

# Prácticas de Visión por Computador

## Grupo 2

### Presentación de la Práctica 2

### Introducción a fastai

Pablo Mesejo

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD  
DE GRANADA



# Índice

- Normas de entrega
- Presentación de la práctica
- Breve introducción a fastai

# Índice

- **Normas de entrega**
- Presentación de la práctica
- Breve introducción a fastai

# Normas de la Entrega de Prácticas

- Se entrega solamente un fichero *.ipynb* integrando directamente código, resultados, análisis y discusión.
- Disponéis de una plantilla en PRADO a partir de la cual trabajar: `Assignment_2_Template.ipynb`
  - Y también de un ejemplo que os puede servir de referencia: `Assignment_2_HG.ipynb`

# Normas de la Entrega de Prácticas

- Subid a PRADO solamente el fichero *.ipynb*.  
¡Nada de subir imágenes a PRADO!
- No escribáis nada en disco (es decir, no grabéis nada en Drive).
- La estructura de la plantilla debe ser respetada.

# Entrega

- Fecha límite: 25 de Noviembre
  - Valoración: 10 puntos (+3)
  - Lugar de entrega: PRADO
- 
- **Se valorará mucho la explicación/discusión que acompañe a código y resultados.**

# Objetivo del trabajo

- Aprender a implementar **redes convolucionales utilizando fastai/PyTorch**.
- Comprender los conceptos de **extracción de características y fine-tuning**.
- Se trata de una práctica muy orientada hacia **clasificación de imágenes usando Deep Learning**.

# ¿De qué materiales disponéis para hacer la práctica?

- Esta presentación de **introducción a la P2 y fastai**
- Una presentación de repaso sobre Deep Learning: **Repaso\_DL.pdf**
- Un template con el enunciado y rutinas de carga de datos: **Assignment\_2\_Template.ipynb**
- Una guía de ayuda con distintos códigos y ejemplos: **Assignment\_2\_HG.ipynb**



# Dudas

[pmesejo@go.ugr.es](mailto:pmesejo@go.ugr.es)

# Índice

- Normas de entrega
- **Presentación de la práctica**
- Breve introducción a fastai

# Apartado 1: BaseNet en CIFAR100 (4 ptos)

1. Partir del dataset CIFAR100
2. Crear un modelo sencillo (llamado BaseNet)
3. Entrenarlo y validarlo con esos datos



# Apartado 1: BaseNet en CIFAR100 (4 ptos)

Layer No.	Layer Type	Kernel size (for conv layers)	Input   Output dimension	Input   Output channels (for conv layers)
1	Conv2D	5	32   28	3   6
2	Relu	-	28   28	-
3	MaxPooling2D	2	28   14	-
4	Conv2D	5	14   10	6   16
5	Relu	-	10   10	-
6	MaxPooling2D	2	10   5	-
7	Linear	-	400   50	-
8	Relu	-	50   50	-
9	Linear	-	50   25	-

**Arquitectura  
que tenéis que  
implementar  
en fastai**

## Apartado 2: Mejora del modelo BaseNet (3 puntos)

- Una vez habéis implementado y validado BaseNet, debéis mejorar la red por medio de aquellas alternativas que juzguéis vosotros:
  - Normalización de datos
  - Aumento de datos
  - Aumento de profundidad de la red
  - Batch Normalization
  - Regularización
    - Dropout
    - Early-Stopping
  - ¿Otros?
- Recordad justificar siempre vuestras decisiones y mostrar claramente en la memoria la arquitectura final resultante.

# Apartado 3: Transferencia de modelos y ajuste fino con ResNet18 para la base de datos Caltech-UCSD (3 puntos).



# Apartado 3: Transferencia de modelos y ajuste fino con ResNet18 para la base de datos Caltech-UCSD (3 puntos).

1. Tenéis que utilizar **ResNet18 como extractor de características**.
  - 1.1. Usad toda la red como extractor de características e **includ solamente una nueva fully-connected layer a la salida** (que debéis re-entrenar)
  - 1.2. Ahora, en lugar de incluir solamente una fully-connected layer, **podéis incorporar bloques convolucionales** y otros bloques que consideréis necesarios.
  - 1.3. Debéis analizar y comparar los resultados de 1.1 y 1.2.
2. **Haced fine-tuning de todo ResNet18 (arquitectura que implementasteis en 1.1.1)**. Analizad estos resultados comparándolos con los anteriores.

# Bonus: propuestas innovadoras de mejora de los modelos propuestos (3 puntos)

Partid de vuestra versión mejorada de BaseNet y **probad todo lo que consideréis pertinente e interesante:**


- Podéis probar cosas que hayáis visto en la teoría o a partir de otras fuentes.
- Dad rienda suelta a vuestra creatividad e intuición.
- Se valorará mucho la innovación, la complejidad y el buen uso de PyTorch/fastai.
- Acordaos de justificar adecuadamente vuestras decisiones (no se trata de probar por probar).



# Índice

- Normas de entrega
- Presentación de la práctica
- **Breve introducción a fastai**

# Referencias Altamente Recomendadas

- **Libro con Colab Notebooks:**  
<https://github.com/fastai/fastbook>
- Curso "Practical Deep Learning for Coders 2022"  
([https://www.youtube.com/playlist?list=PLfYUBJiXbdtSvpQjSnJJ\\_PmDQB\\_VyT5iU](https://www.youtube.com/playlist?list=PLfYUBJiXbdtSvpQjSnJJ_PmDQB_VyT5iU); <https://course.fast.ai/>;  
curso 2019: <https://course19.fast.ai/videos/>).
- Jupyter Notebooks de Jeremy Howard:  <https://www.kaggle.com/jhoward/code>

Founding researcher (fast.ai),  
Distinguished Research  
Scientist (University of San  
Francisco), former President  
and Chief Scientist (Kaggle)

# Referencias Altamente Recomendadas

- Libro con Colab Notebooks:  
<https://github.com/fastai/fastbook>
- Curso "Practical Deep Learning for Coders 2022"  
([https://www.youtube.com/playlist?list=PLfYUBJiXbdtSvpQjSnJJ\\_PmDQB\\_VyT5iU](https://www.youtube.com/playlist?list=PLfYUBJiXbdtSvpQjSnJJ_PmDQB_VyT5iU); <https://course.fast.ai/>;  
curso 2019: <https://course19.fast.ai/videos/>).

**Nota sobre estos materiales:** La estrategia del libro (notebooks y vídeos) es *top-down*: partes del código, experimentas con ello, extraes intuiciones, y luego analizas cómo funciona y profundizas en los fundamentos.

# El framework de desarrollo: fastai

Este año no se usará Keras sino **fastai** (que se basa en **PyTorch**). Principal referencia: <https://docs.fast.ai/>

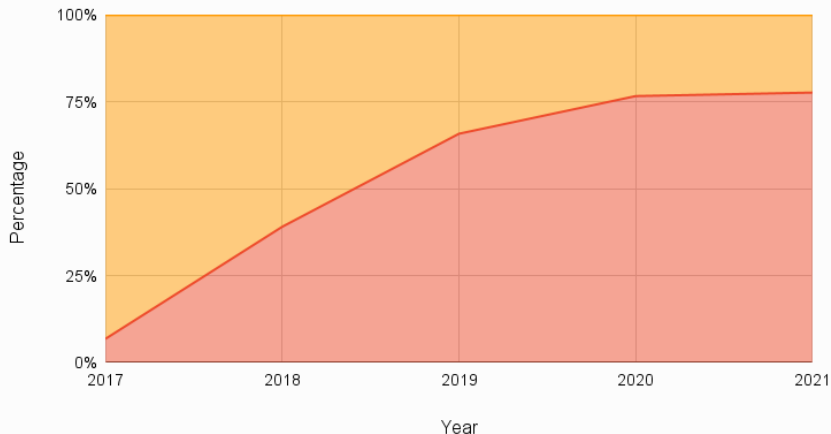
	Keras (API de TensorFlow)	fastai (API de PyTorch)
<b>Fortalezas</b>	<ul style="list-style-type: none"><li>• Muy sencillo</li></ul>	<ul style="list-style-type: none"><li>• Usa PyTorch (seguramente la herramienta de DL más popular a día de hoy)</li><li>• Más completa (permite hacer más cosas)</li></ul>
<b>Debilidades</b>	<ul style="list-style-type: none"><li>• Menos completa y flexible que fastai/PyTorch</li></ul>	<ul style="list-style-type: none"><li>• Posiblemente la curva de aprendizaje sea más larga</li><li>• Tiene tantas funcionalidades de alto nivel, que puede que no comprendáis del todo qué se hace a bajo nivel (ej. Normalización datos de test)</li></ul>

# El framework de desarrollo: fastai

Este año no se usará Keras sino **fastai** (que se basa en **PyTorch**). Principal referencia: <https://docs.fast.ai/>

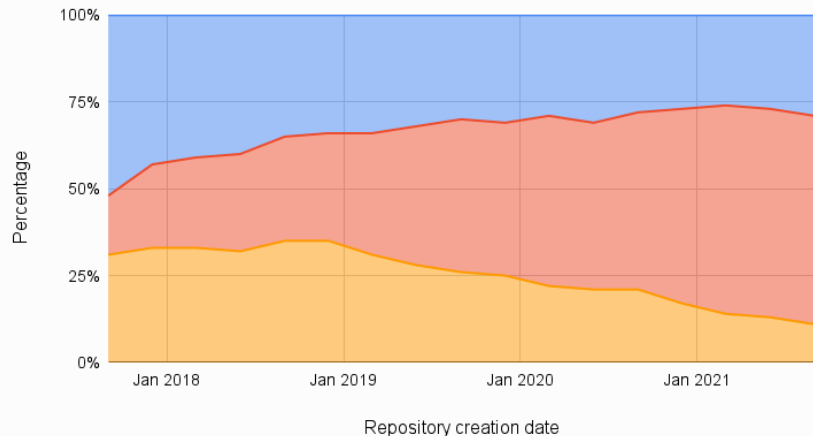
Fraction of Papers Using PyTorch vs. TensorFlow

TensorFlow PyTorch



Percentage of Repositories by Framework

Other PyTorch TensorFlow



# El framework de desarrollo: fastai

Este año no se usará Keras sino **fastai** (que se basa en **PyTorch**). Principal referencia: <https://docs.fast.ai/>

Cuando usas fastai tienes acceso a todo el potencial de PyTorch y a nuevas funcionalidades. Como consecuencia, no tienes que escribir tanto código.

Ejemplo: [https://youtu.be/8SF\\_h3xF3cE?list=PLfYUBJiXbdtSvpQjSnJJ\\_PmDQB\\_VyT5iU&t=1903](https://youtu.be/8SF_h3xF3cE?list=PLfYUBJiXbdtSvpQjSnJJ_PmDQB_VyT5iU&t=1903)

# Primeros pasos

- Clasificación de pájaros
  - <https://www.kaggle.com/code/jhoward/is-it-a-bird-creating-a-model-from-your-own-data> (necesitaréis acceso a internet para ejecutar ese Notebook → activad *SMS account verification*)
  - Presentación por parte de Jeremy Howard:  
[https://youtu.be/8SF\\_h3xF3cE?list=PLfYUBJiXbdtSvpQjSnJJ\\_PmDQB\\_VyT5iU&t=2314](https://youtu.be/8SF_h3xF3cE?list=PLfYUBJiXbdtSvpQjSnJJ_PmDQB_VyT5iU&t=2314)
- Clasificación de osos
  - [https://github.com/fastai/fastbook/blob/master/02\\_production.ipynb](https://github.com/fastai/fastbook/blob/master/02_production.ipynb)
- Clasificación de imágenes de Imagenette
  - <https://docs.fast.ai/tutorial.imagenette.html>

# Primeros pasos: carga y manipulación de datos

- DataBlock y DataLoaders

Preguntas clave que queremos responder de cara a convertir nuestros datos en un objeto de tipo *DataLoaders*:

```
dls = DataBlock(  
    blocks=(ImageBlock, CategoryBlock),  
    get_items=get_image_files,  
    splitter=RandomSplitter(valid_pct=0.2, seed=42),  
    get_y=parent_label,  
    item_tfms=[Resize(192, method='squish')])  
dls.dataloaders(path, bs=32)
```

¿Con qué tipo de dato estamos trabajando?

¿De dónde podemos sacar los ejemplos?

¿Cómo podemos disponer de un conjunto de validación?

¿De dónde obtenemos las etiquetas?



# Primeros pasos: carga y manipulación de datos

- DataBlock y DataLoaders

```
dls = DataBlock(  
    blocks=(ImageBlock, CategoryBlock),  
    get_items=get_image_files,  
    splitter=RandomSplitter(valid_pct=0.2, seed=42),  
    get_y=parent_label,  
    item_tfms=[Resize(192, method='squish')]  
)  
dls.dataloaders(path, bs=32)
```

Las entradas para nuestro modelo van a ser imágenes (*ImageBlock*) y las salidas son categorías (*CategoryBlock*), como “bird” o “forest”

# Primeros pasos: carga y manipulación de datos

- DataBlock y DataLoaders

```
dls = DataBlock(  
    blocks=(ImageBlock, CategoryBlock),  
    get_items=get_image_files,  
    splitter=RandomSplitter(valid_pct=0.2, seed=42),  
    get_y=parent_label,  
    item_tfms=[Resize(192, method='squish')]  
)  
dls.dataloaders(path, bs=32)
```

Recuperamos las imágenes por medio de la función *get\_image\_files*, que devuelve una lista con todas las imágenes en *path*

# Primeros pasos: carga y manipulación de datos

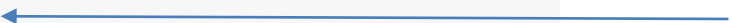
- DataBlock y DataLoaders

```
dls = DataBlock(  
    blocks=(ImageBlock, CategoryBlock),  
    get_items=get_image_files,  
    splitter=RandomSplitter(valid_pct=0.2, seed=42),  
    get_y=parent_label,  
    item_tfms=[Resize(192, method='squish')]  
)  
dls.dataloaders(path, bs=32)
```

Dividimos los datos aleatoriamente entre entrenamiento y validación (20%). Fijamos la semilla aleatoria para particionar los datos siempre de la misma manera.

# Primeros pasos: carga y manipulación de datos

- DataBlock y DataLoaders

```
dls = DataBlock(  
    blocks=(ImageBlock, CategoryBlock),  
    get_items=get_image_files,  
    splitter=RandomSplitter(valid_pct=0.2, seed=42),  
    get_y=parent_label,   
    item_tfms=[Resize(192, method='squish')]  
)  
dls.dataloaders(path, bs=32)
```

Las etiquetas (salidas deseadas) se obtienen a partir del nombre del “padre” de cada fichero (es decir, el nombre de la carpeta en la que están)

# Primeros pasos: carga y manipulación de datos

- DataBlock y DataLoaders

```
dls = DataBlock(  
    blocks=(ImageBlock, CategoryBlock),  
    get_items=get_image_files,  
    splitter=RandomSplitter(valid_pct=0.2, seed=42),  
    get_y=parent_label,  
    item_tfms=[Resize(192, method='squish')]  
)  
dls.dataloaders(path, bs=32)
```

Sobre cada ejemplo (*item*)  
se aplican una serie de  
transformaciones: *resize* a  
192x192 píxeles, y  
“*squishing*” (en oposición a  
“*cropping*”)

# Primeros pasos: carga y manipulación de datos

- DataBlock y DataLoaders

```
dls = DataBlock(  
    blocks=(ImageBlock, CategoryBlock),  
    get_items=get_image_files,  
    splitter=RandomSplitter(valid_pct=0.2, seed=42),  
    get_y=parent_label,  
    item_tfms=[Resize(192, method='squish')]  
)  
dls.dataloaders(path, bs=32)
```

Una transformación interesante es *RandomResizedCrop(size,min\_scale)*, que escoge recortes aleatorios de la imagen que incluyan al menos el *min\_scale*% de la imagen original, y hace *resize* a tamaño *size*.

# Primeros pasos: carga y manipulación de datos

- DataBlock y DataLoaders

```
dls = DataBlock(  
    blocks=(ImageBlock, CategoryBlock),  
    get_items=get_image_files,  
    splitter=RandomSplitter(valid_pct=0.2, seed=42),  
    get_y=parent_label,  
    item_tfms=[Resize(192, method='squish')]  
)  
dls.dataloaders(path, bs=32)
```

Todo el proceso indicado con anterioridad se va a realizar sobre las imágenes/carpetas en *path*. Y las imágenes se van a cargar en batches de 32.

# Primeros pasos: carga y manipulación de datos

- Normalización de datos
  - Una de las transformaciones de los datos más comunes y recomendadas.
  - Dentro del datablock:

```
batch_tfms= Normalize.from_stats(*imagenet_stats)
```

```
batch_tfms= Normalize()
```

```
batch_tfms= Normalize.from_stats(mean, std)
```

**Nota:** “when using a pretrained model through *vision\_learner*, the fastai library automatically adds the proper **Normalize transform**; the model has been pretrained with certain statistics in Normalize (usually coming from the ImageNet dataset), so the library can fill those in for you. Note that this only applies with pretrained models” ([https://github.com/fastai/fastbook/blob/master/07\\_sizing\\_and\\_tta.ipynb](https://github.com/fastai/fastbook/blob/master/07_sizing_and_tta.ipynb))



# Primeros pasos: carga y manipulación de datos

- DataBlock y DataLoaders
  - Hay distintos DataLoaders dependiendo del tipo de dato con el que queráis operar y el problema al que os enfrentéis:
    - ImageDataLoaders
    - SegmentationDataLoaders
    - LMDataLoader
    - TextDataLoader
    - TabularDataLoaders

Data block tutorial:

<https://docs.fast.ai/tutorial.datablock.html>

# Primeros pasos: modelos

- Dos opciones:
  - crear un modelo desde cero
  - cargar un modelo ya existente

# Primeros pasos: crear modelo desde cero

- Ejemplo simple:
  - Nuestras imágenes de entrada son de 32x32x3
  - Queremos una red con (en este orden):
    - Capa convolucional con 5 filtros de 7x7, sin padding y stride=1. Función de activación ReLU
    - MaxPooling de 2x2
    - Capa totalmente conectada con 15 neuronas y función de activación Softmax (con 15 clases de salida)

# Primeros pasos: crear modelo desde cero

```
simpleNet = sequential(  
    nn.Conv2d(in_channels=3,out_channels=5,kernel_size=(7,7)),  
    nn.ReLU(),  
    nn.MaxPool2d(kernel_size=(2,2)),  
    nn.Flatten(),  
    nn.Linear(in_features=845, out_features=15),  
    nn.Softmax()  
)
```

Aquí tenemos un volumen de 26x26x5

Aquí tenemos un volumen de 13x13x5, es decir, 845 elementos

El objeto *Learner* agrupa el modelo (*simpleNet*), los datos (*dls*) y la función de pérdida (*loss\_func*) para poder entrenar.

```
from torch.nn.modules.loss import CrossEntropyLoss  
learn = Learner(dls, simpleNet, loss_func=CrossEntropyLoss, metrics=accuracy)
```

datos

modelo

función de pérdida

<https://docs.fast.ai/losses.html>

métrica para evaluar el rendimiento

<https://docs.fast.ai/metrics.html>

# Primeros pasos: crear modelo desde cero

```
learn.summary()
```

Sequential (Input shape: 32 x 3 x 32 x 32)

Layer (type)	Output Shape	Param #	Trainable
Conv2d ReLU	32 x 5 x 26 x 26	740	True
MaxPool2d	32 x 5 x 13 x 13		
Flatten	32 x 845		
Linear Softmax	32 x 15	12690	True

Total params: 13,430

Total trainable params: 13,430

Total non-trainable params: 0

Batch size

`learn.summary()` os permite verificar que habéis construido correctamente la arquitectura

5 filtros con  $7 \times 7 \times 3 + 1$  (bias) parámetros a aprender

$845 \times 15 + 15$  (bias) parámetros a aprender.

¡Fijaos que el grueso de los pesos están en la fully connected! ¡Cuidado con meter muchas capas de este tipo!

# Primeros pasos: cargar modelo existente

- fastai integra numerosos modelos del estado del arte ya entrenados:
  - <https://timm.fast.ai/>
  - <https://rwightman.github.io/pytorch-image-models/results/>
- **Idea:** partir de algo que ya funciona bien en un problema similar y reutilizarlo en otro (*transfer learning*).

# Primeros pasos: cargar modelo existente

```
model = fastai.vision.models.resnet18
learn = vision_learner(dls, model)
learn.summary()
```

Cargamos el modelo pre-entrenado

Utilizamos *vision\_learner* ([https://docs.fast.ai/vision\\_learner.html](https://docs.fast.ai/vision_learner.html)): “All the functions necessary to build *Learner* suitable for transfer learning in computer vision”

Exploramos la arquitectura cargada, así como sus parámetros entrenables

*cnn\_learner*: Deprecated name for *vision\_learner* – do not use

# Primeros pasos: cargar modelo existente

Automáticamente, fastai elimina la última *fully-connected* e introduce otras capas con dimensión adaptada al problema representado por *dls*. En concreto, *vision\_learner* llama a *create\_vision\_learner* y añade lo siguiente:

AdaptiveAvgPool2d AdaptiveMaxPool2d	Sequential( (0): AdaptiveConcatPool2d( (ap): AdaptiveAvgPool2d(output_size=1) (mp): AdaptiveMaxPool2d(output_size=1) ) (1): Flatten(full=False) (2): BatchNorm1d(768, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (3): Dropout(p=0.25, inplace=False) (4): Linear(in_features=768, out_features=512, bias=False) (5): ReLU(inplace=True) (6): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (7): Dropout(p=0.5, inplace=False) (8): Linear(in_features=512, out_features=10, bias=False) )
Flatten BatchNorm1d Dropout	
Linear ReLU BatchNorm1d Dropout	
Linear	



# Primeros pasos: cargar modelo existente

- ¿Lo anterior contiene los pesos?
  - Por defecto, sí (*pretrained=True*). Véase <https://docs.fast.ai/vision.learner.html>
- ¿Cómo modificar las últimas capas de un modelo preentrenado?

```
custom_head = nn.Sequential(  
    nn.AvgPool2d((7,7)),  
    nn.Flatten(),  
    nn.Linear(512, 200),  
    nn.Softmax())
```

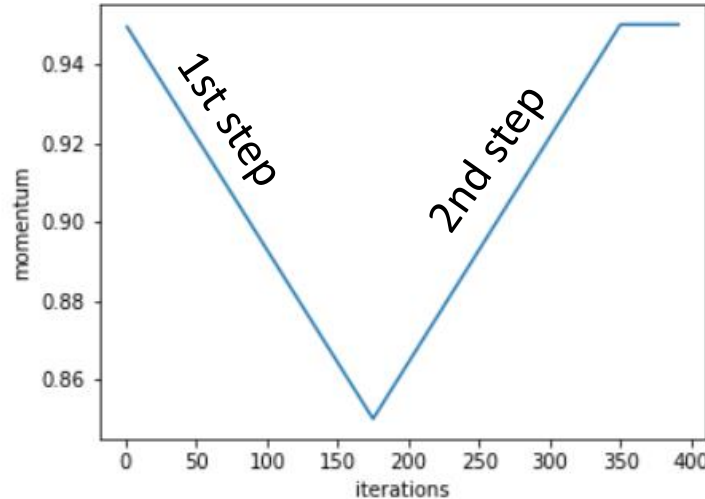
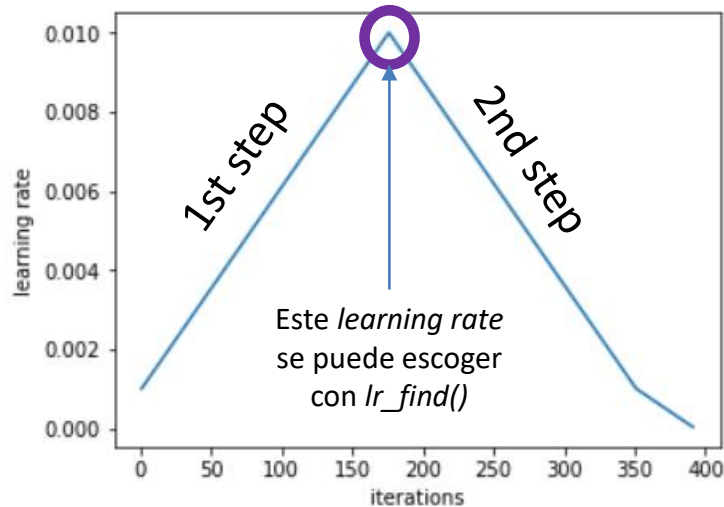
```
learn = vision_learner(dls, resnet18, custom_head=custom_head)
```

# Primeros pasos: entrenamiento

- `learn.fit(n_epoch)`
  - entrena el modelo por un determinado número de épocas
- `learn.fit_one_cycle(n_epoch)`
  - Entrena el modelo por un determinado número de épocas usando la *1cycle policy* de Leslie N. Smith (<https://arxiv.org/abs/1708.07120>)

# Primeros pasos: entrenamiento

- `learn.fit_one_cycle(n_epoch)`
  - Entrena el modelo por un determinado número de épocas usando la *1cycle policy* de Leslie N. Smith (<https://arxiv.org/abs/1708.07120>)



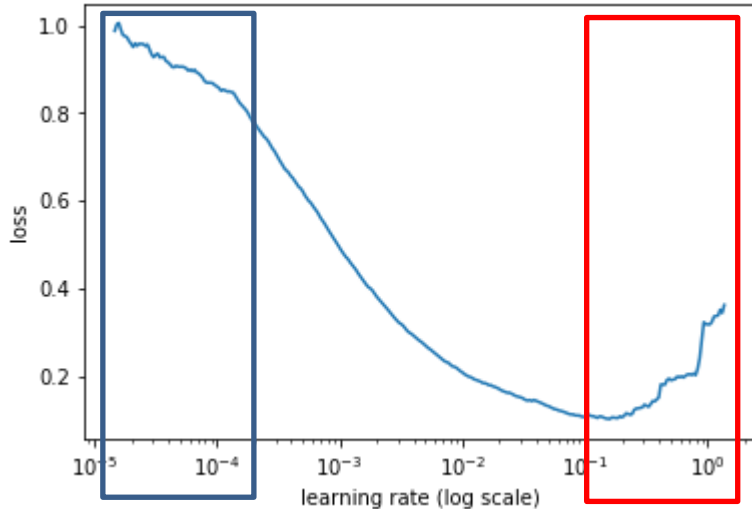
Esta estrategia permite entrenar mucho más rápido (*super-convergence*)

<https://derekchia.com/the-1-cycle-policy/>

<https://www.youtube.com/watch?v=CJKnDu2dxOE&t=7203s>

# Primeros pasos: entrenamiento

- `learn.fit_one_cycle(n_epoch)`
  - El *learning rate* máximo empleado en la *1cycle policy* se escoge con el *Learning Rate Finder*: `learner.lr_find()`



Emplea una época para construir una gráfica como la de la izquierda. Nos ayuda a escoger un *learning rate* no **demasiado grande** ni **demasiado pequeño**.

Queremos escoger un *learning rate* lo más grande posible (sin que haga que el entrenamiento diverja) para avanzar/optimizar lo más rápido posible.

<https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html>

# Primeros pasos: entrenamiento

- `learn.fine_tune(epochs)`
  - 1) Entrena las capas añadidas (*head*) por una época, con todas las otras capas “congeladas”;
  - 2) “Descongela” todas las capas, y las entrena por el número de épocas indicado.

# Primeros pasos: entrenamiento

- Prestad atención a la relación entre distintos parámetros: batch size, learning rate (lr), weight decay (wd), momentum
  - *lr pequeño favorece el sobreajuste → deben usarse valores grandes (sin pasarse);*
  - *batch size pequeño regulariza → en modelos pequeños quizás mejor usar batch grandes;*
  - *wd debe fijarse a valores pequeños (a mayor valor, mayor regularización), de lo contrario nos costará ajustar los datos. Esto nos permitirá usar valores de lr grandes;*
  - *si tenemos un lr grande y también un momentum elevado corremos el riesgo de no converger.*

**Sharp vs Flat Minima**

# Primeros pasos: entrenamiento

- Sobre múltiples llamadas a *fit* o *fit\_one\_cycle*
  - al igual que Keras, si se llama varias veces a estas funciones, estamos entrenando el modelo incrementalmente desde el punto (los pesos) en que nos quedamos tras la llamada anterior.
- Posibilidad interesante (con modelos preentrenados): *discriminative learning*
  - En la función de entrenamiento, emplear *slice()* para indicar el *learning rate* a usar.
    - Ejemplo: *slice(1e-5, 1e-3)*; en el head entrenará con 1e-3 y en capas anteriores utilizará 1e-5 en las primeras y 1e-4 en las intermedias.

# Primeros pasos: predicción

- Una vez que tenemos entrenado nuestro modelo podemos:
  - Realizar la predicción sobre un único ejemplo: `learn.predict(example)`
  - Realizar la predicción sobre un conjunto de ejemplos (test):

```
test_dl = learn.dls.test_dl(files_test,with_labels=True)
preds, targs = learn.get_preds(dl=test_dl)
```
- Resulta clave escalar/normalizar los datos de test siguiendo, exactamente, el mismo protocolo empleado en training (usando misma media y std)
  - Esto fastai, al usar `learn.get_preds` o `learn.dls.test_dl`, ya lo hace automáticamente por vosotros.
    - <https://forums.fast.ai/t/do-we-need-to-normalize-single-image-before-running-predict-function-on-it/44301/3>
    - <https://forums.fast.ai/t/99-accuracy-on-valid-data-1-accuracy-on-test-data-what-am-i-missing/80408/2>



# Primeros pasos: interpretación de resultados

- `interp = ClassificationInterpretation.from_learner(learn)`
- `interp.plot_confusion_matrix(figsize=(12, 12), title='Title')`
- `interp.most_confused(min_val=10)`
- `interp.plot_top_losses(10, nrows=2, figsize=(32, 4))`

Y muchas otras posibilidades en <https://docs.fast.ai/interpret.html>

# Consejos prácticos

- “It's only by practicing (and failing) a lot that you will get an intuition of how to train a model.”
- No dudéis en consultar la ayuda online directamente en el Notebook:

*??function*

*doc(function)*

*??learn.fine\_tune*

*doc(learn.fine\_tune)*

# Prácticas de Visión por Computador

## Grupo 2

### Presentación de la Práctica 2

### Introducción a fastai

Pablo Mesejo

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD  
DE GRANADA

