Aprendizaje Automático Profundo

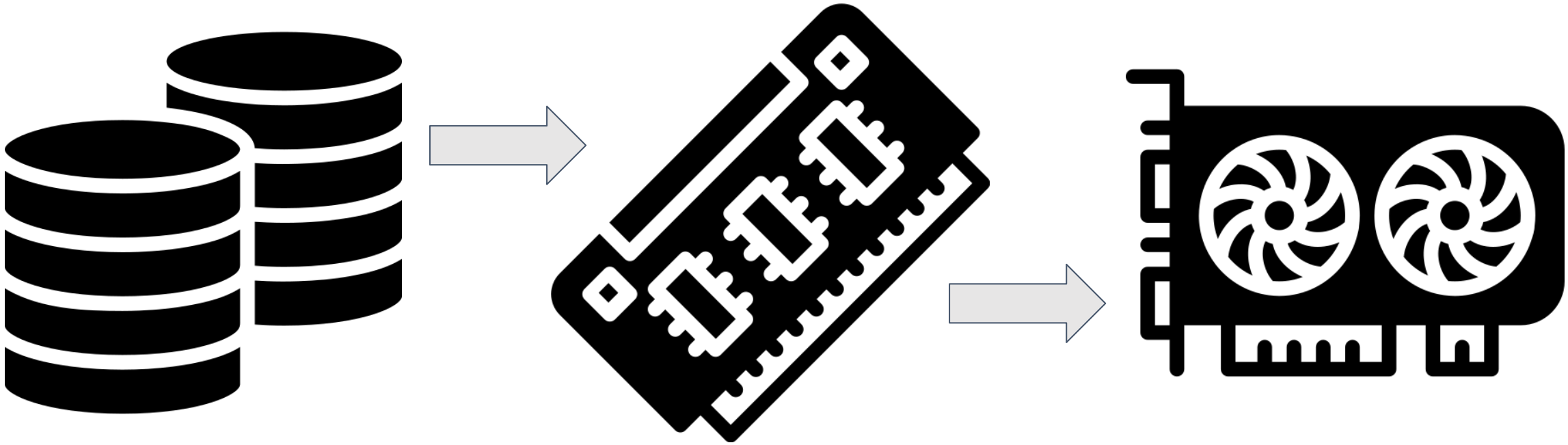
Clase 8 - Tópicos Avanzados

Profs: Franco Ronchetti - Facundo Quiroga

Carga de datos

¿Cómo cargar conjuntos de datos?

- Dos enfoques principales
  - a) Cargar todo en memoria (si entra)
  - b) Cargar *batchs* a medida que se usan
    - Libero memoria de batches usados
    - Múltiples threads si la carga es lenta



¿Cómo cargar datos para entrenar?

- Cargar todo en memoria (si entra)
  - Es el enfoque que usamos hasta ahora con MNIST, CIFAR10 y todos los conjuntos de las prácticas 2 y 3

```
#cargar todos los datos en memoria
x,y=load_data()
print(x_shape[0]) # 10000 ejemplos


…
model= …
# Entrenar con 32 ejemplos por vez
model.fit(x,y,batch_size=32)
```

¿Cómo cargar datos para entrenar?

- Utilizando un *generator*

```
def generator(batch_size):
   n= 10000
   while True: # generador infinito
      for i in range(n//batch_size):
         index=i*batch_size
         x,y=load_samples(from=index,to=index+batch_size)
         yield x,y
model= …
# Entrenar con 32 ejemplos por batch
# Carga a memoria a medida que se usa
model.fit_generator(generator(32),
                    steps_per_epoch=10000/32,...)
```

load_samples carga las imágenes desde **from** hasta **to** desde disco

generator es infinito => steps_per_epoch para saber cuantos batchs por época

¿Cómo cargar datos para entrenar?

- Versiones de *fit*, *evaluate* y *predict* con generators
  - Implementadas en clase *Sequential*

```
#todos los datos en memoria
x,y=load_data()
…
model.fit(x,y,batch_size=32)
…
model.predict(x,y,batch_size=32)
…
model.evaluate(x,y,batch_size=32)
```

```
def generator(batch_size):
   …
g=generator(32)
model.fit_generator(g)
…
model.predict_generator(g)
…
model.evaluate_generator(g)
```

¿Cómo cargar datos para entrenar?

- Utilizando un *generator* subclasificando *Sequence*

```
class CustomSequence(Sequence):
   def __init__(self, batch_size):
      self.batch_size=batch_size
      self.n=10000
   def __len__(self): # cant de batches
      return self.n // self.batch_size
   def __getitem__(self, idx):
      index=idx*batch_size
      x,y=load_samples(from=index,to=index+batch_size)
      return x,y
…
model.fit_generator(CustomSequence(32))
```

- Carga de datos *multithreaded*
  - Carga y entrenamiento en paralelo

No requiere `steps_per_epoch`

¿Cómo cargar datos para entrenar?

- Eficiencia de **fit_generator**

- Sin multithreading
  - **def generator()**

- Con multithreading
  - clase **Sequence**

¿Cómo cargar datos para entrenar?

- *Generators* para imágenes incluidos en Keras

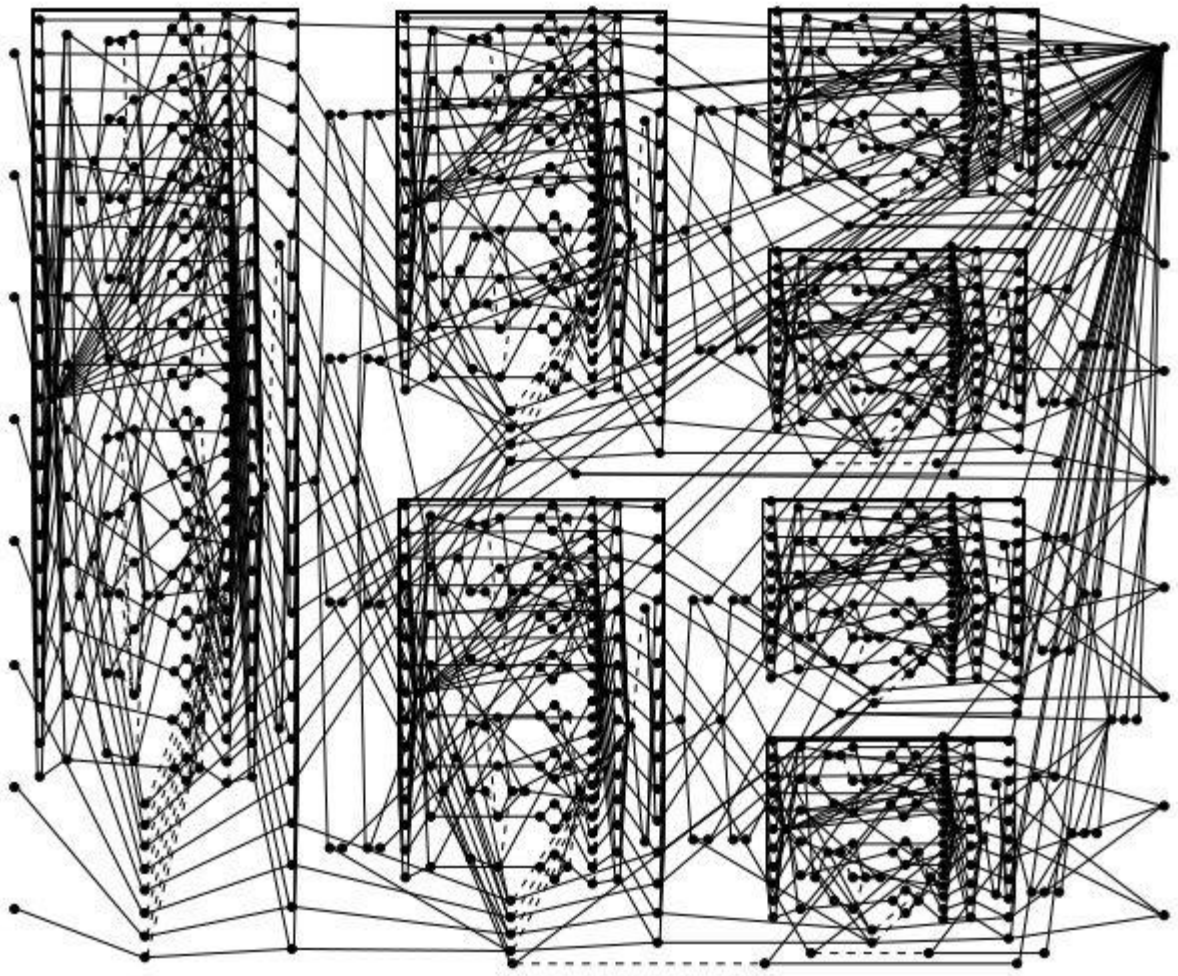
```
datagen = ImageDataGenerator()
train_generator = train_datagen.flow_from_directory(
        'data/train', # dir de las imagenes
        target_size=(150, 150),# resize tam destino
        batch_size=32,
        class_mode='categorical')
model.fit_generator(train_generator,
        steps_per_epoch=len(train_generator)/32,...)
```

- Por cada clase, una subcarpeta con imágenes.
  - class_mode = ‘sparse’ => Codificar clase con *labels,* enteros
  - class_mode = ‘categorical’ => Codificar clase con vectores one_hot
  - class_mode = None => No cargar la clase

Mejora de Modelos

¿Cómo mejorar un modelo?

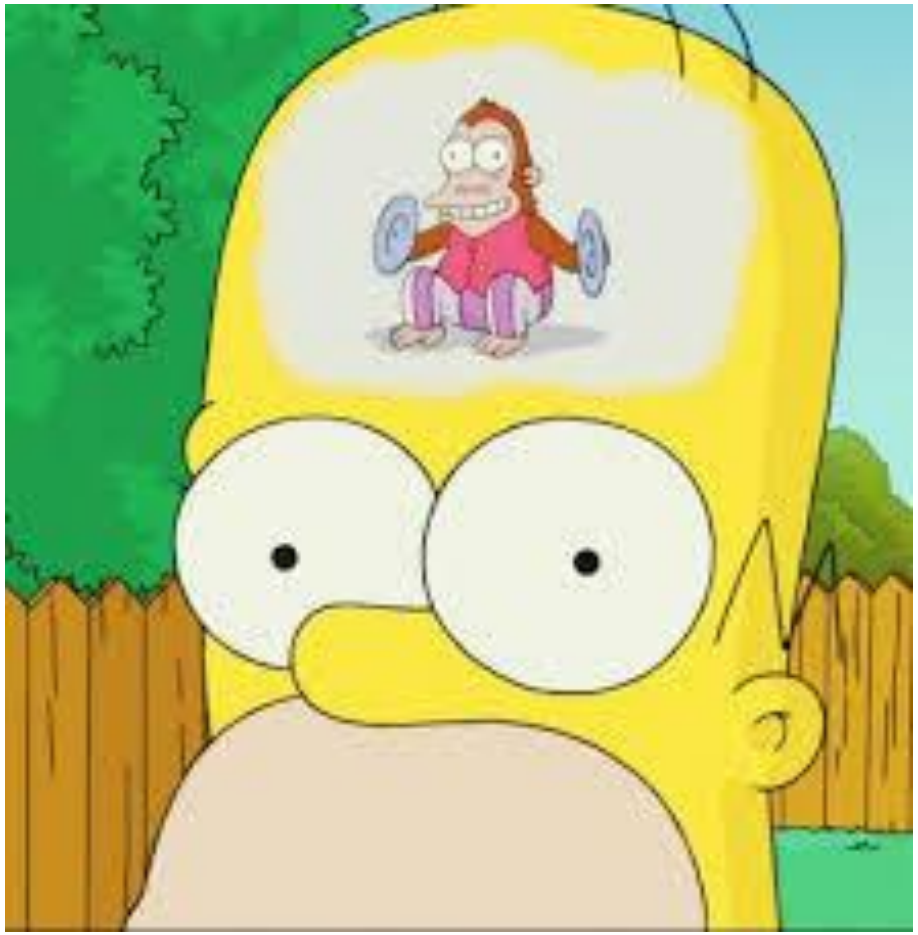
- Opción A
- Diseñar un modelo mejor.
- Requiere:
  - Conocimiento
  - Inteligencia
  - Mucho tiempo/trabajo.



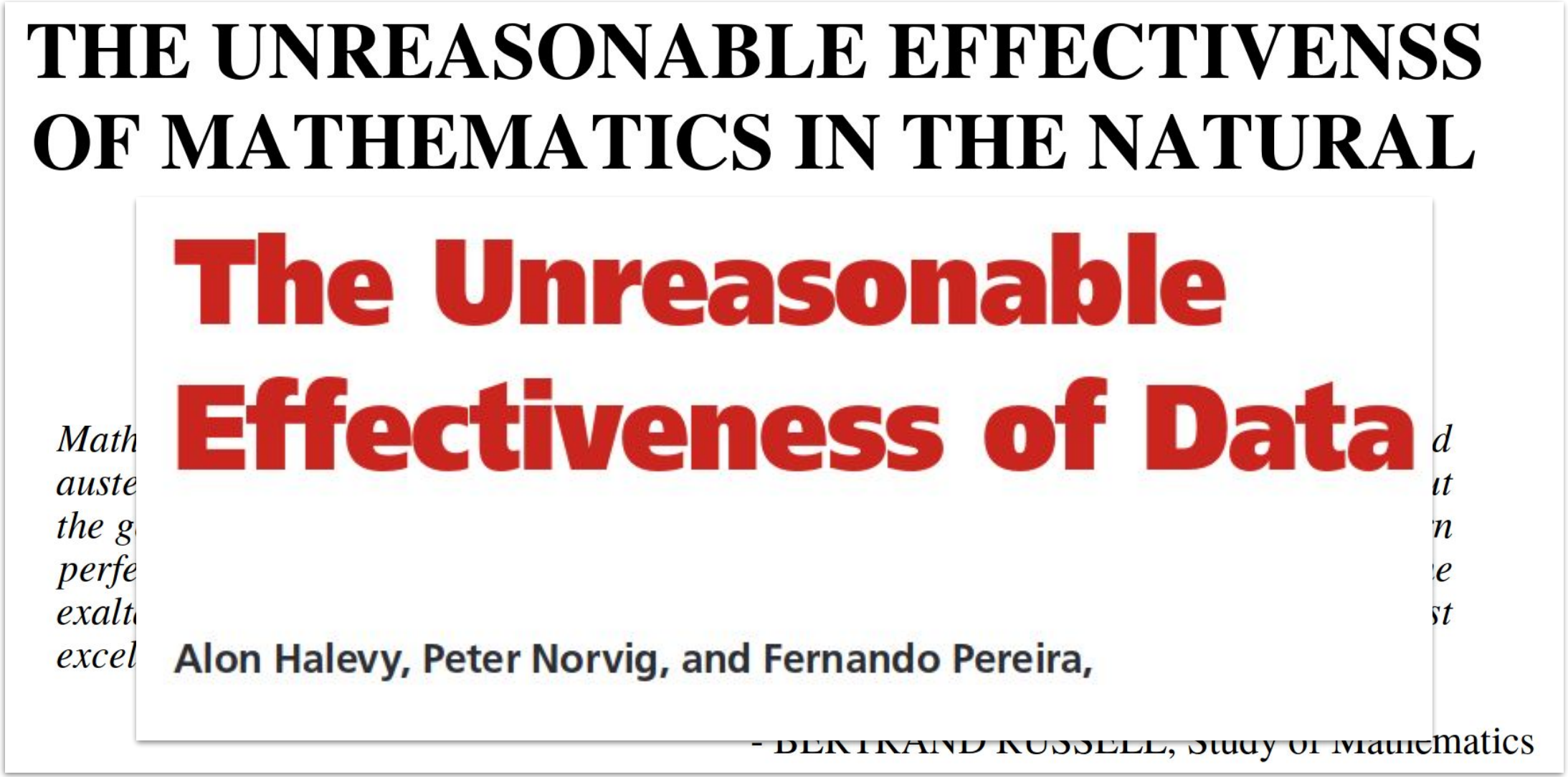
fuente

¿Cómo mejorar un modelo?

- Opción B
- Entrenar un modelo común con más datos.
- Requiere
  - Más datos
- Primera estrategia a probar

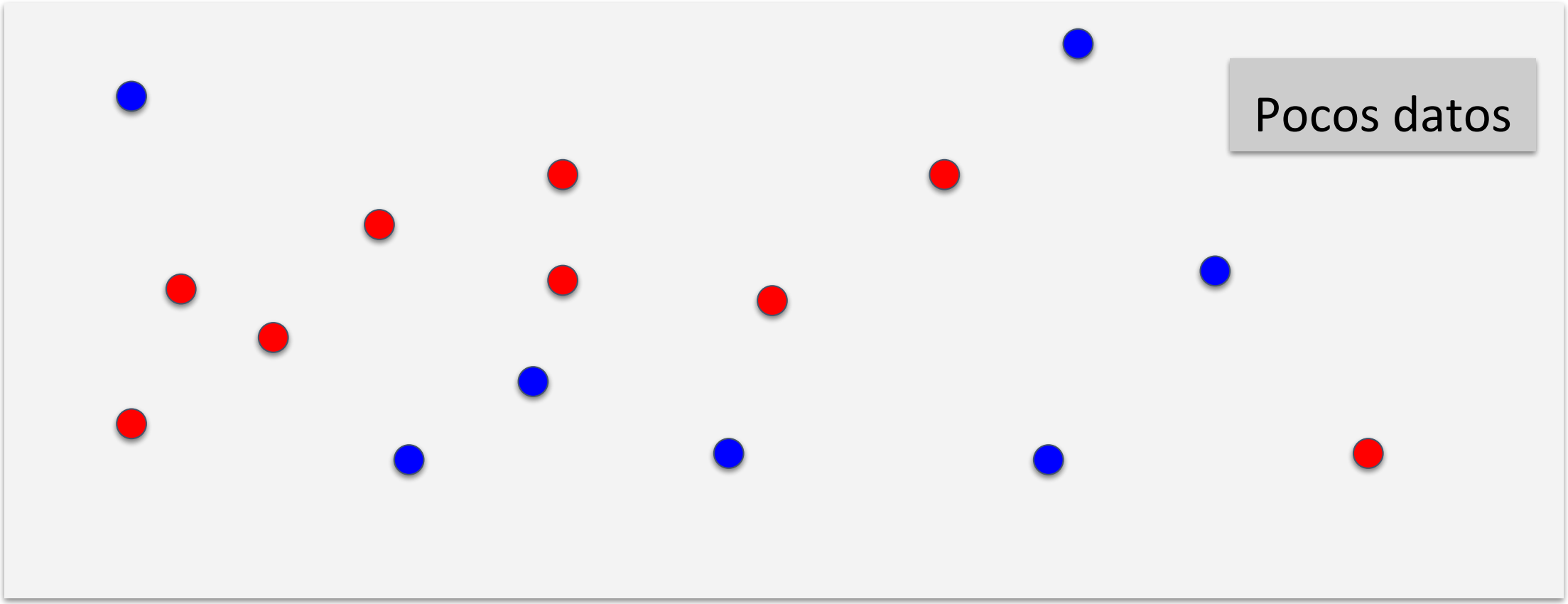


Error vs #ejemplos - Motivación



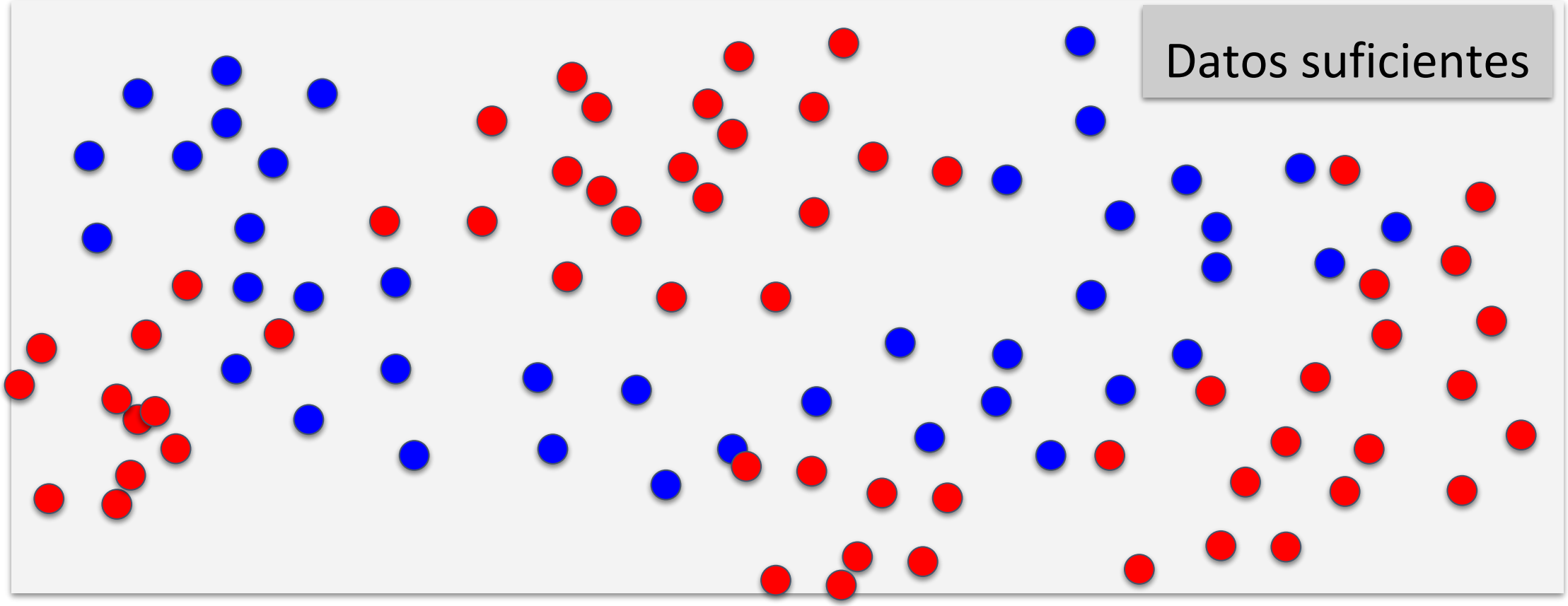
Error vs #ejemplos - Motivación

Los datos son el mejor regularizador del modelo



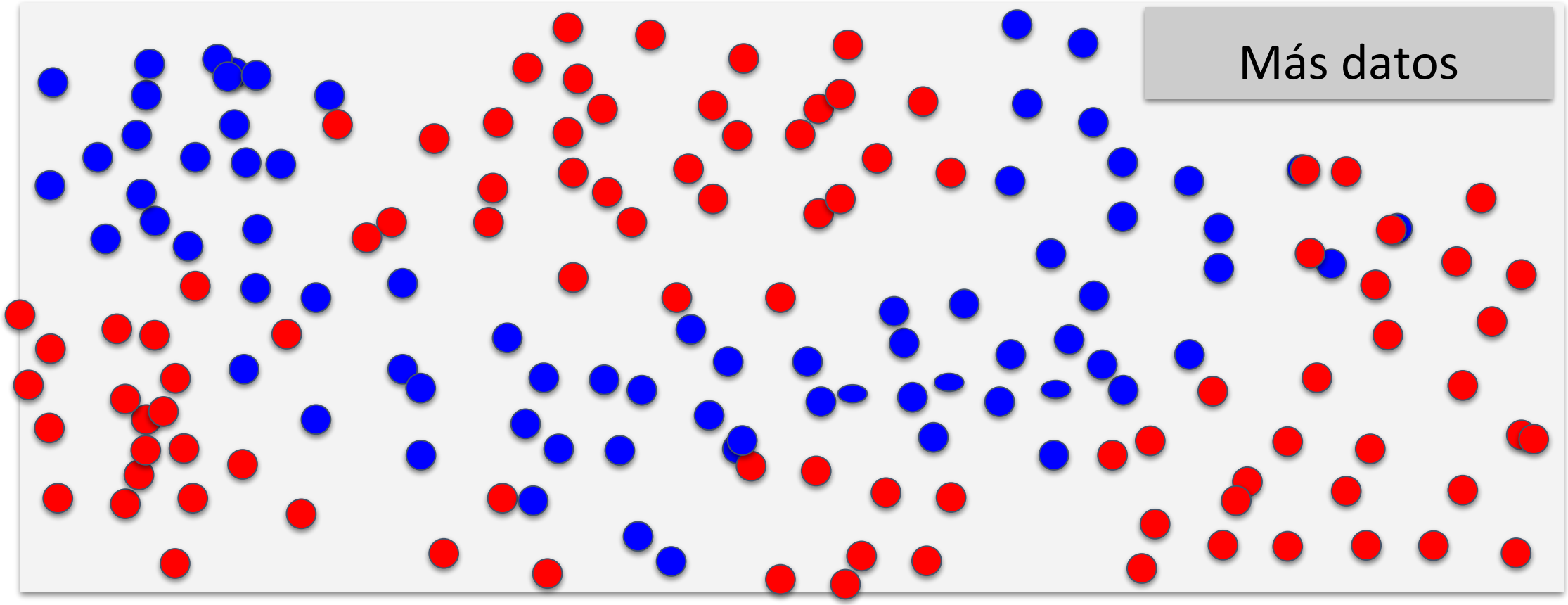
Error vs #ejemplos - Motivación

Los datos son el mejor regularizador del modelo

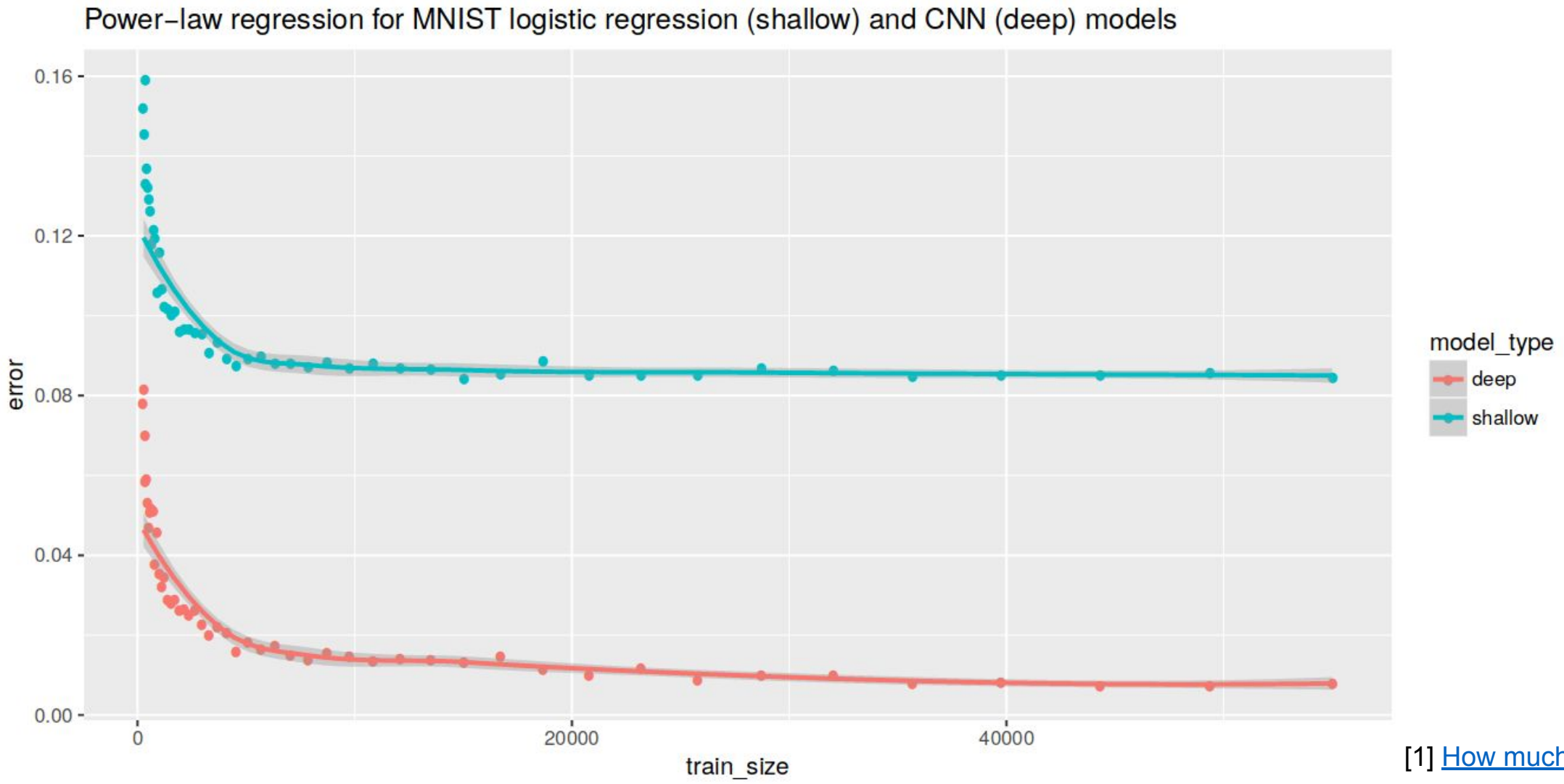


Error vs #ejemplos - Motivación

Los datos son el mejor regularizador del modelo

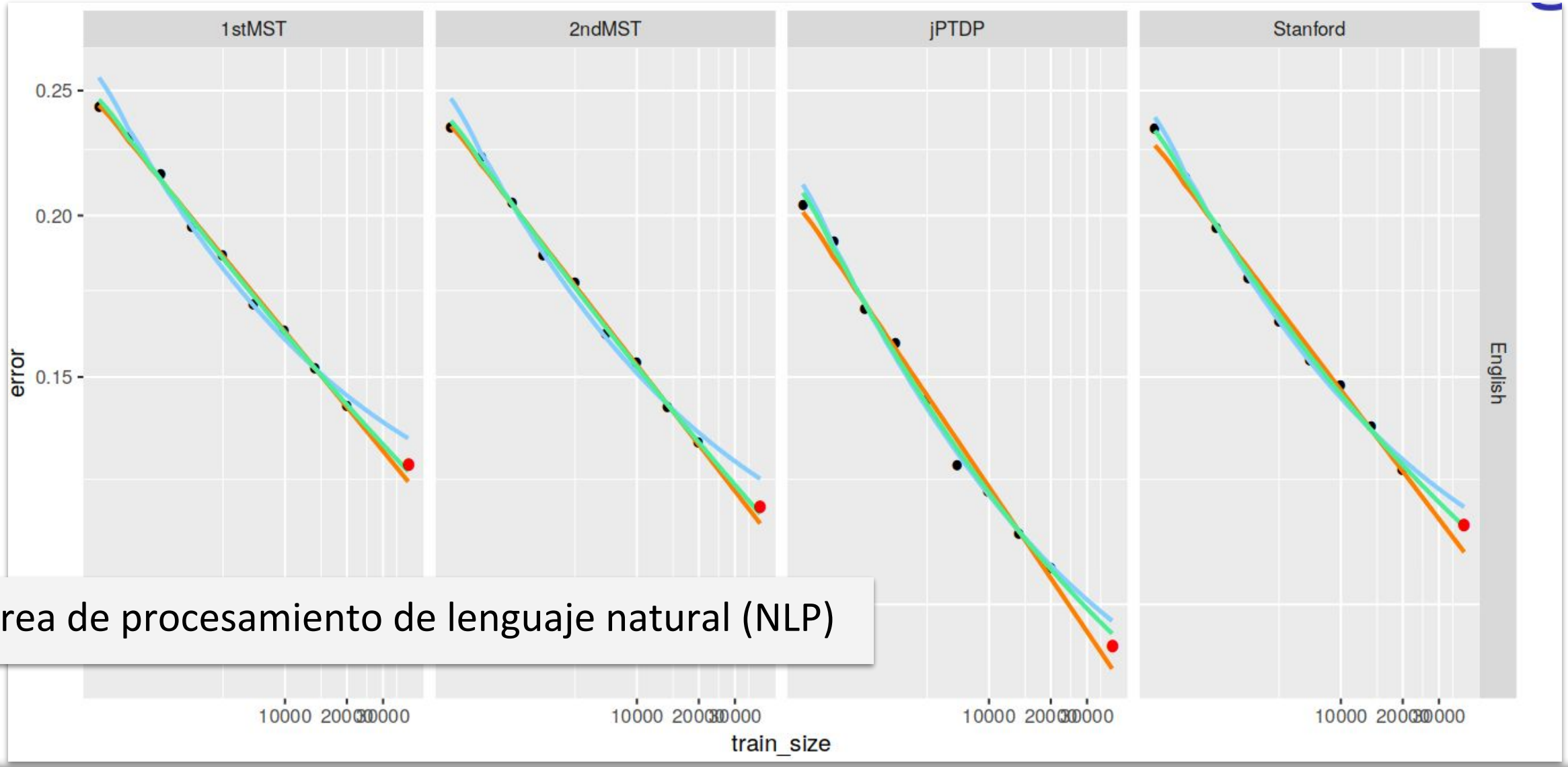


Error vs #ejemplos de train (MNIST)



[1] How much data is enough?

Error vs #ejemplos de train (NLP)

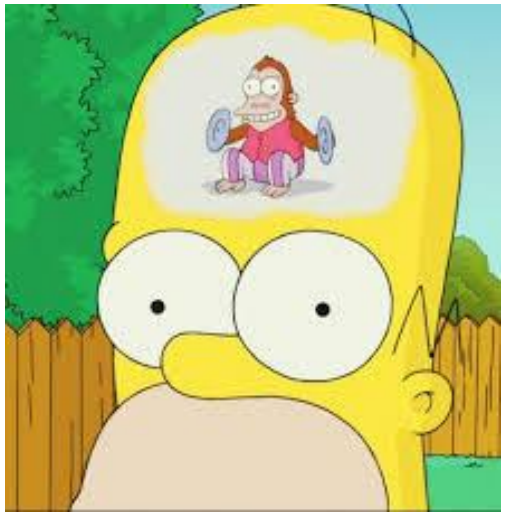


Tarea de procesamiento de lenguaje natural (NLP)

[1] How much data is enough?

¿Cómo mejorar un modelo?

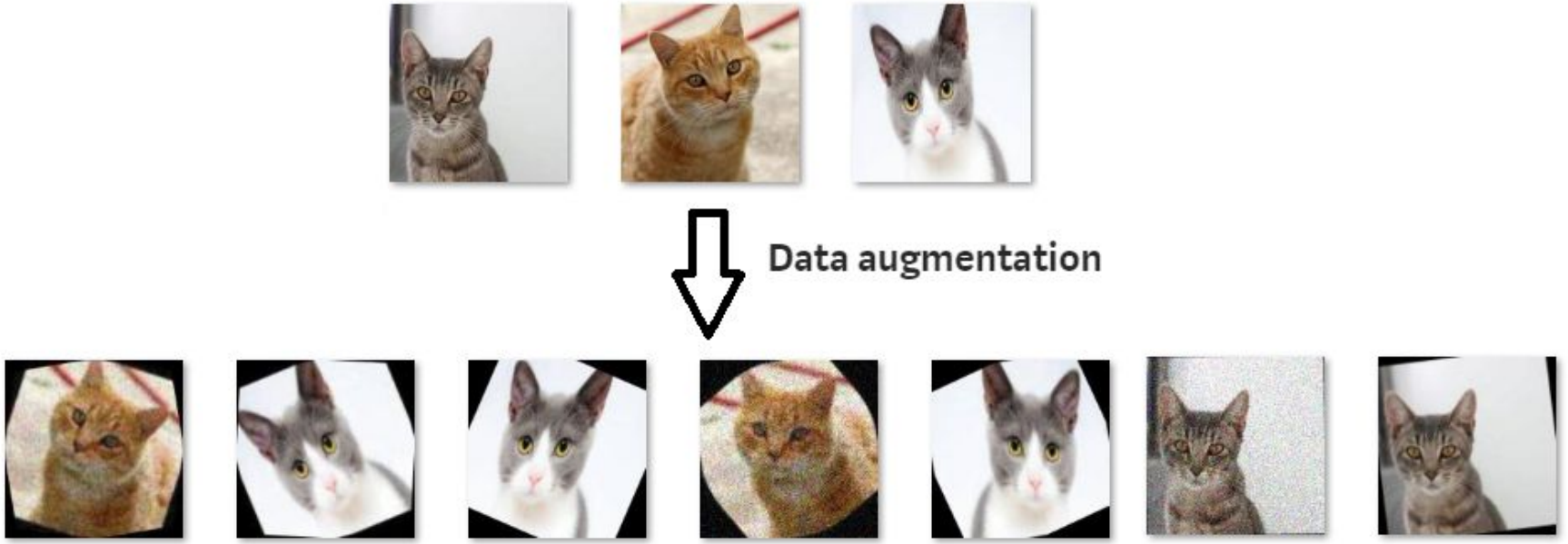
- Los datos etiquetados son “caros”
- Opción B1:
  - Hacer un crawler, bajar imágenes, etiquetarlas a mano [1]
    - Buenos resultados
    - Mucho tiempo de etiquetado y verificación
      - Usualmente, crowdsourcing
    - Generalmente solo reservado para compañias
- Opción B2:
  - **Data augmentation**
    - Multiplicar los datos existentes



Data Augmentation
(Aumentación de datos)

Aumentación de Datos

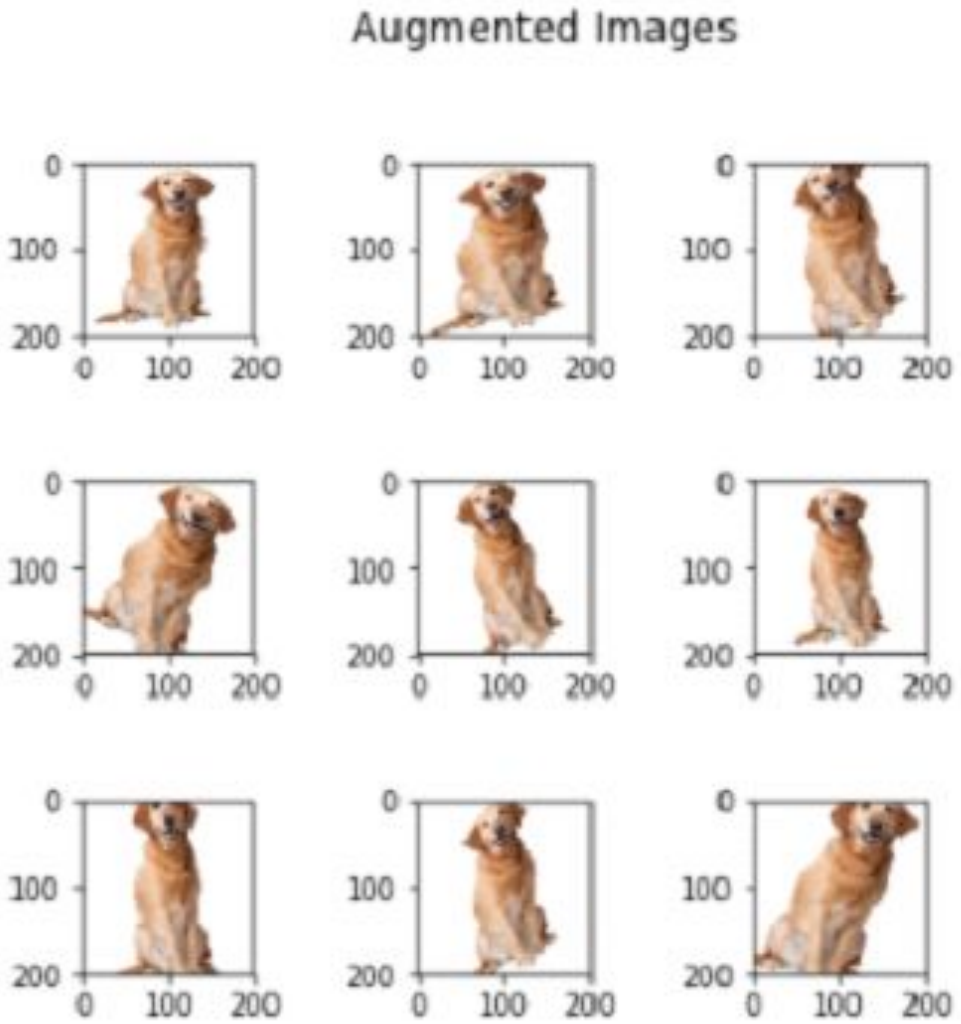
- Generar nuevos ejemplos
  - Definir transformaciones útiles
  - Transformar los ejemplos disponibles



 fuente

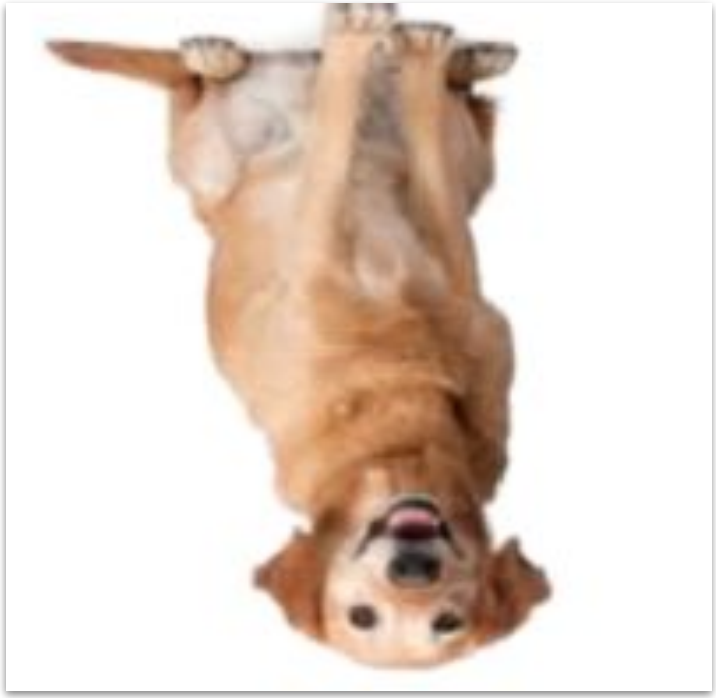
Aumentación de Datos

- Transformaciones más comunes
  - Rotaciones entre -30° y 30°



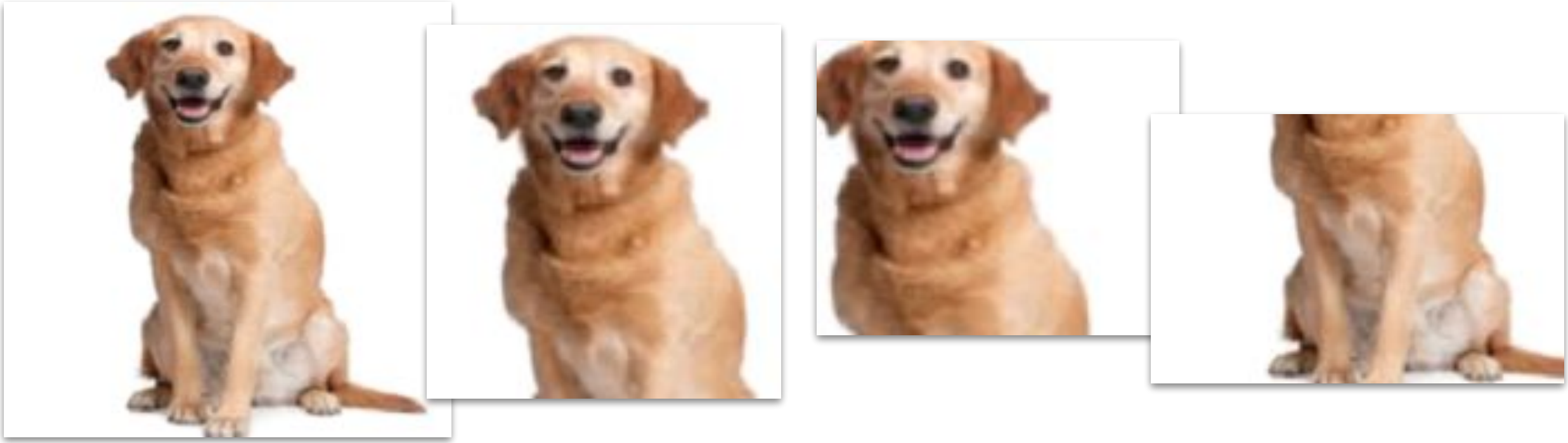
Aumentación de Datos

- Transformaciones más comunes
  - “Flips” horizontales
  - ¿Y verticales?



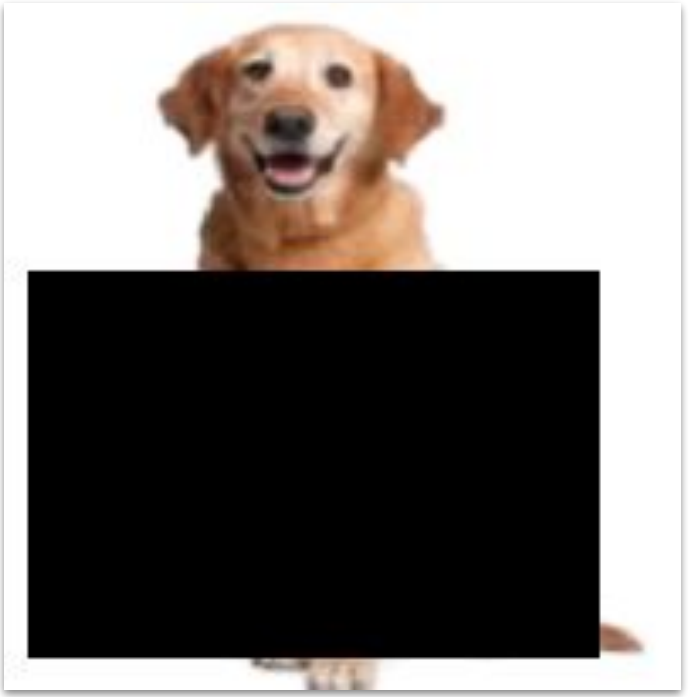
Data Augmentation

- Transformaciones más comunes
  - Crops (recortes)



Aumentación de Datos

- Transformaciones más comunes
  - Oclusiones parciales

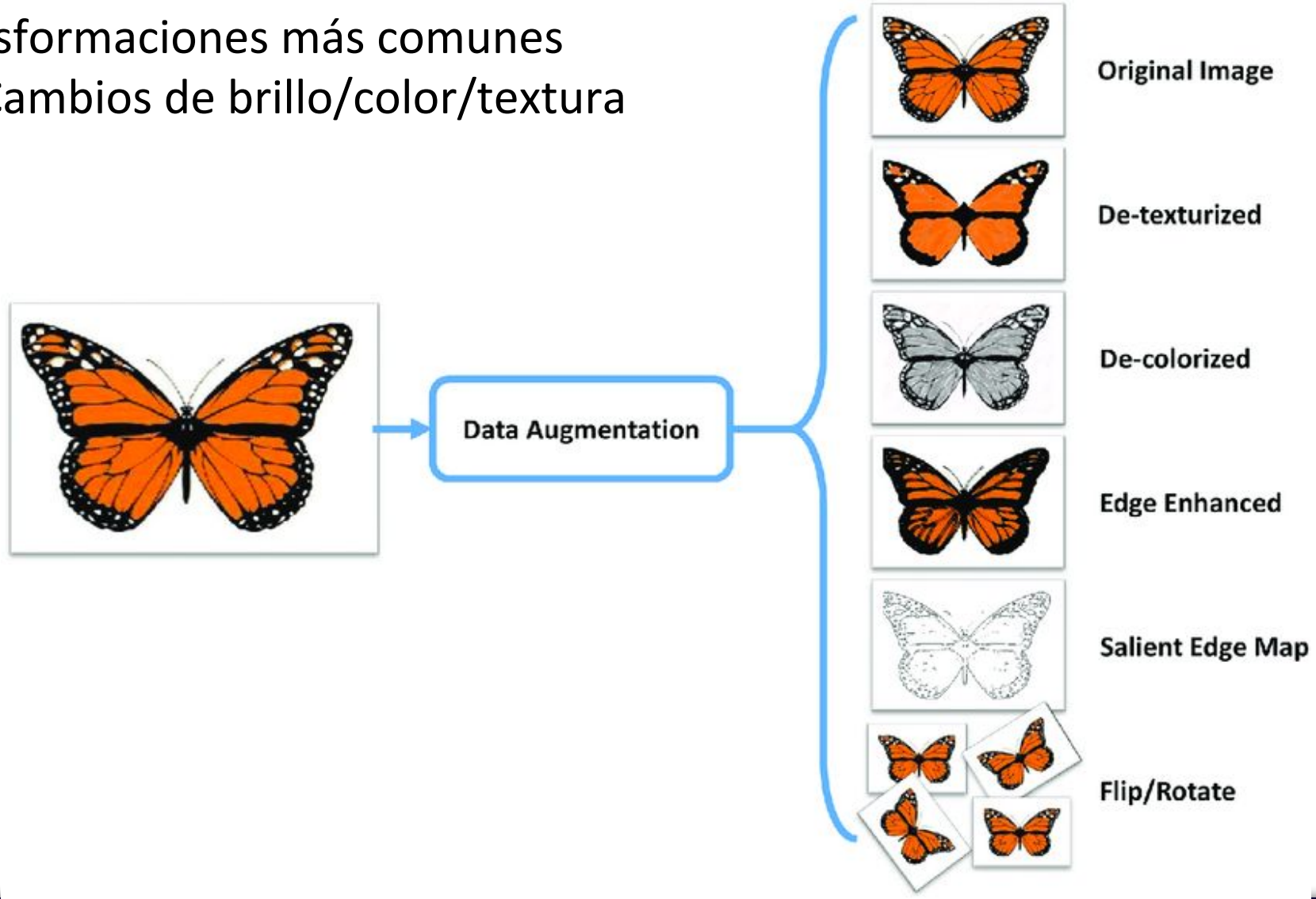


Data Augmentation

- Transformaciones más comunes
  - Cambios del fondo

Aumentación de Datos

- Transformaciones más comunes
  - Cambios de brillo/color/textura



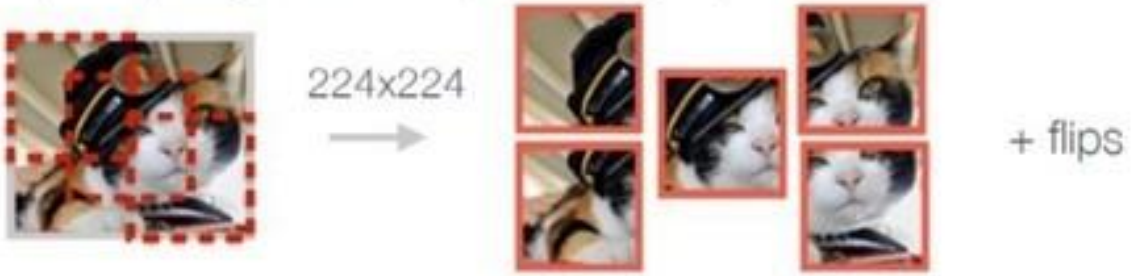
Aumentación de Datos

Data Augmentation:

a. No augmentation (= 1 image)

b. Flip augmentation (= 2 images)

c. Crop+Flip augmentation (= 10 images)



73

Aumentación de Datos + Keras + Imágenes.

- **ImageDataGenerator** con rotaciones de -20°/+20°, flip horizontales aleatorios, y escalado del valor de los pixeles.

```
x,y=load_samples()
datagen = ImageDataGenerator(rotation_range=20,
                             rescale=1./255,
                             horizontal_flip=True,
                            )
# generador que transforma a medida que genera batches
generator = datagen.flow(x,y,batch_size=32)
...
model.fit_generator(generator,
                    steps_per_epoch=len(generator)/32, ...)
```

Aumentación de Datos + Keras + Imágenes.

- **ImageDataGenerator:** transformaciones *entrenables*: normalización

```
x,y=load_samples()
# Centrar y dividir por media y desviación estándar
datagen = ImageDataGenerator(featurewise_center=True,
                             featurewise_std_normalization=True,
                                    )
datagen.fit(x) # calcula la media y desviación
generator = datagen.flow(x,y,batch_size=32)
...
model.fit_generator(generator,
                              steps_per_epoch=len(generator)/32, ...)
```

Aumentación de Datos + Keras + Imágenes.

- **ImageDataGenerator:** transformaciones *entrenables* y conjuntos *train* y *test*

```
x_train,y_train,x_test,y_test=load_samples()
# Centrar y dividir por media y desviación estándar
datagen = ImageDataGenerator(featurewise_center=True,
                             featurewise_std_normalization=True,
                                    )
datagen.fit(x_train) # calcula media y desviación con x_train
train_generator = datagen.flow(x_train,y_train,batch_size=32)
model.fit_generator(train_generator,
                              steps_per_epoch=len(generator)/32, ...)
test_generator = dataget.flow(x_test,y_test,batch_size=32)
model.evaluate_generator(test_generator)
```

Aumentación de Datos + Keras + Imágenes.

- **ImageDataGenerator** parámetros (más detalles):

```
keras.preprocessing.image.ImageDataGenerator(
# normalización
featurewise_center=False, featurewise_std_normalization=False,
samplewise_center=False, samplewise_std_normalization=False,
zca_whitening=False, zca_epsilon=1e-06,
# transformaciones geométricas
rotation_range=0, width_shift_range=0.0,
height_shift_range=0.0, shear_range=0.0, zoom_range=0.0,
horizontal_flip=False, vertical_flip=False, rescale=None,
fill_mode='nearest', cval=0.0, # 2 opciones de las tr. geom.
# Colores
channel_shift_range=0.0,brightness_range=None,
preprocessing_function=None) # transformación custom
```

Aumentación de Datos - NLP

- Data Augmentation
- Natural con imágenes
- No tanto con otros dominios
- Para Procesamiento de Lenguaje Natural (NLP):
  - Reemplazar por sinónimos
  - Insertar palabras
  - Sustituir palabras/frases
  - Borrar palabras

Operation	Sentence
None	A sad, superior human comedy played out on the back roads of life.
SR	A ***lamentable***, superior human comedy played out on the ***backward*** road of life.
RI	A sad, superior human comedy played out on ***funniness*** the back roads of life.
RS	A sad, superior human comedy played out on ***roads*** back ***the*** of life.
RD	A sad, superior human out on the roads of life.

EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks

Optimización de hiperparámetros

Parámetros vs Hiperparámetros

- **Hiperparámetros**: están "afuera" del modelo
  - Cantidad de capas
  - Tipo de capas
  - Optimizador
    - Tasa de aprendizaje
    - Regularización
    - Función de error
  - Cantidad de épocas
  - Data augmentation
  - Inicialización de parámetros
- Se optimizan "a mano"

- **Parámetros**: están "dentro" del modelo
  - Pesos de filtros convolucionales
  - Sesgos (bias)
  - Pesos de capas lineales
- Se optimizan mediante descenso de gradiente

Búsqueda de Hiperparámetros

- **Hiperparámetros**: están "afuera" del modelo
  - Cantidad de capas
  - Tipo de capas
  - Optimizador
    - Tasa de aprendizaje
    - Regularización
    - Función de error
  - Cantidad de épocas
  - Data augmentation
  - Inicialización de parámetros
- ~~Se optimizan "a mano"~~

- "Optimización" o "búsqueda" de hiperparámetros
  - **Fuerza bruta:** probar todas las posibilidades
  - **Búsqueda en grilla:** probar K soluciones distribuidas uniformemente
  - **Búsqueda aleatoria:** Probar K soluciones aleatorias

Búsqueda de Hiperparámetros

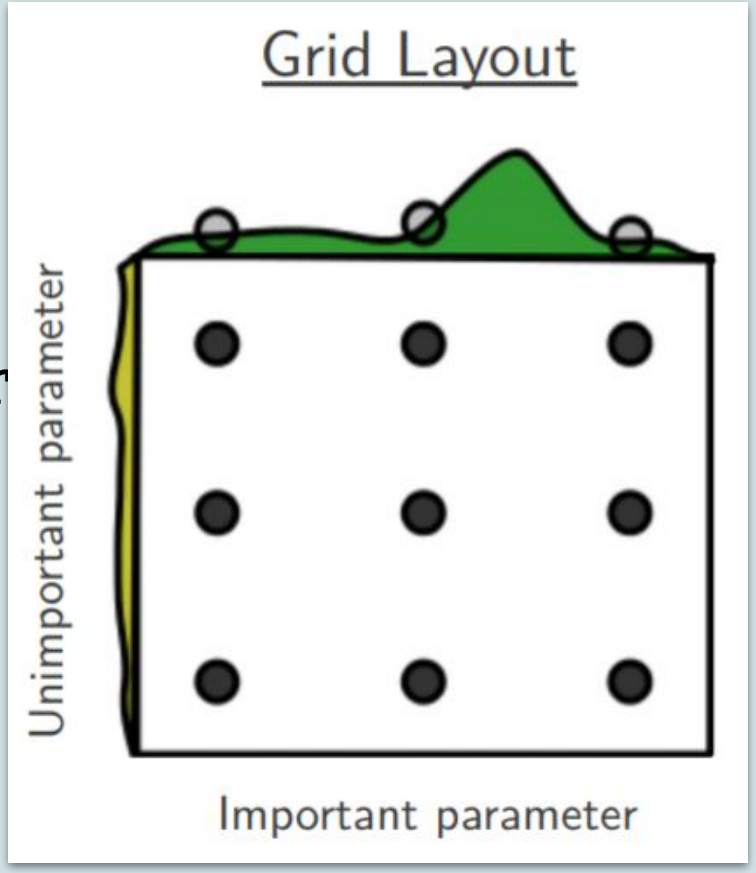
- **Fuerza bruta:** probar todas las posibilidades
  - Explosión combinatoria
  - No es práctico

```
mejor_modelo=None
for epocas in range(500):
   for capas in range(500):
      for tipo_capa in [“conv3x3”,”conv5x5”,...”lineal”]:
         … (más for-loops con más hiperparámetros)
         modelo=entrenar_modelo(epocas,capas,tipo_capa)
         if mejor(modelo, mejor_modelo):
            mejor_modelo = modelo
```

Búsqueda de Hiperparámetros

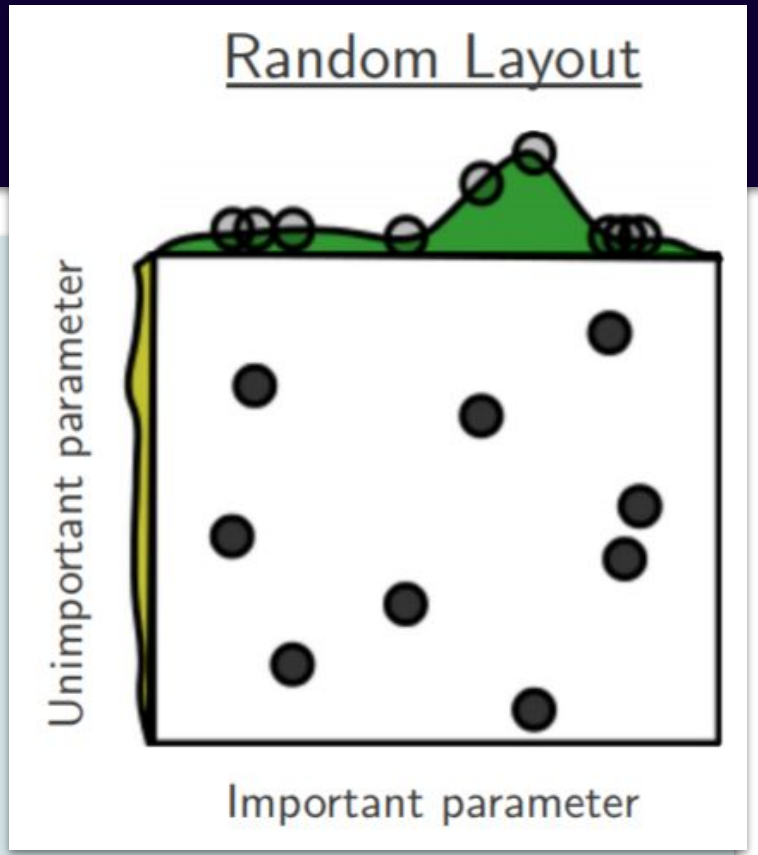
- **Búsqueda en grilla:** como fuerza bruta pero con un presupuesto finito de K combinaciones

```
mejor_modelo=None
for epocas in range(500,step=100):
   for capas in range(500,step=100):
      … (más for-loops con más hiperparámetr
         modelo,score=entrenar_modelo(epocas
         print(modelo,score)
         if mejor(modelo, mejor_modelo):
            mejor_modelo = modelo
```



fuente img

Búsqueda de Hiperparámetros



- **Búsqueda aleatoria:** elegir K combinaciones aleatorias de hiperparámetros
- Casi siempre mejor que Grilla o Fuerza Bruta

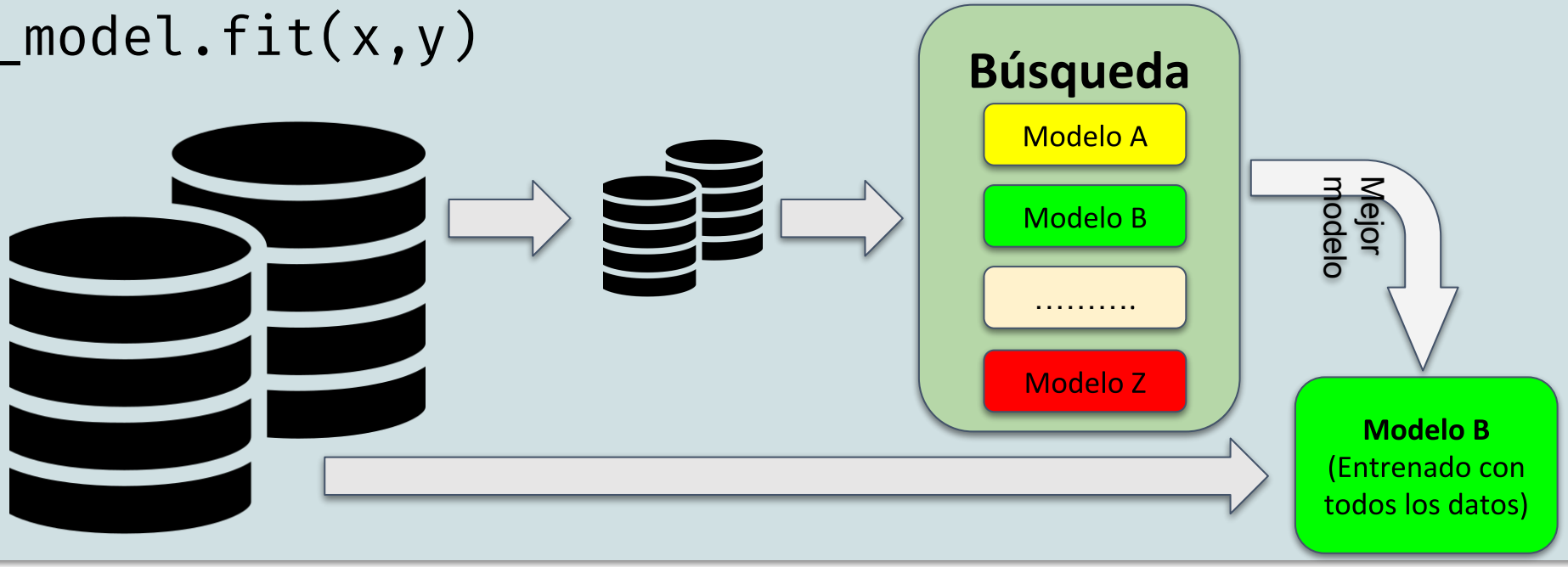
```
mejor_modelo=None
for i in range(K):
   epocas, capas, ... = hiperparametros_aleatorios()
   modelo, score = entrenar_modelo(epocas, capas, ...)
   print(modelo, score)
   if mejor(modelo, mejor_modelo):
      mejor_modelo = modelo
```


fuente img

Búsqueda de Hiperparámetros

- **Utilizar menos datos para determinar hiperparámetros**

```
x, y = load_samples()
x_subset,y_subset = make_subset(x,y,0.1) #10%
best_model = hiperparameters_search(x_subset,y_subset)
best_model.fit(x,y)
```



fuente img

Diseño de arquitecturas

¿Cómo diseñar una arquitectura?

- No es un problema resuelto
- 2 enfoques complementarios
  - Generar muchas arquitecturas y compararlas
  - Entender los errores, diseñar soluciones
- 4 casos de estudio (y técnicas que introdujeron)
  - **VGG**: Convoluciones 3x3, esquema de Bloques
  - **AllConvolutional**: GlobalAveragePooling, sin MaxPooling/Densas
  - **MobileNets**: Bloques con Convoluciones Separables Depthwise
  - **Inception**: Bloques Inception
- **No** hay **una** solución **correcta**
  - Varias arquitecturas *funcionan*
    - Unas mejores que otras

VGG

(Modelo)

(Visual Geometry Group, Oxford)

VGG (notebook, paper)

- Ganador de ILSVRC 2012 (competición ImageNet)
- Ideas principales
  - **Muchas convoluciones 3x3**
    - 2 capas Conv(3x3) ~= 1 capa Conv(7x7)
  - **Diseño en bloques**
    - 5 bloques
    - Bloque: varias Conv2D seguido de MaxPooling
- 6 versiones
  - **VGG D** (16 capas) más popular
    - También llamada **VGG16**
-

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG16 (notebook)

- Generar un block
  - Lista de capas
  - Nombre para identificar más fácil

```
def block(feature_maps,n_conv,name):
   layers=[]
   for i in range(n_conv):
      layers.append(Conv2D(feature_maps, (3, 3),
                           activation='relu',
                           padding='same',
                           name=f'{name}_conv{i}'))
   layers.append(MaxPooling2D((2, 2), strides=(2, 2),
name=f'{name}_pool'))
   return layers
```

VGG16 (notebook)

- 5 bloques
  - incrementa feature maps, achica tamaño espacial

```
fc_layers = [Flatten(),
             Dense(4096, activation='relu'),
             Dense(4096, activation='relu'),
             Dense(classes, activation='softmax')]
all_layers =  [InputLayer(input_shape)] +
              block(64 ,2,"block1") +
              block(128,2,"block2") +
              block(256,3,"block3") +
              block(512,3,"block4") +
              block(512,3,"block5") +
               fc_layers
model = keras.Sequential(all_layers)
```

VGG16 (notebook)

- Red muy grande
- 33M parámetros
  - Tarda en entrenar
- Se utiliza mucho como parte de otros modelos
- Diseño en bloques: nueva forma de pensar las redes
- Disponible en Keras

  keras.applications.vgg16.VGG16()

```
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv0 (Conv2D)        (None, 32, 32, 64)        1792
block1_conv1 (Conv2D)        (None, 32, 32, 64)        36928
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
block2_conv0 (Conv2D)        (None, 16, 16, 128)       73856
block2_conv1 (Conv2D)        (None, 16, 16, 128)       147584
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
block3_conv0 (Conv2D)        (None, 8, 8, 256)         295168
block3_conv1 (Conv2D)        (None, 8, 8, 256)         590080
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
block4_conv0 (Conv2D)        (None, 4, 4, 512)         1180160
block4_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
block5_conv0 (Conv2D)        (None, 2, 2, 512)         2359808
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
flatten_1 (Flatten)          (None, 512)               0
dense_1 (Dense)              (None, 4096)              2101248
dense_2 (Dense)              (None, 4096)              16781312
dense_3 (Dense)              (None, 10)                40970
Total params: 33,638,218
```

VGG16 (notebook)

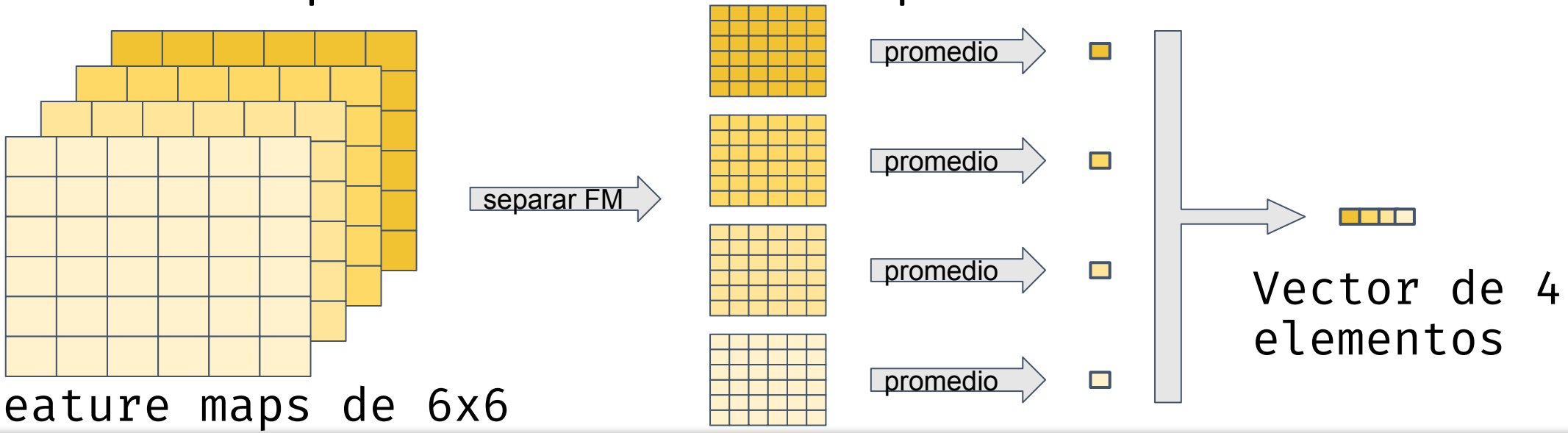
- Relación #feature maps / tamaño feature maps
  - 32x32x64 => 16x16x128 => 8x8x256 => 4x4x512 => 2x2x512
- En Imagenet, similar:
  - Resolución 224x224
  - 224x224x64 => 112x112x128 => 56x56x256 => 23x23x512 => 11x11x512

```
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv0 (Conv2D)        (None, 32, 32, 64)        1792
block1_conv1 (Conv2D)        (None, 32, 32, 64)        36928
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
block2_conv0 (Conv2D)        (None, 16, 16, 128)       73856
block2_conv1 (Conv2D)        (None, 16, 16, 128)       147584
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
block3_conv0 (Conv2D)        (None, 8, 8, 256)         295168
block3_conv1 (Conv2D)        (None, 8, 8, 256)         590080
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
block4_conv0 (Conv2D)        (None, 4, 4, 512)         1180160
block4_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
block5_conv0 (Conv2D)        (None, 2, 2, 512)         2359808
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
flatten_1 (Flatten)          (None, 512)               0
dense_1 (Dense)              (None, 4096)              2101248
dense_2 (Dense)              (None, 4096)              16781312
dense_3 (Dense)              (None, 10)                40970
Total params: 33,638,218
```

GlobalAveragePooling (Capa)

Global Average Pooling

- GlobalAveragePooling
  - Motivación
    - Los feature maps se achican espacialmente
    - Pero crecen en cantidad (#canales)
      - Cada canal detecta una característica
      - La dimensión espacial del feature map deja de importar
  - Método: promediar dimensiones espaciales



Global Average Pooling

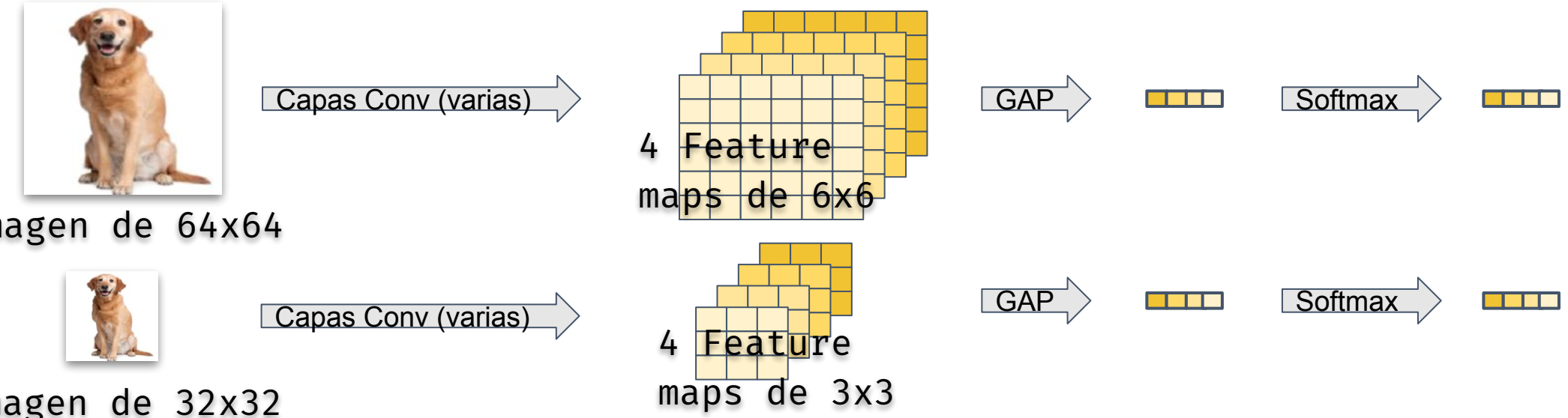
- GlobalAveragePooling
  - Permite calcular puntajes para cada clase
  - Reemplaza las Dense/Flatten
- Promedia las dimensiones espaciales
  - Deja solo la dimensión de canales de un feature map
- Primero se suele utilizar una conv 1x1
  - Tantos feature maps como clases
- Ejemplo si tengo 10 clases:
  - Convolución (1x1) con 10 feature maps de HxW
  - HxWx10 = 8x8x64, entonces la salida de GAP es de 10

```
model.add(Conv2D(classes,(1,1),activation="relu"))
 model.add(GlobalAveragePooling2D())
 model.add(Activation('softmax'))
```

```
def GlobalAveragePooling2D(x):
    h,w,c=x.shape
    y=np.zeros(c)
    for i in range(c):
        y[i]=x[:,:,i].mean
    return y
```

Global Average Pooling

- GlobalAveragePooling
  - Motivación
    - El entrenar de la red, fija la resolución HxW de la imagen de entrada
    - Sucesivas capas con stride>1 achican HxW a H'xW'
    - La capa GlobalAveragePooling borra esa H'xW'
      - No importa la resolución original HxW
    - => **Redes independientes de la resolución de entrenamiento**

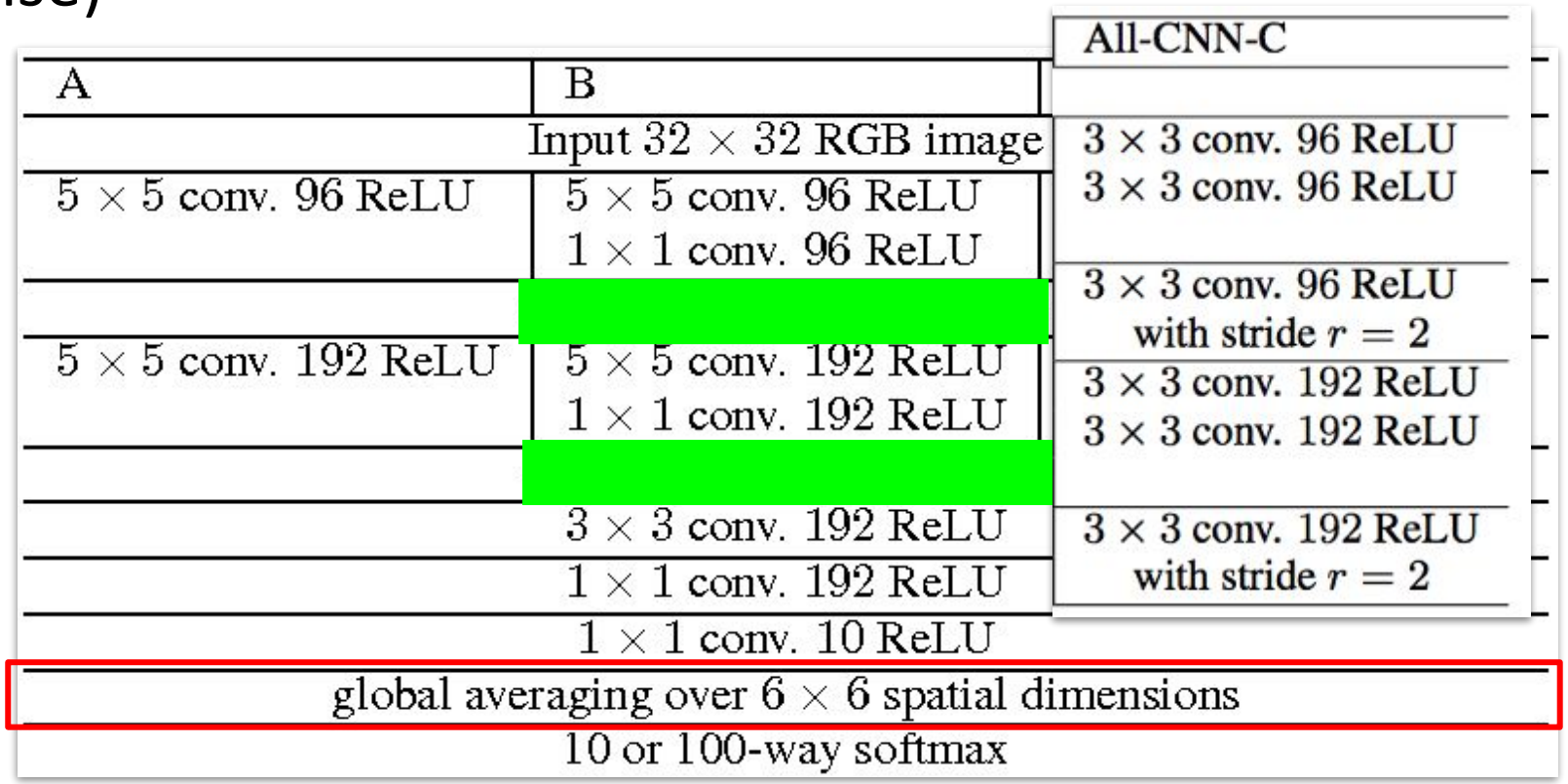


AllConvolutional (Modelo)

AllConvolutional (notebook, paper)

- Idea principal
  - Solo convoluciones
  - Sin MaxPooling => convoluciones con stride=2
  - Sin capas Lineales (Dense)
- Probó que era posible

  usar sólo Convoluciones
- 3 versiones
  - La C es la más popular
- Capa especial
  - GlobalAveragePooling
  - Reemplaza las Dense



A	B	All-CNN-C
	Input 32 × 32 RGB image	
5 × 5 conv. 96 ReLU	5 × 5 conv. 96 ReLU 1 × 1 conv. 96 ReLU	3 × 3 conv. 96 ReLU 3 × 3 conv. 96 ReLU
		3 × 3 conv. 96 ReLU with stride $r = 2$
5 × 5 conv. 192 ReLU	5 × 5 conv. 192 ReLU 1 × 1 conv. 192 ReLU	3 × 3 conv. 192 ReLU 3 × 3 conv. 192 ReLU
	3 × 3 conv. 192 ReLU	3 × 3 conv. 192 ReLU with stride $r = 2$
	1 × 1 conv. 192 ReLU	
	1 × 1 conv. 10 ReLU	
global averaging over 6 × 6 spatial dimensions		
10 or 100-way softmax		

AllConvolutional (notebook, paper)

```
def AllConvolutional(classes,input_shape):
 model = keras.Sequential()
 model.add(InputLayer(input_shape))
 for feature_maps in [96,192]:
    model.add(Conv2D(feature_maps,(3,3),activation="relu",padding="same",))
    model.add(Conv2D(feature_maps,(3,3),activation="relu",padding="same",))
    model.add(Conv2D(feature_maps,(3,3),activation="relu",padding="same",strides=(2,2)))

 model.add(Conv2D(192,(3,3),activation="relu",padding="same",))
 model.add(Conv2D(192,(1,1),activation="relu",padding="same",))

 model.add(Conv2D(classes,(1,1),activation="relu",padding="same",))
 model.add(GlobalAveragePooling2D())
 model.add(Activation('softmax'))
 return model
```

AllConvolutional (notebook, paper)

- Más pequeña que VGG
  - 1.3M parámetros
- Diseñada para CIFAR10 o CIFAR100
  - Más fácil que ImageNet
- Fácil de entrenar
- Buena performance
- Reducción de dimensionalidad espacial (32-> 16, 16->8)

```
input_shape=(32,32,3)
classes=10
model = AllConvolutional(classes,input_shape)
print(model.summary())
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_19 (Conv2D)           (None, 32, 32, 96)        2688
conv2d_20 (Conv2D)           (None, 32, 32, 96)        83040
conv2d_21 (Conv2D)           (None, 16, 16, 96)        83040
conv2d_22 (Conv2D)           (None, 16, 16, 192)       166080
conv2d_23 (Conv2D)           (None, 16, 16, 192)       331968
conv2d_24 (Conv2D)           (None, 8, 8, 192)         331968
conv2d_25 (Conv2D)           (None, 8, 8, 192)         331968
conv2d_26 (Conv2D)           (None, 8, 8, 192)         37056
conv2d_27 (Conv2D)           (None, 8, 8, 10)          1930
global_average_pooling2d_1 ( (None, 10)                0
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 1,369,738
```

AllConvolutional (notebook, paper)

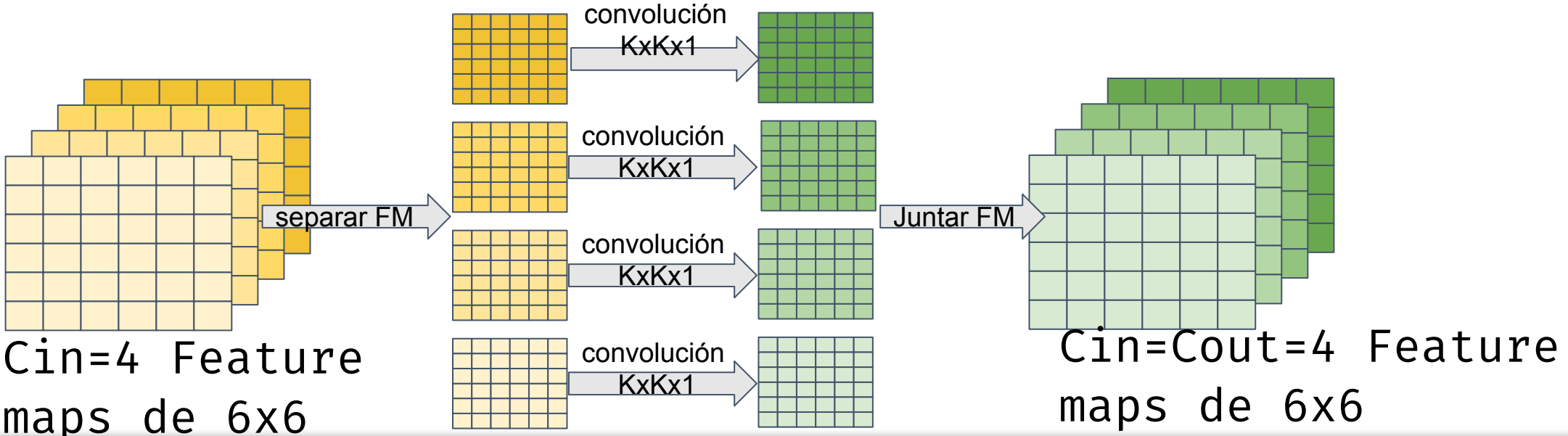
- Resumen
  - Solo Convoluciones :O!
  - Orientada a Cifar10/100
  - Global Average Pooling
    - Independiza de resolución
    - Reemplaza capas Dense

A	B	All-CNN-C
	Input 32 × 32 RGB image	
5 × 5 conv. 96 ReLU	5 × 5 conv. 96 ReLU 1 × 1 conv. 96 ReLU	3 × 3 conv. 96 ReLU 3 × 3 conv. 96 ReLU
	3 × 3 max-pooling stride 2	3 × 3 conv. 96 ReLU with stride $r = 2$
5 × 5 conv. 192 ReLU	5 × 5 conv. 192 ReLU 1 × 1 conv. 192 ReLU	3 × 3 conv. 192 ReLU 3 × 3 conv. 192 ReLU
	3 × 3 max-pooling stride 2	3 × 3 conv. 192 ReLU with stride $r = 2$
	3 × 3 conv. 192 ReLU	
	1 × 1 conv. 192 ReLU	
	1 × 1 conv. 10 ReLU	
	global averaging over 6 × 6 spatial dimensions	
	10 or 100-way softmax	

Convoluciones Separables Depthwise (Capa)

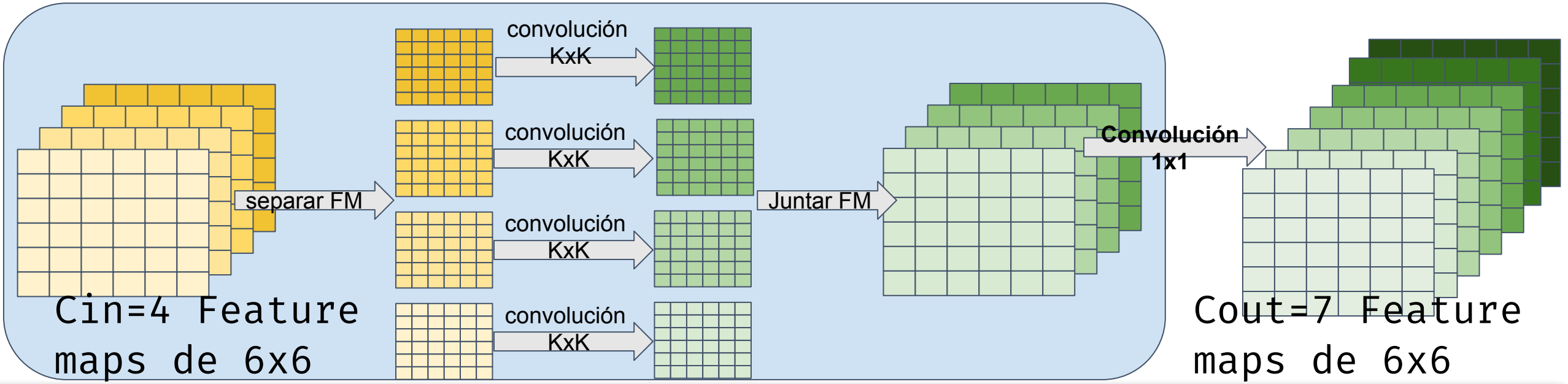
Convoluciones Separables

- Motivación
  - Convoluciones comunes: transforman **HxWxCin** en **HxWxCout**
  - **Cout** Filtros de tamaño **KxKxCin**
    - Generan **Cout** canales de salida
    - **“Mira”** los **Cin** canales de entrada
- Técnica Separable: Cada filtro mira un solo canal de entrada => Filtros **KxKx1**



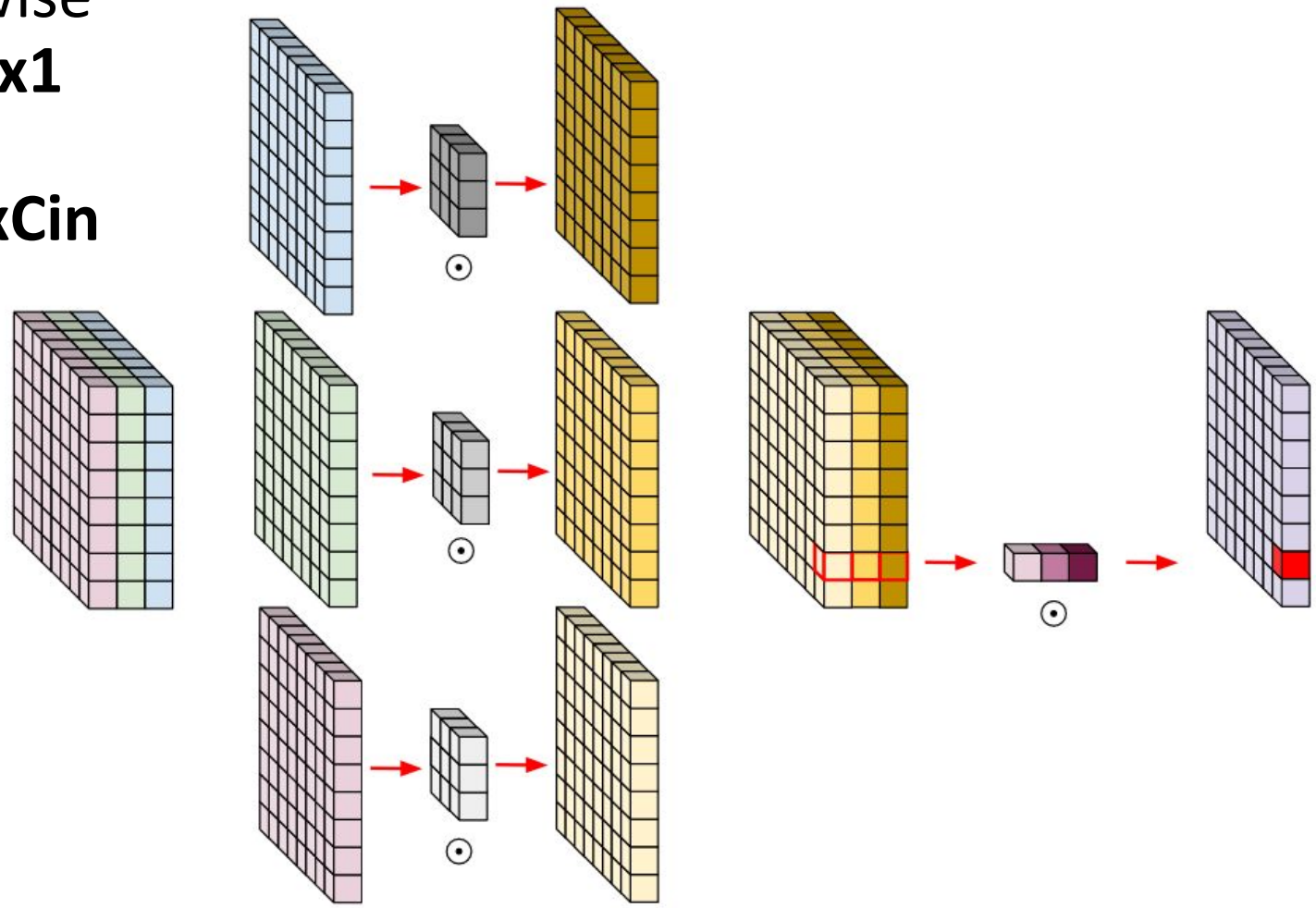
Convoluciones Separables **Depthwise**

- Convoluciones Separables **Depthwise**
  - Motivación
    - Convoluciones comunes: transforman **HxWxCin** en **HxWxCout**
    - Convoluciones separables: transforman **HxWxCin** en **HxWxCin**
  - Técnica:
    - Agregar una convolución 1x1 (**depthwise**) para obtener **HxWxCout**



Convoluciones Separables **Depthwise**

- Convoluciones separables depthwise
  - Primero aplican **Cin** filtros **KxKx1**
    - Filtros espaciales
  - Luego aplican **Cout** filtros **1x1xCin**
    - Filtros de profundidad (depthwise)
- No es equivalente a aplicar **Cout** filtros de tama **KxKxCin**
  - **Tradeoffs**
    - +++ Menos parámetros
    - +++ Menos cómputo
    - - Menos poder de representación (no mucho)



fuente

MobileNets
(Modelo)

MobileNets (notebook, paper)

- Eficiencia en mente
- Convoluciones “Separables” o “Depthwise”
- Poco uso de memoria y CPU
  - => Mobile
- Relación cantidad feature maps / tamaño similar a VGG
  - 112x112x32 => 56x56x128 => 28x28x256 => 14x14x512 => 7x7x1024
- GlobalAveragePooling para quitar dimensiones espaciales

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

MobileNets (notebook, paper)

- Arquitectura:
- 4 bloques con FMs de 64, 128, 256, 512
- Cada bloque:
  - SeparableConv2D(n,(3,3), stride=(1,1))
  - SeparableConv2D(n,(3,3), stride=(2,2))
- Pero irregulares!
- SeparableConv2D(1024,(3,3)) al final
- AvgPooling
  - Pero le agregan una capa Dense (FC)
- Diseñada para ImageNet
  - Imágenes de 224x224x3
- Disponible en Keras
  - keras.applications.mobilenet.MobileNet()

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

MobileNets (notebook, paper)

- Cada bloque tiene **n_conv** convoluciones separables depthwise con stride=1
- Luego una convolución separable con stride=2
- OJO! El bloque 1 no es así exactamente, simplificamos

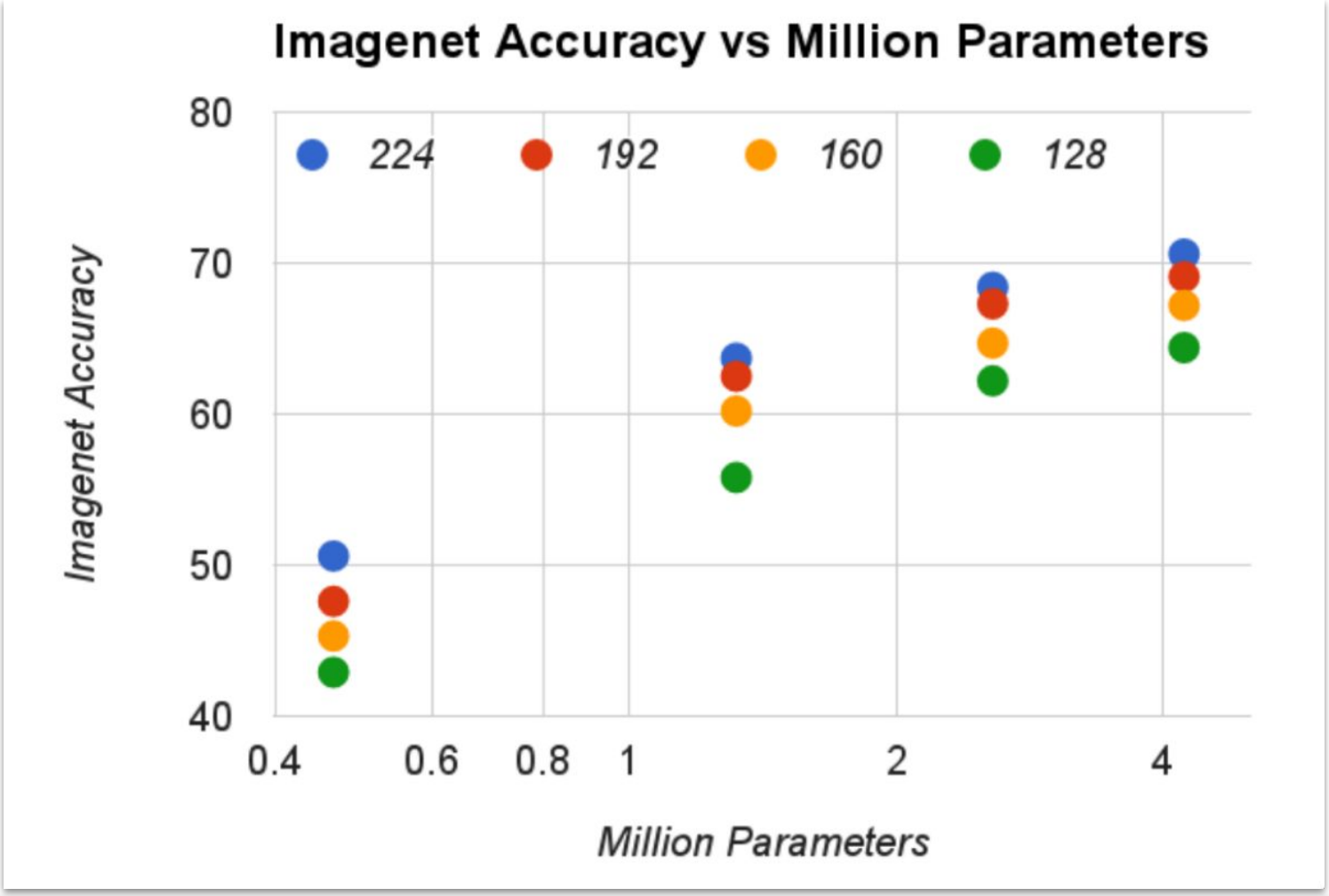
```
def block(n_filters,n_conv):
  layers=[]
  for i in range(n_conv):
    layers.append(SeparableConv2D(n_filters,(3,3),activation="relu",))
  layers.append(SeparableConv2D(n_filters*2,(3,3),
                                activation="relu",strides=(2,2) ))
  return layers
```

MobileNets (notebook, paper)

```
def MobileNet(classes,input_shape):
  model = keras.Sequential()
  model.add(InputLayer(input_shape))
  model.add(Conv2D(32,(3,3), activation="relu", padding="same"))


  for n_filters,n_conv in zip([64,128,256,512],[1,1,1,5]):
     layers=block(n_filters,n_conv)
     for layer in layers:
        model.add(layer)
  model.add(SeparableConv2D(1024,(3,3), activation="relu", padding="same"))
  model.add(GlobalAveragePooling2D())
  model.add(Dense(classes))
  model.add(Activation('softmax'))
  return model
```

MobileNets (notebook, paper)

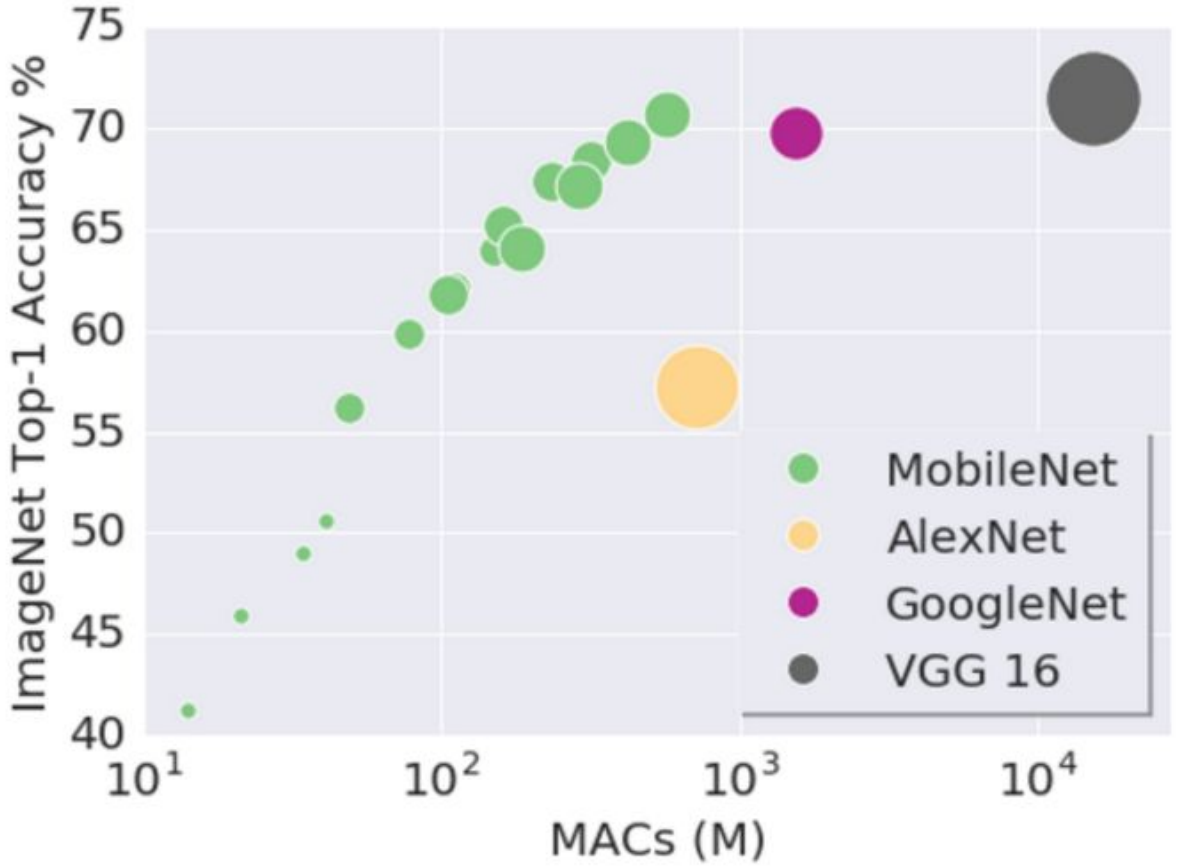
- Accuracy **VS** resolución **VS** #parámetros, en ImageNet



- keras.applications.mobilenet.MobileNet() tiene un hiperparámetro llamado **alpha**
- **alpha** = 1: red común
- **alpha** > 1: más filtros
- **alpha** < 1: menos filtros
- Ejemplo:
  - **alpha** = 0.5
    - Mitad de filtros
    - (mitad de param)

MobileNets (notebook, paper)

- Accuracy vs #Operaciones en MobileNets

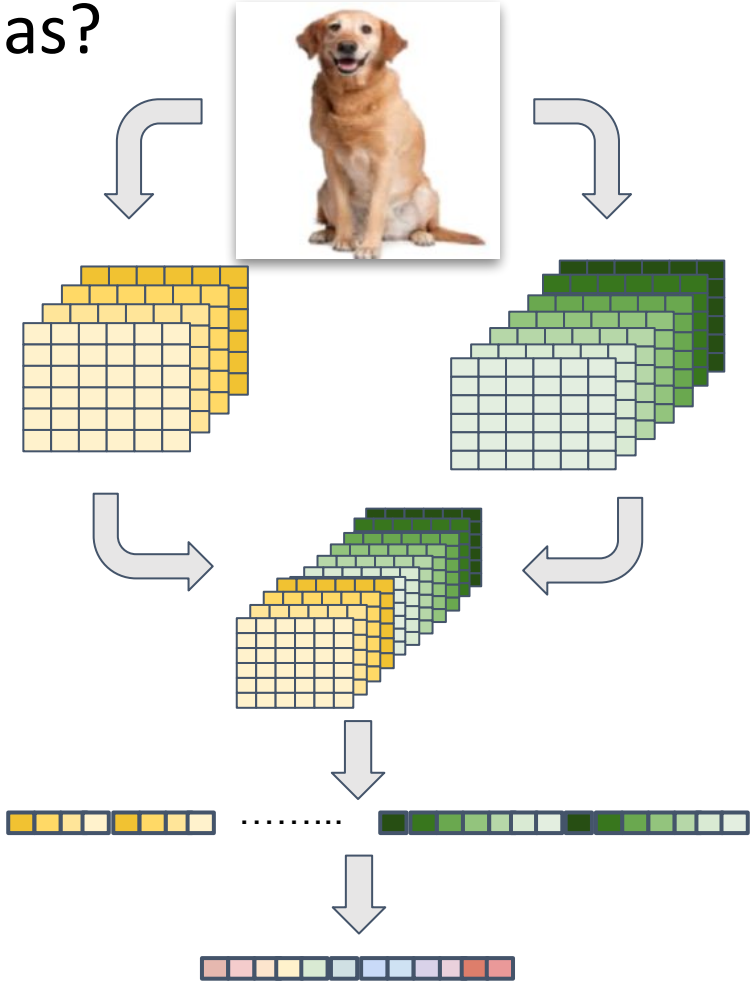


Sequential vs **Model**
en Keras

Modelos de grafos arbitrarios (notebook)

- Hasta ahora todos los modelos que vimos son **Sequential** => **lista** de capas
- ¿Si queremos hacer un modelo que es un **grafo** de capas?
- Ejemplo para CIFAR10 con clase **Model**

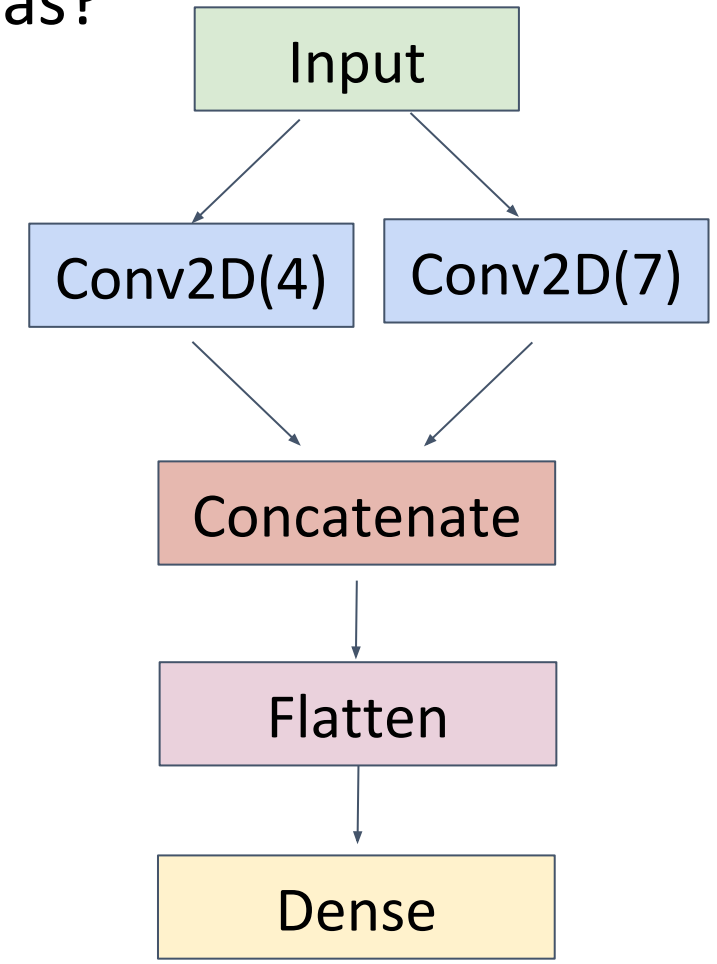
```
from keras import Model
from keras.layers import
InputLayer,Conv2D,Dense, Concatenate,Flatten

x     = Input(shape=(32,32,3))
path1 = Conv2D(4,(3,3),padding="same")(x)
path2 = Conv2D(7,(5,5),padding="same")(x)
join  = Concatenate([path1,path2])
flat  = Flatten()(join)
fc    = Dense(10,activation="softmax")(flat)
model = Model(inputs=[x],outputs=[fc])
```



Modelos de grafos arbitrarios (notebook)

- Hasta ahora todos los modelos que vimos son **Sequential** => **lista** de capas
- ¿Si queremos hacer un modelo que es un **grafo** de capas?
- Ejemplo para CIFAR10 con clase **Model**

```
from keras import Model
from keras.layers import
InputLayer,Conv2D,Dense, Concatenate,Flatten

x     = Input(shape=(32,32,3))
path1 = Conv2D(4,(3,3),padding="same")(x)
path2 = Conv2D(7,(5,5),padding="same")(x)
join  = Concatenate()([path1,path2])
flat  = Flatten()(join)
fc    = Dense(10,activation="softmax")(flat)
model = Model(inputs=[x],outputs=[fc])
```



Modelos de grafos arbitrarios (notebook)

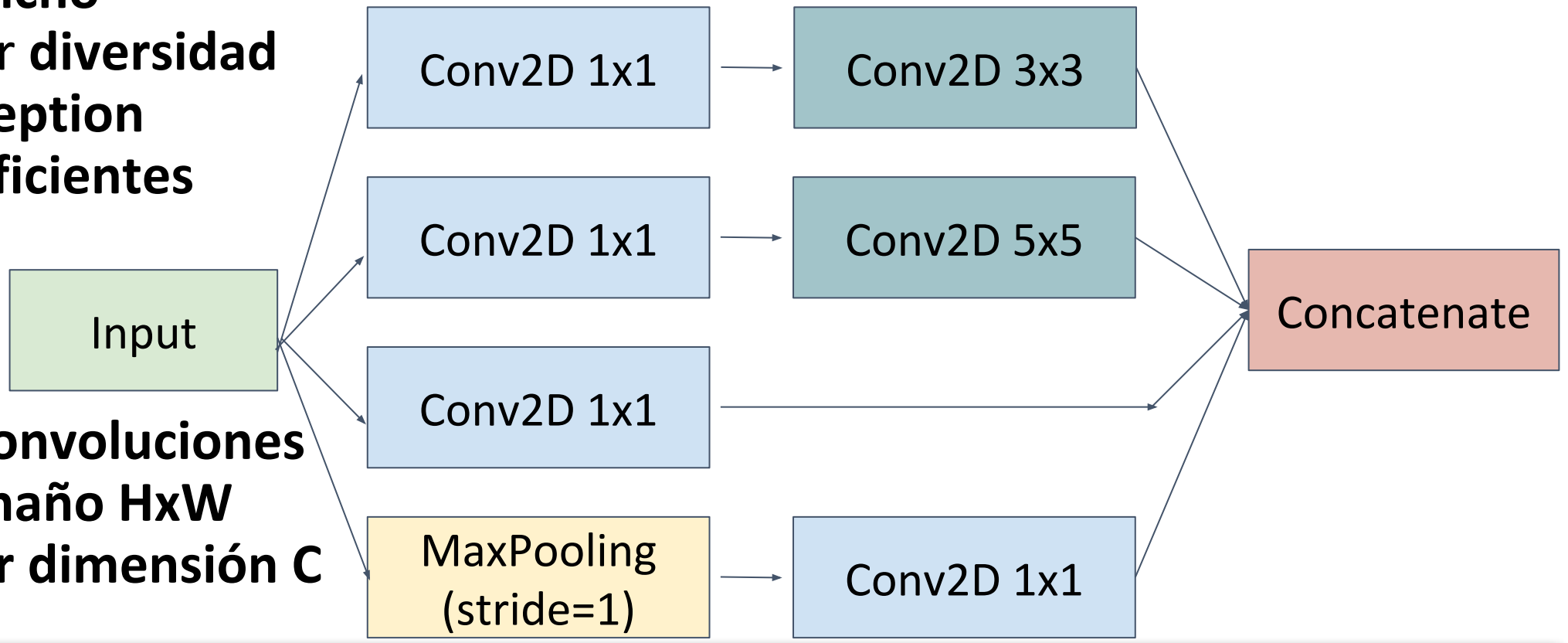
- Hasta ahora todos los modelos que vimos son **Sequential** => **lista** de capas
- ¿Si queremos hacer un modelo que es un **grafo** de capas? clase **Model**
- Ejemplo para CIFAR10

```
print(model.summary())
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 32, 32, 3)    0
conv2d_5 (Conv2D)               (None, 32, 32, 4)    112         input_3[0][0]
conv2d_6 (Conv2D)               (None, 32, 32, 7)    532         input_3[0][0]
concatenate_3 (Concatenate)     (None, 32, 32, 11)   0           conv2d_5[0][0]
                                                                 conv2d_6[0][0]
flatten_2 (Flatten)             (None, 11264)        0           concatenate_3[0][0]
dense_3 (Dense)                 (None, 10)           112650      flatten_2[0][0]
==================================================================================================
Total params: 113,294
```

Bloques Inception

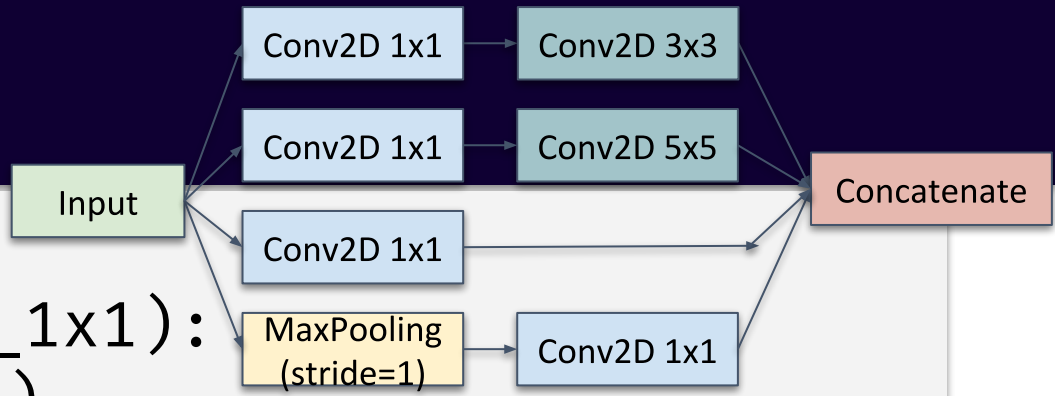
Bloques Inception

- Hasta ahora
  - ¿Mayor **complejidad**? => Mayor **profundidad**
- Otra forma
  - Mayor **ancho**
    - Mayor **diversidad**
- **Bloques Inception**
  - **Conv + eficientes**



- **Método**
  - **Distintas convoluciones**
  - **Mismo tamaño HxW**
  - **Concatenar dimensión C**

Bloques Inception

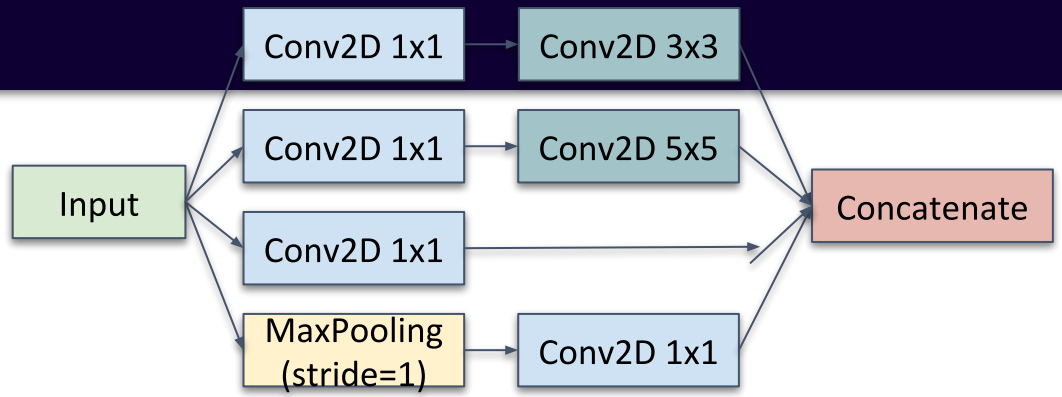



```
def inception(x,F3x3_1x1,F3x3,
                F5x5_1x1,F5x5,F1x1,Fmp_1x1):
  c3x3_1x1 = Conv2D(F3x3_1x1,(1,1))(x)
  c3x3     = Conv2D(F3x3,(3,3))(c3x3_1x1)

  c5x5_1x1 = Conv2D(F5x5_1x1,(1,1))(x)
  c5x5     = Conv2D(F5x5,(5,5))(c5x5_1x1)

  c1x1     = Conv2D(F1x1,(1,1))(x)

  mp       = MaxPooling2D((3,3),strides=(1,1))(x)
  mp_1x1   = Conv2D(Fmp_1x1,(1,1))(mp)
  result  = Concatenate(axis=3)([c1x1, c3x3, c5x5, mp_1x1])

  return result
```

- Falta:padding="same"
  - No cambiar HxW
- Falta: activation="relu”

Bloques Inception

```
x = Input(shape=(32,32,64))
layer = inception(x,5,10,20,30,40,50)
model = Model(inputs=[x],outputs=[layer])
print(model.summary())
```




```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================
input (InputLayer)           (None, 32, 32, 64)        0
block1_3x3_1x1 (Conv2D)      (None, 32, 32, 5)         325         input[0][0]
block1_3x3 (Conv2D)          (None, 32, 32, 10)        460         block1_3x3_1x1[0][0]
block1_5x5_1x1 (Conv2D)      (None, 32, 32, 20)        1300        input[0][0]
block1_5x5 (Conv2D)          (None, 32, 32, 30)        15030       block1_5x5_1x1[0][0]
block1_1x1 (Conv2D)          (None, 32, 32, 40)        2600        input[0][0]
block1_mp (MaxPooling2D)     (None, 32, 32, 64)        0           input[0][0]
block1_mp_1x1 (Conv2D)       (None, 32, 32, 50)        3250        block1_mp[0][0]
block1_concat (Concatenate)  (None, 32, 32, 160)       0           block1_1x1[0][0]
                                                                   block1_3x3[0][0]
                                                                   block1_5x5[0][0]
                                                                   block1_mp_1x1[0][0]

Total params: 22,965
```

- 160 feature maps
- 23 000 parámetros

Bloques Inception (comparación)

- Comparemos con una convolucional 3x3 o 5x5 **común**

```
x = Input(shape=(32,32,64))
layer = Conv2D(160,(3,3))(x)
model = Model(inputs=[x],outputs=[layer])
print(model.summary())
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 32, 32, 64)        0
conv (Conv2D)                (None, 28, 28, 160)       92320
160
Total params: 92320
```

- 160 feature maps
- 5x5: 256.000 param
- 3x3: 92.000 param
- **Inception: 16000 param**

```
x = Input(shape=(32,32,64))
layer = Conv2D(160,(5,5))(x)
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 32, 32, 64)        0
conv (Conv2D)                (None, 28, 28, 160)       256160
Total params: 256,160
```

Inception V1
(Modelo)

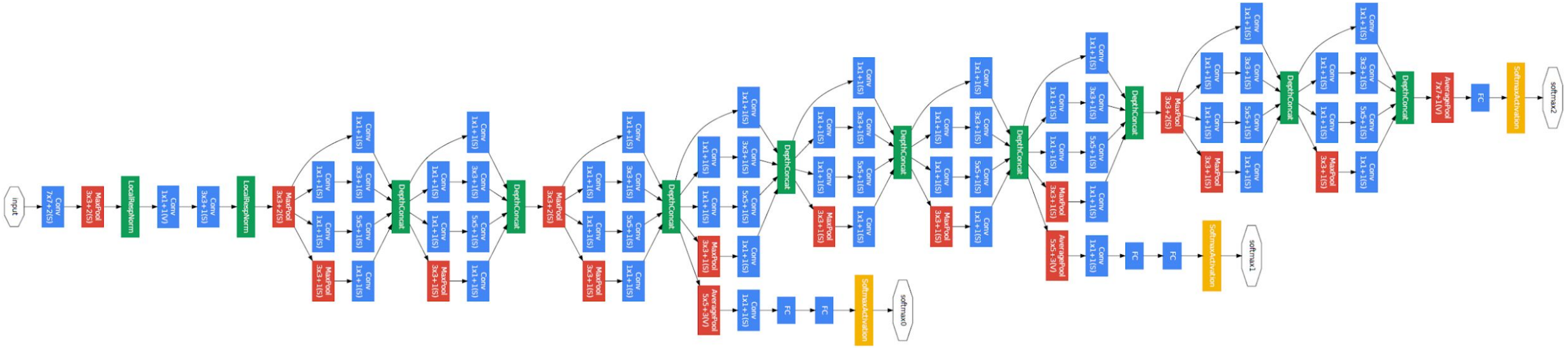
InceptionV1 (notebook, paper)

- Ganador de ILSVRC 2014 (ImageNet)
- También llamado **GoogleNet**
- Introdujo bloques **Inception**

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

InceptionV1 (notebook, paper)

- Diagrama de arquitectura
  - Ignorar las primeras dos salidas amarillas



Inception (notebook, paper)

- Implementación en Keras
- Inception v1
- Muchas partes irregulares
  - Funcionan bien pero no tienen justificación
- Veremos una versión simplificada

```
def create_googlenet(weights_path=None):
    # creates GoogLeNet a.k.a. Inception v1 (Szegedy, 2015)
    input = Input(shape=(3, 224, 224))

    input_pad = ZeroPadding2D(padding=(3, 3))(input)
    conv1_7x7_s2 = Conv2D(64, (7,7), strides=(2,2), padding='valid', activation='relu', name
    conv1_zero_pad = ZeroPadding2D(padding=(1, 1))(conv1_7x7_s2)
    pool1_helper = PoolHelper()(conv1_zero_pad)
    pool1_3x3_s2 = MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid', name='pool1
    pool1_norm1 = LRN(name='pool1/norm1')(pool1_3x3_s2)

    conv2_3x3_reduce = Conv2D(64, (1,1), padding='same', activation='relu', name='conv2/3x3_
    conv2_3x3 = Conv2D(192, (3,3), padding='same', activation='relu', name='conv2/3x3', ker
    conv2_norm2 = LRN(name='conv2/norm2')(conv2_3x3)
    conv2_zero_pad = ZeroPadding2D(padding=(1, 1))(conv2_norm2)
    pool2_helper = PoolHelper()(conv2_zero_pad)
    pool2_3x3_s2 = MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid', name='pool2

    inception_3a_1x1 = Conv2D(64, (1,1), padding='same', activation='relu', name='inception_
    inception_3a_3x3_reduce = Conv2D(96, (1,1), padding='same', activation='relu', name='inc
    inception_3a_3x3_pad = ZeroPadding2D(padding=(1, 1))(inception_3a_3x3_reduce)
    inception_3a_3x3 = Conv2D(128, (3,3), padding='valid', activation='relu', name='inceptio
```

```
pool5_7x7_s1 = AveragePooling2D(pool_size=(7,7), strides=(1,1), name='pool5/7x7_s2')(in
loss3_flat = Flatten()(pool5_7x7_s1)
pool5_drop_7x7_s1 = Dropout(rate=0.4)(loss3_flat)
loss3_classifier = Dense(1000, name='loss3/classifier', kernel_regularizer=l2(0.0002))(
loss3_classifier_act = Activation('softmax', name='prob')(loss3_classifier)

googlenet = Model(inputs=input, outputs=[loss1_classifier_act,loss2_classifier_act,loss
```

Inception (notebook, paper)

- Nuestra implementación, según la tabla anterior

```
def InceptionV1(input_shape,classes):
  input = Input(shape=input_shape,name="input")
  x = Conv2D(64,(7,7),strides=(2,2),padding="same",activation="relu",name="c1")(input)
  x = MaxPooling2D((3,3),strides=(2,2),padding="same",name=f"mp1")(x)
  x = Conv2D(64,(1,1),strides=(2,2),padding="same",activation="relu",name=f"c2_1x1")(x)
  x = Conv2D(192,(3,3),strides=(2,2),padding="same",activation="relu",name=f"c2_3x3")(x)
  x = MaxPooling2D((3,3),strides=(2,2),padding="same",name=f"mp2")(x)
  x = bottleneck(x,96 ,128,16,32,64 ,32,"inception3a")
  x = bottleneck(x,128,128,32,96,128,64,"inception3b")
  x = MaxPooling2D((3,3),strides=(2,2),padding="same",name=f"mp3")(x)
  x = bottleneck(x,96, 208,16,48,192,32,"inception4a")
  x = bottleneck(x,112,224,24,64,160,64,"inception4b")
  x = bottleneck(x,128,256,24,64,128,64,"inception4c")
  x = bottleneck(x,144,288,32,64,112,64,"inception4d")
  x = bottleneck(x,160,320,32,128,256,128,"inception4e")
  x = MaxPooling2D((3,3),strides=(2,2),padding="same",name=f"mp4")(x)
  x = bottleneck(x,160,320,32,128,256,128,"inception5a")
  x = bottleneck(x,192,384,48,128,384,128,"inception5b")
  x = GlobalAveragePooling2D()(x)
  x = Dense(classes,activation="softmax")(x)
```