AI Summer School 2025

Medical Imaging Informatics

University of Pittsburgh

Introduction to PyTorch for Medical Imaging

Instructor: Nick Littlefield, MS

Pitt SHRS
School of Health and Rehabilitation Sciences

Computational Pathology & AI Center of Excellence

IEEE COMPUTER SOCIETY

Learning Objectives

After completing this lecture, you should be able to:

- Explain what PyTorch is and why it is used
- Understand what a tensor is and its properties
- Understand how to work with data in PyTorch using Datasets and DataLoaders
- Explain why GPUs are an important component of deep learning
- Implement and understand how to train and evaluate a neural network using PyTorch

Outline

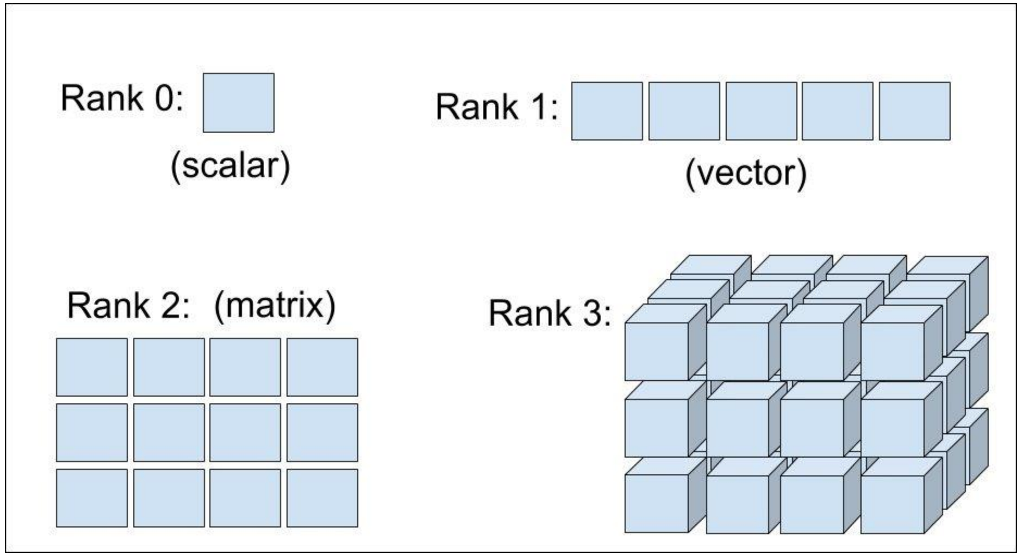
- PyTorch: What and Why?
- Tensors
- Working with Data
- nn Module
- GPUs
- Training and Evaluating a Neural Network
- Hands-on Practice

PyTorch: What and Why?

- **PyTorch** is an easy-to-use, yet powerful Python deep learning library used for applications in computer vision and natural language processing.
- Highly flexible, efficient and scalable, designed to minimize the number of computations required, and be compatible with different varieties of hardware architectures.
- Developed by Facebook’s AI Research Group

Tensors: The Data Structure of Deep Learning

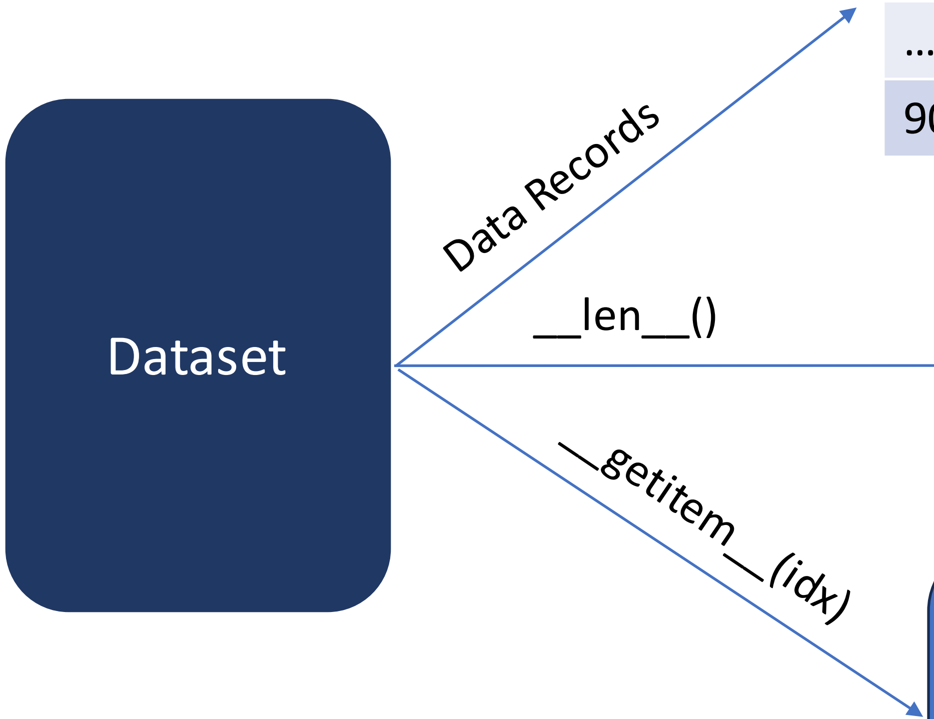
- Tensors are fundamental data structures that efficiently perform mathematical operations on large sets of data
- Can be ***n*-dimensional**
- Three key components:
  - **Shape**: the size of the tensor
  - **Data type**: type of data stored in the tensor
  - **Device:** the device where the tensor is stored (CPU or GPU)



Working with Data

- PyTorch provides functionality for loading training and test data efficiently
- Two essential components:
  - Datasets
  - Data Loaders
- **Dataset:** Provides a uniform interface to access training/testing data
  - __getitem__: returns the i-th data record in the dataset
  - __len__: returns the size of the dataset
  - Usually defined by us, unless we are using a predefined dataset
- **Data Loader:** Efficiently loads and stacks data from the dataset into batches. We define the parameters
  - batch_size: Number of samples
  - shuffle: Shuffle the dataset into random order

Working with Data: Datasets

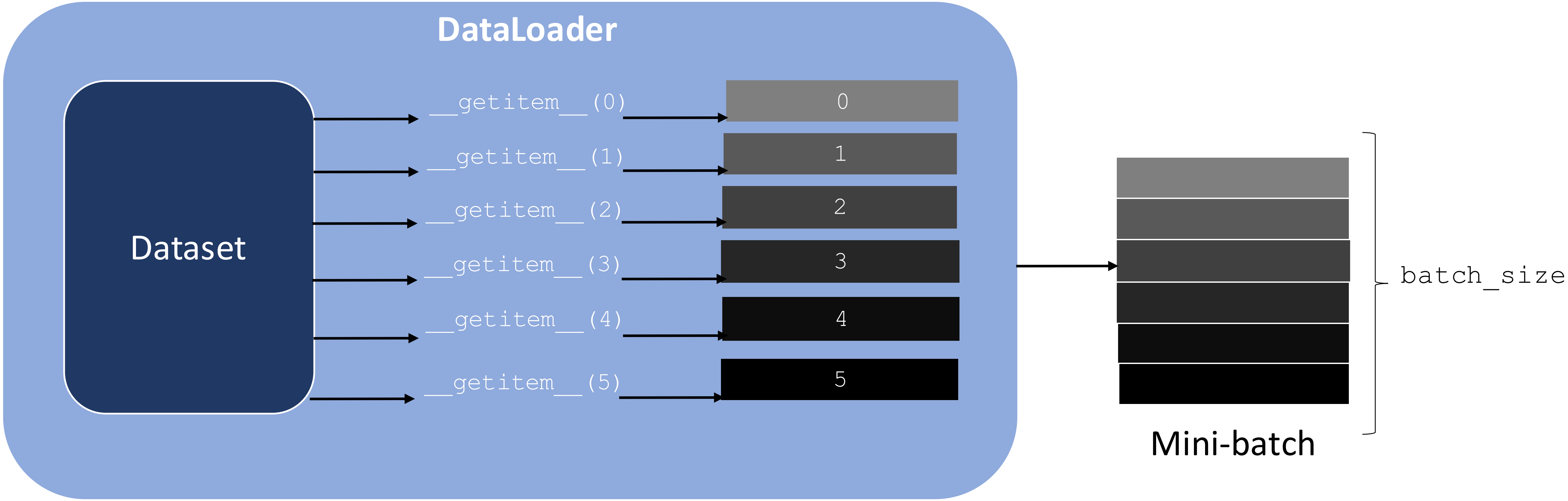


Patient ID	Sex	Race	Age
9002116	Male	Black/AA*	61
…	…	…	…
9003126	Female	White/Caucasian	63

```
def __len__(self):
    return len(self.data_records)
```

```
def __getitem__(self, idx):
   img = self.images[idx]
   label =  self.labels[idx]
   img =
np.array(Image.open(img).convert("RGB"))    if
self.transforms is not None:
       img =
self.transforms(image=img)["image"]        return
img.float(), torch.tensor(label).float()
```

Working with Data: Dataloaders



nn Module: Defining Neural Networks

- Provides functionality for creating neural networks
- Contains collections of different layers, activation functions, and loss functions
- We define both the structure of the network and the `forward()` function to define the way the computation is done

```python
class SimpleClassifier(nn.Module):

    def __init__(self, num_inputs, num_hidden, num_outputs):
        super().__init__()
        # Initialize the modules we need to build the network
        self.linear1 = nn.Linear(num_inputs, num_hidden)
        self.act_fn = nn.Tanh()
        self.linear2 = nn.Linear(num_hidden, num_outputs)

    def forward(self, x):
        # Perform the calculation of the model to determine the prediction
        x = self.linear1(x)
        x = self.act_fn(x)
        x = self.linear2(x)
        return x
```

```python
model = SimpleClassifier(num_inputs=2, num_hidden=4, num_outputs=1)
# Printing a module shows all its submodules
print(model)
```

```
SimpleClassifier(
  (linear1): Linear(in_features=2, out_features=4, bias=True)
  (act_fn): Tanh()
  (linear2): Linear(in_features=4, out_features=1, bias=True)
)
```

torchvision

- Includes a variety of popular pretrained neural networks architectures for computer vision tasks such as classification, segmentation, and object detection

Object Detection

The following object detection models are available, with or without pre-trained weights:

- Faster R-CNN
- FCOS
- RetinaNet
- SSD
- SSDlite

Classification

The following classification models are available, with or without pre-trained weights:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MaxVit
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

GPUs

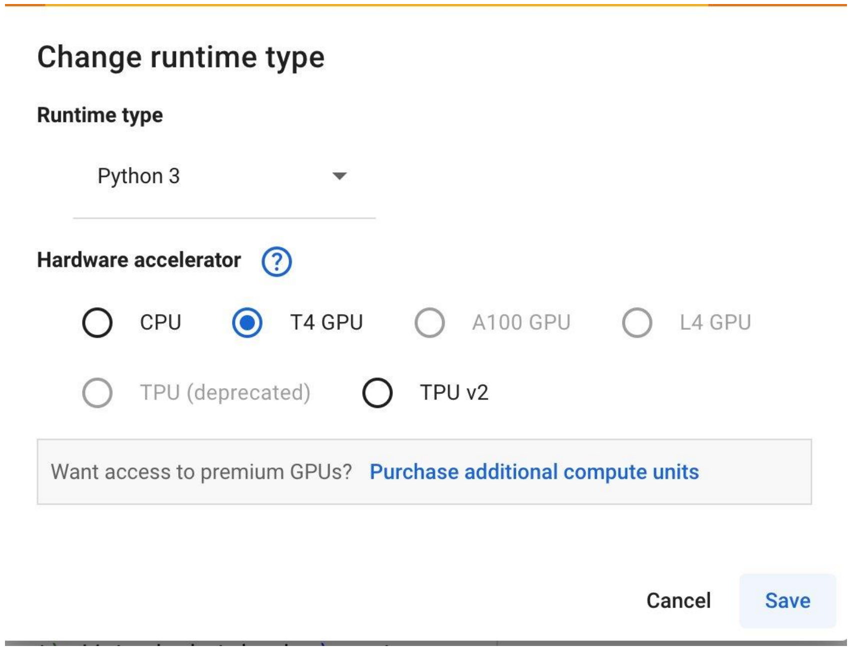
- Training neural networks require a lot of memory
- **Graphics Processing Units (GPUs)** can speed up computations and provide the memory needed
- PyTorch provides ways to transfer data from the CPU to a GPU.
- Steps:
  - Determine whether a GPU is available and get the device name: `torch.cuda.is_available()`
  - Transfer model to GPU
  - During training transfer the data to the GPU

```
# Set device for training
device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
# Move model to device
model = model.to(device)
```

GPUs

- GPUs are available in Google Colab
- In a Colab session:
  - `Select` **`Runtime > Change Runtime Type > T4 GPU > Save`**



Training Neural Networks: Key Components

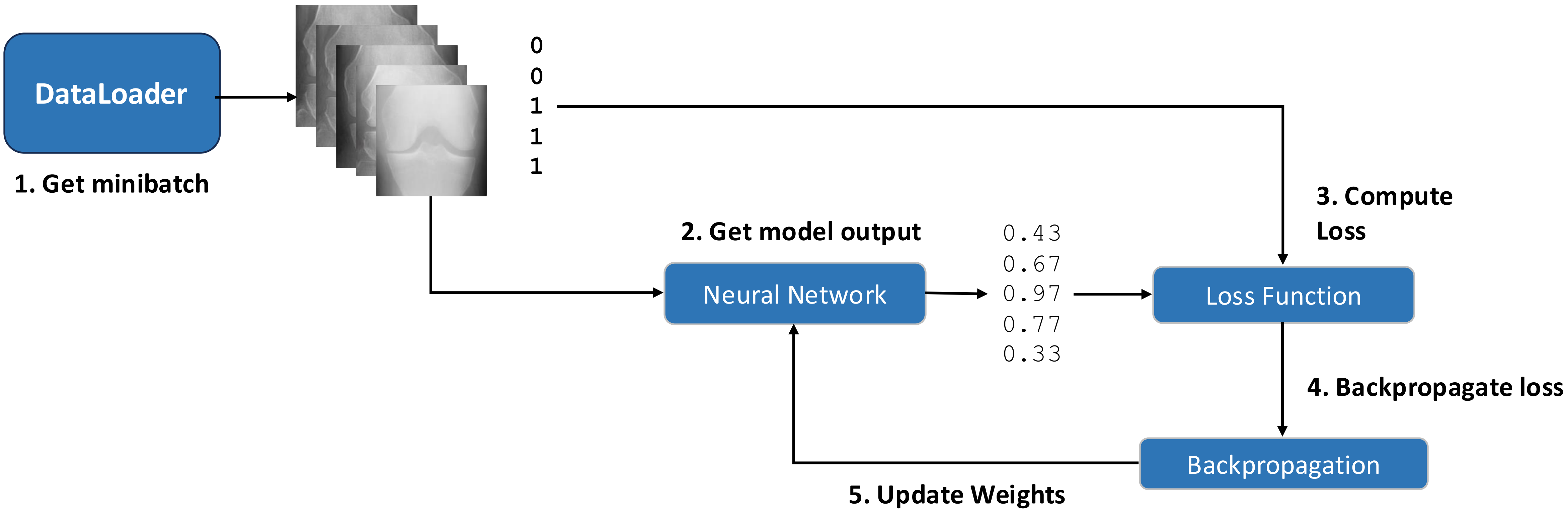
- **Model**: Custom or predefined torchvision model
- **Loss Function**: Guides the models learning process
  - BCELoss/BCELossWithLogits (Binary output), CrossEntropyLoss (Multiple classes)
- **Optimizer**: Controls how the models' parameters are updated during training
  - SGD (Stochastic Gradient Descent), Adam
- **Learning Rate:** Controls how much the models' parameters are adjusted based on the gradient of the loss
- **Epochs:** Controls how long the model is trained for (number of passes through the dataset)
- **Evaluation Metrics**: Assess the models performance
  - Accuracy, Precision, Recall, Intersection over Union (IoU)

Training Neural Networks: Steps

1. Get training batch from the data loader
2. Obtain predictions from the model for the batch
3. Calculate the loss between the predictions and the actual labels
4. Backpropagation
5. Update the model parameters
6. Evaluate on validation (if available)

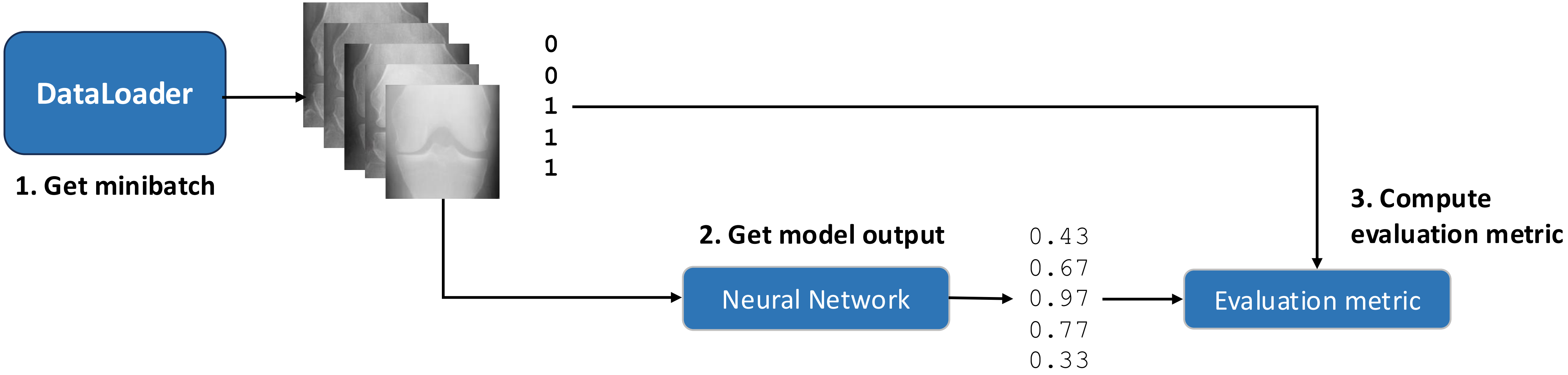
Training Neural Networks: Steps

For *n* epochs:



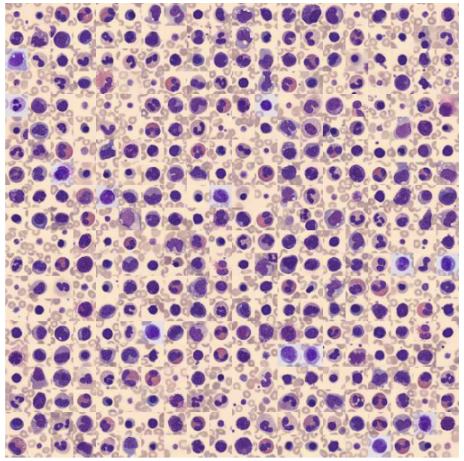
Evaluating a Neural Networks: Validation Steps

1. Get a batch of test data (unseen by the model)
2. Get the predictions from the model
3. Calculate the evaluation metric (accuracy, F1-Score, IoU)



Hands-On Practice: Blood Cell Classification

Facts of **BloodMNIST**



Data Modality: Blood Cell Microscope

Task: Multi-Class (8)

Number of Samples: 17,092 (11,959 / 1,712 / 3,421)

Source Data:

Andrea Acevedo, Anna Merino, et al., "A dataset of microscopic peripheral blood cell images for development of automatic recognition systems," Data in Brief, vol. 30, pp. 105474, 2020.

Thank you!

Questions!