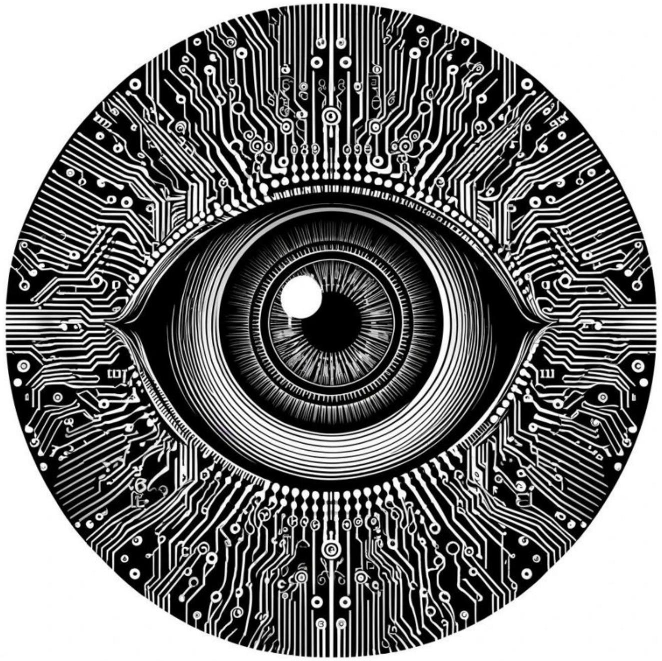
Transfer Learning and Fine-Tuning Pretrained Models



Antonio Rueda-Toicen

SPONSORED BY THE
Federal Ministry
of Education
and Research

Learning goals

- Understand the use of transfer learning in image classification
- Modify an Imagenet-pretrained network to perform a new classification task
- Perform fine-tuning of a Resnet50 after modifying its architecture

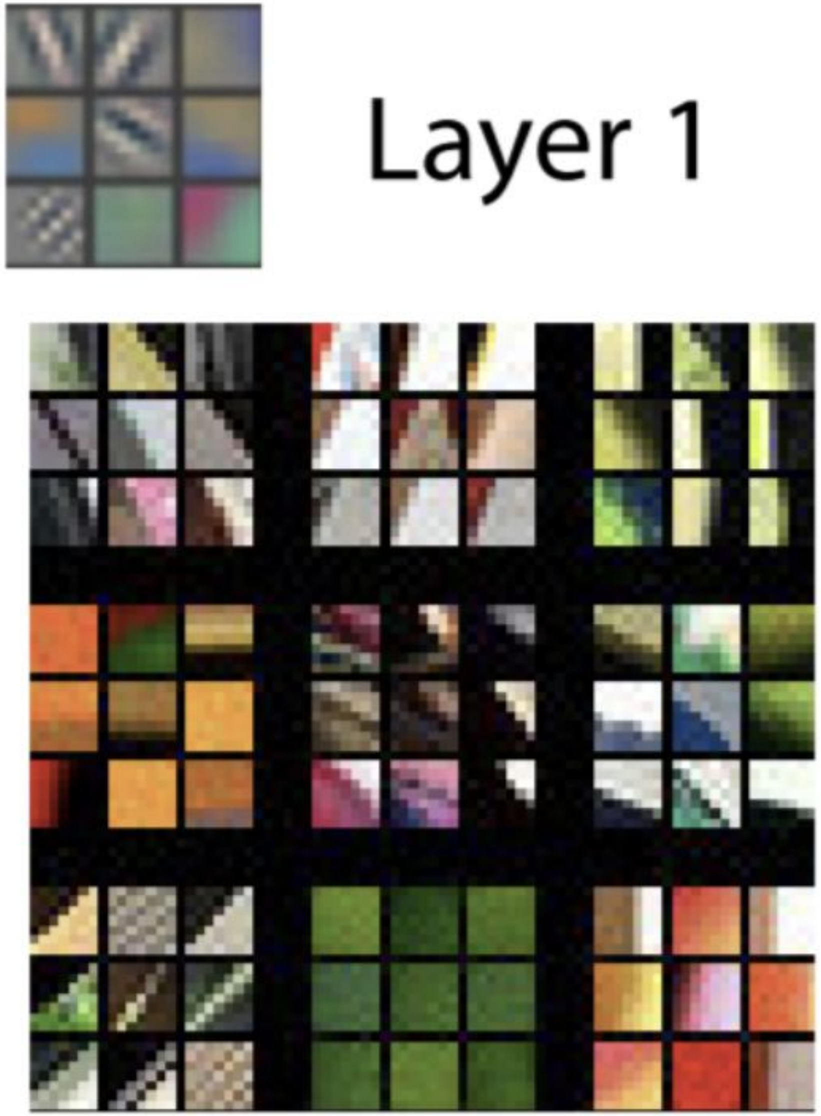
The ImageNet-1K dataset

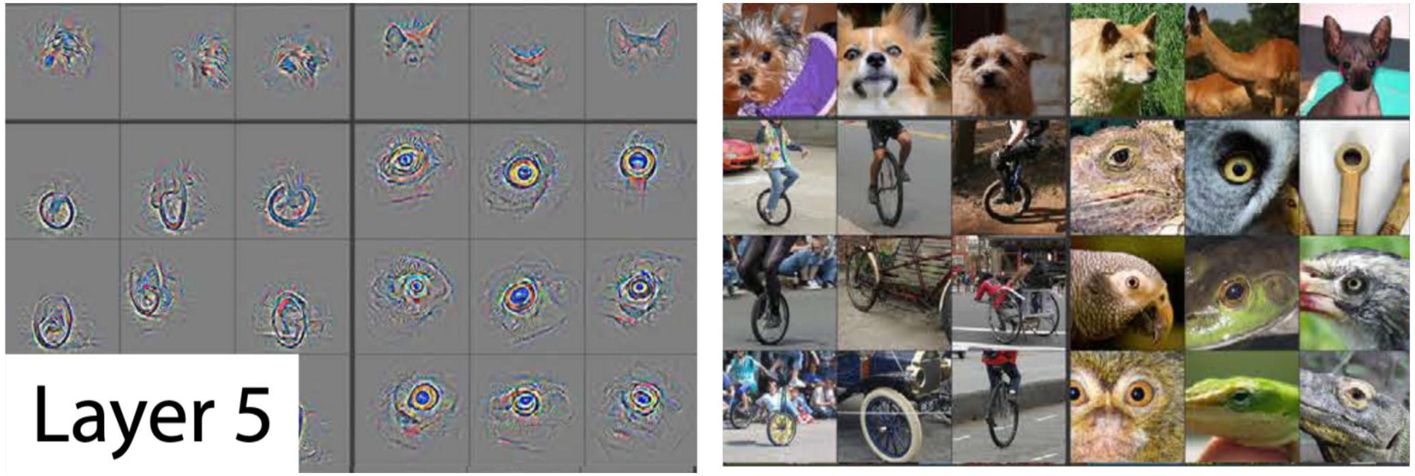
Explore Imagenet classes on this notebook

```
Index 68:  sidewinder
Index 85:  quail
Index 122: American lobster
Index 216: clumber
Index 273: dingo
Index 302: ground beetle
Index 403: aircraft carrier
Index 420: banjo
Index 775: sarong
Index 946: cardoon
```

Image source: https://cs.stanford.edu/people/karpathy/cnnembed/

What an Imagenet-pretrained model already knows

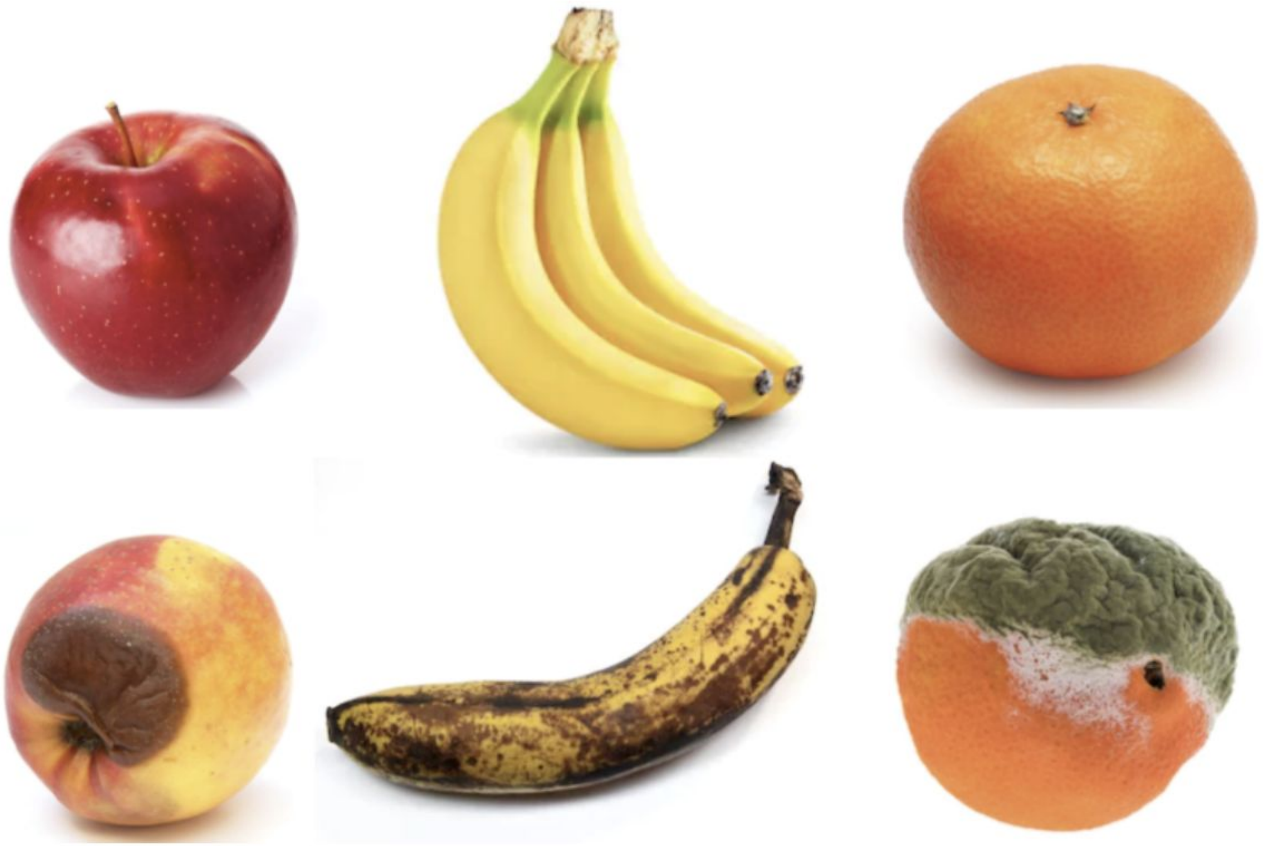




Images from [visualizing and understanding convolutional networks]

We can transfer these learned features to a new classification task

Fresh vs rotten fruit classification



```
ImageNet index for 'Granny Smith': 948
ImageNet index for 'banana': 954
ImageNet index for 'orange': 950
```

Imagenet has no training data for the rotten variants of fruit

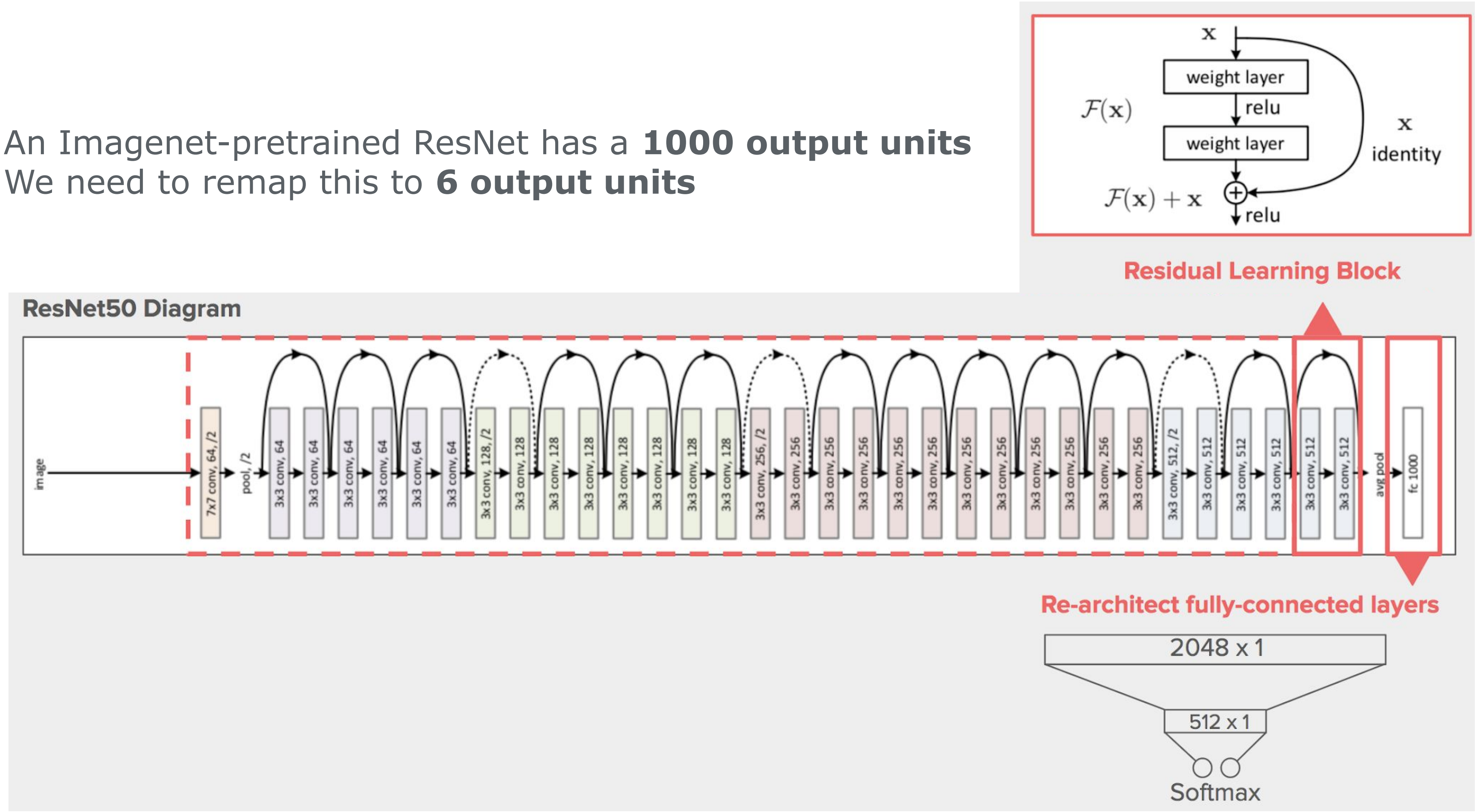
Inference on out-of-training-distribution data

```
pomegranate, probability = 0.50
buckeye, horse chestnut, conker, probability = 0.21
fig, probability = 0.12
bagel, beigel, probability = 0.03
French loaf, probability = 0.02
```

```
face powder, probability = 0.12
golf ball, probability = 0.08
croquet ball, probability = 0.08
baseball, probability = 0.07
ping-pong ball, probability = 0.07
```

Changing an ImageNet-pretrained Resnet50

An Imagenet-pretrained ResNet has a **1000 output units**
We need to remap this to **6 output units**



Using the original image transformations

```python
import torch
import torchvision.models as models
from torchvision.models import ResNet50_Weights

# Create an instance of ResNet50_Weights pretrained on
# ImageNet-1k
weights = ResNet50_Weights.IMAGENET1K_V2

# Load pretrained ResNet50 with specified weights
model = models.resnet50(weights=weights)

# Access the original data transformations used for
training
original_transformation = weights.transforms()
original_transformation
```

```
ImageClassification(
    crop_size=[224]
    resize_size=[232]
    mean=[0.485, 0.456, 0.406]
    std=[0.229, 0.224, 0.225]
    interpolation=InterpolationMode.BILINEAR
)
```

Step 1: inspect the final layers of the model

```
import torch
import torchvision.models as models
from torchvision.models import ResNet50_Weights

# Create an instance of ResNet50_Weights pretrained on ImageNet-1k
weights = ResNet50_Weights.IMAGENET1K_V2

# Load pretrained ResNet50 with specified weights
model = models.resnet50(weights=weights)

# This will print out the architecture, including names of layers
model
```

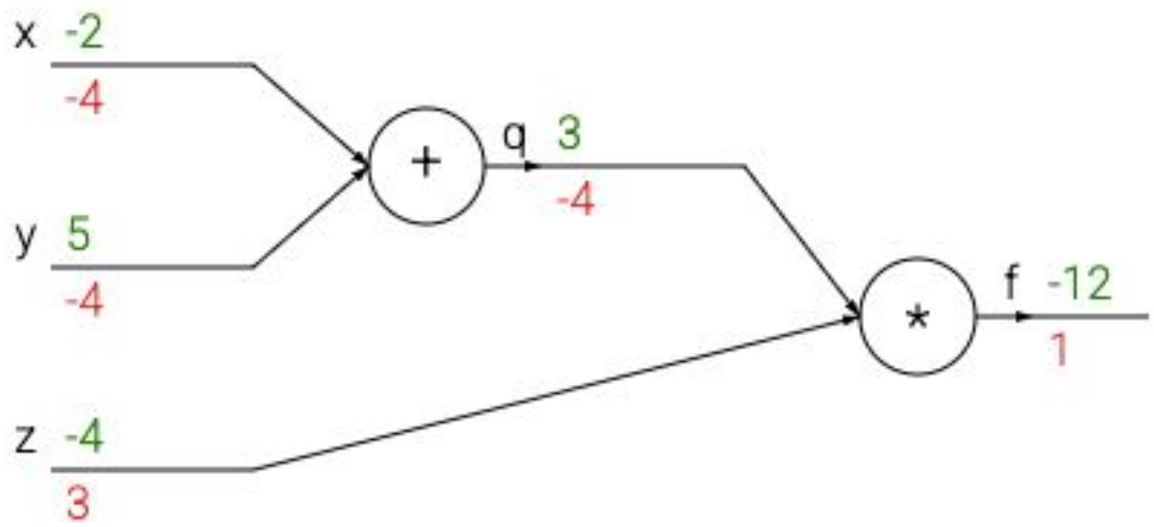
```
(fc): Linear(in_features=2048, out_features=1000, bias=True)
```

Step 2: freeze the base model

Image from https://cs231n.github.io/optimization-2/

```
# Freeze the base model
for param in model.parameters():
    param.requires_grad = False

# Optionally, unfreeze specific layers
for param in model.fc.parameters():
# Example: unfreeze the fully connected layer
   param.requires_grad = True
```



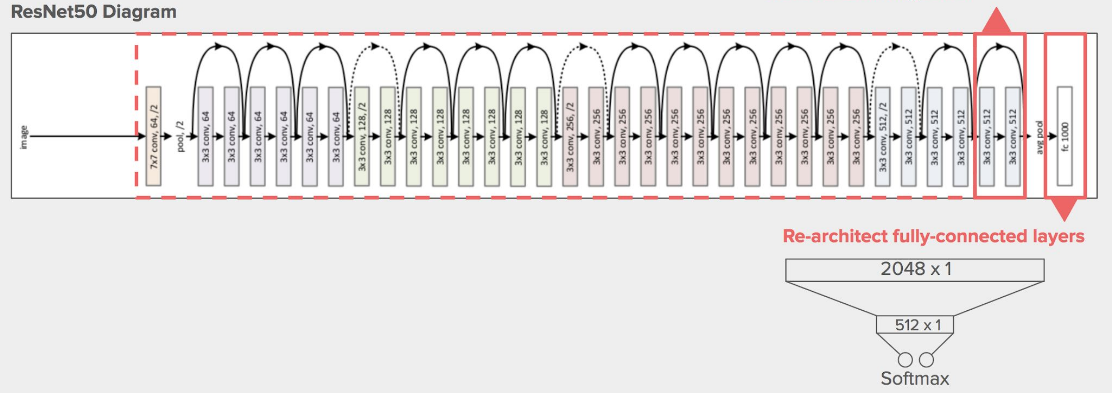
The gradient computation graph is **only** computed on layers with `requires_grad=True`
Only these weights are updated during training

Step 3: add extra layers with the new target

```
num_new_classes = 6

# Approach 1: Add layer after classifier
def add_layer_model():
    model = resnet50(weights=weights)
    model.fc = nn.Sequential(
        model.fc,
        nn.ReLU(),
        nn.Linear(1000, num_new_classes)
    )
    return model

# Approach 2: Replace classifier
def replace_layer_model():
    model = resnet50(weights=weights)
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, num_new_classes)
    return model
```

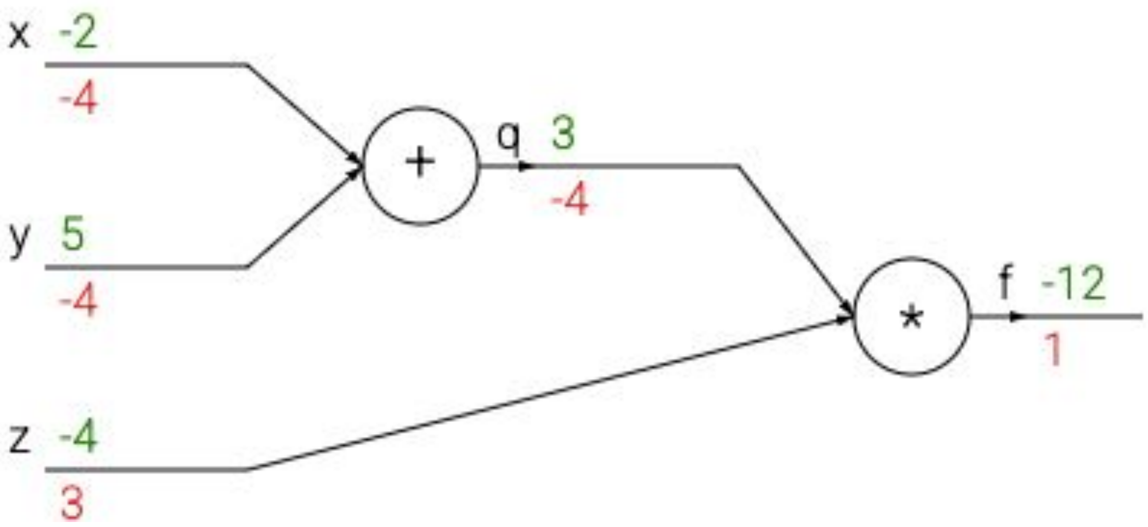


Fine-tuning by unfreezing the convolutional base

```
# Unfreeze specific layers
for param in model.fc.parameters():
# Example: unfreeze the fully connected layer
    param.requires_grad = True

… # train for some epochs,
  # acquire low validation loss

# Unfreeze the whole model
# to perform fine tuning
model.requires_grad_(True)
```



- After updating the final layers with non-random weights, we unfreeze the whole network and train it again
- This two stage process avoids "**catastrophic forgetting**"

Summary

Trained neural networks have hierarchical knowledge

- They learn basic shapes on their first layers and complex patterns on their last layers
- Models pretrained on rich datasets like ImageNet learn universal visual patterns that are useful for many tasks

Model Adaptation

- Modifying network architecture requires strategic choices
- We can add new final layers to preserve more ImageNet knowledge or replace the original final layers for a different representation

Progressive training prevents “catastrophic forgetting”

- Start by freezing the base model, train new layers, then fine-tune all layers for optimal results

SPONSORED BY THE
Federal Ministry
of Education
and Research

KI Service Zentrum by Hasso-Plattner-Institut HPI

Further reading and references

ImageNet

- https://en.wikipedia.org/wiki/ImageNet

Visualizing and understanding convolutional networks

- https://arxiv.org/pdf/1311.2901

Resnet50 in torchvision models

- https://pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html

SPONSORED BY THE

Federal Ministry of Education and Research