Deep Learning Practitioner’s Toolbox
Logistics

• Project proposal submission is due **today**.

• Reviews will be published in the following days.
THAT MOMENT WHEN YOU

HAVE TO START WORKING
ON YOUR FINAL PROJECT
The Toolbox

• Torch hub
• Hooks
• Tensorboard
• Transforms
• Reproducibility
• Saving & Loading Models
Don’t Reinvent the Wheel!

Use Existing Tools!

Image Credits: https://postandparcel.info/80130/news/parcel/reinventing-the-wheel/
Torch Hub

```python
# list of available models in pyTorch's repo.
torch.hub.list('pytorch/vision')

# printed: ['alexnet', 'deeplabv3_mobilenet_v3_large', 'deeplabv3_resnet101', ...]

# load model
model = torch.hub.load('pytorch/vision', 'resnet18', pretrained=True)
model.eval()

input = torch.randn(1, 3, 224, 224)
output = model(input)

# output.shape is 1x1000
```
Hooks

Forward Pass

Model

Backward Pass
Register a Hook

def my_hook(module, input, output):
    # module: the module being hooked
    # input: a tuple of inputs
    # output: a tuple of outputs
    print("hook!", input[0].shape, output[0].shape)

    # register the hook
    net.conv1.register_forward_hook(my_hook)

    # use the hook
    y = net(x)
    # printed: "hook! torch.Size([...]) torch.Size([...])"
Remove a Hook

• A remove handle is returned during the registration.

```python
# register the hook
handle = net.conv1.register_forward_hook(my_hook)

# remove the hook
handle.remove()

# use the hook
y = net(x)
# nothing is printed
```
class FeatureExtractor:
    def __init__(self, model):
        self._feature = None
        self.model = model
        self._handle = self.model.layer1.register_forward_hook(self._get_feature_hook())

    def __del__(self):
        self._handle.remove()

    def _get_feature_hook(self):
        def _get_feature(module, input, output):
            self._feature = output
            return _get_feature
        return _get_feature

    def __call__(self, input):
        output = self.model(input)
        return self._feature
Feature Extraction

```python
# create a model
model = torch.hub.load('pytorch/vision', 'resnet18', pretrained=True)

# create a feature extractor
feature_extractor = FeatureExtractor(model)

# create input
input = torch.randn(1, 3, 224, 224)

# get features
feat = feature_extractor(input)

# feat.shape is 1x64x56x56
```
live loss plot

TensorBoard

Image credit to Tensorflow
# install tensorboard
!pip install tensorboard

%load_ext tensorboard

...

# run the user interface
%s tensorboard --logdir .
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter()
TensorBoard Summary Writer

```python
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter()
```
```python
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter()
```

Everything is here!
# train
for epoch in range(EPOCHS):
    ...
    writer.add_scalar("Loss/train", loss, epoch)
    writer.add_scalar("Loss/val", val_loss, epoch)
    ...

![Graphs showing loss/train and loss/val](image.png)
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter()

# train
for epoch in range(EPOCHS):
    ...
    writer.add_scalar("Loss/train", loss, epoch)
    writer.add_scalar("Loss/val", val_loss, epoch)
    writer.add_scalar("Accuracy/train", acc, epoch)
    writer.add_scalar("Accuracy/val", val_acc, epoch)
    ...
TensorBoard

Summary Writer

Accuracy/train
- tag: Accuracy/train
- Accuracy/val
- tag: Accuracy/val

Loss
- Loss/train
- tag: Loss/train
- Loss/val
- tag: Loss/val
Summary Writer

TensorBoard

Accuracy/train
Accuracy/val
Loss/train
Loss/val

 Runs
Write a regex to filter runs

TOGGLE ALL RUNS

DL4CV Weizmann

Week 13 - Tutorial – DL Toolbox
TensorBoard Log Images

```python
# fetch a batch of data
X, y = next(iter(train_dataloader))

# create an image from all the images
grid = torchvision.utils.make_grid(X)

# add to summary
writer.add_image("images", grid)
```
# create matplotlib figure
figure = ...  

# add to summary
writer.add_figure("batch_example", figure)
TensorBoard

Graphs

\[ \text{writer.add_graph(model, X)} \]
for name, weight in model.named_parameters():
    writer.add_histogram(name, weight, epoch)
    writer.add_histogram(f'{name}.grad', weight.grad, epoch)
writer.add_hparams({'lr': LR, 'batch_size': B, 'num_epochs': EPOCHS},
writer.add_hparams({'lr': LR, 'batch_size': B, 'num_epochs': EPOCHS},
{'hparam/loss': train_loss, 'hparam/loss_val': val_loss,
 'hparam/accuracy': train_acc, 'hparam/accuracy_val': val_loss})
Don’t Reinvent the Wheel!

Use Existing Tools!

Image Credits: https://postandparcel.info/80130/news/parcel/reinventing-the-wheel/
**Albumentations**

- Data Augmentations for various tasks
  - Classification / Representation Learning

```python
import albumentations as A

transform = A.Compose([
    A.RandomCrop(width=256, height=256),
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
])
```

Image credit to albumentations tutorial: [https://albumentations.ai/docs/getting_started/image_augmentation/](https://albumentations.ai/docs/getting_started/image_augmentation/)
Albumentations

• Data Augmentations for various tasks
  • Classification / Representation Learning
  • Object Detection

Image credit to albumentations tutorial:
https://albumentations.ai/docs/getting_started/bounding_boxes_augmentation/
• Data Augmentations for various tasks
  • Classification / Representation Learning
  • Object Detection
  • Keypoint Detection

Image credit to albumentations tutorial: https://albumentations.ai/docs/getting_started/keypoints_augmentation/
Albumentations

• Data Augmentations for various tasks
  • Classification / Representation Learning
  • Object Detection
  • Keypoint Detection
  • Mask Segmentation

Image credit to albumentations tutorial:
https://albumentations.ai/docs/getting_started/mask_augmentation/
import torch
import kornia

frame: torch.Tensor = load_video_frame(...)

out: torch.Tensor = (  
kornia.rgb_to_grayscale(frame)  
)
# compute perspective transform
M = K.get_perspective_transform(points_src, points_dst)

# warp the original image by the found transform
img_warp = K.warp_perspective(img.float(), M, dsize=(h, w))
# create the operator
canny = K.filters.Canny()

# blur the image
x_magnitude, x_canny = canny(data.float())
# create the operator
gauss = K.filters.GaussianBlur2d((11, 11), (10.5, 10.5))

# blur the image
x_blur = gauss(data.float())
# define sharpening mask
sharpen = kornia.filters.UnsharpMask((9, 9), (2.5, 2.5))

# create the sharpened image
sharpened_tensor = sharpen(data)

# get difference between original and sharpened image
difference = (sharpened_tensor - data).abs()
```python
import torch
import torch.nn as nn
import kornia as K

img = load_image(...)  # BxCxHxW

aug = nn.Sequential(
    K.augmentations.ColorJitter(0.15, 0.25, 0.25, 0.25),
    K.augmentation.RandomAffine([-45., 45.], [0., 0.15], [0.5, 1.5], [0., 0.15]),
)

out = aug(img)  # BxCxHxW
```
Reproducibility

Run 1

Run 2

WHAT DO YOU MEAN

"REPRODUCIBILITY"?
Reproducibility

```python
# set random seeds
seed = 42
torch.manual_seed(seed)
random.seed(seed)
np.random.seed(seed)

# use deterministic algorithms only
torch.use_deterministic_algorithms(True)

# use known convolution algorithm in cudnn
torch.backends.cudnn.benchmark = False

# fix workers randomness

def seed_worker(worker_id):
    worker_seed = torch.initial_seed() % 2**32
    numpy.random.seed(worker_seed)
    random.seed(worker_seed)
    g = torch.Generator()
    g.manual_seed(seed)

DataLoader(
    train_dataset,
    batch_size=batch_size,
    num_workers=num_workers,
    worker_init_fn=seed_worker,
    generator=g
)
```

Base on:
Saving & Loading Models

Serialize entire model

torch.save(model, "my_model.pth")
...
model = torch.load("my_model.pth")

Pros: Simple  Cons: Can’t change the model class
            Can’t continue training

# save an object to disk
torch.save(object, path)

# load an object from disk
object = torch.load(path)

Base on:
https://pytorch.org/tutorials/beginner/saving_loading_models.html
Saving & Loading Models

# Define model
class TheModelClass(nn.Module):
    def __init__(self):
        super(TheModelClass, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

        def forward(self, x):
            ...

    # Initialize model
    model = TheModelClass()

Model's state_dict:
conv1.weight torch.Size([6, 3, 5, 5])
conv1.bias torch.Size([6])
conv2.weight torch.Size([16, 6, 5, 5])
conv2.bias torch.Size([16])
fc1.weight torch.Size([120, 400])
fc1.bias torch.Size([120])
fc2.weight torch.Size([84, 120])
fc2.bias torch.Size([84])
fc3.weight torch.Size([10, 84])
fc3.bias torch.Size([10])

Base on:
https://pytorch.org/tutorials/beginner/saving_loading_models.html
Saving & Loading Models

Saving the better way

```python
# save the model’s state dict
torch.save(model.state_dict(), "my_model.pth")

...

# create and load the model’s state dict
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load("my_model.pth"))
```

```python
# save an object to disk
torch.save(object, path)

# load an object from disk
torch.load(path)

# load the state dict to a model
model.load_state_dict(sd)
```

Base on:
https://pytorch.org/tutorials/beginner/saving_loading_models.html
Saving & Loading Models

```python
# Initialize optimizer
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

Optimizer's state_dict:

```python
state  {}
param_groups  [{'lr': 0.001, 'momentum': 0.9, 'weight_decay': 0, ...}
```
Saving & Loading Models

Saving for training

```python
checkpoint = torch.save({'epoch': epoch,
    'model_sd': model.state_dict(),
    'opt_sd': optimizer.state_dict(),
    'loss': loss,
    ...}, 'checkpoint.pth')
```

```python
model = TheModelClass(*args, **kwargs)
optimizer = TheOptimizerClass(*args, **kwargs)

cHECKPOINT = torch.load('checkpoint.pth')
model.load_state_dict(checkpoint['model_sd'])
optimizer.load_state_dict(checkpoint['opt_sd'])
epoch = checkpoint['epoch']
loss = checkpoint['loss']
# continue training
```

Base on:
https://pytorch.org/tutorials/beginner/saving_loading_models.html
YOU GOT IT
Next week:

Implicit 3D Representations
Sources

• Tensorboard
  https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html
• https://medium.com/@iamsdt/using-tensorboard-in-google-colab-with-pytorch-458f9bb95212
• https://towardsdatascience.com/a-complete-guide-to-using-tensorboard-with-pytorch-53cb2301e8c3