PyTorch Bootcamp

Machine Learning and Deep Learning
A. A. 2023/24

Slide credits: *Chiara Plizzari*
Teaching Assistants: *Paolo Rabino, Stephany Ortuño Chanelo*

What you will learn

Lesson 1:

- What is PyTorch?
- What is a Tensor?
- How to build a Dataset & DataLoader

Lesson 2:

- How to train your model

Lesson 3:

- How to set up a complete training pipeline

Lesson 4:

- Training standard CNNs: AlexNet & ResNet

Lesson 5:

- Transfer Learning
- Wandb: How to visualise your models and keep track of your results

What is PyTorch?

- Open source machine learning library
- Developed by Facebook's AI Research lab
- It leverages the power of GPUs
- Automatic computation of gradients
- Makes it easier to test and develop new ideas.

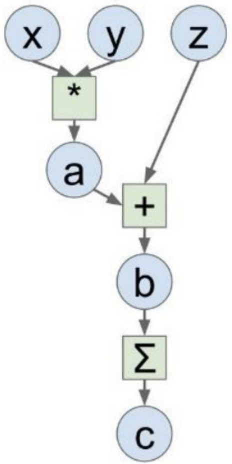
Other libraries?

Why PyTorch?

- It is pythonic- concise, close to Python conventions
- Strong GPU support
- Autograd- automatic differentiation
- Many algorithms and components are already implemented
- Similar to NumPy

Why PyTorch?

Computation Graph



Numpy

```python
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```

Tensorflow

```python
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

PyTorch

```python
import torch

N, D = 3, 4

x = torch.rand((N, D),requires_grad=True)
y = torch.rand((N, D),requires_grad=True)
z = torch.rand((N, D),requires_grad=True)

a =x * y
b =a + z
c=torch.sum(b)

c.backward()
```

Getting Started with PyTorch

Installation:

- Via Anaconda/Miniconda:

```
conda install pytorch -c pytorch
```

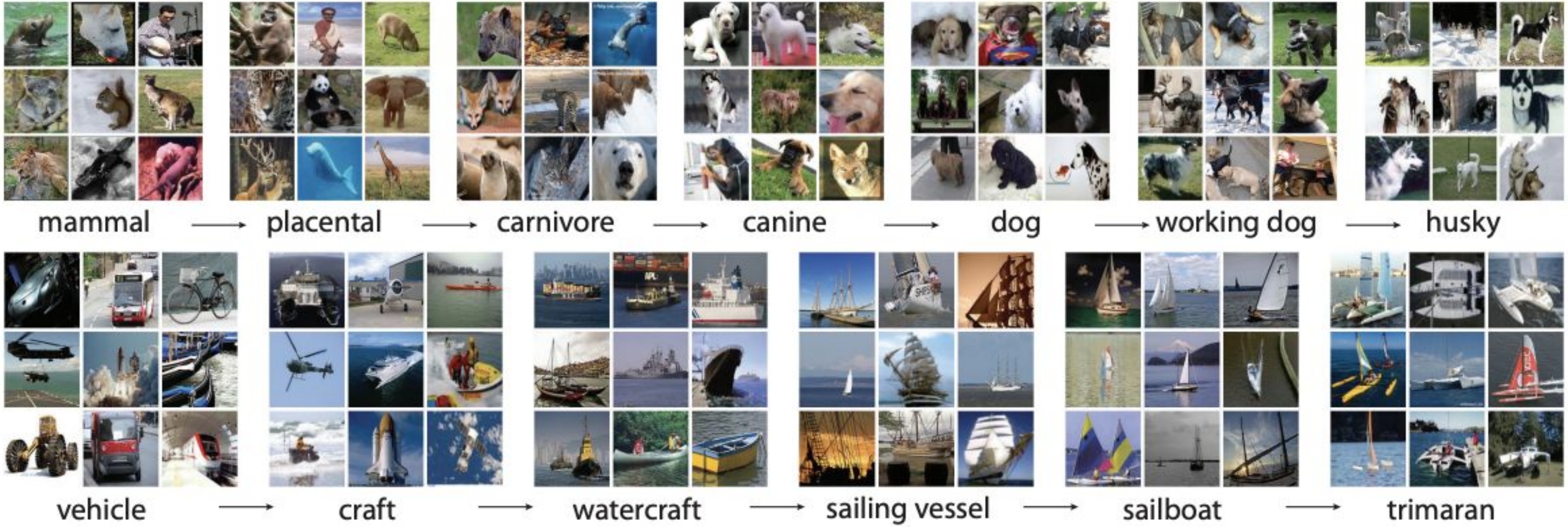
- Via pip:

```
pip3 install torch
```

Dataset & DataLoader

Lesson 1

ImageNet


mammal → placental → carnivore → canine → dog → working dog → husky
vehicle → craft → watercraft → sailing vessel → sailboat → trimaran

Tensors

What a tensor is?

A vector is a 1-dimensional tensor, a matrix is a 2-dimensional tensor, an array with three indices is a 3-dimensional tensor (RGB color images for example). The fundamental data structure for neural networks are tensors and PyTorch (as well as pretty much every other deep learning framework) is built around tensors.

```
#                    R     G     B
img = tensor([[[ 48,  80,  79],
                [175, 104, 207],
                [162,  24, 224],
                [ 97,  27,  28],
                [ 51, 137,  60],
                [124, 214, 249]],
                 ...
             ]])
t.size() ==> [256, 256, 3]
t.device ==> gpu:0
t.dtype ==> torch.float32
```

Initializing a Tensor

Directly from data

```
data = [[1, 2],[3, 4]]
x_data = torch.tensor(data)
```

From a NumPy array

```
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
```

Initializing a Tensor

From another tensor

The new tensor retains the properties (shape, datatype) of the argument tensor, unless specified.

```
x_ones = torch.ones_like(x_data) # retains the properties of x_data
print(f"Ones Tensor: \n {x_ones} \n")

x_rand = torch.rand_like(x_data, dtype=torch.float) # overrides the datatype of x_data
print(f"Random Tensor: \n {x_rand} \n")
```

Out:

```
Ones Tensor:
 tensor([[1, 1],
        [1, 1]])

Random Tensor:
 tensor([[0.9152, 0.2666],
        [0.0863, 0.9133]])
```

Attributes of a tensor

Tensor attributes describe their **shape**, **datatype**, and **device** on which they are stored.

```
tensor = torch.rand(3,4)

print(f"Shape of tensor: {tensor.shape}")
print(f"Datatype of tensor: {tensor.dtype}")
print(f"Device tensor is stored on: {tensor.device}")
```

Out:

```
Shape of tensor: torch.Size([3, 4])
Datatype of tensor: torch.float32
Device tensor is stored on: cpu
```

Initializing a Tensor

By default, tensors are created on the CPU. We need to explicitly move them to the GPU using the **.to** method.

```
# We move our tensor to the GPU if available
if torch.cuda.is_available():
  tensor = tensor.to('cuda')
```

Operations on tensors

Indexing and **slicing**:

```
tensor = torch.ones(4, 4)
print('First row: ',tensor[0])
print('First column: ', tensor[:, 0])
print('Last column:', tensor[..., -1])
tensor[:,1] = 0
print(tensor)
```

Out:

```
First row:  tensor([1., 1., 1., 1.])
First column:  tensor([1., 1., 1., 1.])
Last column: tensor([1., 1., 1., 1.])
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])
```

Operations on tensors

Arithmetic operations:

```
# This computes the matrix multiplication between two tensors. y1, y2, y3 will have the same
value
y1 = tensor @ tensor.T
y2 = tensor.matmul(tensor.T)
```

```
# This computes the element-wise product. z1, z2, z3 will have the same value
z1 = tensor * tensor
z2 = tensor.mul(tensor)
```

… and the simple element-wise addition **add()**

Your turn!

- Start familiarising with Google Colab and tensors!
- You can create a Notebook on https://colab.google/ at any time;
- You can access both CPU and GPUs by changing the runtime.
- Open the notebook at the following link and implement the following exercises!

Notebook @ https://colab.research.google.com/drive/1CfXwKlB4EIgasClLBi6-OggWFjwVIT5U?usp=sharing

Datasets & DataLoaders

Dataset and DataLoader

The `Dataset` and `DataLoader` classes encapsulate the process of pulling your data from storage and exposing it to your training loop in batches.

The `Dataset` is responsible for accessing and processing single instances of data.

The `DataLoader` pulls instances of data from the `Dataset` (either automatically or with a sampler that you define), collects them in batches, and returns them for consumption by your training loop. The `DataLoader` works with all kinds of datasets, regardless of the type of data they contain.

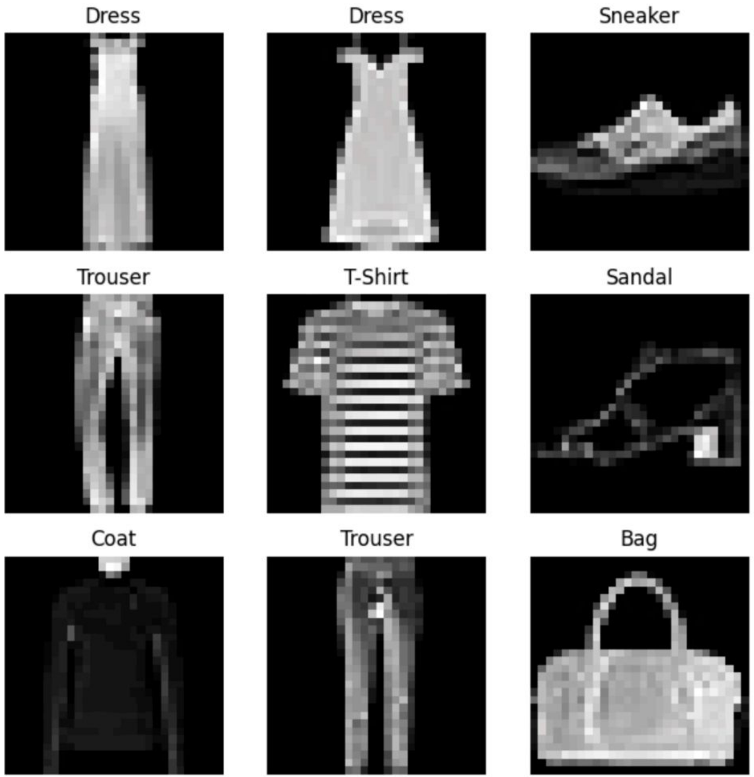
Loading a dataset

```
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor, Lambda
import matplotlib.pyplot as plt


training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

Iterating and visualizing the dataset

```
labels_map = {
    0: "T-Shirt",
    1: "Trouser",
    2: "Pullover",
    3: "Dress",
    4: "Coat",
    5: "Sandal",
    6: "Shirt",
    7: "Sneaker",
    8: "Bag",
    9: "Ankle Boot",
}
figure = plt.figure(figsize=(8, 8))
cols, rows = 3, 3
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(training_data), size=(1,)).item()
    img, label = training_data[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(labels_map[label])
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```

Iterating and visualizing the dataset



Creating a custom Dataset for your data

```
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)
        sample = {"image": image, "label": label}
        return sample
```

1 **download, read data, etc.**

2 **return one item on the index**

3 **return the data length**

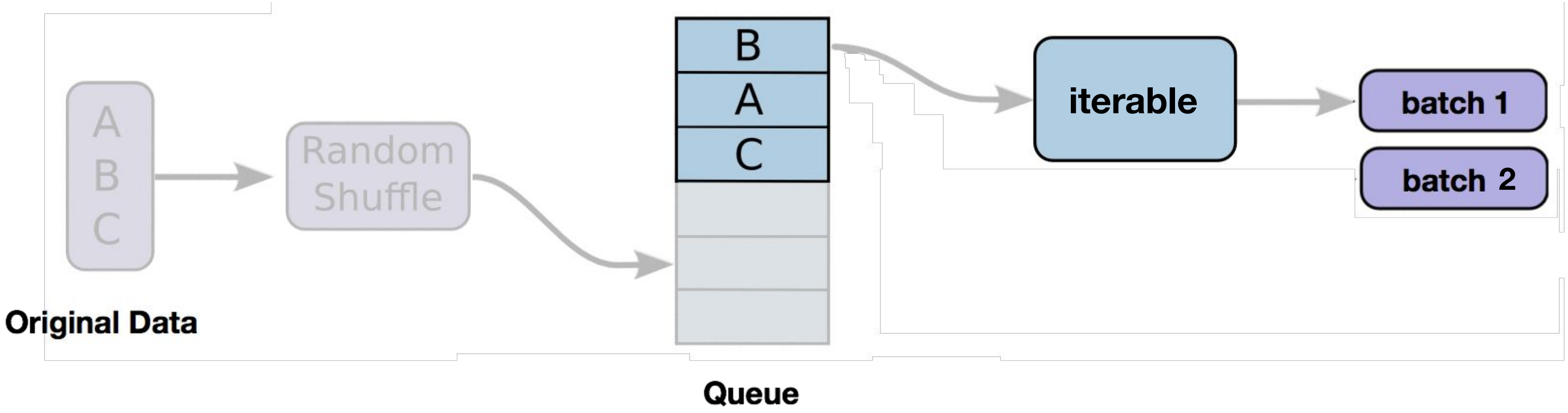
Preparing the data for training with DataLoader

The Dataset retrieves our dataset's features and labels one sample at a time. While training a model, we typically want to pass samples in "minibatches", reshuffle the data at every epoch to reduce model overfitting, and use Python's multiprocessing to speed up data retrieval.

DataLoader is an iterable that abstracts this complexity for us in an easy API.

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

Preparing the data for training with DataLoader



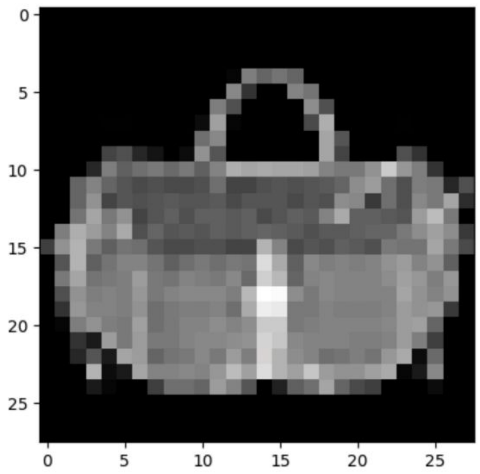

```
for i, data in enumerate(train_loader, 0):
    # get the inputs
    inputs, labels = data

    # Run your training process
    print(epoch, i, "inputs", inputs, "labels", labels)
```

Iterate through the DataLoader

We have loaded that dataset into the DataLoader and can iterate through the dataset as needed. Each iteration below returns a batch of train_features and train_labels (containing batch_size=64 features and labels respectively). Because we specified shuffle=True, after we iterate over all batches the data is shuffled.

```
# Display image and label.
train_features, train_labels = next(iter(train_dataloader))
print(f"Feature batch shape: {train_features.size()}")
print(f"Labels batch shape: {train_labels.size()}")
img = train_features[0].squeeze()
label = train_labels[0]
plt.imshow(img, cmap="gray")
plt.show()
print(f"Label: {label}")
```

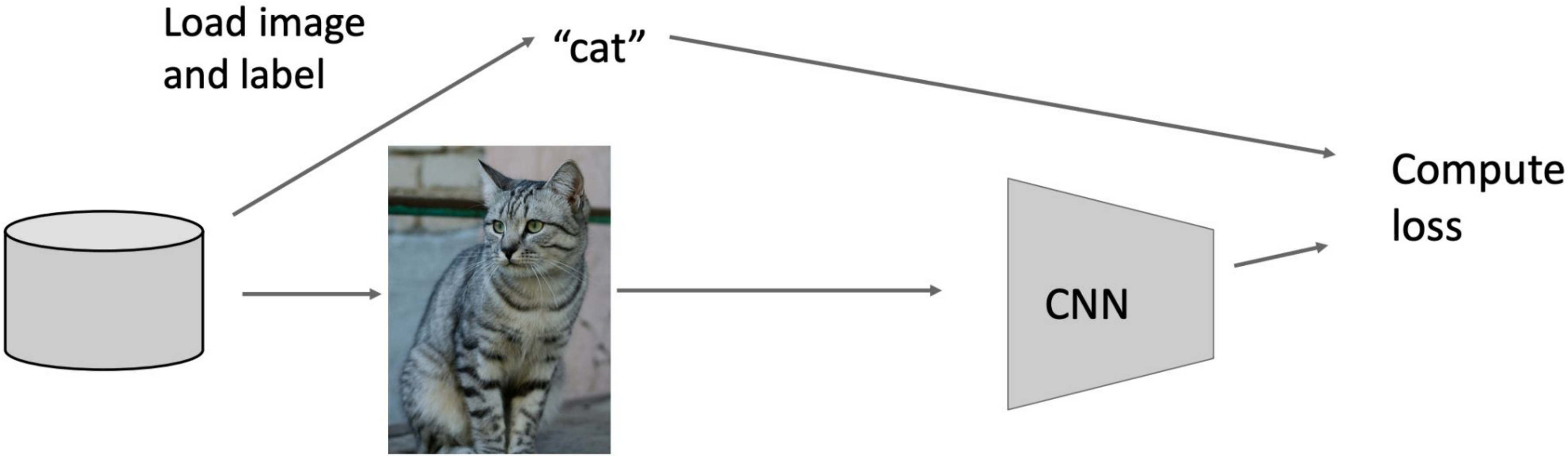
Iterate through the DataLoader



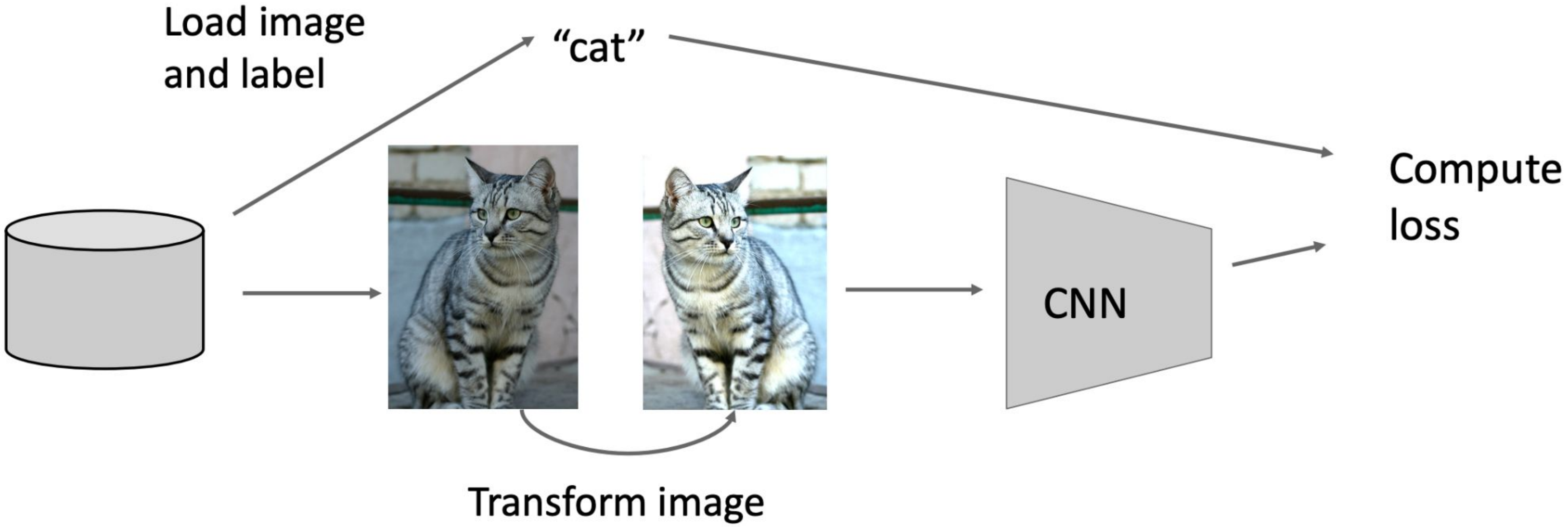
Out:

```
Feature batch shape: torch.Size([64, 1, 28, 28])
Labels batch shape: torch.Size([64])
Label: 8
```

Data Augmentation



Data Augmentation



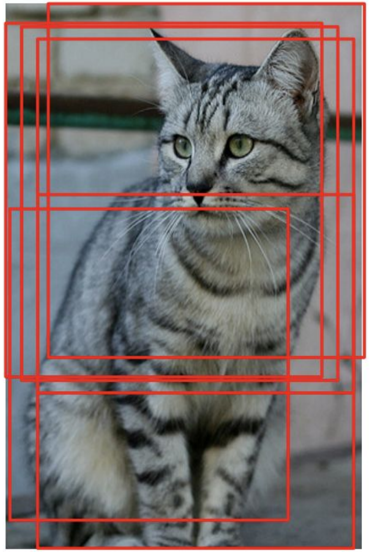
Data Augmentation: Horizontal Flips

```
torchvision.transforms.RandomHorizontalFlip(p=0.5)
torchvision.transforms.RandomVerticalFlip(p=0.5)
```

Data Augmentation: Random Crops and Scales

Training: sample random crops / scales
ResNet:
1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops
ResNet:
1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips



```
torchvision.transforms.RandomCrop(size, padding=None,
pad_if_needed=False, fill=0, padding_mode='constant')
torchvision.transforms.RandomResizedCrop(size, scale=(0.08, 1.0),
ratio=(0.75, 1.3333333333333333), interpolation=2)
```

Data Augmentation: Color Jitter

Simple: Randomize contrast and brightness

More complex:

1. Apply PCA to all [R,G,B] pixels in training set

2. Sample a “color offset” along principal component directions

3. Add offset to all pixels of a training image

```
torchvision.transforms.ColorJitter(brightness=0, contrast=0, saturation=0, hue=0)
```

PyTorch Implementation

```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])
    ]),
}
```

Your turn! 

At this link http://cs231n.stanford.edu/tiny-imagenet-200.zip you can find the TinyImageNet dataset.

1. Load the dataset and create a DataLoader for that.
2. How many classes does it contain?
3. How many samples?
4. Visualize one example for class for 10 of its classes.

Notebook @ https://colab.research.google.com/drive/1AVbofw_tN794gQ8JeH75OxaL4_rao_8f?usp=sharing