# Introduction to Deep Learning (I2DL)

## Tutorial 7: PyTorch

# Today's Outline

- **Exercise 6 Recap**
  - analyze the best submission
- **PyTorch**
  - And other libraries
- **Exercise 7**

# Exercise 6 Recap

# Our Leaderboard

## Leaderboard

The leaderboard shows for each exercise the highest scoring submission from each user. Only valid submissions are displayed.

Exercise 1 | Exercise 3 | Exercise 4 | Exercise 5 | Exercise 6 | Exercise 7 | Exercise 8 | Exercise 9 | Exercise 10 | Exercise 11 | Exercise 12

| # | User | Score |
|---|---|---|
| **1** | u0973 | 93.92 |
| **2** | u1831 | 91.04 |
| **3** | u0735 | 84.96 |
| **4** | u1244 | 84.59 |
| **5** | u0784 | 82.41 |
| **6** | u1314 | 78.80 |
| **7** | u0271 | 77.08 |
| **8** | u0555 | 75.44 |
| **9** | u1355 | 71.77 |
| **10** | u1258 | 69.12 |

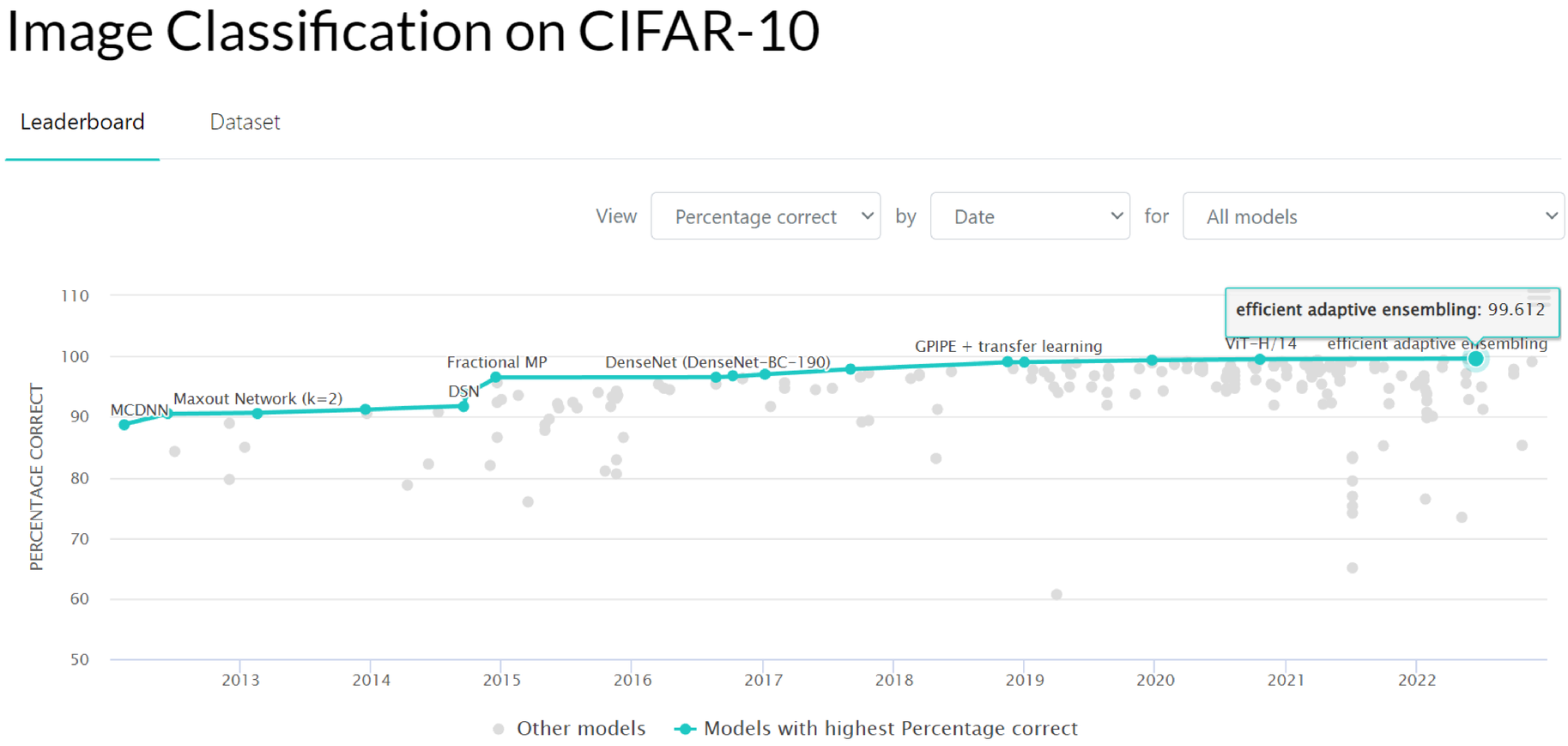
# The Best Submission (out of scope)

- **PyTorch ResNet-32 — wide (base width 32), a convolutional residual net**
  - 3×3 stem, then 3 stages of 5 residual blocks, BN + ReLU, option-A identity shortcuts, Kaiming init
  - Beyond Exercise 6's NumPy fully-connected scope
- **Trained 600 epochs** — SGD + Nesterov, weight decay 5e-4, LR 0.1 → cosine, batch 128
- **Data augmentation**
  - Random crop (pad 4) + horizontal flip + Cutout (8px)
- **93.92% on our evaluation set**

# The Best In-Scope Submission

- **62.01% — exactly what Exercise 6 intended (pure NumPy, no PyTorch, no CNN)**
  - Used the default NumPy fully-connected `ClassificationNet` as-is — no custom network
- **One scaled-up `random_search` — 400 configs × 25 epochs on the full dataset**
  - `hidden_size`: {512, 1024}
  - `learning_rate`, `reg`: log [1e-5, 1e-1]
  - `lr_decay`: [0.95, 1]
  - `num_layer` = 3, LeakyReLU (fixed)

# State of the Art on CIFAR-10

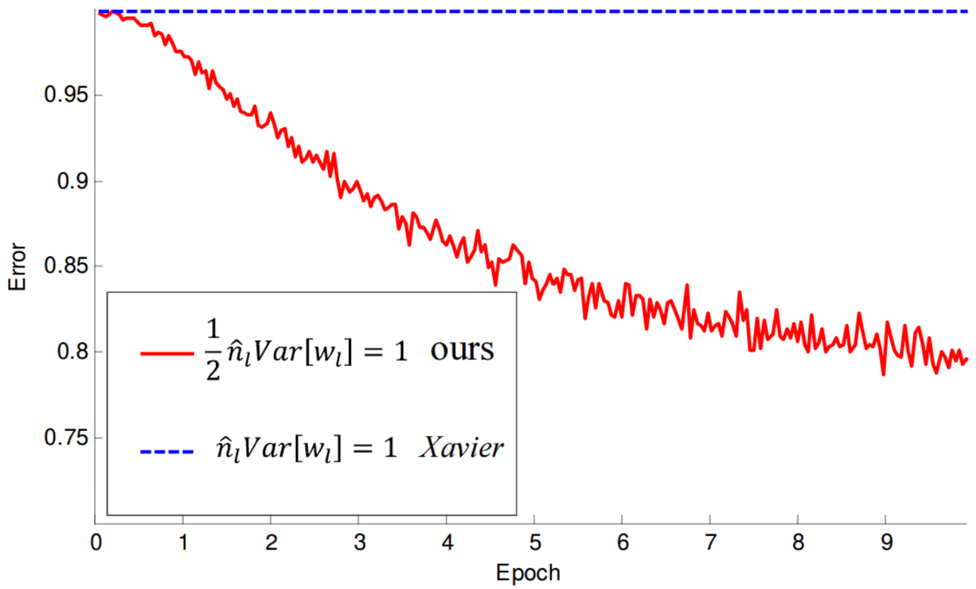


# Some Limiting Factors

| Problem | What helps |
|---|---|
| Computational power and/or time | **PyTorch** → GPU support |
| Specialized architectures | **CNNs** → Lecture 9 |
| More knowledge<br>*(e.g., proper initialization)* | Lectures |

# Lecture Recap: Initialization

- **Lecture**
  - Network weights shouldn't only be randomly initialized.
  - They should be tailored to our activation function.



# PyTorch

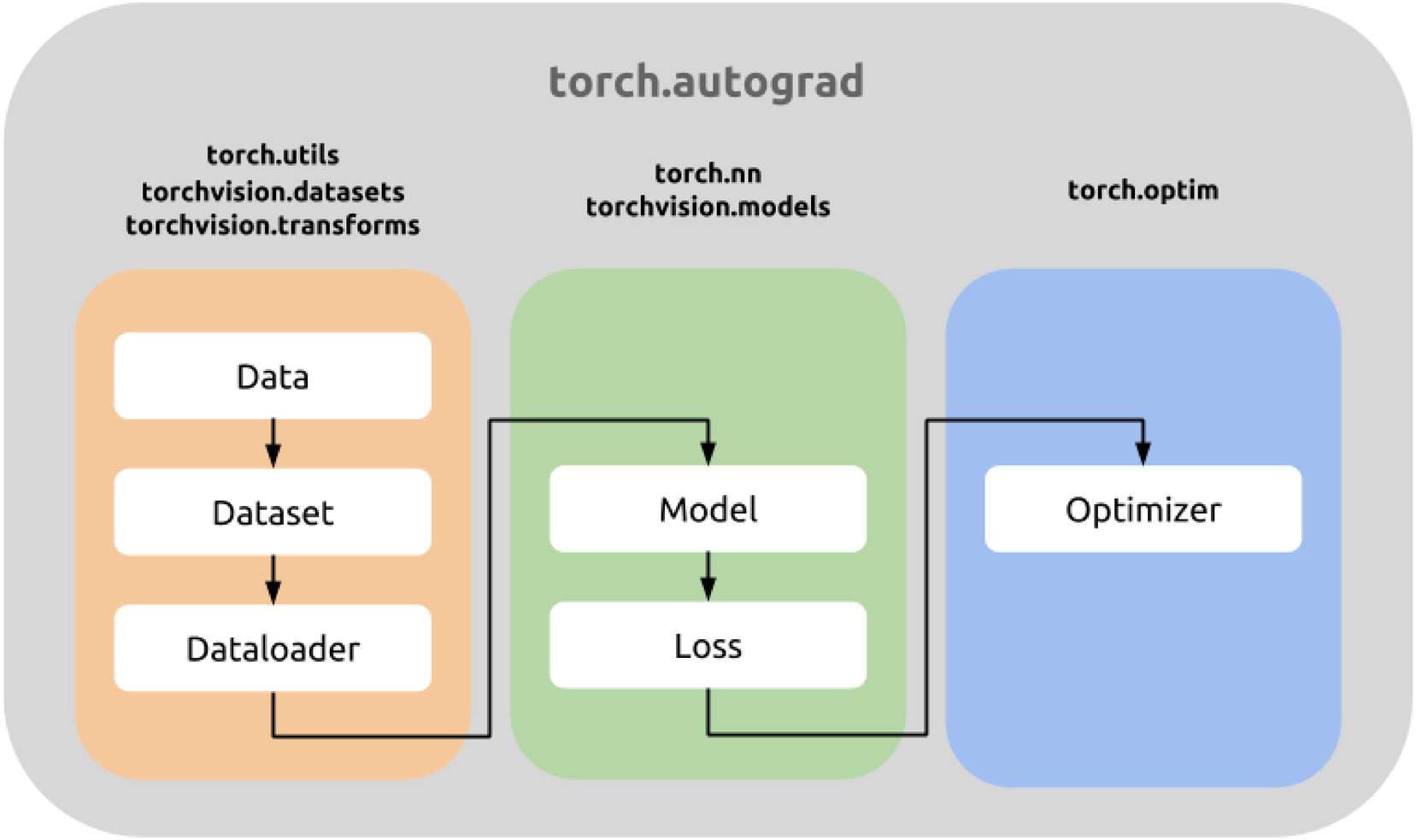
# Exercise Overview

| | |
|---|---|
| Exercise 01: Organization<br>Exercise 02: Math Recap | **Intro** |
| Exercise 03: Dataset and Dataloader<br>Exercise 04: Solver and Linear Regression<br>Exercise 05: Neural Networks<br>Exercise 06: Hyperparameter Tuning | **NumPy (Reinvent the wheel)** |
| **Exercise 07: Introduction to PyTorch**<br>Exercise 08: Autoencoder | **PyTorch / Tensorboard** |
| Exercise 09: Convolutional Neural Networks<br>Exercise 10: Semantic Segmentation<br>Exercise 11: Transformers (1/2)<br>Exercise 12: Transformers (2/2) | **Applications (Hands-off)** |

# Deep Learning Frameworks

- **We will use PyTorch**
  - The most widely used deep learning framework, in both academia and industry
- **Other popular frameworks**
  - TensorFlow
  - JAX


# PyTorch: Overview

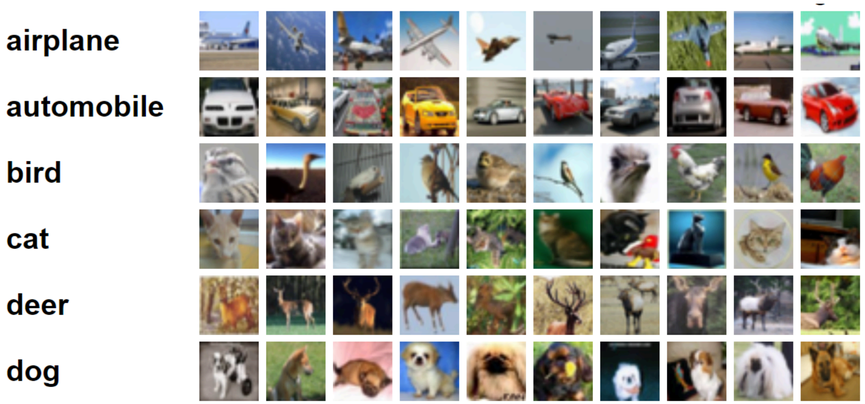


# Some Key Features

- Simple device management — move tensors and modules to GPU with one call.
- **Implementations of**
  - Optimizers (SGD, Adam, ...), schedulers.
  - Datasets and transforms via *torchvision*.
  - Automatic gradients — *autograd*.

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)

print(f"Original device: {x.device}") # "cpu", integer

tensor = x.to(device)
print(f"Current device: {x.device}") #"cpu" or "cuda", double
```

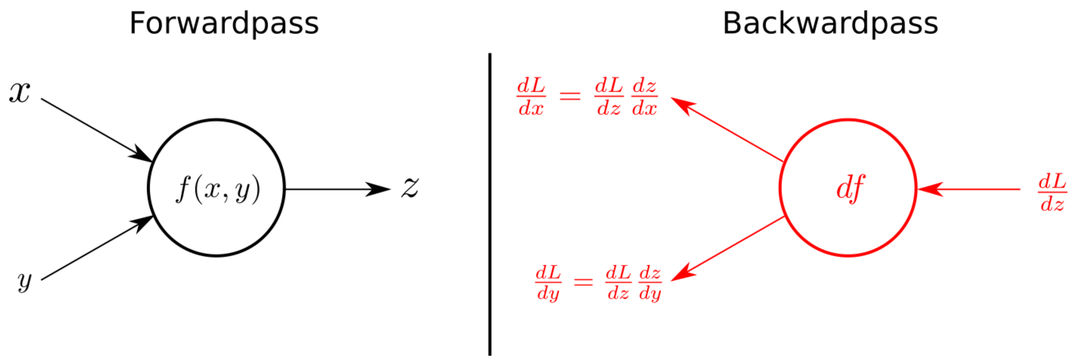
```
cpu
Original device: cpu
Current device: cpu
```



# Easy Network Creation

> Where is the backward pass?

```
import torch.nn as nn
# defining the model
class Net(nn.Module):
    def __init__(self, input_size=1*28*28, output_size=100):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, output_size)

    def forward(self, x):
        x = self.fc1(x)
        return x
net = Net()
net = net.to(device)
```

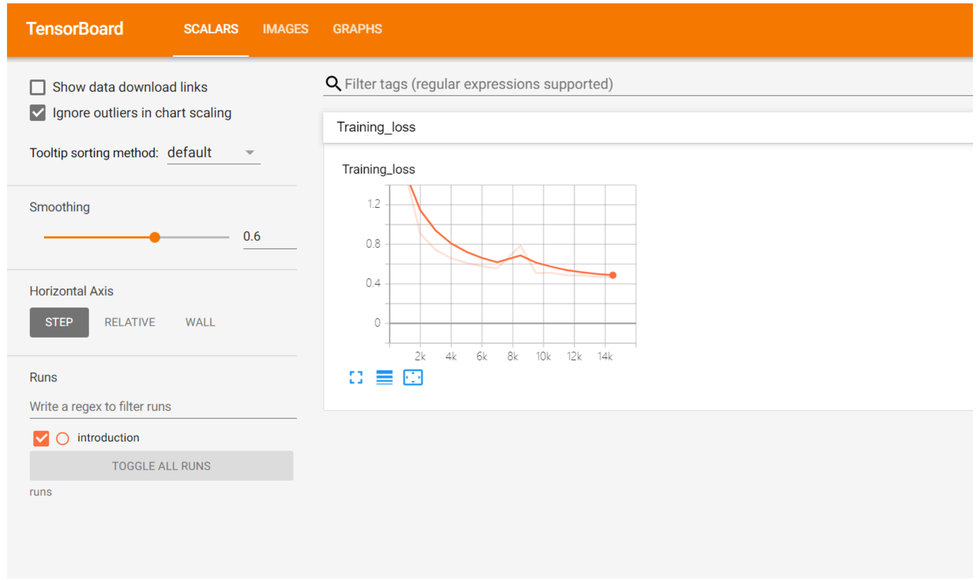


# References on PyTorch

- **Repository:** github.com/pytorch/pytorch
- **Examples (recommendation):** github.com/pytorch/examples
- **PyTorch for NumPy users:** github.com/wkentaro/pytorch-for-numpy-users
- Look up your own and share!

# Tensorboard (also in PyTorch)

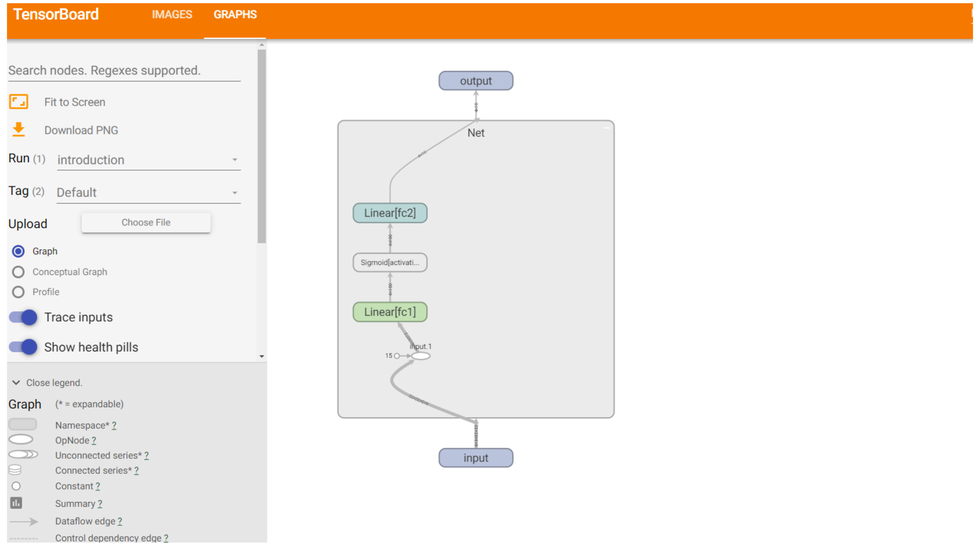
- Directly access Tensorboard from inside your training loop.
- Tensorboard generates the graph, timestamps, and other metadata for you.



# Visualize Networks

> Graph creation needs the network and one batch.

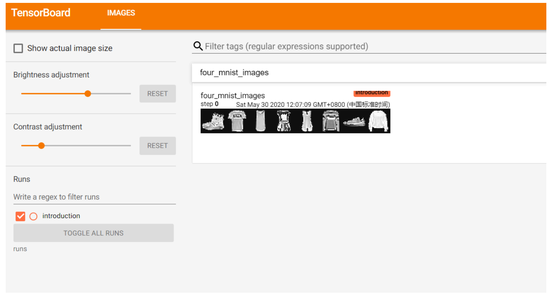
- With a single forward pass, Tensorboard can map and display your network graph.

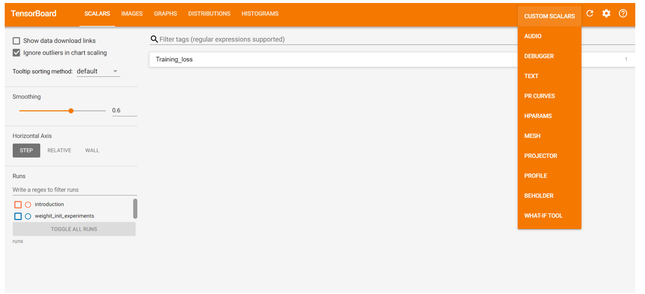


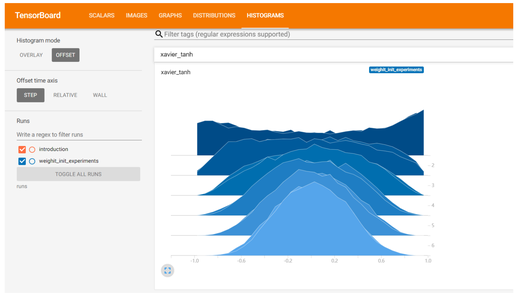
# In Short: Document Everything!

> Everything is saved "automatically".

- Scalars, sample images, weight histograms — all logged from a few lines.
- Filter runs, smooth curves, and compare experiments side by side.

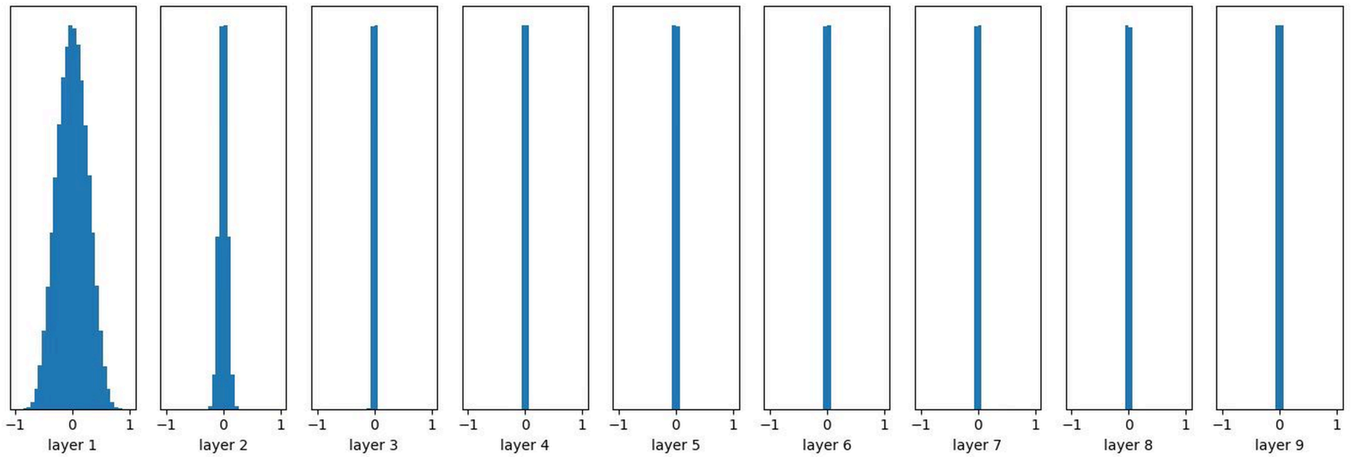


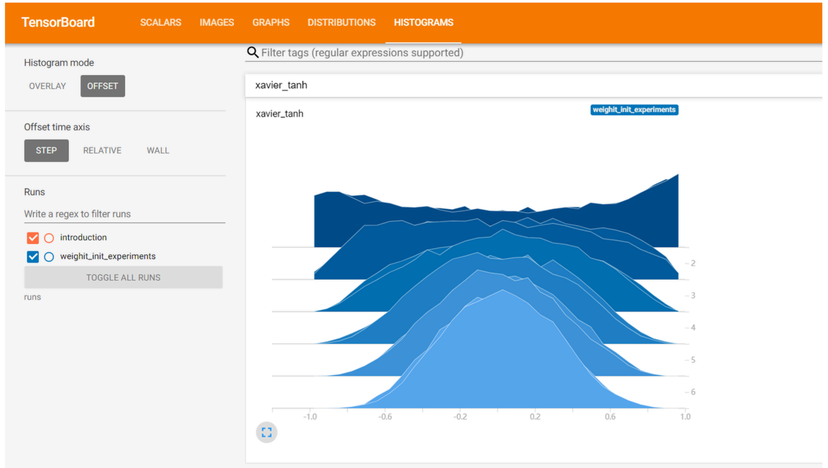



# Example: Weight Initialization

- Histogram visualization for layer outputs can show off effects of weight initialization as shown in the lecture



# More Abstraction: PyTorch Lightning

- **Classify our code into three categories.**
  - **Research** code — the exciting part. Changes with new tasks, models, ideas. → LightningModule
  - **Engineering** code — the same for every project and model. → Trainer
  - **Non-essential** code — logging, organising runs. → Callbacks

# Lightning Module

- **Methods to implement:** `__init__`, `forward`, `training_step`, `configure_optimizers`.

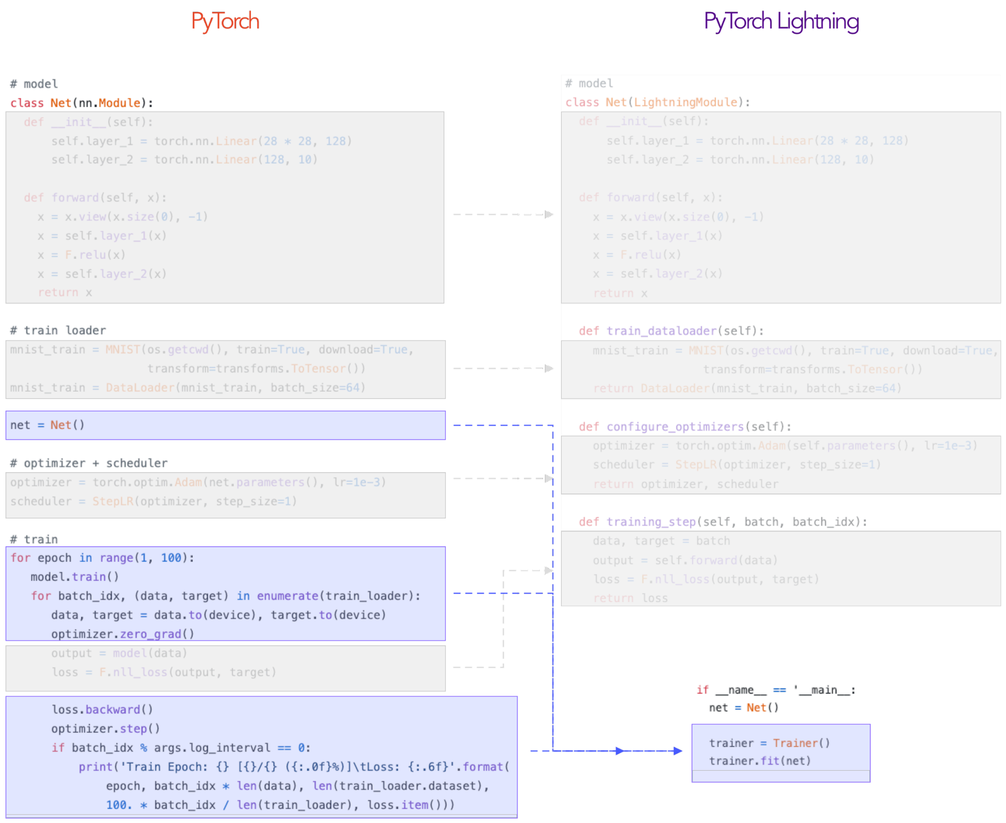
PyTorch

```
# model
class Net(nn.Module):
  def __init__(self):
      self.layer_1 = torch.nn.Linear(28 * 28, 128)
      self.layer_2 = torch.nn.Linear(128, 10)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    x = self.layer_1(x)
    x = F.relu(x)
    x = self.layer_2(x)
    return x

# train loader
mnist_train = MNIST(os.getcwd(), train=True, download=True,
                    transform=transforms.ToTensor())
mnist_train = DataLoader(mnist_train, batch_size=64)

net = Net()

# optimizer + scheduler
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
scheduler = StepLR(optimizer, step_size=1)

# train
for epoch in range(1, 100):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)

        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

PyTorch Lightning

```
# model
class Net(LightningModule):
  def __init__(self):
      self.layer_1 = torch.nn.Linear(28 * 28, 128)
      self.layer_2 = torch.nn.Linear(128, 10)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    x = self.layer_1(x)
    x = F.relu(x)
    x = self.layer_2(x)
    return x

def train_dataloader(self):
    mnist_train = MNIST(os.getcwd(), train=True, download=True,
                        transform=transforms.ToTensor())
    return DataLoader(mnist_train, batch_size=64)

def configure_optimizers(self):
    optimizer = torch.optim.Adam(self.parameters(), lr=1e-3)
    scheduler = StepLR(optimizer, step_size=1)
    return optimizer, scheduler

def training_step(self, batch, batch_idx):
    data, target = batch
    output = self.forward(data)
    loss = F.nll_loss(output, target)
    return {'loss': loss}
```

# Lightning Trainer



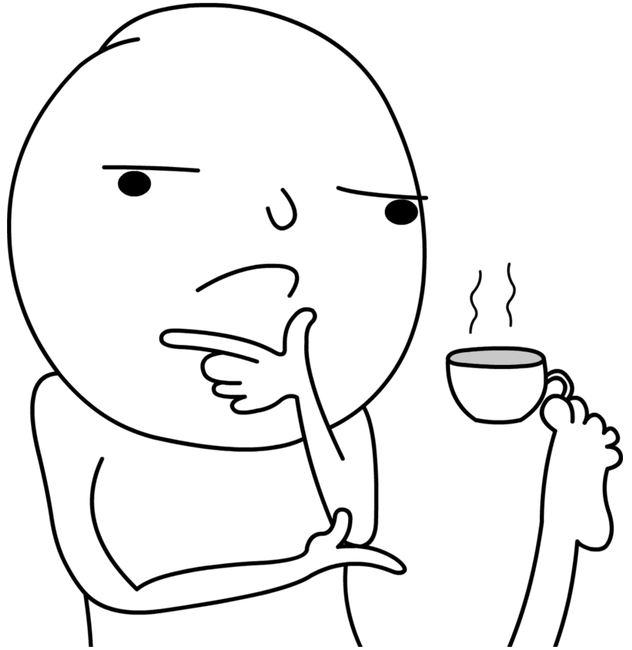

## Trainer

- Initialise the model with hyperparameters (e.g. a dict).
- The Trainer contains all the code relevant for training your network.
- Call **.fit()** to train.

That is all you need.

# What to Use? Your Call

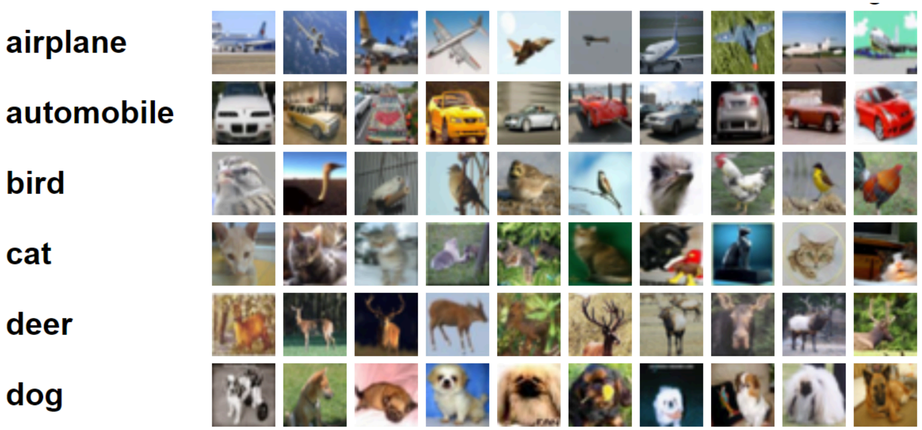
- **Advantages**
  - Better overview of the relevant code.
  - Nice debugging features.
  - Many automated options, like logging.
- **Potential problems**
  - Can have issues like any stock library.
  - Not always straightforward to add features yourself.



# Exercise 7

# CIFAR-10… Again

- **Task:** CIFAR-10 classification — *but now in PyTorch.*
- **What is new**
  - More knowledge from Lecture 7.
  - You can use everything you have learned, but **no** convolutional layers, transformers, or pre-trained networks.
  - Filesize and parameter limit are enforced.



# So… Tuning Again?

- **Make sure to**
  - Get into PyTorch — read the docs and the source code.
  - Improve upon your previous submission.
  - Pick hyperparameters well.

# See You Next Week!