



Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Redes Convolucionales

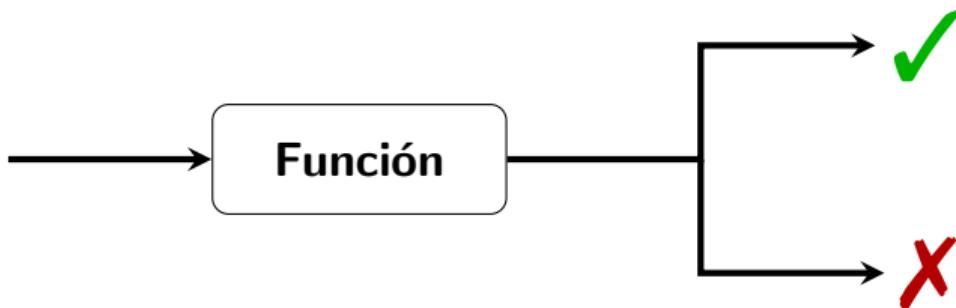
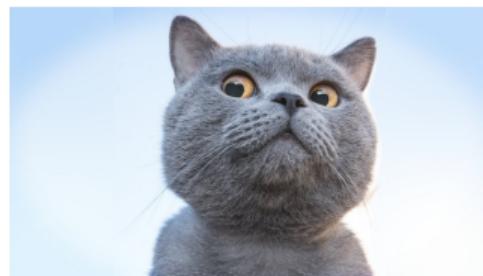
Rodrigo Toro Icarte (rntoro@uc.cl)

MCD - Aprendizaje profundo

7 de Noviembre, 2023

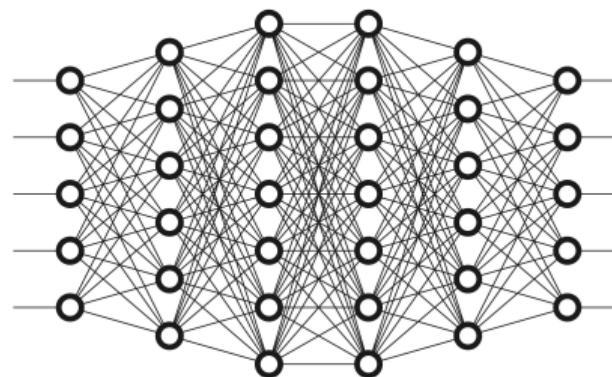
Recordatorio

Deep learning



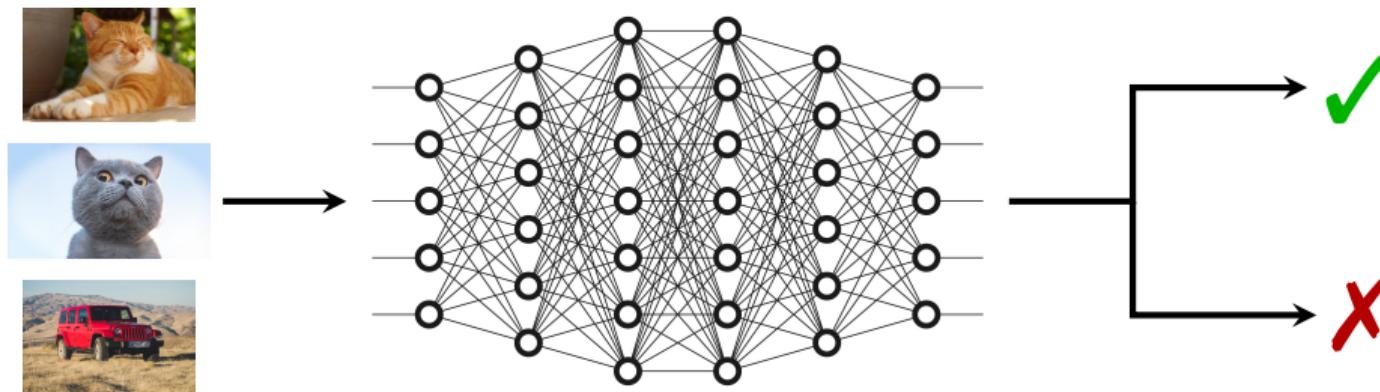
Deep learning

Esta función es una red neuronal, cuya salida depende del valor de sus pesos.



Deep learning

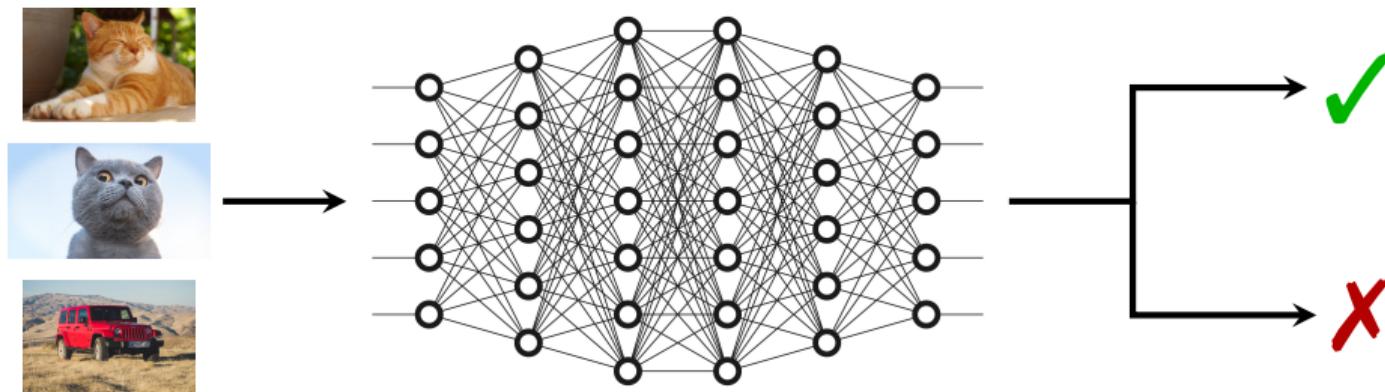
Esta función es una red neuronal, cuya salida depende del valor de sus pesos.



Entrenar la red significa encontrar pesos que generan la salida correcta.

Deep learning

Esta función es una red neuronal, cuya salida depende del valor de sus pesos.

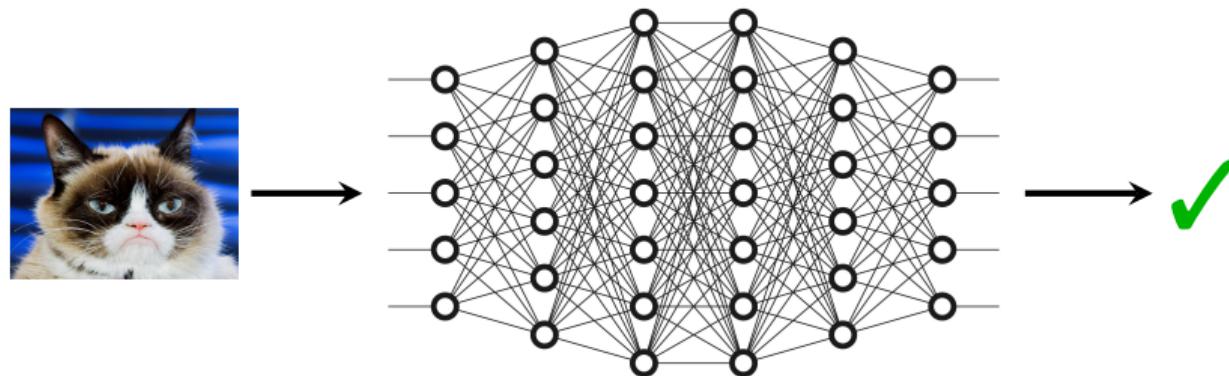


Entrenar la red significa encontrar pesos que generan la salida correcta.

La gracia de deep learning es que generaliza bien.

Deep learning

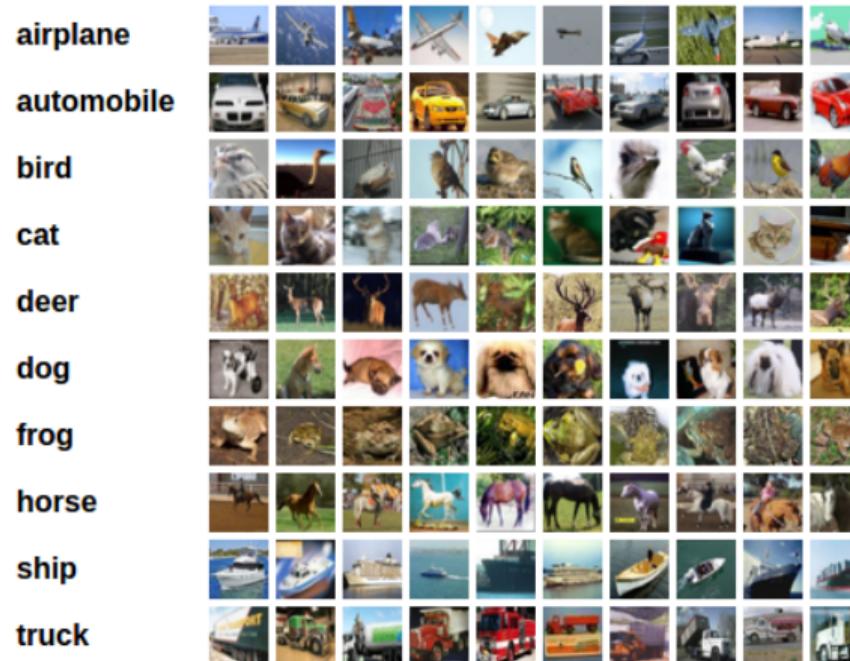
Esta función es una red neuronal, cuya salida depende del valor de sus pesos.



Entrenar la red significa encontrar pesos que generan la salida correcta.

La gracia de deep learning es que generaliza bien.

Ejemplo: CIFAR10 dataset



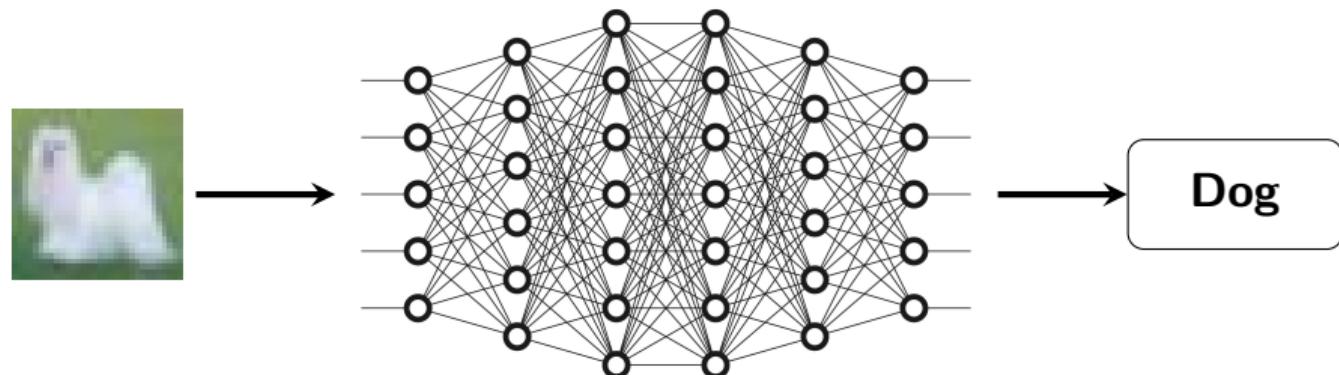
Objetivo

Entrenar una red que nos diga si una imagen contiene un avión, auto, pájaro, gato, ciervo, perro, rana, caballo, barco o camión.

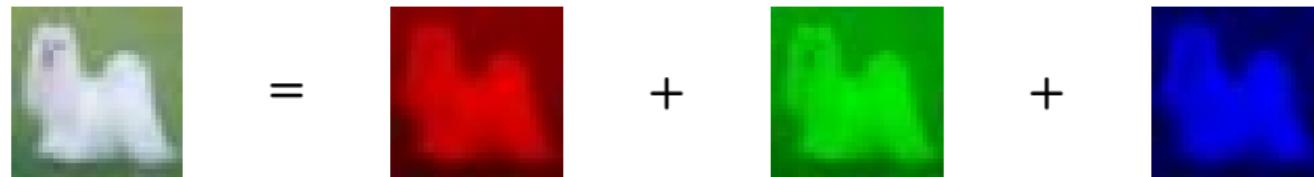
CIFAR-10:

- Input: Imágenes de 32×32 pixeles RGB.
- Output: Diez clases posibles.
- Contiene 50000 ejemplos de entrenamiento.
- Contiene 10000 ejemplos de testeo (que sirven para medir generalización).

Deep learning



¿Cómo codificamos el input de la red?



Input: Un vector de $32 \times 32 \times 3$ números entre 0 y 255.



¿Cómo codificamos el input de la red?



Input: Un vector de $32 \times 32 \times 3$ números entre 0 y 255.



¿Cómo codificamos el output de la red?

Tenemos 10 neuronas de salida, una por cada clase posible, con activación softmax.

Arquitectura red:

- Input: $32 \times 32 \times 3$.
- Capa fully-connected de 120 neuronas relu.
- Capa fully-connected de 84 neuronas relu.
- Capa fully-connected de 10 neuronas softmax.

Arquitectura red:

- Input: $32 \times 32 \times 3$.
- Capa fully-connected de 120 neuronas relu.
- Capa fully-connected de 84 neuronas relu.
- Capa fully-connected de 10 neuronas softmax.

Función de pérdida: Cross-entropy.

Arquitectura red:

- Input: $32 \times 32 \times 3$.
- Capa fully-connected de 120 neuronas relu.
- Capa fully-connected de 84 neuronas relu.
- Capa fully-connected de 10 neuronas softmax.

Función de pérdida: Cross-entropy.

... y entrenamos esta red usando *descenso de gradiente estocástico*.

Descenso de gradiente estocástico:

- Comenzamos desde pesos cualquiera θ_0 .
- Repetir por $t = 1 \dots T$ épocas:
 - Dividir el set de entrenamiento en N batches de manera aleatoria.
 - Repetir por $B_n \in \{B_1 \dots B_N\}$:
 - Calculamos el gradiente respecto al batch B_n :

$$\nabla_{\theta} J(\theta_t; B_n) = \sum_{(x,y) \in B_n} J(\theta_t; x, y)$$

- Actualizamos los pesos en contra del gradiente:

$$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla_{\theta} J(\theta_t; B_n)$$

Descenso de gradiente estocástico:

- Comenzamos desde pesos cualquiera θ_0 .
- Repetir por $t = 1 \dots T$ épocas:
 - Dividir el set de entrenamiento en N batches de manera aleatoria.
 - Repetir por $B_n \in \{B_1 \dots B_N\}$:
 - Calculamos el gradiente respecto al batch B_n :

$$\nabla_{\theta} J(\theta_t; B_n) = \sum_{(x,y) \in B_n} J(\theta_t; x, y)$$

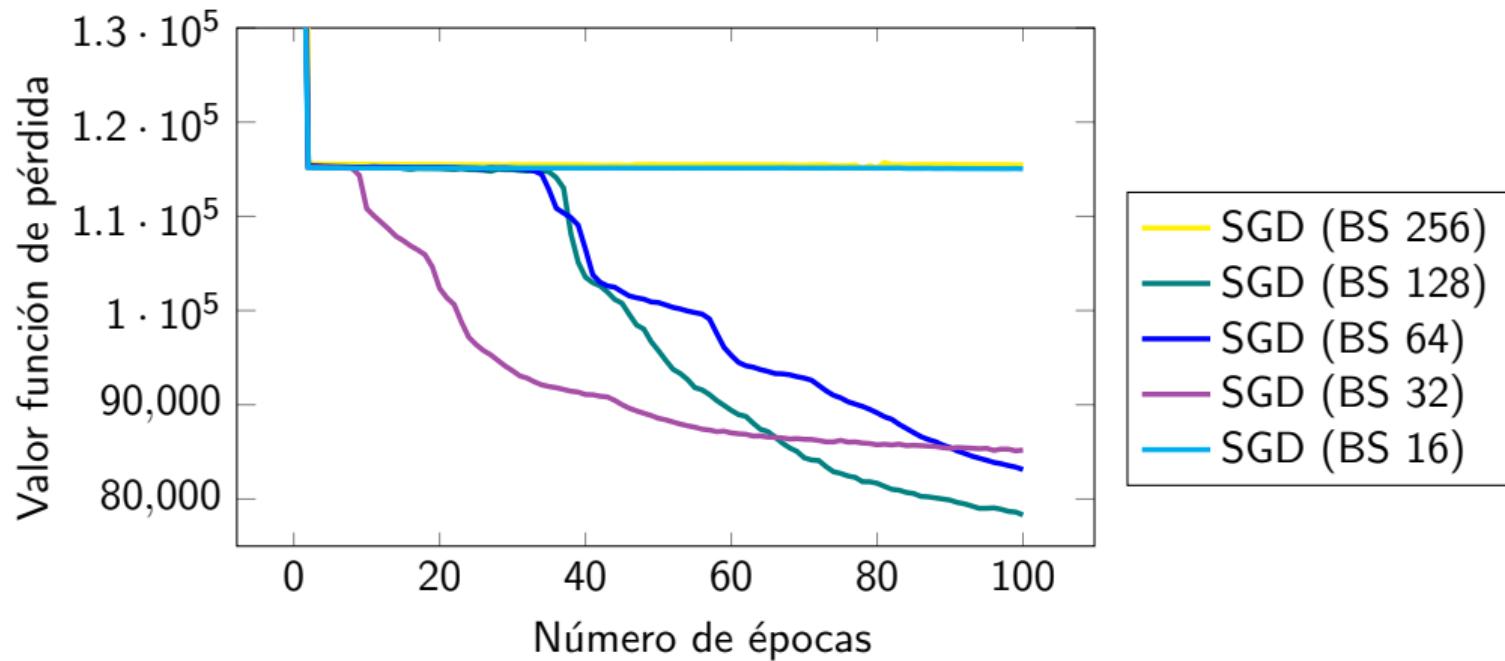
- Actualizamos los pesos en contra del gradiente:

$$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla_{\theta} J(\theta_t; B_n)$$

Consejo: En cualquier aplicación de deep learning, el primer paso es encontrar un modelo que le vaya bien en el set de entrenamiento.

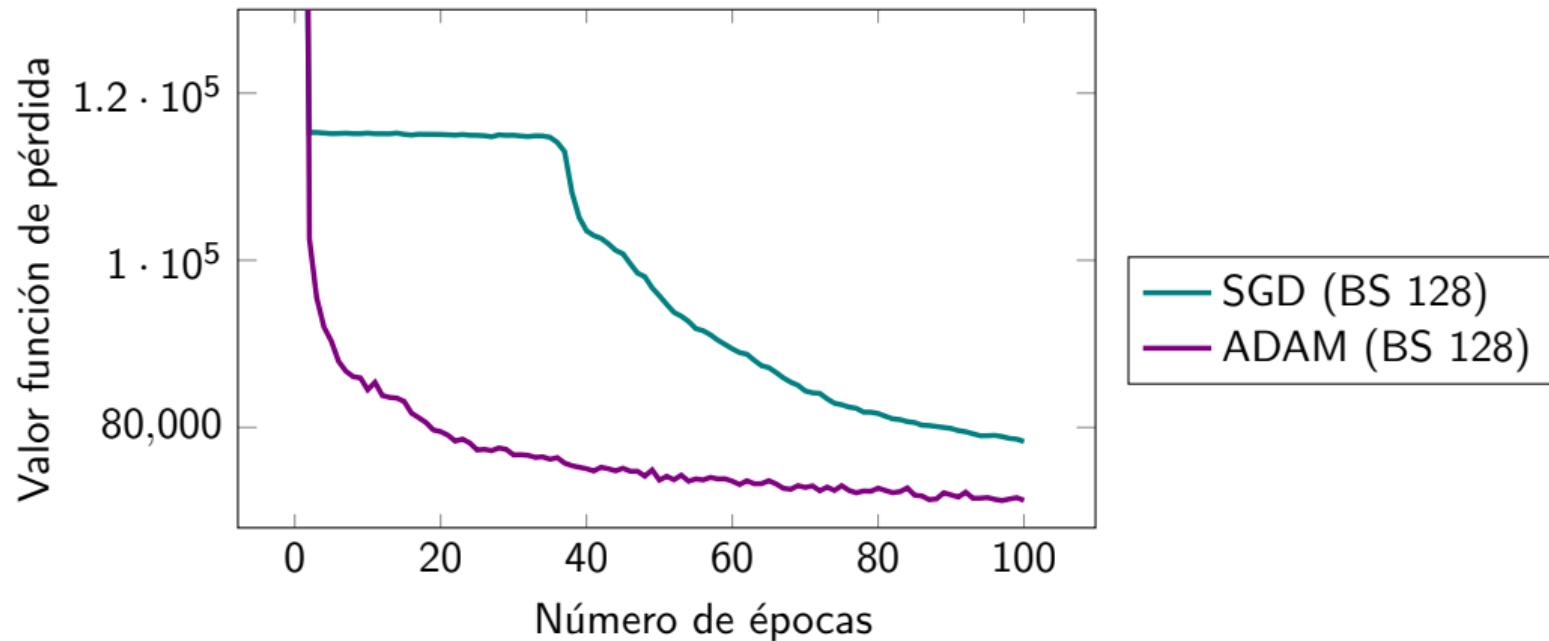
Para ello, vimos que el *batch size* importa.

Para ello, vimos que el *batch size* importa.



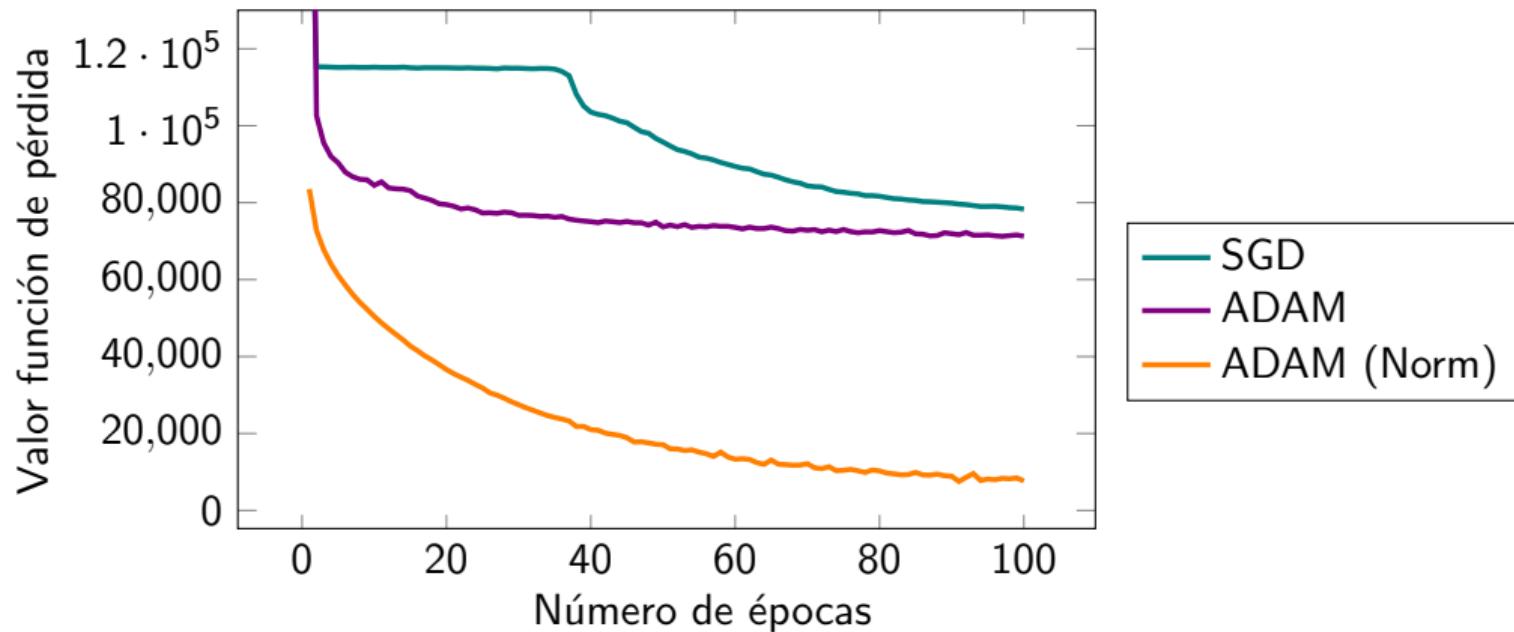
Vimos que el optimizador importa.

Vimos que el optimizador importa.



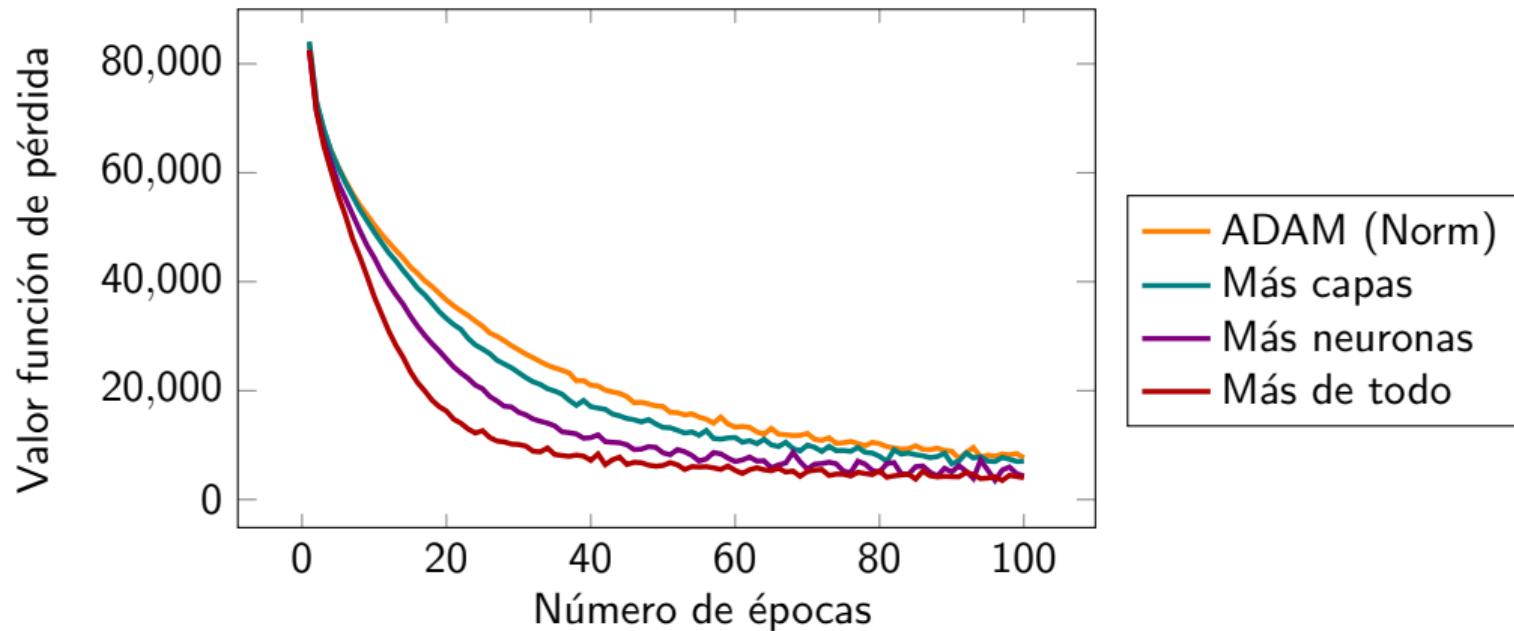
Hay que normalizar los datos.

Hay que normalizar los datos.



El modelo debe ser lo suficientemente grande.

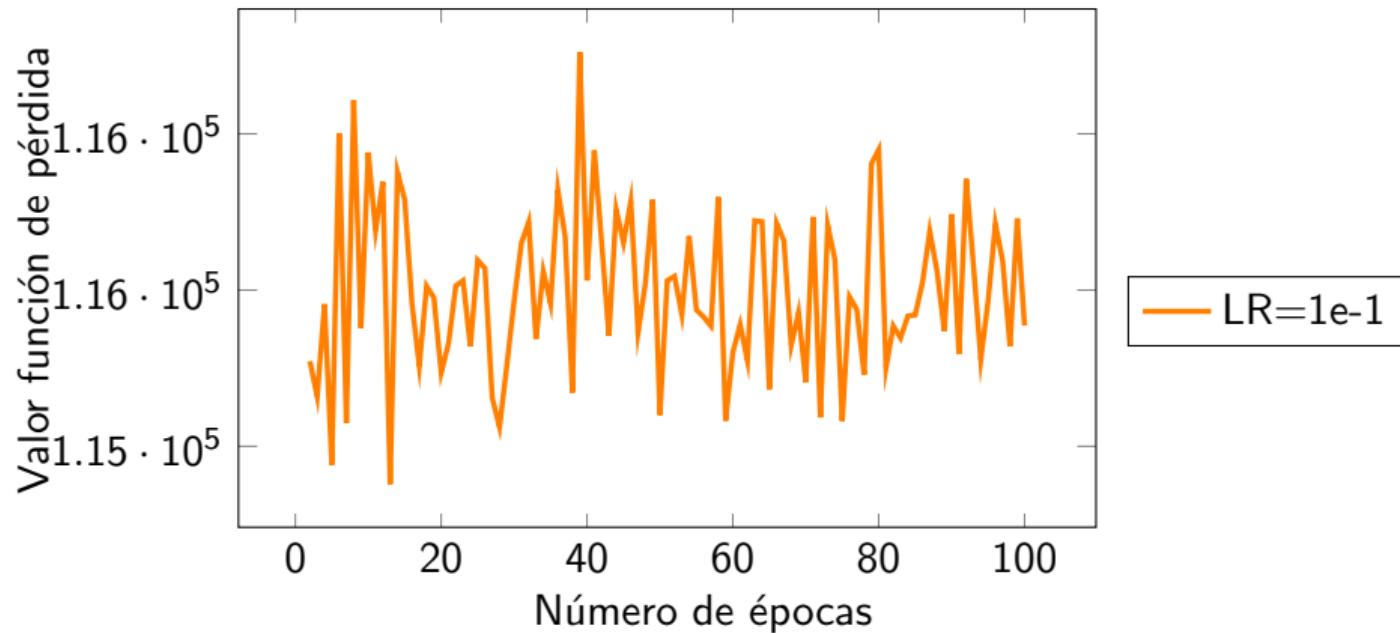
El modelo debe ser lo suficientemente grande.



Finalmente, hay que ajustar el *learning rate*.

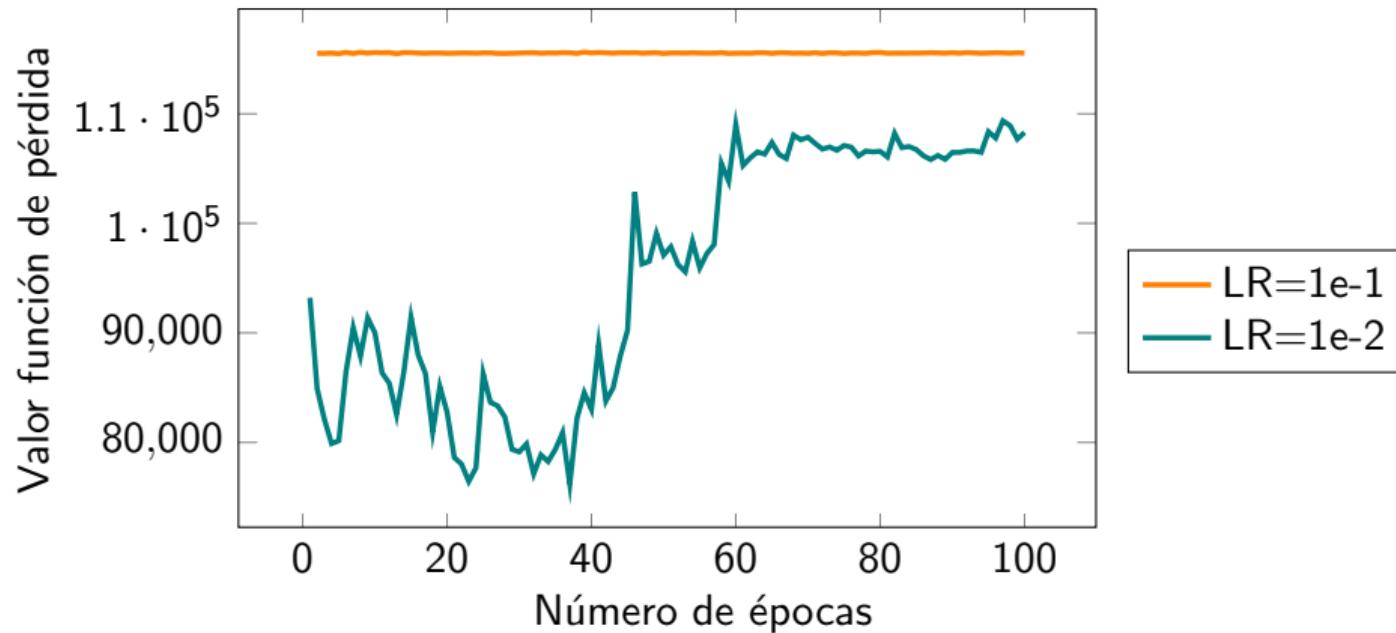
Resumen

Finalmente, hay que ajustar el *learning rate*.



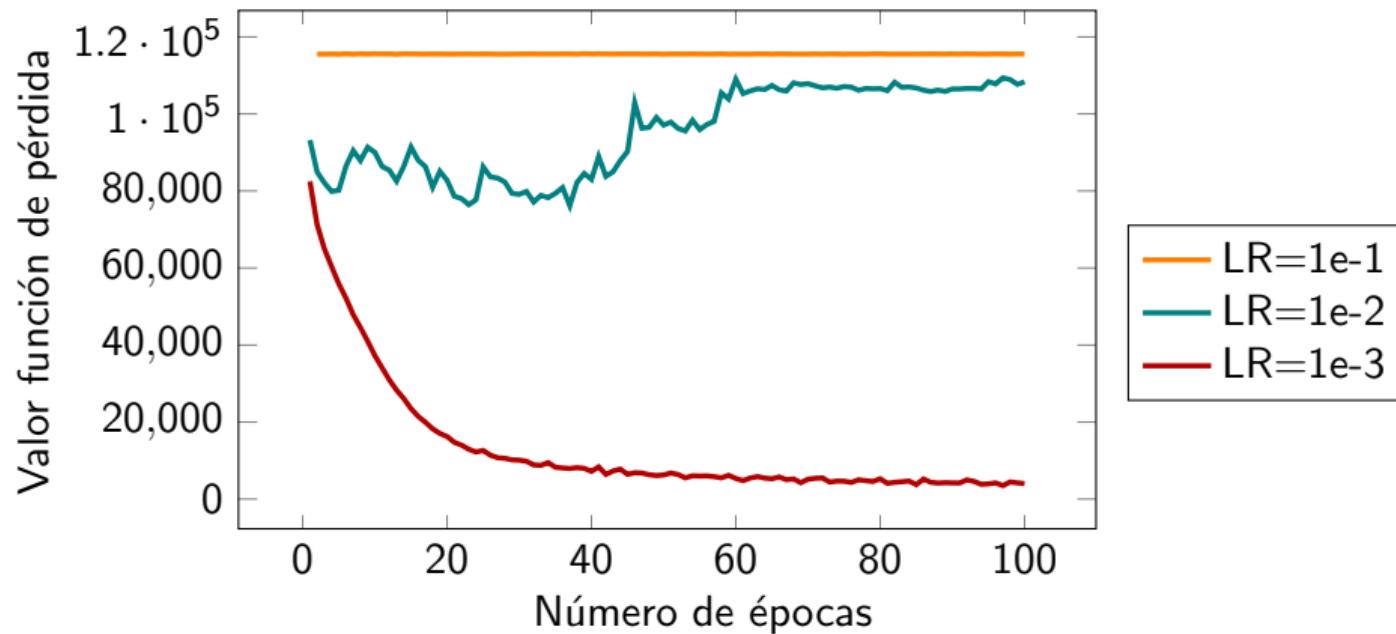
Resumen

Finalmente, hay que ajustar el *learning rate*.



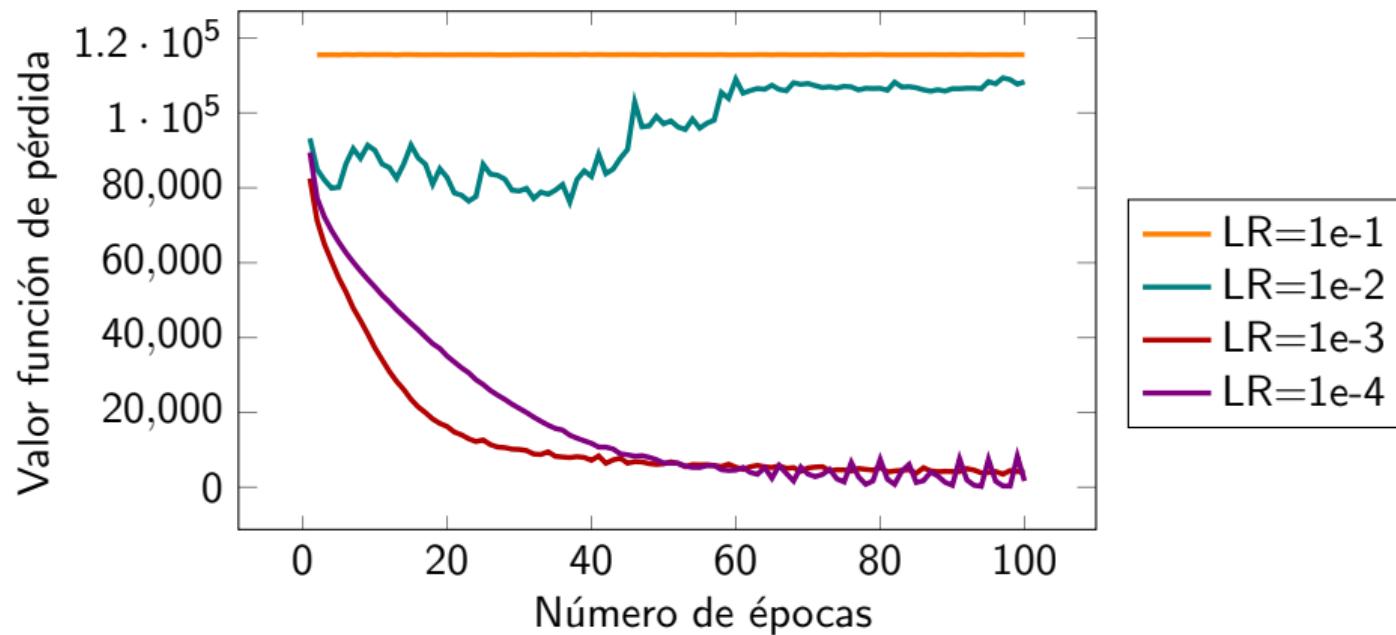
Resumen

Finalmente, hay que ajustar el *learning rate*.



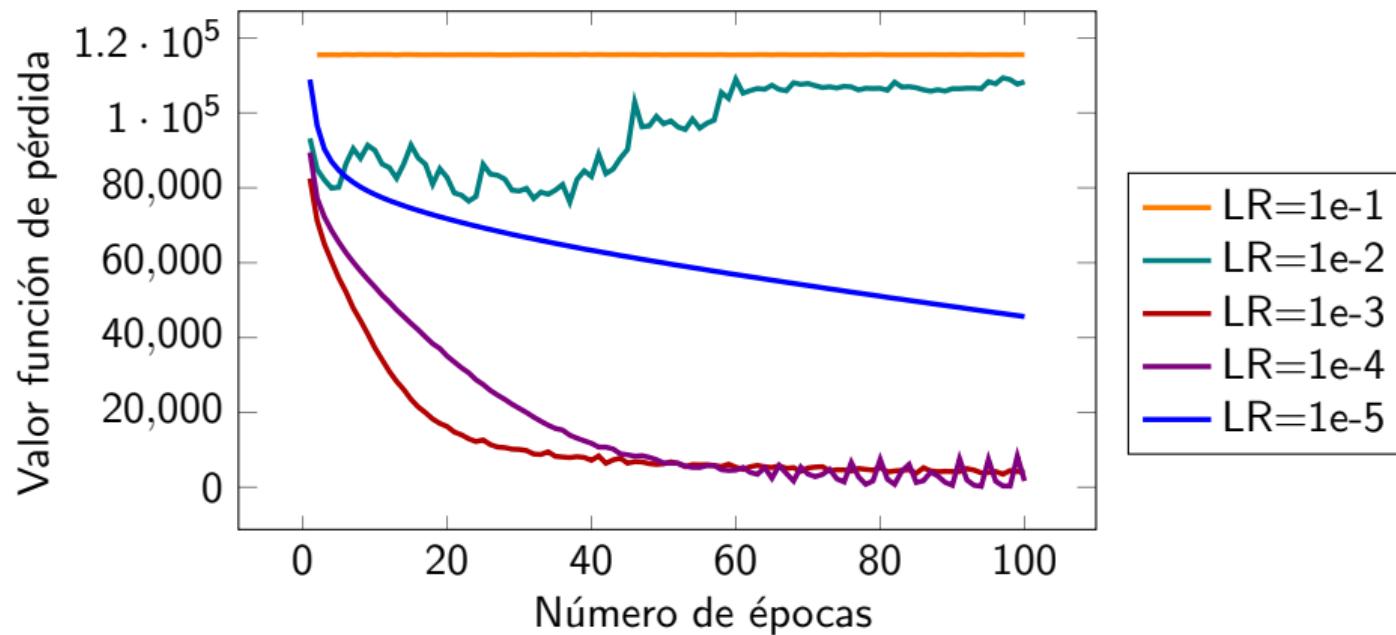
Resumen

Finalmente, hay que ajustar el *learning rate*.



Resumen

Finalmente, hay que ajustar el *learning rate*.



Generalización

Luego de encontrar un modelo que le va bien en el set de entrenamiento, es momento de ver qué tan bien generaliza.

Generalización

Luego de encontrar un modelo que le va bien en el set de entrenamiento, es momento de ver qué tan bien generaliza.

Para medir generalización usamos un *set de test*. Este set no debe ser utilizado ni para entrenar el modelo ni para ajustar sus *hiperparámetros*.

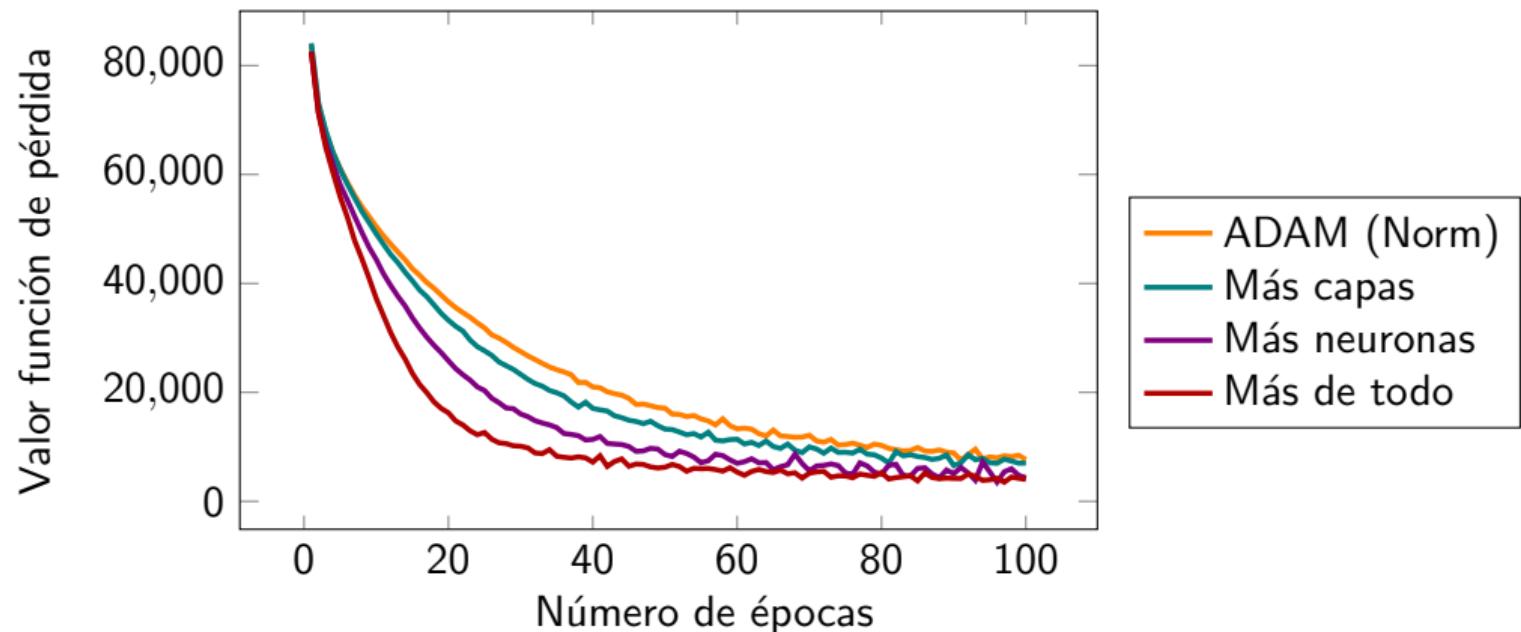
Generalización

Luego de encontrar un modelo que le va bien en el set de entrenamiento, es momento de ver qué tan bien generaliza.

Para medir generalización usamos un *set de test*. Este set no debe ser utilizado ni para entrenar el modelo ni para ajustar sus *hiperparámetros*.

Entonces... ¿Qué tan bien generaliza nuestra red entrenada en CIFAR-10?

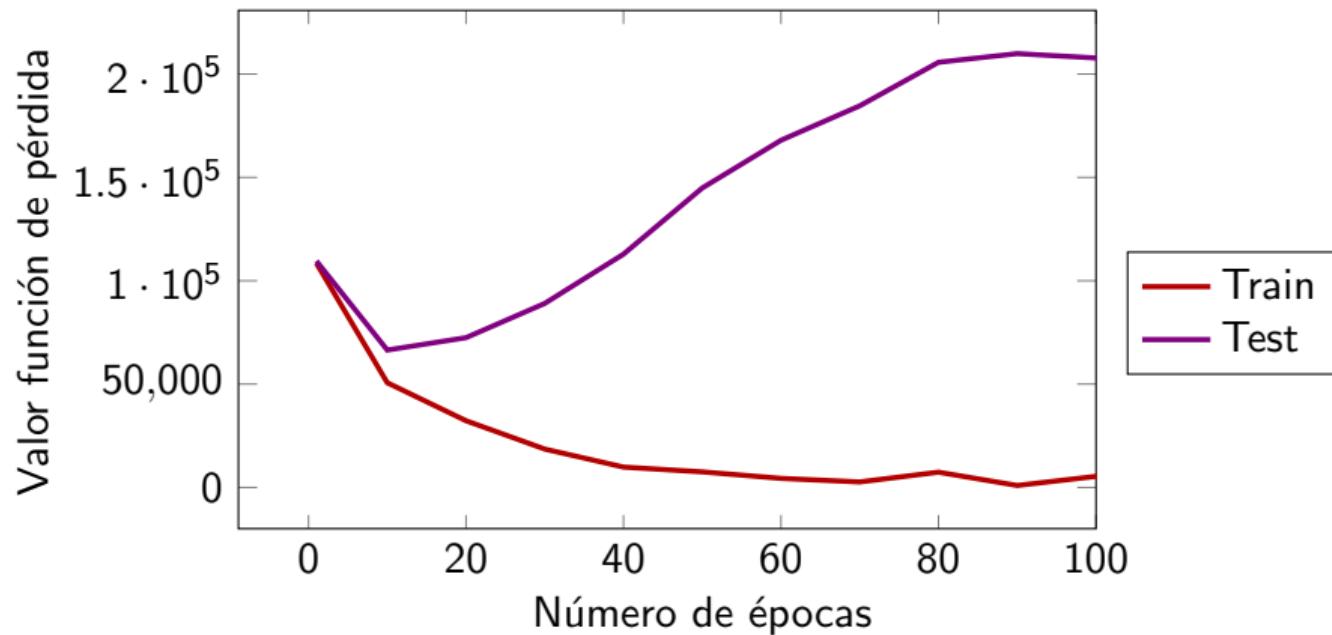
Generalización



Modelo más grande: 97% train pero 52% test :(

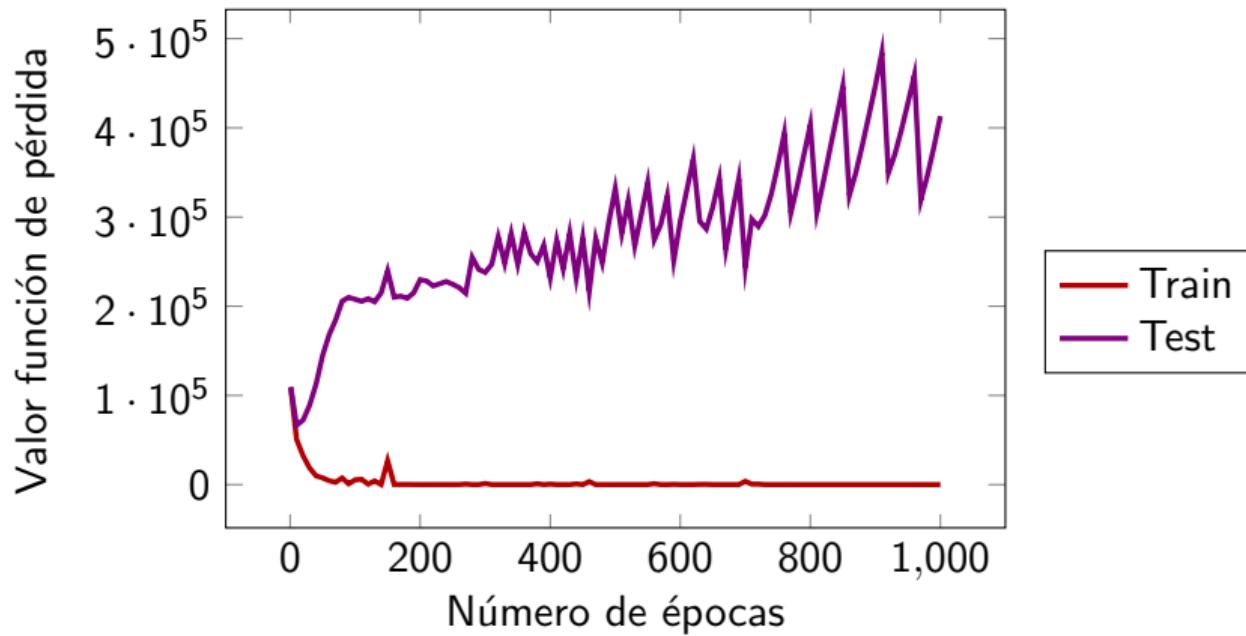
Generalización

Les presento a overfitting 😱



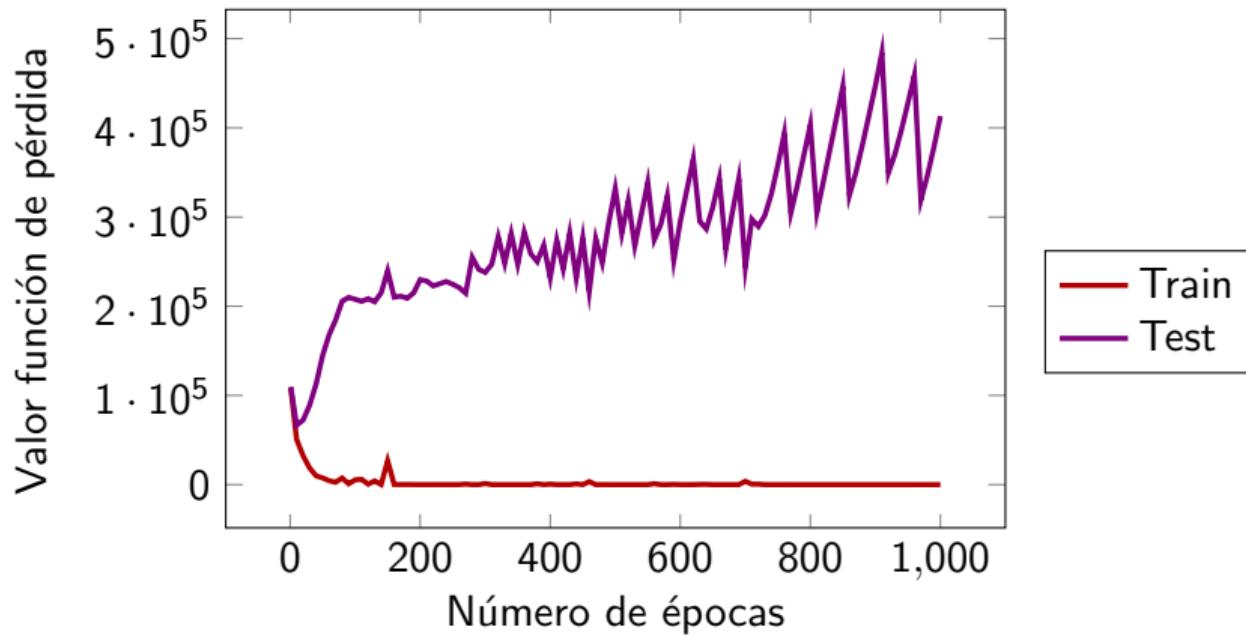
Generalización

Les presento a overfitting 😱



Generalización

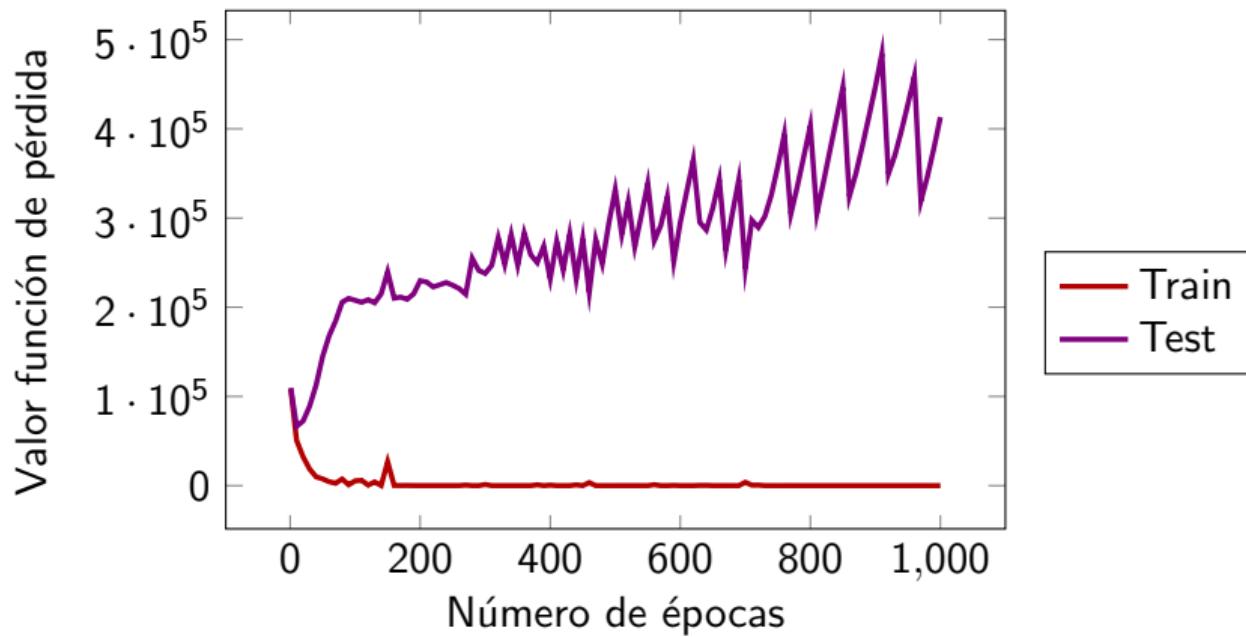
Les presento a overfitting 😱



¿Cómo podemos lograr que nuestro modelo generalice mejor?

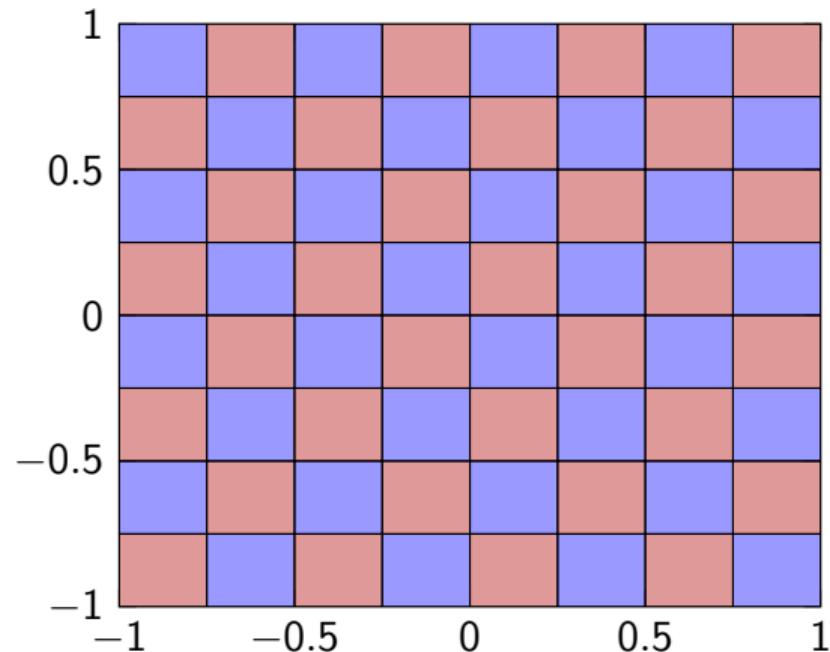
Generalización

Les presento a overfitting 😱



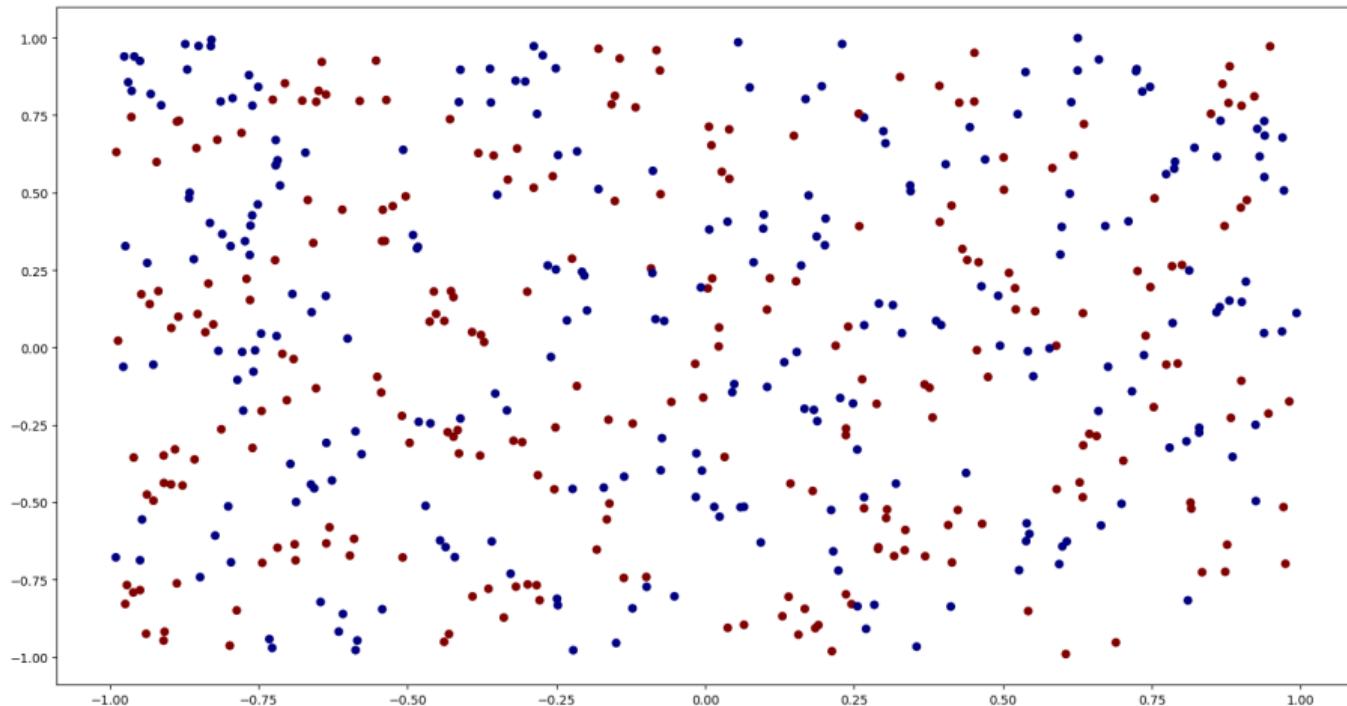
Opción 1: Agregando más ejemplos de entrenamiento.

Generalización



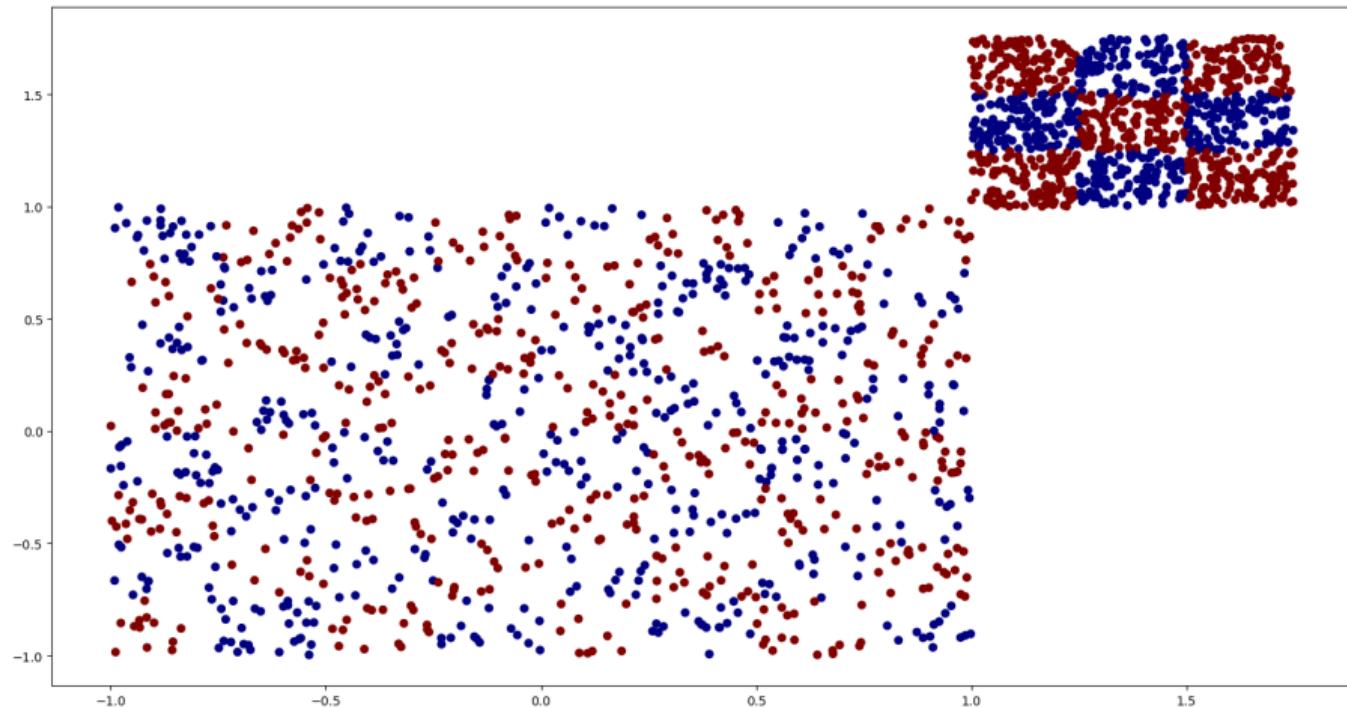
Generalización

Ejemplos de entrenamiento:



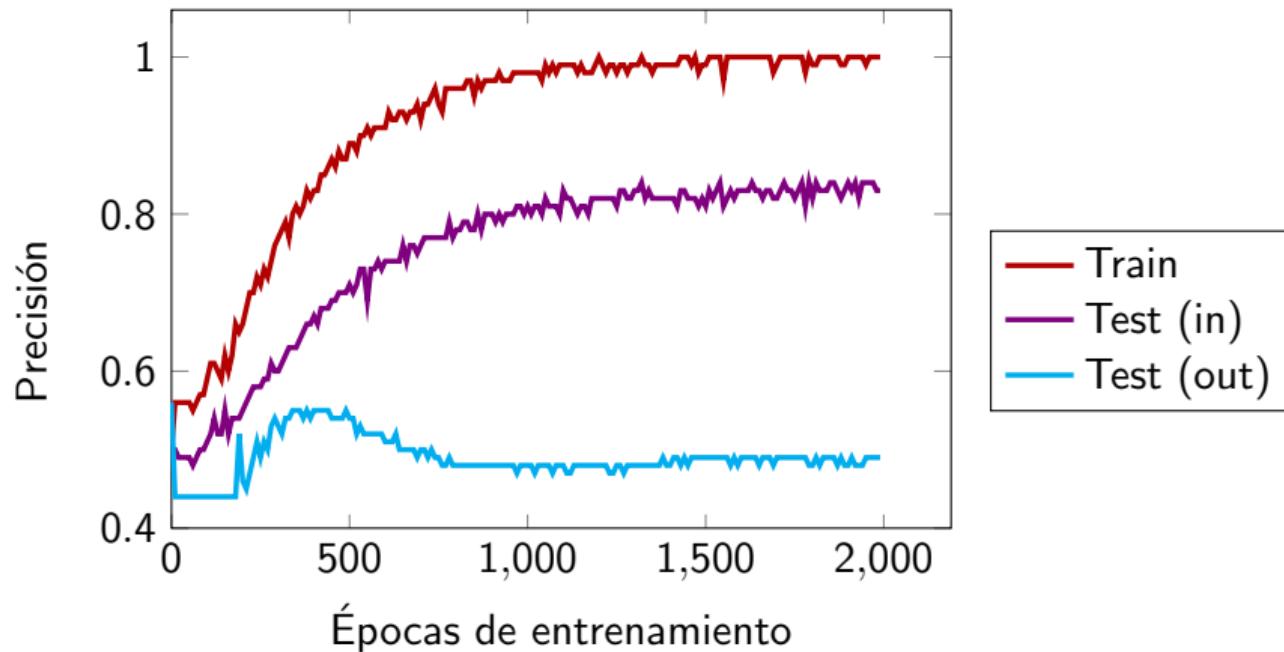
Generalización

Ejemplos de testeo:



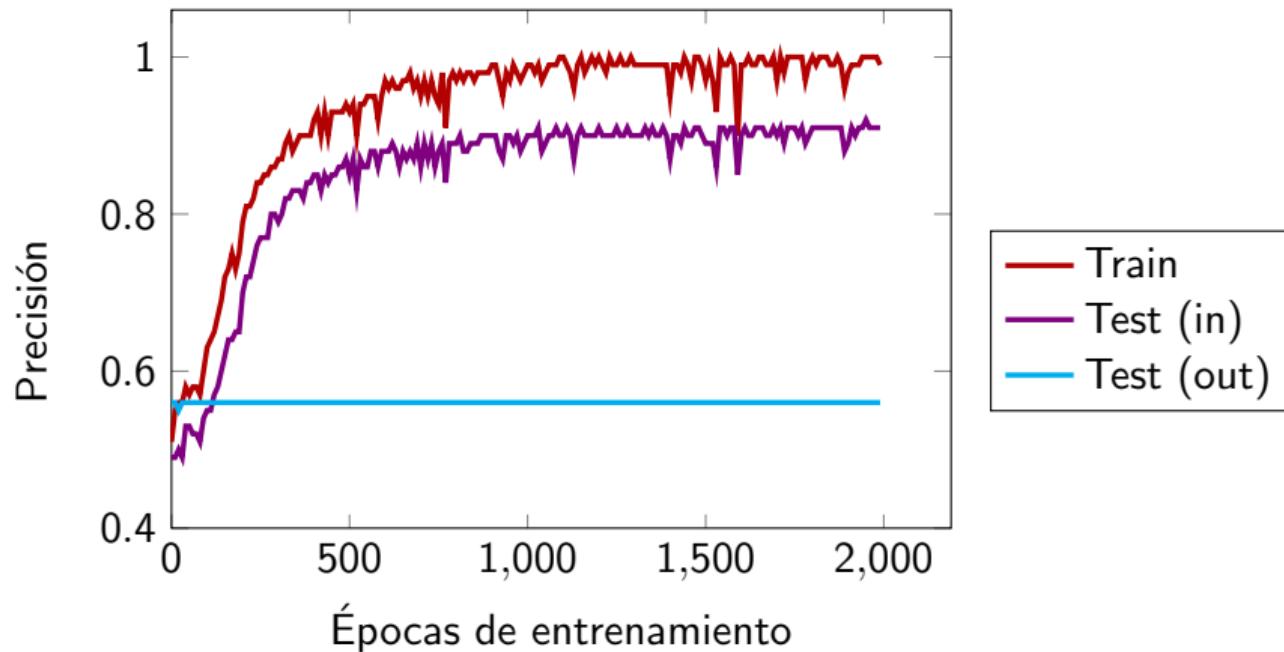
Generalización

Resultados usando 500 ejemplos de entrenamiento:



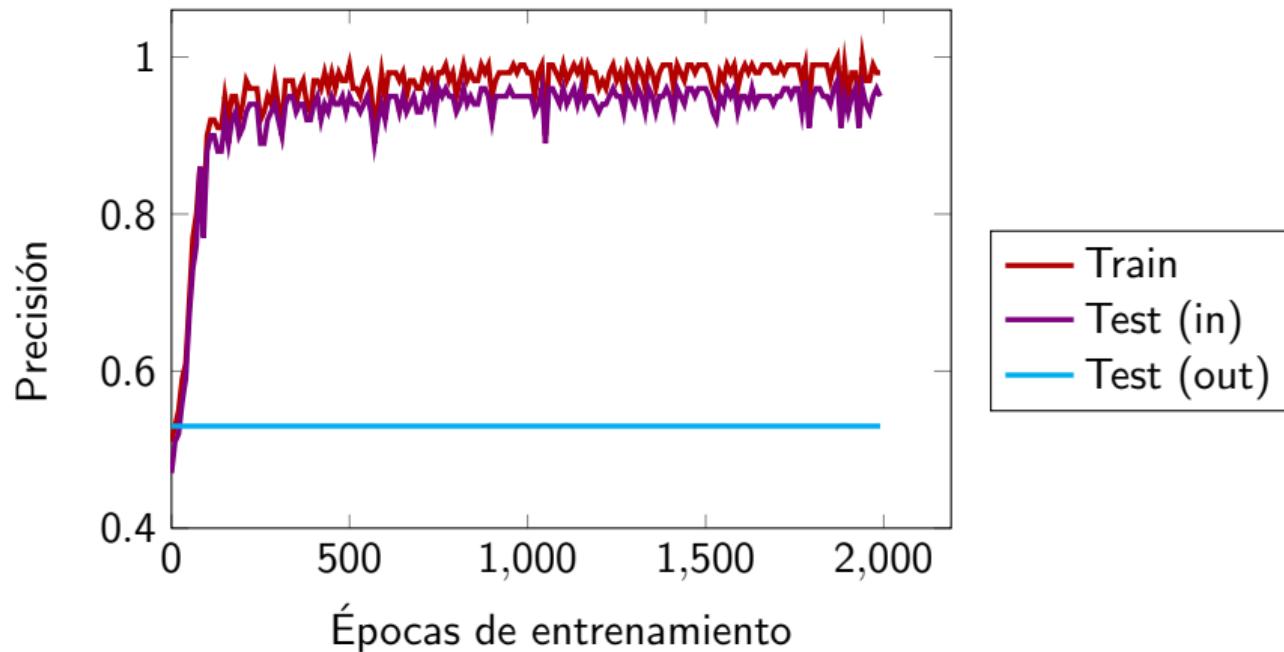
Generalización

Resultados usando 1000 ejemplos de entrenamiento:

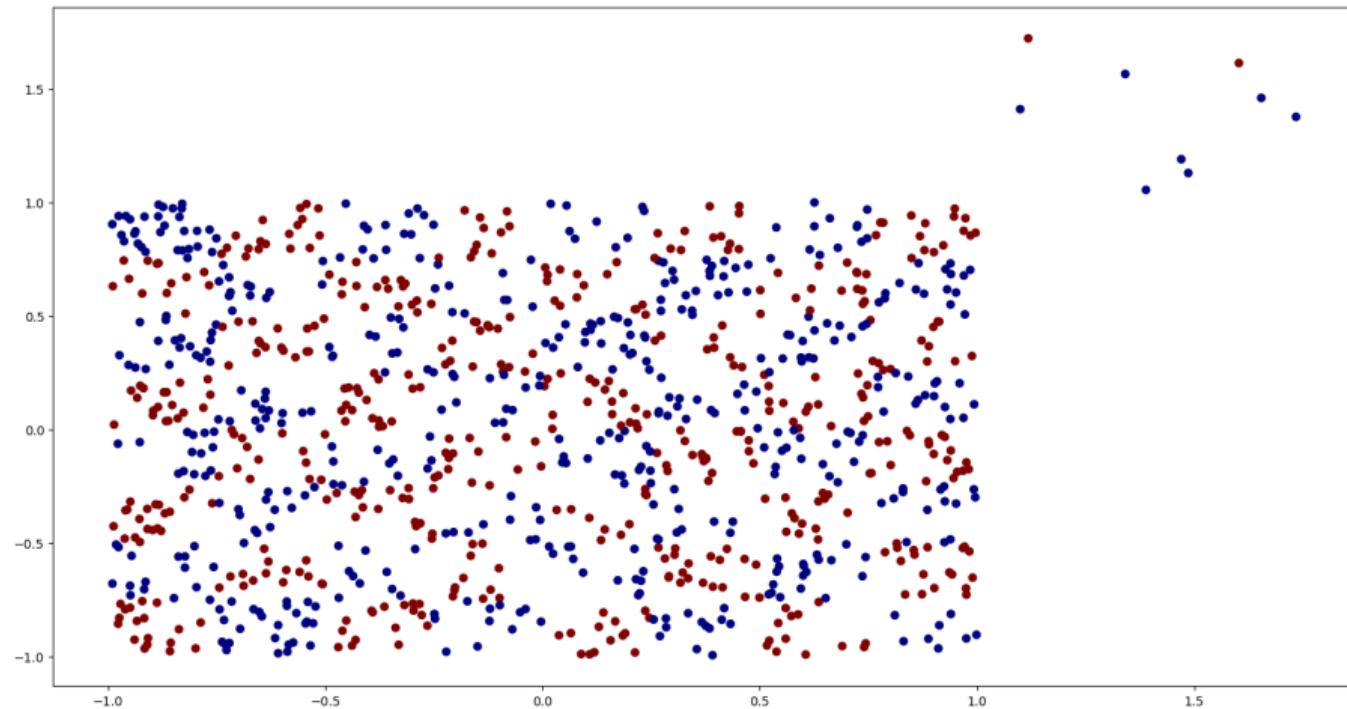


Generalización

Resultados usando 5000 ejemplos de entrenamiento:

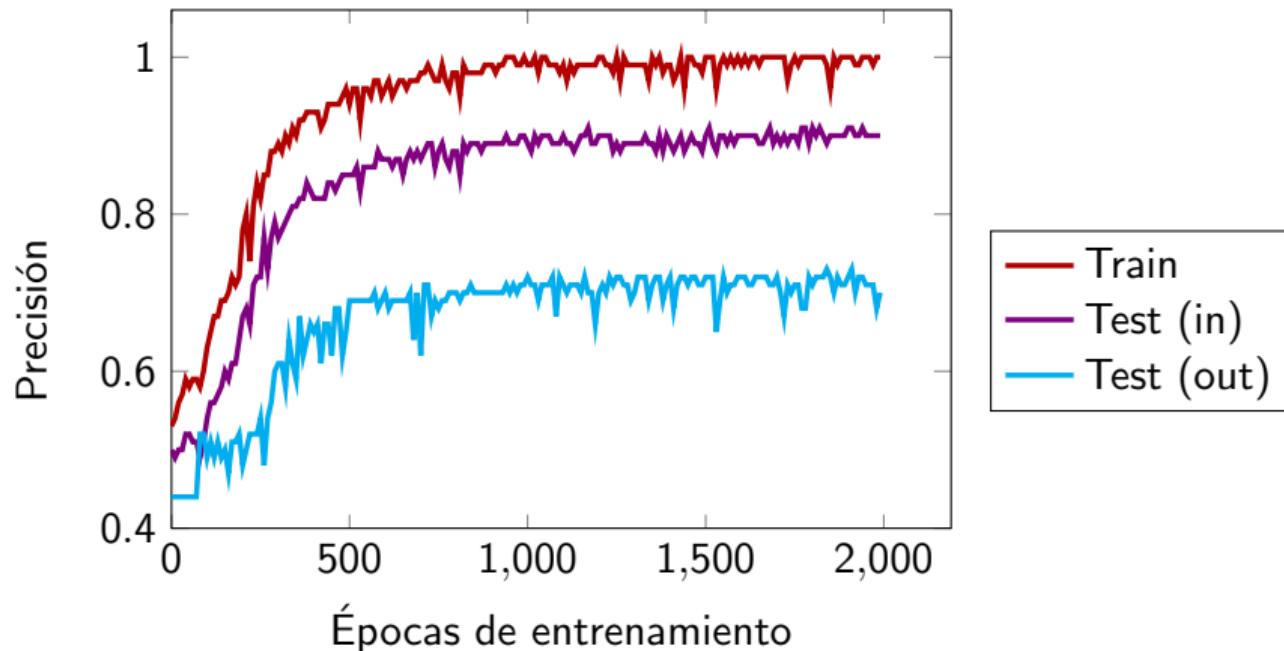


Generalización



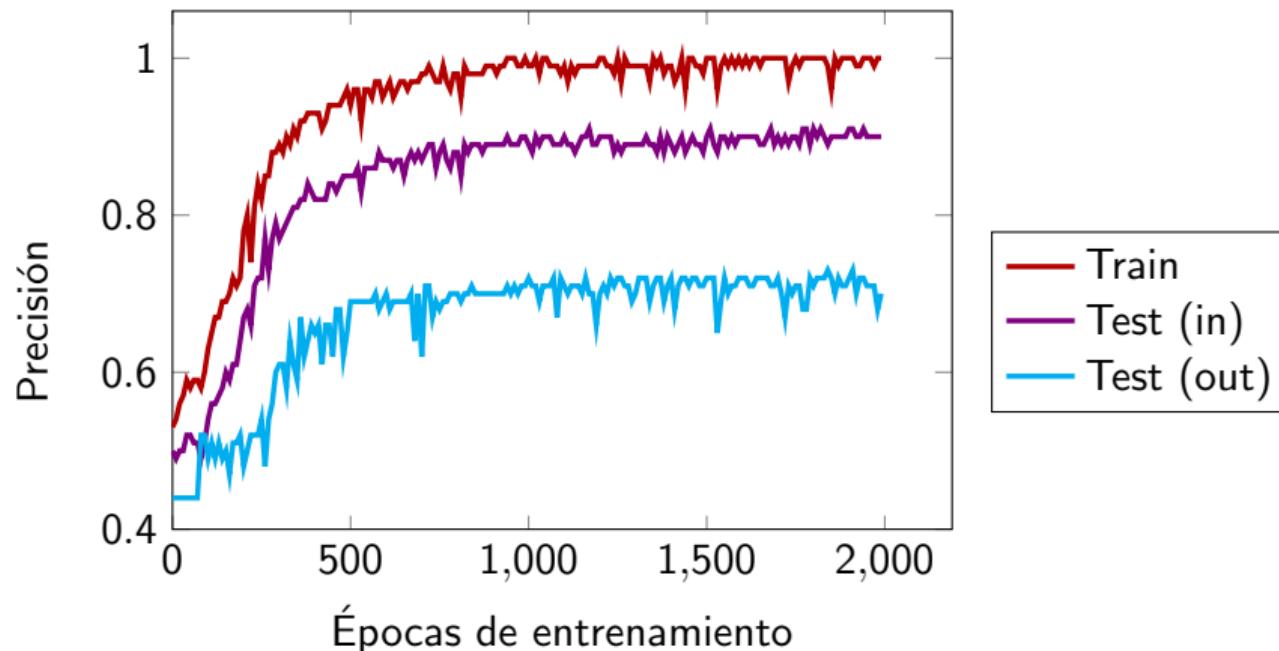
Generalización

Resultados usando 1000 ejemplos de entrenamiento + 9 ejemplos OOD.



Generalización

Resultados usando 1000 ejemplos de entrenamiento + 9 ejemplos OOD.



¿... y si no tenemos más datos? ¿hay algo que podamos hacer?

En la clase de hoy

En la clase de hoy:

- Regularizadores.
- Redes convolucionales.

Regularizadores

Regularizadores

Los regularizadores son:

“Any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

— Deep Learning by Goodfellow et al.

Regularizadores

Los regularizadores son:

“Any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

— Deep Learning by Goodfellow et al.

Es decir, los regularizadores son medidas que tomamos para generalizar mejor:

- No siempre funcionan... y a veces produce mejoras despreciables.
- Pero otras veces pueden hacer una gran diferencia.
- Hoy veremos los regularizadores más importantes.

Regularizadores

Los regularizadores son:

“Any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

— Deep Learning by Goodfellow et al.

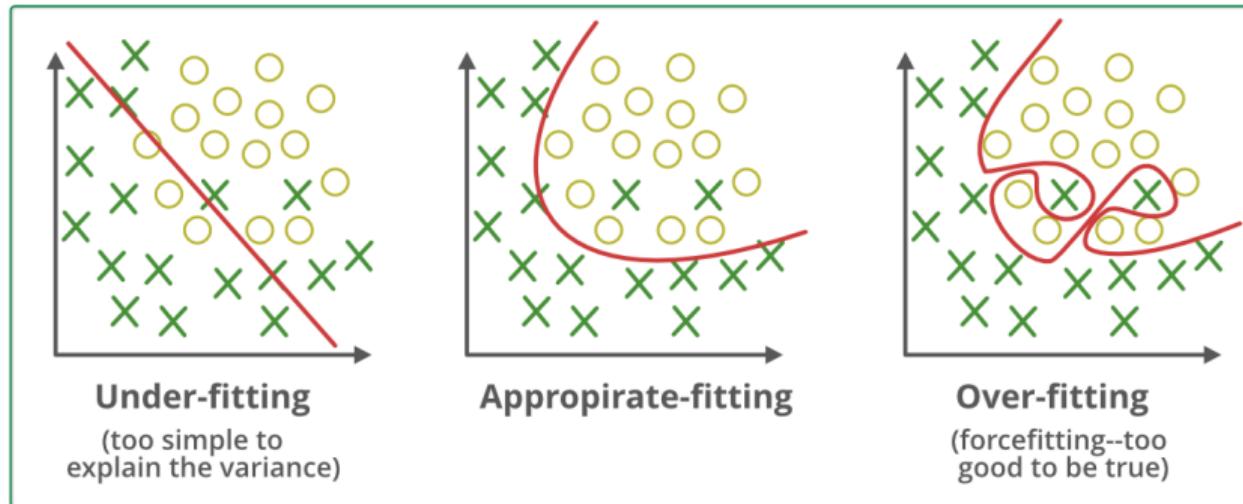
Es decir, los regularizadores son medidas que tomamos para generalizar mejor:

- No siempre funcionan... y a veces produce mejoras despreciables.
- Pero otras veces pueden hacer una gran diferencia.
- Hoy veremos los regularizadores más importantes.

Casi todos ellos buscan evitar el overfitting.

¿Por qué se produce el overfitting?

¿Por qué se produce el overfitting?



En deep learning, la frontera se vuelve más compleja a medida que entrenamos.

Regularizadores

Una forma de evitar esto es penalizando soluciones demasiado “*complejas*”.

Regularizadores

Una forma de evitar esto es penalizando soluciones demasiado “*complejas*”.

Hasta ahora, nuestras funciones de pérdida solo dependen de los datos:

$$J(\theta; \mathcal{T}) = - \sum_{(x,y) \in \mathcal{T}} \mathbf{e}_y^\top \cdot \log(f_\theta(x))$$

Regularizadores

Una forma de evitar esto es penalizando soluciones demasiado “*complejas*”.

Hasta ahora, nuestras funciones de pérdida solo dependen de los datos:

$$J(\theta; \mathcal{T}) = - \sum_{(x,y) \in \mathcal{T}} \mathbf{e}_y^\top \cdot \log(f_\theta(x))$$

Pero podemos agregar más cosas, como una penalización a soluciones complejas:

$$\hat{J}(\theta; \mathcal{T}) = J(\theta; \mathcal{T}) + \lambda \Omega(\theta)$$

Regularizadores

Una forma de evitar esto es penalizando soluciones demasiado “*complejas*”.

Hasta ahora, nuestras funciones de pérdida solo dependen de los datos:

$$J(\theta; \mathcal{T}) = - \sum_{(x,y) \in \mathcal{T}} \mathbf{e}_y^\top \cdot \log(f_\theta(x))$$

Pero podemos agregar más cosas, como una penalización a soluciones complejas:

$$\hat{J}(\theta; \mathcal{T}) = J(\theta; \mathcal{T}) + \lambda \Omega(\theta)$$

El regularizador más utilizado de este estilo es la norma L^2 :

$$\Omega(\theta) = \sum_{w \in \theta} w^2$$

Regularizadores

Por ejemplo, al agregar L^2 a nuestra pérdida en CIFAR-10 queda lo siguiente:

$$\hat{J}(\theta; \mathcal{T}) = - \sum_{(x,y) \in \mathcal{T}} \mathbf{e}_y^\top \cdot \log(f_\theta(x)) + \lambda \sum_{w \in \theta} w^2$$

... esta penalización debería evitar que los pesos de la red crezcan demasiado.

Regularizadores

Por ejemplo, al agregar L^2 a nuestra pérdida en CIFAR-10 queda lo siguiente:

$$\hat{J}(\theta; \mathcal{T}) = - \sum_{(x,y) \in \mathcal{T}} \mathbf{e}_y^\top \cdot \log(f_\theta(x)) + \lambda \sum_{w \in \theta} w^2$$

... esta penalización debería evitar que los pesos de la red crezcan demasiado.

Notar también lo siguiente:

- Si $\lambda = 0$, volvemos al objetivo inicial sin regularizador.
- Si λ es alto, la solución óptima podría ser setear todos los pesos a cero.

Regularizadores

Por ejemplo, al agregar L^2 a nuestra pérdida en CIFAR-10 queda lo siguiente:

$$\hat{J}(\theta; \mathcal{T}) = - \sum_{(x,y) \in \mathcal{T}} \mathbf{e}_y^\top \cdot \log(f_\theta(x)) + \lambda \sum_{w \in \theta} w^2$$

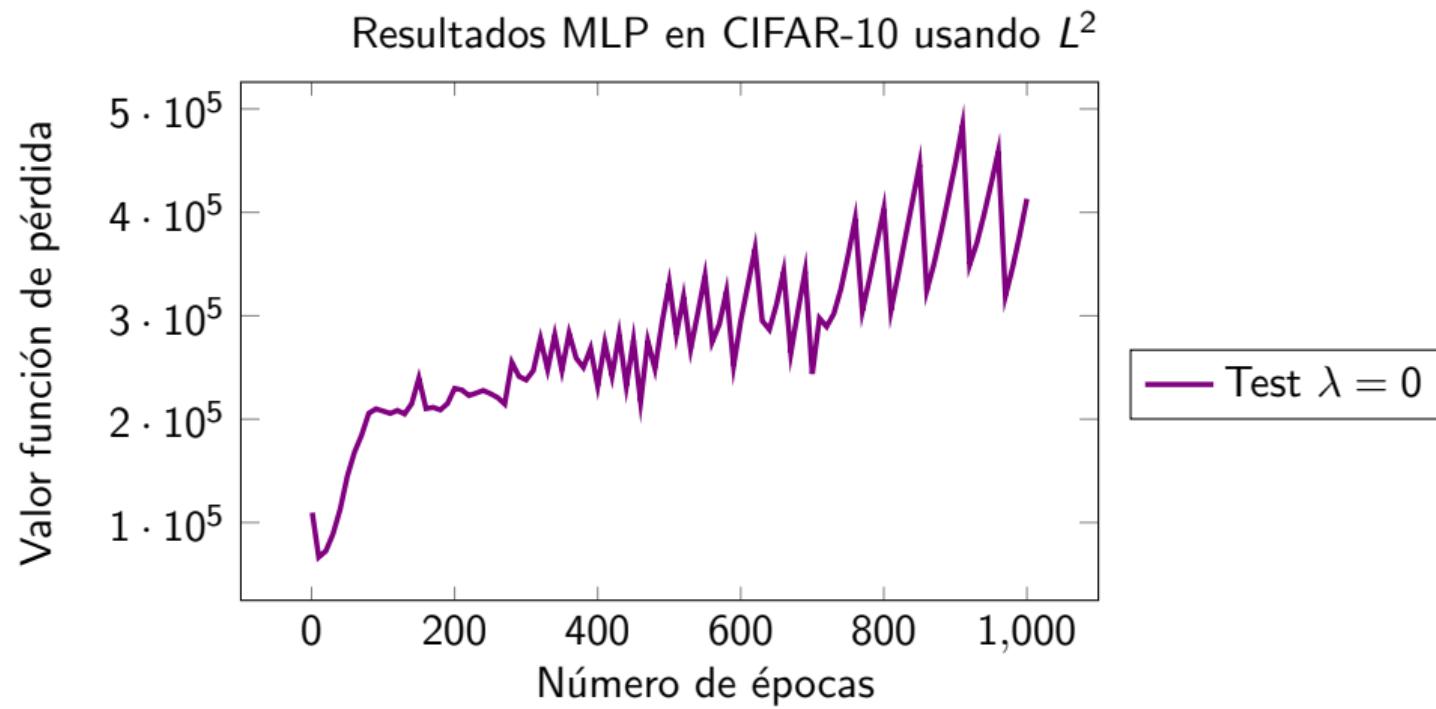
... esta penalización debería evitar que los pesos de la red crezcan demasiado.

Notar también lo siguiente:

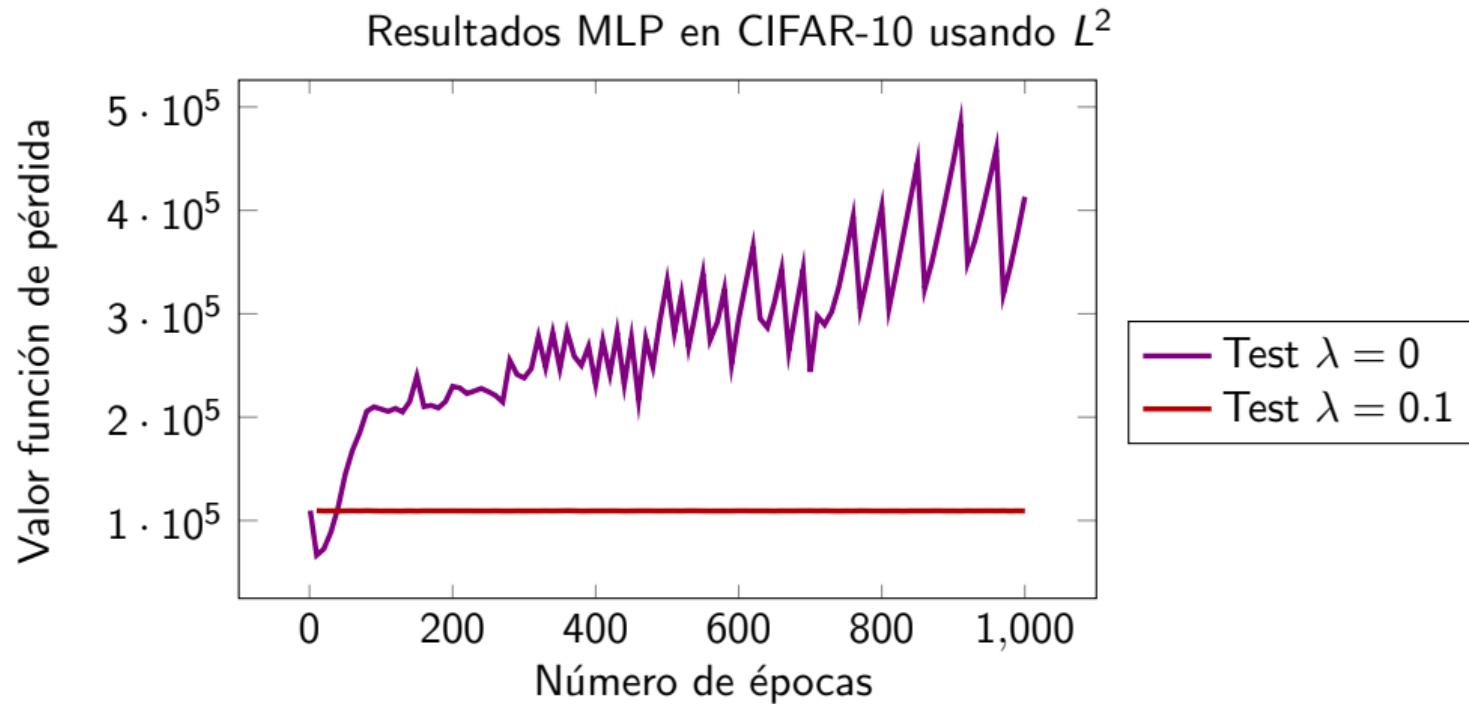
- Si $\lambda = 0$, volvemos al objetivo inicial sin regularizador.
- Si λ es alto, la solución óptima podría ser setear todos los pesos a cero.

¿Cómo le va experimentalmente a esta función en CIFAR-10?

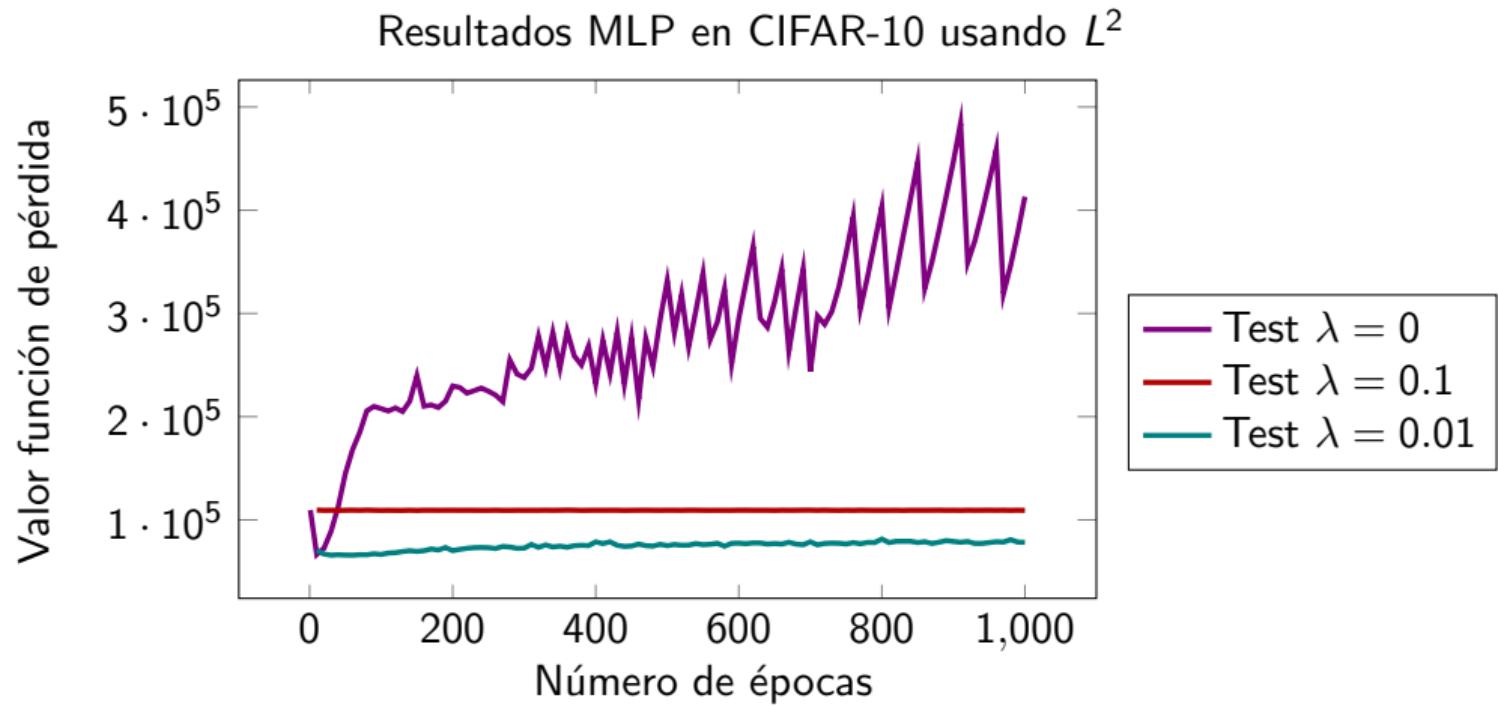
Regularizadores



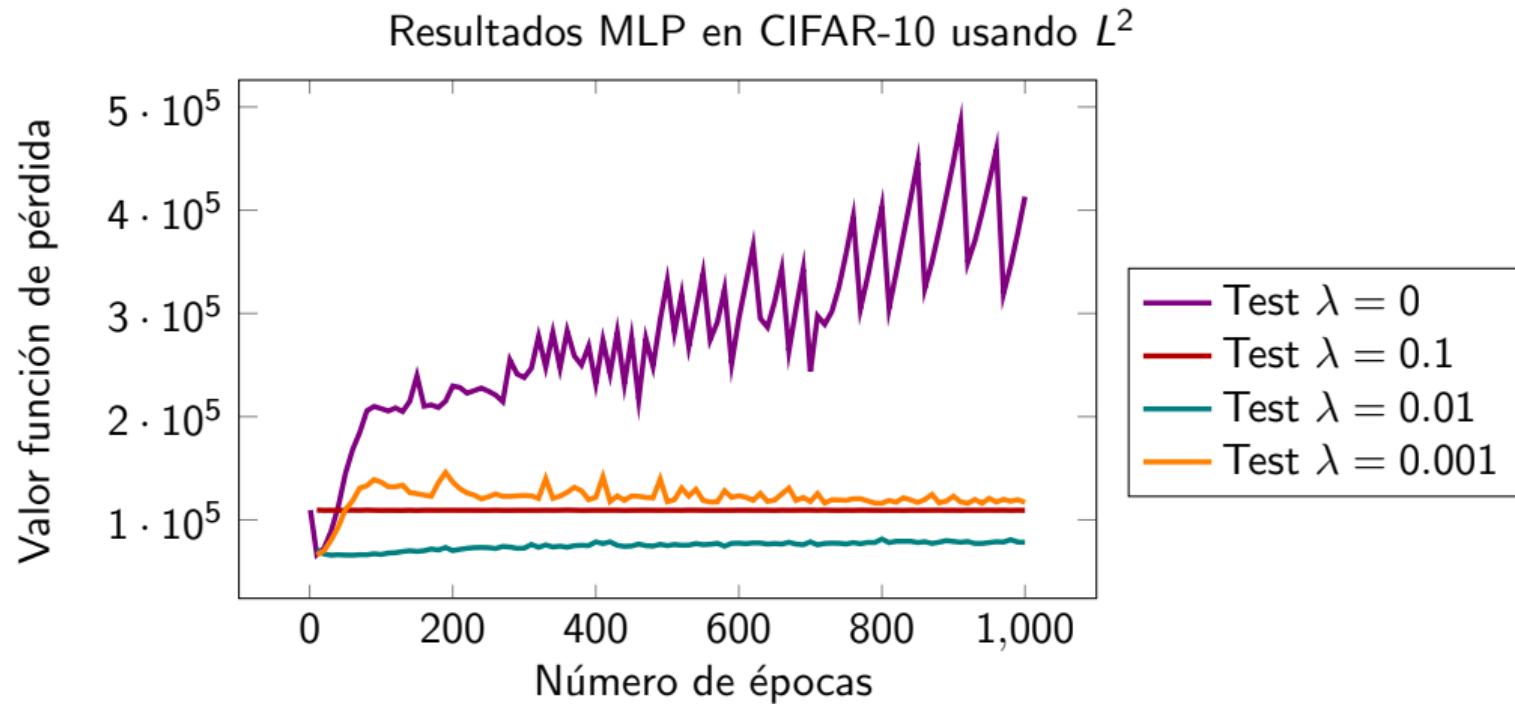
Regularizadores



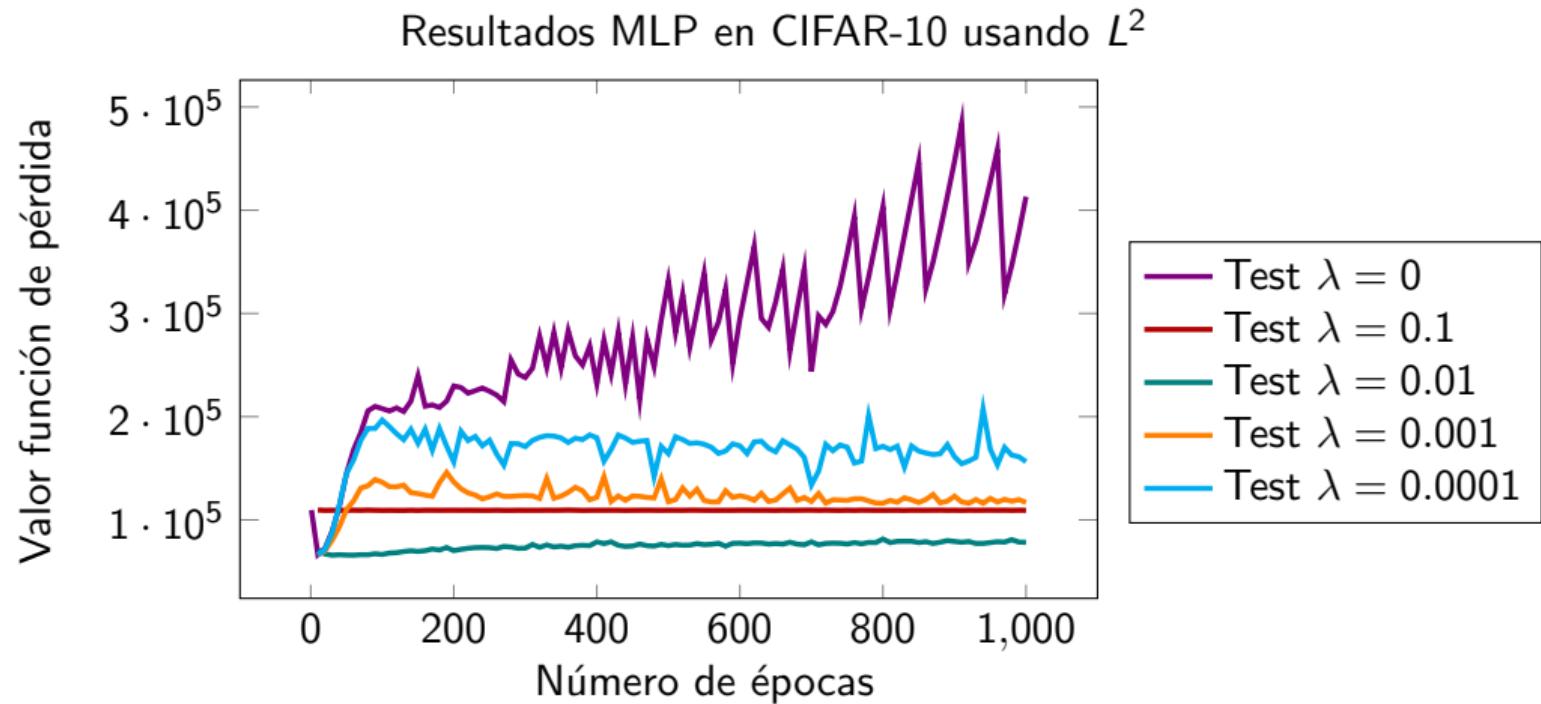
Regularizadores



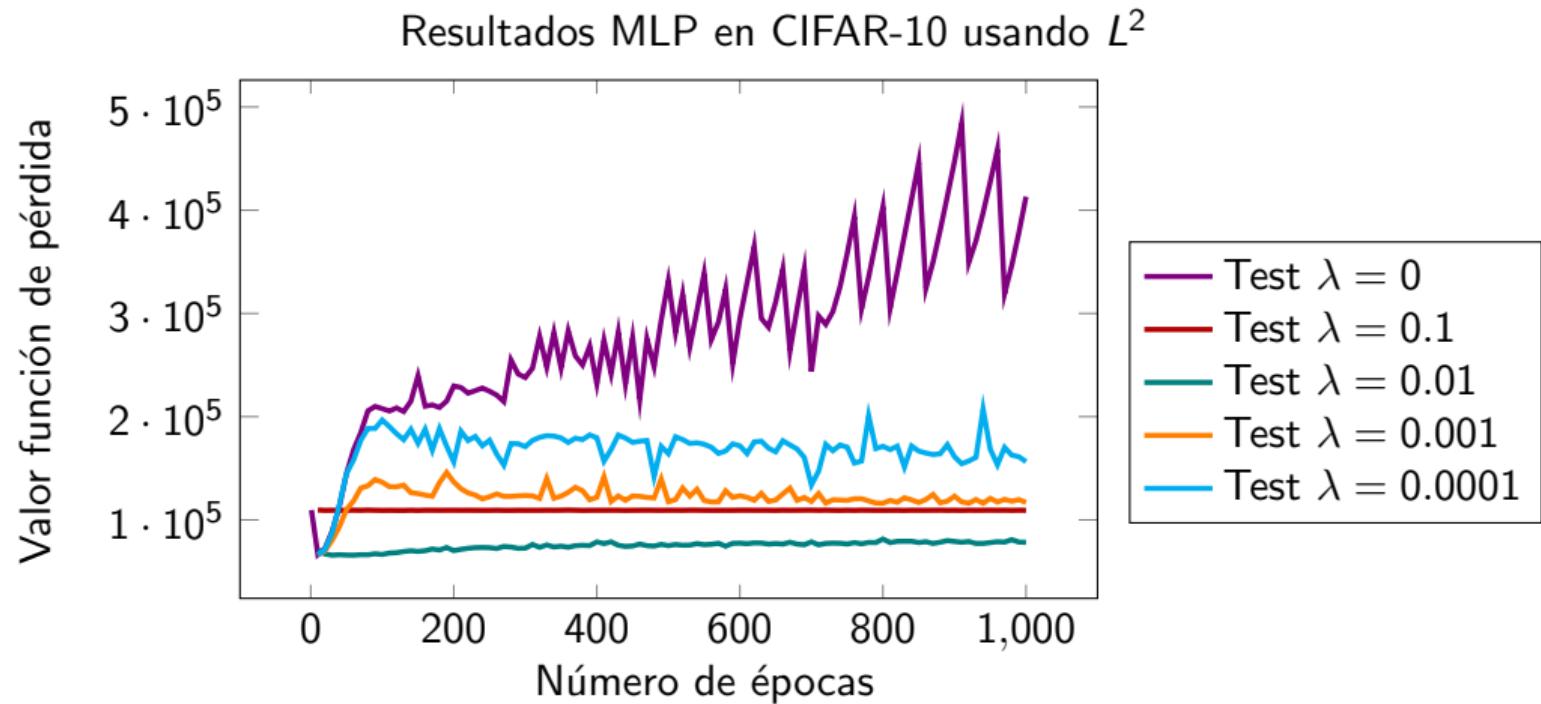
Regularizadores



Regularizadores

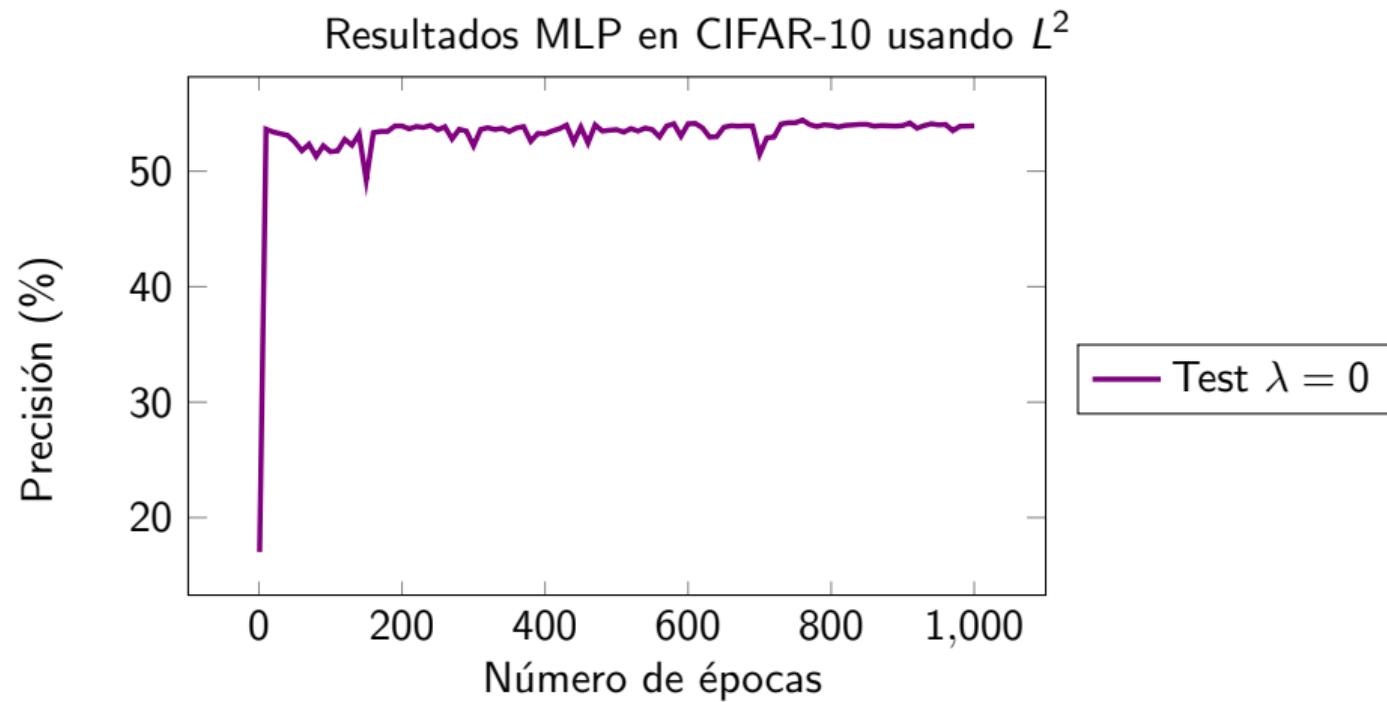


Regularizadores

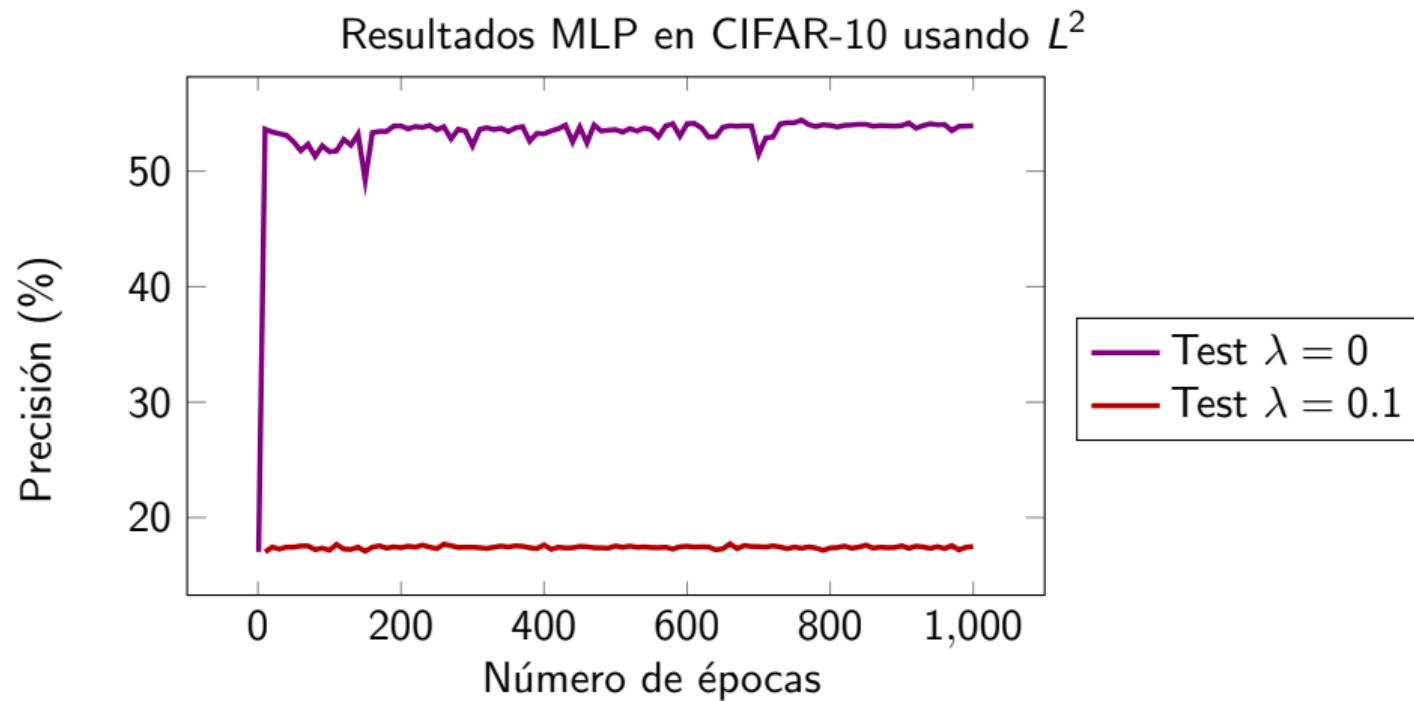


¿Cuál de estas redes generaliza mejor?

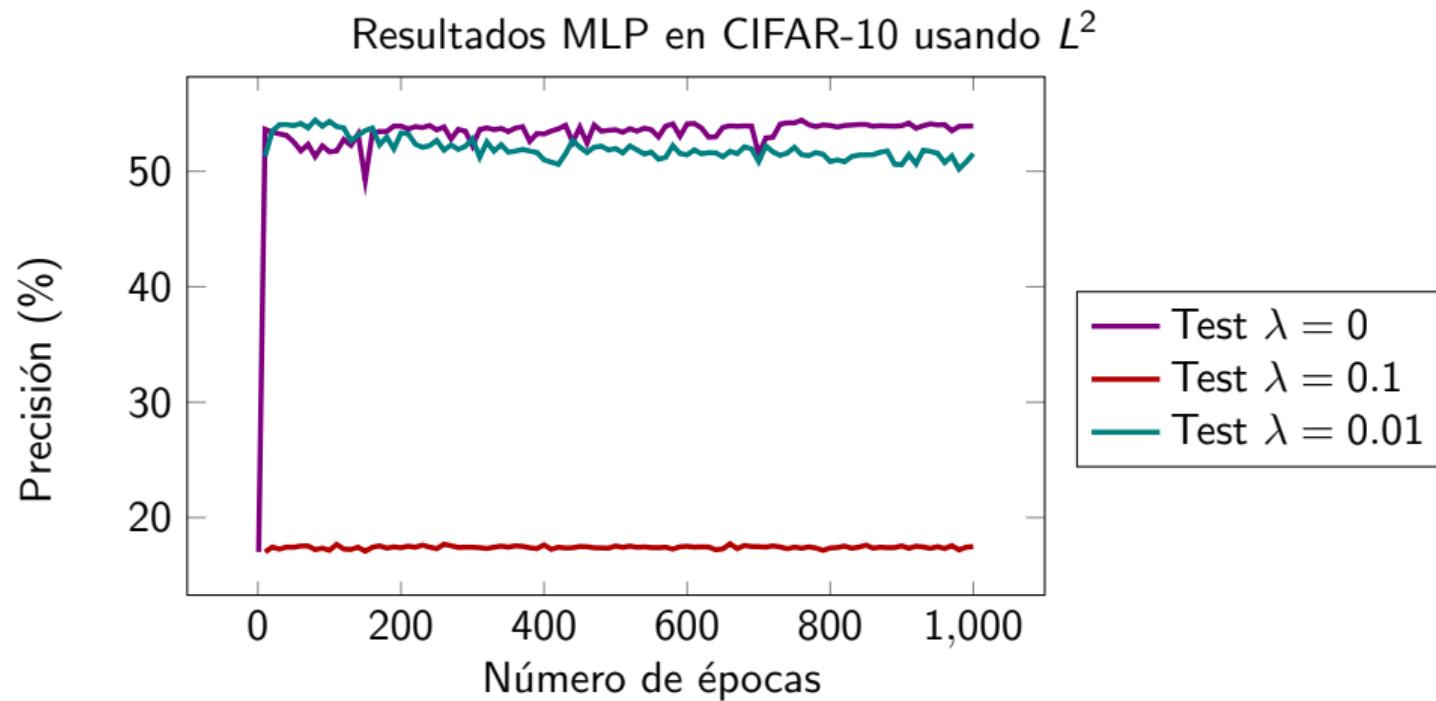
Regularizadores



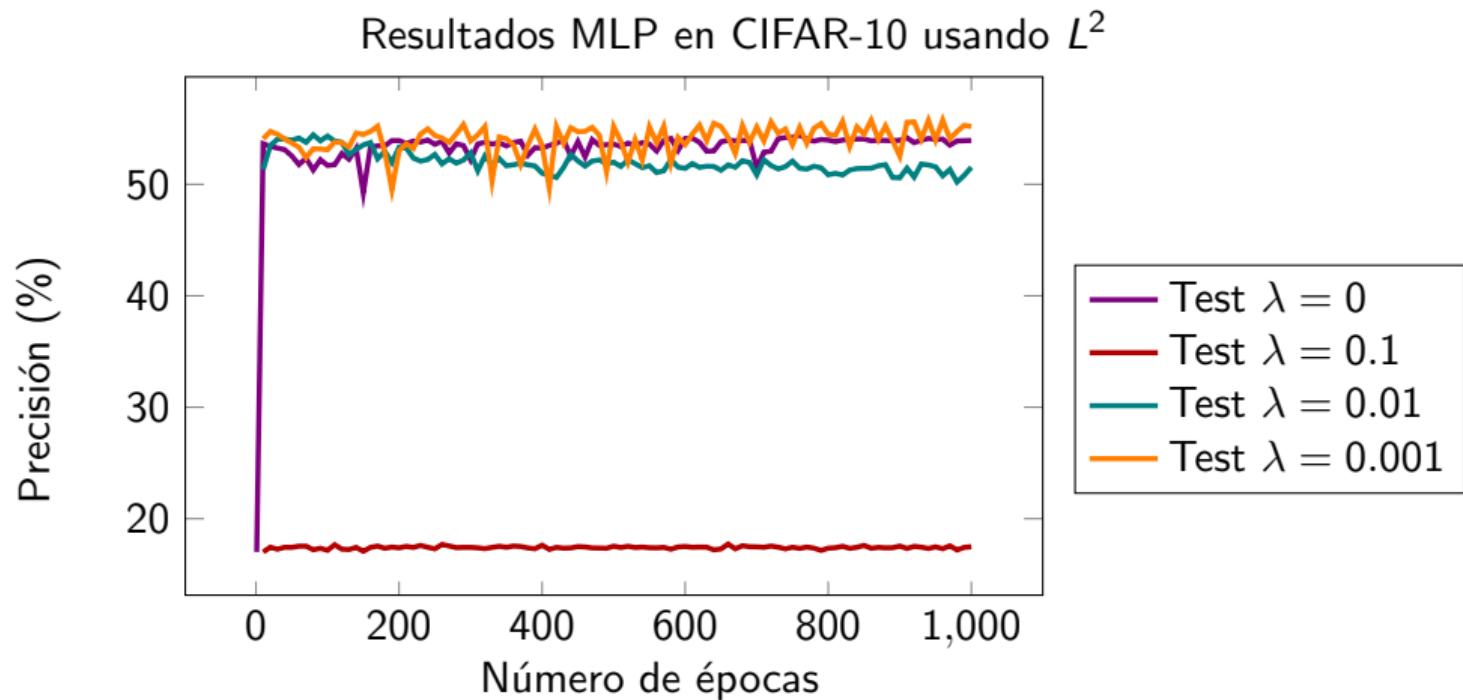
Regularizadores



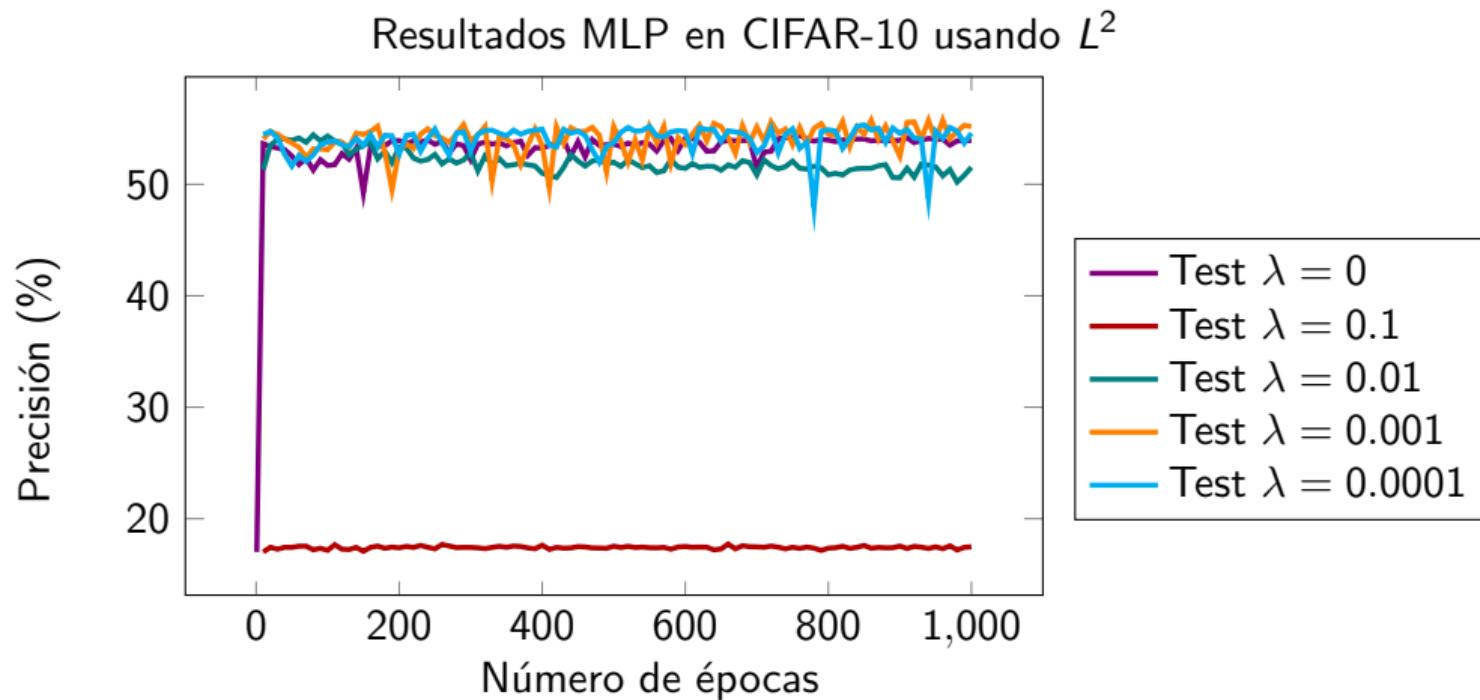
Regularizadores



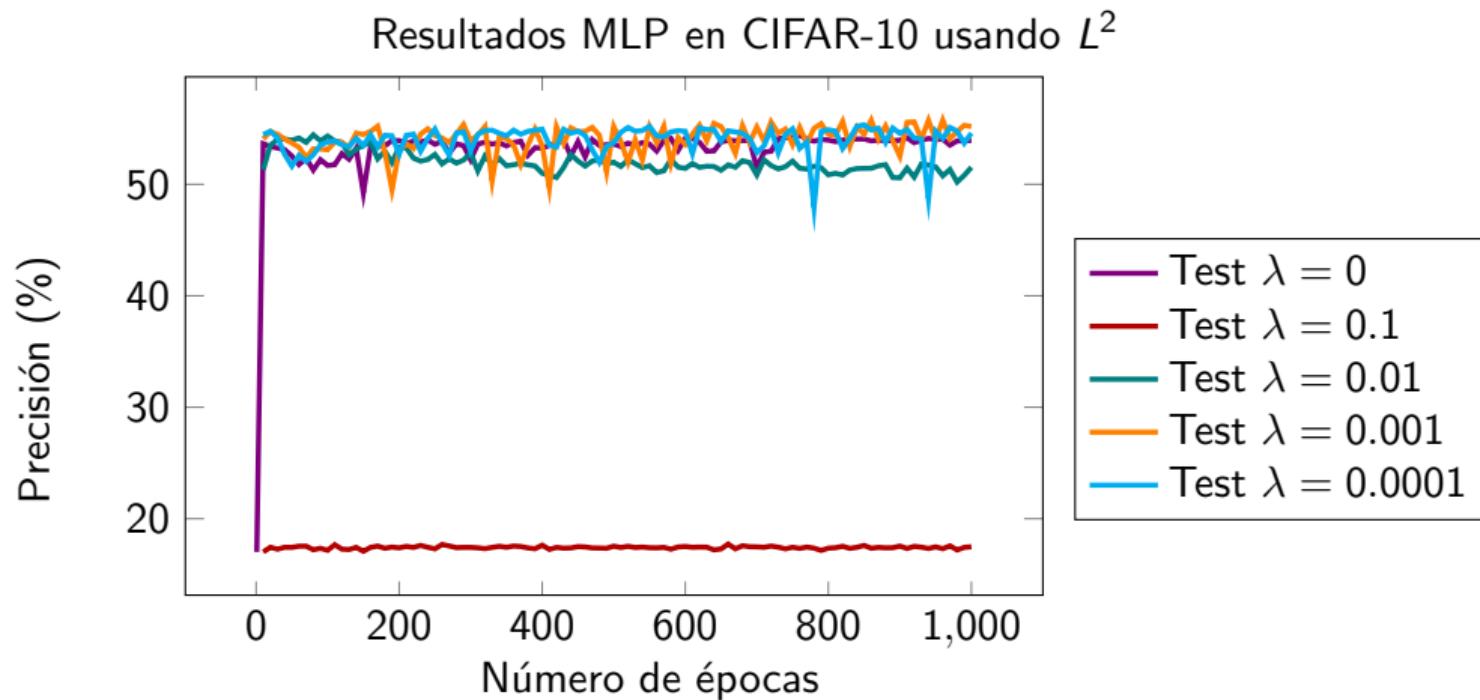
Regularizadores



Regularizadores



Regularizadores



El mejor rendimiento es 55.76% obtenido con $\lambda = 0.001$

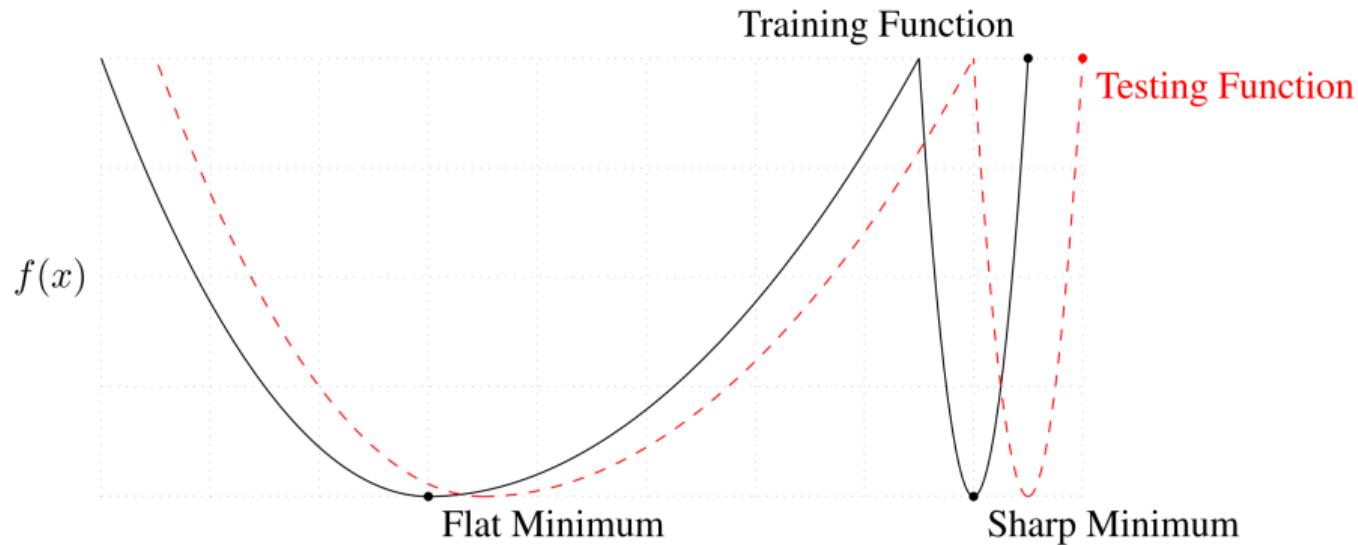
Regularizadores

Otro descubrimiento importante es que deep learning tiende a encontrar soluciones que generalizan mejor cuando se usan mini-batch chicos¹

¹Keskar et al. "On large-batch training for deep learning: Generalization gap and sharp minima." ArXiv (2016).

Regularizadores

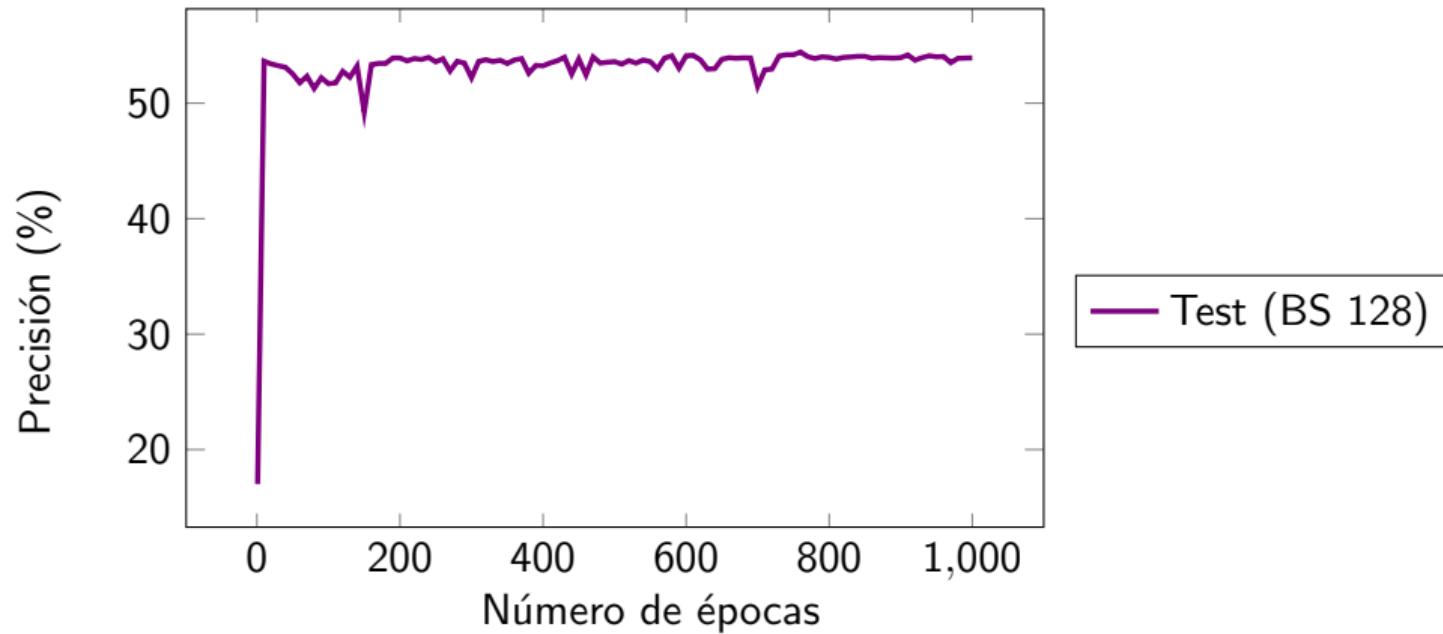
Otro descubrimiento importante es que deep learning tiende a encontrar soluciones que generalizan mejor cuando se usan mini-batch chicos¹



¹Keskar et al. "On large-batch training for deep learning: Generalization gap and sharp minima." ArXiv (2016).

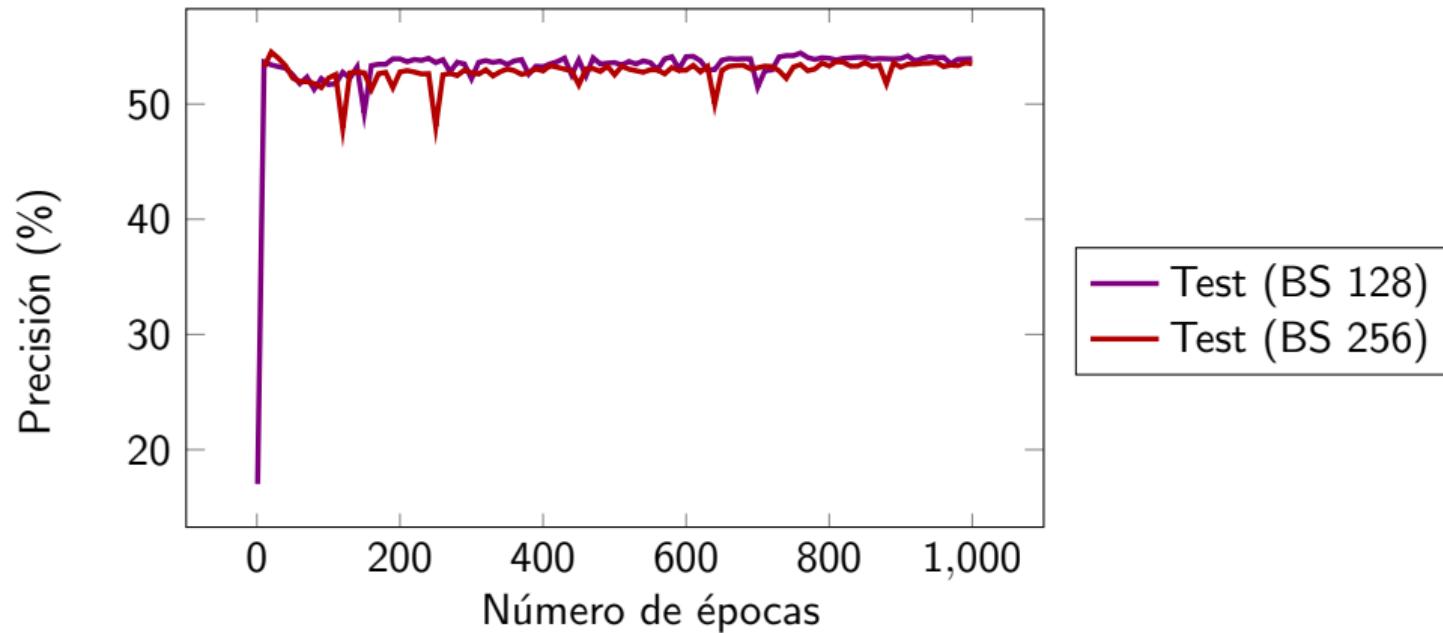
Regularizadores

Resultados MLP en CIFAR-10 usando distintos *batch sizes*.



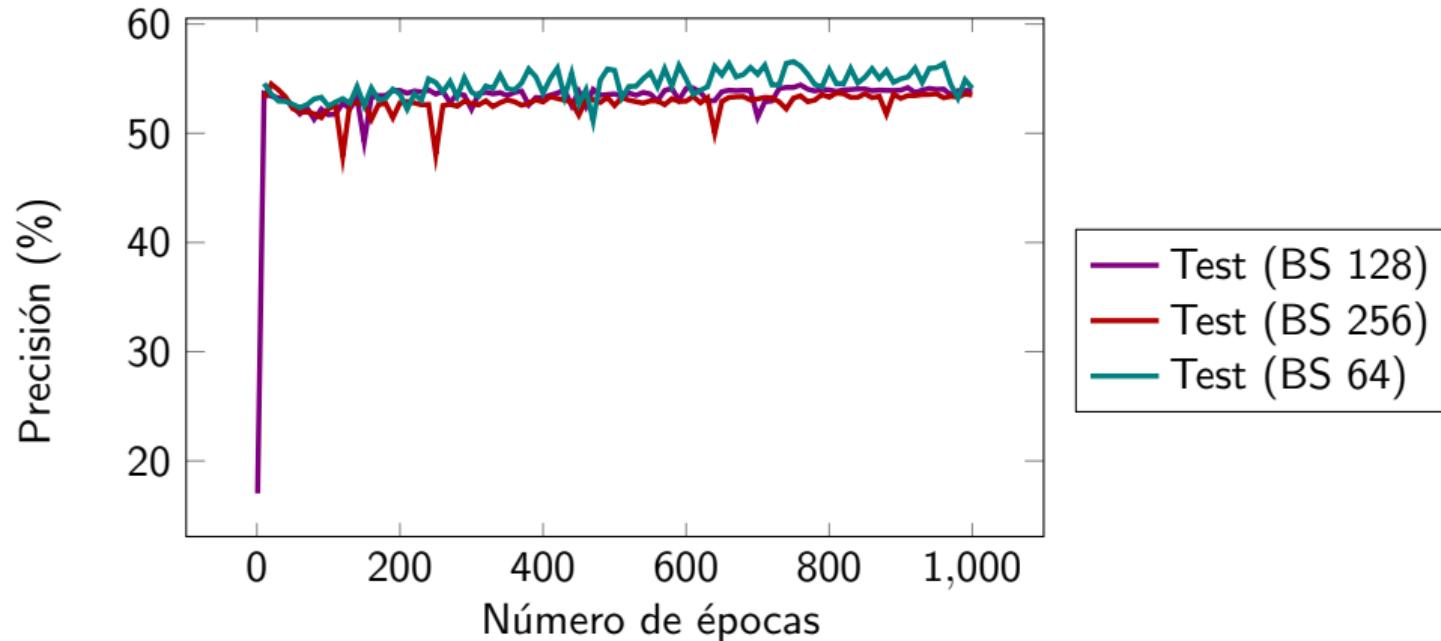
Regularizadores

Resultados MLP en CIFAR-10 usando distintos *batch sizes*.



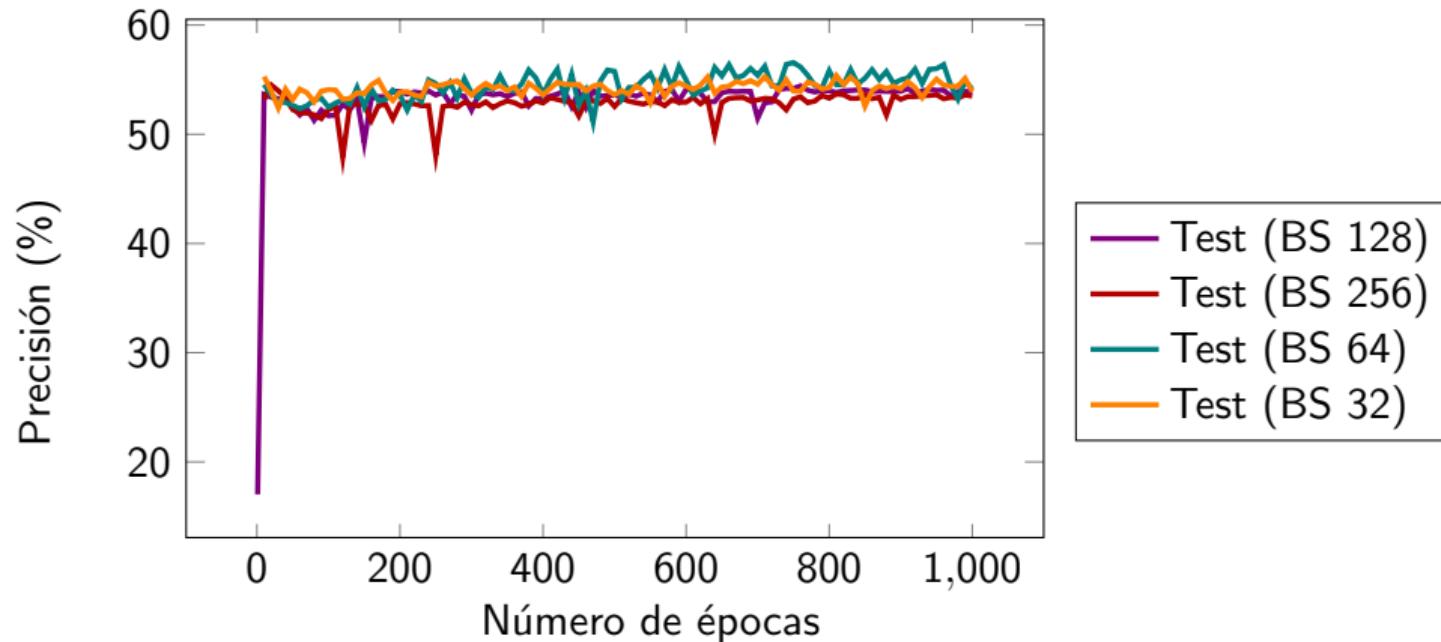
Regularizadores

Resultados MLP en CIFAR-10 usando distintos *batch sizes*.



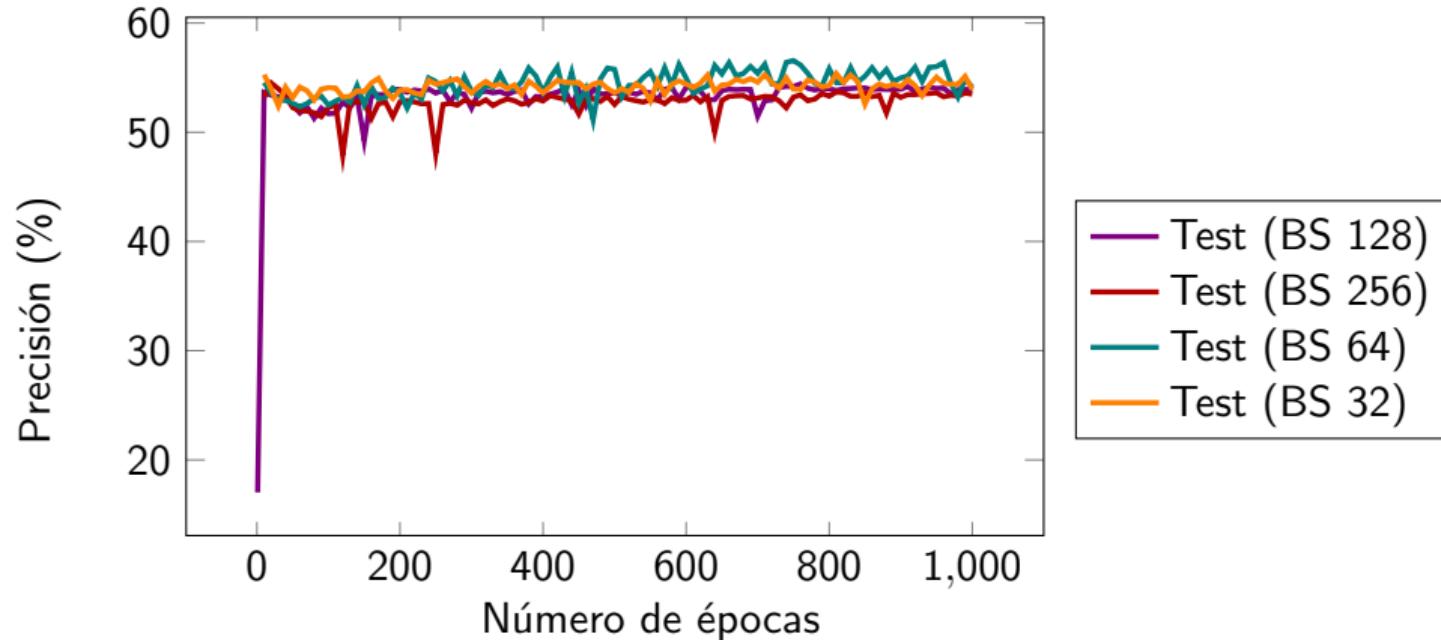
Regularizadores

Resultados MLP en CIFAR-10 usando distintos *batch sizes*.



Regularizadores

Resultados MLP en CIFAR-10 usando distintos *batch sizes*.



El mejor rendimiento es 56.57% obtenido con batch de 64.

Regularizadores

Como pueden ver, estos regularizadores nos permiten ganar un poco más de rendimiento en generalización...

Regularizadores

Como pueden ver, estos regularizadores nos permiten ganar un poco más de rendimiento en generalización... pero no son la gran cosa :(

Regularizadores

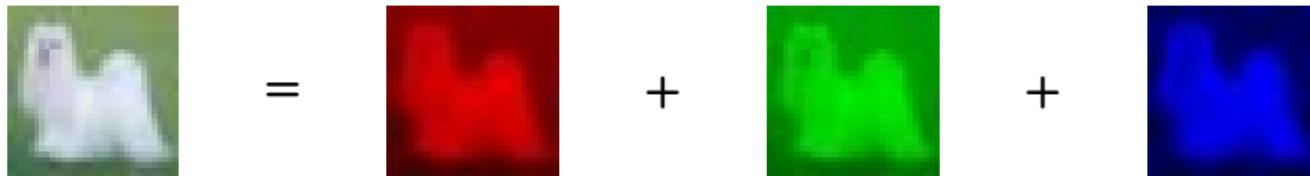
Como pueden ver, estos regularizadores nos permiten ganar un poco más de rendimiento en generalización... pero no son la gran cosa :(

Lo que causó el gran salto fue la idea de introducir “*conocimiento previo*” en la arquitectura de la red.

Redes Convolucionales

Redes Convolucionales

Esta idea de transformar la imagen en un vector es extraña 🤔

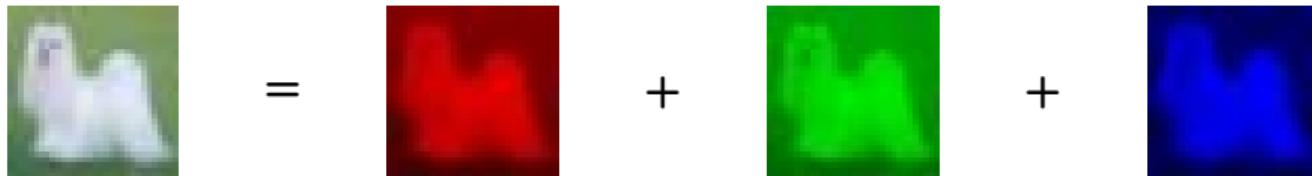


Input: Un vector de $32 \times 32 \times 3$ números entre 0 y 255.



Redes Convolucionales

Esta idea de transformar la imagen en un vector es extraña 🤔



Input: Un vector de $32 \times 32 \times 3$ números entre 0 y 255.

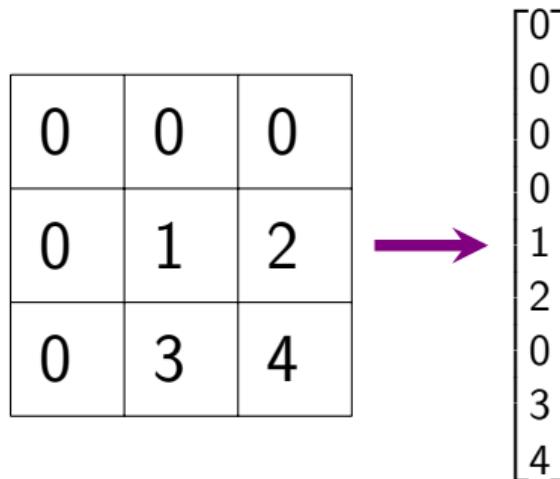


-
- Se pierde mucha de la estructura de la imagen.
 - Si la imagen se mueve un poco, el vector cambia completamente.

Redes Convolucionales

0	0	0
0	1	2
0	3	4

Redes Convolucionales

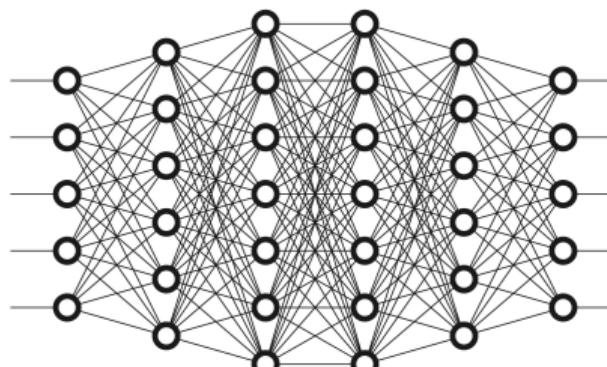


Redes Convolucionales

0	0	0
0	1	2
0	3	4

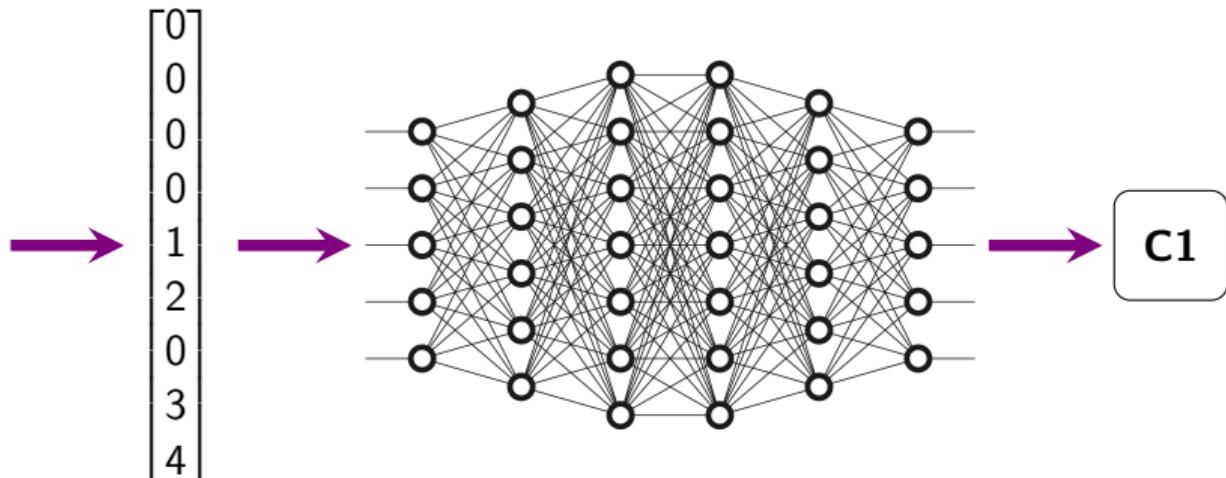


$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 3 & 4 \end{bmatrix}$



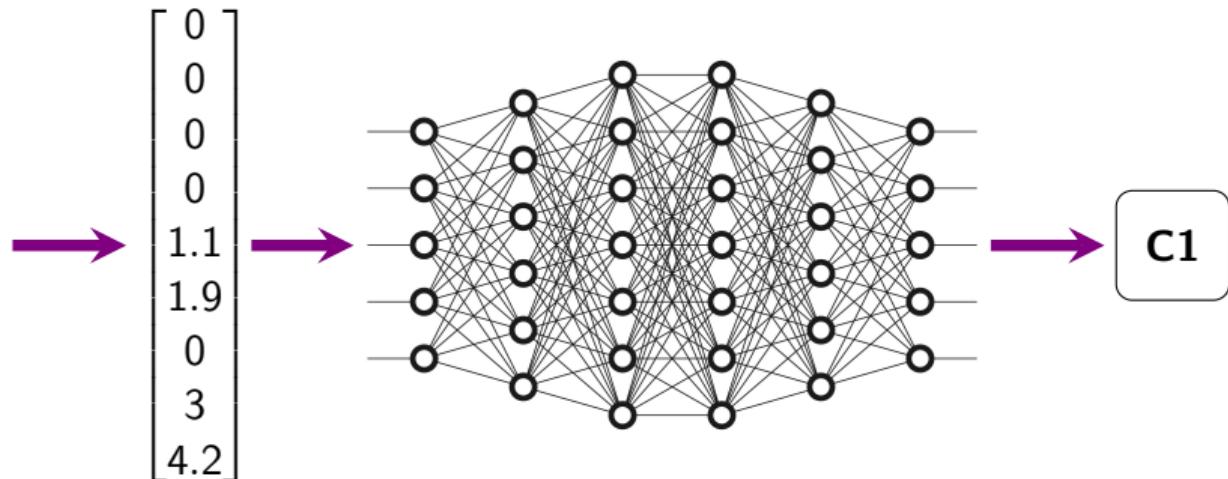
Redes Convolucionales

0	0	0
0	1	2
0	3	4



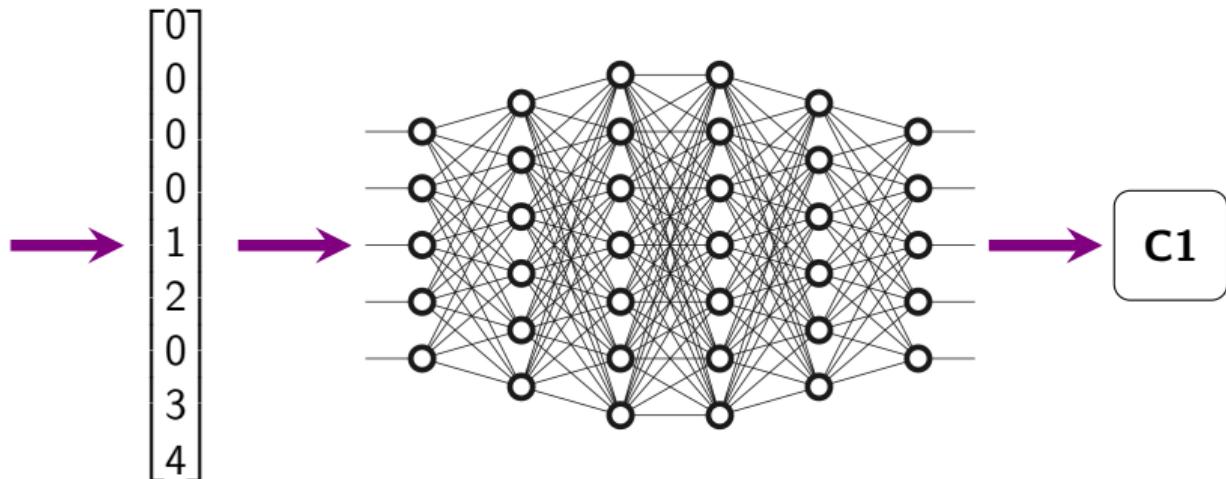
Redes Convolucionales

0	0	0
0	1.1	1.9
0	3	4.2



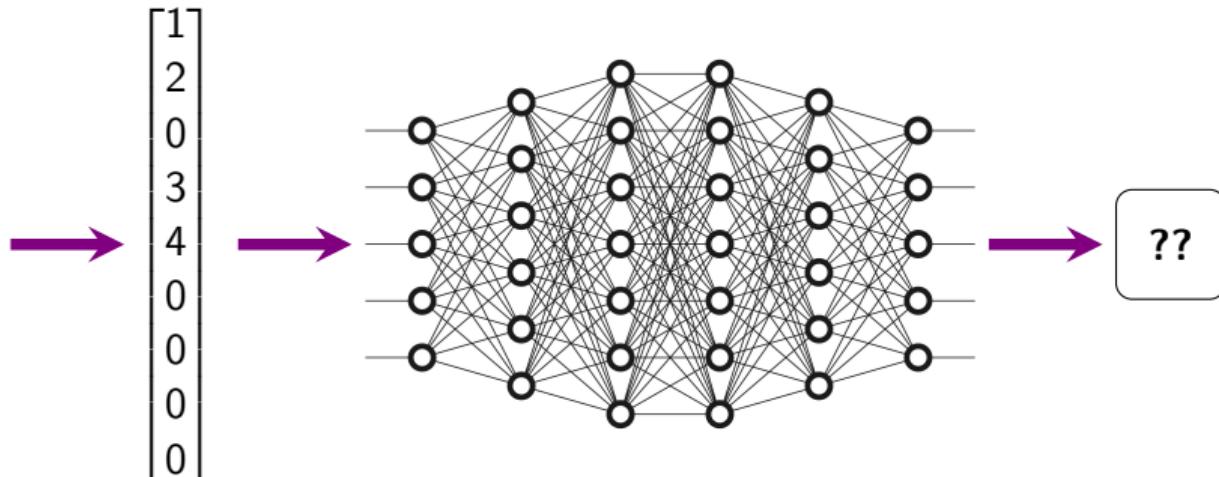
Redes Convolucionales

0	0	0
0	1	2
0	3	4

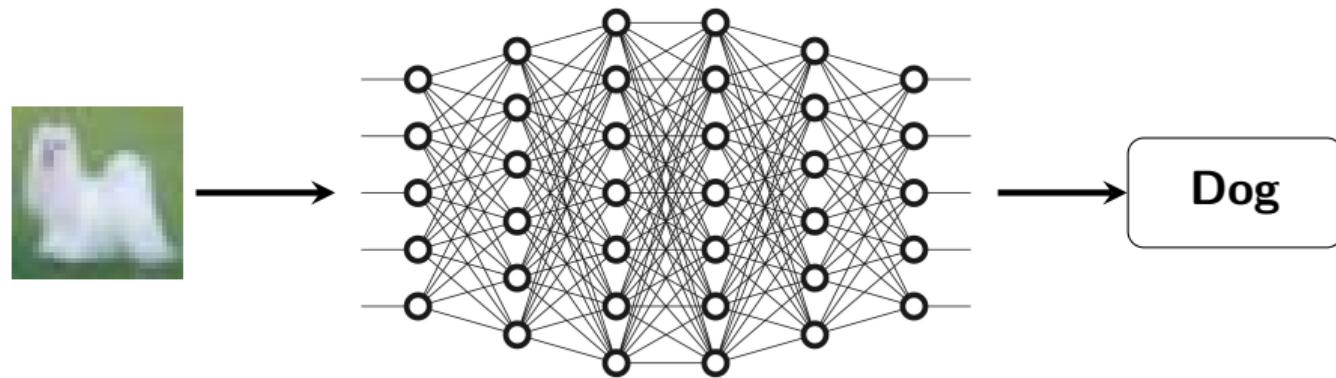


Redes Convolucionales

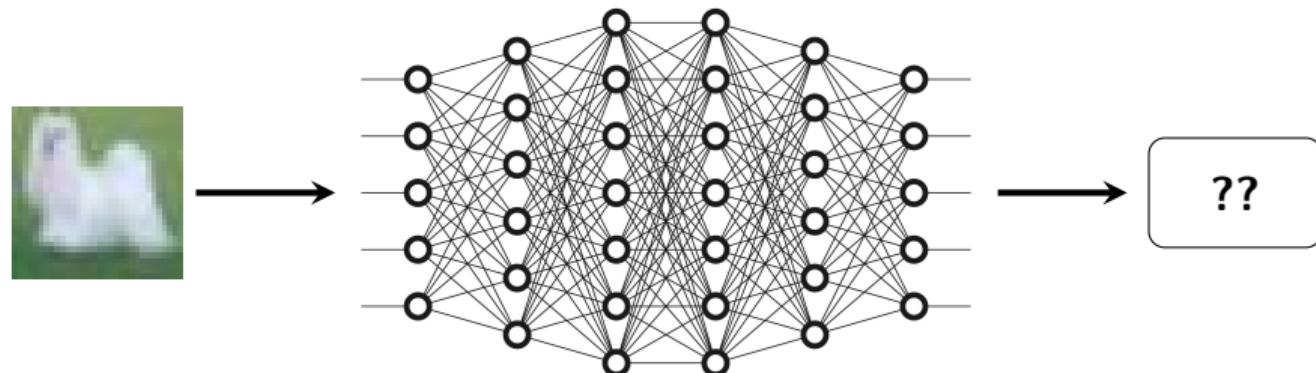
1	2	0
3	4	0
0	0	0



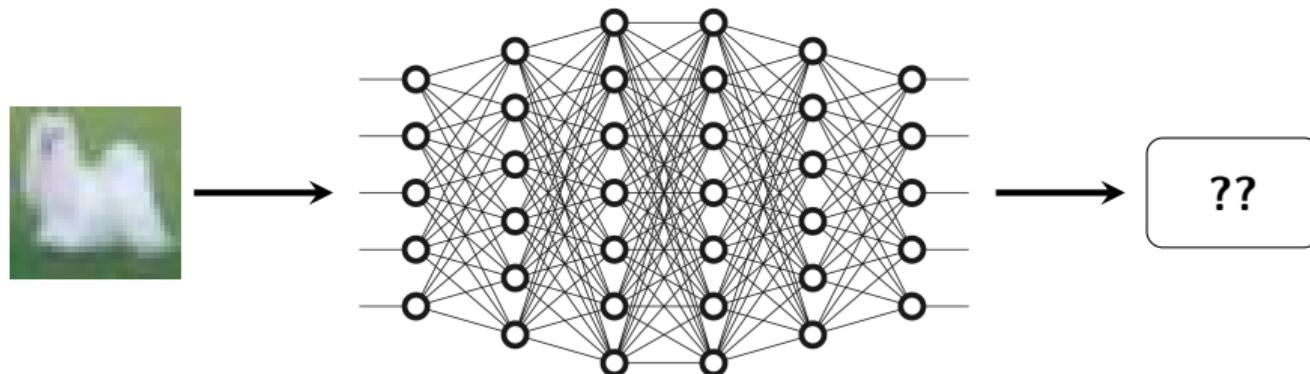
Redes Convolucionales



Redes Convolucionales



Redes Convolucionales



Las redes convolucionales obligan a la red a aprender patrones que son invariantes a translaciones.

Las capas *fully-connected* están conectadas a todos los inputs de la capa anterior.

0	0	0
0	1	2
0	3	4

Redes Convolucionales

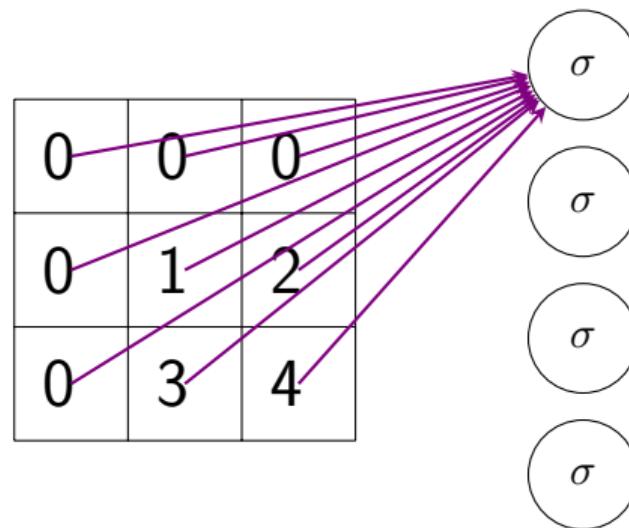
Las capas *fully-connected* están conectadas a todos los inputs de la capa anterior.

0	0	0
0	1	2
0	3	4

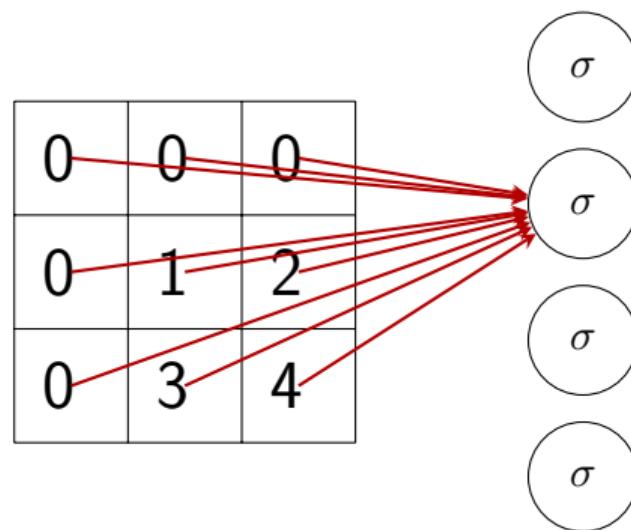


Redes Convolucionales

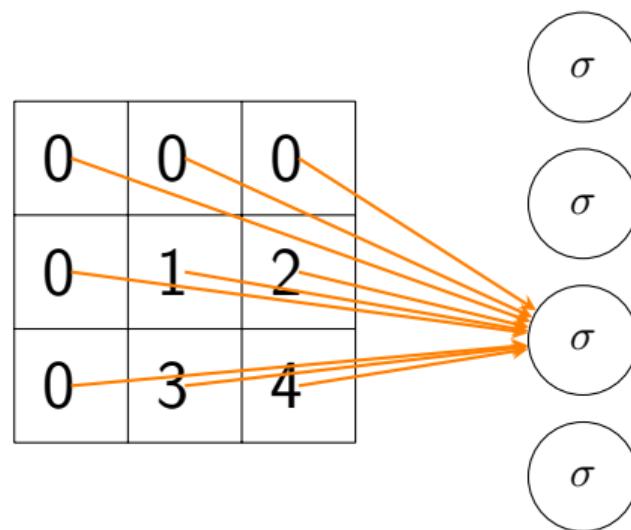
Las capas *fully-connected* están conectadas a todos los inputs de la capa anterior.



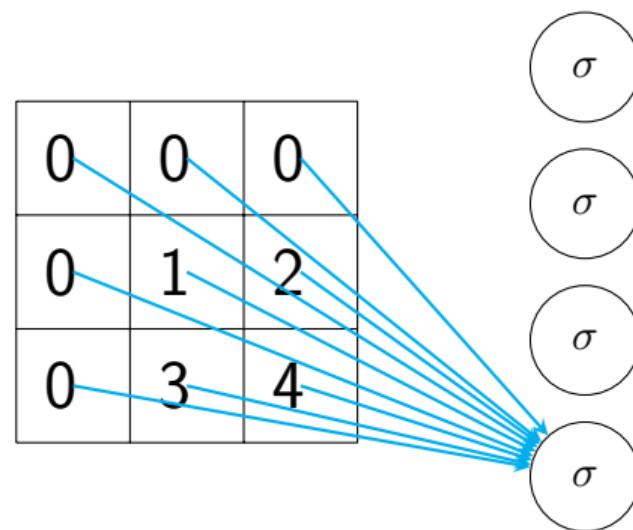
Las capas *fully-connected* están conectadas a todos los inputs de la capa anterior.



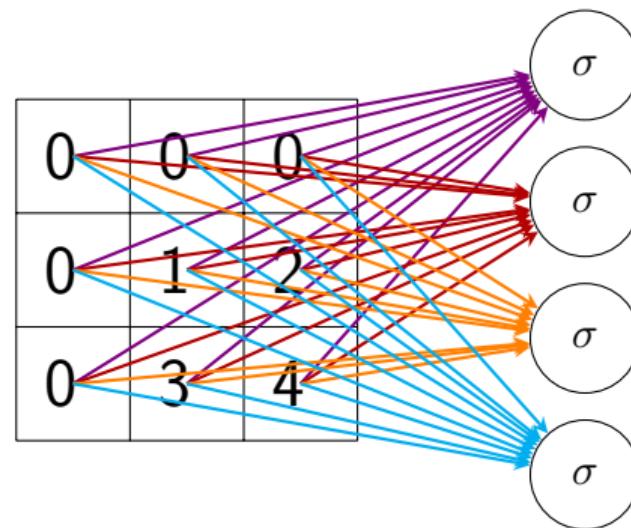
Las capas *fully-connected* están conectadas a todos los inputs de la capa anterior.



Las capas *fully-connected* están conectadas a todos los inputs de la capa anterior.



Las capas *fully-connected* están conectadas a todos los inputs de la capa anterior.



Redes Convolucionales

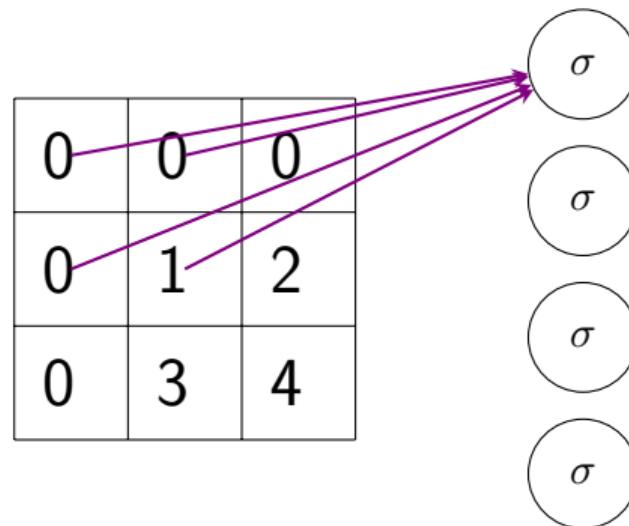
Las capas *locally connected* están conectadas a algunos inputs de la capa anterior.

0	0	0
0	1	2
0	3	4



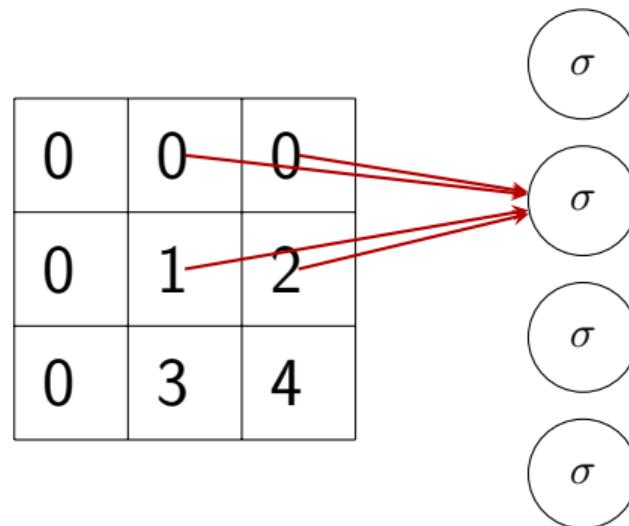
Redes Convolucionales

Las capas *locally connected* están conectadas a algunos inputs de la capa anterior.



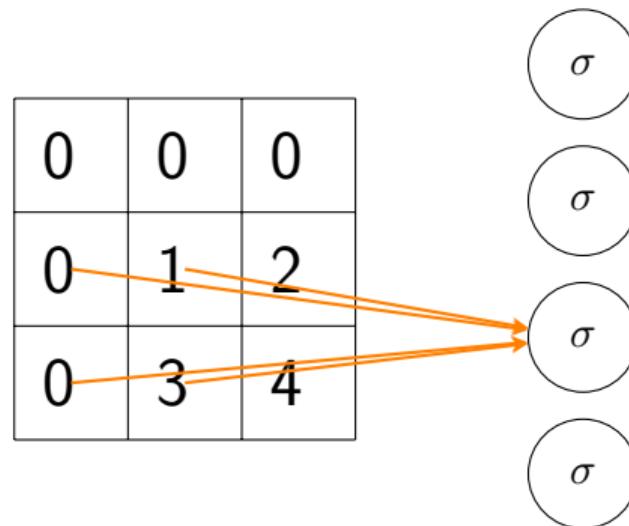
Redes Convolucionales

Las capas *locally connected* están conectadas a algunos inputs de la capa anterior.



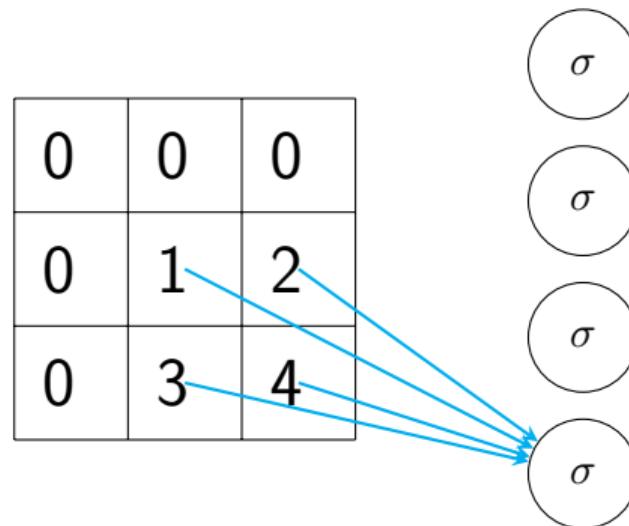
Redes Convolucionales

Las capas *locally connected* están conectadas a algunos inputs de la capa anterior.



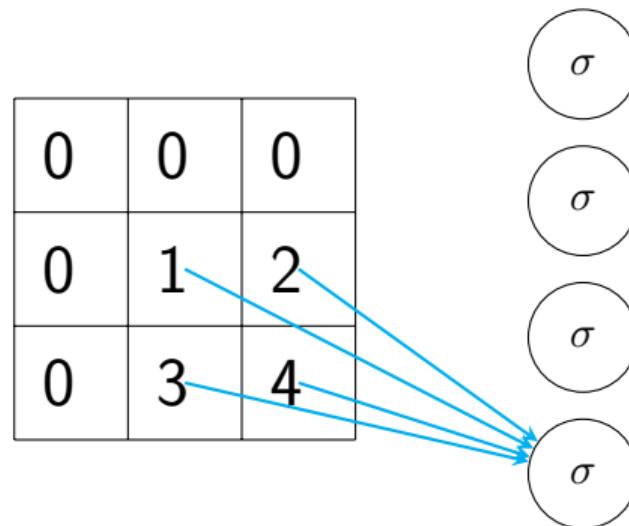
Redes Convolucionales

Las capas *locally connected* están conectadas a algunos inputs de la capa anterior.



Redes Convolucionales

Las capas *locally connected* están conectadas a algunos inputs de la capa anterior.



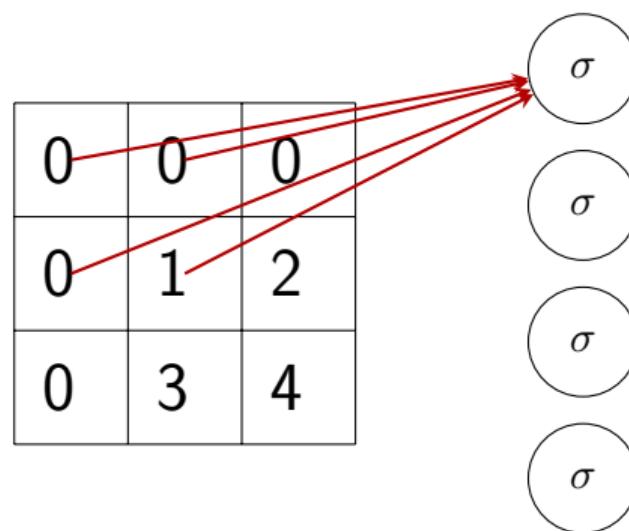
Este tipo de conexiones permite aprender patrones locales.

... pero como queremos forzar a que el patrón aprendido sea el mismo, independiente de la posición, forzamos a que los pesos de las neuronas sean iguales.

0	0	0
0	1	2
0	3	4

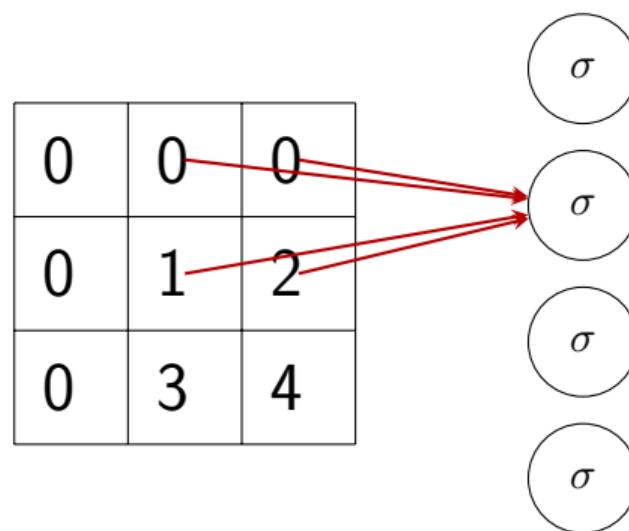


... pero como queremos forzar a que el patrón aprendido sea el mismo, independiente de la posición, forzamos a que los pesos de las neuronas sean iguales.



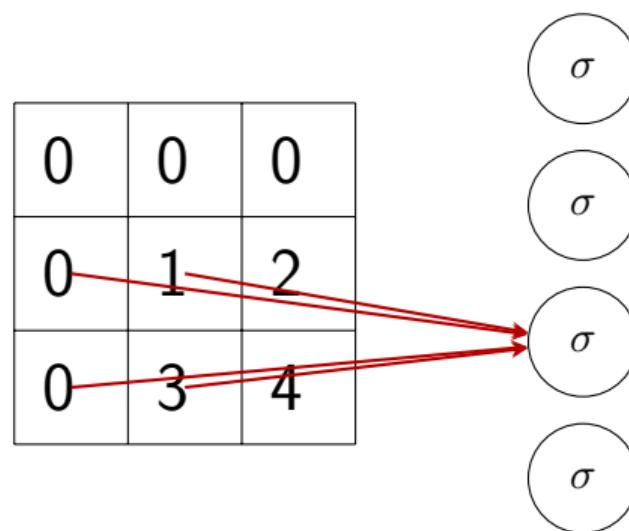
Redes Convolucionales

... pero como queremos forzar a que el patrón aprendido sea el mismo, independiente de la posición, forzamos a que los pesos de las neuronas sean iguales.

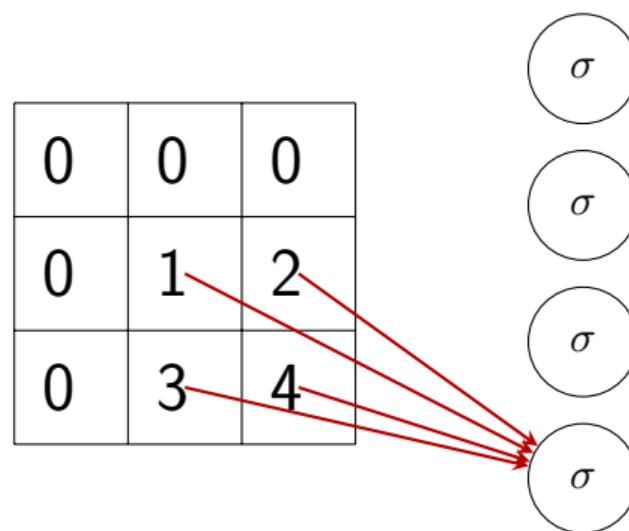


Redes Convolucionales

... pero como queremos forzar a que el patrón aprendido sea el mismo, independiente de la posición, forzamos a que los pesos de las neuronas sean iguales.

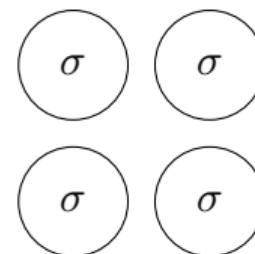


... pero como queremos forzar a que el patrón aprendido sea el mismo, independiente de la posición, forzamos a que los pesos de las neuronas sean iguales.

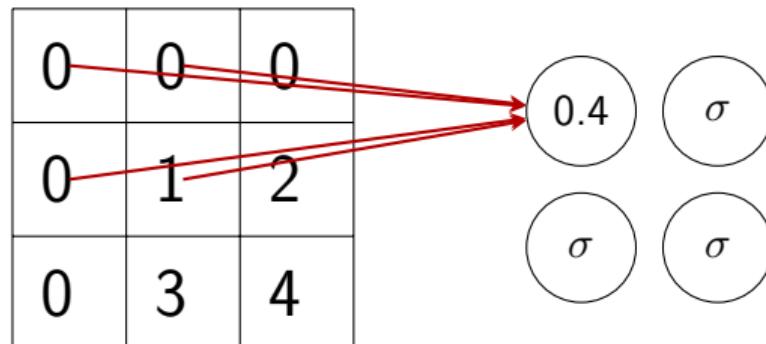


Por ejemplo, reordenando las neuronas y asumiendo que los pesos son $\{0.1, 0.2, 0.3, 0.4\}$:

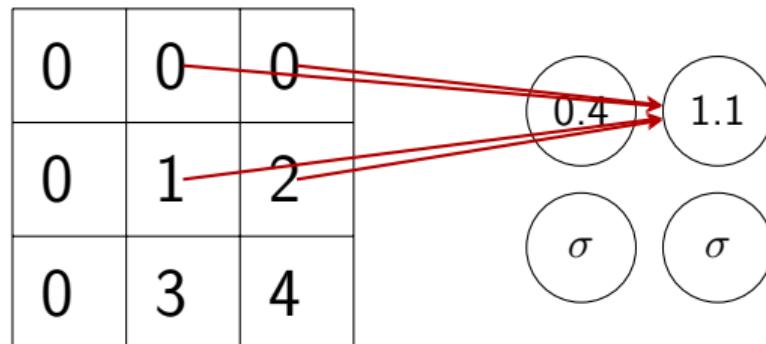
0	0	0
0	1	2
0	3	4



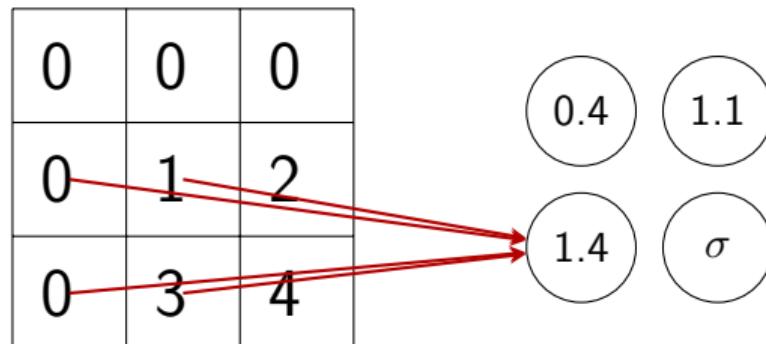
Por ejemplo, reordenando las neuronas y asumiendo que los pesos son $\{0.1, 0.2, 0.3, 0.4\}$:



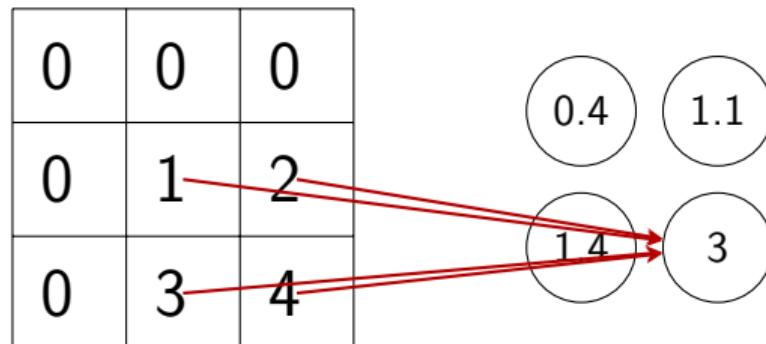
Por ejemplo, reordenando las neuronas y asumiendo que los pesos son $\{0.1, 0.2, 0.3, 0.4\}$:



Por ejemplo, reordenando las neuronas y asumiendo que los pesos son $\{0.1, 0.2, 0.3, 0.4\}$:

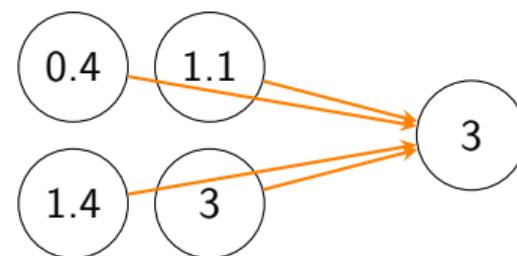


Por ejemplo, reordenando las neuronas y asumiendo que los pesos son $\{0.1, 0.2, 0.3, 0.4\}$:



Por ejemplo, reordenando las neuronas y asumiendo que los pesos son $\{0.1, 0.2, 0.3, 0.4\}$:

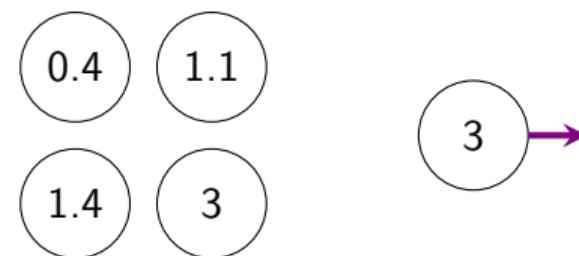
0	0	0
0	1	2
0	3	4



Luego de esto, se suele poner una capa de *Max Pooling*.

Por ejemplo, reordenando las neuronas y asumiendo que los pesos son $\{0.1, 0.2, 0.3, 0.4\}$:

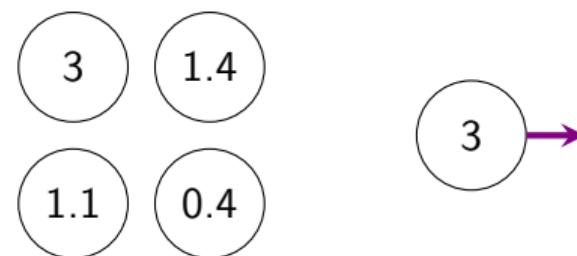
0	0	0
0	1	2
0	3	4



Luego de esto, se suele poner una capa de *Max Pooling*.

Por ejemplo, reordenando las neuronas y asumiendo que los pesos son $\{0.1, 0.2, 0.3, 0.4\}$:

1	2	0
3	4	0
0	0	0



Luego de esto, se suele poner una capa de *Max Pooling*.

Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

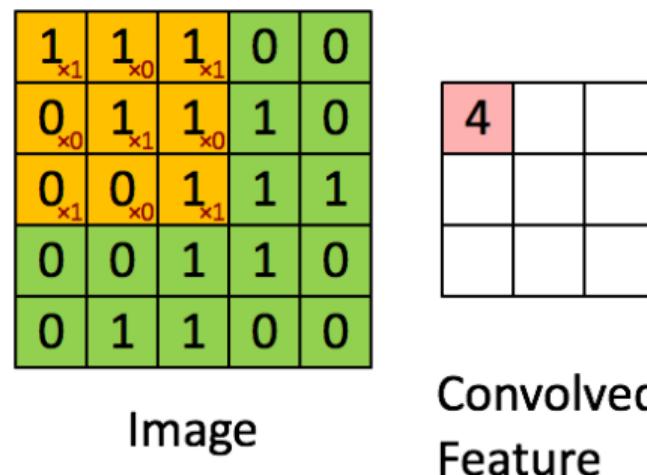
- Convoluciones.
- Max Pooling.

Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

- Convoluciones.
- Max Pooling.

Las convoluciones implican aplicar la misma transformación local sobre toda la imagen.

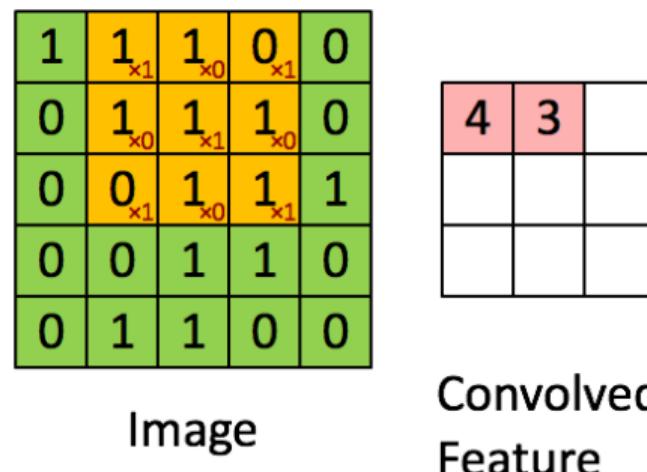


Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

- Convoluciones.
- Max Pooling.

Las convoluciones implican aplicar la misma transformación local sobre toda la imagen.

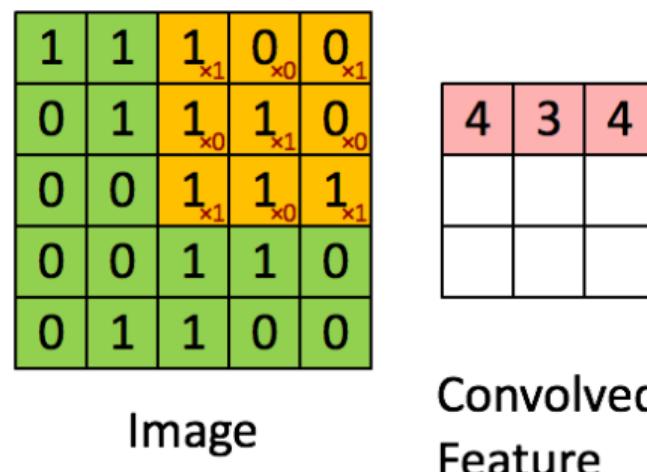


Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

- Convoluciones.
- Max Pooling.

Las convoluciones implican aplicar la misma transformación local sobre toda la imagen.

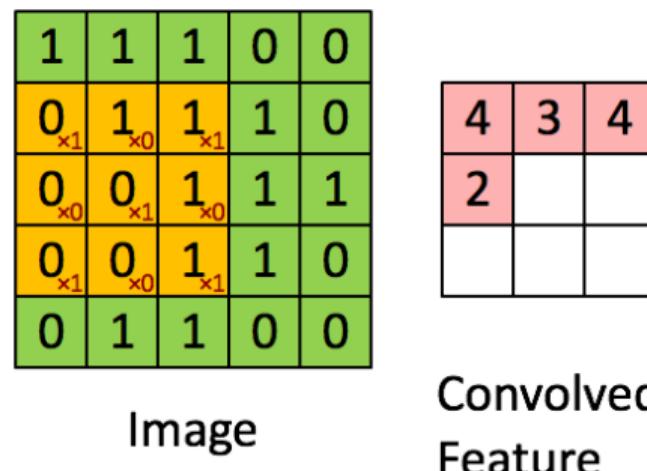


Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

- Convoluciones.
- Max Pooling.

Las convoluciones implican aplicar la misma transformación local sobre toda la imagen.

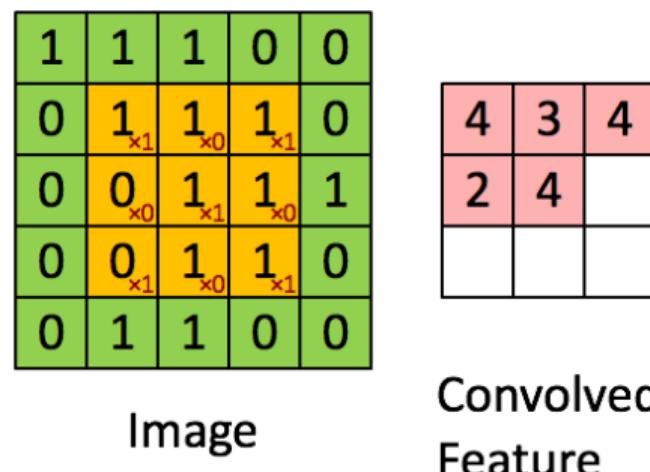


Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

- Convoluciones.
- Max Pooling.

Las convoluciones implican aplicar la misma transformación local sobre toda la imagen.

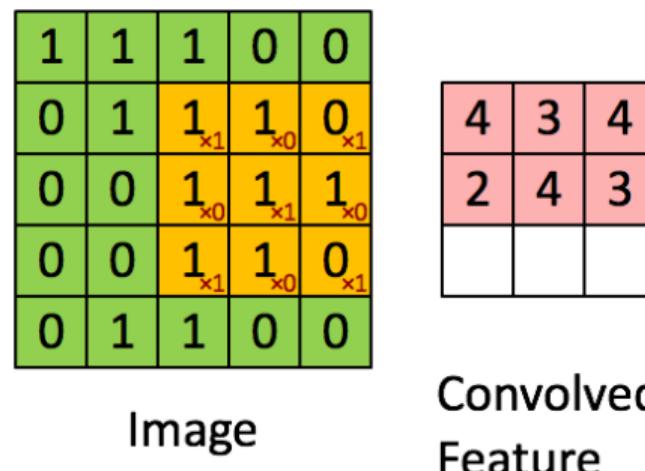


Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

- Convoluciones.
- Max Pooling.

Las convoluciones implican aplicar la misma transformación local sobre toda la imagen.

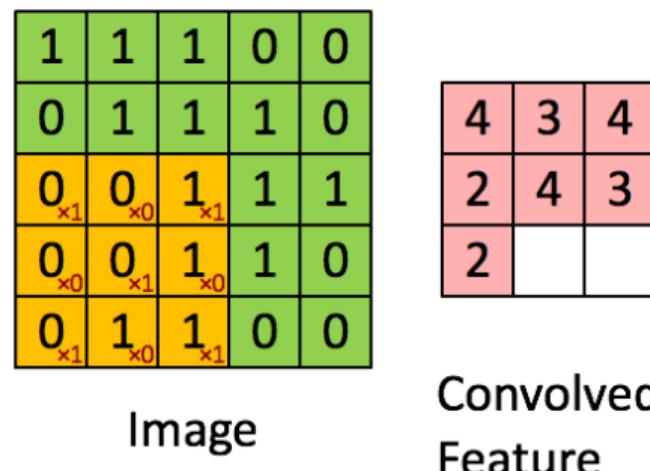


Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

- Convoluciones.
- Max Pooling.

Las convoluciones implican aplicar la misma transformación local sobre toda la imagen.

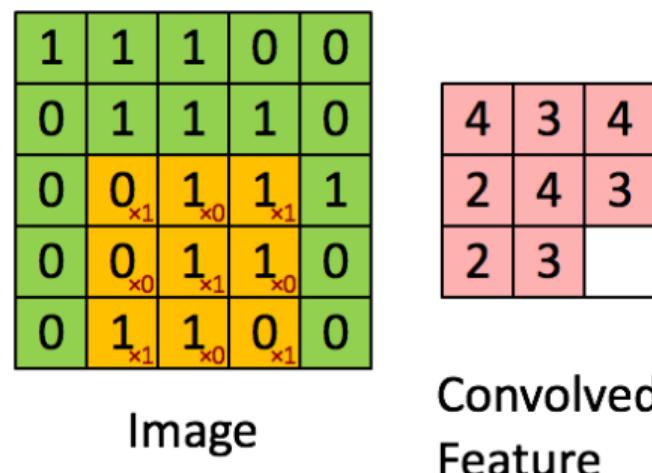


Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

- Convoluciones.
- Max Pooling.

Las convoluciones implican aplicar la misma transformación local sobre toda la imagen.

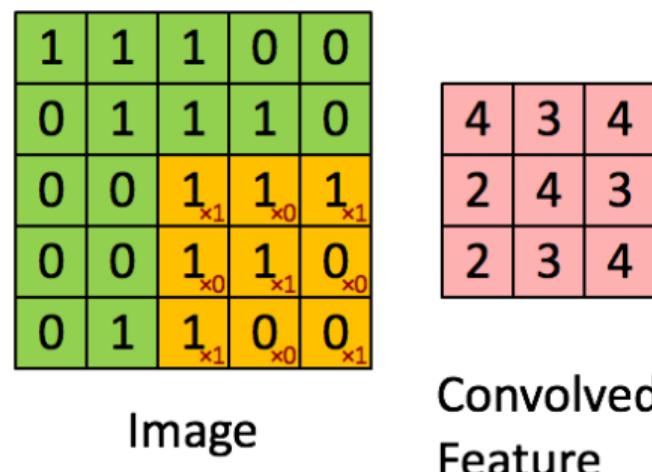


Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

- Convoluciones.
- Max Pooling.

Las convoluciones implican aplicar la misma transformación local sobre toda la imagen.



Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

- Convoluciones.
- Max Pooling.

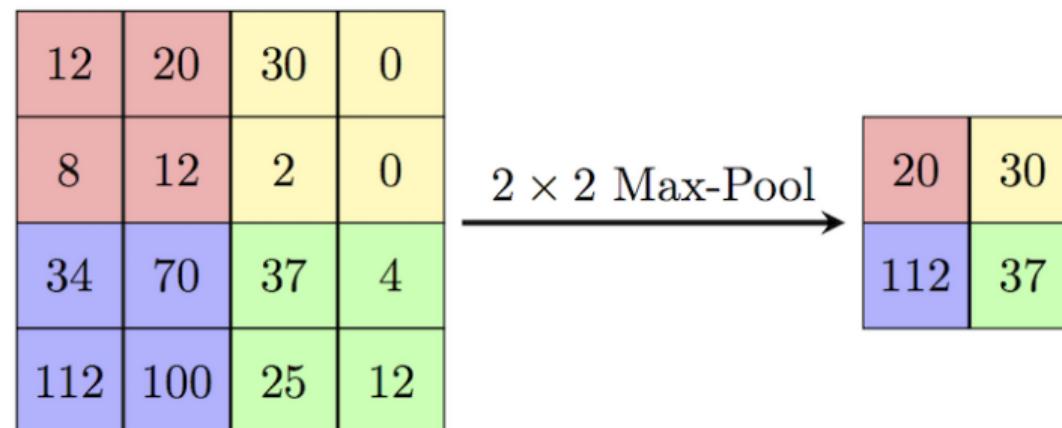
Max Pooling divide la imagen en distintos cuadrantes y se queda con el máximo.

Redes Convolucionales

Con este ejemplo acabamos de introducir dos conceptos claves:

- Convoluciones.
- Max Pooling.

Max Pooling divide la imagen en distintos cuadrantes y se queda con el máximo.

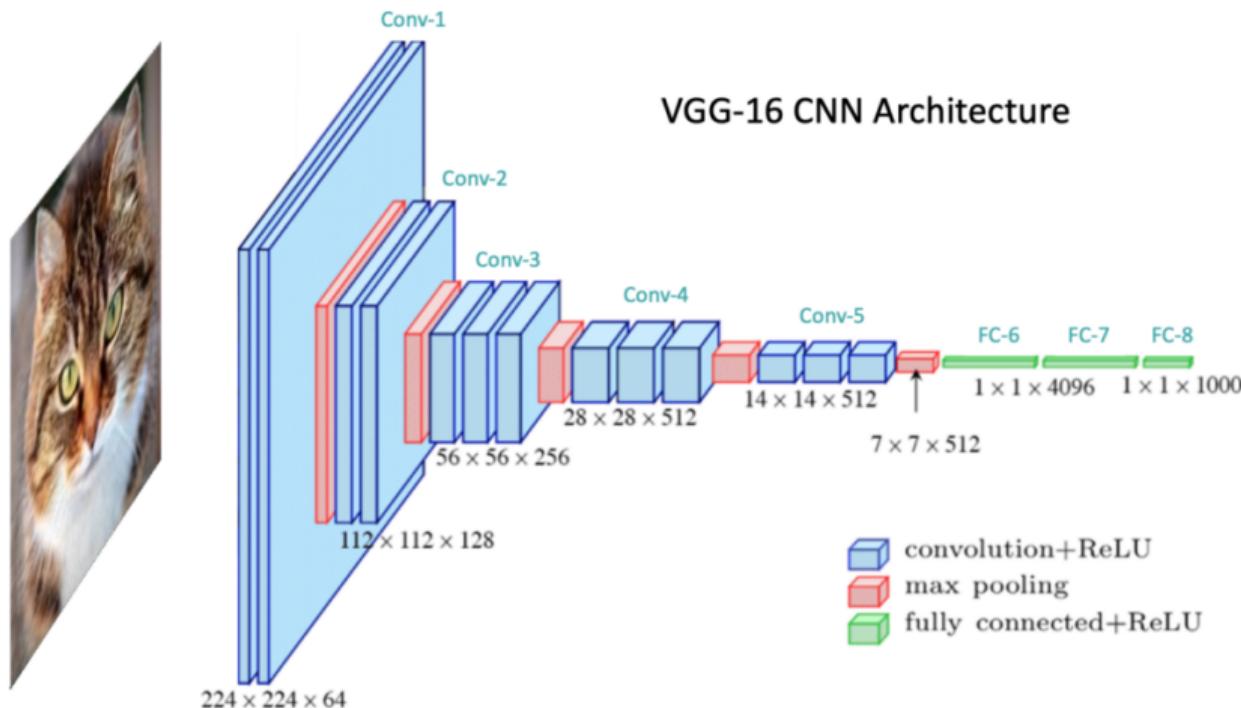


Redes Convolucionales

Entonces, las redes convolucionales se ven más o menos así:

Redes Convolucionales

Entonces, las redes convolucionales se ven más o menos así:

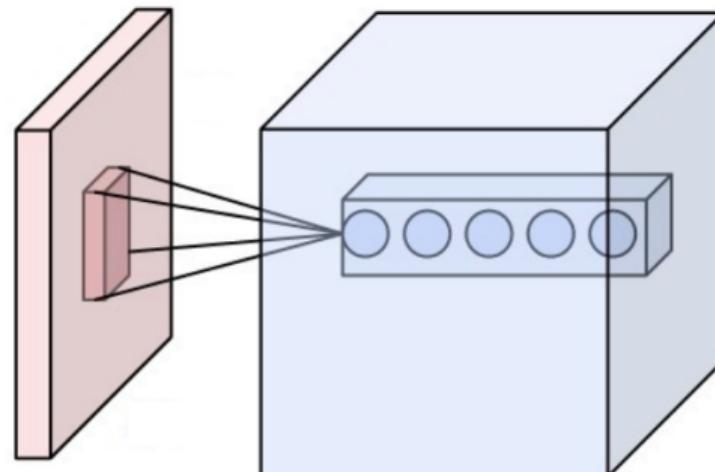


Redes Convolucionales

Las capas convolucionales se ven más o menos así:

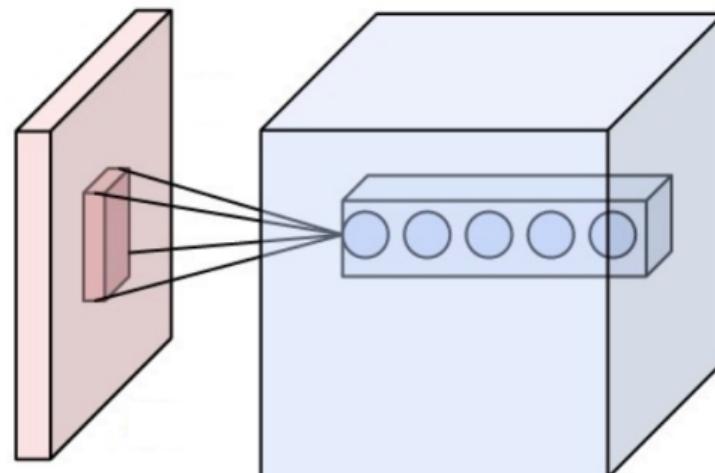
Redes Convolucionales

Las capas convolucionales se ven más o menos así:



Redes Convolucionales

Las capas convolucionales se ven más o menos así:



Al definir una capa convolucional tenemos que tomar las siguientes decisiones:

- *in channels, out channels, kernel size y stride.*

Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10.

Redes Convolucionales

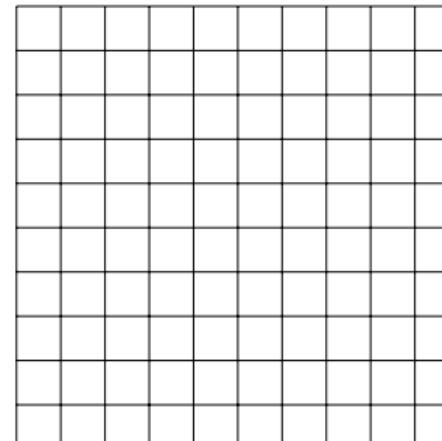
Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.

Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

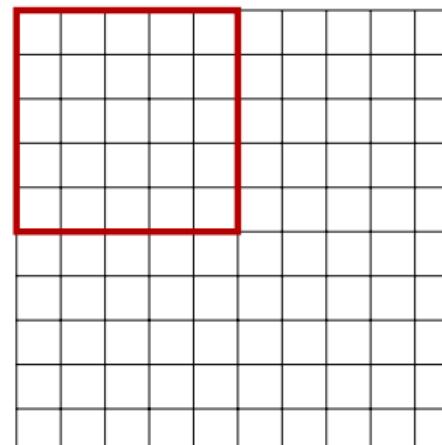
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

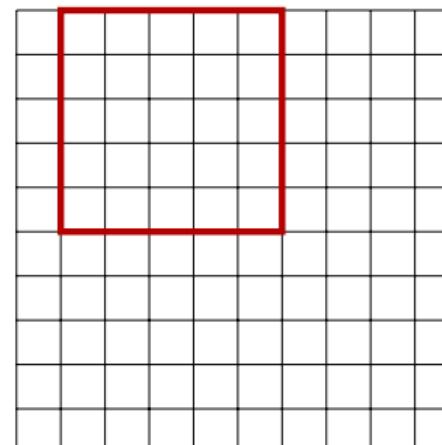
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

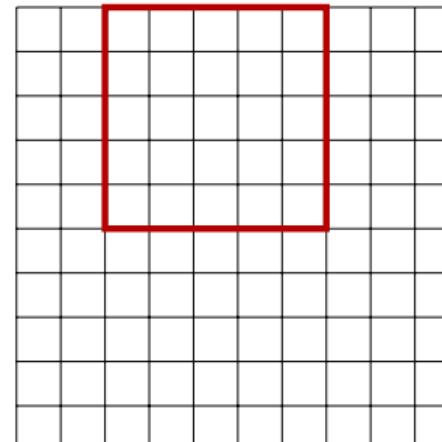
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

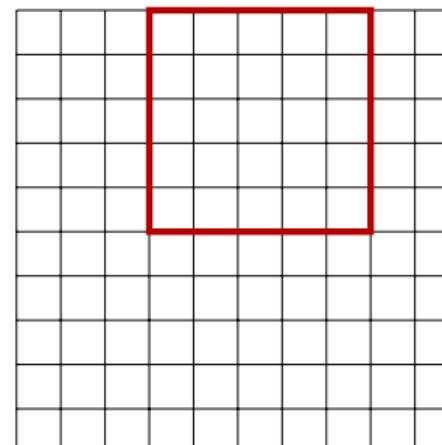
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

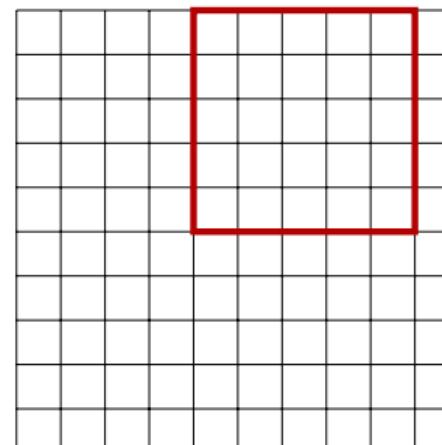
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

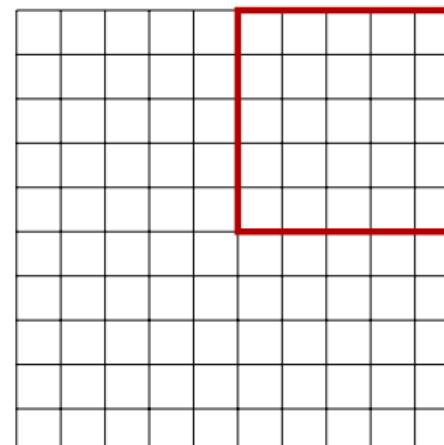
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

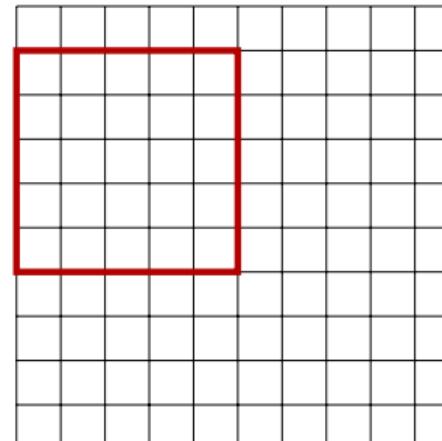
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

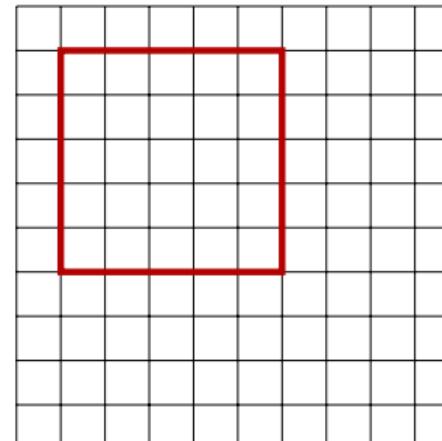
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

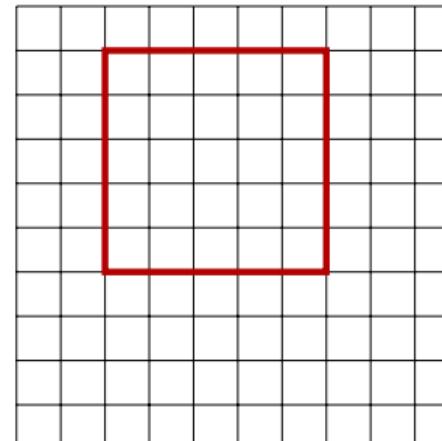
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

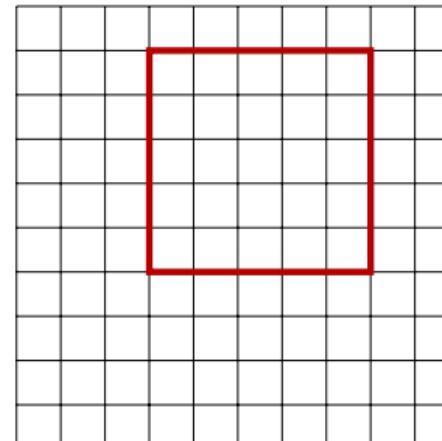
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

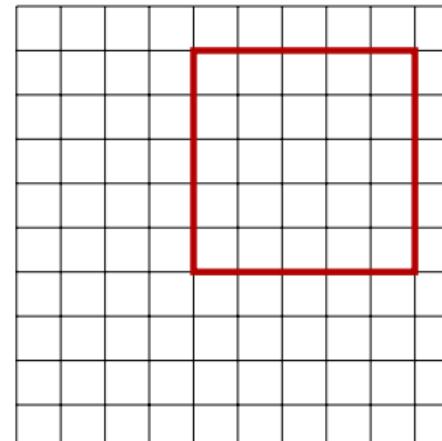
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

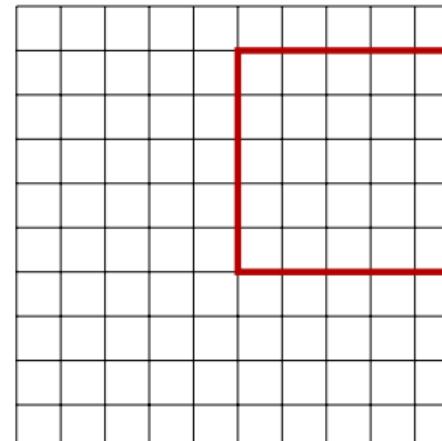
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.



Redes Convolucionales

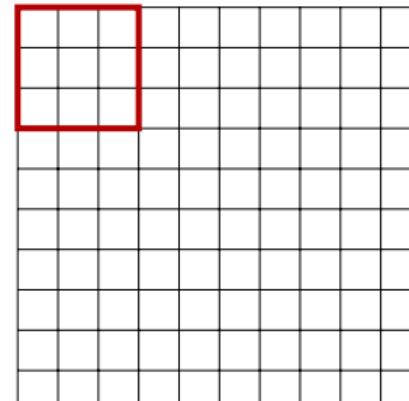
Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 3.
- *stride*: 2.

Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

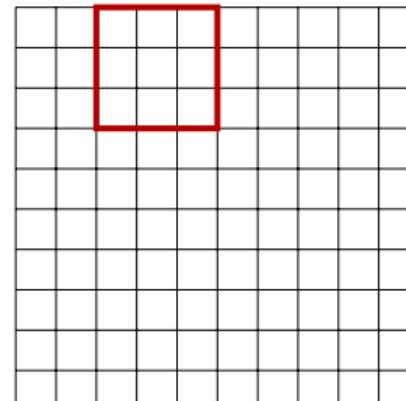
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 3.
- *stride*: 2.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

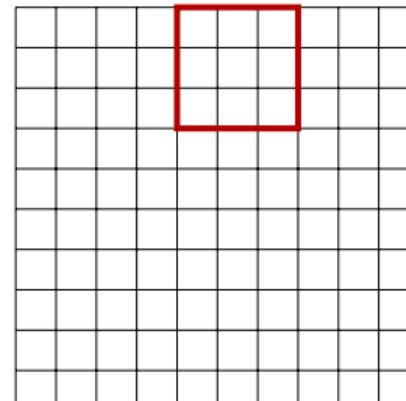
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 3.
- *stride*: 2.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

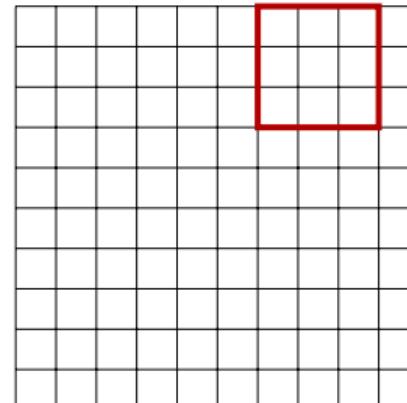
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 3.
- *stride*: 2.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

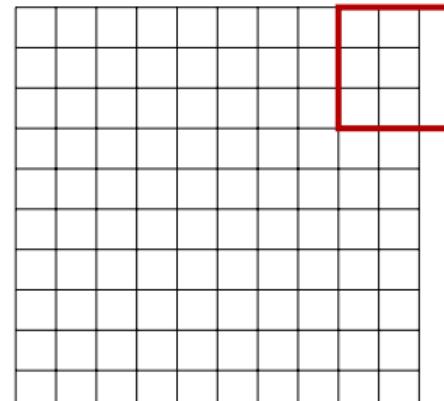
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 3.
- *stride*: 2.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 3.
- *stride*: 2.



Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

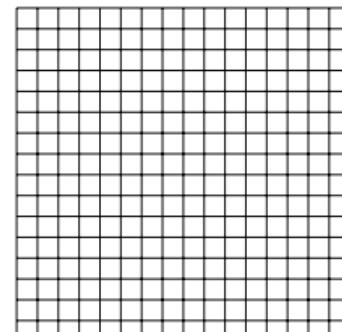
- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.

Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.

Esta primera capa convolucional transforma la salida de $32 \times 32 \times 3$ a $28 \times 28 \times 6$.

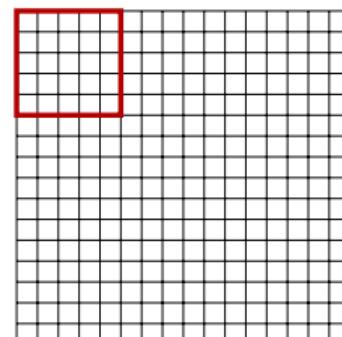


Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.

Esta primera capa convolucional transforma la salida de $32 \times 32 \times 3$ a $28 \times 28 \times 6$.

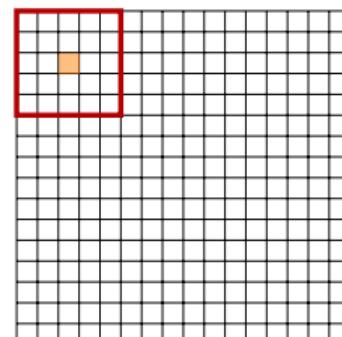


Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.

Esta primera capa convolucional transforma la salida de $32 \times 32 \times 3$ a $28 \times 28 \times 6$.

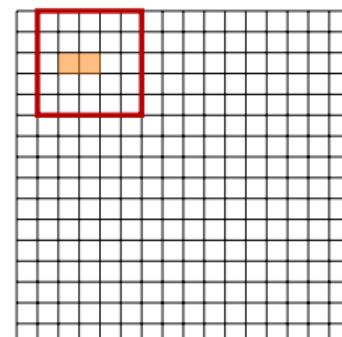


Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.

Esta primera capa convolucional transforma la salida de $32 \times 32 \times 3$ a $28 \times 28 \times 6$.

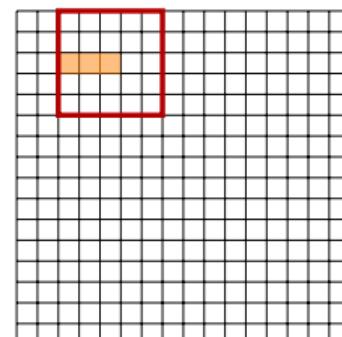


Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.

Esta primera capa convolucional transforma la salida de $32 \times 32 \times 3$ a $28 \times 28 \times 6$.

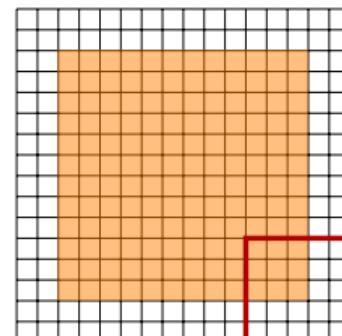


Redes Convolucionales

Definamos una primera capa convolucional para CIFAR-10. El input es $32 \times 32 \times 3$:

- *in channels*: 3.
- *out channels*: 6.
- *kernel size*: 5.
- *stride*: 1.

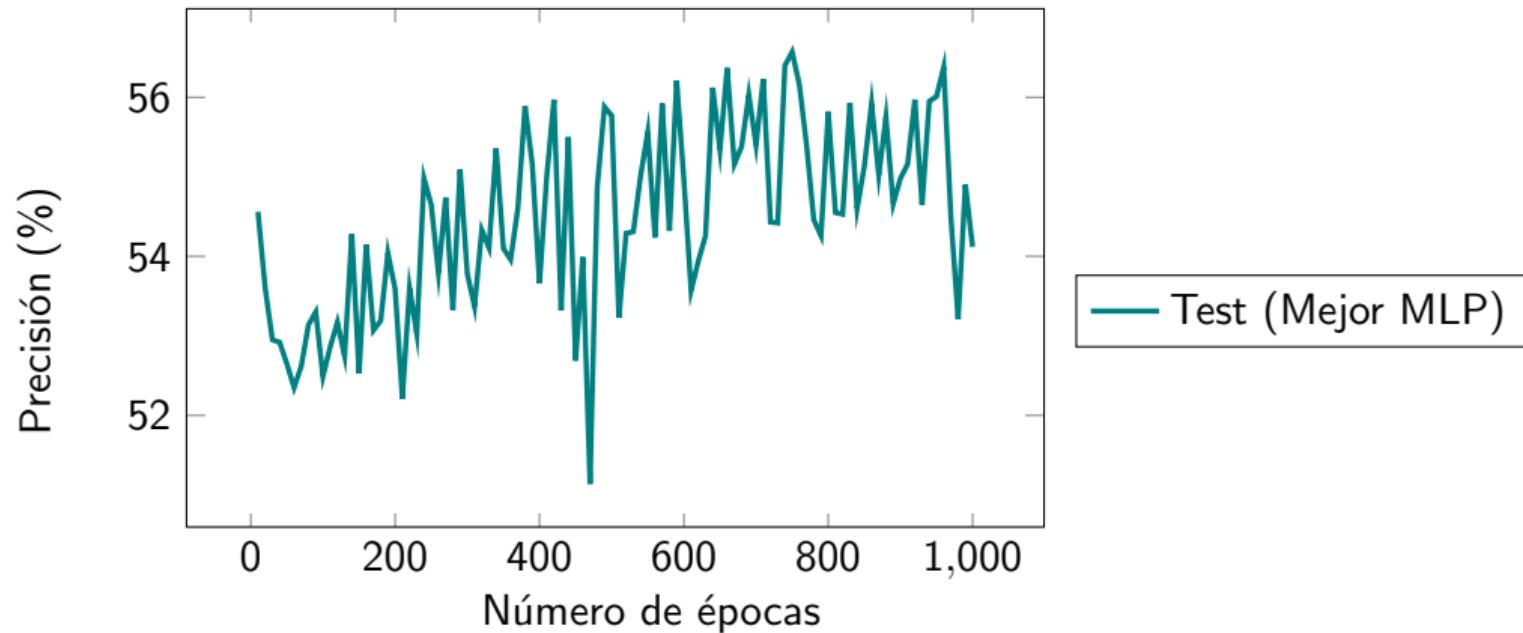
Esta primera capa convolucional transforma la salida de $32 \times 32 \times 3$ a $28 \times 28 \times 6$.



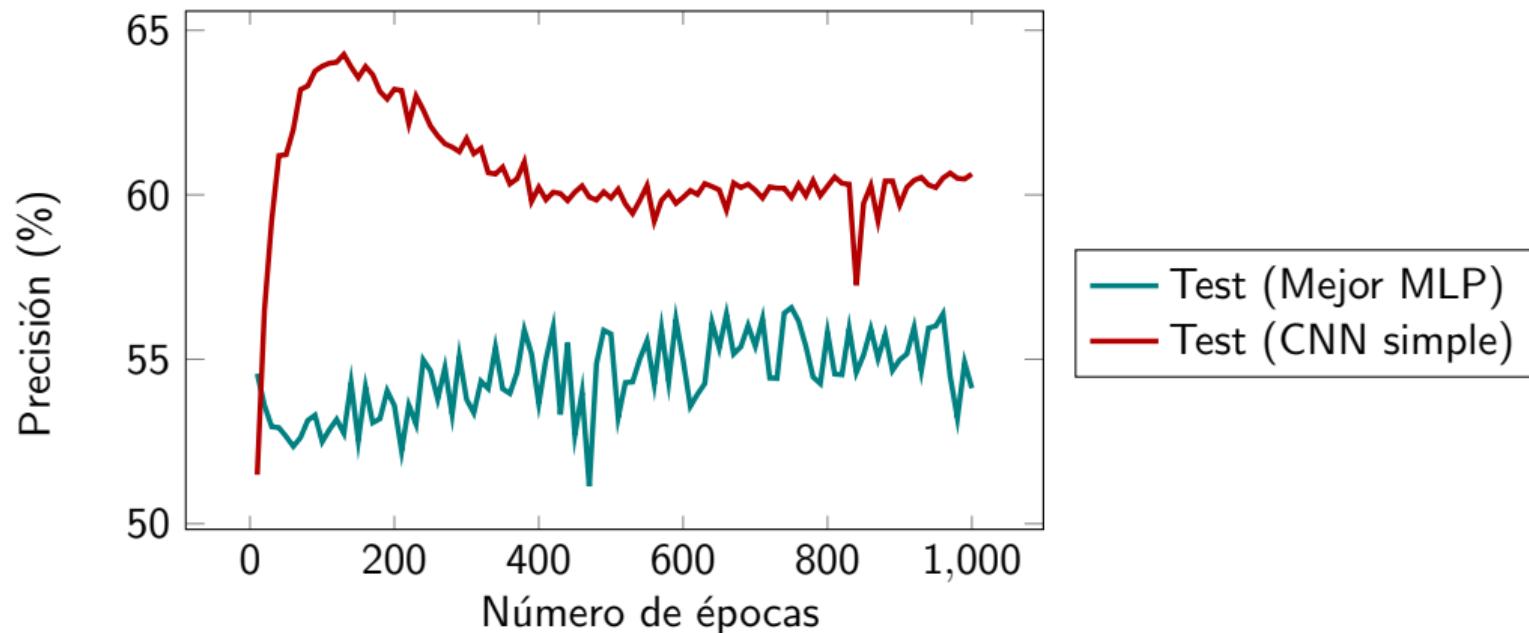
CNN simple para CIFAR-10:

- Input: $32 \times 32 \times 3$:
- Conv + Relu: *in channels*: 3, *out channels*: 6, *kernel size*: 5 y *stride*: 1.
- Max pooling.
- Conv + Relu: *in channels*: 6, *out channels*: 16, *kernel size*: 5 y *stride*: 1.
- Max pooling.
- Fully-connected + Relu: 120.
- Fully-connected + Relu: 80.
- Fully-connected + Softmax: 10.

Resultados en CIFAR-10 usando distintas arquitecturas.



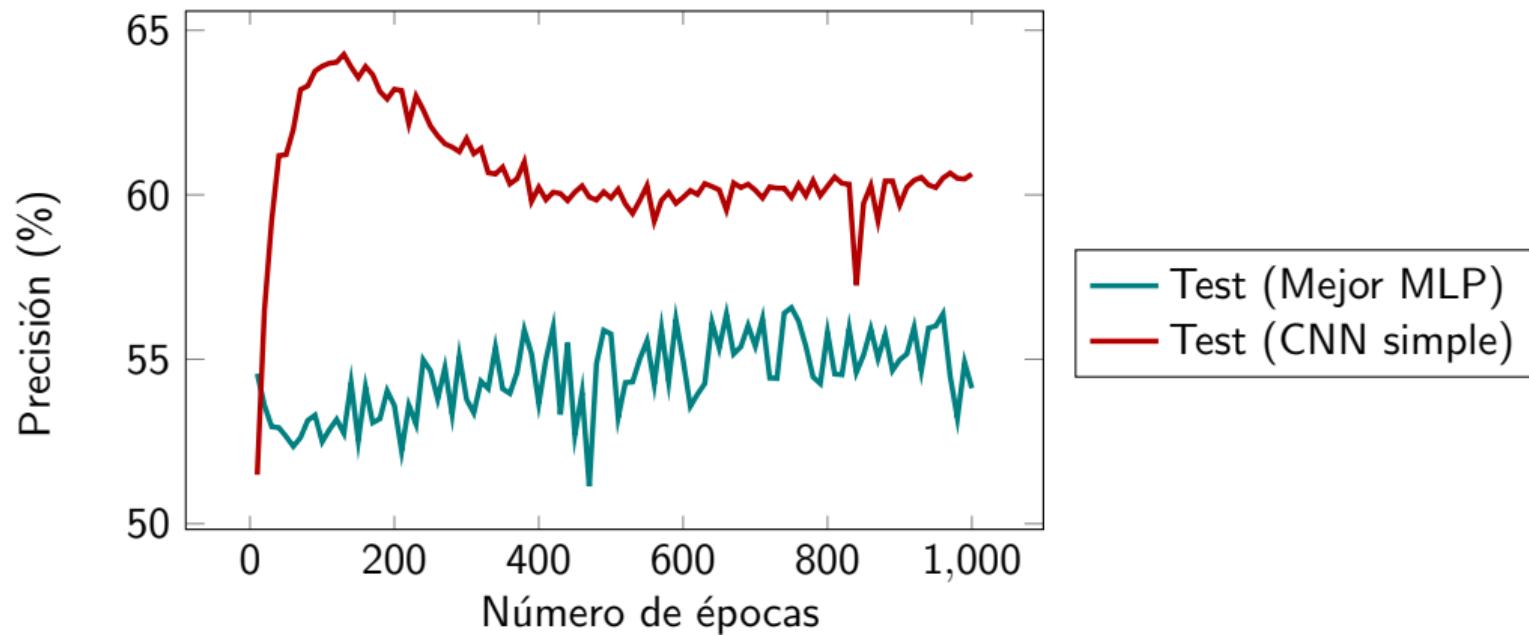
Resultados en CIFAR-10 usando distintas arquitecturas.



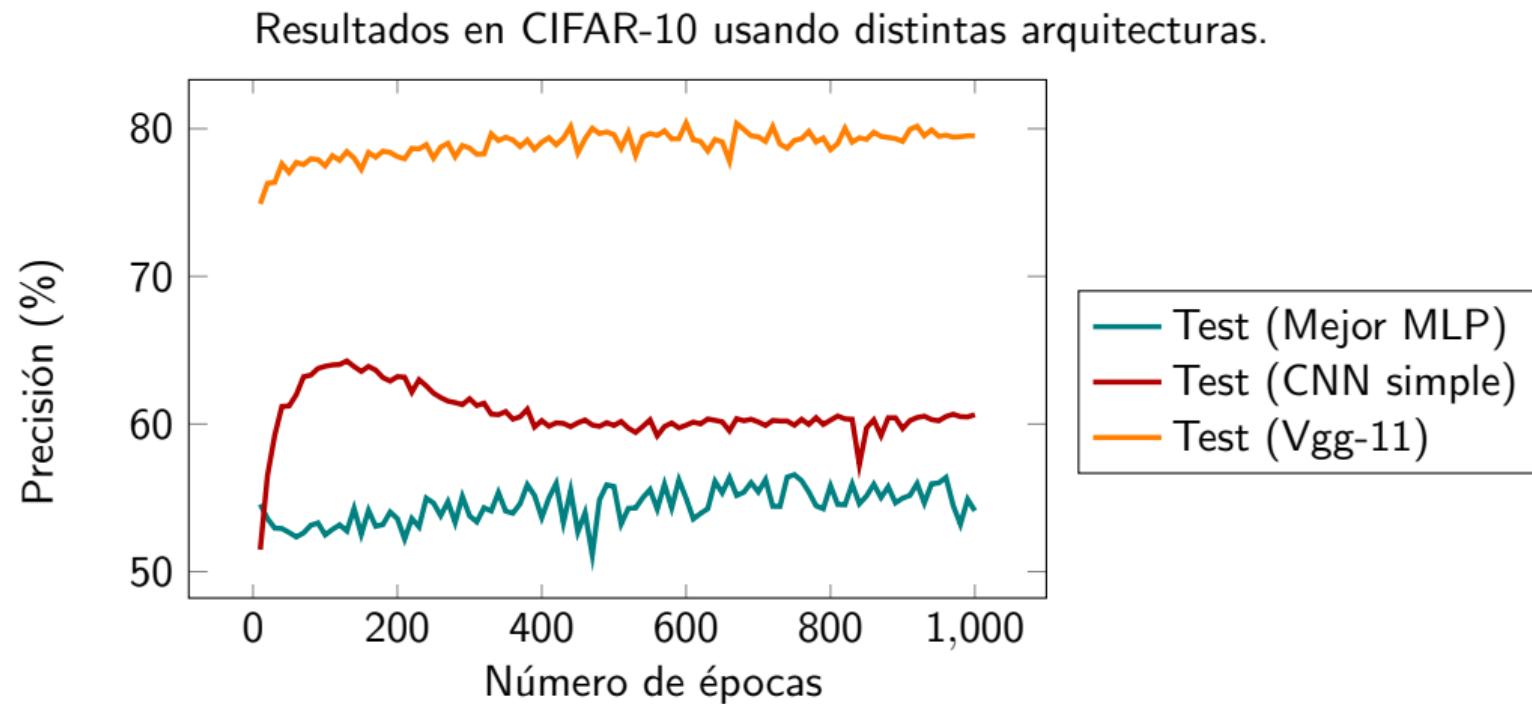
Ahora hagamos la red más *deep*. Este modelo se conoce como **Vgg-11**:

- Input: $32 \times 32 \times 3$:
- Conv3x3 (out 64) + Relu + Max pooling.
- Conv3x3 (out 128) + Relu + Max pooling.
- Conv3x3 (out 256) + Relu + Conv3x3 (out 256) + Relu + Max pooling.
- Conv3x3 (out 512) + Relu + Conv3x3 (out 512) + Relu + Max pooling.
- Conv3x3 (out 512) + Relu + Conv3x3 (out 512) + Relu + Max pooling.
- Fully-connected + Relu: 512.
- Fully-connected + Relu: 512.
- Fully-connected + Softmax: 10.

Resultados en CIFAR-10 usando distintas arquitecturas.

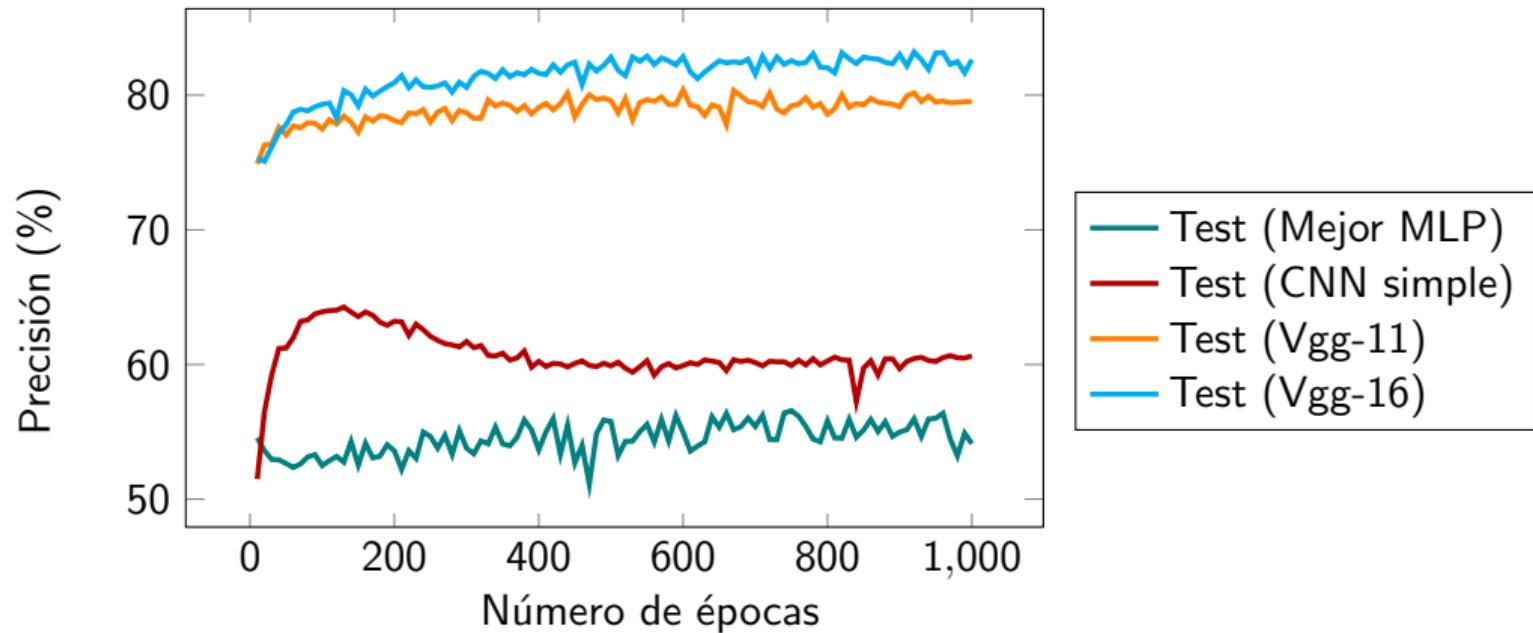


Redes Convolucionales



Redes Convolucionales

Resultados en CIFAR-10 usando distintas arquitecturas.

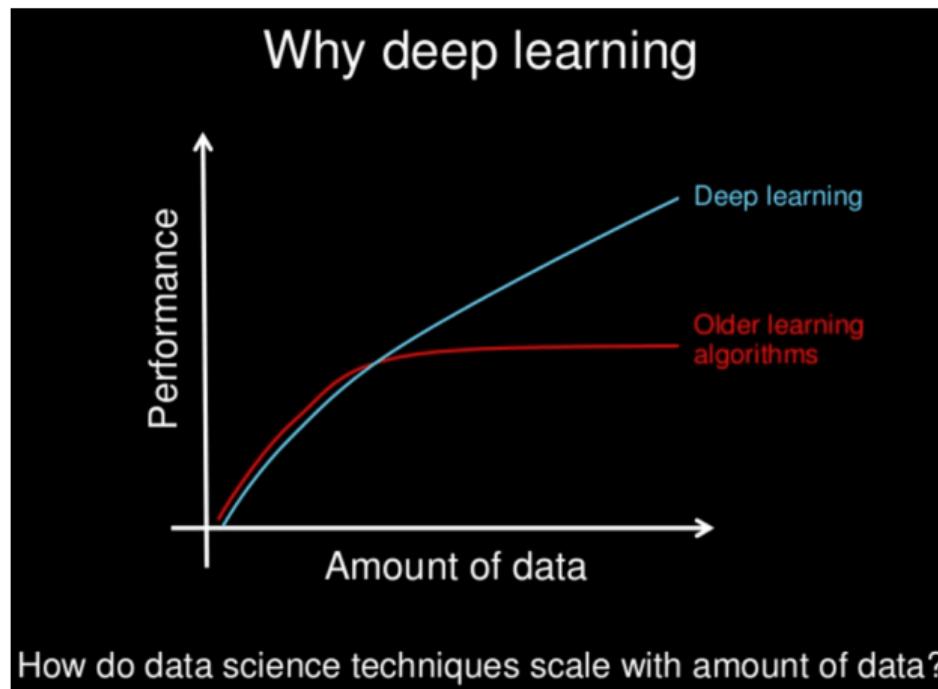


Redes Convolucionales

Estas redes profundas, mientras más datos tienan, mejor les va.

Redes Convolucionales

Estas redes profundas, mientras más datos tienan, mejor les va.



Redes Convolucionales

En CIFAR-10, el último gran salto se da cuando se usa *data augmentation*.

Redes Convolucionales

En CIFAR-10, el último gran salto se da cuando se usa *data augmentation*.

Para generar más datos automáticamente, hacemos dos cosas:

- Reflejamos de manera aleatoria las imágenes.
- Quitamos de manera aleatoria partes de la imagen.

Redes Convolucionales

En CIFAR-10, el último gran salto se da cuando se usa *data augmentation*.

Para generar más datos automáticamente, hacemos dos cosas:

- Reflejamos de manera aleatoria las imágenes.
- Quitamos de manera aleatoria partes de la imagen.

Así, a partir de un solo ejemplo podemos crear muchos más ejemplos:



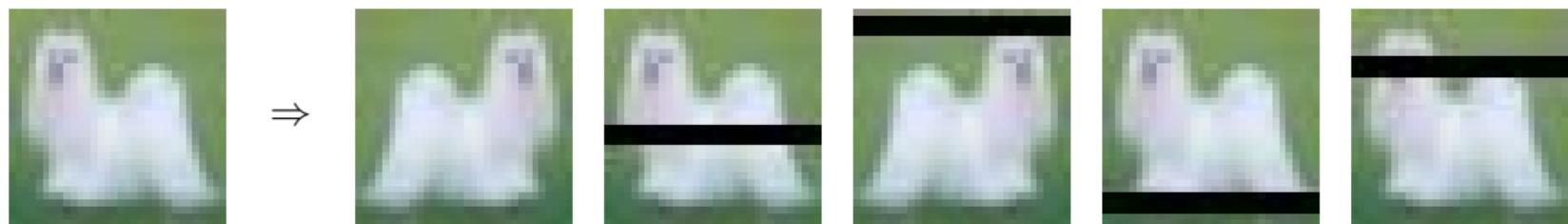
Redes Convolucionales

En CIFAR-10, el último gran salto se da cuando se usa *data augmentation*.

Para generar más datos automáticamente, hacemos dos cosas:

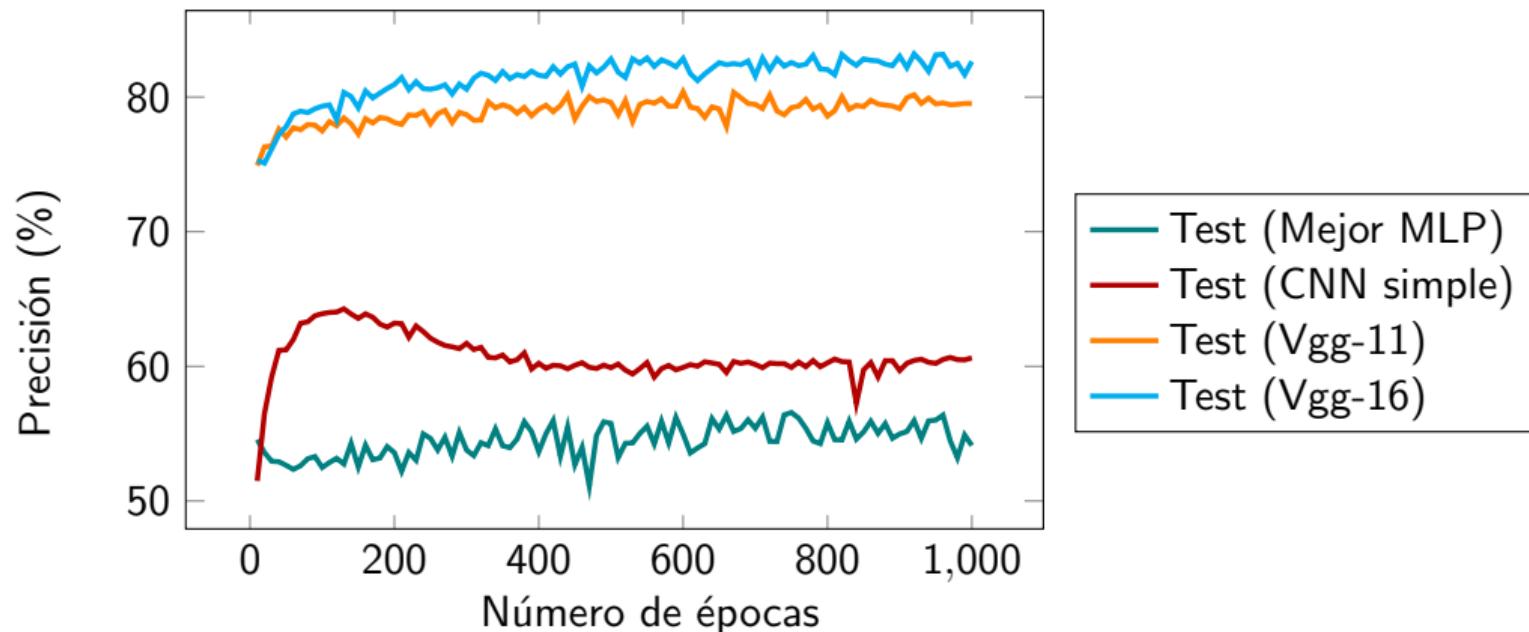
- Reflejamos de manera aleatoria las imágenes.
- Quitamos de manera aleatoria partes de la imagen.

Así, a partir de un solo ejemplo podemos crear muchos más ejemplos:

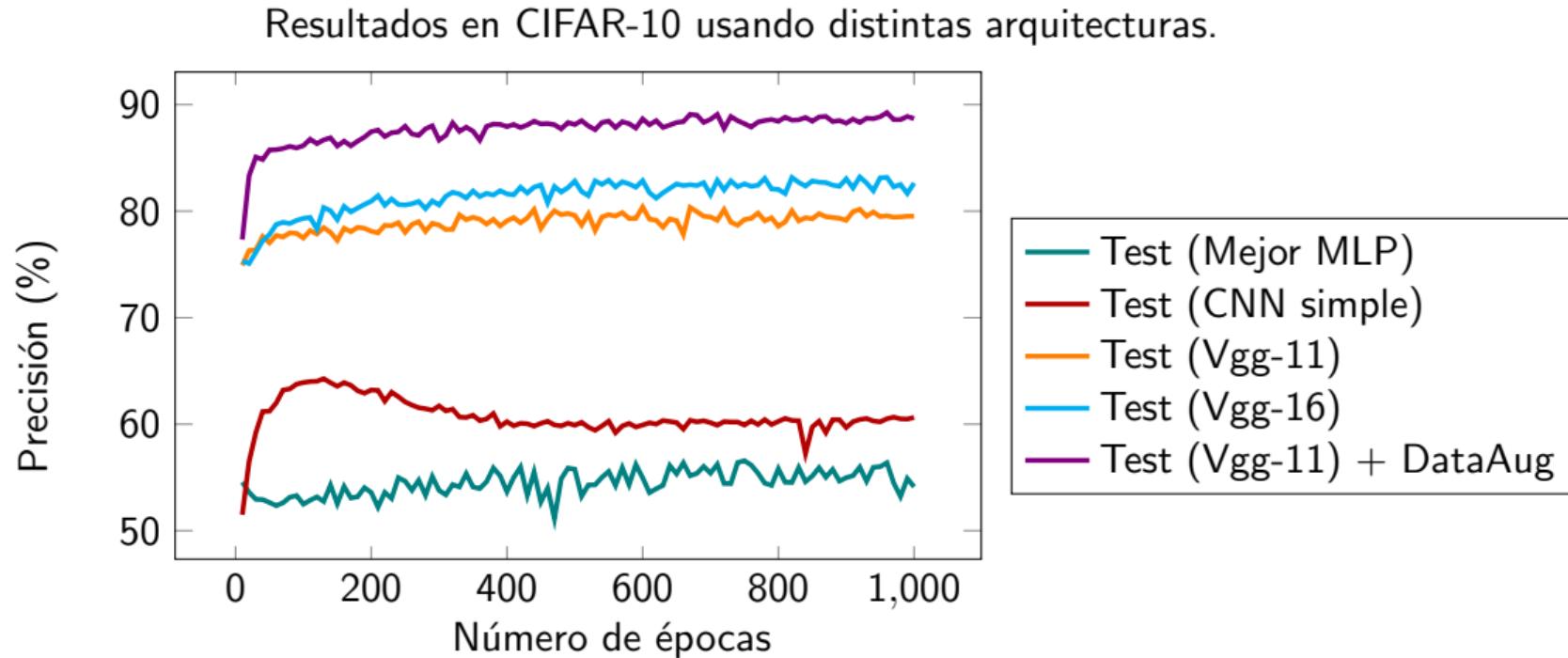


Redes Convolucionales

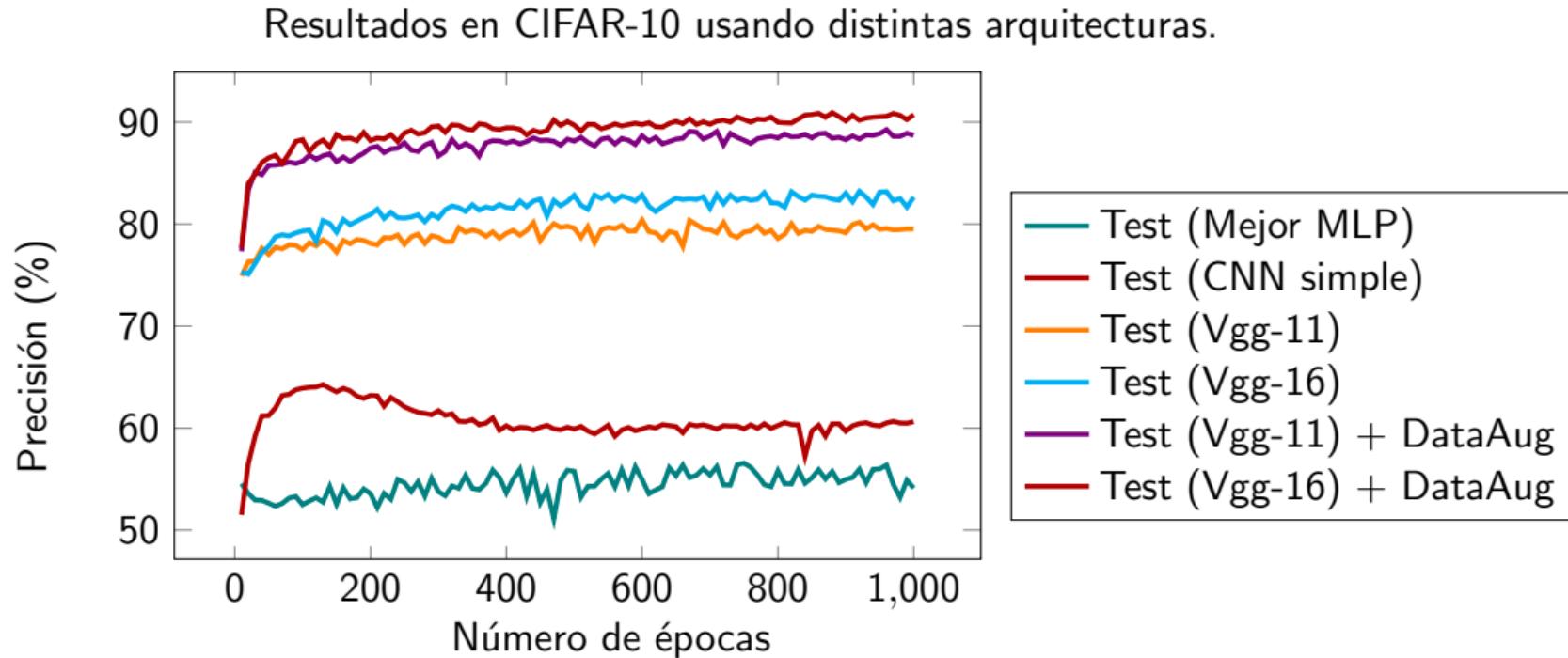
Resultados en CIFAR-10 usando distintas arquitecturas.



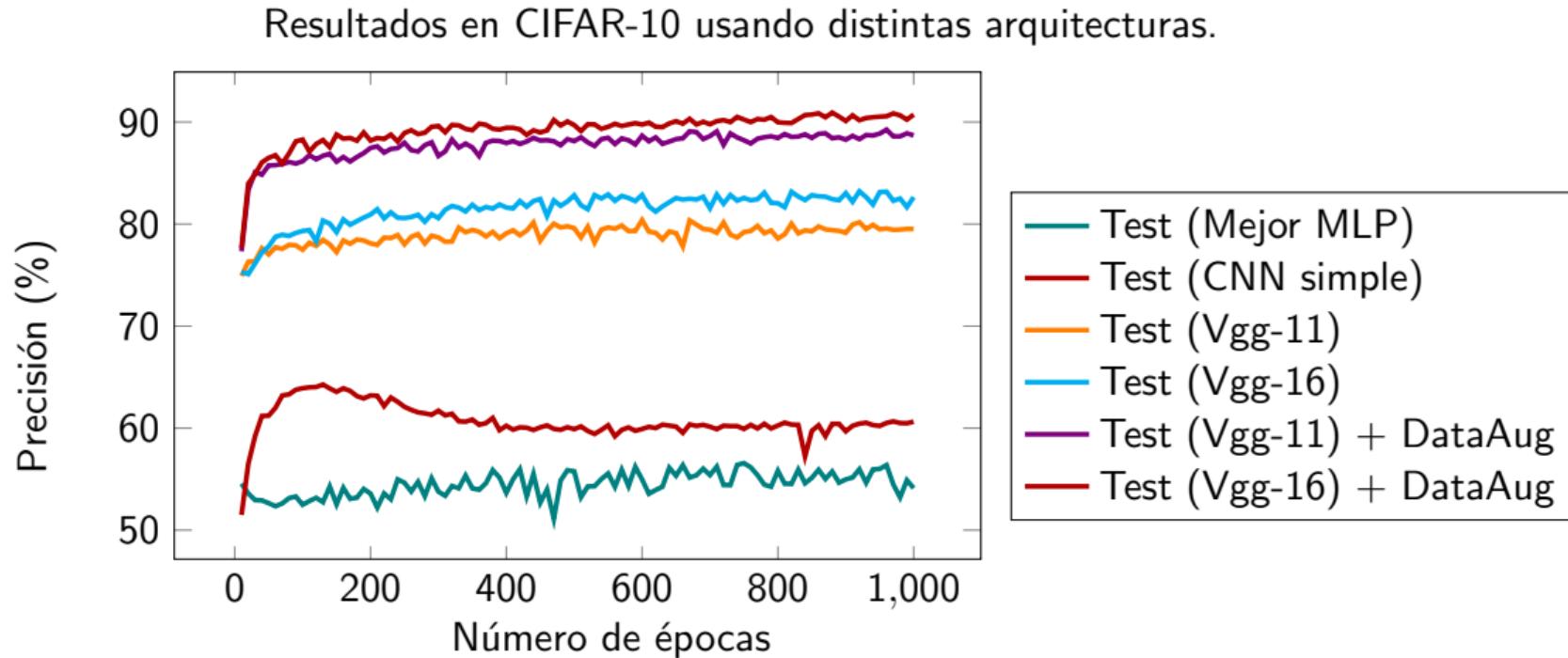
Redes Convolucionales



Redes Convolucionales



Redes Convolucionales



El mejor modelo obtuvo 90.92% en el set de test.

Ideas Finales

Ideas Finales

Los regularizadores son cambios que realizamos a nuestro algoritmo de aprendizaje para que generalice mejor – pero que no buscan minimizar el error en entrenamiento.

Ideas Finales

Los regularizadores son cambios que realizamos a nuestro algoritmo de aprendizaje para que generalice mejor – pero que no buscan minimizar el error en entrenamiento.

Vimos algunos regularizadores clásicos (que ayudan un poquito):

- Norma L^2 .
- Minimizar el batch size.

Ideas Finales

Los regularizadores son cambios que realizamos a nuestro algoritmo de aprendizaje para que generalice mejor – pero que no buscan minimizar el error en entrenamiento.

Vimos algunos regularizadores clásicos (que ayudan un poquito):

- Norma L^2 .
- Minimizar el batch size.

... y también vimos regularizadores muy efectivos:

- Forzar invarianzas en la arquitectura (ej, CNNs).
- Aumentar los datos de entrenamiento.

Ideas Finales

Los regularizadores son cambios que realizamos a nuestro algoritmo de aprendizaje para que generalice mejor – pero que no buscan minimizar el error en entrenamiento.

Vimos algunos regularizadores clásicos (que ayudan un poquito):

- Norma L^2 .
- Minimizar el batch size.

... y también vimos regularizadores muy efectivos:

- Forzar invarianzas en la arquitectura (ej, CNNs).
- Aumentar los datos de entrenamiento.

Existen muchos regularizadores. Algunos los veremos la próxima clase, y otros los invito a estudiarlos por su cuenta :)

Preguntas