Redes Convolucionales aplicadas a imágenes

Nicolas Peretti
Pablo Pastore

Casos de uso

Retrieval

Detección

Casos de uso

Segmentación

Vehículos autónomos

Casos de uso

Descripción de imágen

Reconocimiento facial

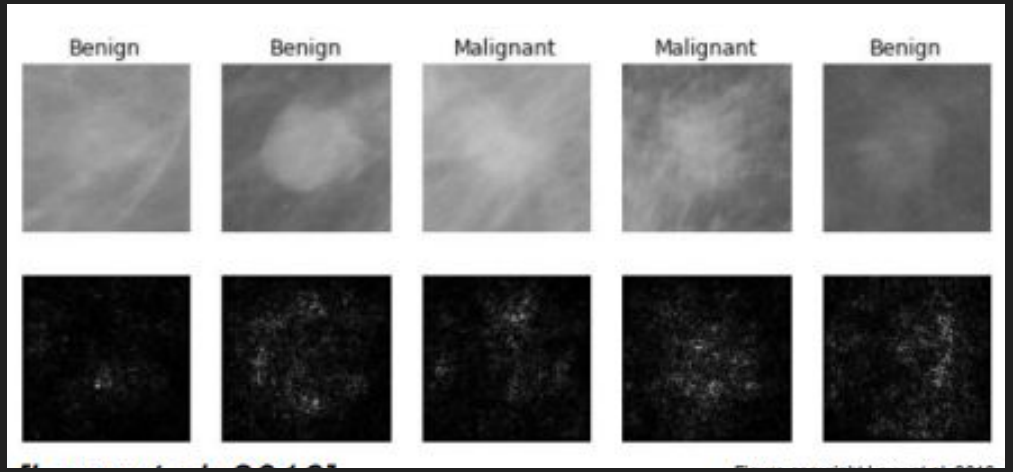
Casos de uso

Clasificación de taxonomía de galaxias

Detección de caminos

Casos de uso

Análisis de mamografía



Clasificación de señales de tránsito

Casos de uso

Estimación de pose

Casos de uso

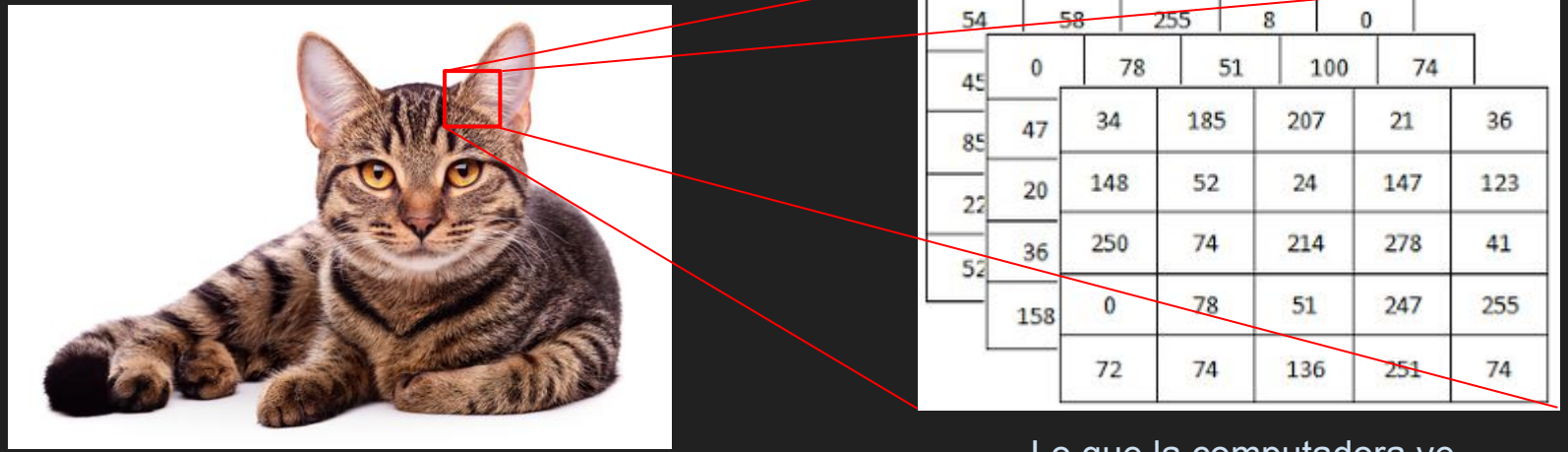
GANs

Motivación: Clasificación de imágenes

Clases fijas
{gato, avión, auto,...}

gato

Motivación: Clasificación de imágenes



Lo que la computadora ve

Motivación: Clasificación de imágenes

Algunos desafíos:

variación de clases

iluminación

deformación

oclusión

Motivación: Clasificación de imágenes

```
def predict(image):
    # ????
    return class_label
```

- ➔ Construcción robusta
- ➔ No hay forma obvia
- ➔ Depende del dominio

Usemos machine learning

- ➔ Conseguir imágenes con su clase
- ➔ Usar machine learning para entrenar un clasificador
- ➔ Evaluar el clasificador con imágenes fuera del conjunto de entrenamiento

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Primer intento: Clasificador lineal

[32, 32, 3]
(imagen=32x32 píxeles x 3 canales)

imagen parámetros

$f(x,W)$

3 números, indicando el score de cada clase

Asumimos 3 clases (eg, gato, perro, auto)

Clasificador lineal

0.2	-0.5	0.1	2.0
1.5	1.3	2.1	0.0
0	0.25	0.2	-0.3

W

*

56
231
24
2

x

\+

1.1
3.2
-1.2

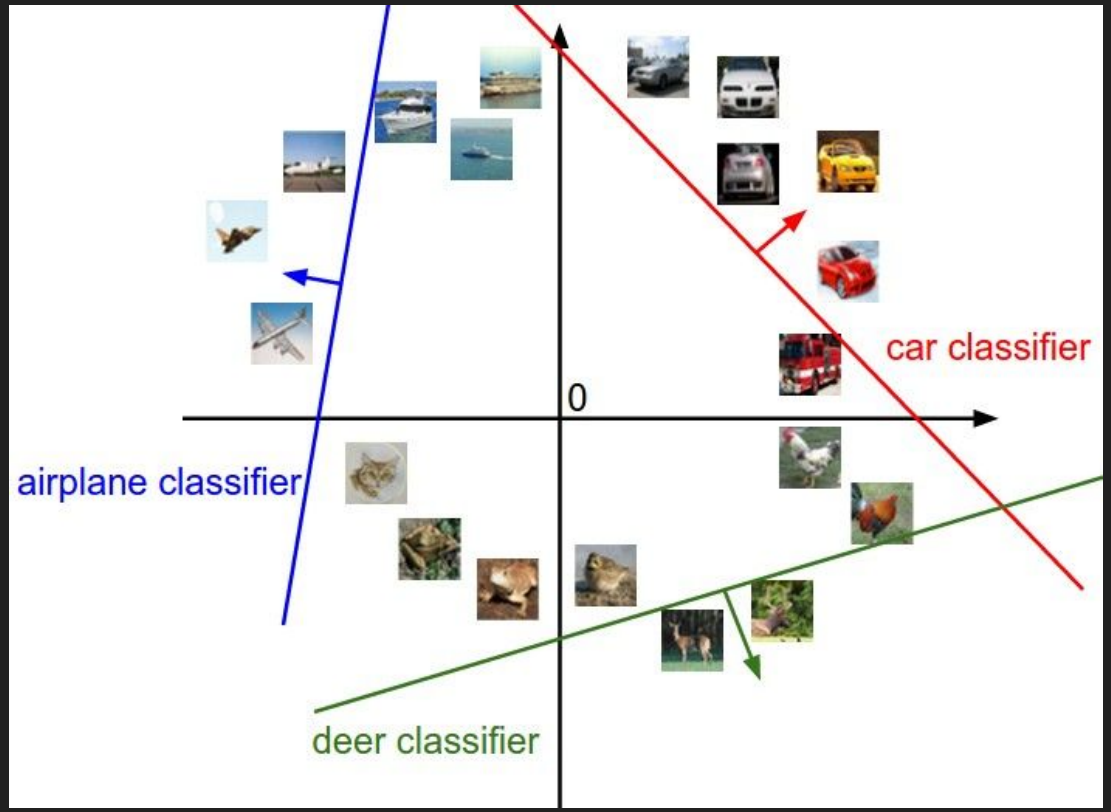
b

scores

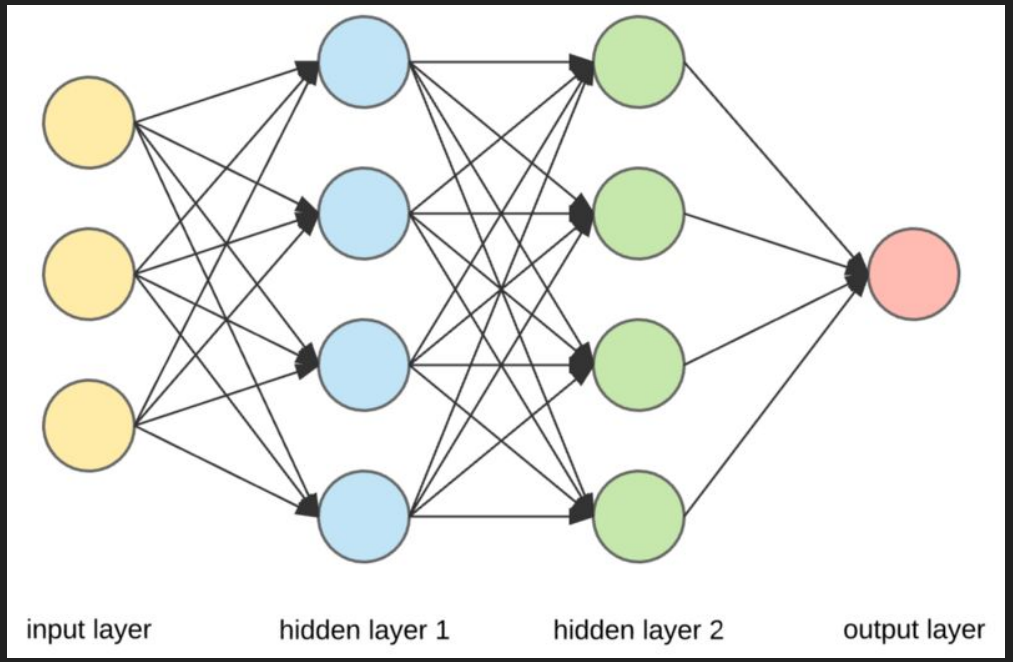
-96.8	gato
437.9	perro
61.95	auto

f(x;W,b)

Clasificador lineal

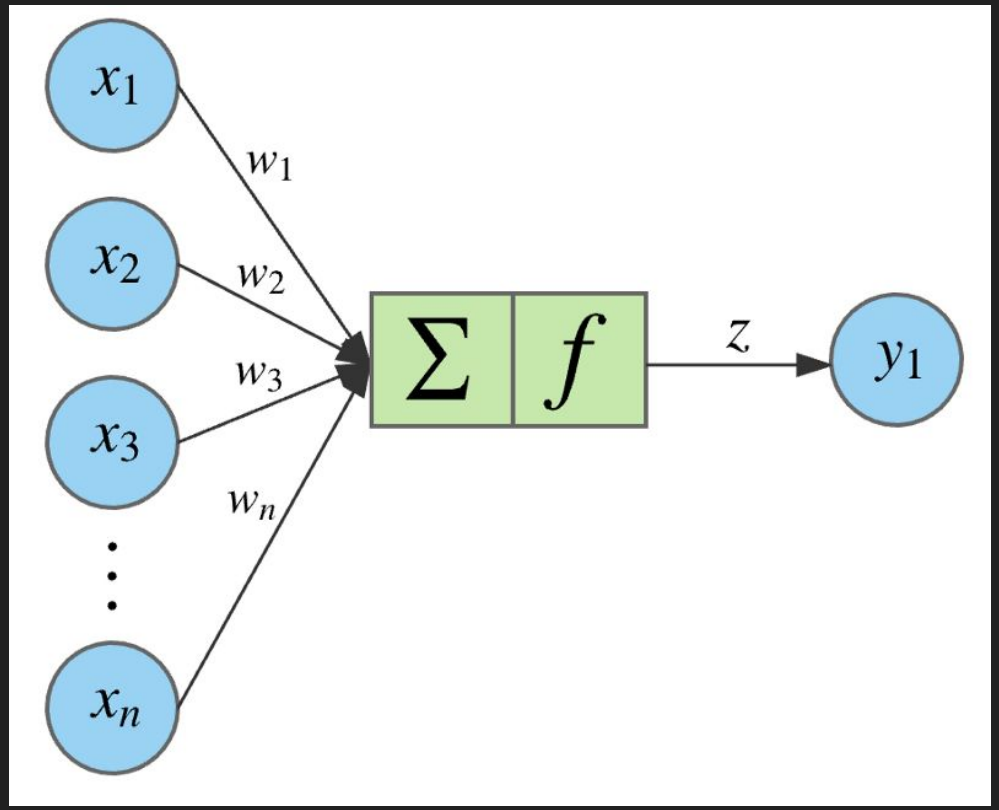


Redes Neuronales

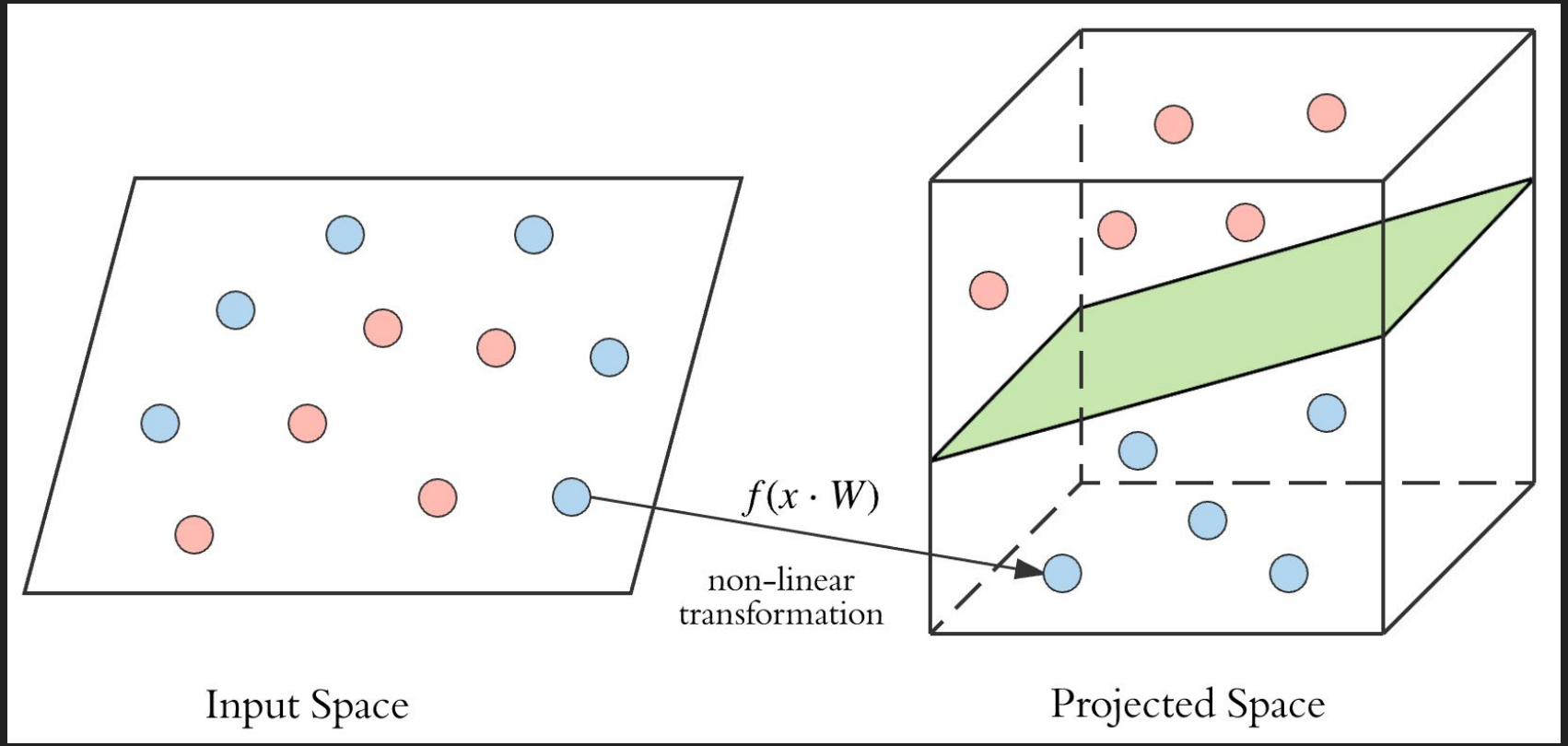


- ➔ Función lineal:
  - ◆ f = W * x
- ➔ Red neuronal de dos capas:
  - ◆ f = W’ * max(0, W * x)
- ➔ Red neuronal de tres capas:
  - ◆ f = W” * tanh(W’ * max(0, W * x))

Redes Neuronales



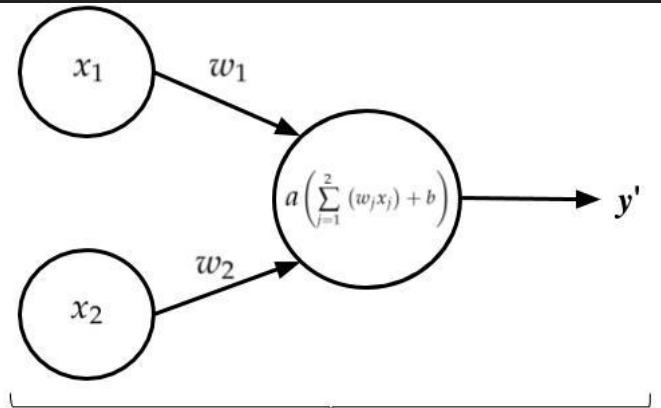
Redes Neuronales



Función de costo

x_1(Masa)	x_2(Color)	y (Etiqueta)
162	0.85	0
156	0.84	0
158	0.79	1
	...	
142	0.75	1
170	0.88	0

0 =  1 = 

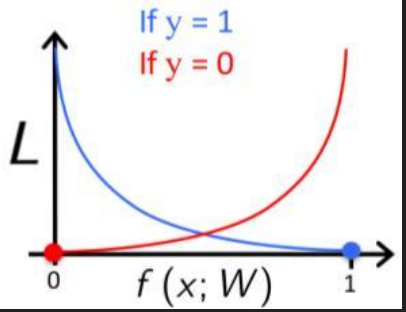


$f(x_1, x_2; W) = y'$,
$0 \le y' \le 1$, $W = [w_1, w_2, b]$

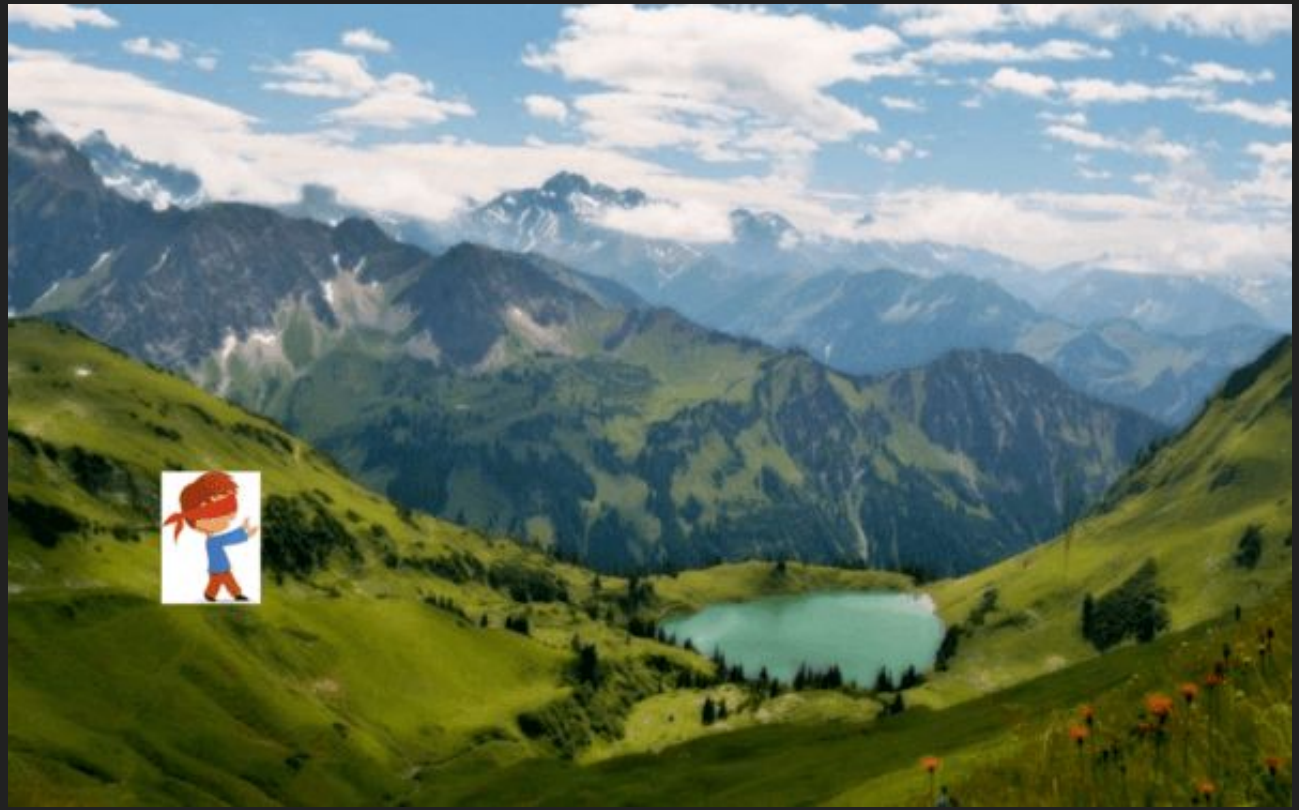
➔ Supongamos que deseamos entrenar un clasificador de frutas

➔ Debemos evaluar la exactitud del modelo: **Función de Costo**

$$L(x, y; W) = \begin{cases} -\log(f(x; W)) & y = 1 \\ -\log(1 - f(x; W)) & y = 0 \end{cases}$$

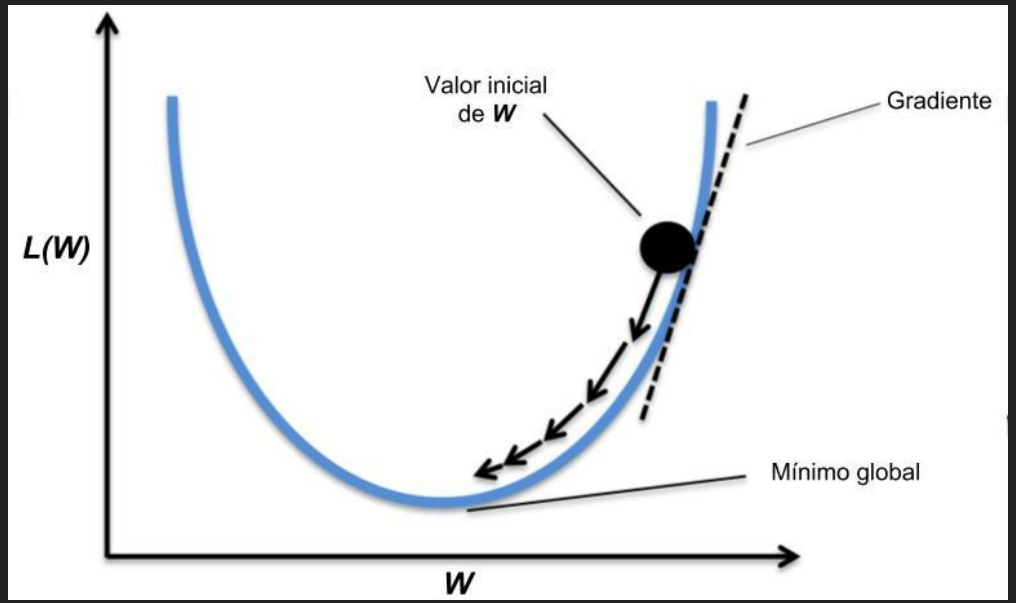


Aprendizaje por descenso del gradiente



Aprendizaje por descenso del gradiente

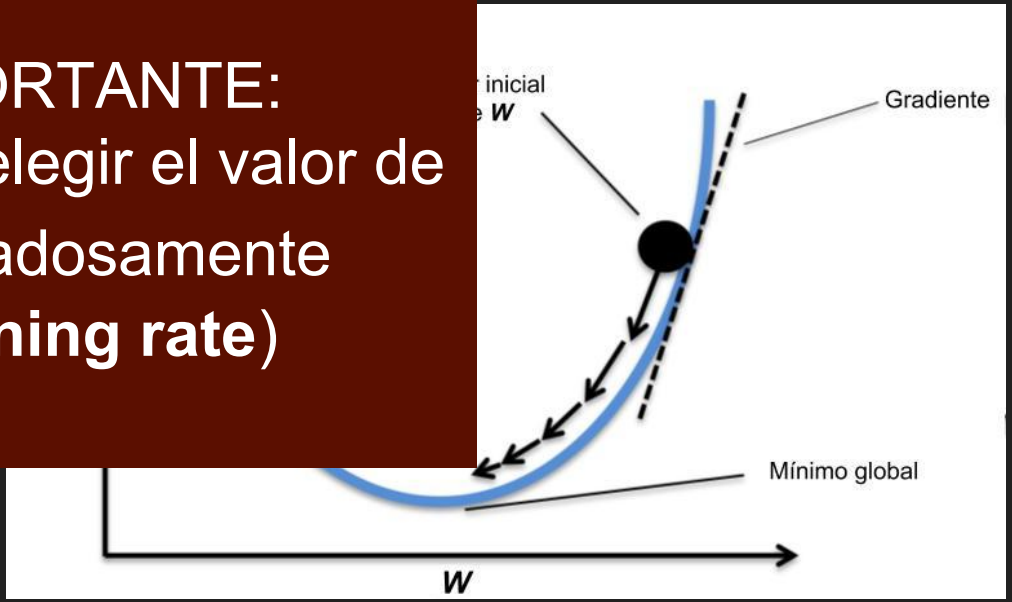
$$W_{t+1} = W_t - \delta \nabla_{W_t} L(W_t)$$



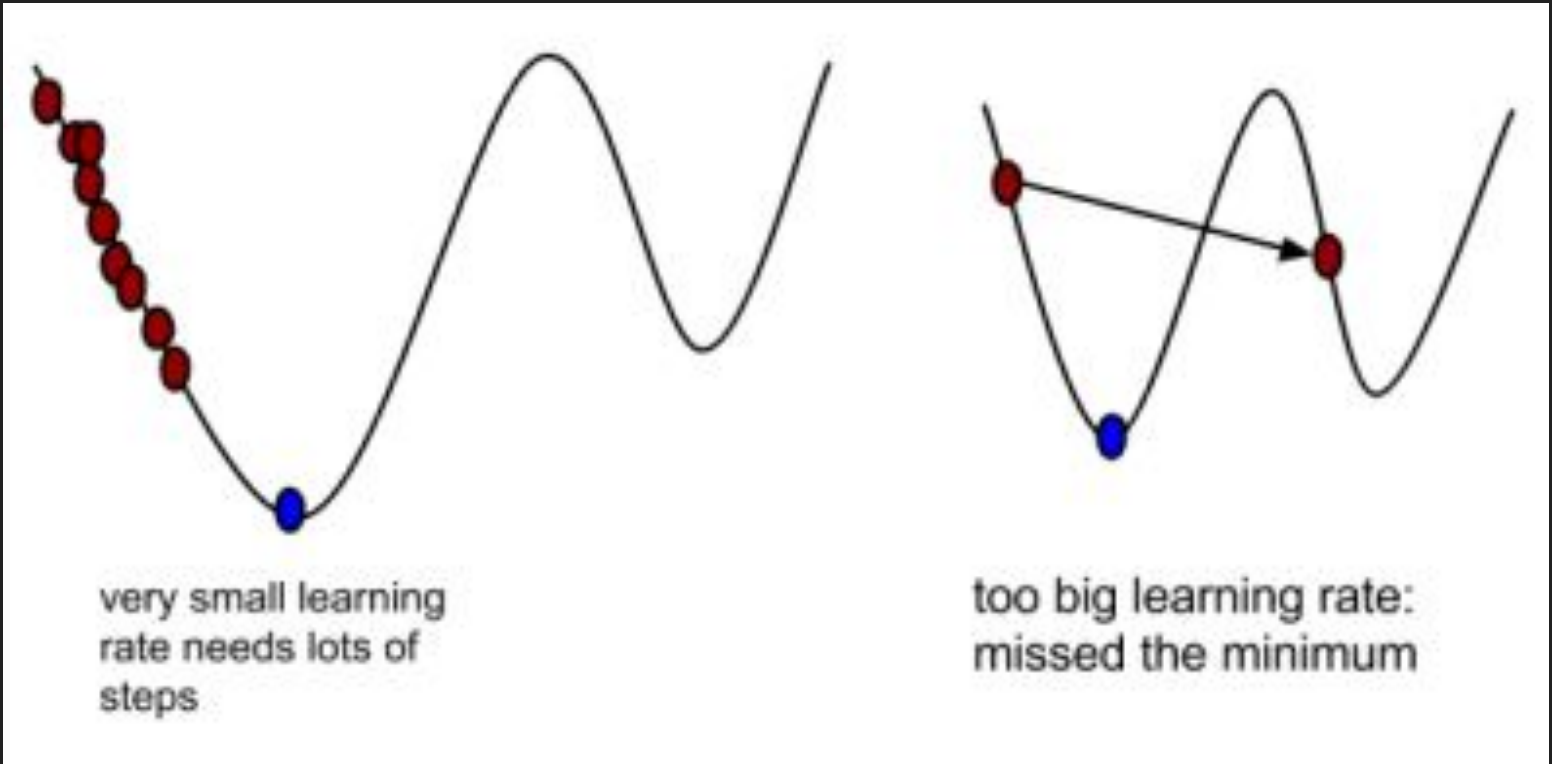
Aprendizaje por descenso del gradiente

$$W_{t+1} = W_t - \delta \nabla_{W_t} L(W_t)$$

IMPORTANTE:
Debemos elegir el valor de δ cuidadosamente (**learning rate**)



Aprendizaje por descenso del gradiente



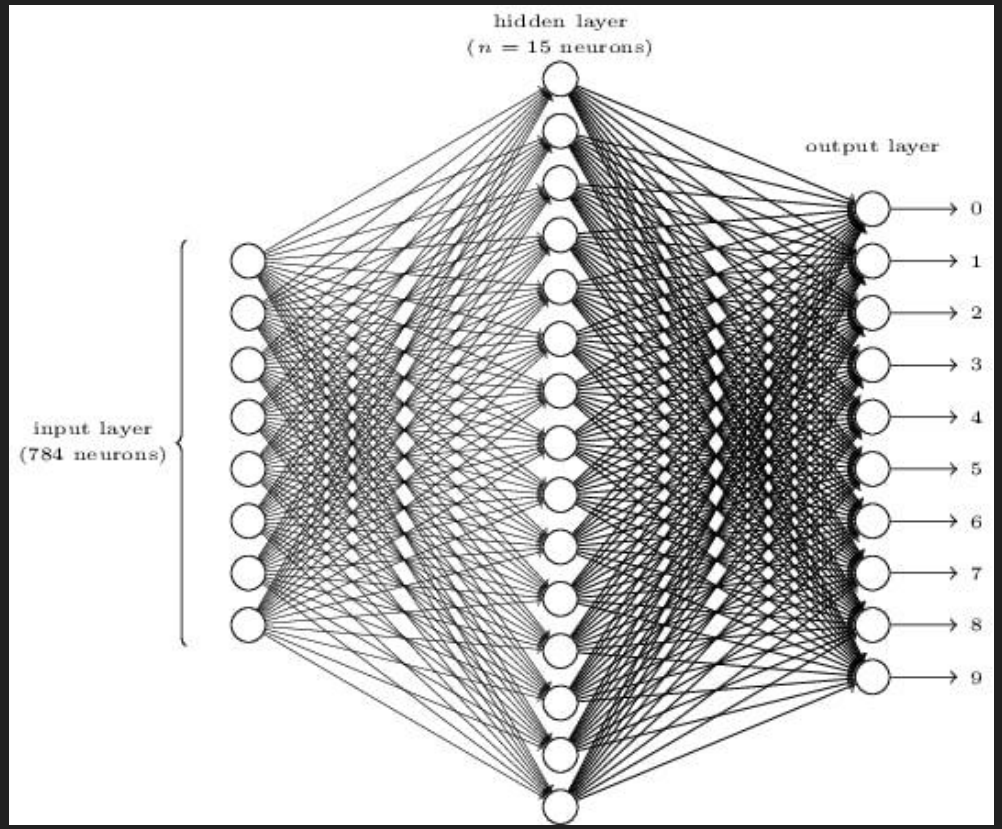
(mini batch) SGD

- Problemas con el algoritmo previo:
  - Costoso en grandes volúmenes de datos

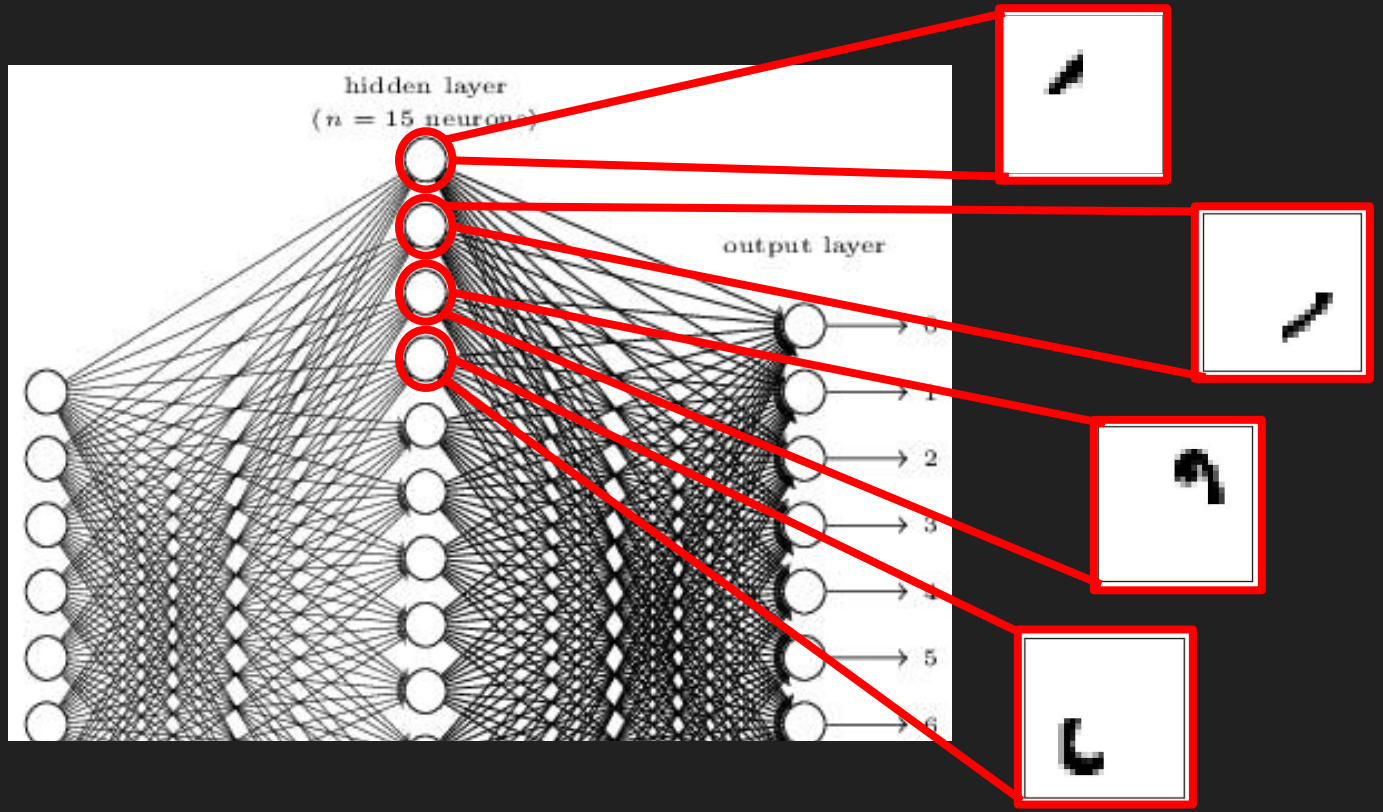
- Solución:
  - 1) Seleccionar un subconjunto de nuestro conjunto de entrenamiento.
  - 2) Computar gradiente para la función de costo en ese conjunto y actualizar pesos.
  - 3) Volver al paso (1)

Red neuronal como clasificador de dígitos

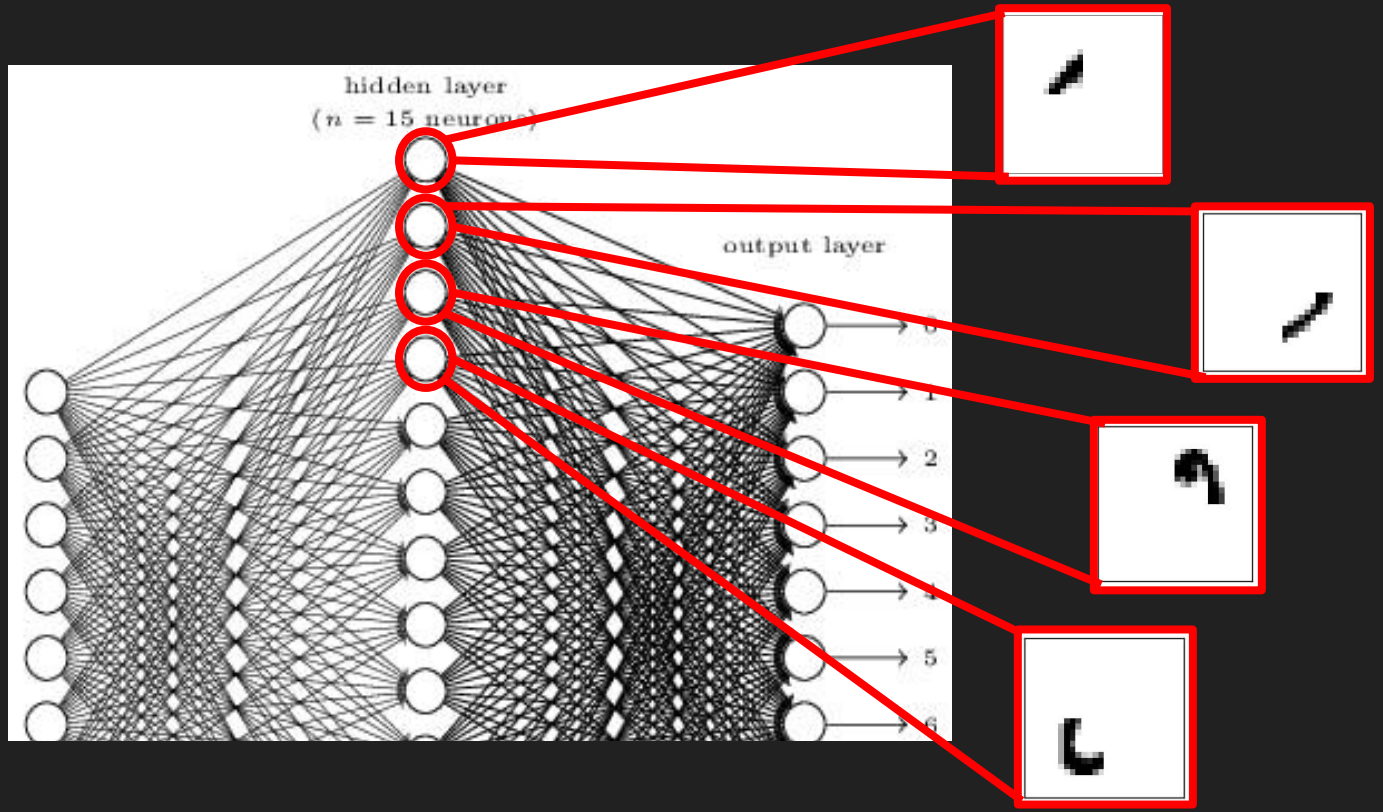
- ➔ Imagenes de dígitos de 28 x 28 píxeles
- ➔ 28 x 28 = 784 píxeles
- ➔ 784 valores de entrada a nuestra red

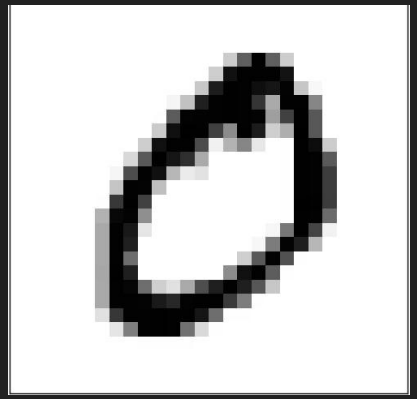


Red neuronal como clasificador de dígitos



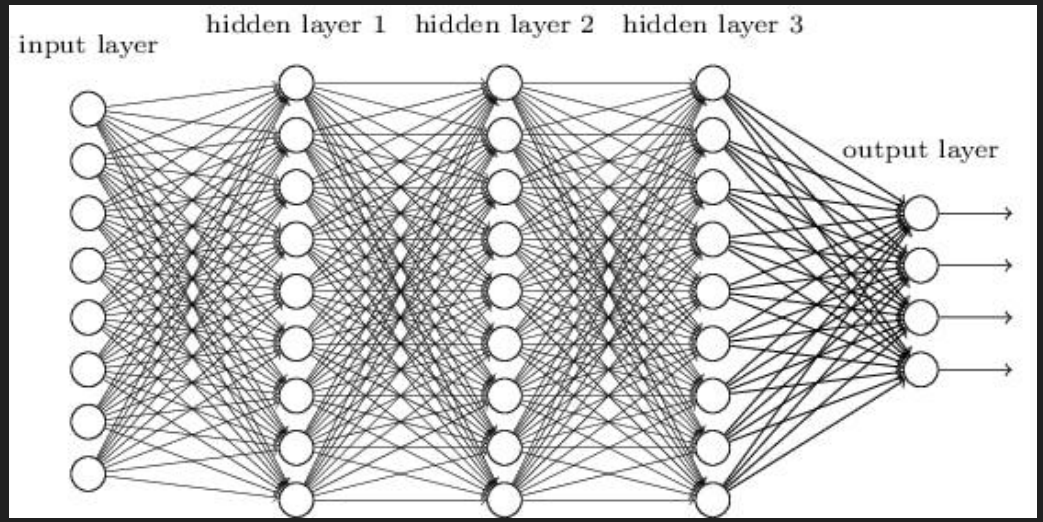
Red neuronal como clasificador de dígitos





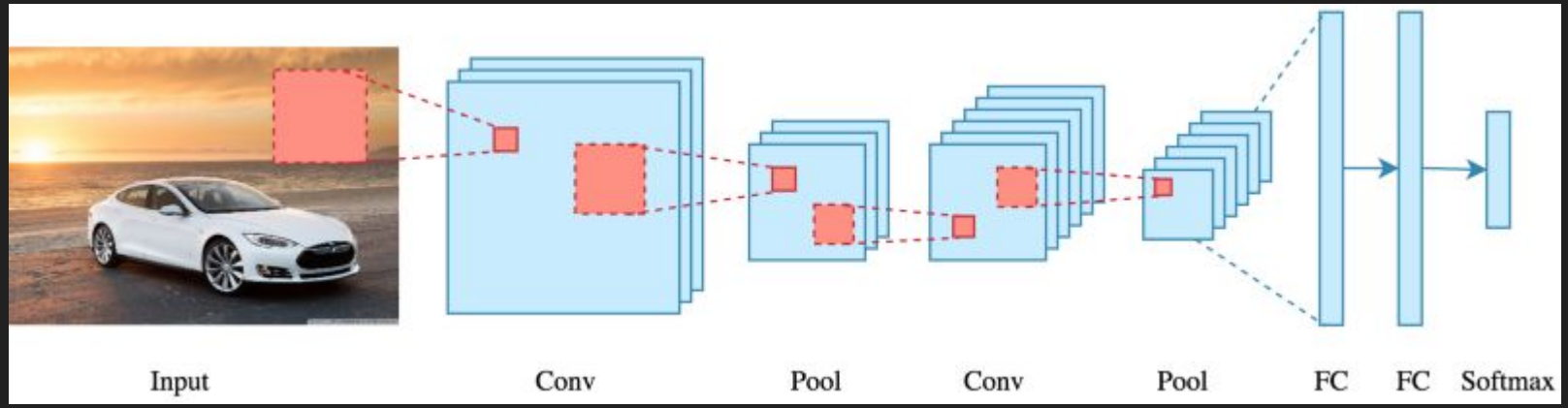
Red neuronal en imágenes

- ➔ No tiene en cuenta estructura espacial de la imagen.
- ➔ Deseamos construir redes más profundas
  - ◆ Crece la cantidad de parámetros
  - ◆ En la práctica: 3 capas mejor que 2 capas. 4, 5, etc capas no siempre ayudan

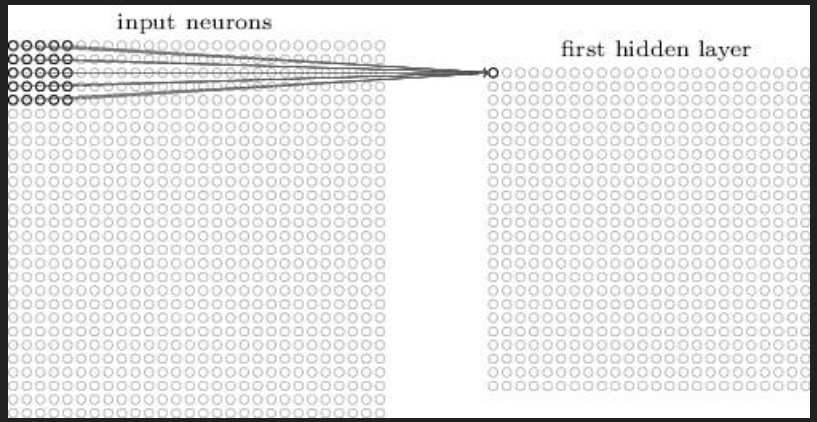


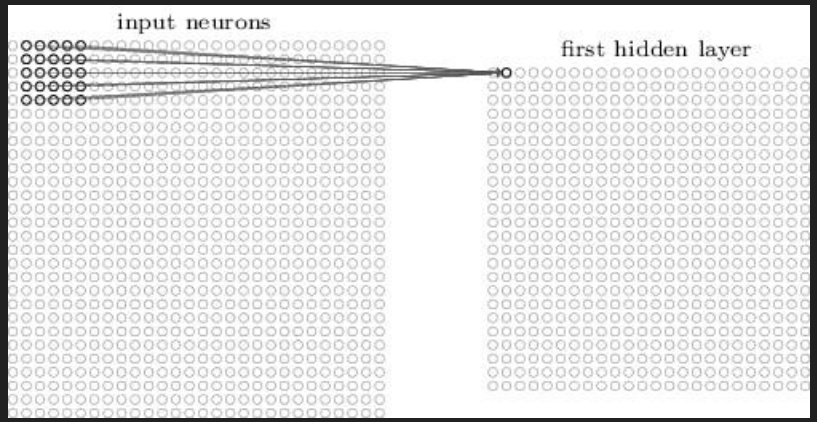
Redes convolucionales (CNN)

- ➔ Approach red neuronal: trabajar con toda la imagen a la vez
- ➔ Approach convolucional: inspeccionar la imagen de a pequeñas partes



CNN: Convolución

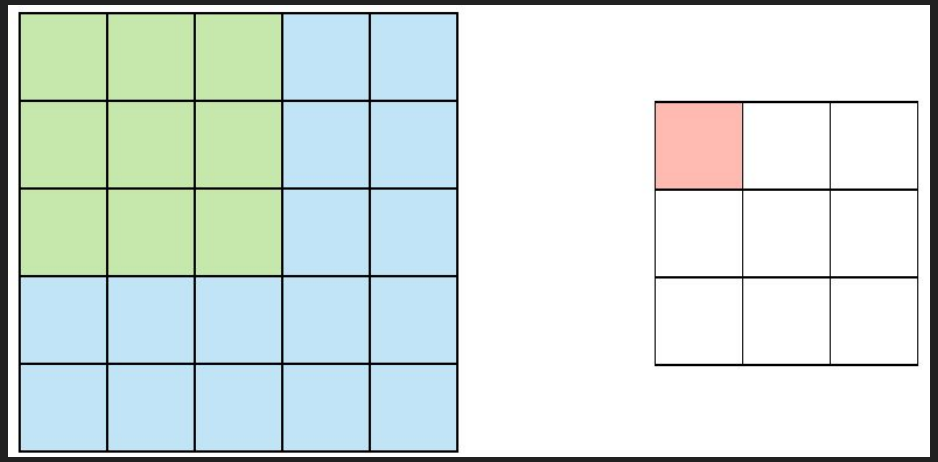




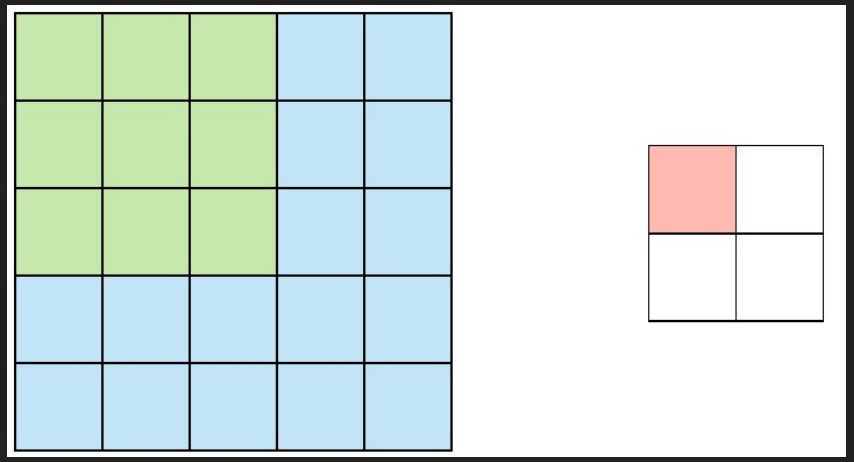
- ➔ Campo receptivo local (5x5 pixeles)
- ➔ Pesos Compartidos
- ➔ Stride = 1

CNN: Convolución

Resultados con diferentes valores de ***stride***:

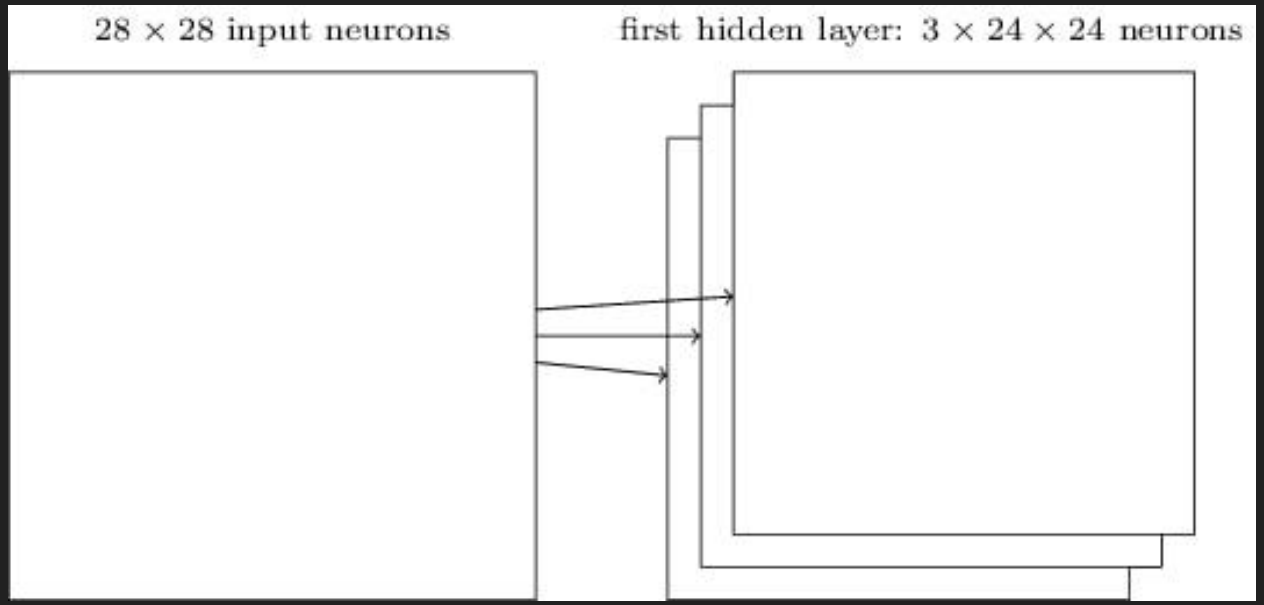


Stride = 1



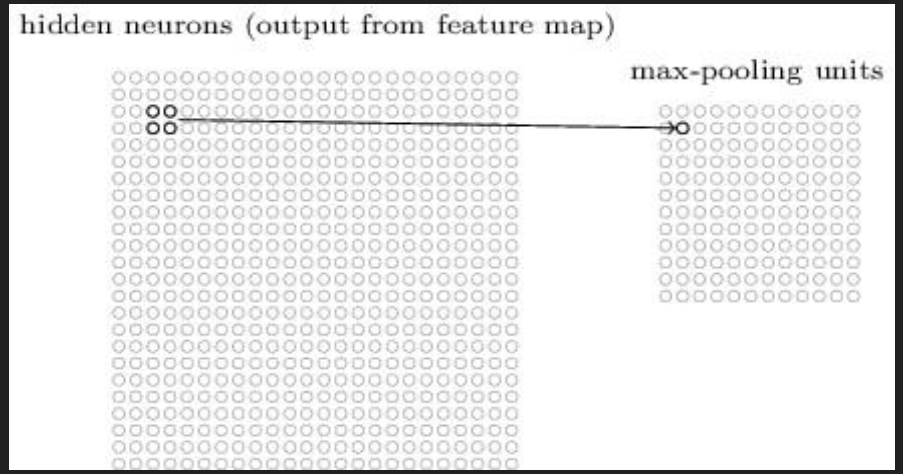
Stride = 2

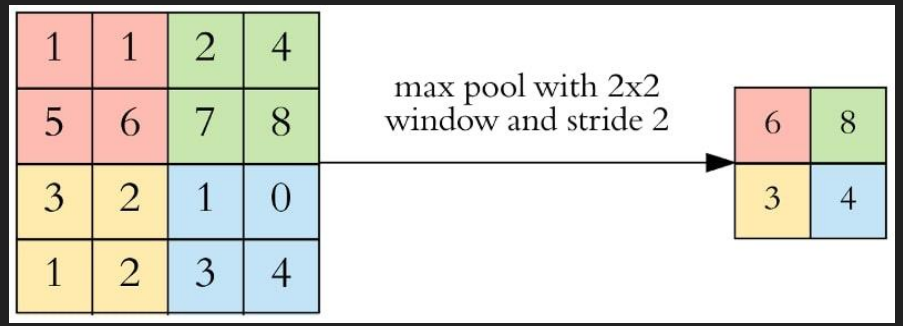
CNN: Convolución



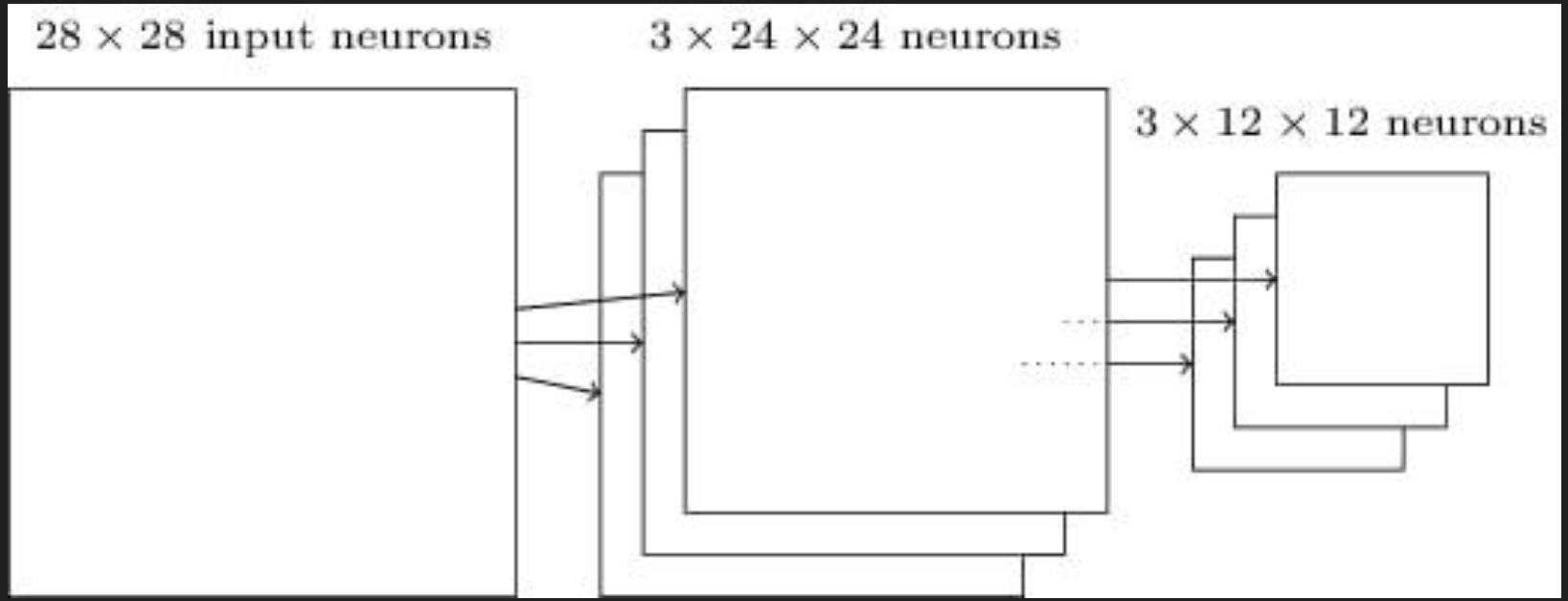
Convolución con tres filtros (o kernels)

CNN: Pooling

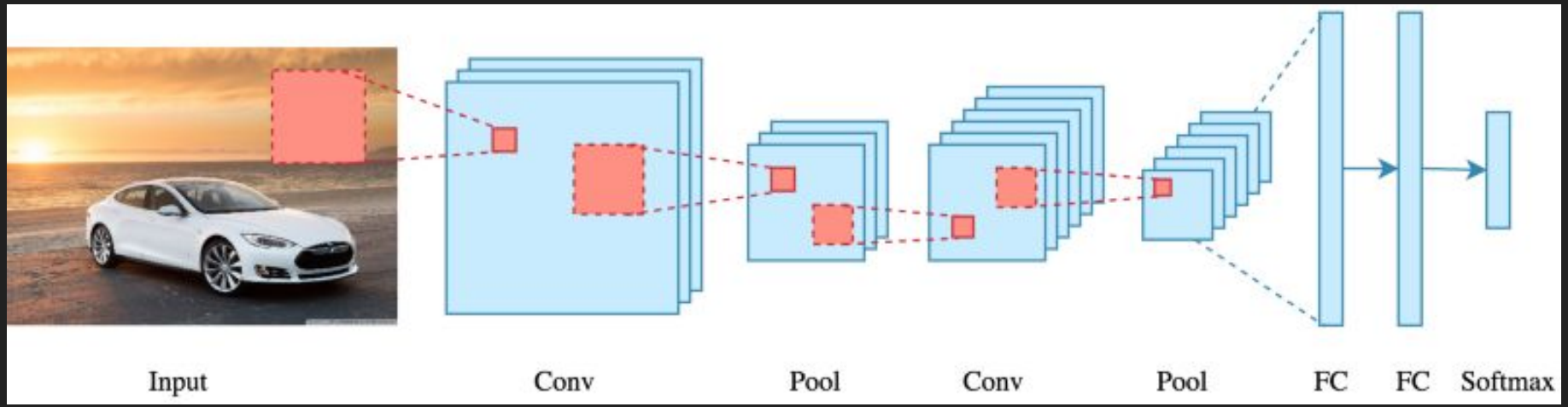

hidden neurons (output from feature map)
max-pooling units


1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2
window and stride 2
6 8
3 4

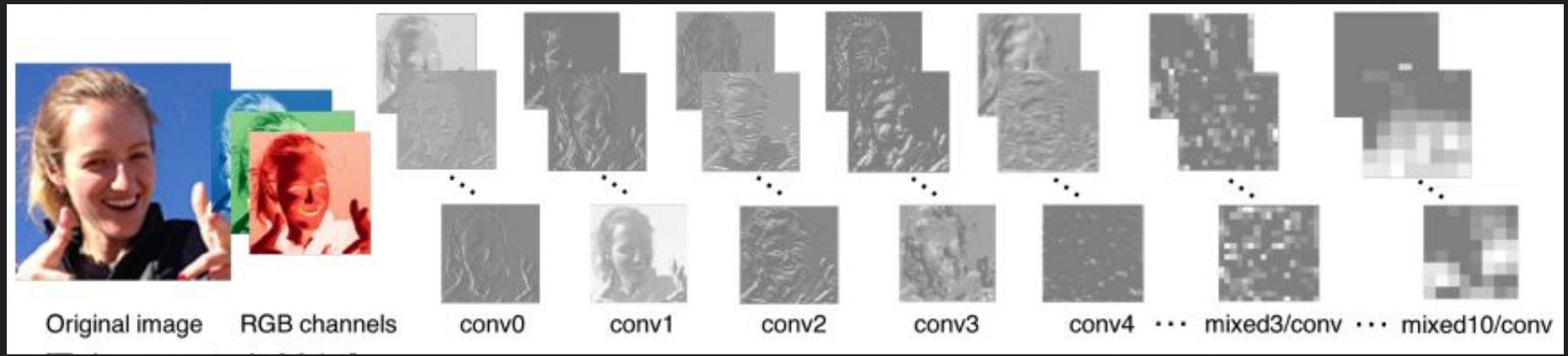
CNN: Conv + Pool



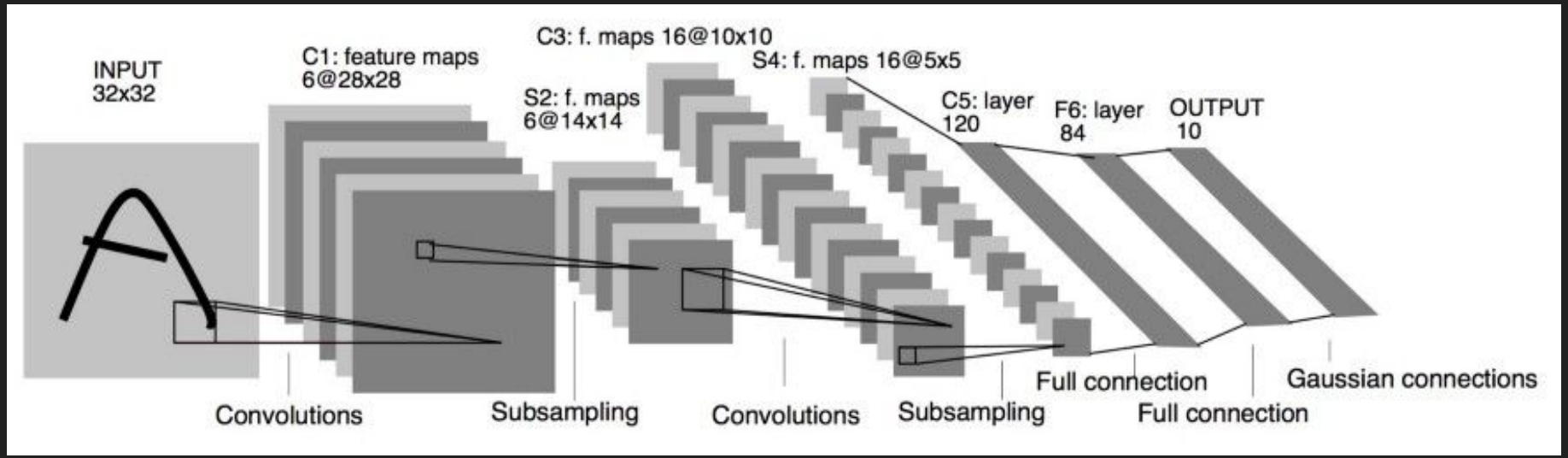
CNN: Arquitectura completa



CNN: Visualización

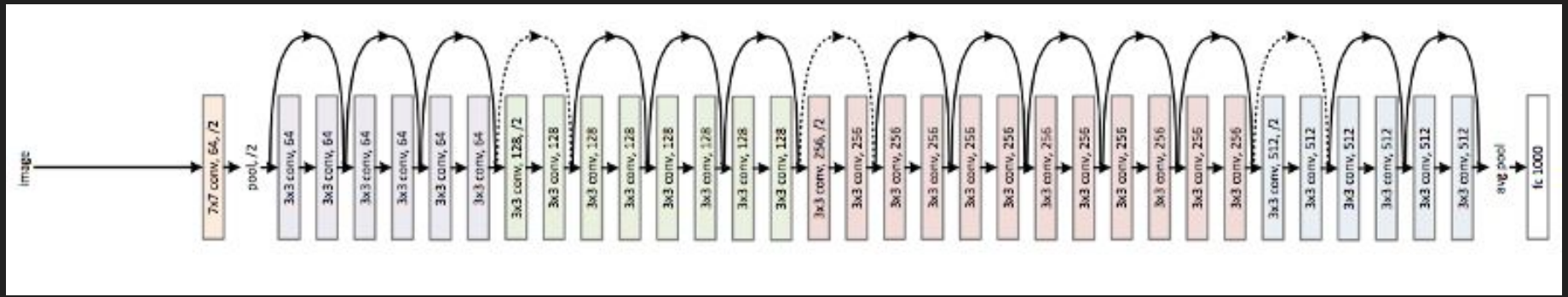


CNN: LeNet (1998)



CNN: ResNet (2015)

- ➔ 152 capas (“ultra-deep”)
- ➔ “mantener” el input original
- ➔ 8 GPUs (2->3 semanas)



CNN: ResNet

- ➔ 152 capas (“ultra-deep”)
- ➔ “mantener” el input original
- ➔ 8 GPUs (2->3 semanas)

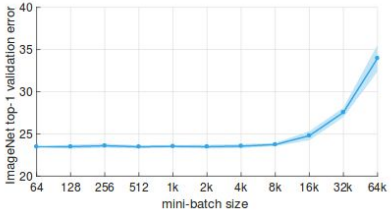
Accurate, Large Minibatch SGD:
Training ImageNet in 1 Hour

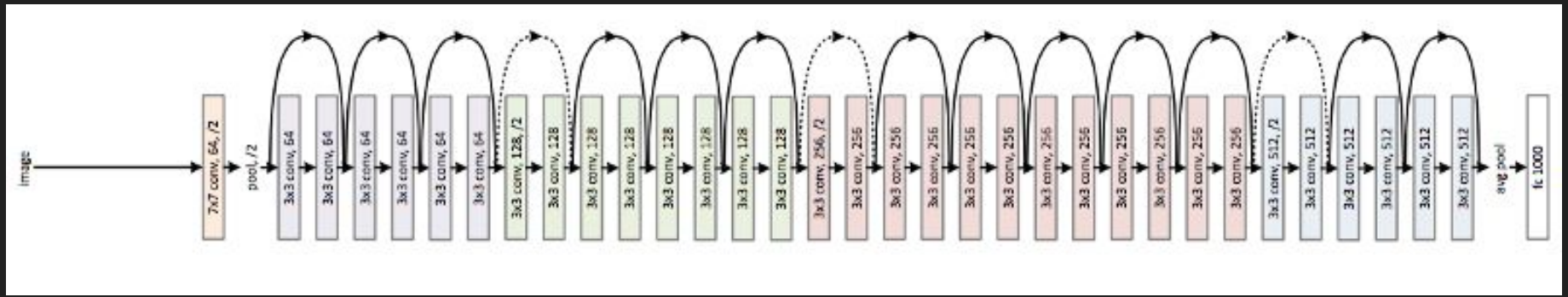
Priya Goyal Piotr Dollár Ross Girshick Pieter Noordhuis
Lukasz Wesolowski Aapo Kyrola Andrew Tulloch Yangqing Jia Kaiming He

Facebook

Abstract

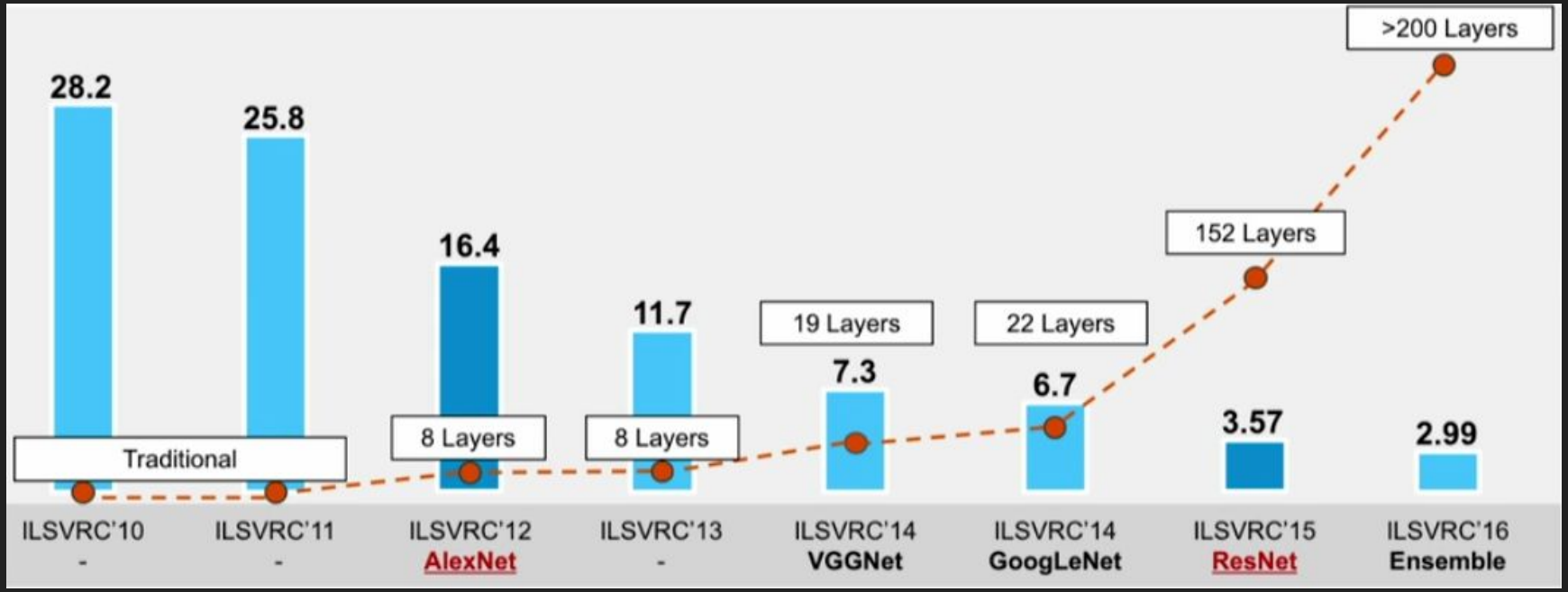
Deep learning thrives with large neural networks and large datasets. However, larger networks and larger datasets result in longer training times that impede research and development progress. Distributed synchronous SGD offers a potential solution to this problem by dividing SGD minibatches over a pool of parallel workers. Yet to make this scheme efficient, the per-worker workload must be large, which implies nontrivial growth in the SGD mini-batch size. In this paper, we empirically show that on the






ImageNet Large Scale Visual Recognition Challenges

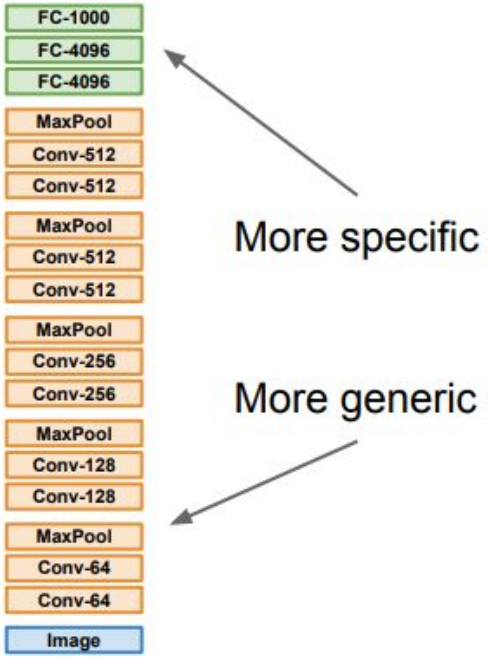
CNN: Imagenet



Imagenet: Error rate top 5

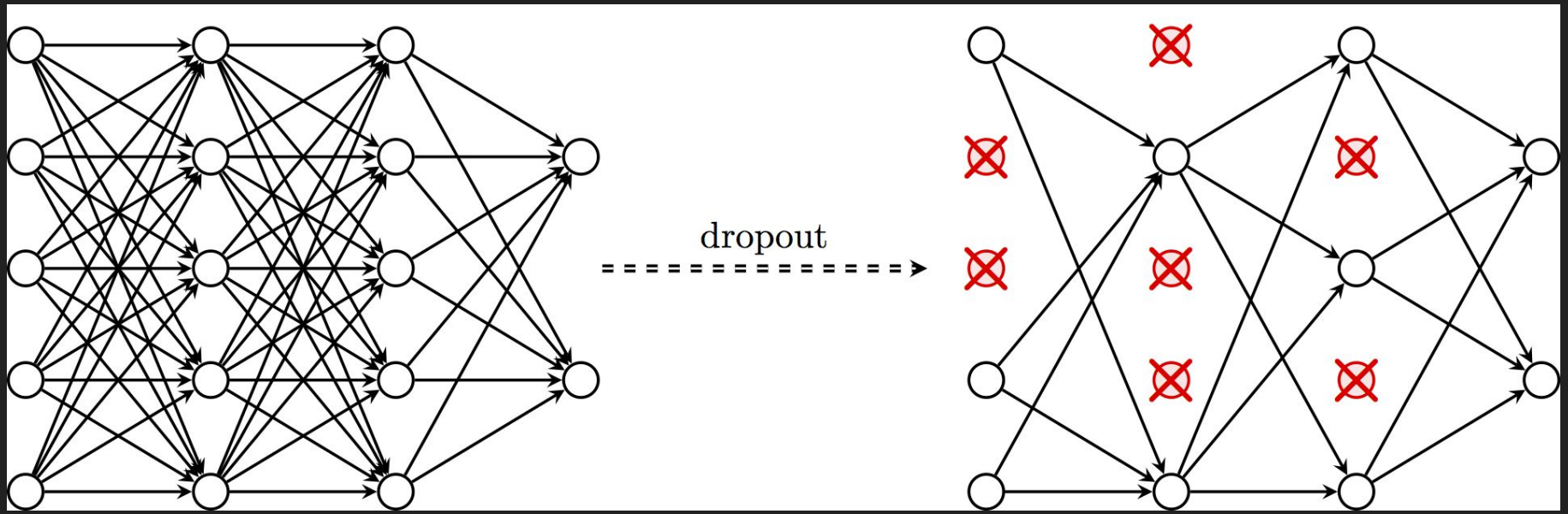
CNN: Transfer learning

~~"You need a lot of a data if you want to train/use CNNs"~~



	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Entrenamiento: Tips/Tricks

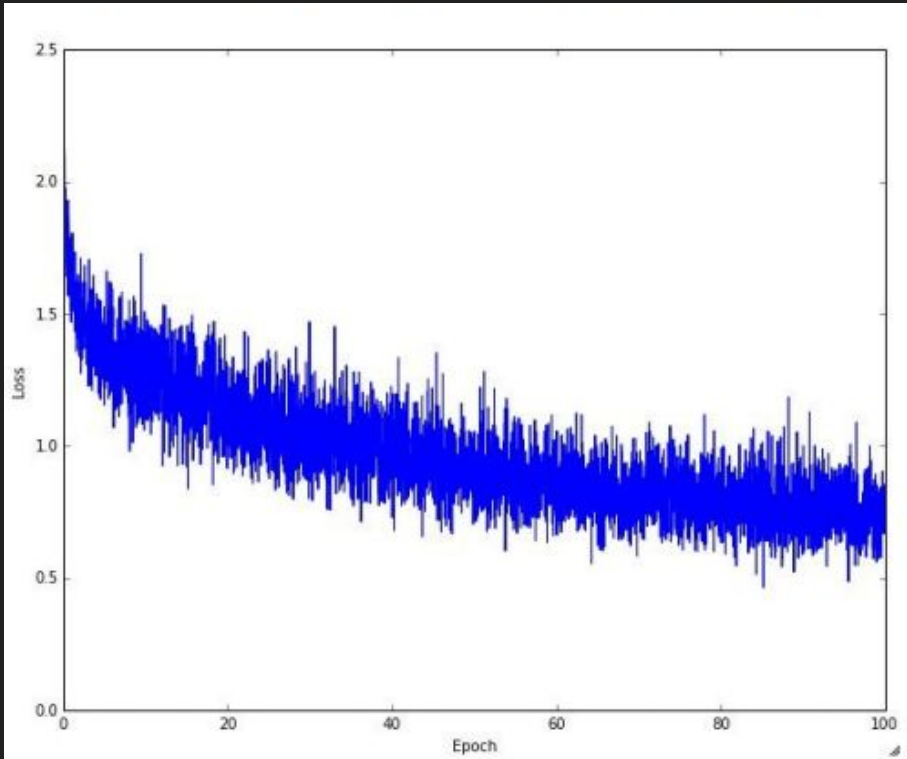


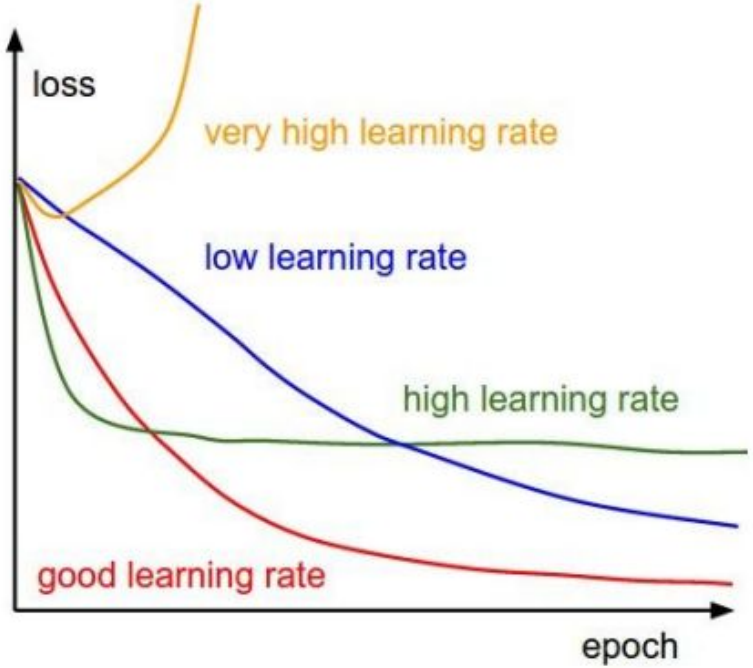
Entrenamiento: Tips/Tricks

Ajuste de hiperparametros

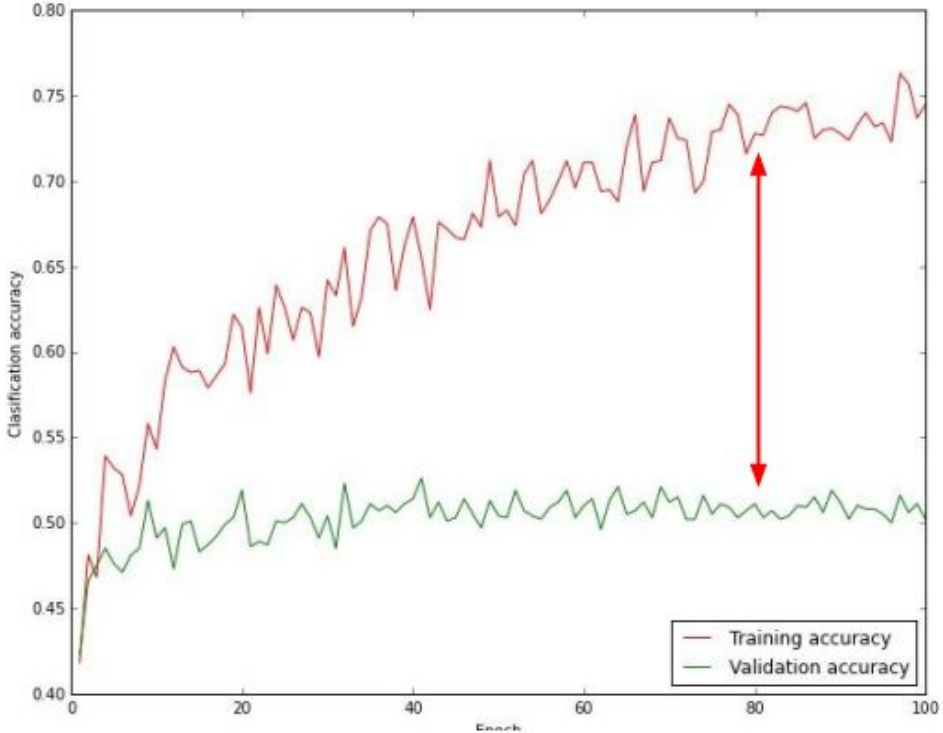
- ➔ Arquitectura de red neuronal.
- ➔ Learning rate inicial, actualización.
- ➔ Regularización (ej. dropout).
- ➔ Más: https://goo.gl/p5Voeh

Entrenamiento: Tips/Tricks





Entrenamiento: Tips/Tricks



big gap = overfitting

=> increase regularization strength?

no gap

=> increase model capacity?

Más info:

- Intro a CNNs: http://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
- CS231n
  - Course: http://cs231n.stanford.edu/
  - Videos: https://www.youtube.com/playlist?list=PL16j5WbGpaM0_Tj8CRmurZ8Kk1gEBc7fg
- Michael Nielsen's Book: http://neuralnetworksanddeeplearning.com/
- Deep Learning Book: https://www.deeplearningbook.org/
- Andrej Karpathy's Blog: http://karpathy.github.io/

¿Preguntas?