PyTorch Tutorial

Reference: https://people.cs.pitt.edu/~kovashka/cs2770_sp20/

Outline

- Introduction
- Training a Model in Pytorch
    1. Create a Model
    2. Load Data
    3. Iterate Over Data and Train Model
- Test the Trained Model in PyTorch

Introduction

What is PyTorch?

- It's a Python-based scientific computing package targeted at two sets of audiences [1]:
- A replacement for NumPy to use the power of GPUs
- A deep learning research platform that provides maximum flexibility and speed

1. https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html

What is Deep Learning?

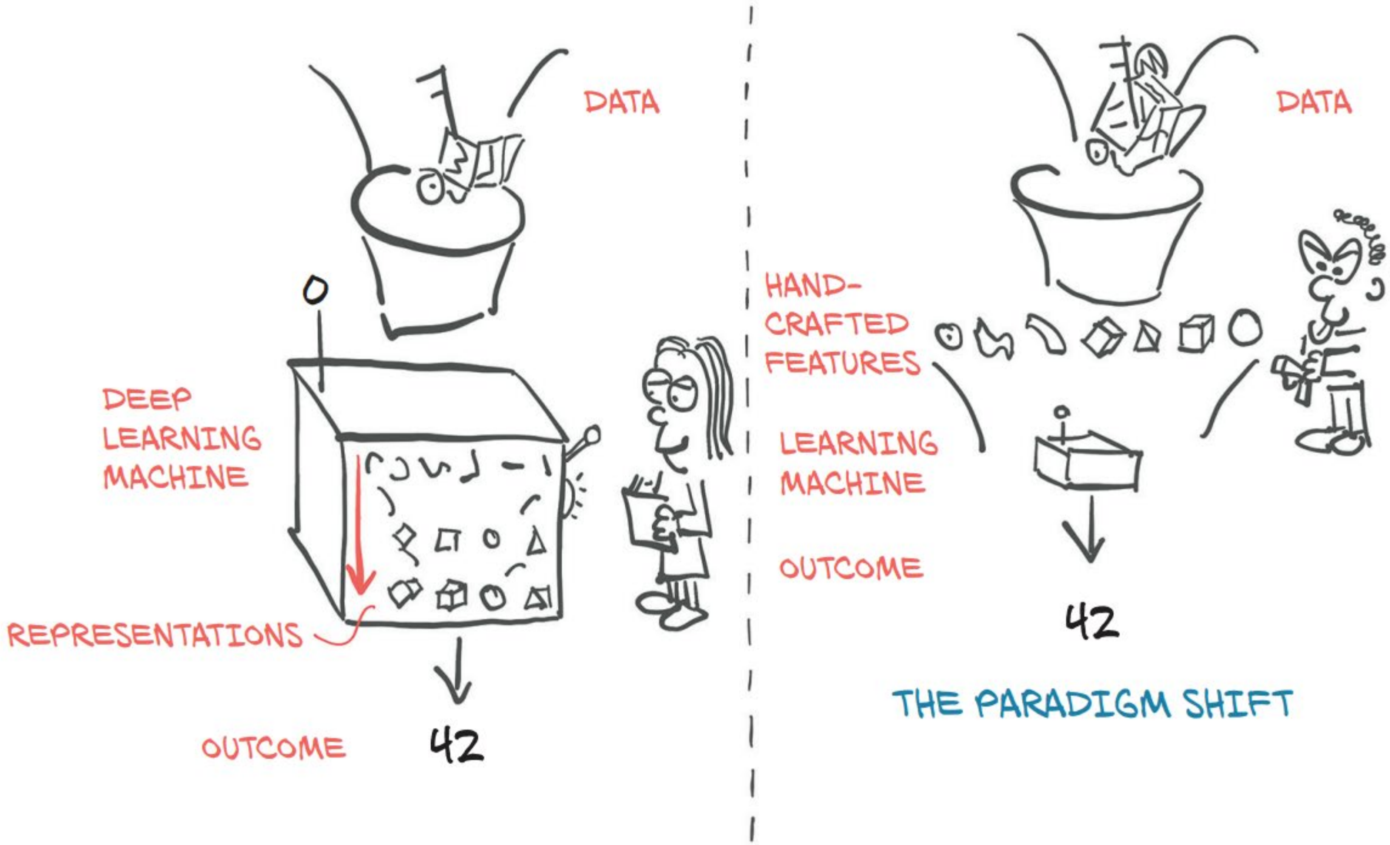


Figure 1.1 **Deep learning exchanges the need to handcraft features for an increase in data and computational requirements.**

Deep Learning with PyTorch, Eli Stevens, Luca Antiga, and Thomas Viehmann

An overview of how PyTorch supports deep learning projects

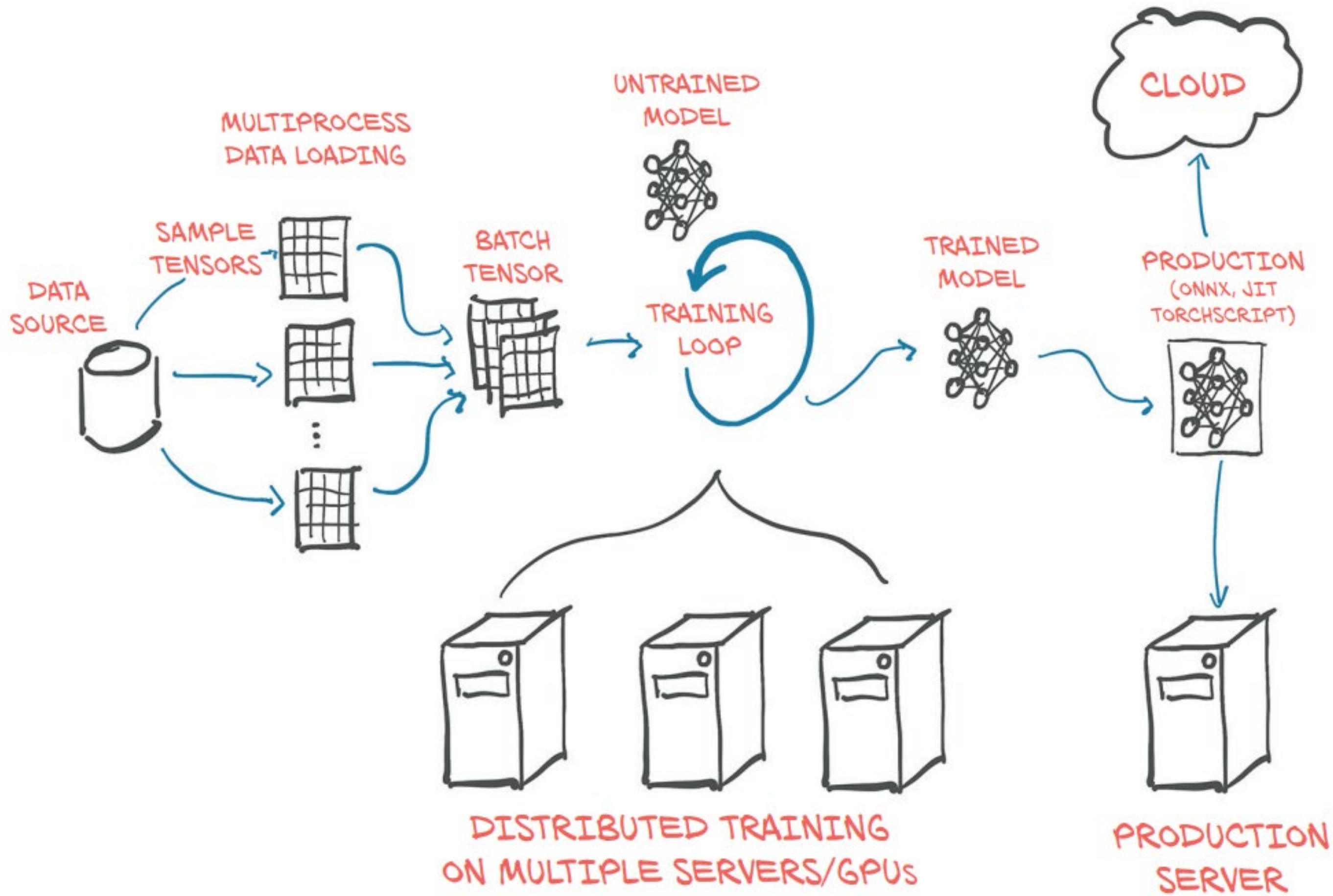


Figure 1.2 Basic, high-level structure of a PyTorch project, with data loading, training, and deployment to production

Deep Learning with PyTorch, Eli Stevens, Luca Antiga, and Thomas Viehmann

What is Tensor in PyTorch?

- A PyTorch Tensor is basically the same as a numpy array: it does not know anything about deep learning or computational graphs or gradients, and is just a generic n-dimensional array to be used for arbitrary numeric computation [1].
- The biggest difference between a numpy array and a PyTorch Tensor is that a PyTorch Tensor can run on either CPU or GPU. To run operations on the GPU, just cast the Tensor to a cuda datatype [1].

1. https://pytorch.org/tutorials/beginner/examples_tensor/two_layer_net_tensor.html

What is Tensor in PyTorch?

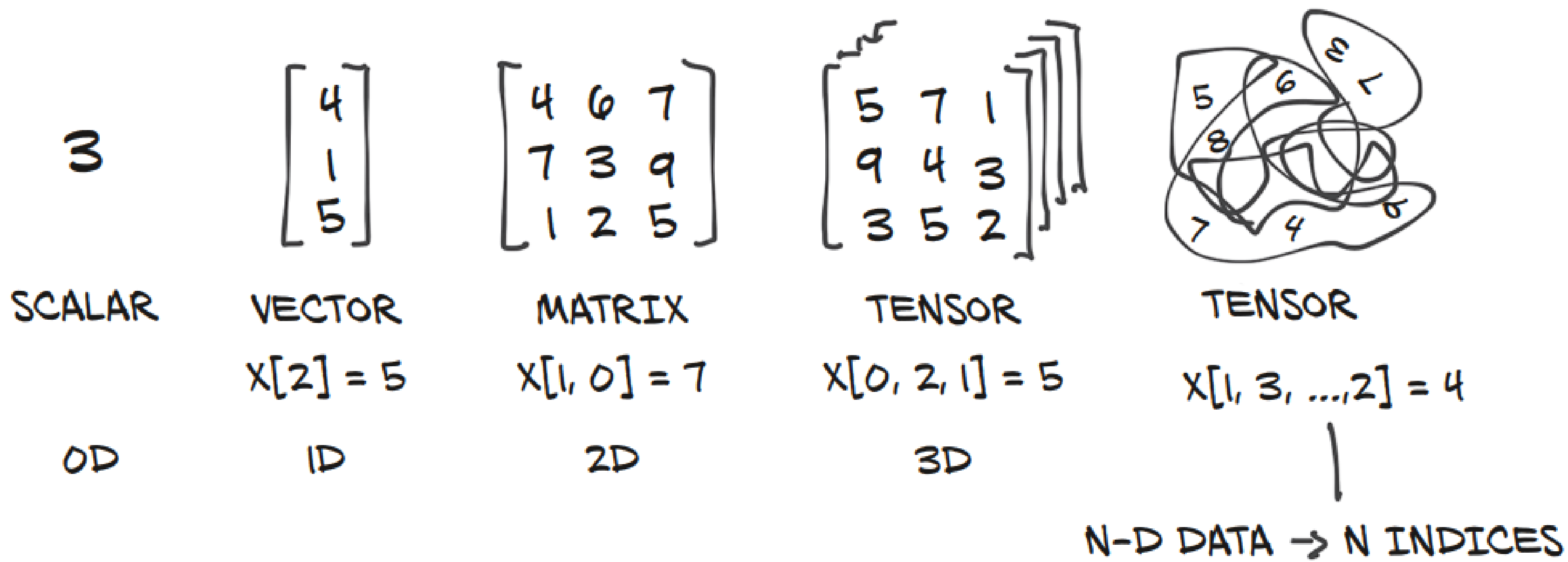


Figure 3.2 **Tensors are the building blocks for representing data in PyTorch.**

Deep Learning with PyTorch, Eli Stevens, Luca Antiga, and Thomas Viehmann

Training a Model in PyTorch

Load Required Classes and Modules

```python
import torch
```
To use the Torch in Python

```python
import torch.nn as nn
import torch.nn.functional as F
```
To create a model by layers

```python
import torch.optim as optim
from torch.optim import lr_scheduler
```
To set the optimization

```python
import numpy as np
```
To manipulate arrays

```python
import torchvision
from torchvision import datasets, models, transforms
```
To Process the data and use the existing Models

```python
import os
import copy
```
To save the best model and get data files

Code Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Data Preprocessing: normalization

- In general , in order to handle noise in data, data can be transformed globally to change the scale or range of data (normalize).[1]

- In Convolutional Neural Network if we don't scale (normalize) the values, the range of different features (e.g. image channels) will be different.[2]

- Since the values are multiplied by learning rate, the features that have **larger scale** might be **over-compensated** and features with **smaller scale** might be **under-compensated**.[2]

1. https://www.coursera.org/lecture/data-genes-medicine/data-normalization-jGN7k
2. https://stats.stackexchange.com/questions/185853/why-do-we-need-to-normalize-the-images-before-we-put-them-into-cnn

More Data Preprocessing

- In addition to the mentioned data preprocessing, there are some transformation that are used mainly for **data augmentation**:
  - transforms.RandomHorizontalFlip()
  - transforms.RandomResizedCrop(224)
- **Data augmentation** is a strategy that enables practitioners to significantly increase the diversity of data available for training models, **without actually collecting new data.**[1]

1. https://bair.berkeley.edu/blog/2019/06/07/data_aug/

Mini Batch and Epoch

- Batch: Number of images which is propagated to a model iteration.
- Epoch: An epoch refers to one cycle through the full training dataset.[1]

```
batch_size = 4
num_epochs = 30
```

- Example:
  - Number of Images = 1024
  - Batch Size = 4
  - Number of Iterations in Every Epoch: 256

1. https://deepai.org/machine-learning-glossary-and-terms/epoch

Load Data and Set Device

Dataset Directory

```
data_dir = 'datasets/hw2'
```

Load Data

```
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x), data_transforms[x])
          for x in ['train', 'val']}


dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4, shuffle=True, num_workers=4)
      for x in ['train', 'val']}
```

```
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}


class_names = image_datasets['train'].classes
```

Get number of images and name of classes

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Set the Device to GPU or CPU

Code Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Sample Network

- Here is an example of a PyTorch model

```
class Sample_Network(nn.Module):
    def __init__(self):
        super(Sample_Network, 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):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

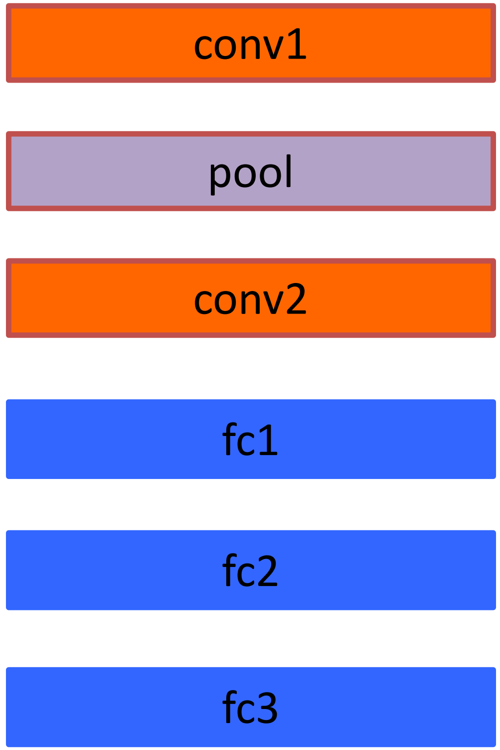
Define the layers of model (1)

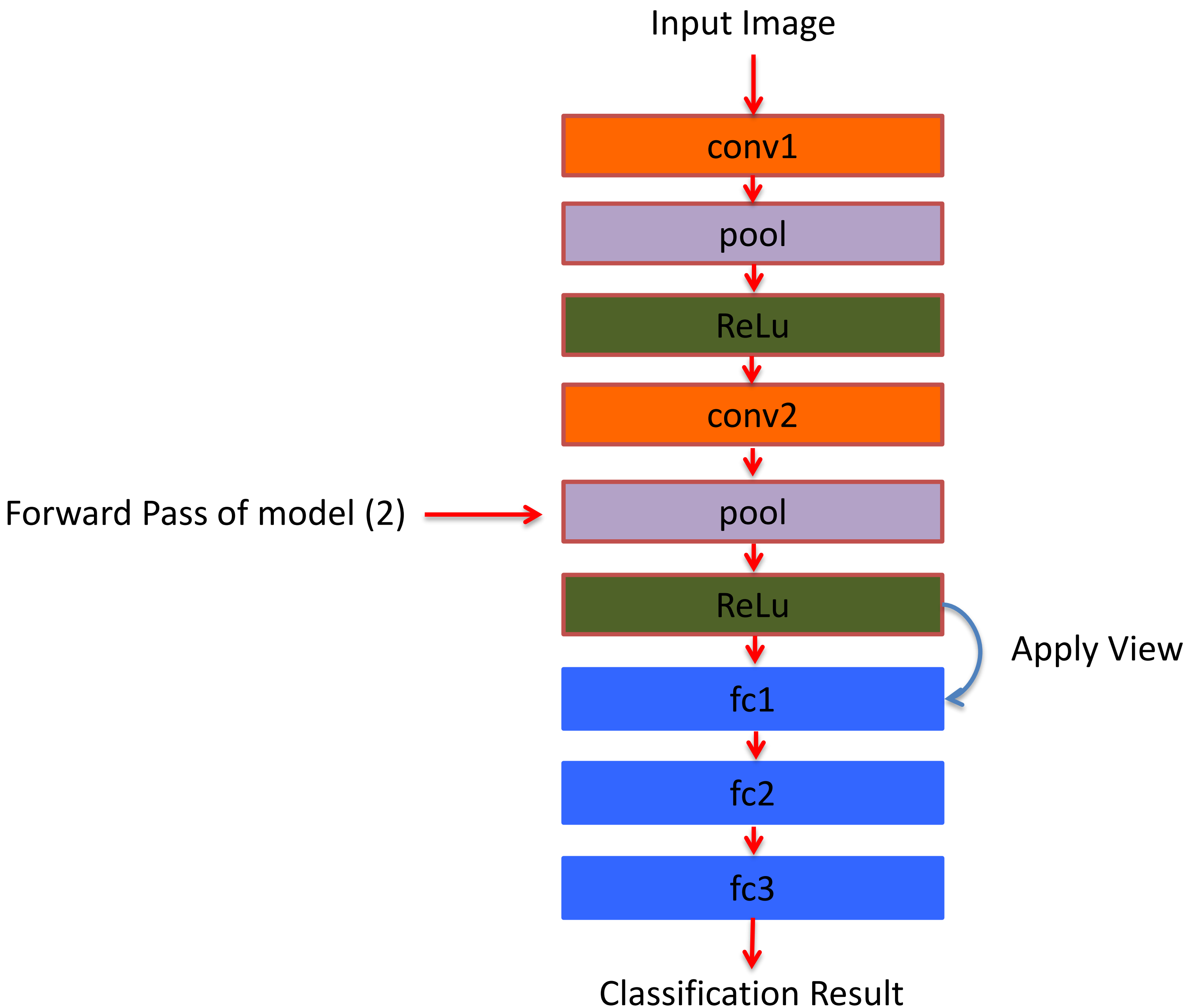
Forward function is called during forward pass (2)

Code Reference:
https://github.com/pytorch/tutorials/blob/master/beginner_source/blitz/neural_networks_tutorial.py

Visualization of Sample Network

Layers which have been
declared in model initialization
(1)




Input Image
conv1
pool
ReLu
conv2
Forward Pass of model (2)
pool
ReLu
Apply View
fc1
fc2
fc3
Classification Result

Before Start Training

- For starting the training process we need to
  1. Initialize an instance from the model which we have already defined
  2. Specify the criterion (loss) for evaluation of model
  3. Specify the setting of optimizer
  4. Specify the way learning rate changes during training

1
```
model = Sample_Network()
```
2
```
criterion = nn.CrossEntropyLoss()
```
3
```
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```
4
```
scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
```

Code Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Save the Best Model Parameter

- We need to train the network for the specified number of epochs.
- Before training process, we save the initial weight as the best model weight and set the best accuracy as zero.
- In every epoch and after finishing the training process, we use the trained model to select the model which has best performance on the validation set.

```
best_model_wts = copy.deepcopy(model.state_dict())
best_acc = 0.0
```

Code Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Iterate Over Train and Validation Sets in every Epoch

- In every epoch we either train the model or just use it for evaluation.
- For training, we need to set the model to **train** mode and for test we need to set to **eval** mode.

```
for phase in ['train', 'val']:
        if phase == 'train':
                model.train()
        else:
                model.eval()
```

Code Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Iterate Over every Minibatch

- We use the data loader which we have created in previous slides to go thorough the data.
- What we get from data loader are tensors for **images (inputs)** and **labels** and we need to transfer them to the device which we have created before.
- Note: Phase here is ‘train’ and ‘test’

```
for inputs, labels in dataloaders[phase]:
        inputs = inputs.to(device)
        labels = labels.to(device)
```

Code Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Prediction and Back Propagation

```
optimizer.zero_grad()
```

Zero the gradient before start of a new mini batch

```
outputs = model(inputs)
```

Apply Forward Function and get logit

```
_, preds = torch.max(outputs, 1)
```

Get the highest logic as prediction

```
loss = criterion(outputs, labels)
```

Compute the loss based on predicted value

```
if phase == 'train':
        loss.backward()
        optimizer.step()
```

Back propagate if we are in train phase

```
running_loss += loss.item() * inputs.size(0)
```

Sum the loss of batch with all loss values

```
running_corrects += torch.sum(preds == labels.data)
```

Sum correctly predicted values in batch with all loss values

Code Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Finish Iterating over Data in One Epoch

- When iteration over all data finished then we need to compute the loss and save the best model.

Scheduler setting (e.g. learning rate) needs to be updates

```
if phase == 'train':
    scheduler.step()
```

Loss and accuracy needs to be computed at the end of epoch

```
epoch_loss = running_loss / dataset_sizes[phase]
epoch_acc = running_corrects.double() / dataset_sizes[phase]
```

```
if phase == 'val' and epoch_acc > best_acc:
    best_acc = epoch_acc
    best_model_wts = copy.deepcopy(model.state_dict())
    torch.save(best_model_wts , 'best_model_weight.pth')
```

Save the best model

Code Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Test on the Best Model Weight

Load Data for Test

Transform the test images

```
data_transforms = {
'test': transforms.Compose([
        transforms.Resize(256),
        transforms.ToTensor()
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
]),
}
```

```
data_dir = 'datasets/hw2'


image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x), data_transforms[x])
                  for x in ['test']}

dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4, shuffle=True, num_workers=4)
              for x in ['test']}

dataset_sizes = {x: len(image_datasets[x]) for x in ['test']}
```

Load the data and get the dataset size

Code Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Test the Loaded Data

```
model.eval()
```

Set the model in evaluation mode

```
phase = 'test'

for inputs, labels in dataloaders[phase]:
        inputs = inputs.to(device)
        labels = labels.to(device)


        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        loss = criterion(outputs, labels)


        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels.data)
```

Iterate over test data and compute loss and correctly predicted values

```
test_loss = running_loss / dataset_sizes[phase]
test_acc = running_corrects.double() / dataset_sizes[phase]
```

Compute the loss and Accuracy over all data

Code Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html