Lecture 4

Convolutional neural networks

Data augmentation, Batch normalization, Dropout

Transfer learning et Finetuning

Minh-Tan Pham, Matthieu Le Lain

BUT2 INFO, 2023-2024

IUT de Vannes, Université Bretagne Sud

minh-tan.pham@univ-ubs.fr

Planning

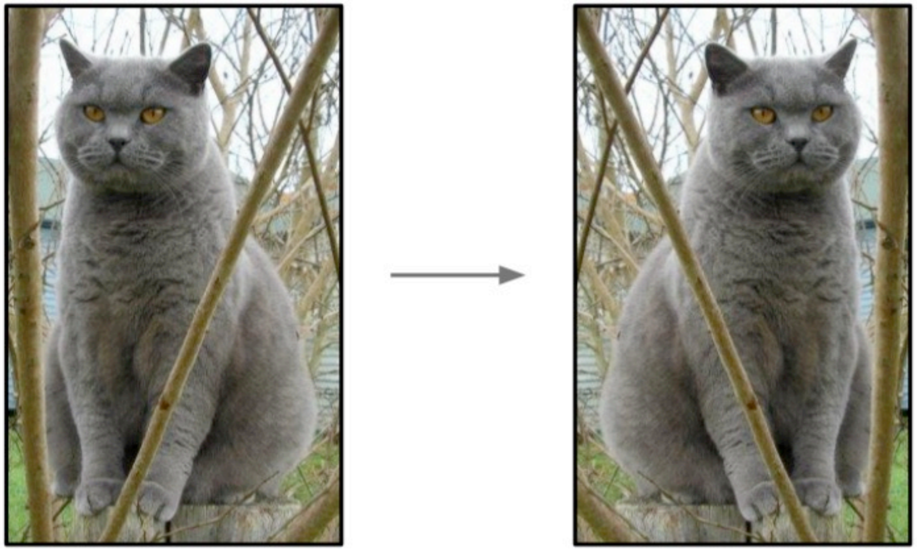
- Data augmentation
- Batch normalization
- Dropout
- Transfer learning and finetuning
- Lab Assignments

Data augmentation

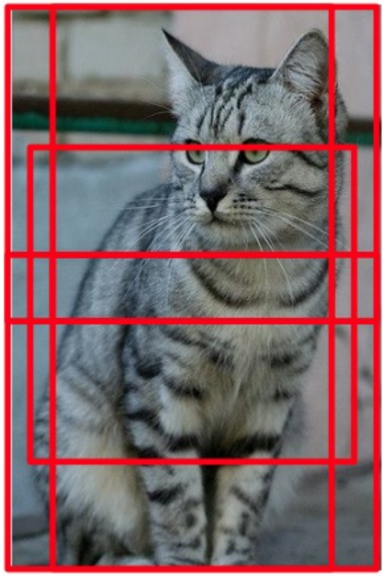
Data augmentation

- Apply some transformations → increase the number of training samples
- Applied to training set, NOT to validation/test sets

- **Flipping**



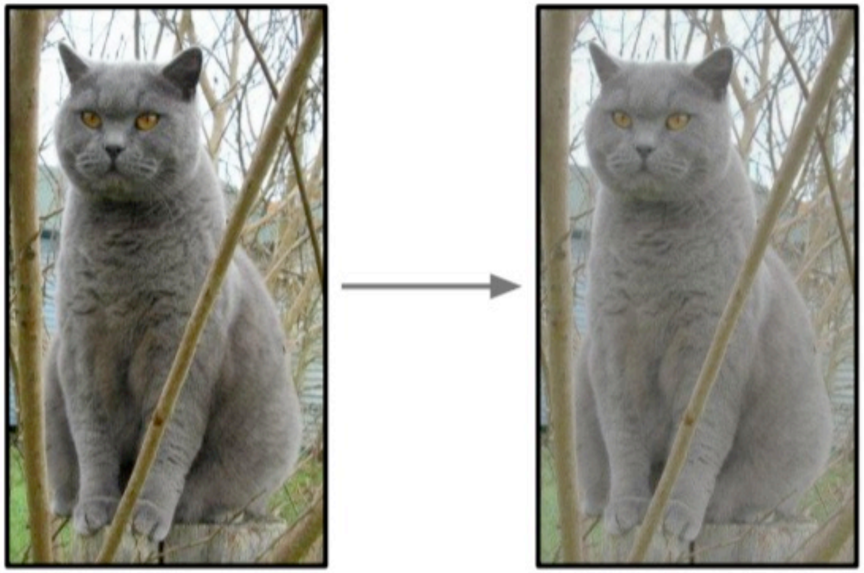
- **Cropping**



Data augmentation

- Apply some transformations → increase the number of training samples
- Applied to training set, NOT to validation/test sets

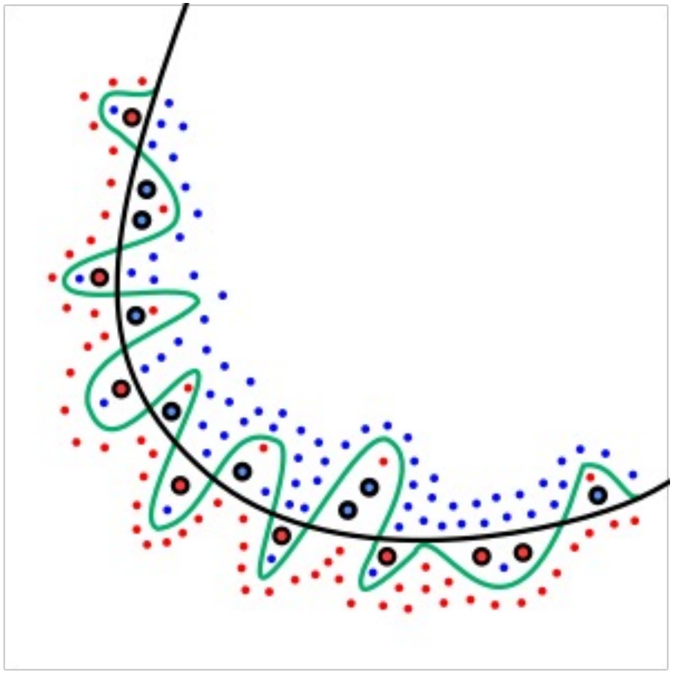
- **Color jittering**



- **Translation, rotation, rescaling,...**

Data augmentation

- Help to enhance the model ability to generalize to unseen data by exposing it to more diverse and varied training samples
- Act as a *regularization technique* → **avoid model overfitting** (model learns the training set to well but poorly on unseen data (test data)



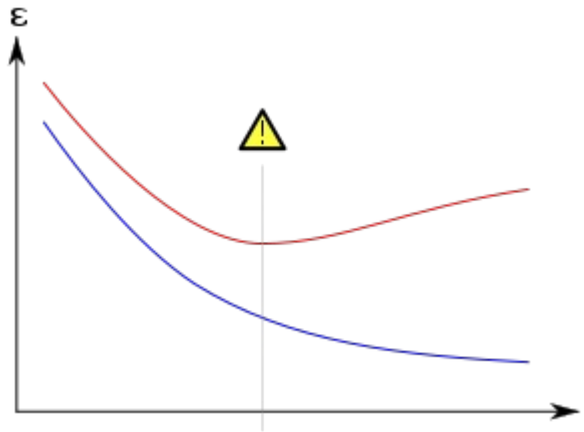


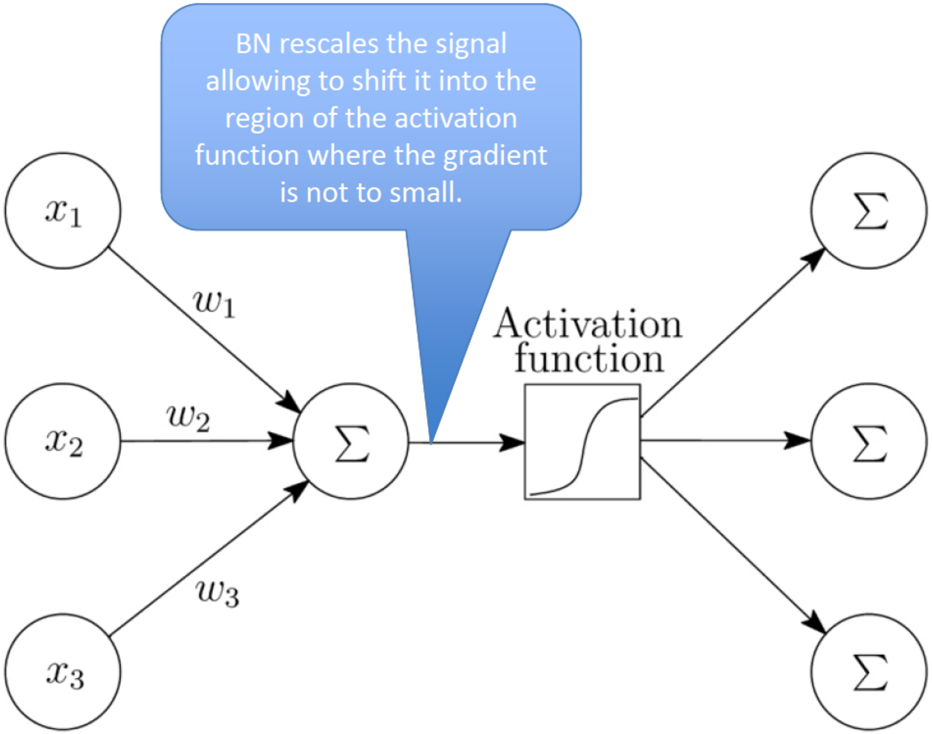
Image credits: wikipedia

Batch normalization

Batch normalization

Batch normalization

- Normalize the input of each layer by adjusting and scaling the activations
- Use to mean and standard-deviation of the mini-batch for the normalization
- **Used in most NN/CNN architectures**

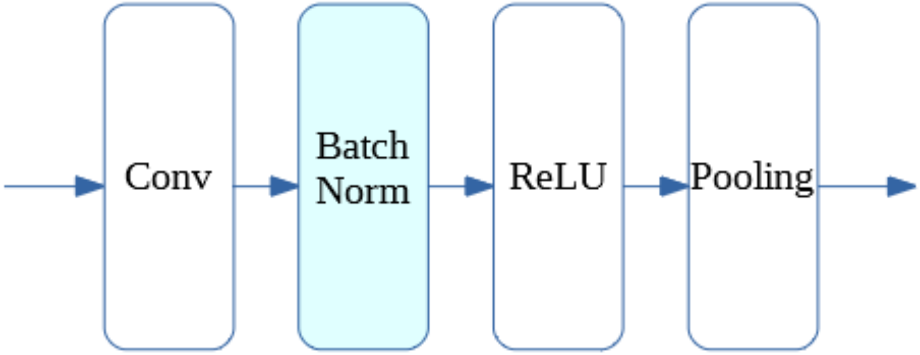


Batch normalization

BatchNorm (BN)

- Perform BN before the activation function (ReLu)
- Reduce a strong dependence on initialization
- Improve the gradient flow through the network

Example: ...-[Conv]-[BN]-[ReLu]-[Pool]...-[FC]-[BN]-[Relu]-...



Batch normalization

How to apply in pytorch ?

```
import torch
import torch.nn as nn

# Define a simple neural network
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(784, 256)  # Input size: 784, Output size: 256
        self.bn1 = nn.BatchNorm1d(256)  # Batch normalization after first fully conne
        self.fc2 = nn.Linear(256, 10)  # Input size: 256, Output size: 10 (for class:

    def forward(self, x):
        x = torch.relu(self.bn1(self.fc1(x)))  # Applying batch normalization after :
        x = self.fc2(x)
        return x

# Instantiate the neural network
model = NeuralNetwork()
```

Batch normalization

How to apply in pytorch ?

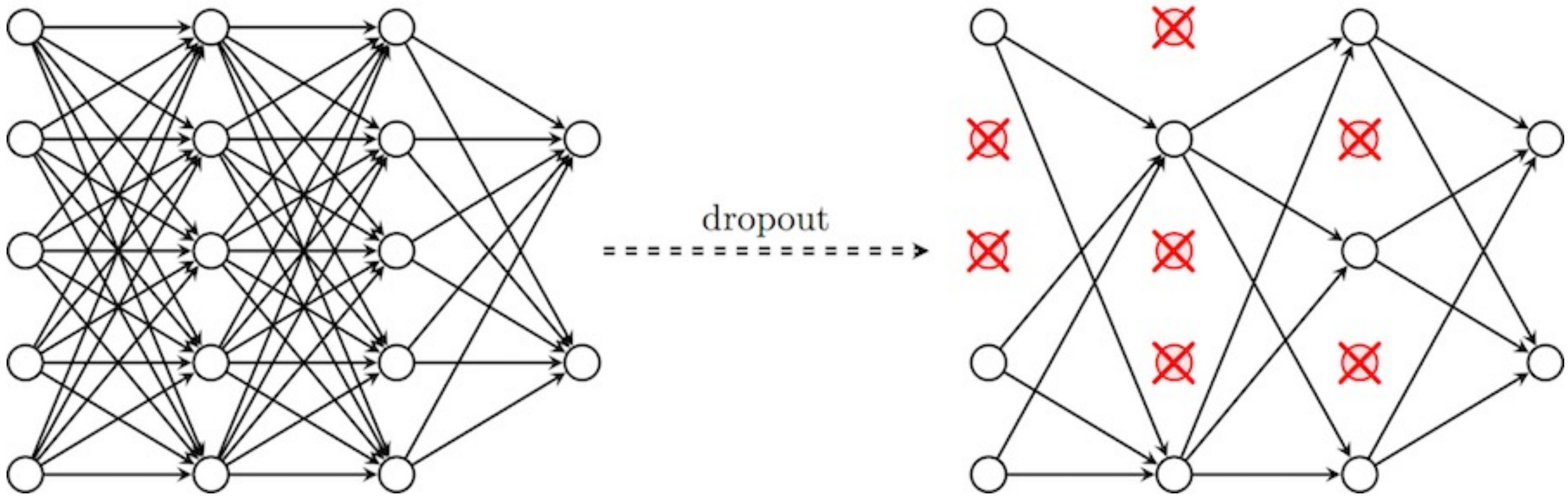
```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=
        self.bn1 = nn.BatchNorm2d(16)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride
        self.bn2 = nn.BatchNorm2d(32)
        self.fc = nn.Linear(32 * 8 * 8, 10)  # Assuming input image size of 32x32 and

    def forward(self, x):
        x = self.relu(self.bn1(self.conv1(x)))
        x = self.maxpool(x)
        x = self.relu(self.bn2(self.conv2(x)))
        x = self.maxpool(x)
        x = x.view(x.size(0), -1)  # Flatten the tensor for fully connected layers
        x = self.fc(x)
        return x
```

Dropout

Dropout

Dropout

- A regularization technique
- **Randomly remove** some neuron connections (by setting a probability 0.5 for example)
- Avoid overfitting



Dropout

How to apply in pytorch ?

```
class CNNWithDropout(nn.Module):
    def __init__(self):
        super(CNNWithDropout, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=
        self.relu = nn.ReLU()
        self.dropout1 = nn.Dropout(p=0.2)  # Apply dropout with a probability of 0.2
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride
        self.dropout2 = nn.Dropout(p=0.2)  # Apply dropout with a probability of 0.2
        self.fc = nn.Linear(32 * 8 * 8, 10)  # Assuming input image size of 32x32 and

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.dropout1(x)
        x = self.maxpool(x)
        x = self.relu(self.conv2(x))
        x = self.dropout2(x)
        x = self.maxpool(x)
        x = x.view(x.size(0), -1)  # Flatten the tensor for fully connected layers
        x = self.fc(x)
        return x
```

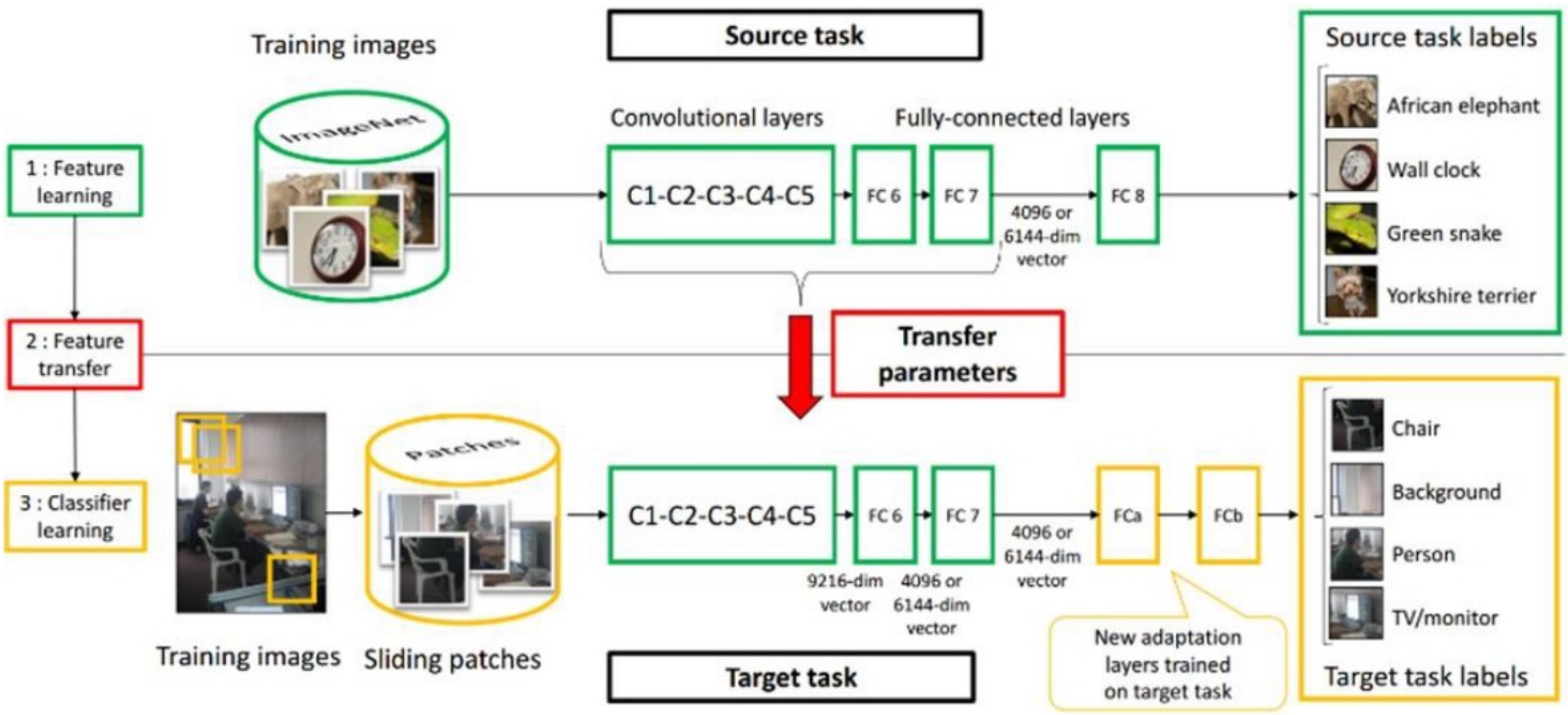
Remarks

Data augmentation, batch normalization and dropout techniques do not add more parameters to a network

Transfer learning and finetuning

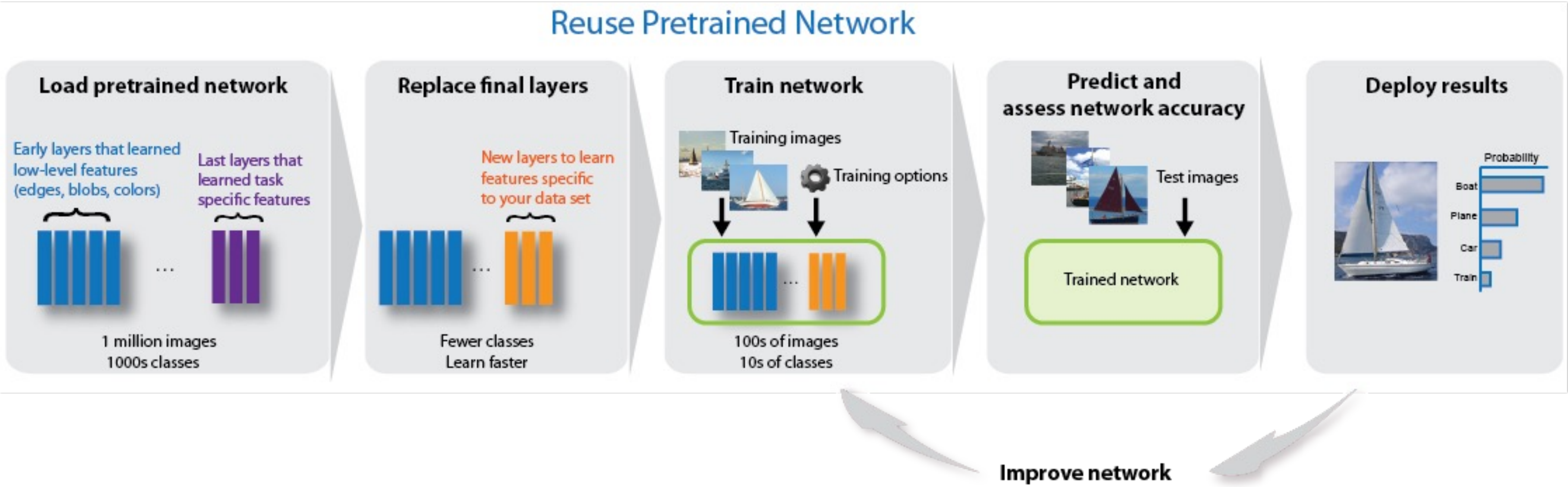
Transfer learning

- Learn a new task (target) through the **transfer of knowledge** from a related task (source) which has been trained.
- Use weights **from pre-trained CNN** as initialization to train new network

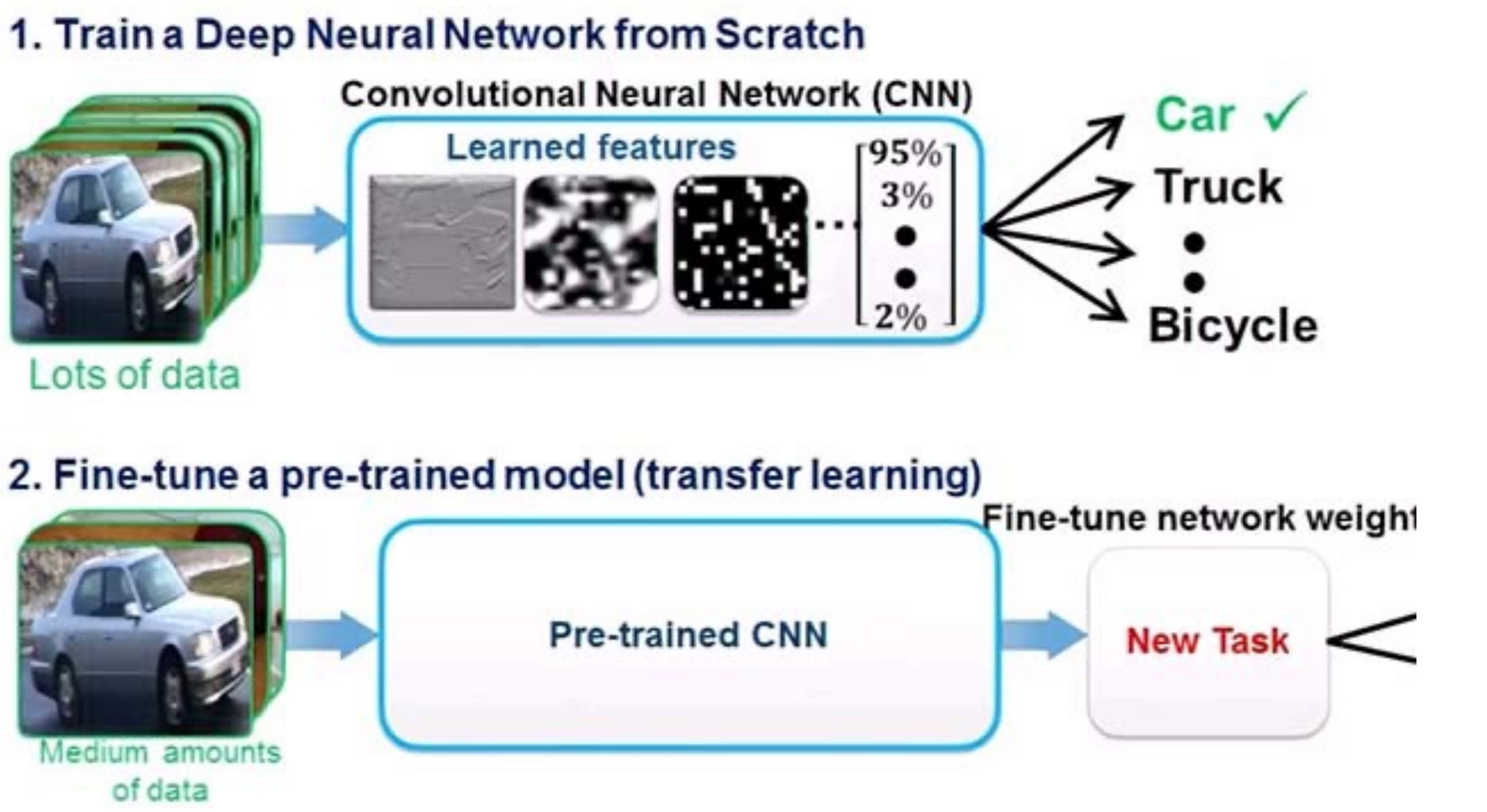


Finetuning

- **Fine-tuning:** replace and retrain some last layers on top of the CNN using new dataset
- Fine-tune a network with transfer learning is much faster and easier than **training from scratch** (construct and train a new network)
- Relevant when <u>using small number of trained samples</u>



Finetuning

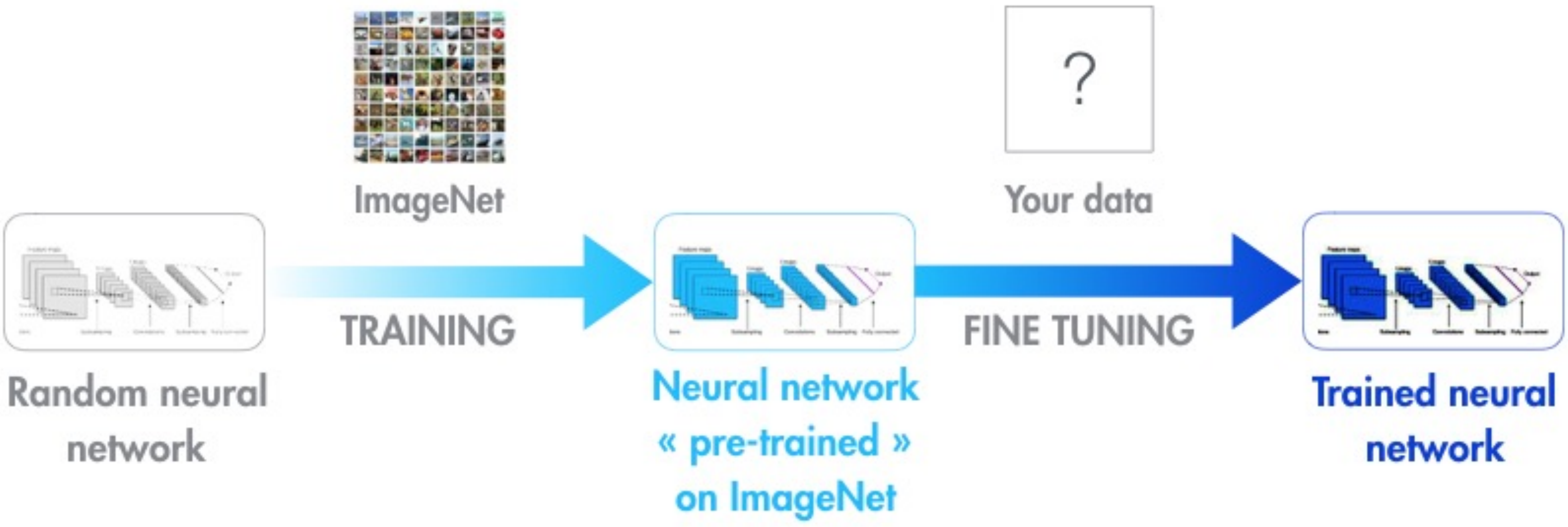


1. Train a Deep Neural Network from Scratch
Convolutional Neural Network (CNN)
Learned features
95%
3%
2%
Car ✓
Truck
Bicycle
Lots of data
2. Fine-tune a pre-trained model (transfer learning)
Fine-tune network weight
Pre-trained CNN
New Task
Medium amounts of data

In practice

It is common to :

- pretrain a CNN on a very large dataset **(ImageNet)**
- use the pre-trained network to ne-tune the new task

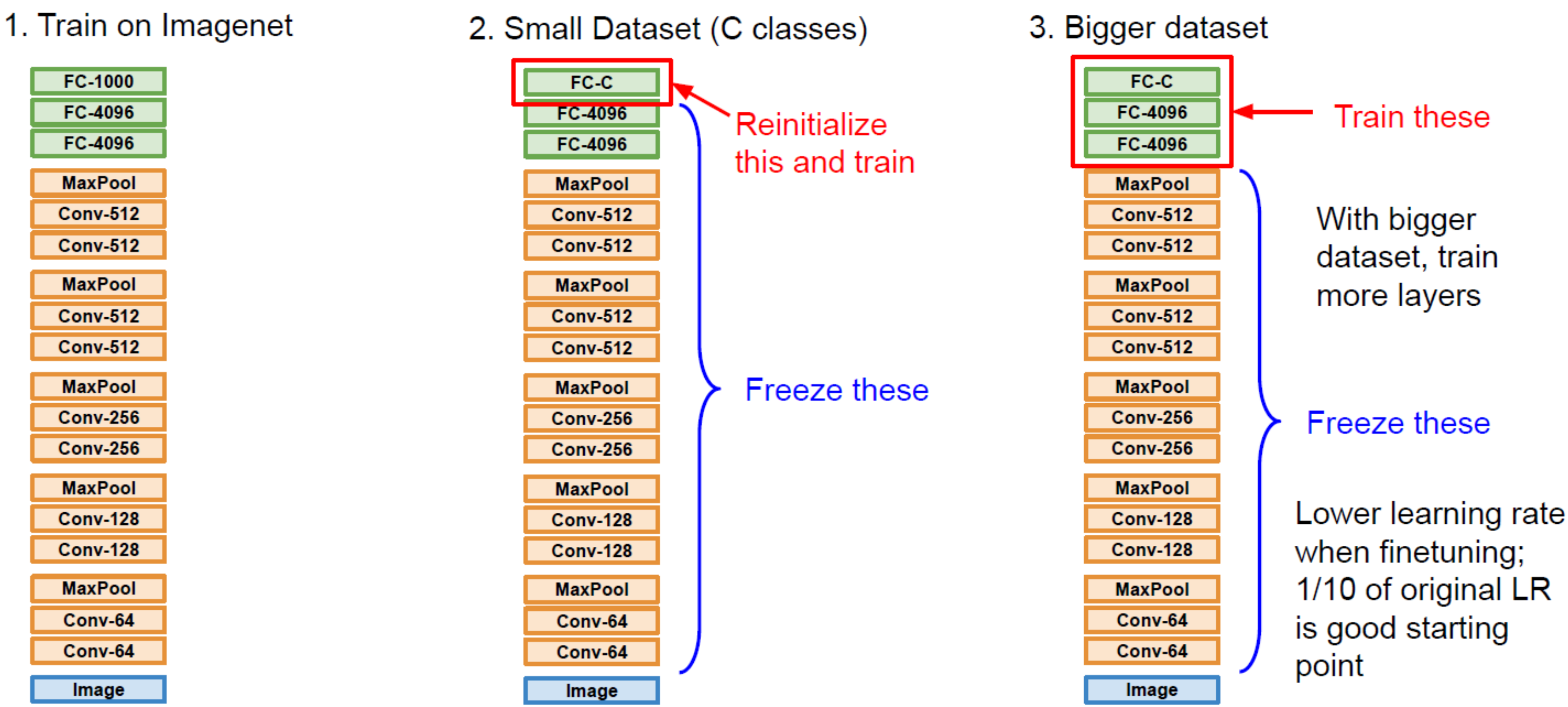


In practice

ImageNet dataset :

- 1000 object classes
- 1.2M trained images
- 100k test images

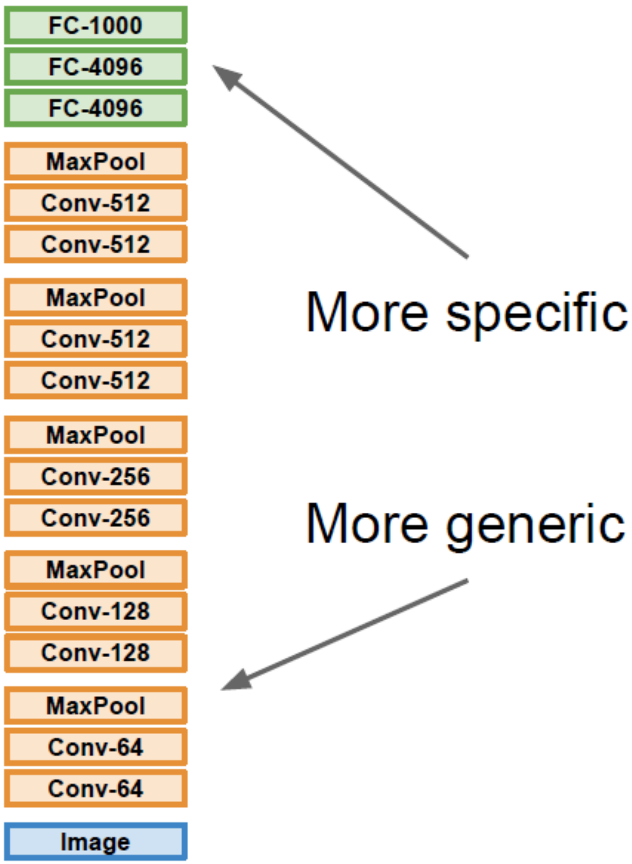
In practice



Use CNN backbone as feature extractor

Finetuning few layers or all the network

In practice



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

Pytorch pretrained models

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MaxVit
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

https://pytorch.org/vision/stable/models.html

Finetuning with pytorch

Finetuning the convnet

Load a pretrained model and reset final fully connected layer.

```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to nn.Linear(num_ftrs, len(class_names)).
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

Using CNN as feature extractor

ConvNet as fixed feature extractor

Here, we need to freeze all the network except the final layer. We need to set `requires_grad = False` to freeze the parameters so that the gradients are not computed in `backward()`.

You can read more about this in the documentation here.

```
model_conv = torchvision.models.resnet18(weights='IMAGENET1K_V1')
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()
```

Lab assignment

<u>Practical lab:</u> Comparing training from scratch vs transfer learning

- Based on the **Lab2**, create a jupyter notebook file named **R4A13_lab3.ipynb**

- Based on this tutorial (*https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html*), your task will be to compare 3 approaches: <u>training from scratch</u> (as done in **Lab2**), <u>CNN-based feature extraction</u> and <u>finetuning</u>, on the **CIFAR-10 dataset**

- Perform experiments using 2 different networks among the following: *SqueezeNet, AlexNet, VGG and ResNet*

- At the end you should create a comparative table as follows (same number of epochs for training)

Model	Number of trainable params	Test accuracy	Training time per epoch
From scratch			
CNN1 (feat_extract)			
CNN1 (finetuning)			
CNN2 (feat_extract)			
CNN2 (finetuning)			

- Note that **you need to submit your compiled notebook with all outputs**

References and Sources

- **Convolutional neural networks for visual recognition** - Stanford

  https://cs231n.github.io/

- **Neural networks and deep learning** (free online book)

  http://neuralnetworksanddeeplearning.com/index.html

- **Machine learning course** – Oxford

  https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/

- **Neural network course** - Hugo Larochelle

  https://info.usherbrooke.ca/hlarochelle/neural_networks/content.html

- **Deep learning course** – François Fleuret

  https://fleuret.org/dlc/