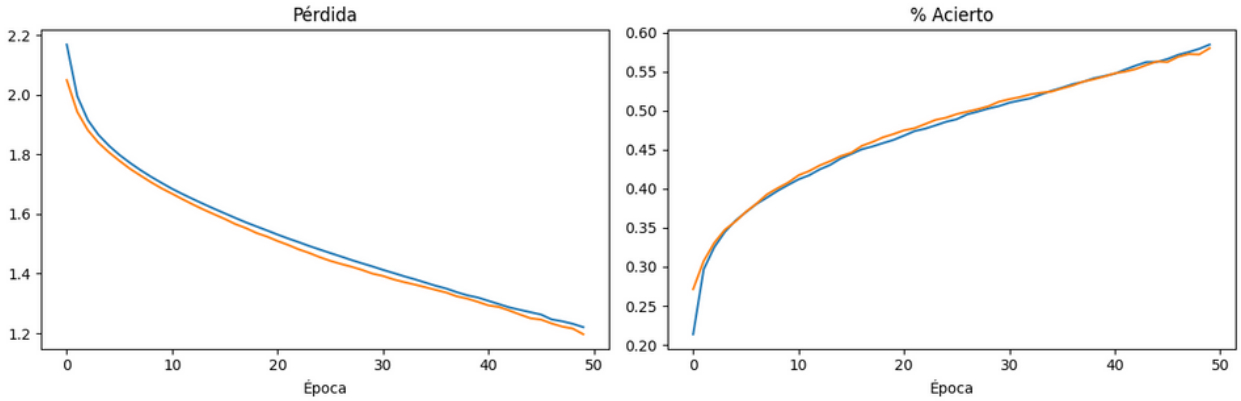
1.1.1. 1 Capa



1.1.2. 2 Capas



1.1.3. 3 Capas

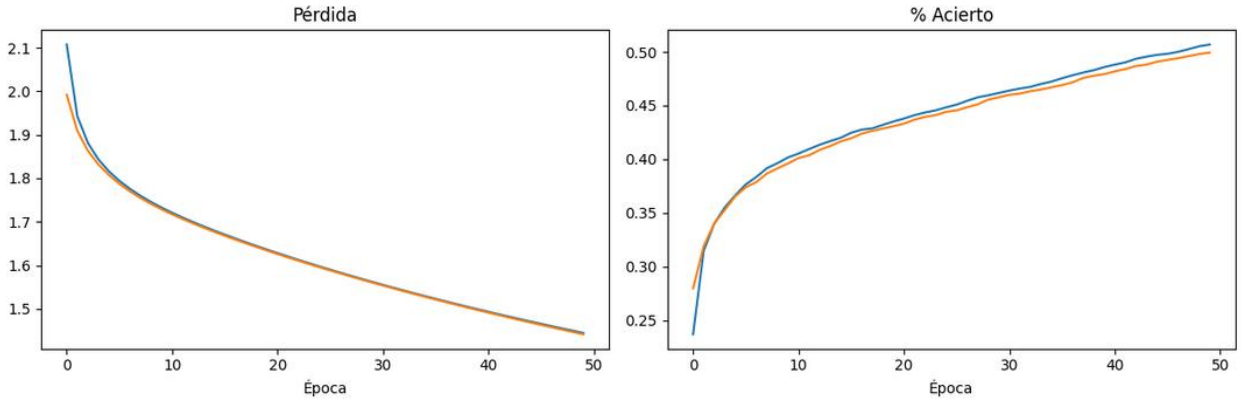


1.1.4. Resultados

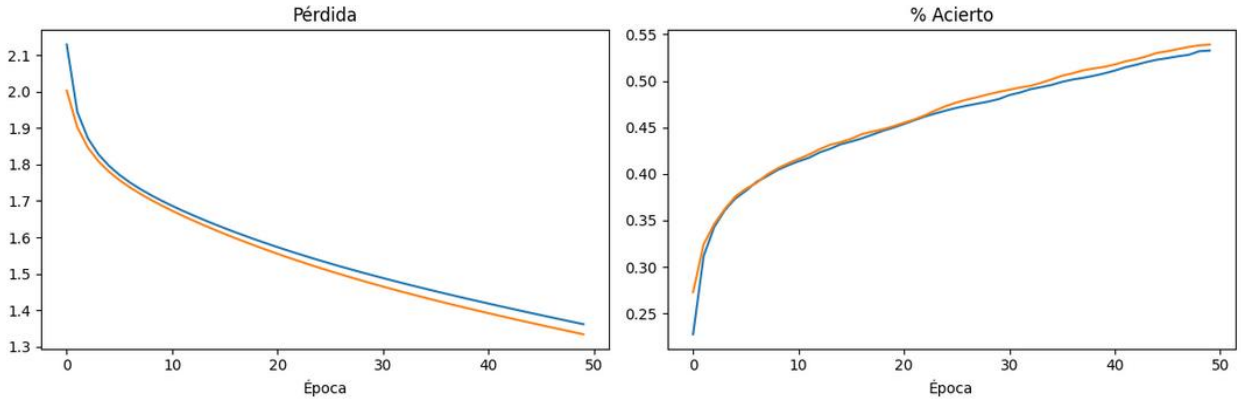
	Relu				
Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*
1	0,5448	1,3521	0,5414	1,3416	82
2	0,5731	1,2609	0,5685	1,2531	60
3	**0,59**	**1,2237**	**0,5802**	**1,1967**	**60**

1.2. SGD + Tanh

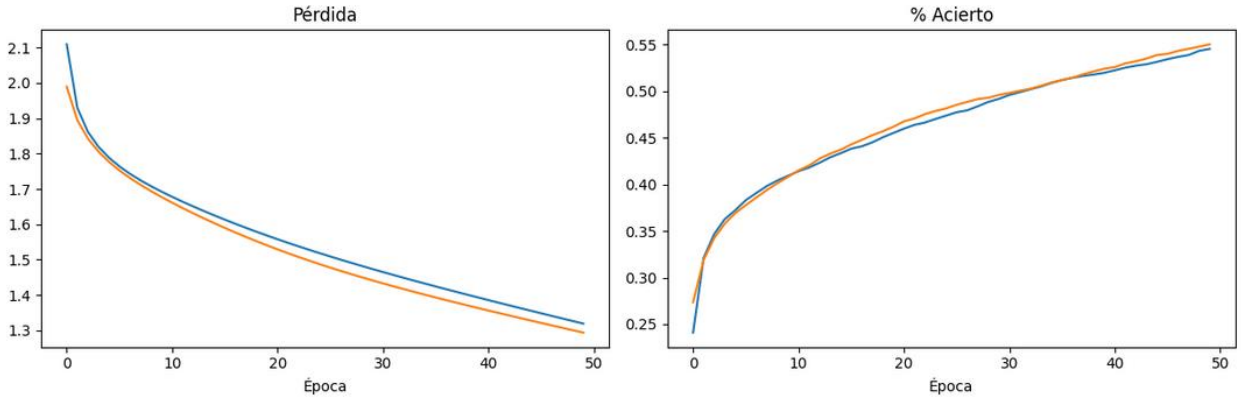
1.2.1. 1 capa



1.2.2. 2 capas



1.2.3. 3 capas



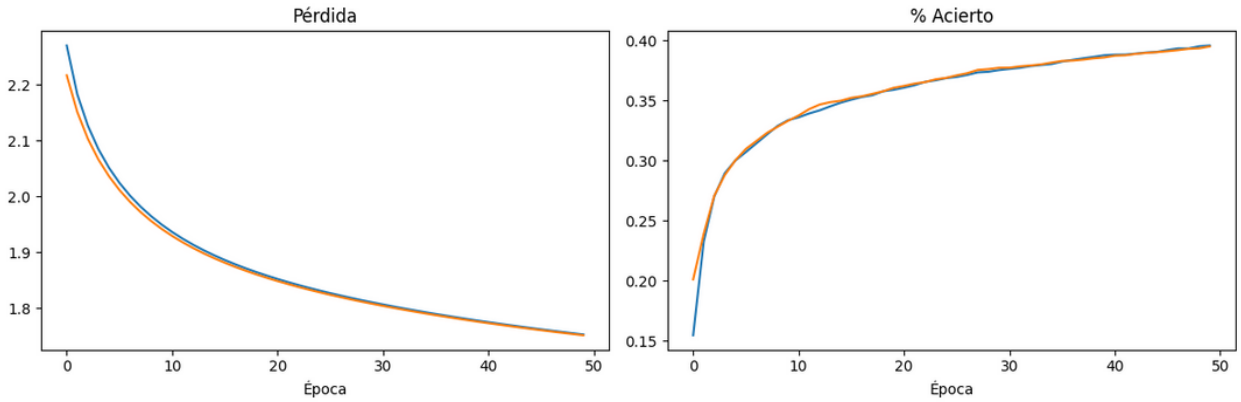
1.2.4. Resultados

	Tanh				
Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*
1	0,5083	1,4454	0,4993	1,4415	62

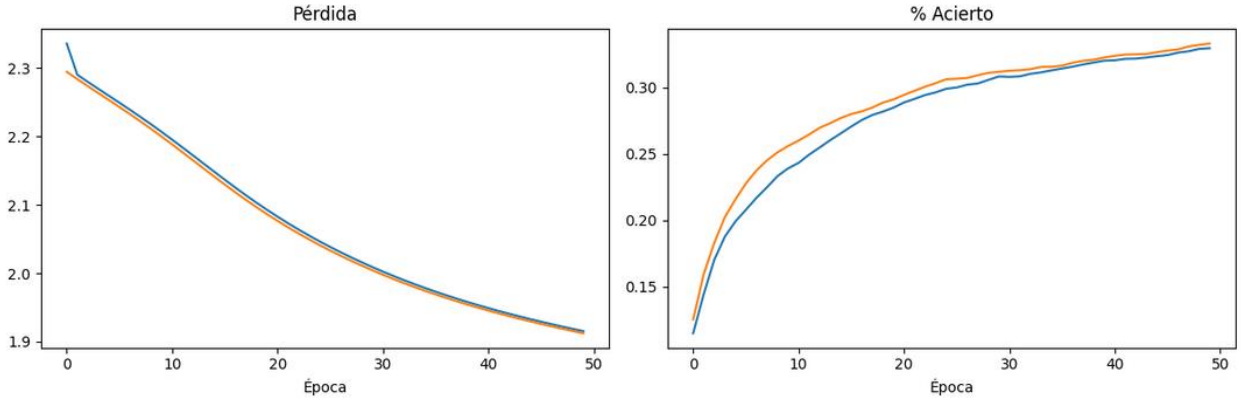
2	0,5334	1,3618	0,5391	1,3339	65
3	**0,5454**	**1,3208**	**0,5503**	**1,2933**	**68**

1.3. SGD + Sigmoid

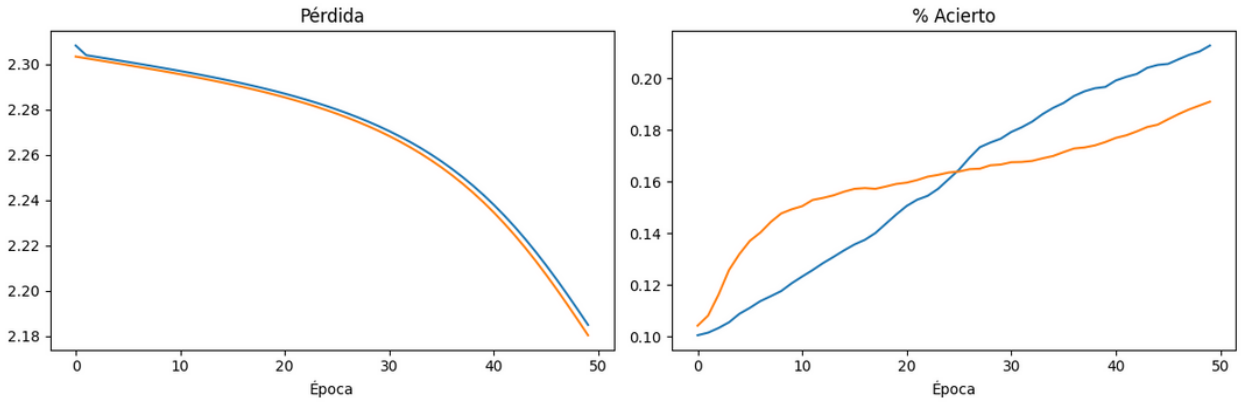
1.3.1. 1 Capa



1.3.2. 2 Capas



1.3.3. 3 Capas



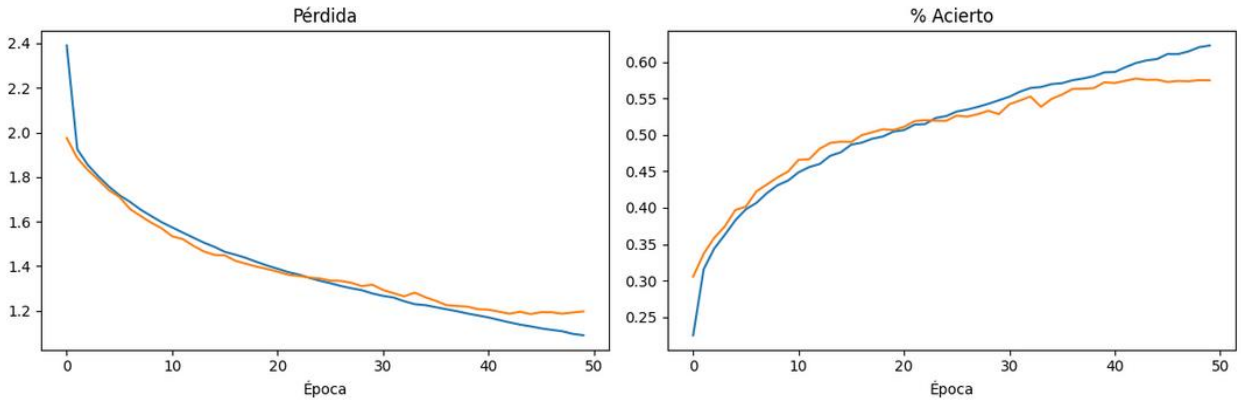
1.3.4. Resultados

	SGD + Sigmoid				
Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*
1	**0,4004**	**1,7536**	**0,3949**	**1,7515**	**59**
2	0,3269	1,9224	0,3330	1,9123	65
3	0,2126	2,1884	0,1909	2,1804	67

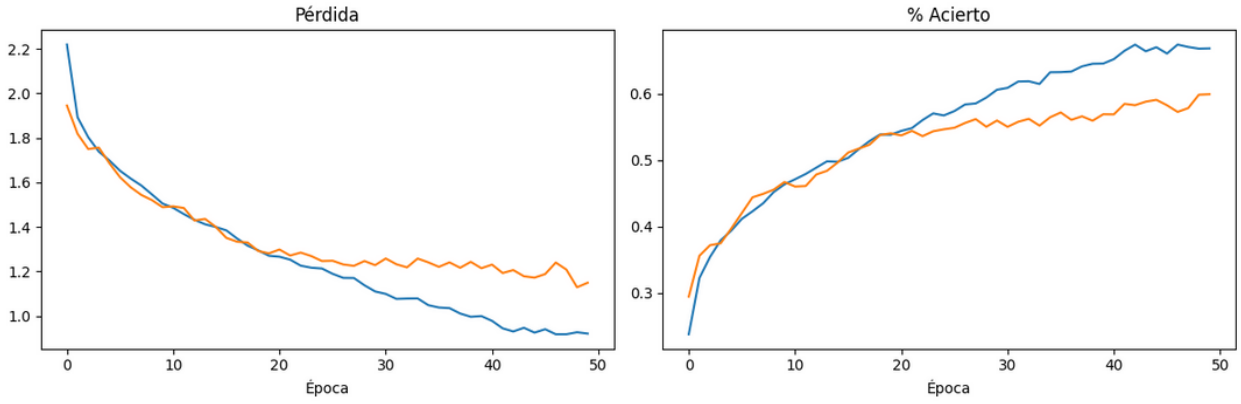
2. ADAM

2.1. ADAM + ReLu

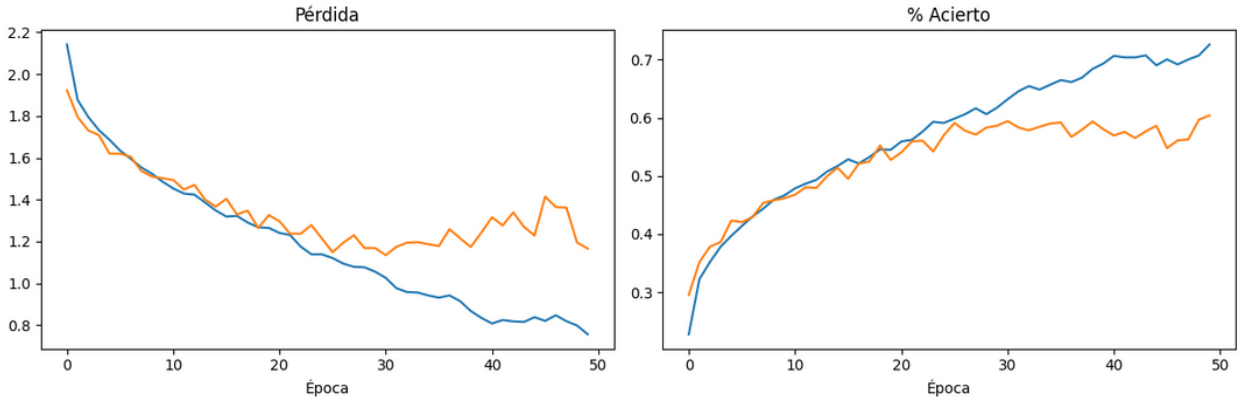
2.1.1. 1 Capa



2.1.2. 2 Capas



2.1.3. 3 Capas

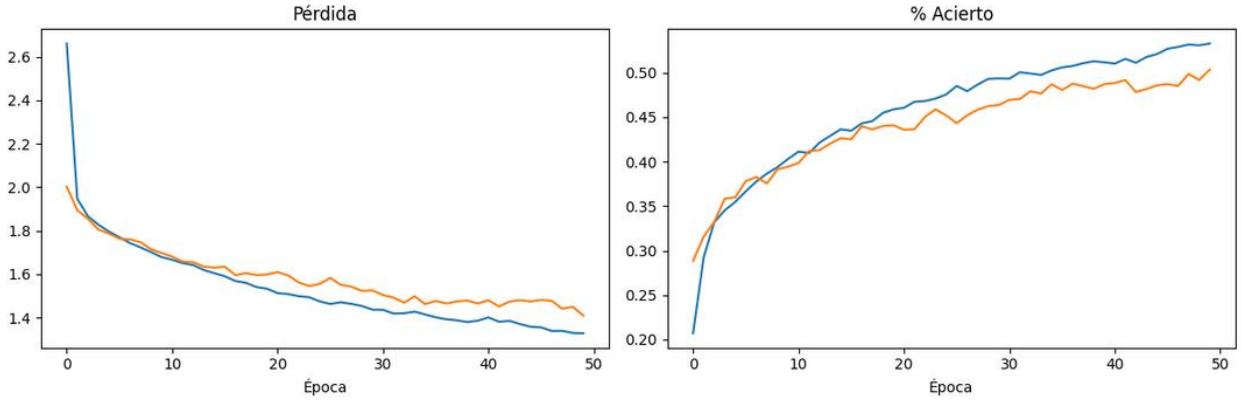


2.1.4. Resultados

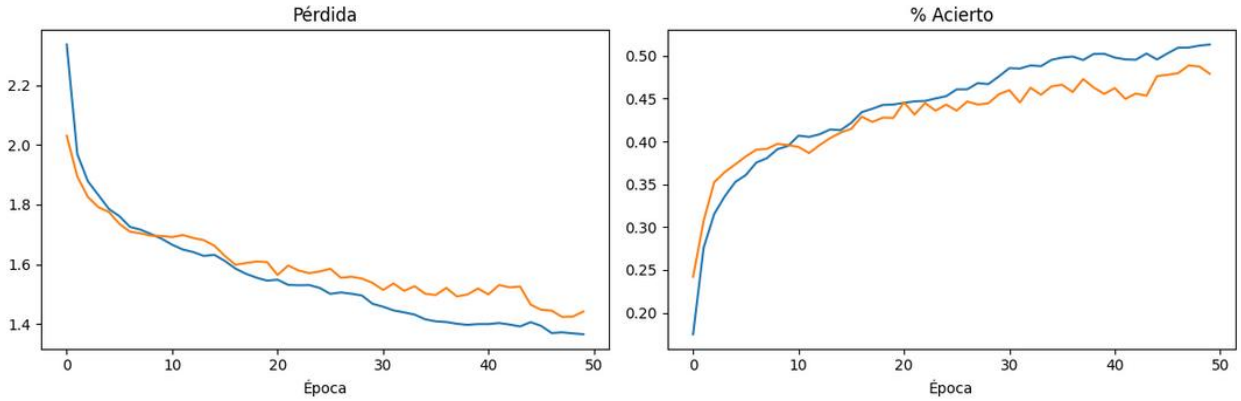
	ADAM + Relu				
Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*
1	0,6154	1,1054	0,5748	1,1962	174
2	0,6713	0,9236	0,5989	1,1495	180
3	**0,7160**	**0,7959**	**0,6039**	**1,1658**	**299**

2.2. ADAM + Tanh

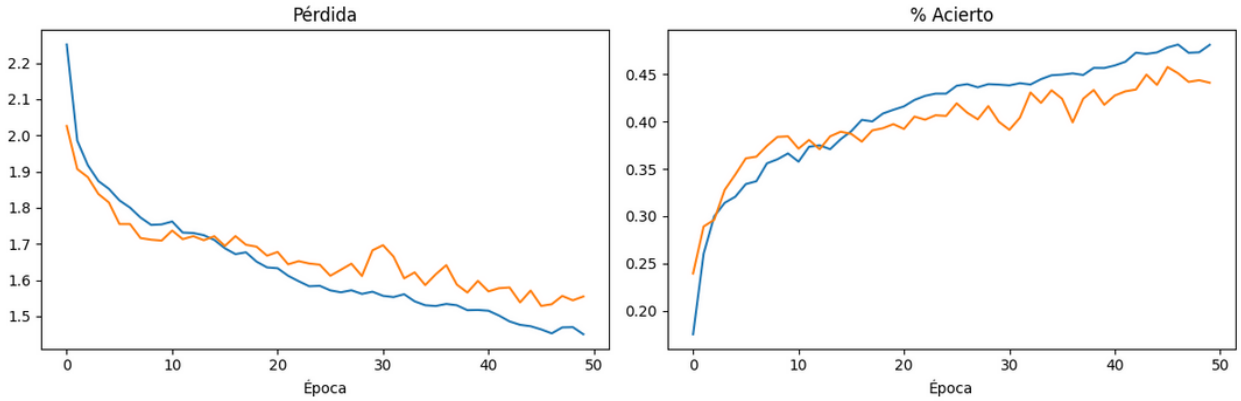
2.2.1. 1 Capa



2.2.2. 2 Capas



2.2.3. 3 Capas

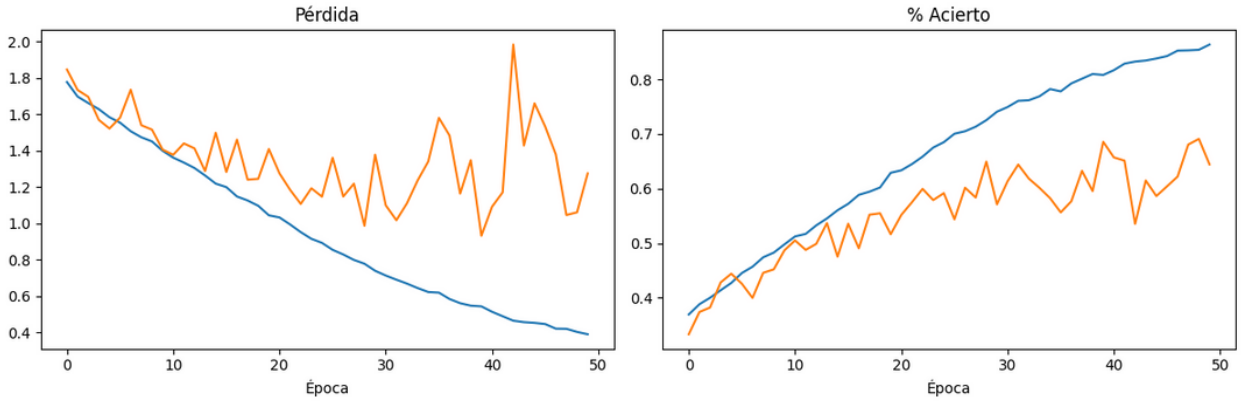


2.2.4. Resultados

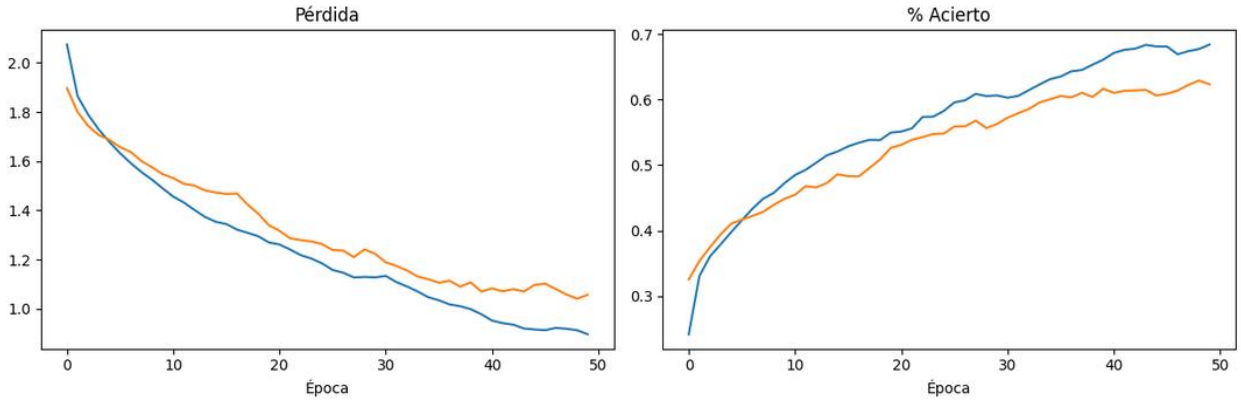
	Adam + Tanh				
Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*
1	**0,5350**	**1,3251**	**0,5034**	**1,4098**	**158**
2	0,5191	1,3531	0,4791	1,4417	183
3	0,4814	1,4582	0,4411	1,5545	193

2.3. ADAM + Sigmoid

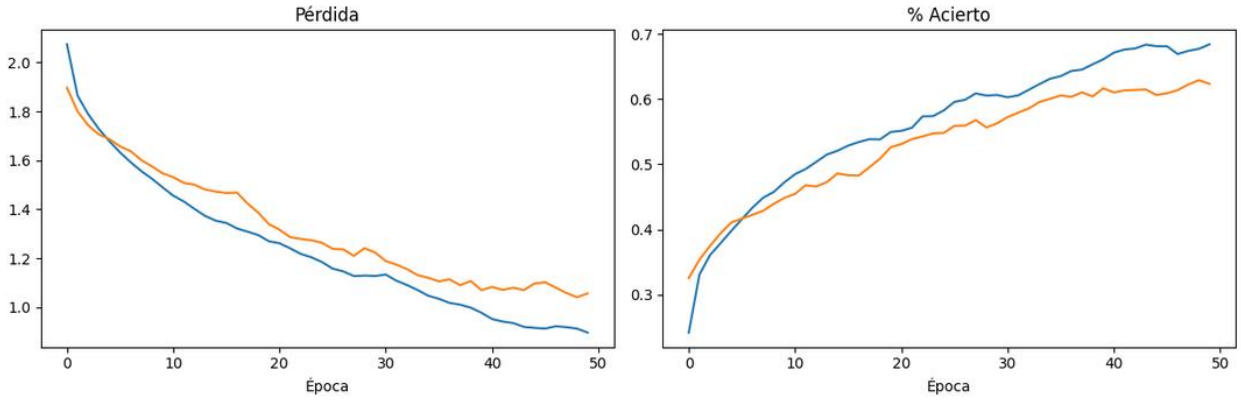
2.3.1. 1 Capas



2.3.2. 2 Capas



2.3.3. 3 Capas



2.3.4. Resultados

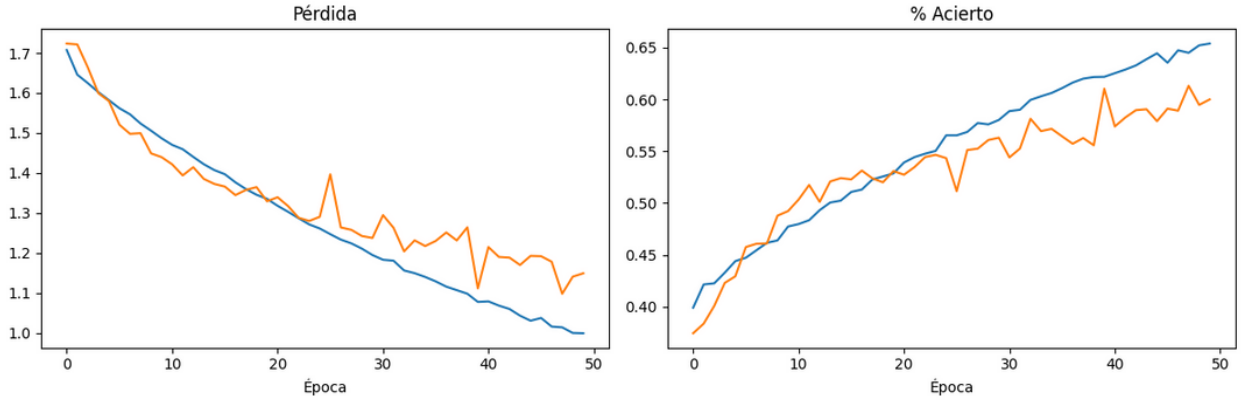
	ADAM + Sigmoid				
Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*

1	**0,7184**	**0,8806**	**0,6015**	**1,1168**	**175**
2	0,6796	0,9099	0,6235	1,0565	149
3	0,6276	1,0324	0,5667	1,2117	139

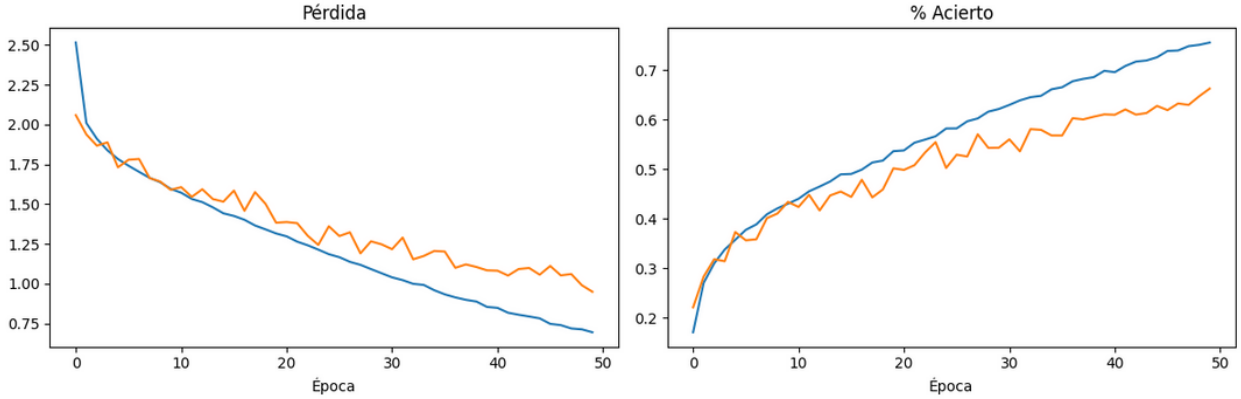
3. RMSProp

3.1. RMSProp + ReLu

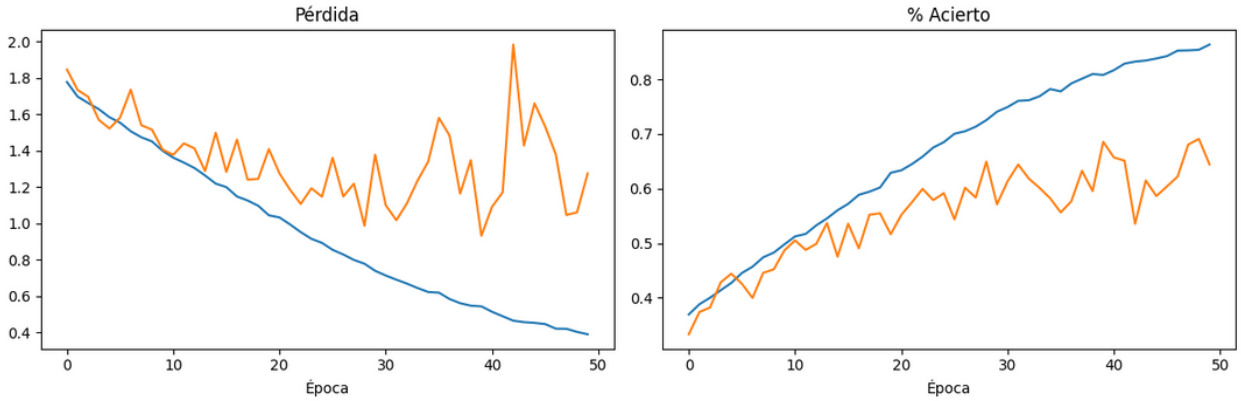
3.1.1. 1 Capa



3.1.2. 2 Capas



3.1.3. 3 Capas

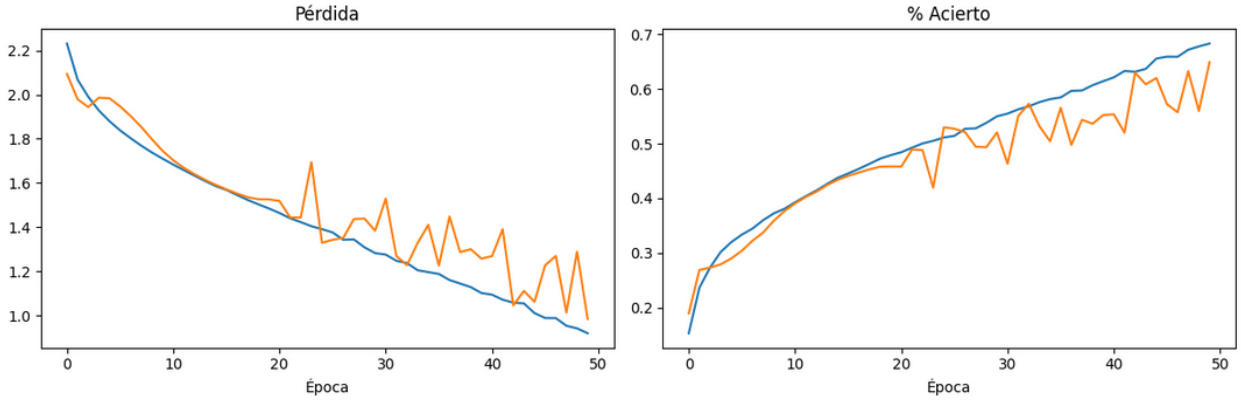


3.1.4. Resultados

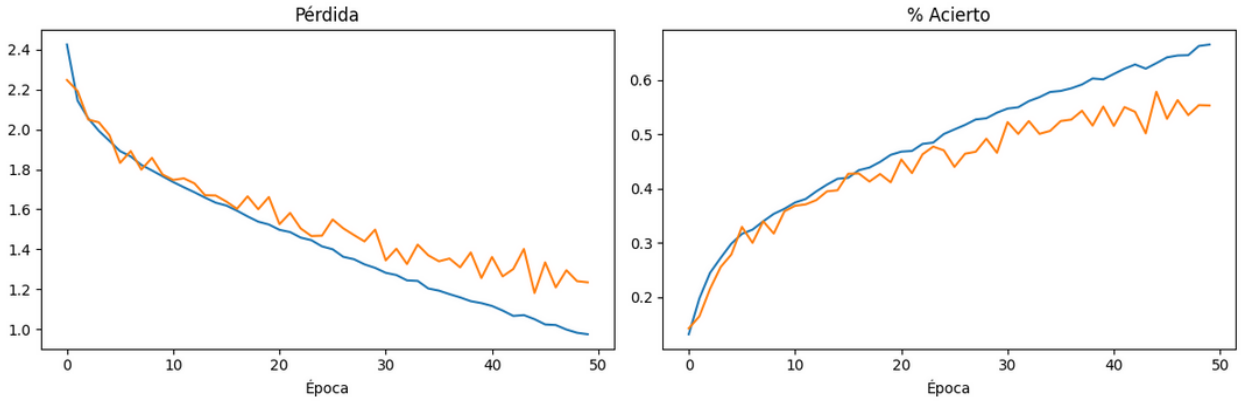
RMSProp + ReLu					
Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*
1	0,6474	1,0088	0,6000	1,1492	119
2	0,7576	0,6916	0,6630	0,9490	123
3	**0,8641**	**0,3901**	**0,6443**	**1,2746**	**132**

3.2. RMSProp + Tanh

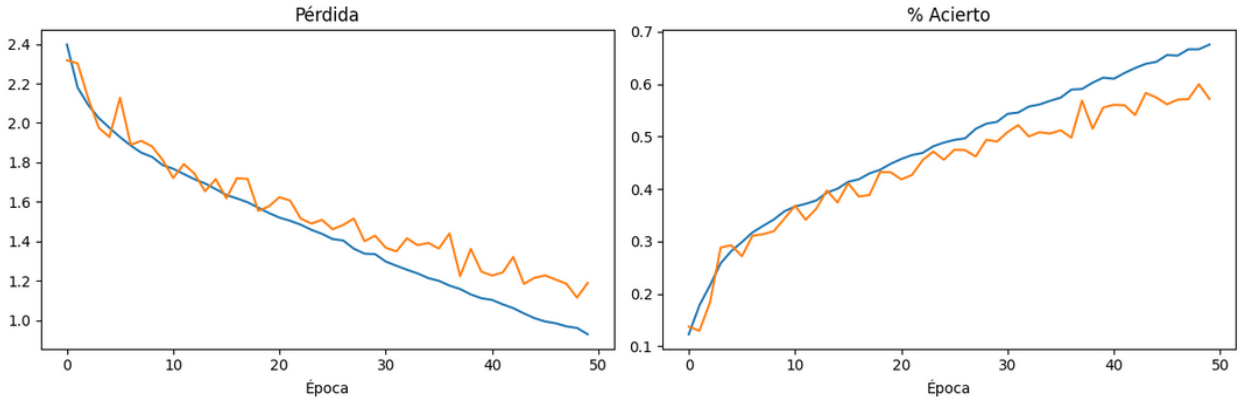
3.2.1. 1 Capa



3.2.2. 2 Capas



3.2.3. 3 Capas

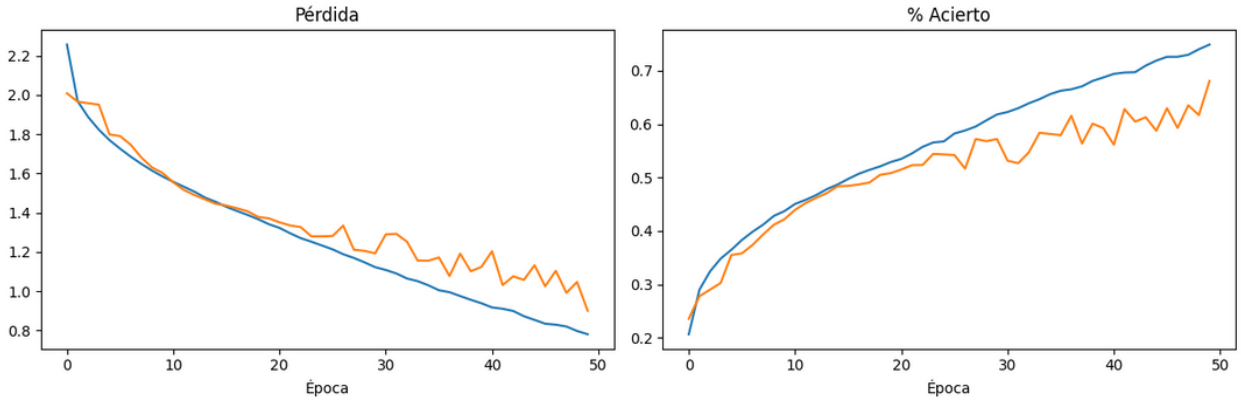


3.2.4. Resultados

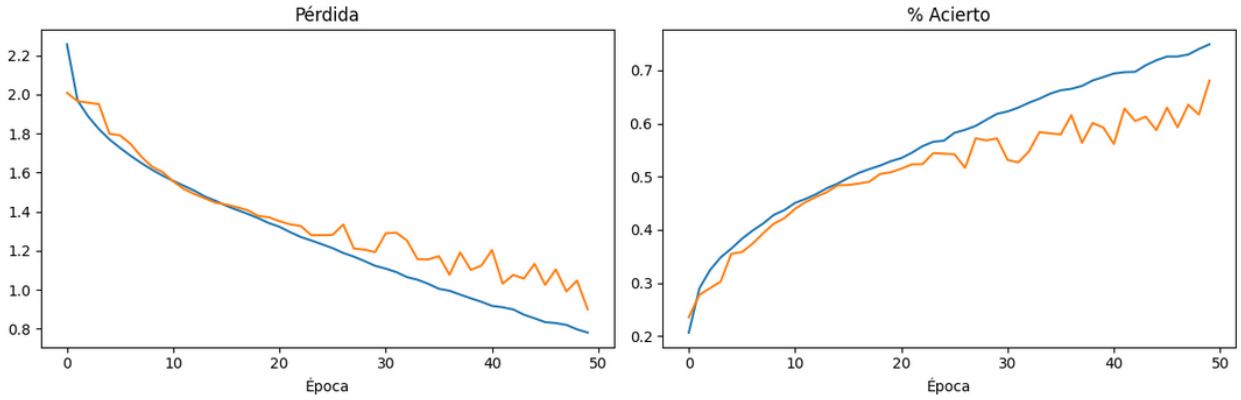
	RMSProp + Tanh				
Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*
1	0,6218	1,108	0,4997	1,4631	117
2	0,66	0,9886	0,5533	1,2350	126
3	**0,6771**	**0,9359**	**0,5717**	**1,1889**	**129**

3.3. RMSProp + Sigmoid

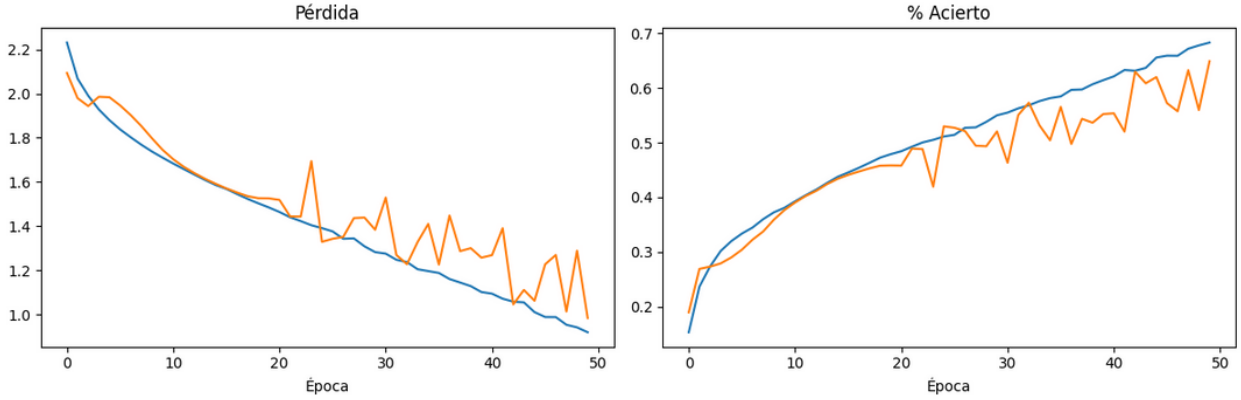
3.3.1. 1 Capa



3.3.2. 2 Capas



3.3.3. 3 Capas



3.3.4. Resultados

	RMSPRop + Sigmoid				
Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*
1	**0,7184**	**0,8806**	**0,6015**	**1,1168**	**175**

2	0,6796	0,9099	0,6235	1,0565	149
3	0,6276	1,0324	0,5667	1,2117	139

Justificación de la solución

Conclusiones respecto del trabajo realizado

Observaciones:

- Más no siempre es mejor: Dependiendo la combinación, rendimiento mejora con menos capas.
- Regularizadores no mejora necesariamente accuracy pero sí el rendimiento de datos de prueba, especialmente con Sigmoid.
- La combinación SGD + ReLu mostró el mejor rendimiento, es decir, la mejor relación de accuracy entre los datos de entrenamiento y datos de prueba.

Premios:

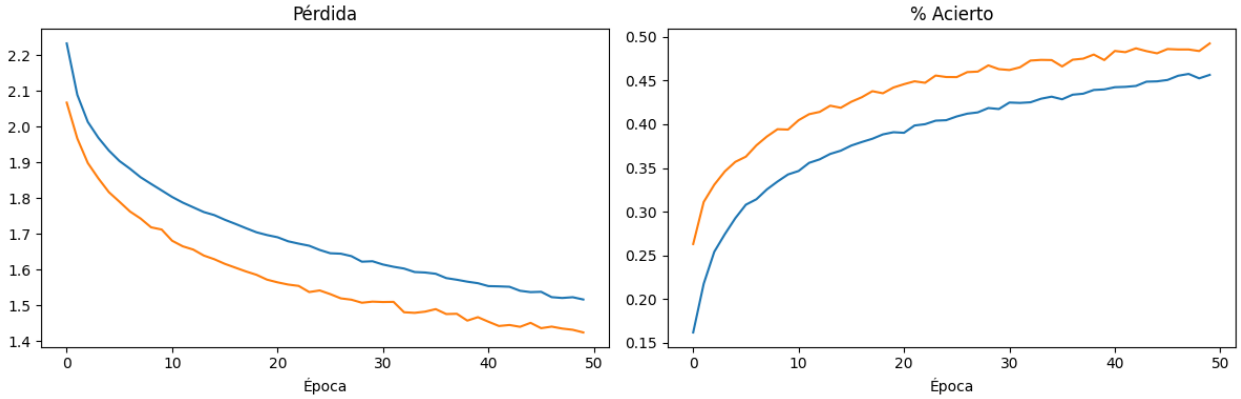
- Mejor optimizador: **SGD**
- Mejor función de activación: **ReLu**
- Mejor regularizador: **Dropout**
- Mejores combinaciones:
  - SGD: ReLu + 3 capas (60% vs 58%)
  - ADAM: ReLu + 3 capas (71% vs 60%)
  - RMSProp: Tanh + 3 capas (35% vs 41%)
  - Dropout: SGD + ReLu + 3 capas (45% vs 49%)
  - EarlyStopping: Adam + Tanh + 1 capa (40% vs 40%)

Propuesta de mejora al proyecto utilizando arquitecturas especializadas para una versión mejorada

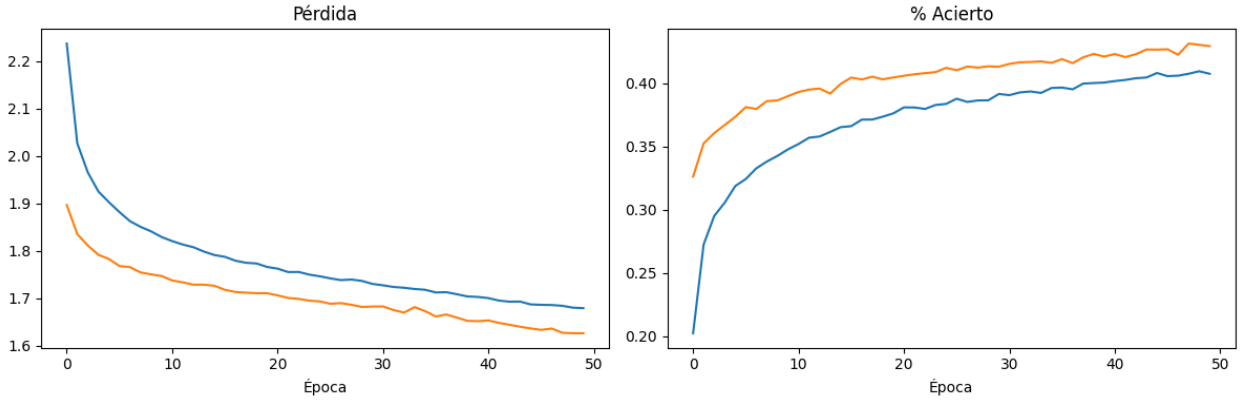
1. Regularización con Dropout

1.1. SGD + Dropout

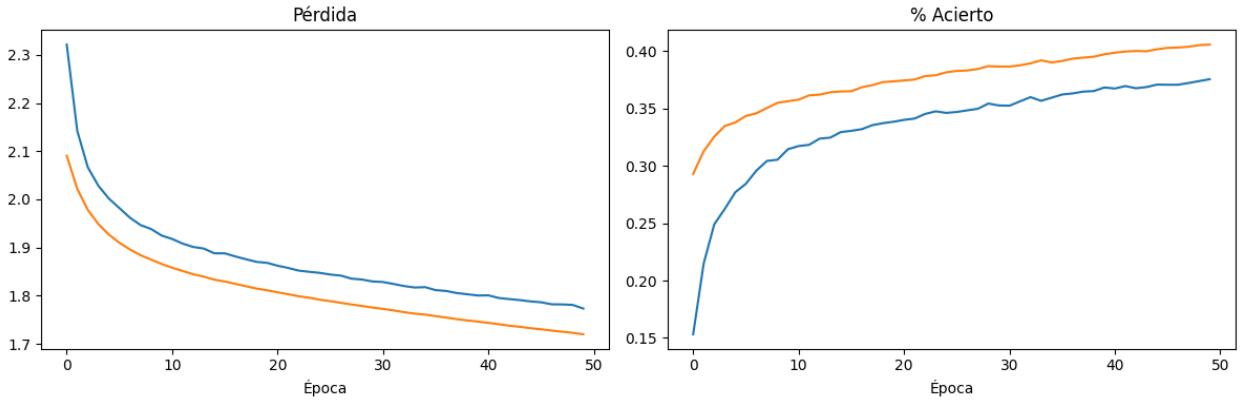
1.1.1. SGD + ReLu + Dropout (3 capas)



1.1.2. SGD + Tanh + Dropout (3 capas)



1.1.3. SGD + Sigmoid (1 capa)

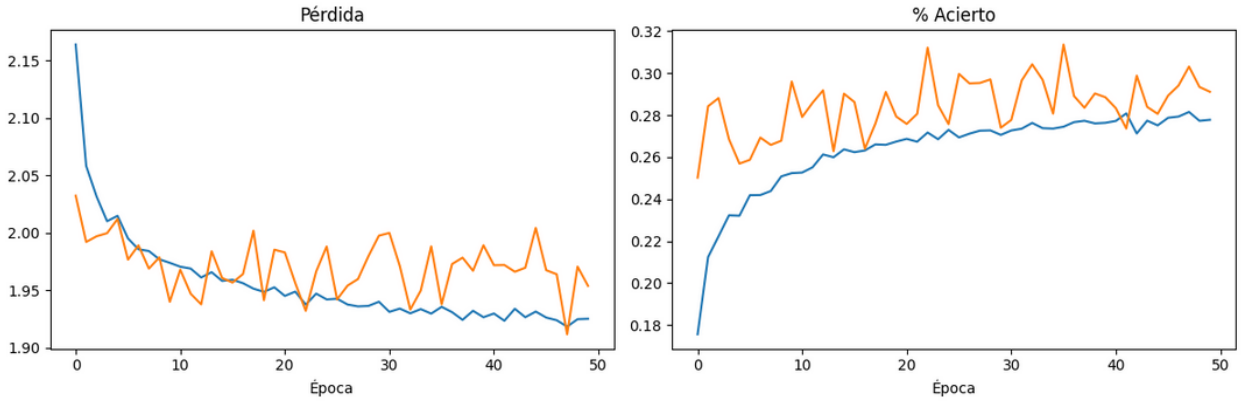


1.1.4. Resultados

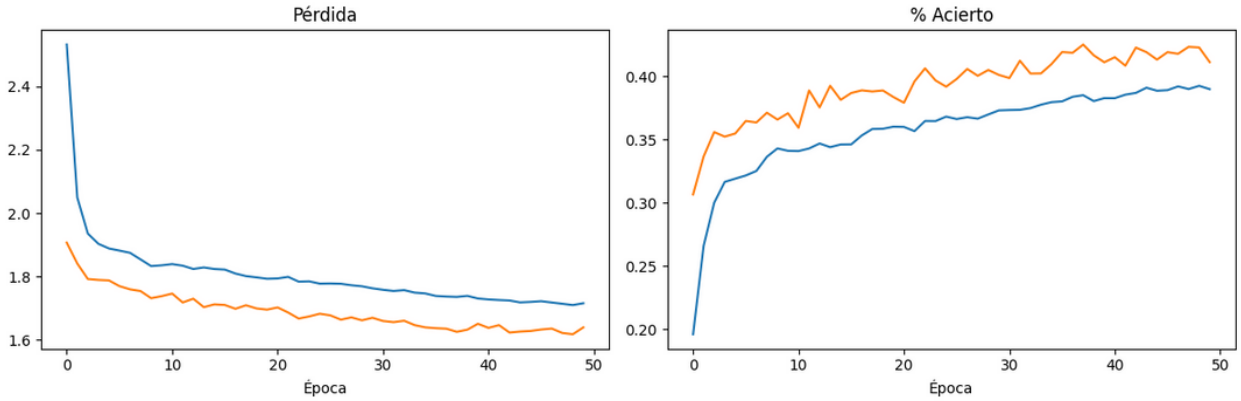
	SGD + Dropout					
	Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*
Relu	3	0,4537	1,5168	0,4924	1,4245	247
Tanh	3	0,4083	1,6801	0,4296	1,6265	244
Sigmoid	1	0,3705	1,7757	0,4058	1,7198	189

1.2. ADAM + Dropout

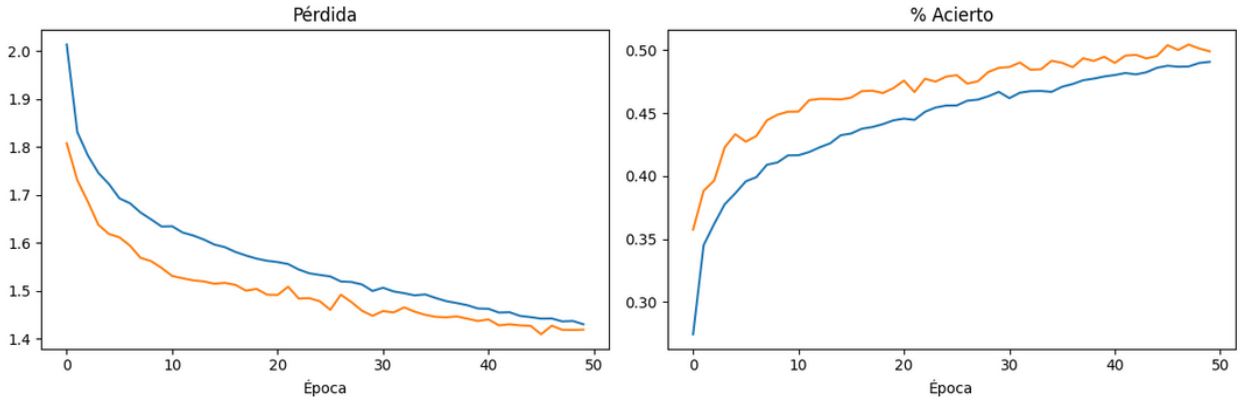
1.2.1. ADAM + ReLu + Dropout (3 capas)



1.2.2. ADAM + Tanh + Dropout (1 capa)



1.2.3. ADAM + Sigmoid + Dropout (1 capa)

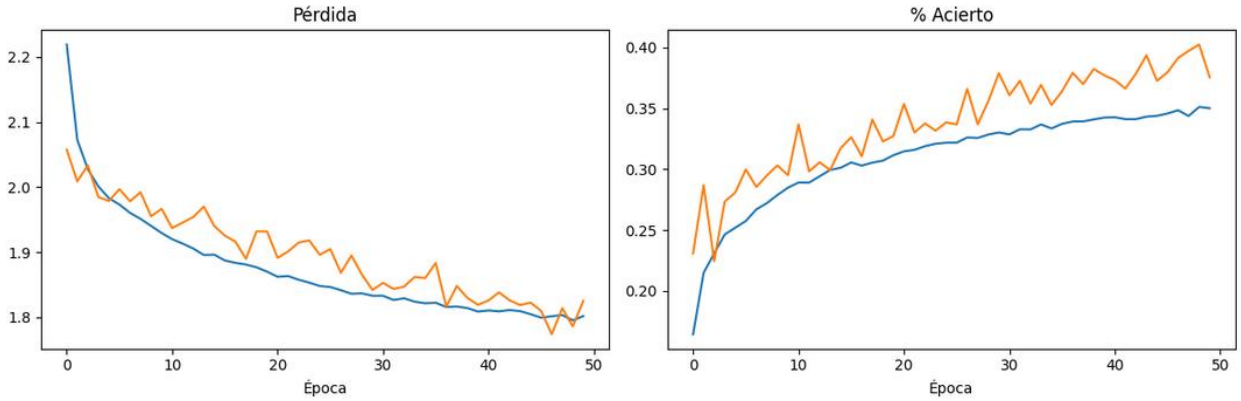


1.2.4. Resultados

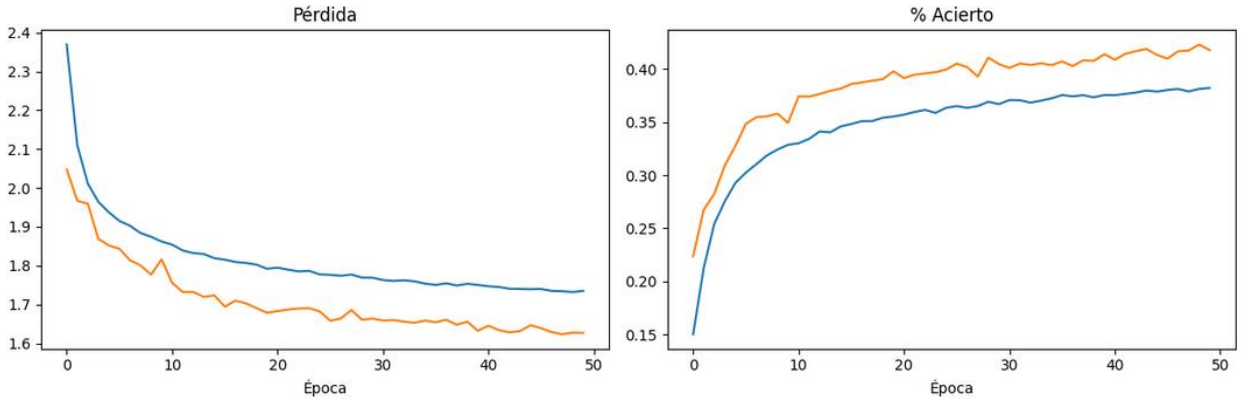
	ADAM + Dropout					
	Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*
Relu	3	0,2766	1,9304	0,2911	1,9537	501
Tanh	1	0,3905	1,7166	0,4114	1,6395	435
Sigmoid	1	0,4919	1,4266	0,4991	1,4188	491

1.3. RMSProp + Dropout

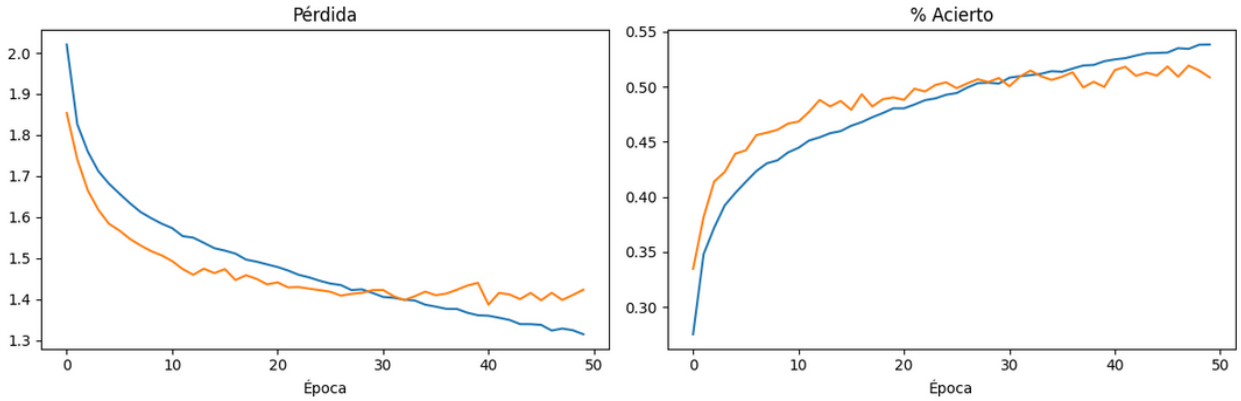
1.3.1. RMSProp + ReLu + Dropout



1.3.2. RMSProp + Tanh + Dropout



1.3.3. RMSProp + Sigmoid + Dropout



1.3.4. Resultados

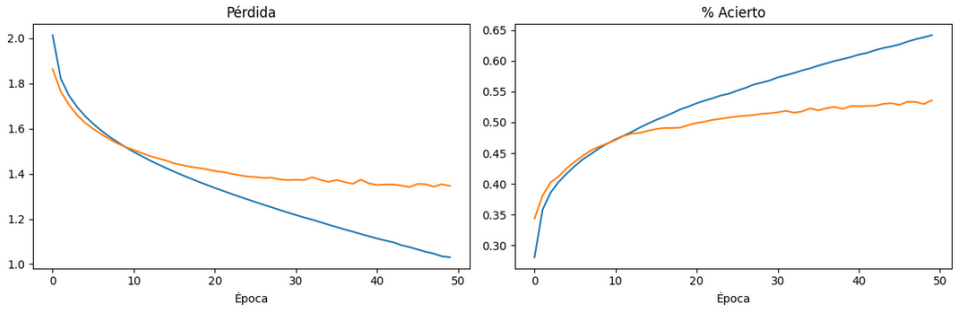
	RMSProp + Dropout					
	Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*
Relu	3	0,3462	1,8060	0,3756	1,8251	450

Tanh	3	0,3835	1,7362	0,4178	1,6266	465
Sigmoid	1	0,5405	1,3153	0,5084	1,4231	422

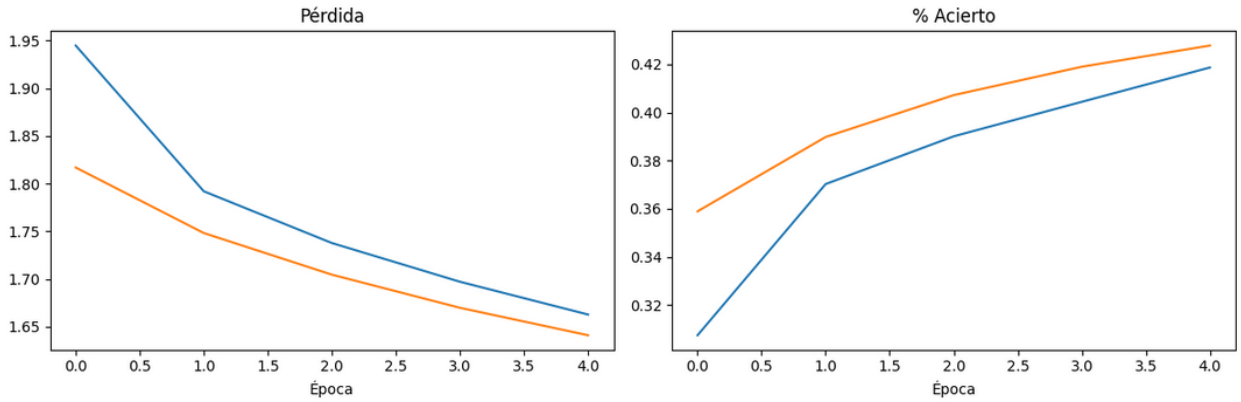
2. Regularización con Earlystopping

2.1. SGD + Earlystopping

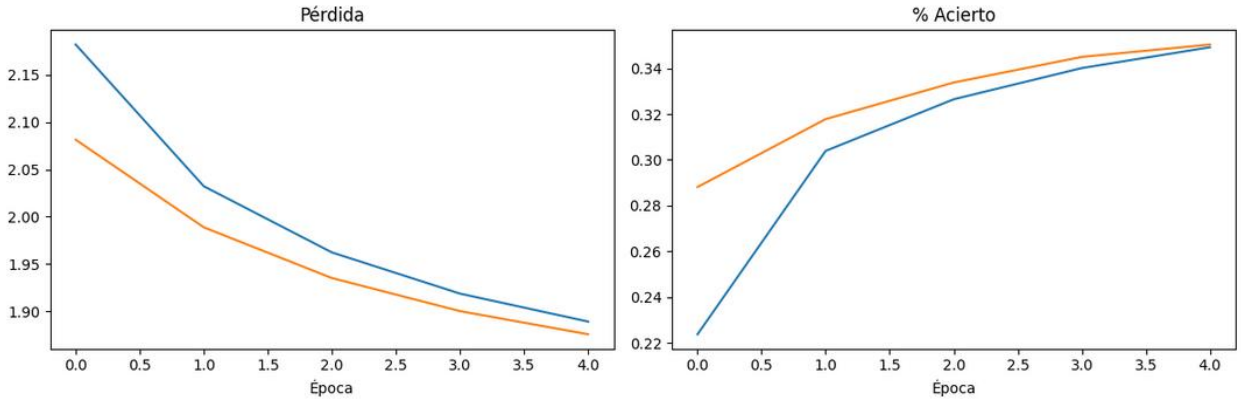
2.1.1. SGD + ReLu + Earlystopping



2.1.2. SGD + Tanh + Earlystopping



2.1.3. SGD + Sigmoid + Earlystopping

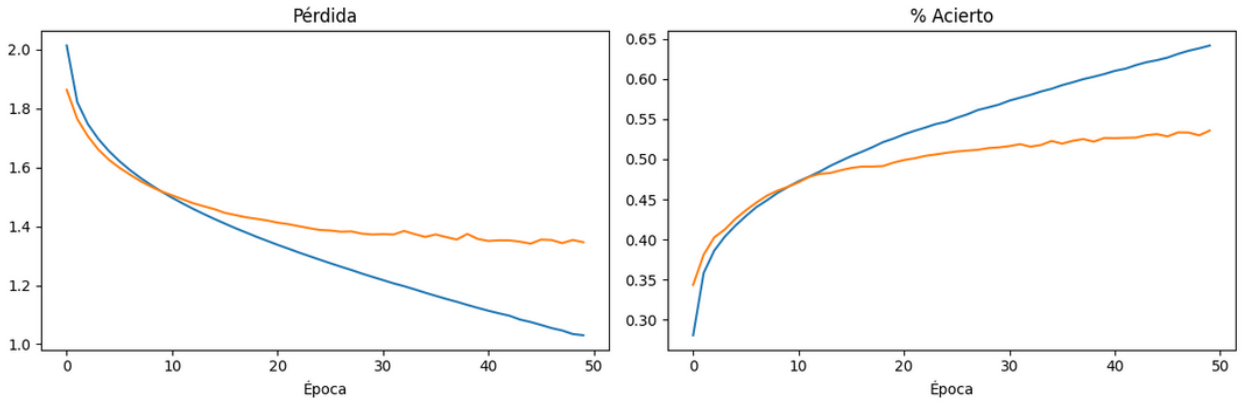


2.1.4. Resultados

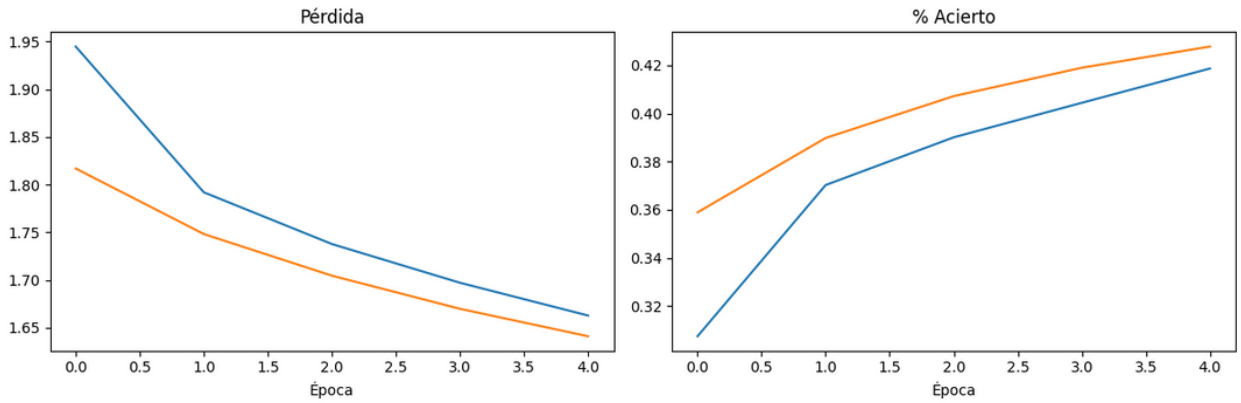
	SGD + Earlystopping						
	Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*	*epoc*
Relu	3	0,6413	1,0331	0,5357	1,3461	230	50
Tanh	3	0,4171	1,6724	0,4278	1,6412	27	5
Sigmoid	1	0,3473	1,8967	0,3504	1,8757	23	5

2.2. Adam + Earlystopping

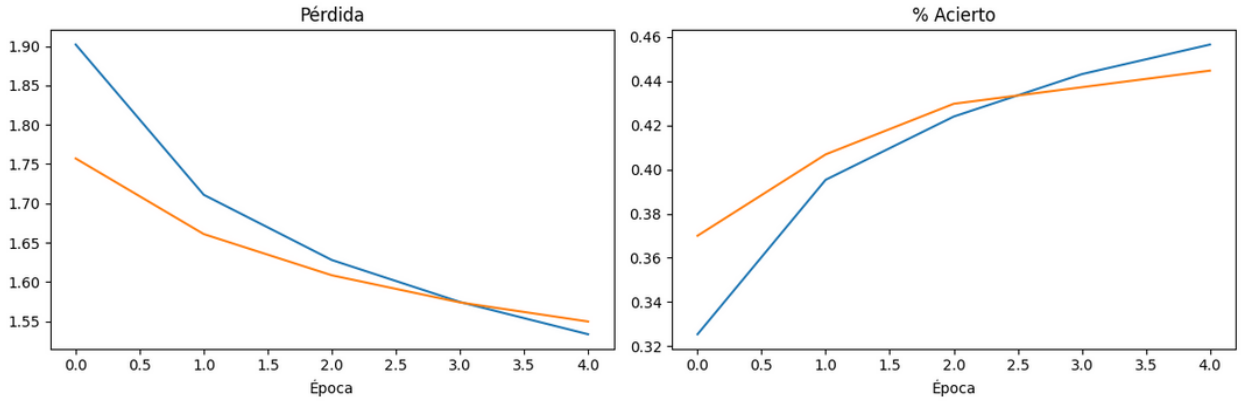
2.2.1. Adam + ReLu + Earlystopping



2.2.2. Adam + Tanh + Earlystopping



2.2.3. Adam + Sigmoid + Earlystopping

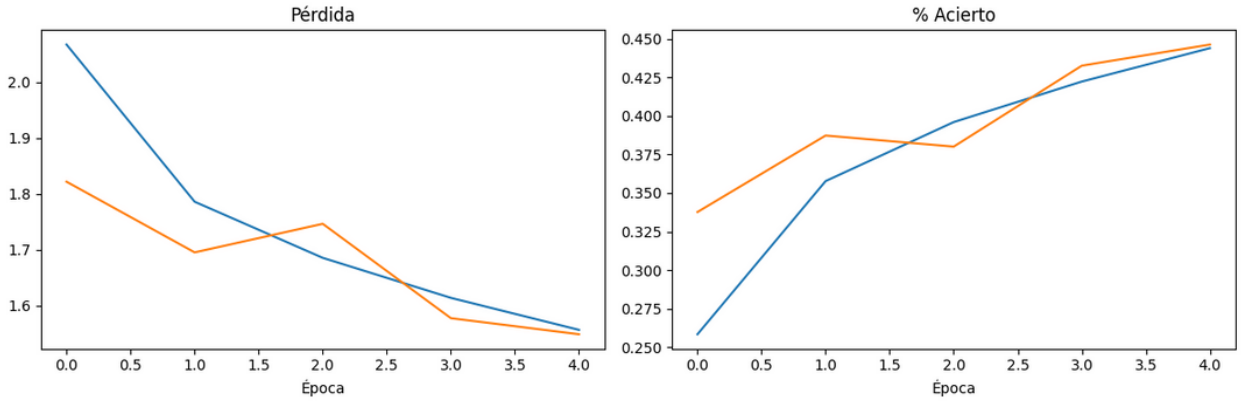


2.2.4. Resultados

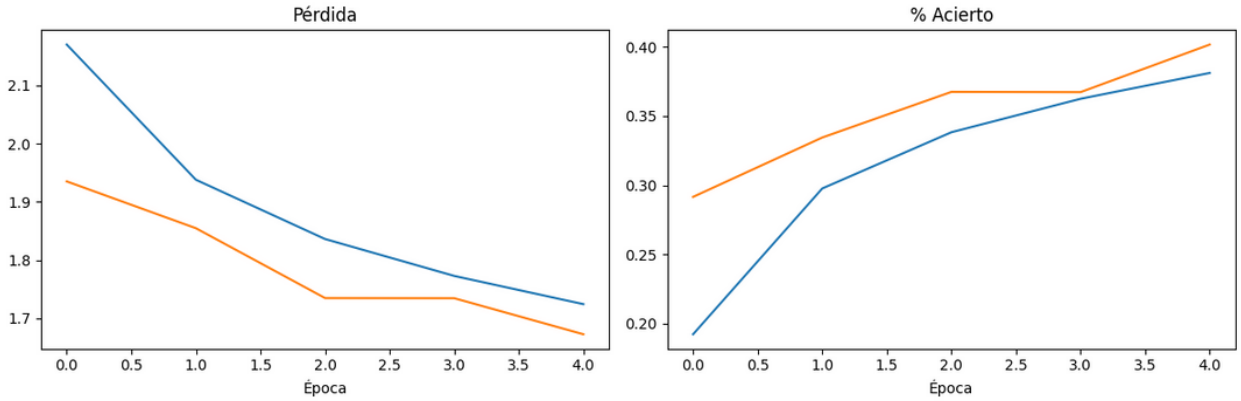
	ADAM + Earlystopping						
	Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*	*epoc*
Relu	3	0,4620	1,4975	0,4621	1,5093	52	5
Tanh	1	0,4096	1,6627	0,4064	1,6590	45	5
Sigmoid	1	0,4522	1,5431	0,4447	1,5496	48	5

2.3. RMSProp + Earlystopping

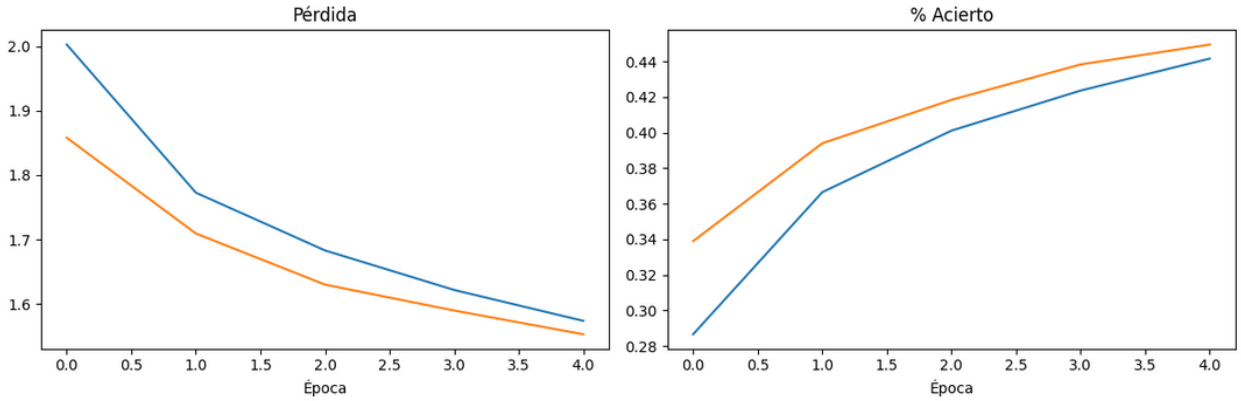
2.3.1. RMSProp + ReLu + Earlystopping



2.3.2. RMSProp + Tanh + Earlystopping



2.3.3. RMSProp + Sigmoid + Earlystopping



2.3.4. Resultados

	RMSProp + Earlystopping						
	Capas	*accuracy*	*loss*	*val_accuracy*	*val_loss*	*tiempo*	*epoc*
Relu	3	0,4364	1,5700	0,4463	1,5485	44	5

Tanh	3	0,3706	1,7445	0,4017	1,6727	48	5
Sigmoid	1	0,4354	1,5872	0,4495	1,5596	44	5

4 ANEXO: Implementación Red Convolucional

4.1. **Introducción de Implementación Red Convolucional**

Para lo anterior debemos preparar nuestros datos tanto de entrenamiento como de prueba. Se normalizarán los datos dividiéndolos por 255 y nuestros datos de prueba se les aplicará el método to_categorical, que convertirá un vector de clase (enteros) en una matriz de clase binaria. Dicho de otra manera, se convertirán las etiquetas a one-hot ecnoding. Esto será útil para su uso con categorical_crossentropy.

4.1. **Red Convolucional: Conv2D y MaxPool2D**

Todos los modelos que ejecutemos en el presente trabajo se implementarán 2 veces: Uno con una versión base, esto es, con los elementos básicos y necesarios que nos permitan ejecutar nuestra red y que han sido proporcionados en clases.

Luego, se analizará el rendimiento de estos modelos y se le aplicarán mejoras. En algunos casos estas mejoras nos aumentarán el tiempo de ejecución, los cuales quedarán documentados, pero mejorarán la precisión de nuestro modelo de manera relevante.

4.1.1. **Red Convolucional Base**

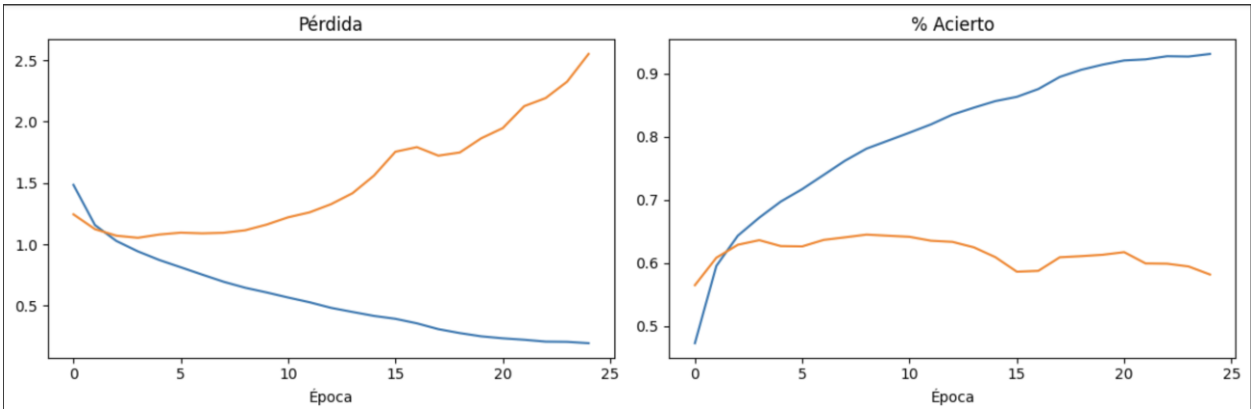
Nuestra red convolucional base se caracteriza por los siguientes elementos:

- Se le aplicará una capa Conv2D (64 neuronas y activador ReLu) y otra capa de pooling MaxPool2D
- Se le aplicarán una capa Flatten, con el cual trasnformaremos nuestra entrada multidimensional en una matriz unidimensional.
- Otra capa densa de 128 neuronas con activador ReLu y otra capa de salida con 10 neuronas y activador softmax.
- Se utilizará el optimizador Adam, ya que es el que mejores resultados ofrece en redes convolucionales.

Layer (type)	Output Shape	Param #
conv2d_69 (Conv2D)	(None, 30, 30, 64)	1,792
max_pooling2d_5 (MaxPooling2D)	(None, 15, 15, 64)	0
flatten_5 (Flatten)	(None, 14400)	0
dense_8 (Dense)	(None, 128)	1,843,328
dense_9 (Dense)	(None, 10)	1,290

Total params: 1,846,410 (7.04 MB)
Trainable params: 1,846,410 (7.04 MB)
Non-trainable params: 0 (0.00 B)

Tiempo de ejecución = 16:41 mins



Análisis: Nuestra red convolucional base arroja resultados dispares: Si bien muestra un alto acierto con los datos de entrenamiento, falla con los datos de prueba, lo que muestra que nuestro modelo está sobreajustado. Vamos a mejorar nuestro modelo.

4.1.2. **Red Convolucional Mejorado**

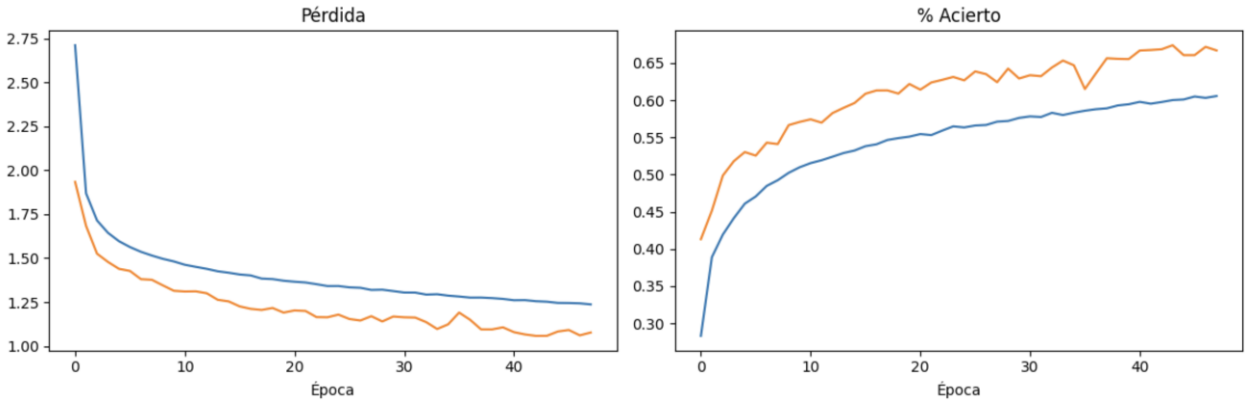
A nuestra red convolucional se le han implementado algunas mejoras:

- Se **aumenta la profundidad de la red**, añadiendo más capas convolucionales (Conv2D) y de pooling (MaxPool2D).
- Se implementa técnicas de aumento de datos (**data augmentation**). Esto implica crear nuevas versiones de las imágenes de entrenamiento a través de transformaciones aleatorias como rotaciones, desplazamientos y flips.
- Se le incorporan dos capas de regularización **Dropout**.
- Se añade regularizador **L2** a las capas densas.
- A nuestro optimizador Adam se le le ajusta el la tasa de aprendizaje (**learning_rate**) a 0.0001
- Con el objeto de optimizar los recursos de nuestra máquina se implementa también la técnica de **Early Stopping** para detener el entrenamiento cuando el modejo deje de mejorar en el conjunto de validación.
- Es por ello que se aumentan también las **épocas** de 20 a 50.

Layer (type)	Output Shape	Param #
conv2d_67 (Conv2D)	(None, 30, 30, 64)	1,792
max_pooling2d_3 (MaxPooling2D)	(None, 15, 15, 64)	0
conv2d_68 (Conv2D)	(None, 13, 13, 128)	73,856
max_pooling2d_4 (MaxPooling2D)	(None, 6, 6, 128)	0
dropout_4 (Dropout)	(None, 6, 6, 128)	0
flatten_4 (Flatten)	(None, 4608)	0
dense_6 (Dense)	(None, 128)	589,952
dropout_5 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 10)	1,290

Total params: 666,890 (2.54 MB)
Trainable params: 666,890 (2.54 MB)
Non-trainable params: 0 (0.00 B)

Tiempo de ejecución = 57:07 mins



Análisis: Como podemos observar, el redimiento de nuestro modelo mejoró considerablemente, con una tasa de aprendizaje del 66% en la última época que el modelo siguió mejorando sus datos de aprendizaje, gracias al Early Stopping implementado. Hay que hacer presente que a pesar del Early Stopping, el tiempo de ejecución de nuestra red aumentó de 16 a 57 minutos.

4.2. **Red Convolucional: ResNet50**

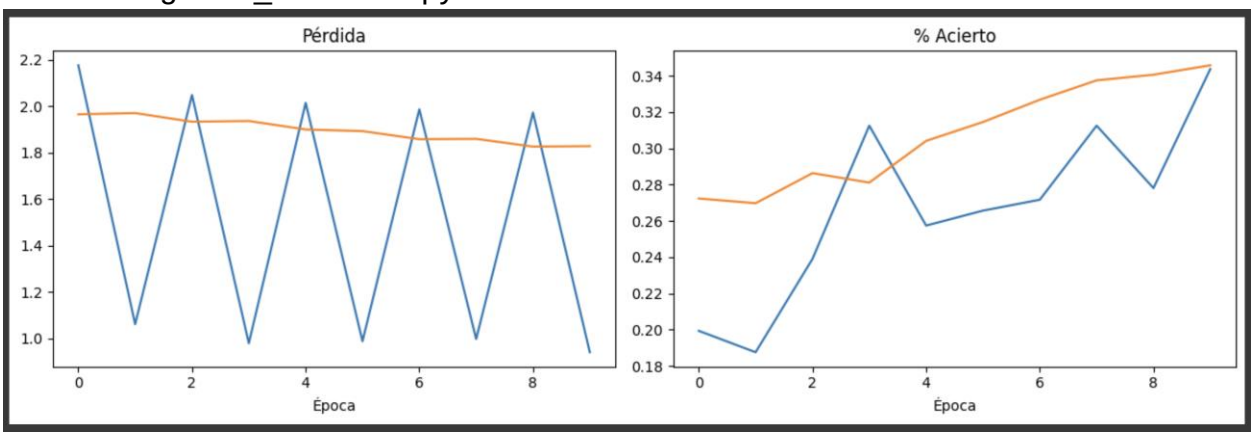
4.2.1. **ResNet50 Base**

En primer lugar, se implementará un modelo ResNet50 con los elementos básicos para su funcionamiento. Al igual que el caso anterior, se analizarán sus resultados y se aplicarán cambios con el fin de conseguir mejorarlos.

Este modelo base tiene ciertas características:

- Se utiliza modelo ResNet50 preentrenado en ImageNet, excluyendo las capas superiores (include_top=False).
- Asimismo, se añaden capas adicionales para ajustar la arquitectura a nuestro problema de clasificación de 10 clases.

- Se compilará utilizando optimizador Adam y función de pérdida categorical_crossentropy.

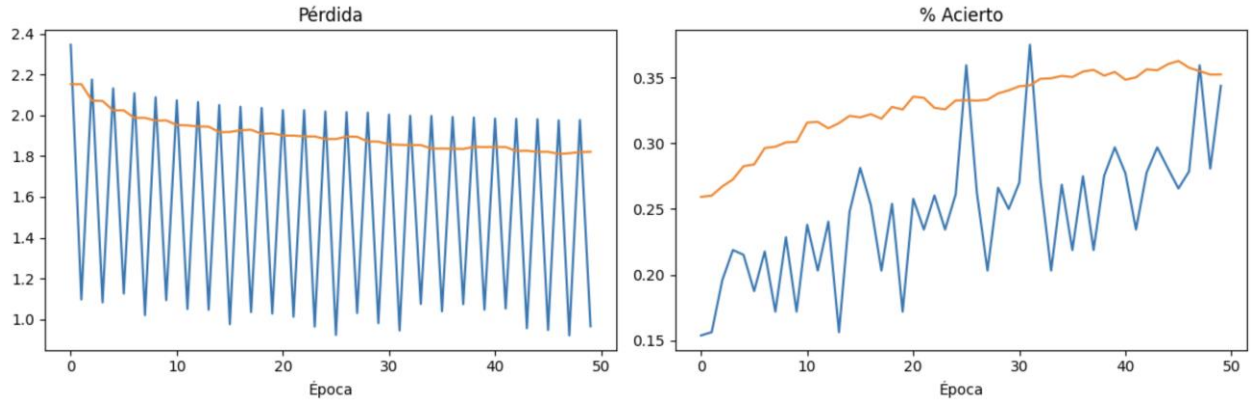


Análisis: Nuestros resultados han resultado más bien irregulares, con noveles de acierto bajos (35%) y pérdidas (loss) altas. Es por eso que implementaremos algunas mejoras a continuación.

4.2.2. **ResNet50 Mejorado**

Para nejorar el rendimiento de nuestro ResNet50 haremos lo siguiente:

- Se agrega una capa de **Dropout**, sobreajuste al desactivar aleatoriamente algunas neuronas durante el entrenamiento.
- Se añade **Zoom** y **Shear** para mejorar la variabilidad de los datos de entrenamiento.
- Se disminuye la tasa de aprendizaje (**learning_rate**) para que el modelo aprenda de manera más estable.
- Se aumenta el número de **épocas**, de 10 a 50, para permitir que el modelo aprenda mejor.



Análisis: Los resultados nos enseñan que el modelo ha mejorado su capacidad de aprendizaje, aunque a un ritmo más lento (de 11 a 51 minutos). Es importante señalar

que durante la ejecución es posible encontrar algunos warnings, los cuales significan que posiblemente no hay datos suficientes en nuestro dataset para cada época.

4.3. **Red Convolucional: Transfer Learning**

De acuerdo a la Unir, "*el transfer learning (aprendizaje por transferencia) se basa en el concepto de reutilización de elementos de los modelos de machine learning (aprendizaje automático) preentrenados en nuevos modelos que se utilizarán para fines similares. Esto permite optimizar los recursos y los datos etiquetados necesarios para el entrenamiento*" (https://www.unir.net/ingenieria/revista/transfer-learning/).

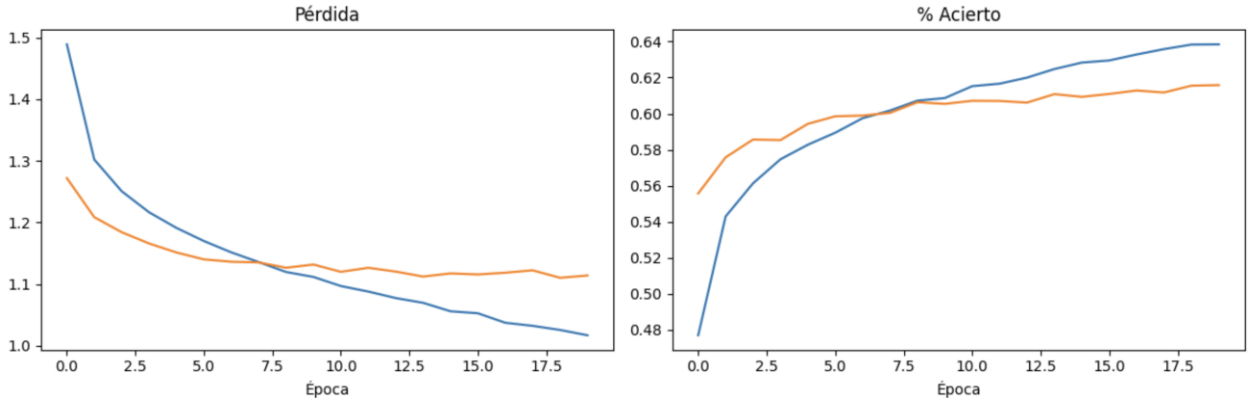
Mediante esta técnica podemos reutilizar elementos ya entrenados de un modelo de machine learning, de manera que nos permite optimizar los recursos y la cantidad de datos etiquetados. Es recomendable su uso cuando la tarea a ejecutar utilice muchos recursos.

En este caso se utilizará la arquitectura VGG16 y se adaptará par usarlo en nuestro dataset Cifar10. Al igual que en los casos anteriores, se implementará un primer modelo base y luego se procederá a mejorarlo para aumentar su rendimiento.

4.3.1. **VGG16 Base**

Comenzaremos con una VGG16 base, esto es, sin la parte superior (la última capa de clasificación). Además, se han implementado los siguientes elementos:

- Se congelan las capas para que sus pesos no se actualicen durante el entrenamiento (layer.trainable = False)
- Se añaden capas Flatten y Dropout, además de dos capas densas, con el fin de adaptar nel modelo de clasificación a las 10 clases.
- Se complila usando Adam y categorical_crossentropy.

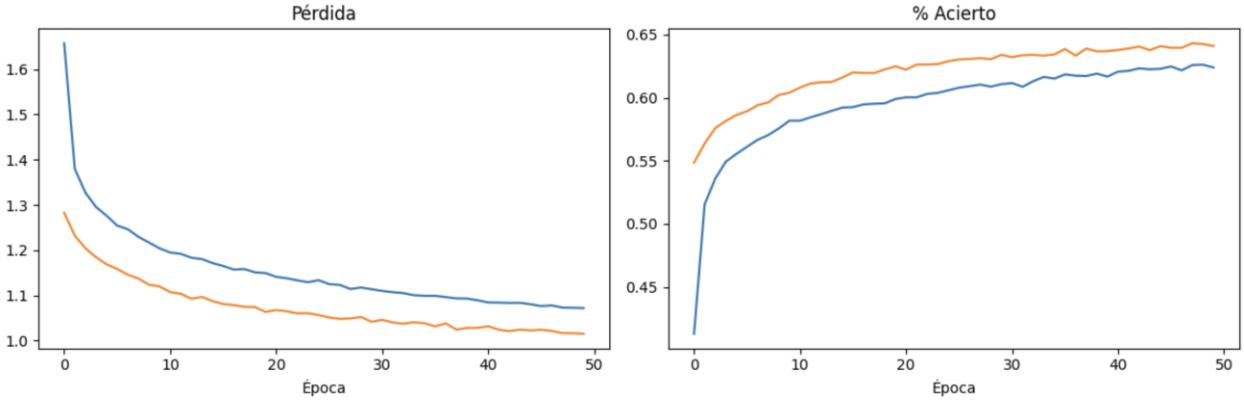


Análisis: Es posible apreciar que si bien nuestro modelo tuvo un buen comienzo, alcanzando un nivel aceptable de precisión (cercana al 60%), ya en la época 10 podemos observar que nuestro modelo ha dejado de aprender, por lo que debemos implementar mejoras. ¿Podemos mejorar nuestro modelo? Veamos.

4.3.2. **VGG16 Mejorado**

Para mejorar el rendimiento de nuestro VGG16, se han hecho los siguientes ajustes:

- Se ha ajustado la tasa de aprendizaje (**learning_rate**) a 1e-5.
- Se han aplicado técnicas de augmentación con **ImageDataGenerator**.
- Se implementan ténincas de entrenamiento de precisión mixta (**Mixed Precision Training**), aprovechando la capaidad de la GPU para acelerar el entrenamiento.
- Implementar **Model Checkpoint**, con el cual podamos entrenar una sola vez el modelo y en etapas posteriores poder retomar el entrenamiento desde el estado guardado.
- Implementar **Early Stopping** para detener la ejecución del entrenamiento cuando el modelo no sea capaz de seguir aprendiendo.



Análisis: Nuestro VGG16 mejorado demostró un aprendizaje efectivo y una buena capacidad de generalización, como lo indica la mejora constante en precisión y la disminución en la pérdida tanto en los conjuntos de entrenamiento como de validación. A pesar del Early Stopping, el modelo llegó hasta las 50 épocas establecidas, lo que sugiere que el modelo aún tenía capacidad de seguir aprendiendo. Sin embargo, debemos constatar que a pesar de las técnicas implementadas para mejorar el rendimiento, el tiempo de ejecución sigue siendo alto, con 1 hora 40 minutos en completarse. Podría esto limitarse disminuyendo el número de épocas, aunque a costa de sacrificar precisión.

4.4. **Evaluación**

4.4.1. Resultados Red Convolucional

Modelo	Capas	Épocas	Kernel	Tamaño Kernel	Padding	Pooling	Stride	Val-Acc	Val_loss	Tiempo (mins)
Base	4	25	Conv2d	3x3	valid	MaxPool 2D	1	0.5817	2.5520	16:41
Mejorado	8	50	Conv2d	3x3	valid	MaxPool 2D	1	0.6670	1.0771	57:07

4.4.2. Resultados ResNet50

Modelo	Capas	Épocas	Kernel	Tamaño Kernel	Padding	Pooling	Stride	Val-Acc	Val_loss	Tiempo (mins)
Base	50	10	Conv2d (ResNet 50)	3x3	same	GlobalAveragePooling2D	2	0.3458	1.8575	11:18
Mejorado	50	50	Conv2d (ResNet 50)	3x3	same	GlobalAveragePooling2D	2	0.3458	1.8213	51:34

4.4.3. Resultados VGG16

Modelo	Capas	Épocas	Kernel	Tamaño Kernel	Padding	Pooling	Stride	Val-Acc	Val_loss	Tiempo (mins)
Base	16	20	Conv2d (VGG16)	3x3	same	MaxPool 2D	1	0.6158	2.1139	50:55
Mejorado	16	50	Conv2d (VGG16)	3x3	same	MaxPool 2D	1	0.6409	1.0151	104

4.5. **Conclusiones**

Al comparar los modelos, observamos las siguientes tendencias:

- Precisión y pérdida de validación:
  - Todos los modelos mejorados superan a los modelos base en precisión de validación (Val_Acc) y pérdida de validación (Val_Loss), lo cual demuestra que las técnicas de mejora implementados a cada una de las arquitecturas planteadas han resultado ser efectivas.
  - Las mejoras son consistentes y significativas en todos los modelos, pasando de un promedio de Val_Acc de 0.5817 a 0.6670 y reduciendo la Val_Loss de aproximadamente un promedio de 2.5520 a 1.0771.

- Tiempo de entrenamiento:
  - El modelo mejorado de la Red Convolucional personalizada requiere aproximadamente 57 minutos, que es un incremento alto comparado con su base, pero que mejoró considerablemente sus resultados.
  - El modelo mejorado de ResNet50 toma 51 minutos, lo cual es bastante eficiente dado su incremento en precisión y reducción de pérdida.

  - El modelo mejorado de VGG16 toma significativamente más tiempo (104 minutos), casi el doble de su versión base. Una manera de disminuir de manera rápida este tiempo es reduciendo las épocas, a costa eso sí de disminuir la precisión del modelo. Sin emabrgo, después de la época 10 la tasa de mejora es moderado, por lo que podría ser un límite aceptable a implementar.

- ¿Peor modelo?
  - Por los resultados mostrados, ResNet50 no mostró mejorar significativamente los resultados del entrenamiento después de la implementación de mejoras, por lo que resultó ser el peor modelo.

- ¿Mejor modelo?
  - Tanto VGG16 como nuestra CNN personalizada mostraron buenos resultados. Ahora bien, nuestra CNN personalizada y mejorada muestra un menor tiempo de ejecución, por lo que en menos tiempo, con menos capas y general con menos recursos es capaz de obtener los mismos resultados, siendo el mejor modelo.