Computer Vision: Assignment 2
Deep Learning for Computer Vision

Pablo Mesejo and Víctor Vargas
Universidad de Granada
Departamento de Ciencias de la Computación e Inteligencia Artificial

Index

- Submission rules
- Assignment description
- Brief introduction to fastai

Index

- **Submission rules**
- Assignment description
- Brief introduction to fastai

Submission rules

- Only *.ipynb* files should be submitted (including code, results, analysis and discussion).
  - Don’t upload the images!

- Don’t write anything to disc/Drive.

- The template estructure must be respected.

Submission

- Deadline: 26 November
- Maximum score: 10 points
- Submission Site: https://pradogrado2324.ugr.es/

- **Explanation/discussion accompanying code and results is essential.**

Goals

- Learn how to implement **convolutional neural networks using fastai/PyTorch**.
- Understand the concepts of **feature extraction** and **fine-tuning**.

- This is an assignment oriented towards **image classification using Deep Learning**.

Materials at your disposal

- This **introduction to the P2 and fastai**
- A Deep Learning overview: **DL_Review.pdf**
- Templates:
  Assignment_2_Template_parts1and2.ipynb
  Assignment_2_Template_part3.ipynb
- A help guide with different codes and examples:
  Assignment_2_HG.ipynb

Index

- Submission rules
- **Assignment description**
- Brief introduction to fastai

Section 1: BaseNet in CIFAR100 (3 points)

1. Start from the CIFAR100 dataset
2. Create a simple model (called BaseNet)
3. Train and validate it with these data

Exercise 1: BaseNet in CIFAR100 (3 points)

Layer Type	Kernel Size (for convolutional layers)	Input | Output dimension	Input | Output channels (for convolutional layers)
Conv	7x7	32x32 | 26x26	3 | 4
ReLU	-	26x26 | 26x26	-
MaxPooling	2x2	26x26 | 13x13	-
Conv	5x5	13x13 | 9x9	4 | 10
ReLU	-	9x9 | 9x9	-
FC	-	810 | 50	-
ReLU	-	50 | 50	-
FC	-	50 | 25	-

Architecture you have to implement in fastai

Taking into account that this is a multiclass classification problem, what is the most natural/common choice for the activation function and loss function? 🤔

Exercise 2: Improvement of the BaseNet model

(3.5 points)

- Once you have implemented and validated BaseNet, you should improve the network by means of those alternatives that you judge to be appropriate:
  - Data normalization
  - Data augmentation
  - Network depth augmentation
  - Batch normalization
  - Regularization
    - Dropout
    - Early-Stopping
  - Others?
    - You can try things you have seen in theory or from other sources.
    - Unleash your creativity and intuition.
    - Innovation, complexity and good use of PyTorch/fastai will be highly valued.

- Always remember to **justify your decisions** (it is not about testing for the sake of testing) and to **clearly show the resulting final architecture**.

Section 3: Model transfer and fine-tuning with ResNet18 for Caltech-UCSD database (3.5 points).

Section 3: Model transfer and fine-tuning with ResNet18 for Caltech-UCSD database (3.5 points).

1. **Use ResNet18 as a feature extractor**:
   i. Remove the final fully-connected (FC) layer of ResNet18, replace it by a FC layer of the dimensionality of the new problem, and **train the new weights of this FC layer** (while keeping frozen the remaining weights in the network).
   ii. Instead of a single FC layer, employ the **head introduced by default in fastai**. Train these new weights (while keeping frozen the remaining weights in the network).
   iii. **Create your own head, combining all types of blocks you want** (convolutional, FC layers,...). Train all these new weights (while keeping frozen the remaining weight in the network).

2. Make a **fine-tuning of the entire ResNet18**.

3. **Train from scratch the entire ResNet18**.

Index

- Submission rules
- Assignment description
- **Brief introduction to fastai**

Highly Recommended References

- **Book with Colab Notebooks: https://github.com/fastai/fastbook**

- Course "Practical Deep Learning for Coders 2022" (https://www.youtube.com/playlist?list=PLfYUBJiXbdtSvpQjSnJJ_PmDQB_VyT5iU; https://course.fast.ai/; academic year 2019: https://course19.fast.ai/videos/).

- Jupyter Notebooks of 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)

Highly Recommended References

- **Book with Colab Notebooks: https://github.com/fastai/fastbook**

- Course "Practical Deep Learning for Coders 2022" (https://www.youtube.com/playlist?list=PLfYUBJiXbdtSvpQjSnJJ_PmDQB_VyT5iU; https://course.fast.ai/; academic year 2019: https://course19.fast.ai/videos/).

Note on these materials: The strategy of the book (notebooks and videos) is *top-down*: you start from the code, experiment with it, extract intuitions, and then analyze how it works and go deeper into the fundamentals.

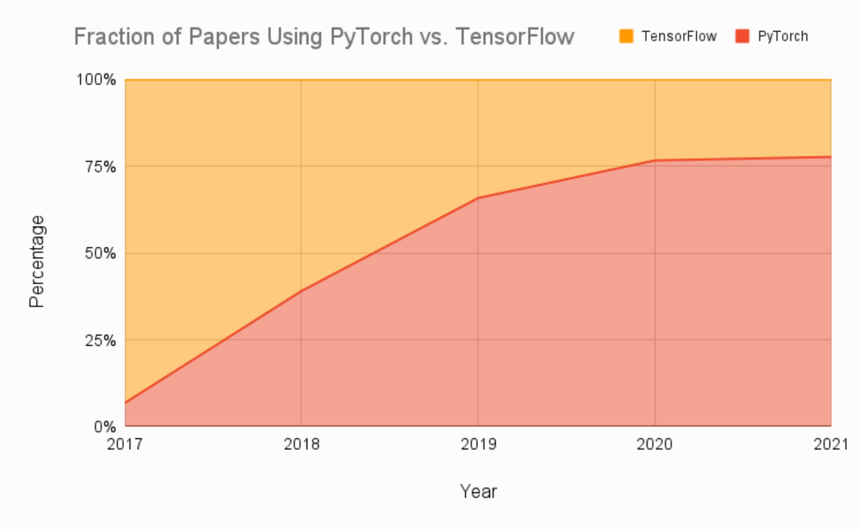
The development framework: fastai

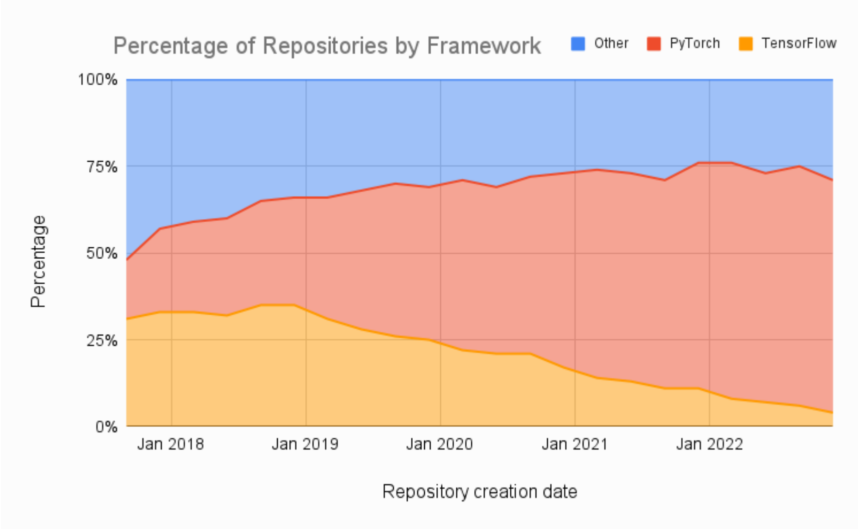
We use **fastai** (based on **PyTorch**).
Main reference: https://docs.fast.ai/

	Keras (TensorFlow API)	fastai (PyTorch API)
Strenghs	• Very simple	• Uses PyTorch (probably the most popular DL tool today) • More complete (allows you to do more things)
Weaknesses	• Less complete and flexible than fastai/PyTorch	• Possibly, longer learning curve • It has so many high-level funcionalities, that you may not fully understand what is being done at lower level (e.g. test data normalization)

The development framework: fastai

We use **fastai** (based on **PyTorch**).
Main reference: https://docs.fast.ai/





https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/

The development framework: fastai

We use **fastai** (based on **PyTorch**).
Main reference: https://docs.fast.ai/

When using fastai you have access to the full potential of PyTorch and new features. As a result, you do not have to write so much code.

Example using AdamW's step in PyTorch vs fastai:
https://youtu.be/8SF_h3xF3cE?list=PLfYUBJiXbdtSvpQjSnJJ_PmDQB_VyT5iU&t=1903

First steps

- Bird classification
  - https://www.kaggle.com/code/jhoward/is-it-a-bird-creating-a-model-from-your-own-data (you will need internet access to run this Notebook → activate *SMS account verification*)
  - Presentation by Jeremy Howard: https://youtu.be/8SF_h3xF3cE?list=PLfYUBJiXbdtSvpQjSnJJ_PmDQB_VyT5iU&t=2314
- Bear classification
  - https://github.com/fastai/fastbook/blob/master/02_production.ipynb
- Imagenette images classification
  - https://docs.fast.ai/tutorial.imagenette.html

First steps: data loading and manipulation

- DataBlock and DataLoaders

Key questions we want to answer to convert our data into a *DataLoaders* object:

```
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')]
).dataloaders(path, bs=32)
```

What kind of data are we working with?

Where can we get the examples from?

How can we have a validation set?

Where do we get the labels from?

First steps: data loading and manipulation

- DataBlock and 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')]
).dataloaders(path, bs=32)
```

The inputs for our model will be images (*ImageBlock*) and the outpust are categories (*CategoryBlock*), such as "*bird*" or "*forest*"

First steps: data loading and manipulation

- DataBlock and 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')]
).dataloaders(path, bs=32)
```

We retrieve the images using the *get_image_files* function, which returns a list with all the images in *path* (https://docs.fast.ai/data.transforms.html#get_image_files)

First steps: data loading and manipulation

- DataBlock and 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')]
).dataloaders(path, bs=32)
```

We split the data randomly between training and validation (20%). We set the random seed to partition the data always in the same way.

First steps: data loading and manipulation

- DataBlock and 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')]
).dataloaders(path, bs=32)
```

The labels (desired outputs) are obtained from the name of the parent directory of each file (i.e. the name of the folder they are in) (https://docs.fast.ai/data.transforms.html#parent_label)

First steps: data loading and manipulation

- DataBlock and 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')]
).dataloaders(path, bs=32)
```

On each example (*item*) a series of transformations is applied: *resize* to 192x192 pixels, and “*squishing*” (as opposed to “*cropping*”)

First steps: data loading and manipulation

- DataBlock and 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')]
).dataloaders(path, bs=32)
```

An interesting transformation is *RandomResizedCrop(size,min_scale)*, which picks random crops that include at least *min_scale*% of the original image, and resizes to *size*.

https://github.com/fastai/fastbook/blob/master/02_production.ipynb
https://docs.fast.ai/vision.augment.html#randomresizedcrop

First steps: data loading and manipulation

- DataBlock and 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')]
).dataloaders(path, bs=32)
```

All this process will be performed on the images/folders in *path*. And the images will be loaded in batches (*bs*) of 32.

First steps: data loading and manipulation

- Data normalization
  - One of the most common and recommended data transformations.
  - Inside the datablock:

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

Note: “**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)

First steps: data loading and manipulation

- DataBlock and DataLoaders
  - There are different DataLoaders depending on the type of data you want to deal with and the problem you face:
    - ImageDataLoaders
    - SegmentationDataLoaders
    - LMDataLoader
    - TextDataLoader
    - TabularDataLoaders

Data block tutorial:
https://docs.fast.ai/tutorial.datablock.html
DataLoaders documentation:
https://docs.fast.ai/data.load.html

First steps: models

- Two options:
  - creating a model from scratch
  - loading an existing model

First steps: creating a model from scratch

- Simple example:
  - Our input images are 32x32x3
  - We want a network with (in this order):
    - Convolutional layer with 5 7x7 filters, no padding and stride=1. ReLU activation function.
    - 2x2 MaxPooling.
    - Fully-connected layer with 15 neurons and Softmax activation function (with 15 output classes).

First steps: creating a model from scratch

```
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()
)
```

Here we have a volume of 26x26x5

Here we have a volume of 13x13x5, i.e. 845 elements

Is this really necessary?? Carefully check the documentation.

The *Learner* object includes the model (*simpleNet*), the data (*dls*) and the loss function (*loss_func*). We have everything to train our model.

```
learn = Learner(dls, simpleNet, loss_func=CrossEntropyLossFlat(), metrics=accuracy)
```

data

model

loss function
https://docs.fast.ai/losses.html

Metric for assessing performance
https://docs.fast.ai/metrics.html

First steps: creating a model from scratch

```
learn.summary()
```

```
Sequential (Input shape: 32 x 3 x 32 x 32)
============================================================================
Layer (type)         Output Shape         Param #    Trainable
============================================================================
                     32 x 5 x 26 x 26
Conv2d                                    740        True
ReLU
____________________________________________________________________________
                     32 x 5 x 13 x 13
MaxPool2d
____________________________________________________________________________
                     32 x 845
Flatten
____________________________________________________________________________
                     32 x 15
Linear                                    12690      True
Softmax
____________________________________________________________________________

Total params: 13,430
Total trainable params: 13,430
Total non-trainable params: 0
```

Batch size

`learn.summary()` allows you to verify that you have built the architecture correctly

5 filters with 7x7x3+1 (bias) parameters to learn

845*15 + 15 (bias) parameters to learn.

Note that **most of the weights are in the fully-connected**! Be careful not to include too many layers of this type when you create your own architecture!

First steps: loading an existing model

- fastai integrates numerous trained state-of-the-art models:
  - https://timm.fast.ai/
  - https://rwightman.github.io/pytorch-image-models/results/
- **Idea**: take something that already works well for a similar problem and reuse it in another problem (*transfer learning*).

First steps: loading an existing model

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

We load the pre-trained model

We use *vision_learner* (https://docs.fast.ai/vision.learner.html): “All the functions necessary to build *Learner* suitable for transfer learning in computer vision”

We explore the architecture, as well as its trainable parameters

cnn_learner: Deprecated name for *vision_learner* – do not use it

First steps: loading an existing model

Automatically, fastai removes the last *fully-connected* and introduces other layers with dimension adapted to the problem represented by *dls*. Specifically, *vision_learner* calls *create_vision_learner* and adds the following:

AdaptiveAvgPool2d
AdaptiveMaxPool2d
	`Sequential(`
	`(0): AdaptiveConcatPool2d(`
	`(ap): AdaptiveAvgPool2d(output_size=1)`
	`(mp): AdaptiveMaxPool2d(output_size=1)`
	`)`

Flatten
BatchNorm1d
Dropout
	`(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)`

Linear
ReLU
BatchNorm1d
Dropout
	`(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)`

Linear	`(8): Linear(in_features=512, out_features=10, bias=False)` `)`

https://wandb.ai/capecape/imagenette_timm/reports/Checking-Out-the-New-fastai-timm-Integration--VmlldzoxOTMzNzMw

First steps: loading an existing model

- Does the above contain the weights?
  - By default, yes (*pretrained=True*). See https://docs.fast.ai/vision.learner.html
- How to modify the last layers of a pre-trained model?

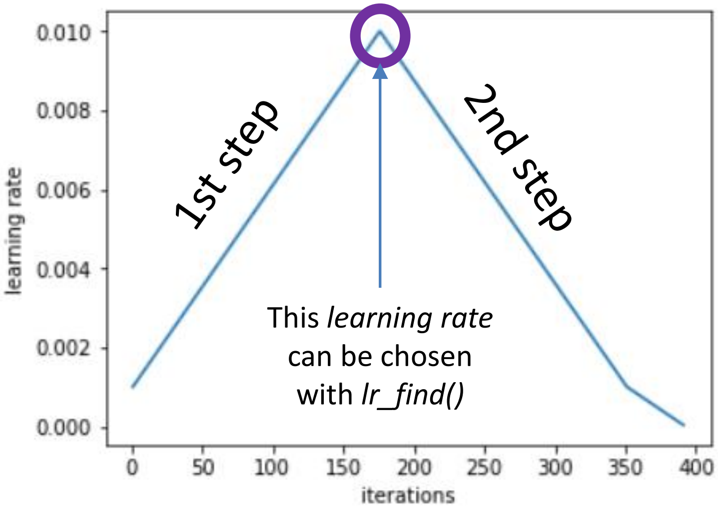
```
custom_head = nn.Sequential(
    nn.AvgPool2d((7,7)),
    nn.Flatten(),
    nn.Linear(512, 200))

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

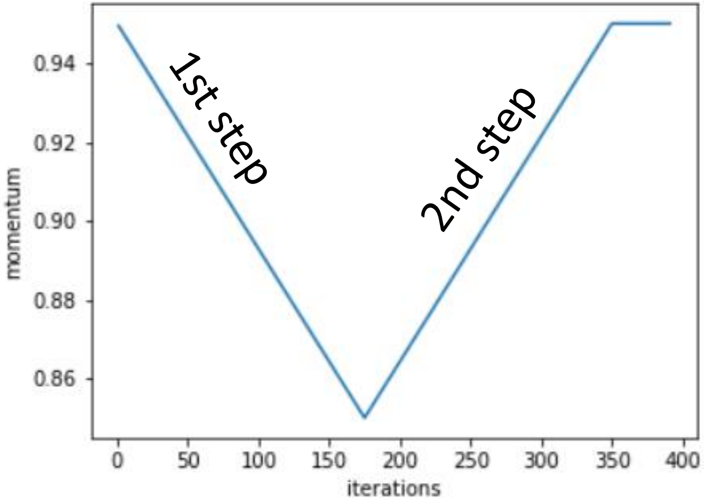
First steps: training

- learn.fit(n_epoch)
  - Trains the model for a certain numbers of epochs
- learn.fit_one_cycle(n_epoch)
  - Trains the model for a certain number of epochs using the *1cycle policy* of Leslie N. Smith (https://arxiv.org/abs/1708.07120)

First steps: training

- learn.fit_one_cycle(n_epoch)
  - Trains the model for a certain number of epochs using the *1cycle policy* of Leslie N. Smith (https://arxiv.org/abs/1708.07120)





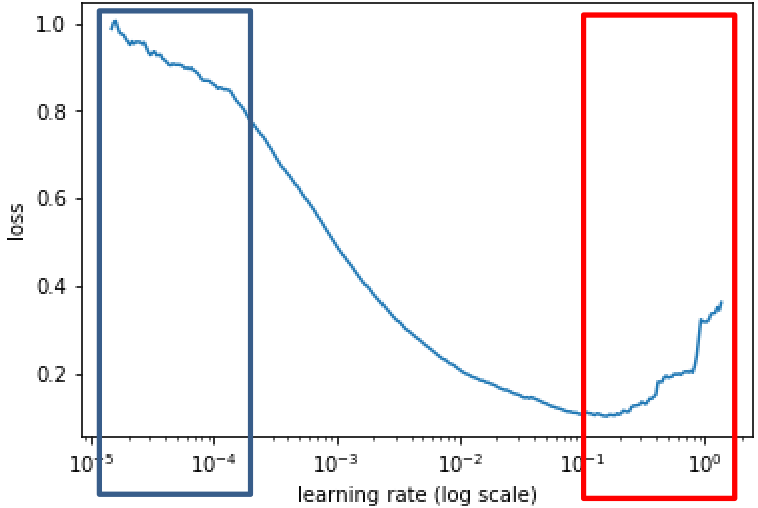
This combined strategy allows you to train much faster (*super-convergence*)

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

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

First steps: training

- learn.fit_one_cycle(n_epoch)
  - The maximum *learning rate* used in the *1cycle policy* is chosen with the *Learning Rate Finder*: *learner.lr_find()*



It uses an epoch to build a graph like the one on the left. It helps us to choose a *learning rate* not **too big** or **too small**.
We want to choose a *learning rate* as large as possible (without making the training diverge) to advance/train/optimize as fast as possible.

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

First steps: training

- learn.fine_tune(epochs, freeze_epochs)

  1) Trains the added layers (*head*) for one epoch (by default, freeze_epochs=1), with all other layers “frozen”.

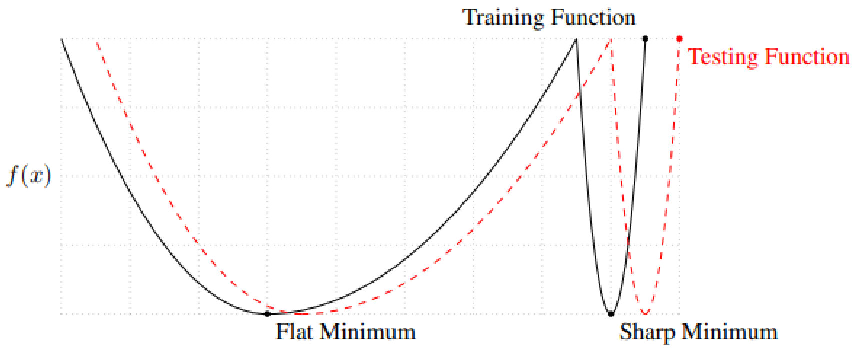
  2) “Unfreezes” all layers, and trains them for the indicated number of epochs.

First steps: training

- Pay attention to the relationship between different parameters: batch size (*bs*), learning rate (*lr*), weight decay (*wd*), momentum
  - small *lr* favors overfitting → large values should be used (but avoiding too large values that may prevent from converging).
  - small *bs* regularizes (it appears to locate minima with better generalization properties) → in small models perhaps better use big *bs*.
  - *wd* should be set to small values (the larger the value, the more regularization (higher large weights penalization)), otherwise it will be difficult to fit the data.
  - if we have a large *lr* and also a high momentum we run the risk of not converging.

First steps: training

- Relevant concepts:
  - Sharp vs Flat Minima / Optimization of our loss function vs Generalization ability
    - https://www.inference.vc/sharp-vs-flat-minima-are-still-a-mystery-to-me/
    - https://towardsdatascience.com/what-can-flatness-teach-us-understanding-generalisation-in-deep-neural-networks-a7d66f69cb5c
    - Zhou et al. (2020). Towards theoretically understanding why SGD generalizes better than Adam in deep learning. *Advances in Neural Information Processing Systems*, *33*, 21285-21296.
    - Hochreiter, S., & Schmidhuber, J. (1997). Flat minima. *Neural computation*, 9(1), 1-42.
    - Keskar et al. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836.



First steps: training

- Multiple calls to *fit* or *fit_one_cycle*
  - like Keras, if these functions are called several times, we'd be training the model incrementally from the point/weights obtained from the previous call.
- Interesting possibility (with pre-trained models): *discriminative learning rates*
  - In the training function, use *slice()* to indicate the *learning rate*.
    - Example: `learn.fit_one_cycle(3, lr_max=slice(1e-5, 1e-3))`

    The head will train with 1e-3 and in previous layers will use smaller learning rates (1e-5 in the first layer group and 1e-4 in the second one).

    `learn.fine_tune()` includes it by default.

If you start with an uninitialized network, that you want to train from scratch, would it make sense to use it?

First steps: training

- You can define, initialize and use different optimizers
  - https://docs.fast.ai/optimizer.html
  - Example using Adagrad (https://pytorch.org/docs/stable/generated/torch.optim.Adagrad.html)

```
learn = Learner(dls, yourNetwork, metrics=accuracy,
                loss_func=LabelSmoothingCrossEntropy(),
                     opt_func=partial(OptimWrapper,
                           opt=torch.optim.Adagrad,
                           lr=0.0001,lr_decay=0.01,
                                weight_decay=0.1))
```

First steps: prediction

- Once we have trained our model we can:
  - Perform prediction on a single example: `learn.predict(example)`
  - Perform prediction on a set of examples (test):

```
test_dl = learn.dls.test_dl(files_test,with_labels=True)
preds, targs = learn.get_preds(dl=test_dl)
```

- It is key to scale/normalize the test data following exactly the same protocol used in training (using the same mean and std)
  - This is done automatically for you by fastai, using `learn.get_preds` or `learn.dls.test_dl`.
    - 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

First steps: interpretation of the results

- `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))`

And many other possibilities in https://docs.fast.ai/interpret.html

Problems with Colab?

- General advice: **use Colab resources judiciously**. Otherwise, RAM problems may arise or you may be temporarily blocked/restricted from using the GPUs.

- **Don't pay for the Colab Pro version!!!**

- If you employ good coding practices, and reasonable experiments are carried out in an orderly manner, there should be no problem.

Problems with Colab?

1) Modify Google Colab services.

- For instance, increase the available RAM in Colab (https://analyticsindiamag.com/5-google-colab-hacks-one-should-be-aware-of/)

2) Optimize the code.

- The type of data used could be optimized (e.g. https://stackoverflow.com/questions/62977311/how-can-i-stop-my-colab-notebook-from-crashing-while-normalising-my-images).
- It is also advisable to eliminate unnecessary objects that may be in memory (**`del command`**) and/or use the *garbage collector* to free memory (https://stackoverflow.com/questions/61188185/how-to-free-memory-in-colab)

3) Divide the Notebook into several files, which would be executed independently. When restarting the runtime between exercises there should be no problem.

General advice

- **"It's only by practicing (and failing) a lot that you will get an intuition of how to train a model."**
- Do not hesitate in consulting the online help directly in the Notebook:

??function `??learn.fine_tune`

doc(function) `doc(learn.fine_tune)`

Computer Vision: Assignment 2
Deep Learning for Computer Vision

Pablo Mesejo and Víctor Vargas
Universidad de Granada
Departamento de Ciencias de la Computación e Inteligencia Artificial