Transfer Learning

Basic concepts and code ecamples

MAC 2014-2020
Cooperación Territorial

Interreg
Fondo Europeo de Desarrollo Regional
EUROPEAN UNION

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

T
M E D
C
Tecnología Médica
para el Desarrollo Sostenible

1
Basic concepts

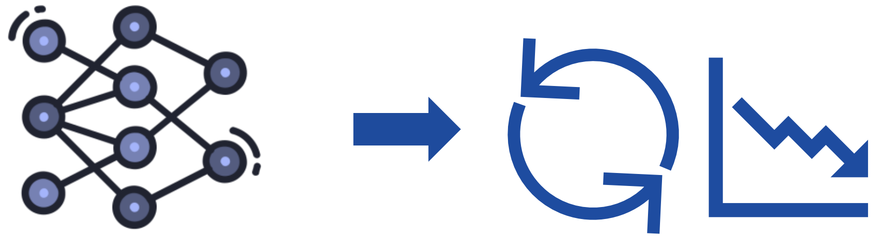
1.1 Transfer learning: Definition

Reminder

In section 3 we have mentioned that the traditional training process consists of **initializing the network parameters randomly** and optimizing them based on the dataset.

However, this process can be **inefficient** when a sufficiently large dataset is not available to train the model on a problem.

Transfer learning is a technique in which we use **models that have been previously trained** (usually with large databases) to **apply them to a specific dataset of interest**. Working with pre-trained networks allows us to take advantage of the knowledge acquired by these models and apply it to another related task.



Building or picking a pretrained model

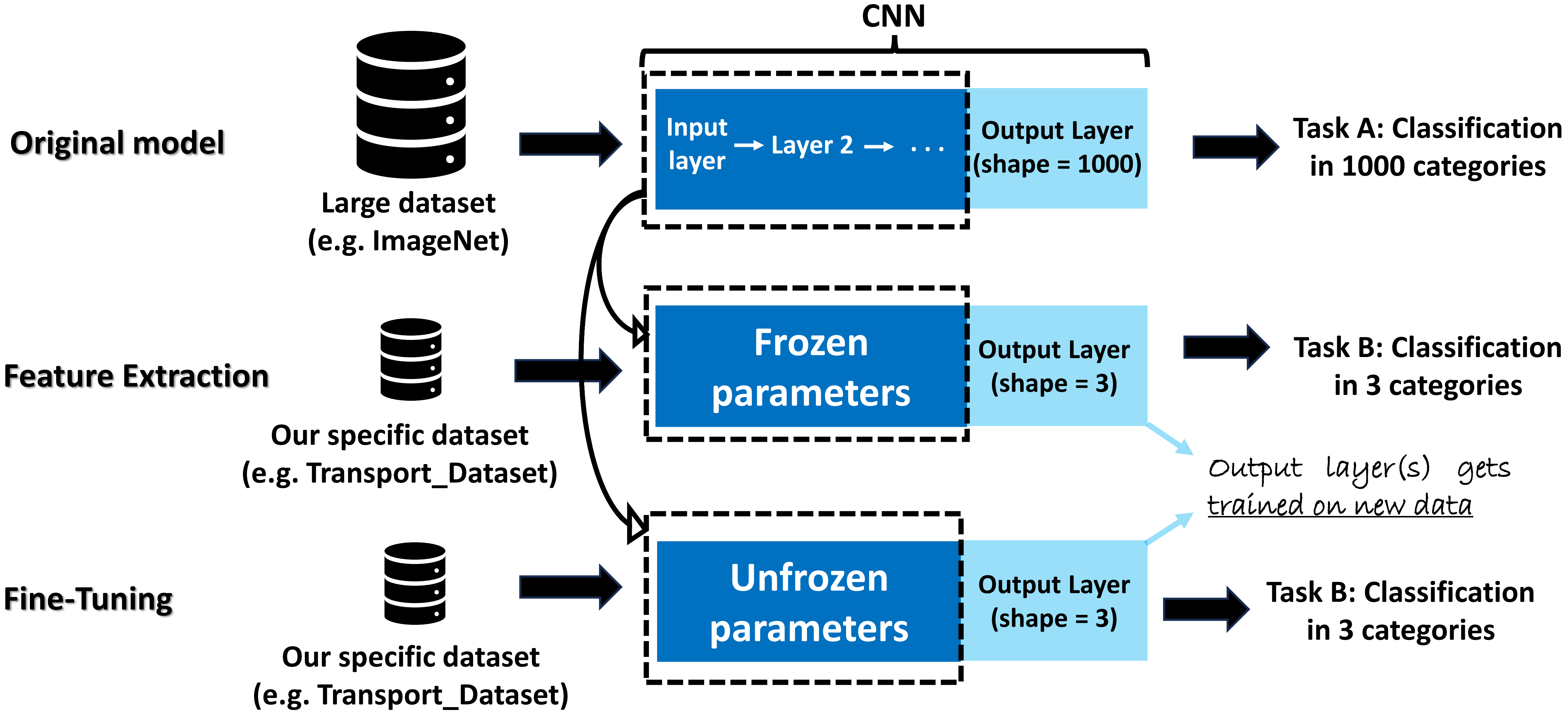
Training

When **transfer learning** is used, the model **performs better** than when **trained from scratch** (with random initialization)

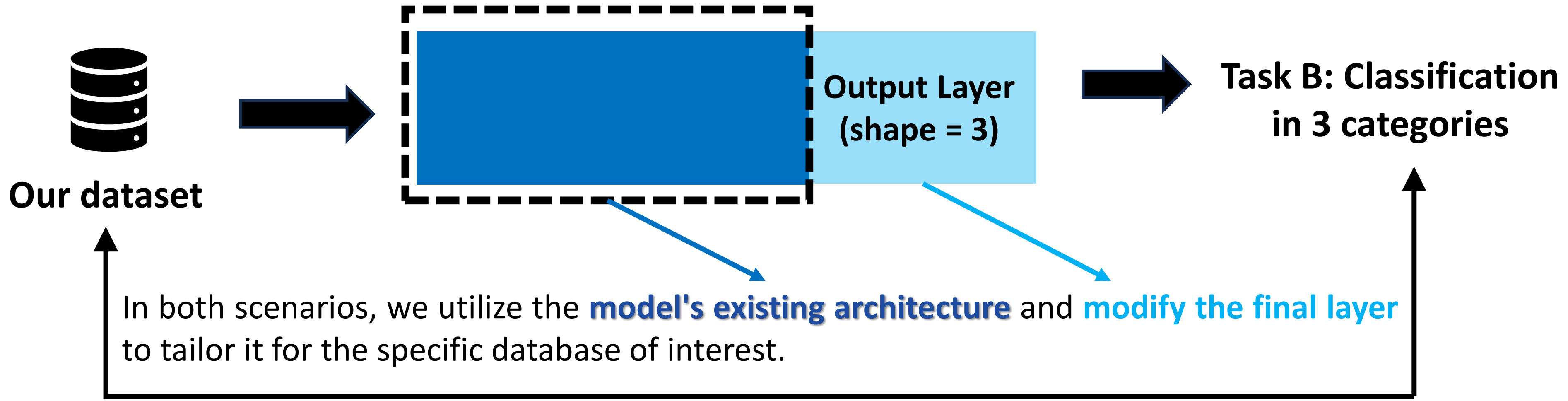
1.2 Feature extraction and fine-tuning

There are mainly two approaches to transfer knowledge from one model to another: **Feature extraction** and **Fine tuning**.

“Frozen”: original model layers **don’t update** during training



1.2 Feature extraction and fine-tuning



Feature extraction: During feature extraction, the weights of the initial layers are frozen, meaning they are not updated during the training process on the new task. Only the additional layers (usually a new classifier) added on top of the pre-trained model are trained on the new data.

Fine-tuning: It involves updating the weights of many or all the layers of the pre-trained model, including both the feature extractor and the classifier layers, on the new task-specific data. Fine-tuning usually requires more data than feature extraction

2
Transfer learning in PyTorch

2.1 Get a model

There are several places you can find pretrained models to use for your own problems:

- PyTorch domains libraries (e.g torchvision.models)
- HuggingFace Hub
- Torch Image Models (timm library)

```python
import torchvision
model = torchvision.models.resnet18(weights = 'IMAGENET1K_V1')
print(model)
```

The original model is pretrained on a large dataset, such as ImageNet

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
                         . . .
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

Extracts features from image. We will maintain the model's existing architecture

Turns features into a feature vector

Utilizing the feature vector, the model generates predictions for 1000 classes

2.2 Replace the Fully Connected layer

Substitute the last layer of the model for a classification into three categories:

```
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 3)
```

```
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=3, bias=True)
)
```

Output Layer (shape = 3)

To learn more about our model, let's use torchinfo.summary()

```
==========================================================================================================
Layer (type (var_name))              Input Shape           Output Shape          Param #        Trainable
==========================================================================================================
ResNet (ResNet)                      [32, 3, 224, 224]     [32, 2]               --             True
├─Conv2d (conv1)                     [32, 3, 224, 224]     [32, 64, 112, 112]    9,408          True
├─BatchNorm2d (bn1)                  [32, 64, 112, 112]    [32, 64, 112, 112]    128            True
├─ReLU (relu)                        [32, 64, 112, 112]    [32, 64, 112, 112]    --             --
├─MaxPool2d (maxpool)                [32, 64, 112, 112]    [32, 64, 56, 56]      --             --
├─Sequential (layer1)                [32, 64, 56, 56]      [32, 64, 56, 56]      --             True
│    └─BasicBlock (0)                [32, 64, 56, 56]      [32, 64, 56, 56]      --             True
│    │    └─Conv2d (conv1)           [32, 64, 56, 56]      [32, 64, 56, 56]      36,864         True
│    │    └─BatchNorm2d (bn1)        [32, 64, 56, 56]      [32, 64, 56, 56]      128            True

        . . .

├─AdaptiveAvgPool2d (avgpool)        [32, 512, 7, 7]       [32, 512, 1, 1]       --             --
├─Linear (fc)                        [32, 512]             [32, 2]               1,026          True
==========================================================================================================
Total params: 11,177,538
Trainable params: 11,177,538
Non-trainable params: 0
Total mult-adds (G): 58.03
```

Input and output shape of data per layer

Informs if the layers are unfrozen

Total number of parameters and trainable parameters

The entire model is trainable: Fine tuning

2.3 Choose the trainable layers/parameters

```python
model = torchvision.models.resnet18(weights = 'IMAGENET1K_V1')

for param in model.parameters():
    param.requires_grad = False

num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 3)
```

Freezing the model's parameters to avoid updating them during training

```
==========================================================================================================
Layer (type (var_name))               Input Shape          Output Shape         Param #           Trainable
==========================================================================================================
ResNet (ResNet)                       [32, 3, 224, 224]    [32, 2]              --                Partial
├─Conv2d (conv1)                      [32, 3, 224, 224]    [32, 64, 112, 112]   (9,408)           False
├─BatchNorm2d (bn1)                   [32, 64, 112, 112]   [32, 64, 112, 112]   (128)             False
├─ReLU (relu)                         [32, 64, 112, 112]   [32, 64, 112, 112]   --                --
├─MaxPool2d (maxpool)                 [32, 64, 112, 112]   [32, 64, 56, 56]     --                --
├─Sequential (layer1)                 [32, 64, 56, 56]     [32, 64, 56, 56]     --                False
│    └─BasicBlock (0)                 [32, 64, 56, 56]     [32, 64, 56, 56]     --                False
│    │    └─Conv2d (conv1)            [32, 64, 56, 56]     [32, 64, 56, 56]     (36,864)          False

            • • •
├─AdaptiveAvgPool2d (avgpool)         [32, 512, 7, 7]      [32, 512, 1, 1]      --                --
├─Linear (fc)                         [32, 512]            [32, 3]              1,539             True
==========================================================================================================
Total params: 11,178,051
Trainable params: 1,539
Non-trainable params: 11,176,512
Total mult-adds (G): 58.03
```

Most layers are untrainable (frozen)

Less trainable parameters (That is why feature extraction tends to result in faster training)

Only last layer is trainable: Feature extraction

2.2 Code examples

Let’s code!

To gain hands-on experience with transfer learning, we will utilize a pretrained ResNet-18 model on the Imagenet database and fine-tune it for our specific 4-class classification problem. Subsequently, we will examine some inference examples and assess the performance of the model.

- **notebooks**
  - 05_transfer_learning.ipynb
  - **exercise_solutions**
    - 05_transfer_learning_solutions.ipynb

We will see that, compared to the training carried out in section 3, the use of Transfer learning has 2 fundamental advantages:

- It can leverage an **existing neural network architecture** proven to work on problems similar to the current task.
- It can leverage a working network architecture that **has already learned patterns from similar data** to the current task, often resulting in **superior results with less data.**