Machine learning for vision and multimedia
(01URPOV)

Lab 04 – Advanced Pytorch
Francesco Manigrasso

2025 – 2026

POLITECNICO DI TORINO

Learning objectives

Learn how to train more complex and effective models by exploiting

- Transfer learning
- Data augmentation
- Non-sequential models

TRANSFER LEARNING

Transfer learning

- Transfer learning is an enabler in computer vision as it allows to reuse knowledge on different tasks
- Formally, a task $T = \{\mathcal{Y}, f\}$ is defined as (learning a) mapping function f between an input space $\mathcal{X}$ and a label space $\mathcal{Y}$
- In a fully supervised setting, the task is learnt from a series of labelled examples (X, Y)
- Labels may come in several forms, e.g.:
  - class labels (e.g. "cat", "dog", "house") → classification task
  - object bounding boxes → object detection task
  - pixel-wise class label → image segmentation task
  - depth map → depth estimation task, etc….

Transfer learning

- When does transfer learning make sense?
- Transfer from Task A to Task B if
  - Task A and B have the same type of input (e.g., images)
  - You have a lot more data for Task A than Task B
  - Low level features from Task A could be helpful for learning Task B
- Question
  - Good examples of Task A in computer vision?

Transfer learning

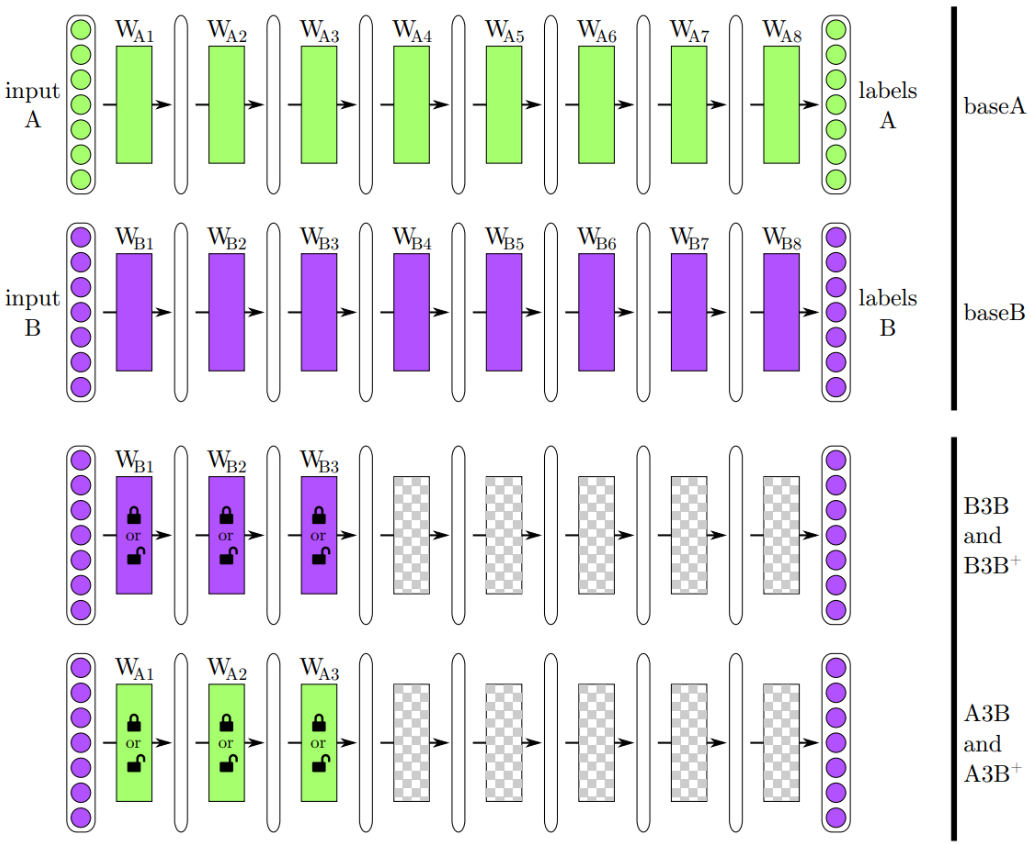
- When transfer learning makes sense?
- Transfer from Task A to Task B if
  - Task A and B have the same type of input (e.g., images)
  - You have a lot more data for Task A than Task B
  - Low level features from Task A could be helpful for learning Task B

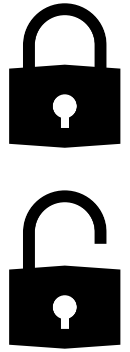
Scenario 1:
Transfer across classification tasks
- Different labels
- Different input distribution

Scenario 2:
Transfer across different types of tasks, e.g. from classification to object detection

Feature transferability

ImageNet split into two groups (A and B)





standard supervised setting

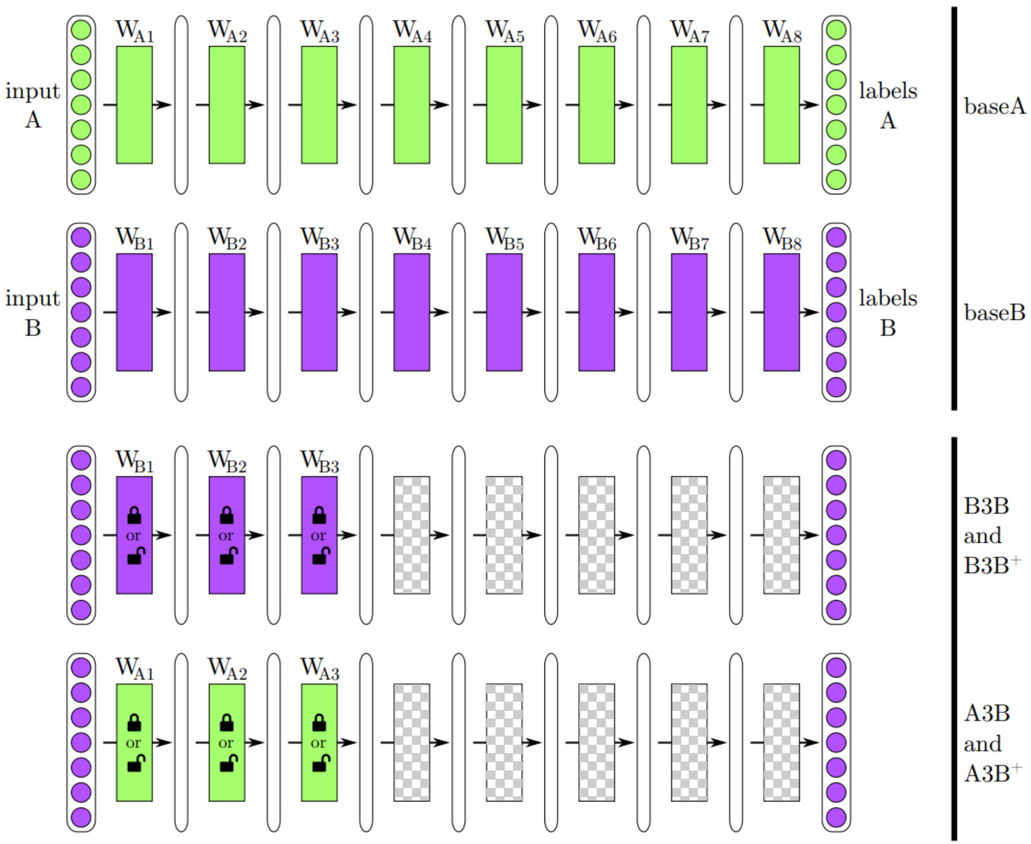
B → B (control)

A → B (transfer learning)

[Yosinski, How transferable are features in deep neural networks? , 2016]

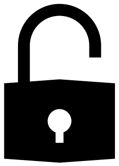
Feature transferability

ImageNet split into two groups (A and B)



standard supervised setting

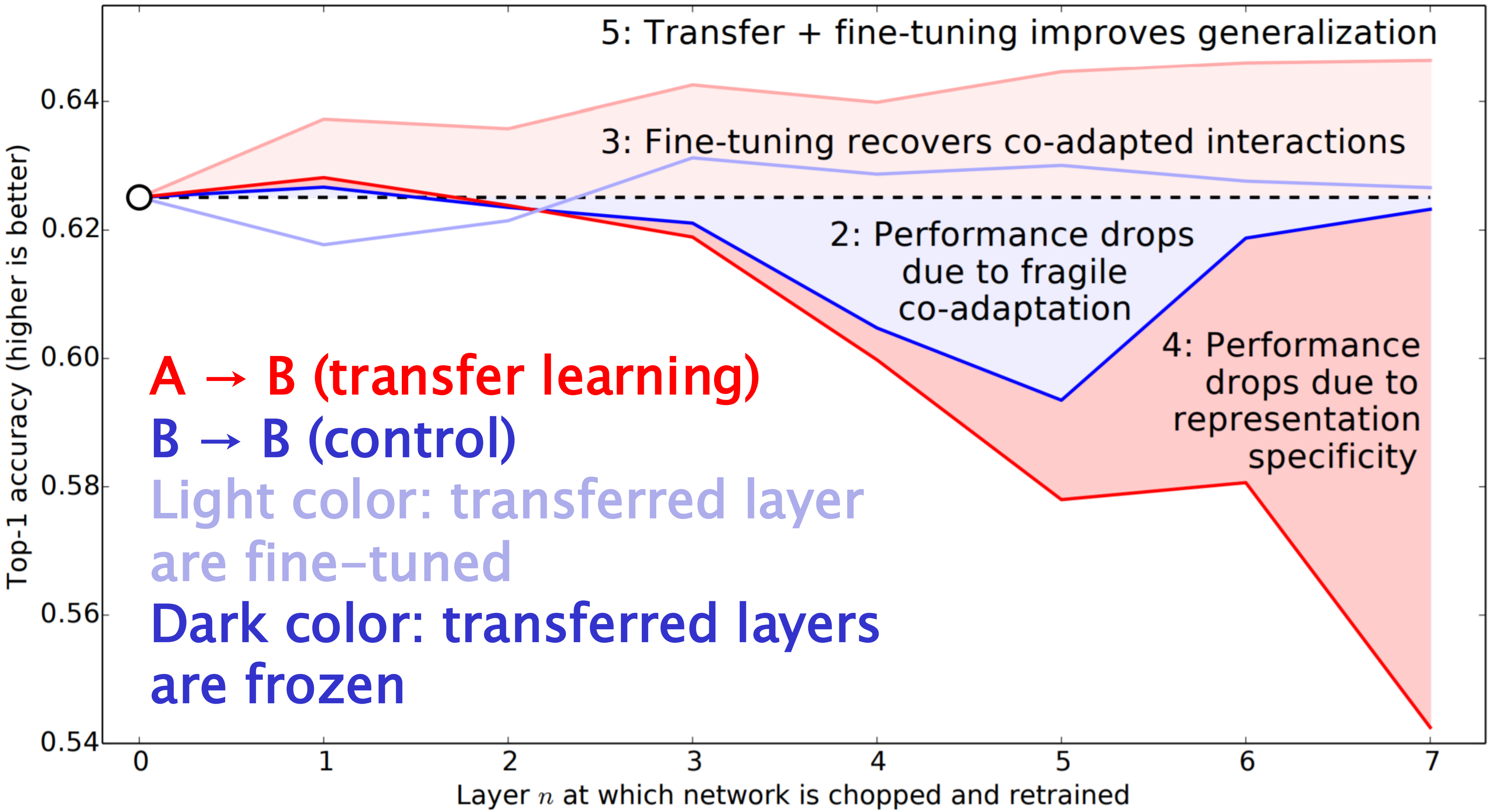
frozen



not frozen

[Yosinski, How transferable are features in deep neural networks? , 2016]

Feature transferability

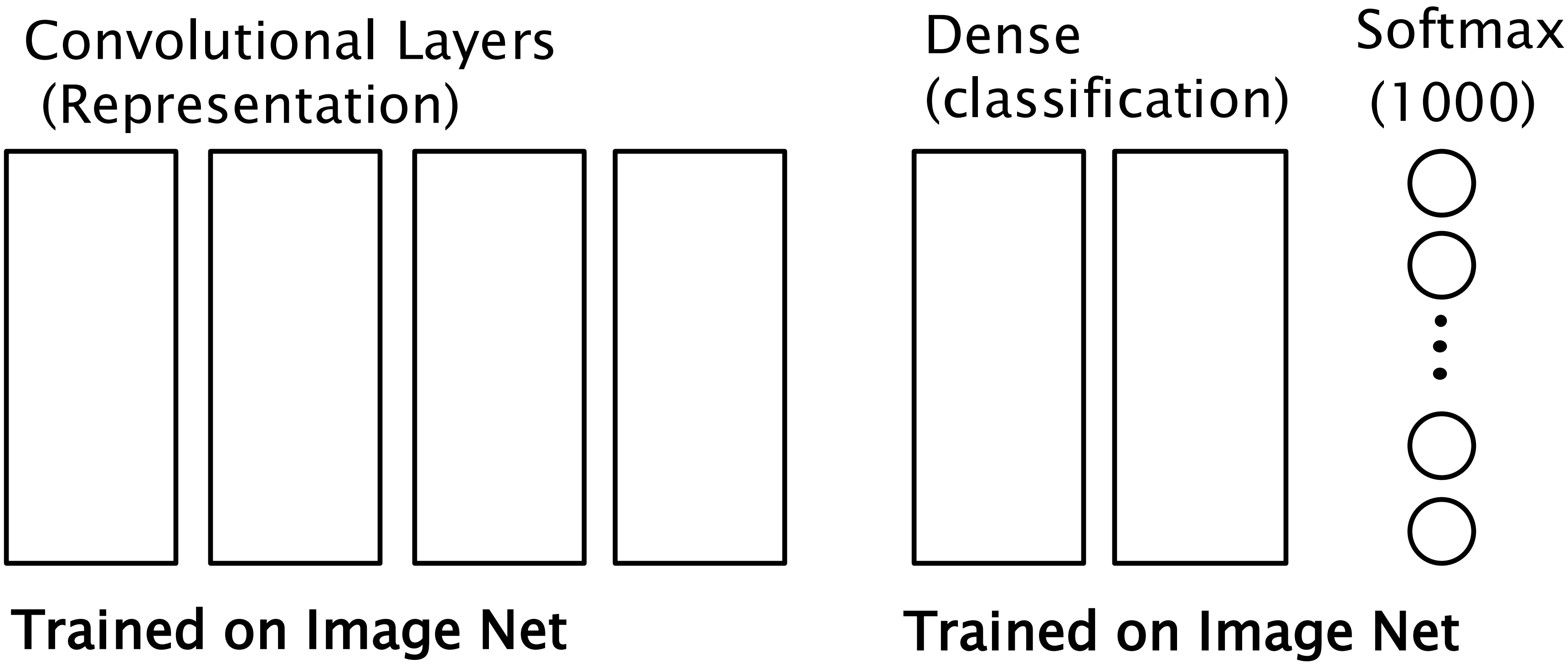


[Yosinski, How transferable are features in deep neural networks? , 2016]

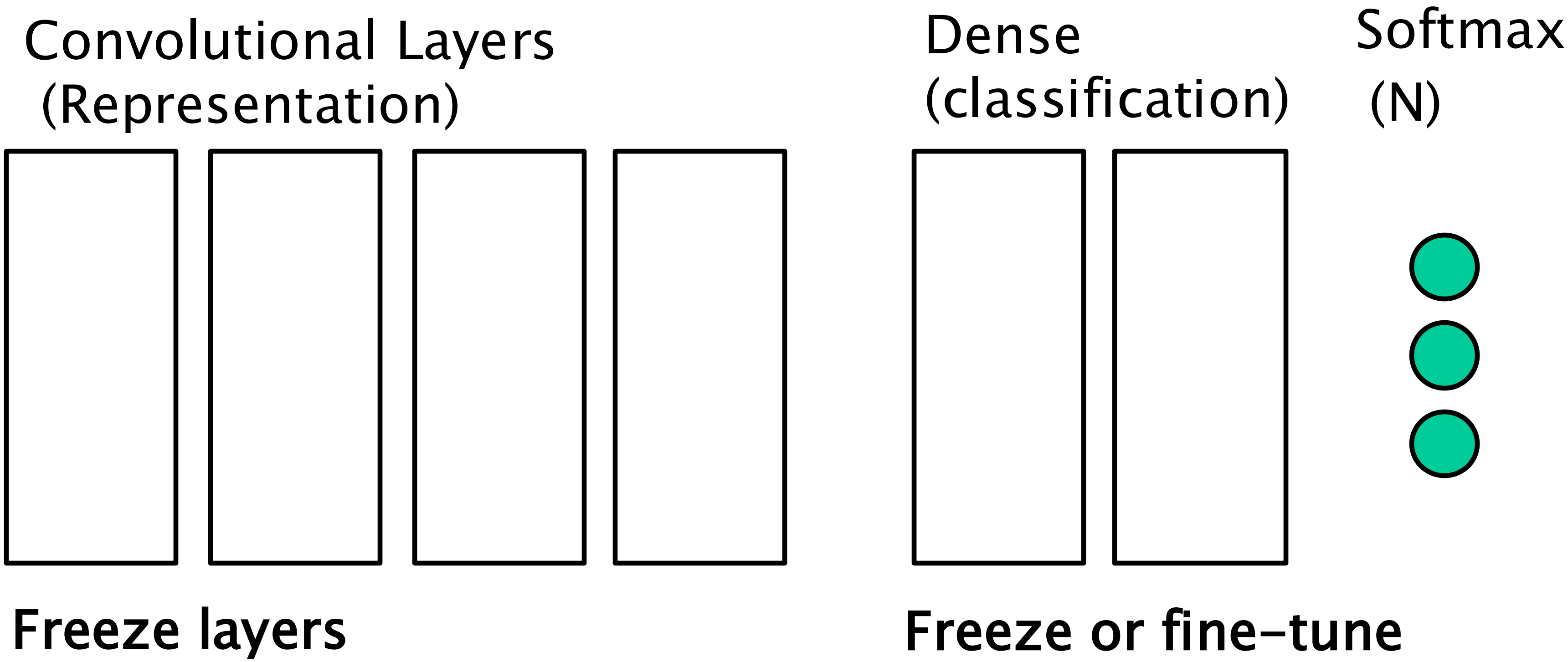
Transfer strategies

- There are two fundamental transfer strategies:
- Strategy A: Use the pre-trained network as fixed **feature extraction**, and train a classifier on those features (not necessarily a neural network)
- Strategy B: **Finetune the network** trained on the source task by replacing the final layers and selectively retrain some (all) of the previous layers
- The optimal strategy depends on:
  - the amount of labelled samples
  - the similarities between the source and target task

Base model


Convolutional Layers
(Representation)
Dense
(classification)
Softmax
(1000)
Trained on Image Net
Trained on Image Net

Transfer learning strategy A

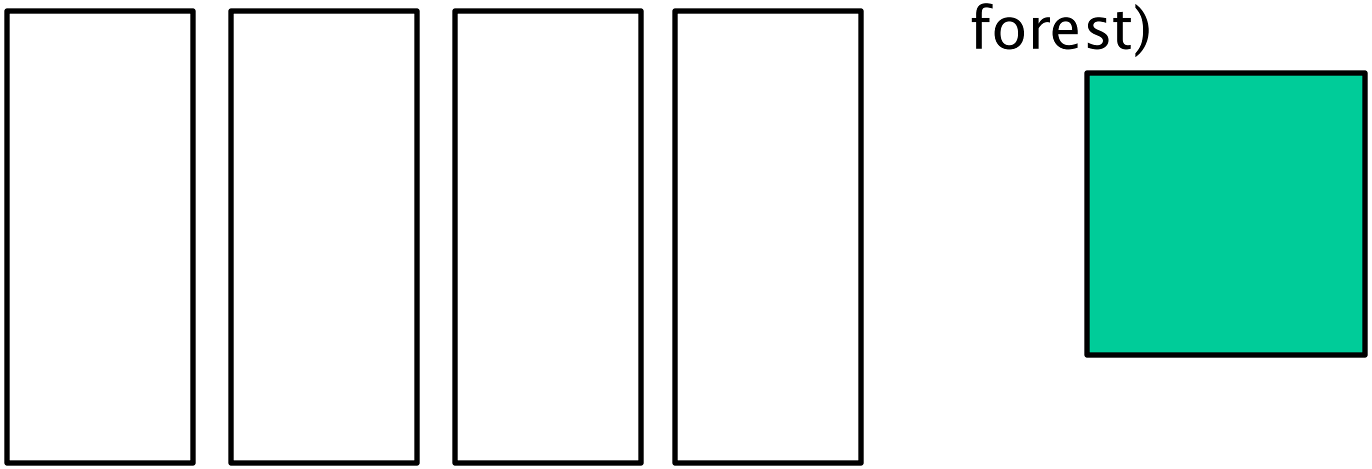


Small – medium dataset
Similar to original data
Input size unchanged

Transfer learning strategy A

Convolutional Layers
(Representation)

Non-neural classifier
(linear, SVM, random
forest)



Freeze layers

Train on new dataset

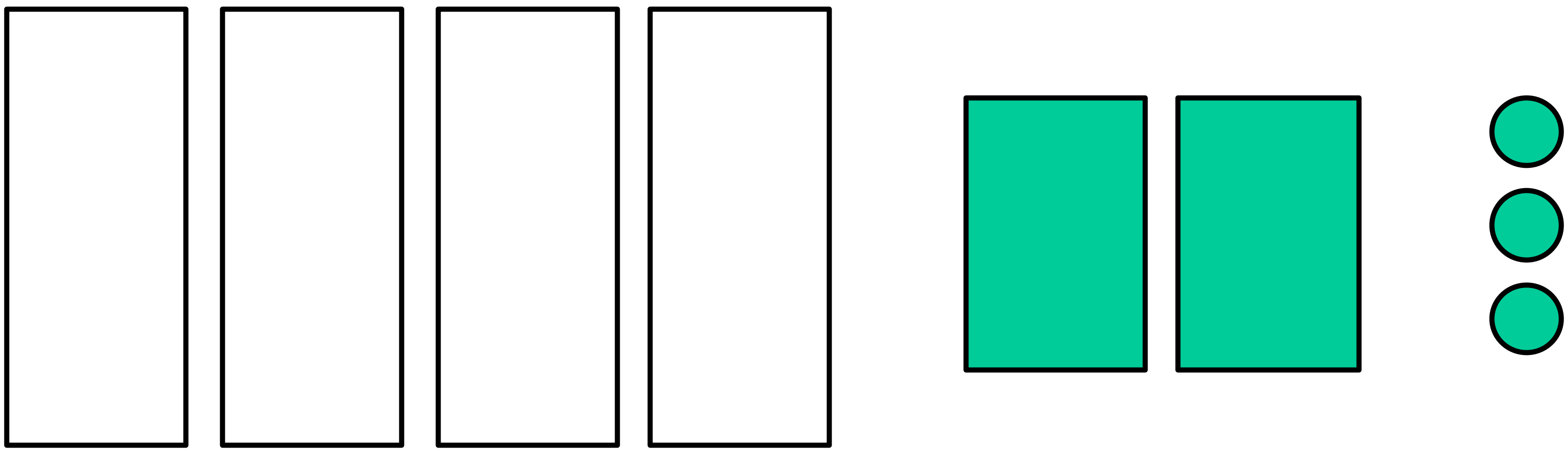
Small data
Similar to original data
Input size can change

Transfer learning strategy B

Convolutional Layers
(Representation)

Dense
(classification)

Softmax
(N)



Freeze layers

Small – medium dataset
Similar to original data
Input size can change

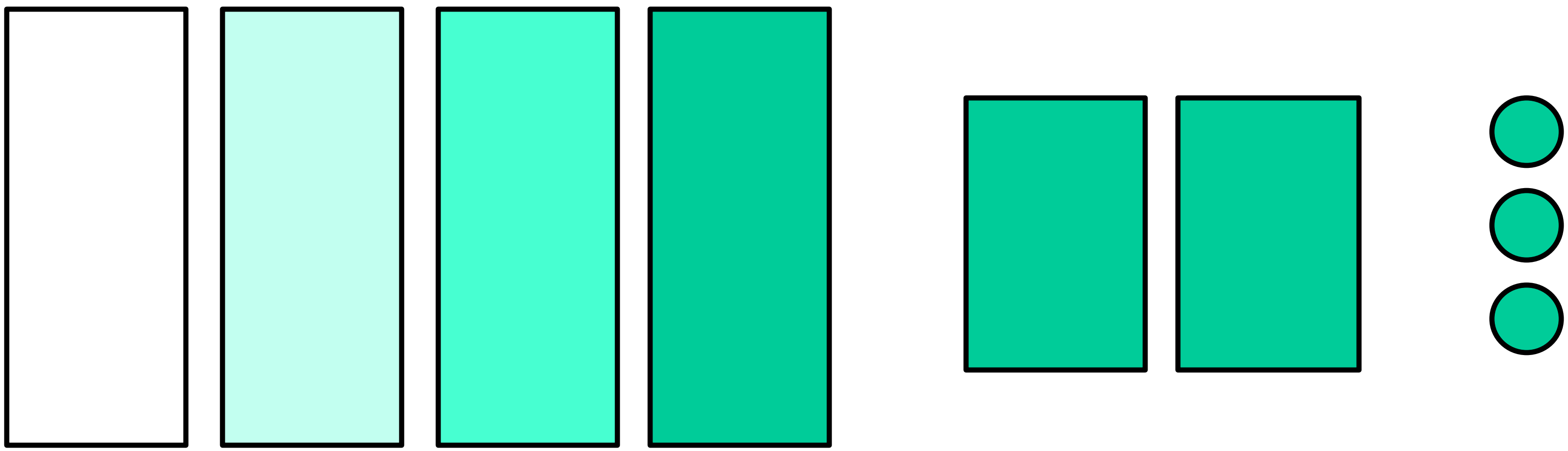
Replace layers
Randomly initialize +
Train on new dataset

Transfer learning strategy B

Convolutional Layers
(Representation)

Dense
(classification)

Softmax
(N)



Fine-tune layers

Small – medium dataset
Different from original data
Input size can change

Replace layers
Randomly initialize +
Train on new dataset

Transfer learning in Pytorch

- Transfer learning in Pytorch can be implemented in two steps
  - Create the new model by detaching the old classification head and attaching the new head
  - Selectively "freeze" and "unfreeze" layers in order to use them as fixed feature extractors or finetuning them
- Each parameter has a `requires_grade` property: when set to `False`, gradients are not computed
- To freeze all layers:

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

- The `summary()` method prints whether a layer is trainable or not

Modifying an existing model

- To change an existing model
  - Define a new model that uses parts of the old model, or..
  - Import the model and then change some of the layers, e.g., the last layer

```
import torchvision
from torchvision import datasets, models, transforms
model_ft = models.vgg16(weights='IMAGENET1K_V1')
num_ftrs = model_ft.classifier[6].in_features
Model_ft.classifier[6] = nn.Linear(num_ftrs,
num_classes)
```

```python
from torchsummary import summary
model_ft = models.vgg16(weights='IMAGENET1K_V1')
summary(model_ft, input_size=(3,224,224))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
        MaxPool2d-17          [-1, 256, 28, 28]               0
           Conv2d-18          [-1, 512, 28, 28]       1,180,160
             ReLU-19          [-1, 512, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       2,359,808
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
        MaxPool2d-24          [-1, 512, 14, 14]               0
           Conv2d-25          [-1, 512, 14, 14]       2,359,808
             ReLU-26          [-1, 512, 14, 14]               0
           Conv2d-27          [-1, 512, 14, 14]       2,359,808
             ReLU-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
        MaxPool2d-31            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-32            [-1, 512, 7, 7]               0
           Linear-33                 [-1, 4096]     102,764,544
             ReLU-34                 [-1, 4096]               0
          Dropout-35                 [-1, 4096]               0
           Linear-36                 [-1, 4096]      16,781,312
             ReLU-37                 [-1, 4096]               0
          Dropout-38                 [-1, 4096]               0
           Linear-39                 [-1, 1000]       4,097,000
================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 218.78
Params size (MB): 527.79
Estimated Total Size (MB): 747.15
----------------------------------------------------------------
```

Modifying an existing model (II)

```
import torchvision
from torchvision import datasets, models,
transforms
model_ft =
models.vgg16(weights='IMAGENET1K_V1')
num_ftrs =
model_ft.classifier[6].in_features
model_ft.classifier[6] =
nn.Linear(num_ftrs, num_classes)
```

```python
from torchsummary import summary
model_ft = models.vgg16(weights='IMAGENET1K_V1')
summary(model_ft, input_size=(3,224,224))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
        MaxPool2d-17          [-1, 256, 28, 28]               0
           Conv2d-18          [-1, 512, 28, 28]       1,180,160
             ReLU-19          [-1, 512, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       2,359,808
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
        MaxPool2d-24          [-1, 512, 14, 14]               0
           Conv2d-25          [-1, 512, 14, 14]       2,359,808
             ReLU-26          [-1, 512, 14, 14]               0
           Conv2d-27          [-1, 512, 14, 14]       2,359,808
             ReLU-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
        MaxPool2d-31            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-32            [-1, 512, 7, 7]               0
           Linear-33                 [-1, 4096]     102,764,544
             ReLU-34                 [-1, 4096]               0
          Dropout-35                 [-1, 4096]               0
           Linear-36                 [-1, 4096]      16,781,312
             ReLU-37                 [-1, 4096]               0
          Dropout-38                 [-1, 4096]               0
           Linear-39                 [-1, 1000]       4,097,000
================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 218.78
Params size (MB): 527.79
Estimated Total Size (MB): 747.15
----------------------------------------------------------------
```

```python
num_classes = 6
num_ftrs = model_ft.classifier[6].in_features
model_ft.classifier[6] = nn.Linear(num_ftrs, num_classes)
summary(model_ft, input_size=(3,224,224))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
        MaxPool2d-17          [-1, 256, 28, 28]               0
           Conv2d-18          [-1, 512, 28, 28]       1,180,160
             ReLU-19          [-1, 512, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       2,359,808
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
        MaxPool2d-24          [-1, 512, 14, 14]               0
           Conv2d-25          [-1, 512, 14, 14]       2,359,808
             ReLU-26          [-1, 512, 14, 14]               0
           Conv2d-27          [-1, 512, 14, 14]       2,359,808
             ReLU-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
        MaxPool2d-31            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-32            [-1, 512, 7, 7]               0
           Linear-33                 [-1, 4096]     102,764,544
             ReLU-34                 [-1, 4096]               0
          Dropout-35                 [-1, 4096]               0
           Linear-36                 [-1, 4096]      16,781,312
             ReLU-37                 [-1, 4096]               0
          Dropout-38                 [-1, 4096]               0
           Linear-39                    [-1, 6]          24,582
================================================================
Total params: 134,285,126
Trainable params: 134,285,126
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 218.77
Params size (MB): 512.26
Estimated Total Size (MB): 731.60
----------------------------------------------------------------
```

```python
num_classes = 6
num_ftrs = model_ft.classifier[6].in_features
model_ft.classifier[6] = nn.Linear(num_ftrs, num_classes)
summary(model_ft, input_size=(3,224,224))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
        MaxPool2d-17          [-1, 256, 28, 28]               0
           Conv2d-18          [-1, 512, 28, 28]       1,180,160
             ReLU-19          [-1, 512, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       2,359,808
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
        MaxPool2d-24          [-1, 512, 14, 14]               0
           Conv2d-25          [-1, 512, 14, 14]       2,359,808
             ReLU-26          [-1, 512, 14, 14]               0
           Conv2d-27          [-1, 512, 14, 14]       2,359,808
             ReLU-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
        MaxPool2d-31            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-32            [-1, 512, 7, 7]               0
           Linear-33                 [-1, 4096]     102,764,544
             ReLU-34                 [-1, 4096]               0
          Dropout-35                 [-1, 4096]               0
           Linear-36                 [-1, 4096]      16,781,312
             ReLU-37                 [-1, 4096]               0
          Dropout-38                 [-1, 4096]               0
           Linear-39                    [-1, 6]          24,582
================================================================
Total params: 134,285,126
Trainable params: 134,285,126
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 218.77
Params size (MB): 512.26
Estimated Total Size (MB): 731.60
----------------------------------------------------------------
```

```python
for i in model_ft.features.parameters():
    i.requires_grad = False

summary(model_ft, (3,224,224))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
        MaxPool2d-17          [-1, 256, 28, 28]               0
           Conv2d-18          [-1, 512, 28, 28]       1,180,160
             ReLU-19          [-1, 512, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       2,359,808
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
        MaxPool2d-24          [-1, 512, 14, 14]               0
           Conv2d-25          [-1, 512, 14, 14]       2,359,808
             ReLU-26          [-1, 512, 14, 14]               0
           Conv2d-27          [-1, 512, 14, 14]       2,359,808
             ReLU-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
        MaxPool2d-31            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-32            [-1, 512, 7, 7]               0
           Linear-33                 [-1, 4096]     102,764,544
             ReLU-34                 [-1, 4096]               0
          Dropout-35                 [-1, 4096]               0
           Linear-36                 [-1, 4096]      16,781,312
             ReLU-37                 [-1, 4096]               0
          Dropout-38                 [-1, 4096]               0
           Linear-39                    [-1, 6]          24,582
================================================================
Total params: 134,285,126
Trainable params: 119,570,438
Non-trainable params: 14,714,688
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 218.77
Params size (MB): 512.26
Estimated Total Size (MB): 731.60
----------------------------------------------------------------
```

Batch normalization

- Batch normalization behaves differently from most other layers due to an additional option specifying whether the layer should operate in training or inference mode

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch normalization

- Batch normalization behaves differently from most other layers because it has an additional options specifying whether the layer should operate in training or inference mode

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- In **training** mode, the layer uses the mean and variance of the current batch
- In **inference** mode, the layer uses the accumulated mean and variance learnt during training
- These are **not** counted as trainable parameters

Batch normalization

- Batch normalization behaves differently from most other layers because it has an additional options specifying whether the layer should operate in training or inference mode

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Scale and shift are standard trainable parameters: freezing is controlled by the `requires_grad` property

Evaluation mode

- `model.eval()` sets the entire model in evaluation mode
  - Must be done before inference!
- `model.train()` sets the entire model in trainig mode
  - Must be done before training!
- Layers that are affected by the mode are:
  - BatchNorm
  - Dropout (regularization layer)

Where to find pre-trained models

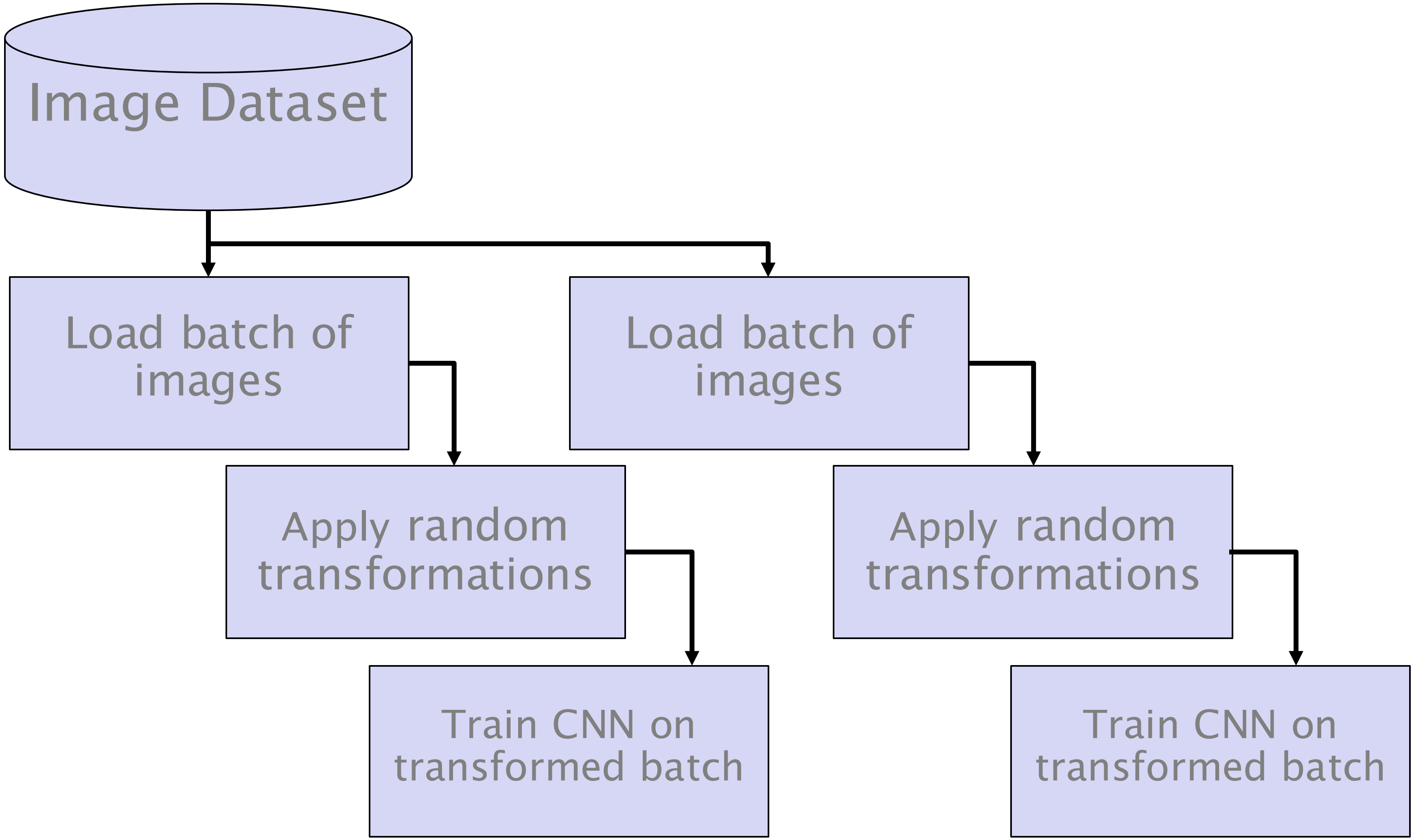
- Pre-defined neural networks for addressing different tasks are available
- https://pytorch.org/vision/stable/models.html

DATA AUGMENTATION

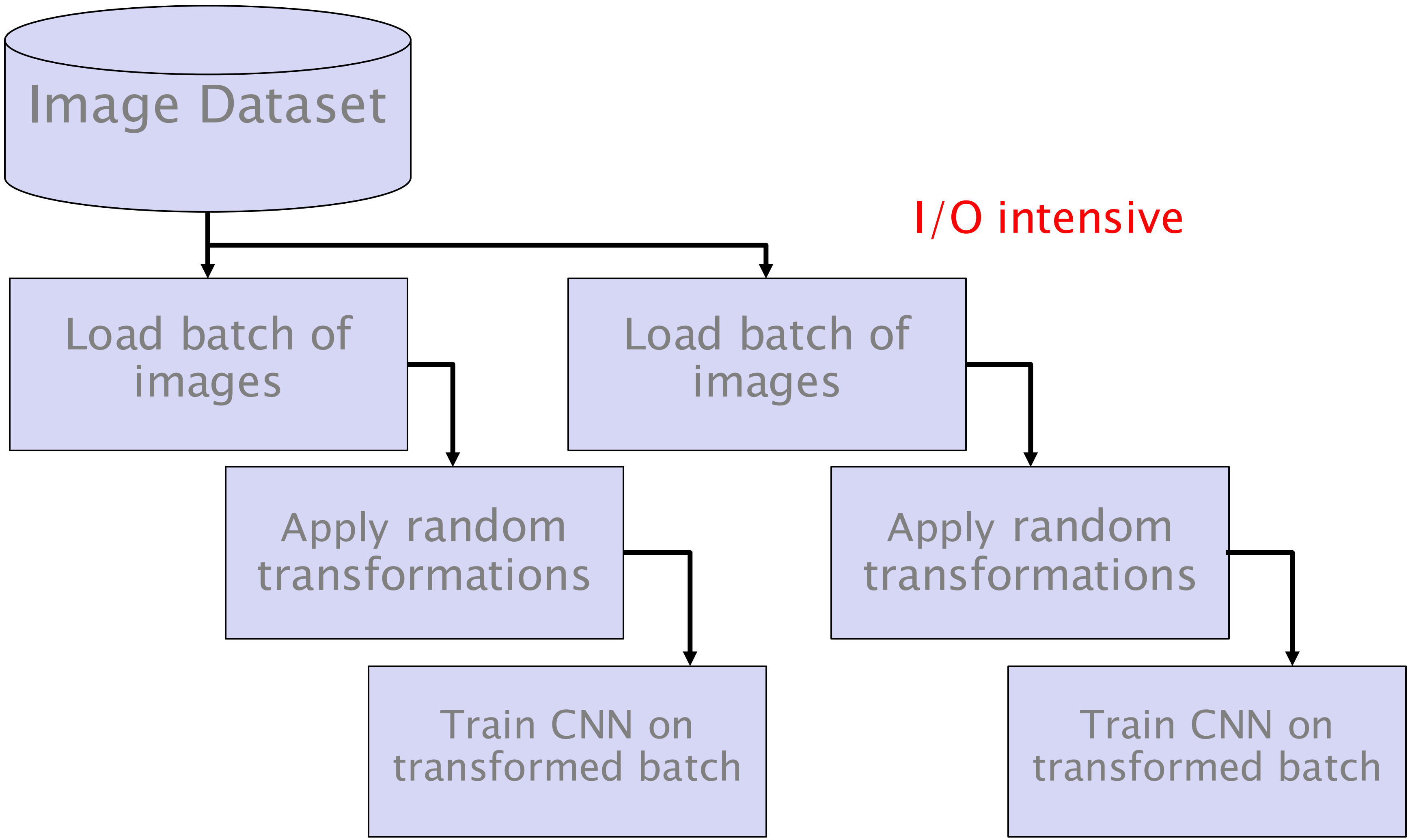
Data augmentation

- CNNs, by constructions, are only invariant to translation
- We would like our networks to be invariant to rotation, scale, stretching, mirroring, illumination, contrast, noise, etc.
- Acts as a regularizer by applying random *realistic* transformations *that should not affect the class or label*

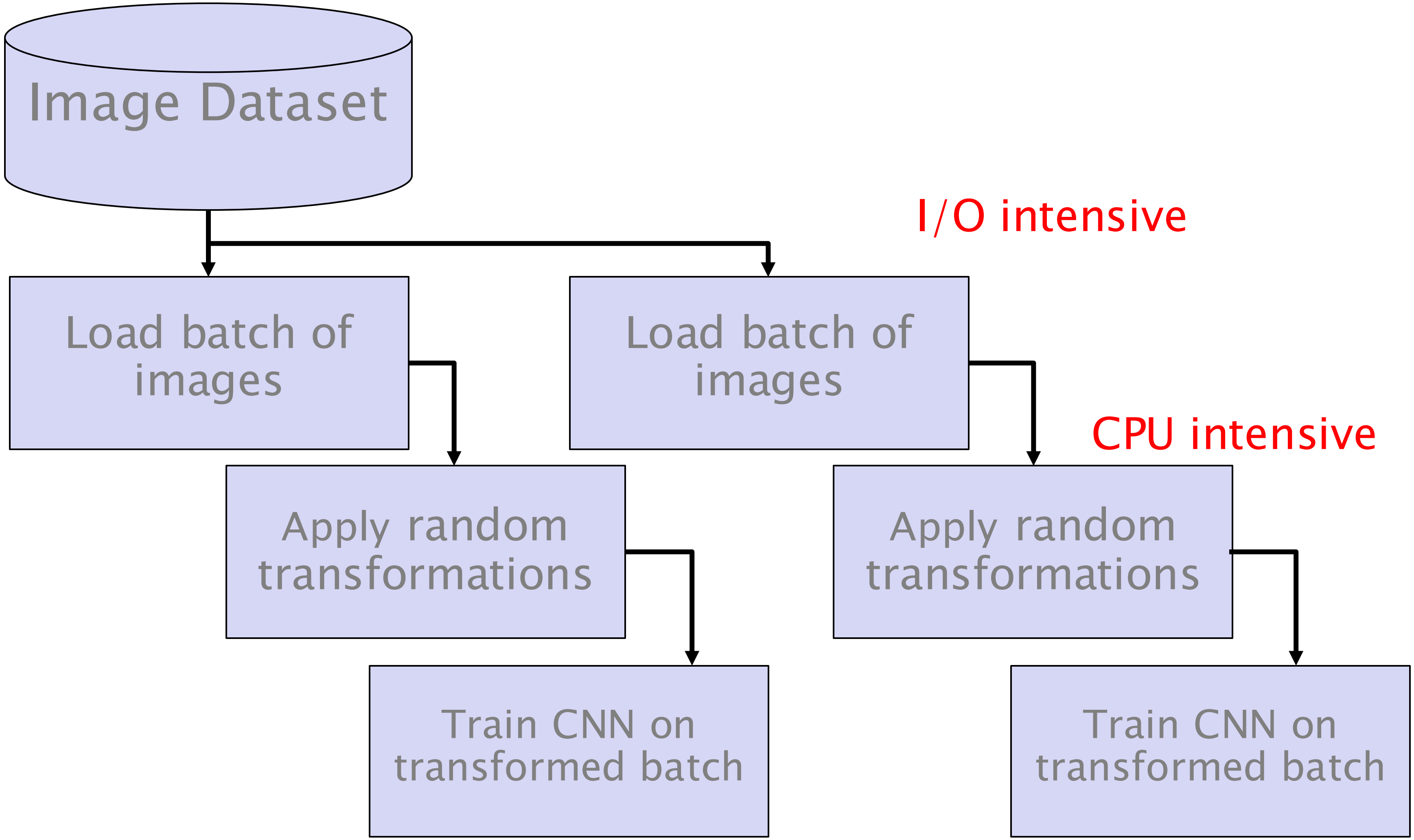
On-the-fly data augmentation



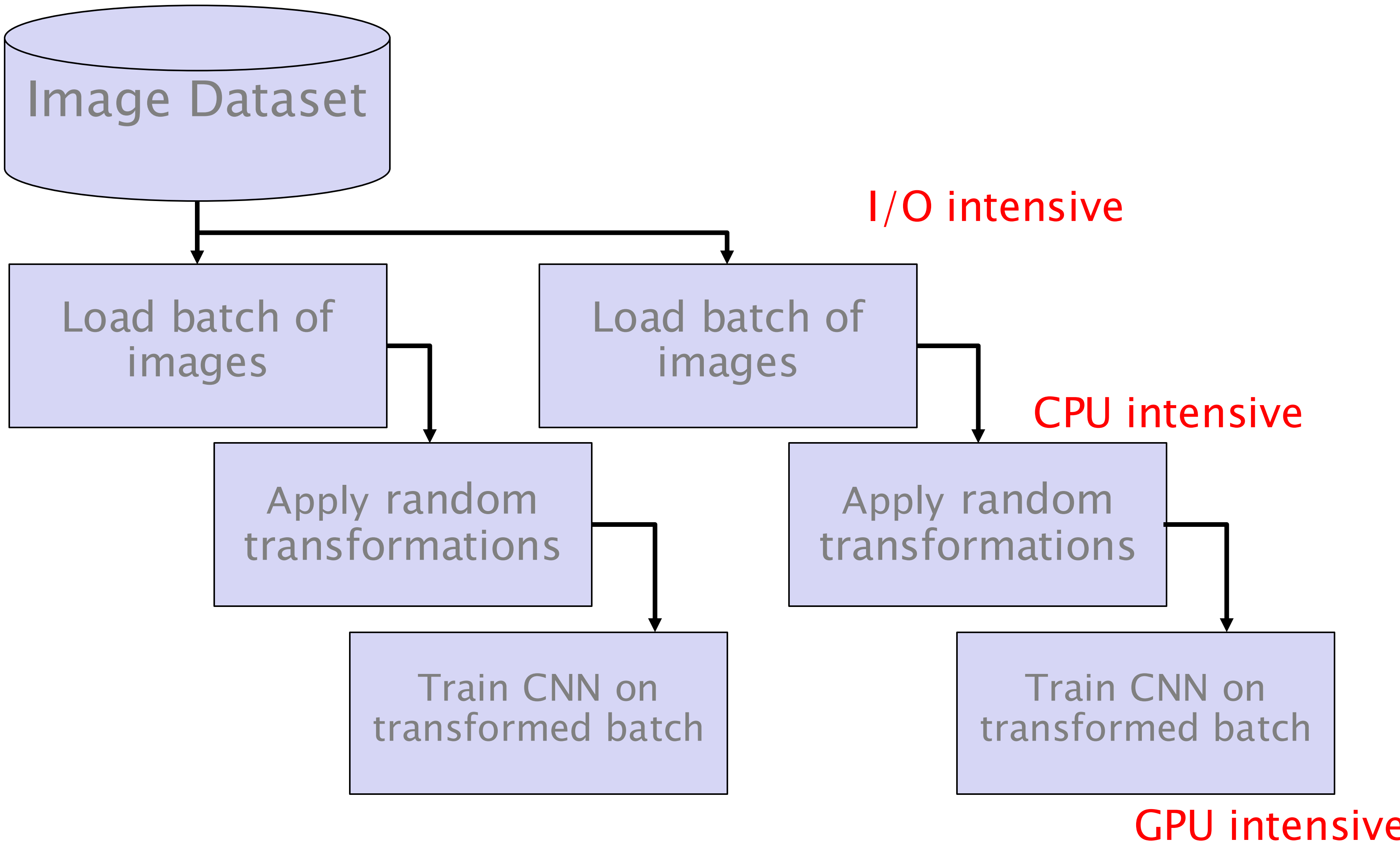
On-the-fly data augmentation



On-the-fly data augmentation



On-the-fly data augmentation



How to define a dataset

```python
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

How to define a dataset

```python
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

Assuming labels are stored in a csv

```
tshirt1.jpg, 0
tshirt2.jpg, 0
......
ankleboot999.jpg, 9
```

How to define a dataset

```
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

Transformations that should be applied to the input and target

How to define a dataset

```
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

Return the number of samples in the dataset

How to define a dataset

```
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

Loads and returns a sample from the dataset at the given index idx

How to define a dataset

```
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

Preprocessing and data augmentation can be performed here

Data augmentation

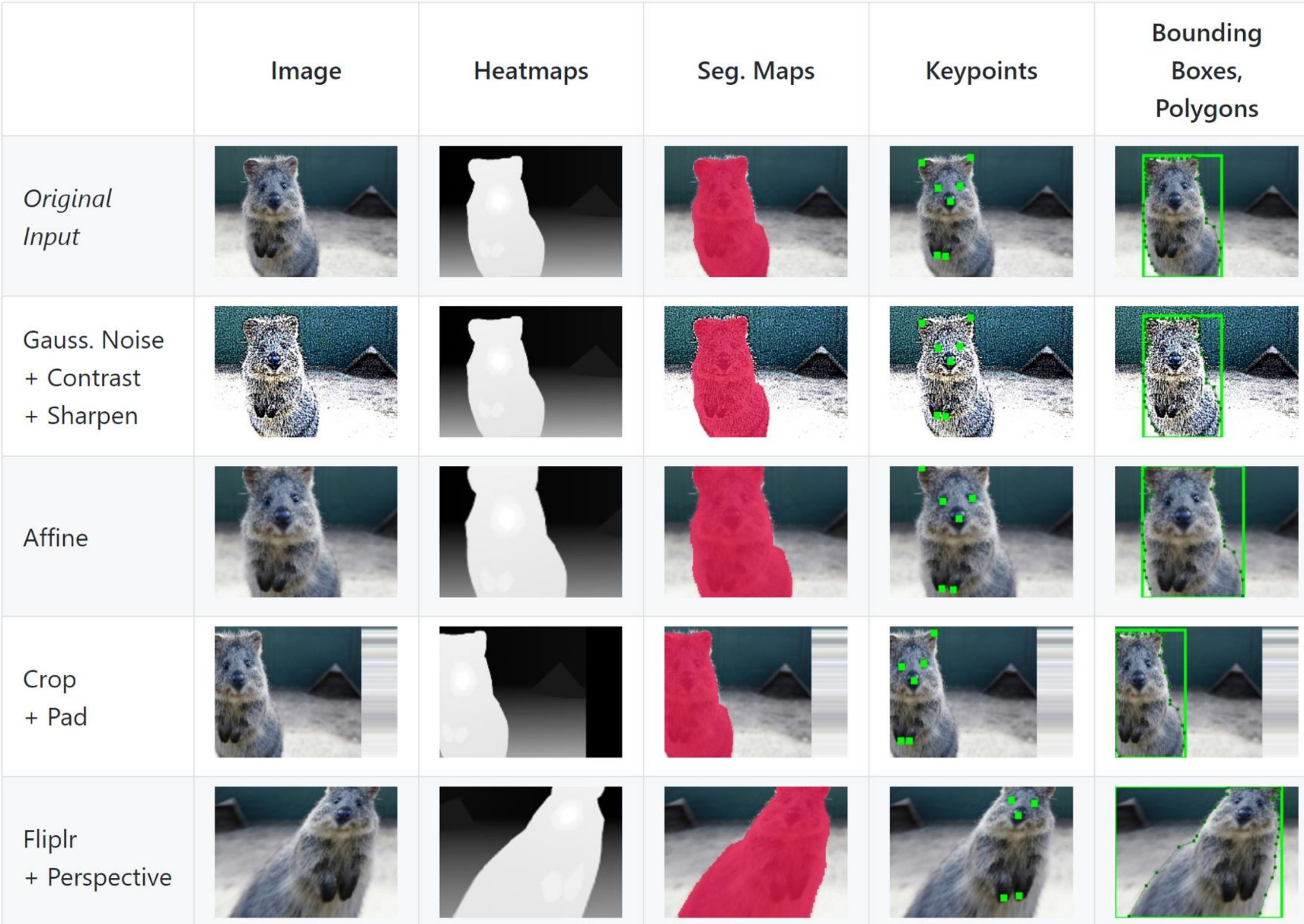
- Torchvision supports common computer vision transformations
  - modules: torchvision.transforms and torchvision.transfor ms.v2
  - can be used to transform or augment data for training or inference of different tasks (image classification, detection, segmentation, video classification)
- Transformations includes
  - Preprocessing and normalization
  - Data augmentation
- Transformations can be composed

Example

```python
# Data augmentation and normalization for training
# Just normalization for validation
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

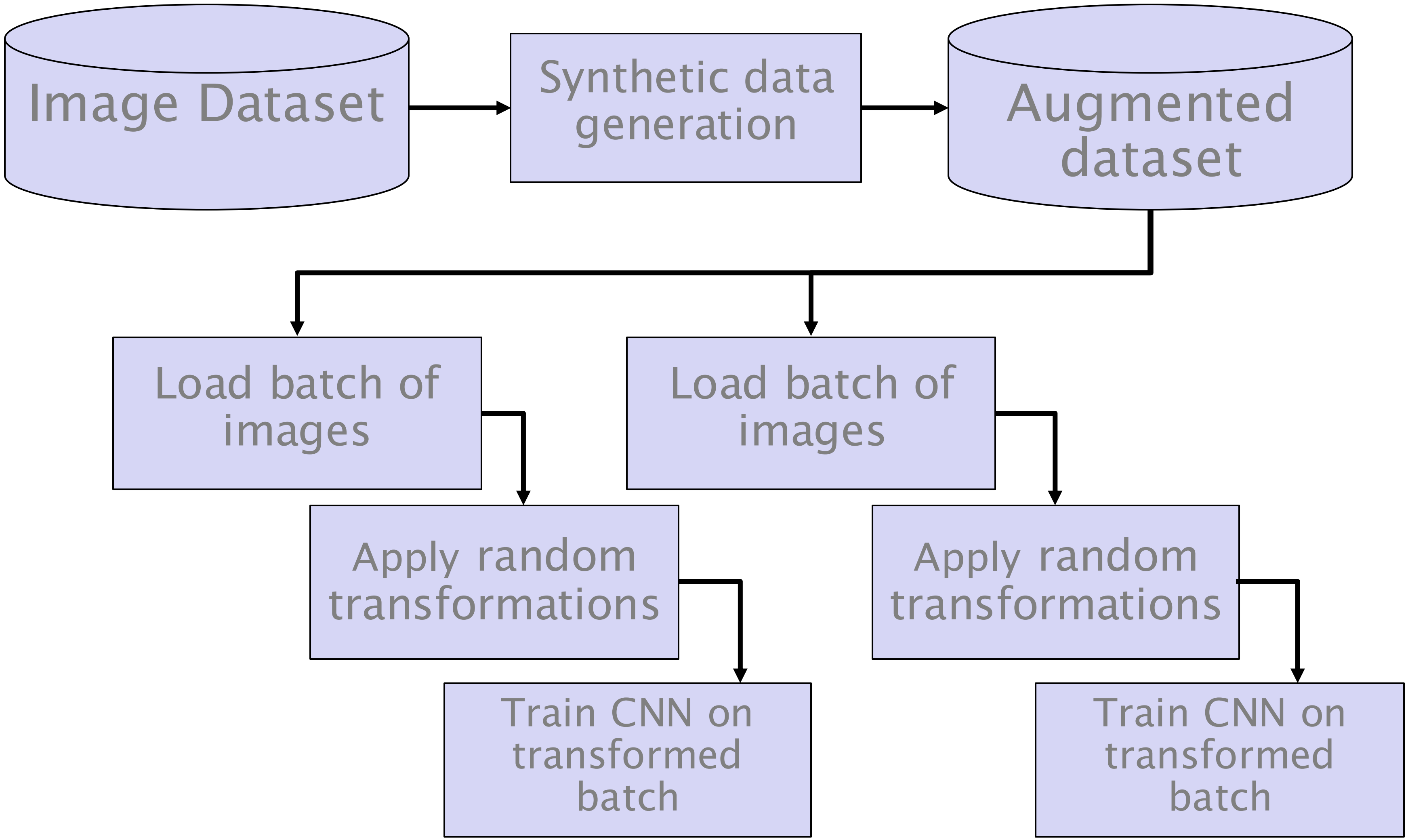
Statistics calculated on the ImageNet dataset (by default, Pytorch converts images between 0 and 1)

Beyond classification

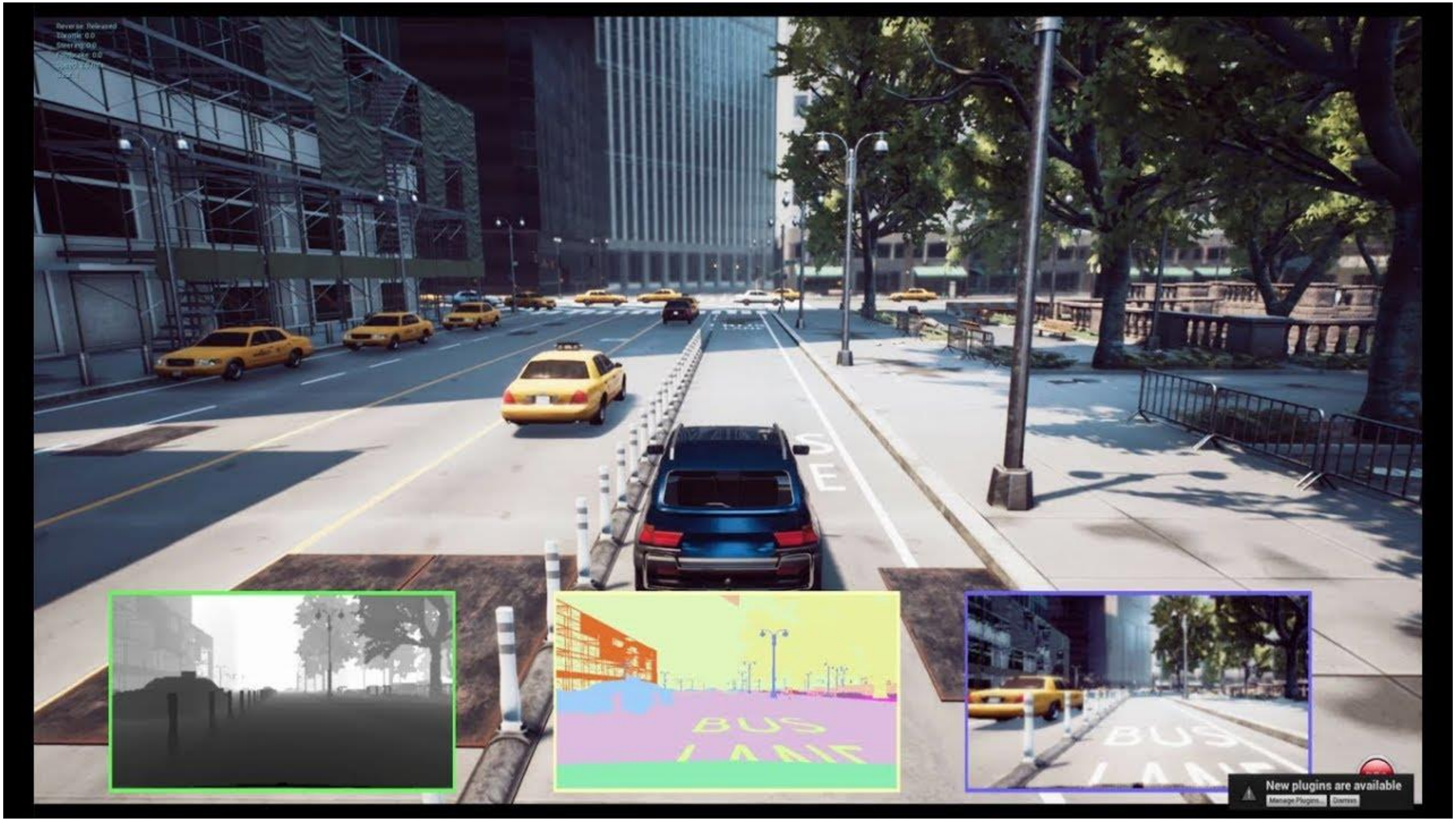


https://github.com/aleju/imgaug

Offline data augmentation

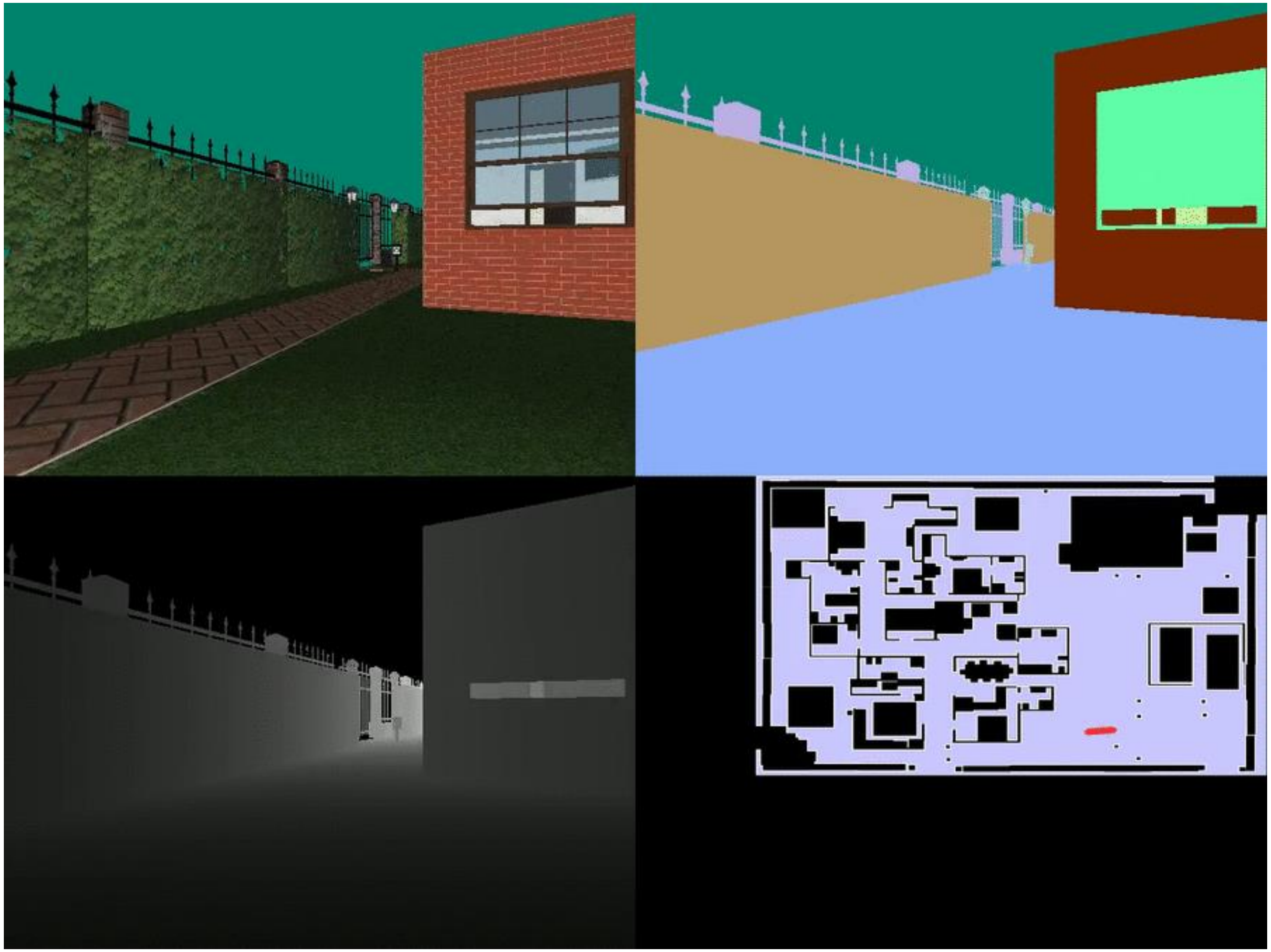


Synthetic data generation



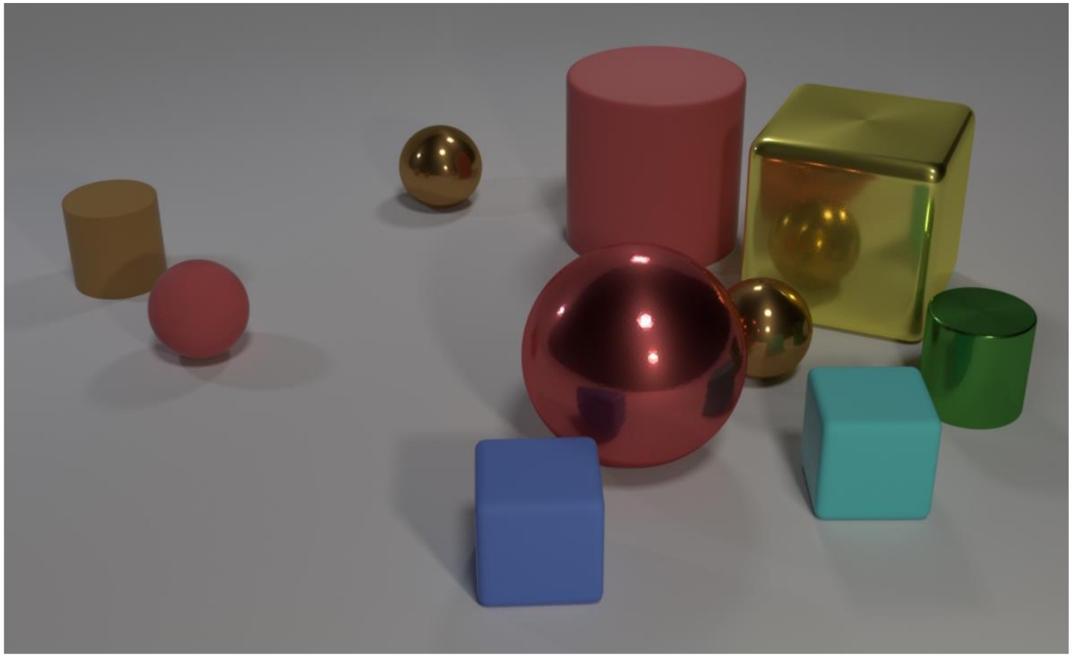
https://microsoft.github.io/AirSim/

Synthetic data generation



https://github.com/facebookresearch/House3D

Synthetic data generation



Q: Are there an **equal number** of **large things** and **metal spheres**?
Q: **What size** is the **cylinder** **that is left of** the **brown metal** thing **that is left of** the **big sphere**?
Q: There is a **sphere** with the **same size as** the **metal cube**; is it **made of the same material as** the **small red sphere**?
Q: **How many** objects are **either** **small cylinders** or **red** things?

https://cs.stanford.edu/people/jcjohns/clevr/

Synthetic data generation

https://research.nvidia.com/publication/2018-06_Falling-Things

References

- https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
- https://pytorch.org/vision/main/transforms.html

NON-SEQUENTIAL MODELS

Non sequential architectures

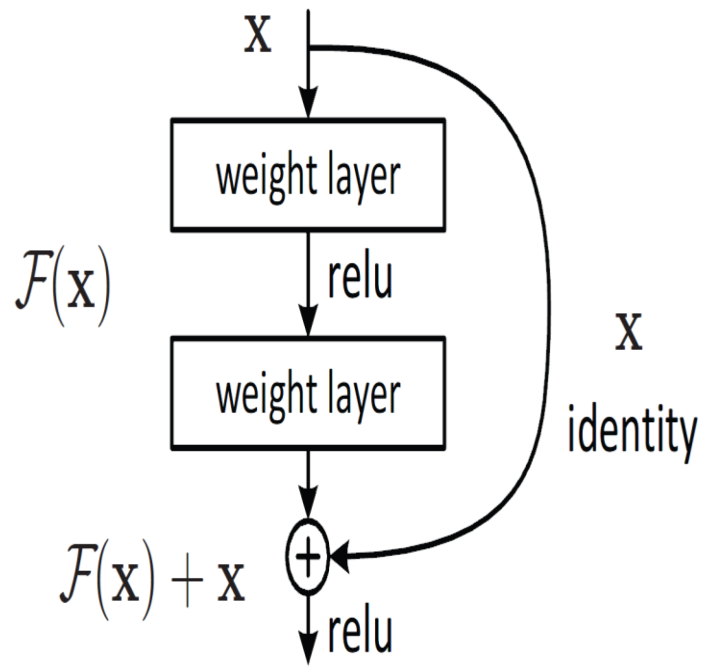
- Residual or parallel blocks
  - Residual networks
  - Inception Layers
  - Multiple branches
- Multiple Inputs - Outputs
- Parameter sharing
  - Since a layer is a function, it can be re-used, with the same parameters, in different parts of the network
  - If multiple path flow through the same layer, the gradients will accumulate updating the same weights
- Extraction of subnetworks from existing models
  - Model “surgery”

Residual Networks

- We can define functions and modules to implement re-usable blocks

```
output =
residual_block(x)
```

- These blocks can be used to compose other networks as if they were layer



Residual block

```
class ResidualBlock(nn.Module):
def __init__(self, in_channels, out_channels):
    super(ResidualBlock, self).__init__()
    self.conv1 = nn.Sequential(
                 nn.Conv2d(in_channels, out_channels,
                  kernel_size = 3, padding = 1),
                  nn.ReLU())
    self.conv2 = nn.Conv2d(out_channels, out_channels,
         kernel_size = 3, stride = 1, padding = 1)
    self.relu = nn.ReLU()
    self.out_channels = out_channels

def forward(self, x):
    residual = x
    out = self.conv1(x)
    out = self.conv2(out)
    out += residual
    out = self.relu(out)
    return out
```

Multiple inputs – outputs

- A Model object can take an array of tensors as either inputs or outputs

```
def forward(self, x):
   # forward pass
    x1 = ...
   x2 = ...
   return x1, x2
```

Multiple inputs – outputs

- A Model object can take an array of tensors as either inputs or outputs

Age? 50 Gender? M

Loss = αMSE(age) + βCE(gender)

Multiple inputs – outputs

- A Model object can take an array of tensors as either inputs or outputs

- Since gradient descent requires to minimize a *scalar*, the loss is calculated as the **(weighted) sum** of the loss for each output

```
out1, out2 = model(data)
loss1 = mse(out1, target1)
loss2 = ce(out2, target2)
loss = alfa*loss1 + beta*loss2
loss.backward()
```

Age? 50 Gender? M

Loss = αMSE(age) + βCE(gender)