A Union of Scikit-learn and PyTorch

Thomas J Fan - @thomasjpfan

github.com/thomasjpfan/skorch_talk

scikit
learn

SciKit-Learn API

```python
clf = SGDClassifier(alpha=0.01)

clf.fit(X, y)

y_pred = clf.predict(X)

clf.partial_fit(X, y)

clf.set_params(alpha=0.1)
```

PyTorch

PyTorch Training - Training

```python
for epoch in range(10):
    net.train()
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        with torch.set_grad_enabled(True):
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```

PyTorch Training - Recording Metrics

```
train_losses = []
for epoch in range(10):
    running_loss = 0.0
    for inputs, label in train_loader:
        ...
        running_loss += loss.item() * inputs.size(0)
    epoch_loss = running_loss / len(train_loader.dataset)
    train_losses.append(epoch_loss)
```

PyTorch Training – Validation

```
net.eval()

with torch.set_grad_enabled(False):
    for data in valid_loader:
        inputs, labels = data
        outputs = net(inputs)
        loss = criterion(outputs, labels)
```

PyTorch Training – The Rest

- Recording validation losses
- Saving the best performing model
- Recording other metrics
- Logging
- …

SKORCH

1. Scikit-Learn compatible neural network library that wraps PyTorch.
2. Abstracts away the training loop.
3. Reduces the amount of boilerplate code with callbacks.

Skorch NeuralNet

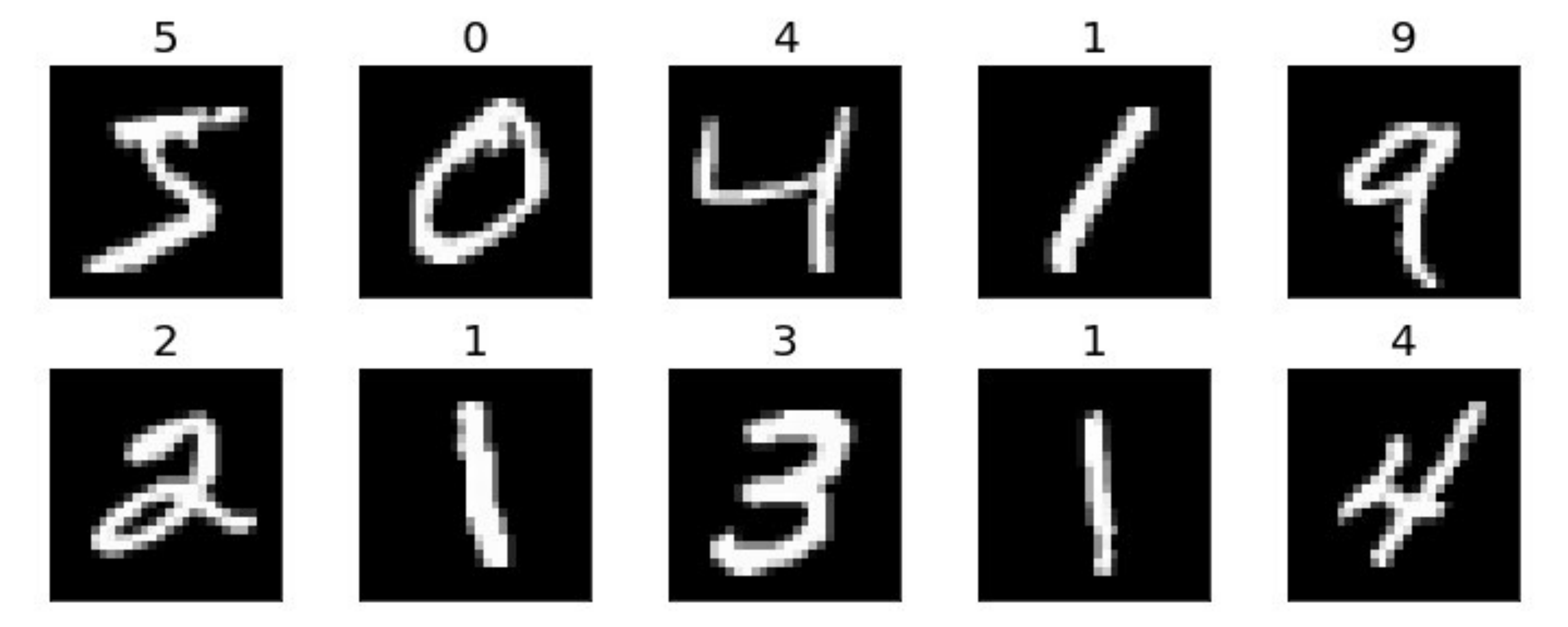
SKORCH

```
from skorch import NeuralNet

net = NuetralNet(
     module,
     criterion= ... ,
     callbacks=[ ... ])
```

Exploring Skorch's API

1. MNIST
2. Ants and Bees
3. 2018 Kaggle Data Science Bowl

MNIST - Data



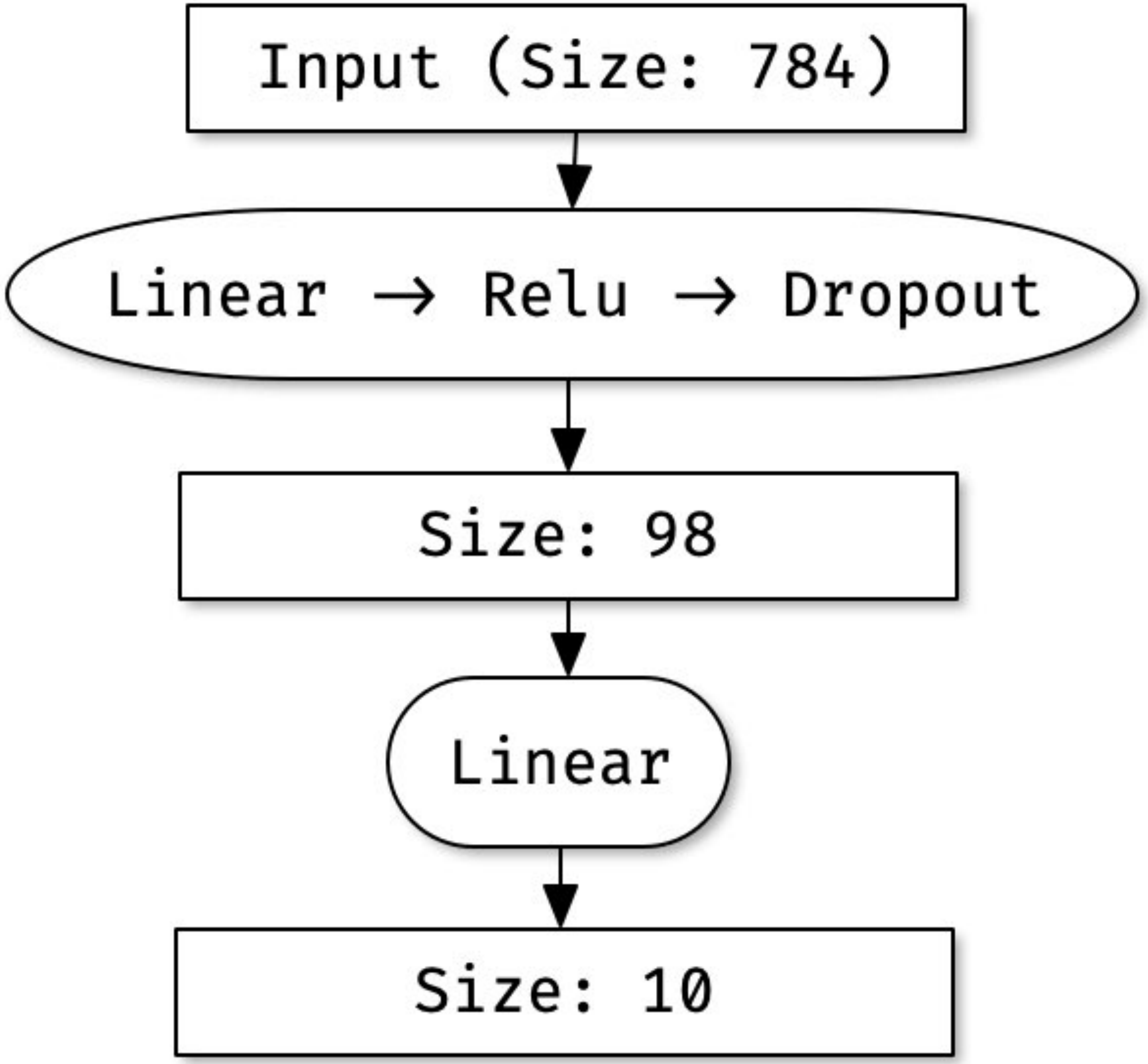

```
print(X.shape, y.shape)
# (70000, 784) (70000,)
```

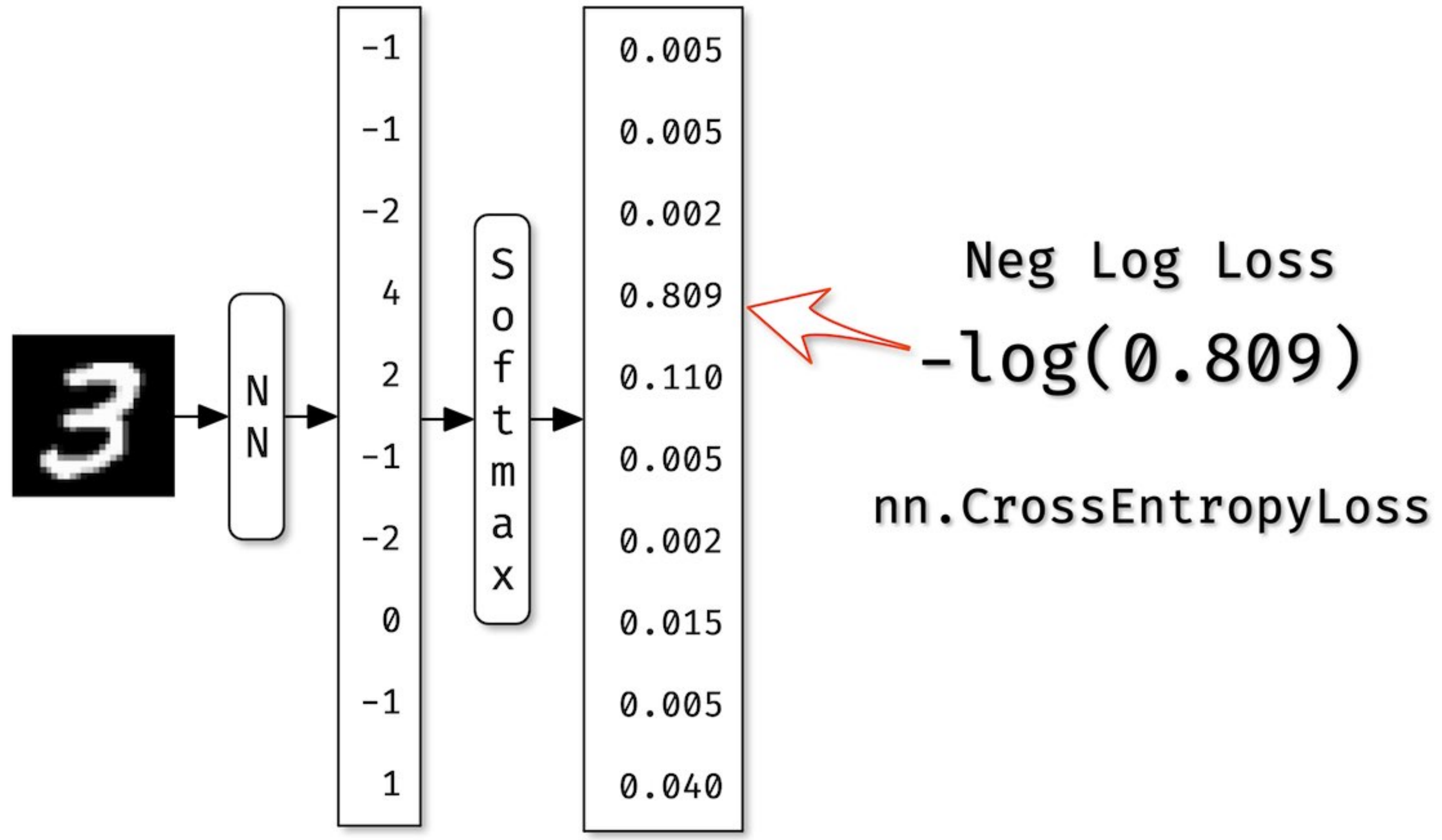
MNIST - Data Code

```
from sklearn.model_selection import train_test_split

X_scaled = X / X.max()

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.25, random_state=42)
```

MNIST - Neural Network Module

```
from torch.nn as nn

class SimpleFeedforward(nn.Module):
    def __init__(self, dropout=0.5):
        super().__init__()
        self.module = nn.Sequential(
            nn.Linear(784, 98),
            nn.ReLU(inplace=True),
            nn.Dropout(dropout),
            nn.Linear(98, 10))

    def forward(self, X):
        return self.module(X)
```




N N
-1
-1
-2
4
2
-1
-2
0
-1
1
Softmax
0.005
0.005
0.002
0.809
0.110
0.005
0.002
0.015
0.005
0.040
Neg Log Loss
-log(0.809)
nn.CrossEntropyLoss

MNIST - Loss function skorch

```python
from skorch import NeuralNet

net = NeuralNet(
    SimpleFeedforward,
    criterion=nn.CrossEntropyLoss,
    max_epochs=10,
    lr=0.3,
    device='cuda', # comment to train on cpu
)
```

MNIST - Fitting

```
_ = net.fit(X_train, y_train)
```

epoch	train_loss	valid_loss	dur
1	0.5772	0.3568	0.4763
2	0.3260	0.2167	0.4688
3	0.2723	0.1936	0.4730
4	0.2429	0.2328	0.4733
5	0.2244	0.1475	0.4709
6	0.2065	0.1422	0.4756
7	0.1974	0.1407	0.4841
8	0.1881	0.1378	0.4747
9	0.1814	0.1409	0.4759
10	0.1740	0.1212	0.4786

MNIST – Continue Training

```
net.set_params(max_epochs=5)
_ = net.partial_fit(X_train, y_train)
```

11	0.1668	0.1161	0.4888
12	0.1635	0.1245	0.4815
13	0.1592	0.1099	0.4876
14	0.1569	0.1185	0.4840
15	0.1500	0.1100	0.4805

MNIST - History

```
len(net.history)
# 15

net.history[-1, 'valid_loss']
# 0.10163110941932314

net.history[-2:, 'train_loss']
# [0.13314295971961249,
#  0.1330454680351984]
```

MNIST - Accuracy Score

```
y_pred = net.predict(X_test)
print(y_pred[0, :])
# [ -3.7149053   -3.4425774    4.2277        6.454636   -11.068788
#    2.8243313   -4.5492096   -2.9567776   11.11874     -0.02089611]

def accuracy_argmax(y_true, y_pred):
    return np.mean(y_true == np.argmax(y_pred, -1))

print('test accuracy:', accuracy_argmax(y_test, y_pred))
# test accuracy: 0.9634857142857143
```

MNIST - EpochScoring

```python
from sklearn.metrics import make_scorer
accuracy_argmax_scorer = make_scorer(accuracy_argmax)

from skorch.callbacks import EpochScoring

epoch_acc = EpochScoring(
    accuracy_argmax_scorer,
    name='valid_acc',
    lower_is_better=False)

net = NeuralNet( ... ,
    callbacks=[epoch_acc])
```

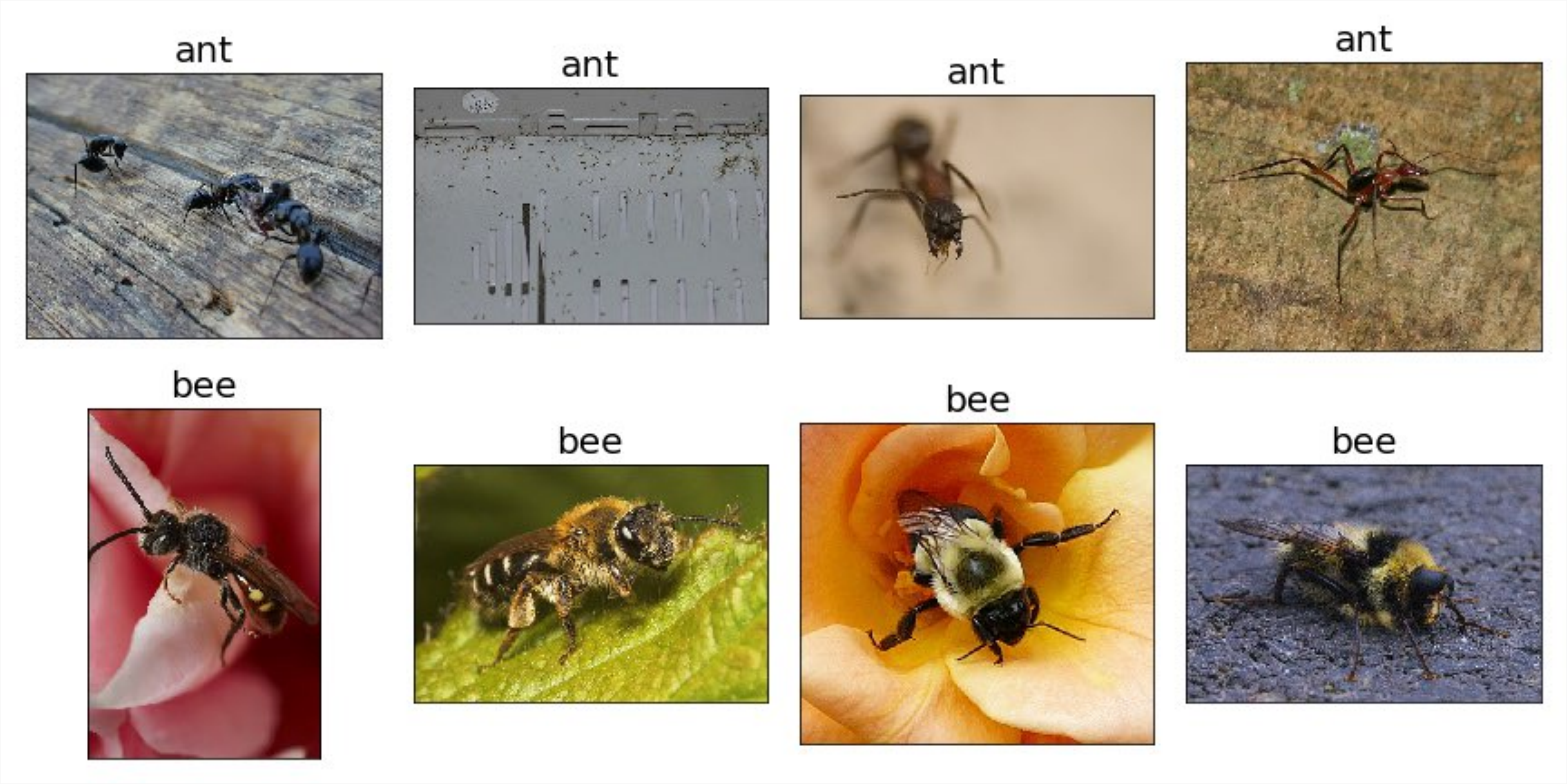
MNIST - Fitting With EpochScoring

```
_ = net.fit(X, y)
```

epoch	train_loss	valid_acc	valid_loss	dur
1	0.5751	0.8995	0.3288	0.5127
2	0.3145	0.9332	0.2230	0.4830
3	0.2653	0.9447	0.1778	0.4798
4	0.2357	0.9500	0.1609	0.4767
5	0.2147	0.9497	0.1620	0.4835
6	0.2026	0.9510	0.1518	0.4834
7	0.1906	0.9608	0.1303	0.4868
8	0.1824	0.9605	0.1287	0.4635
9	0.1754	0.9592	0.1339	0.4845
10	0.1730	0.9614	0.1247	0.4662

MNIST - Scikit-Learn Integration

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler

pipe = Pipeline([
    ("min_max", MinMaxScaler()),
    ("net", net)])

_ = pipe.fit(X_train, y_train)
```


ant
ant
ant
ant
bee
bee
bee
bee

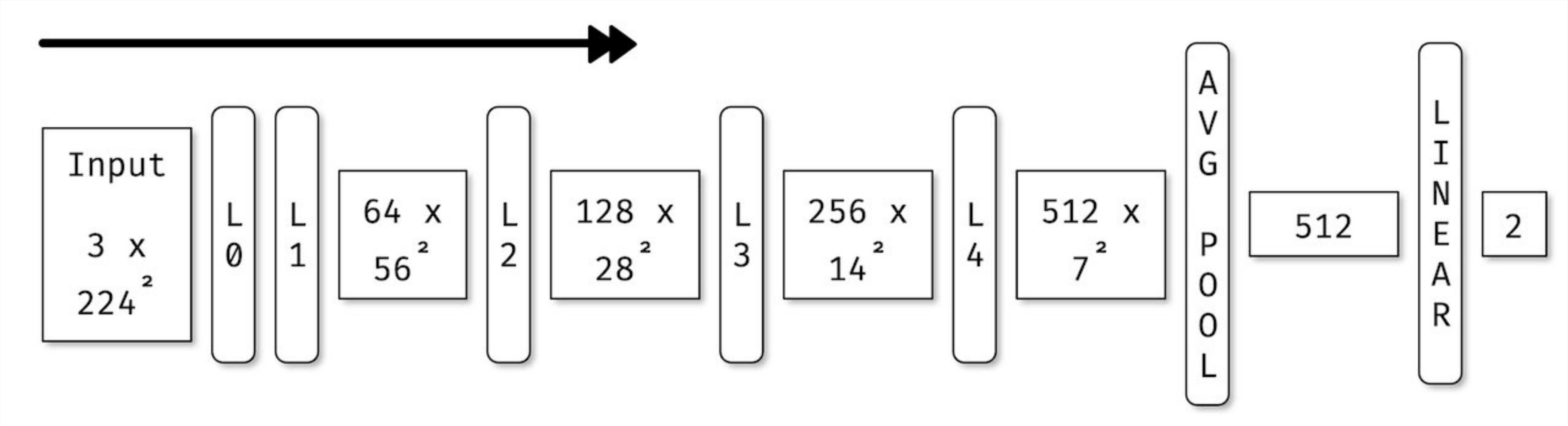
Ants and Bees - ImageFolder Transformations

```python
import torchvision.transforms as tfms

train_tfms = tfms.Compose([
    tfms.RandomResizedCrop(224),
    tfms.RandomHorizontalFlip(),
    tfms.ToTensor(),
    tfms.Normalize([0.485, 0.456, 0.406],
                   [0.229, 0.224, 0.225])])

train_ds = ImageFolder(
    "datasets/hymenoptera_data/train" , train_tfms)
valid_ds = ImageFolder(
    "datasets/hymenoptera_data/val" , val_tfms)
```

Ants and Bees - ImageNet

- 1000 classes
- 1300 images for each class
- Mean of ImageNet: `[0.485, 0.456, 0.406]`
- Standard Deviation of ImageNet: `[0.229, 0.224, 0.225]`

Ants and Bees – ResNet Model

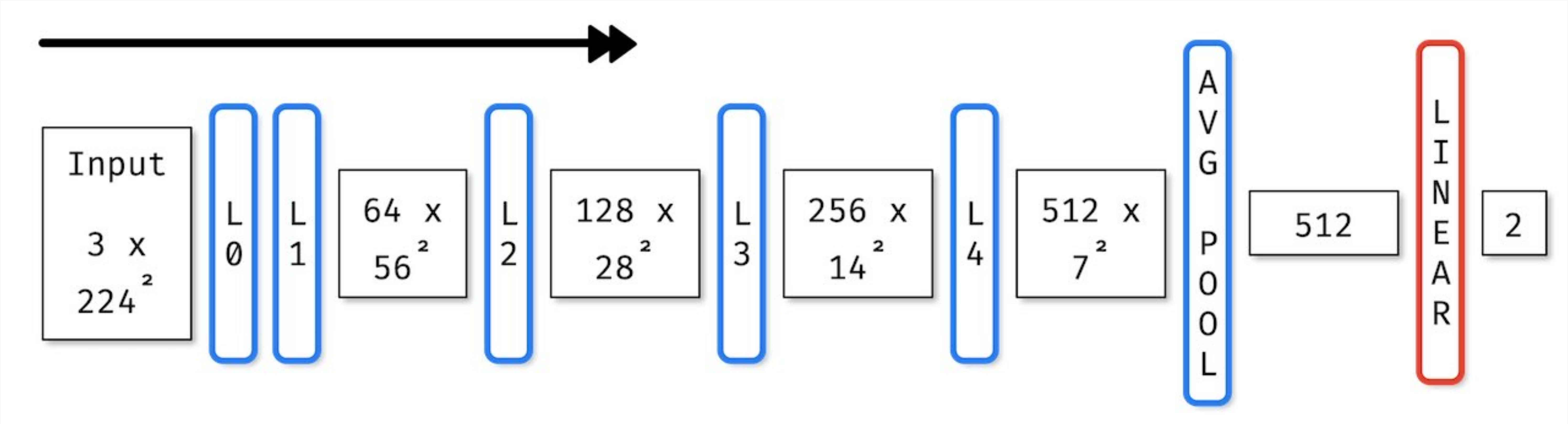


K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of CVPR, pages 770–778, 2016. arxiv.org/abs/1512.03385

Ants and Bees - ResNet Model Code

```
from torchvision.models import resnet18
import torch.nn as nn

class PretrainedModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = resnet18(pretrained=True)
        self.model.fc = nn.Linear(512, 2)

    def forward(self, X):
        return self.model(X)
```

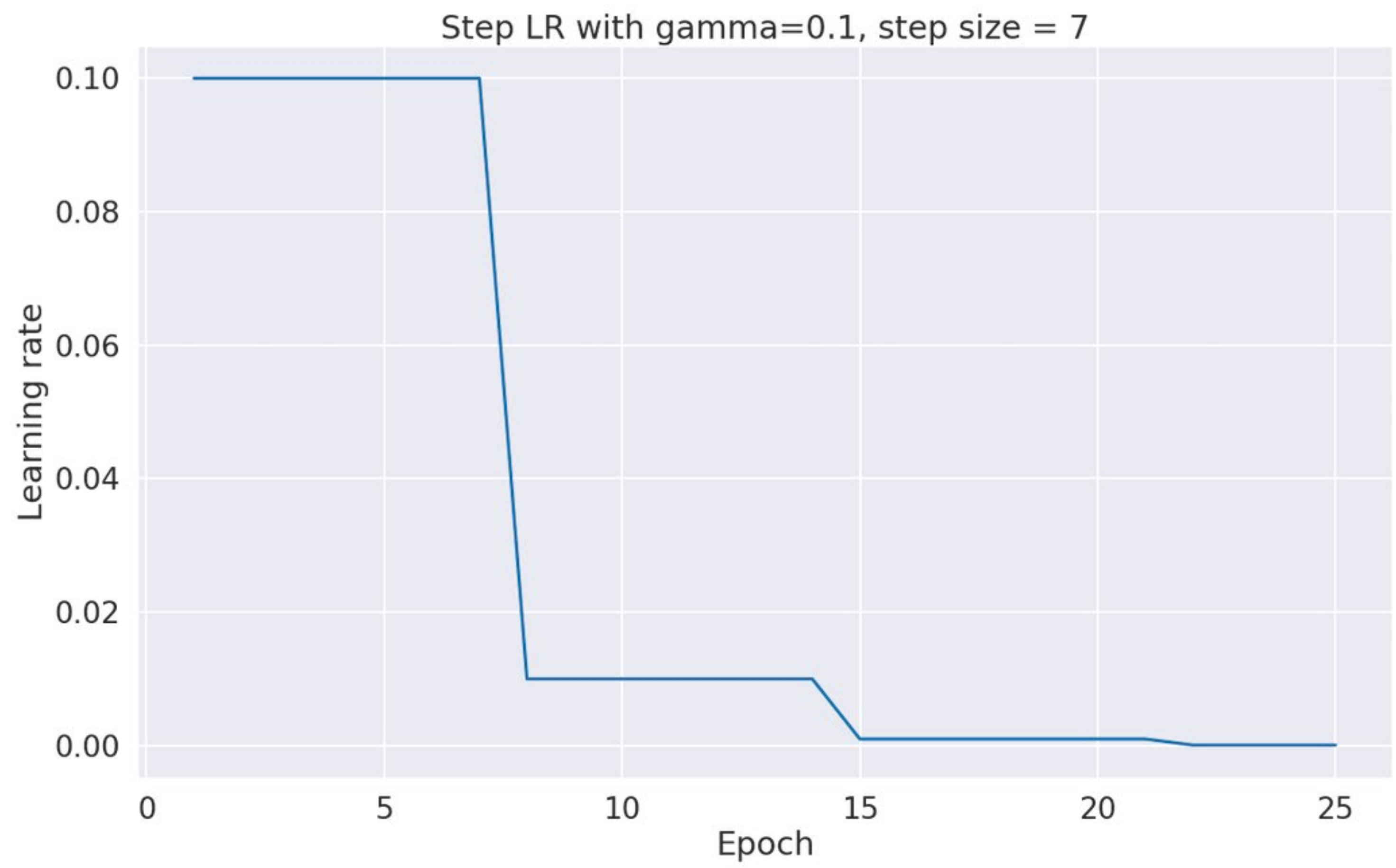
Ants and Bees - Freezer




```python
from skorch.callbacks import Freezer

freezer = Freezer(
    lambda name: not name.startswith("model.fc"))
```

Ants and Bees - Learning Rate Scheduler

```python
from skorch.callbacks import (
    LRScheduler
)

lr_scheduler = LRScheduler(
    policy="StepLR",
    step_size=7,
    gamma=0.1)
```



Ants and Bees - Checkpoints

```python
from skorch.callbacks import Checkpoint

epoch_acc = EpochScoring( ... , name='valid_acc',
    lower_is_better=False)

checkpoint = Checkpoint(
    dirname="exp_01_bee_vs_ant", monitor="valid_acc_best")
```

Ants and Bees - Skorch NeuralNet

```
from skorch.helper import predefined_split

net = NeuralNet(
    PretrainedModel,
    lr=0.001, batch_size=4,
    train_split=predefined_split(val_ds),
    callbacks=[freezer, lr_scheduler,
        epoch_acc, checkpoint],
     ...
)
```

Ants and Bees - Fitting

```
_ = net.fit(train_ds)
```

epoch	train_loss	valid_acc	valid_loss	cp	dur
1	0.5656	0.8824	0.2681	+	0.9402
2	0.6011	0.9281	0.2290	+	0.9465
3	0.4898	0.9281	0.2085		0.9154
4	0.5193	0.8824	0.2966		0.9741
5	0.6659	0.8627	0.3104		0.9467
6	0.3655	0.9216	0.2233		0.9127
7	0.3398	0.8954	0.2936		0.9244
8	0.3809	0.9346	0.1581	+	0.9516
9	0.3890	0.9281	0.2194		0.9435
10	0.4015	0.9085	0.2124		0.9573

Ants and Bees – Checkpoint Files

```
exp_01_bee_vs_ant
├── history.json
├── optimizer.pt
└── params.pt
```

Ants and Bees - Loading from Checkpoint

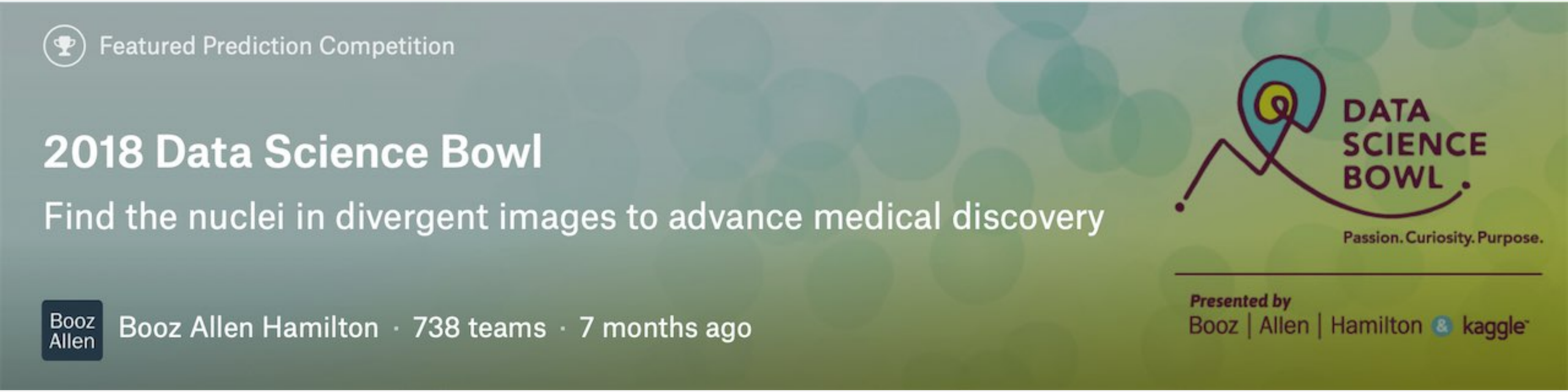
```
# net.fit( ... ) was called
net.load_params(checkpoint=checkpoint)

val_output = net.predict(val_ds)
```

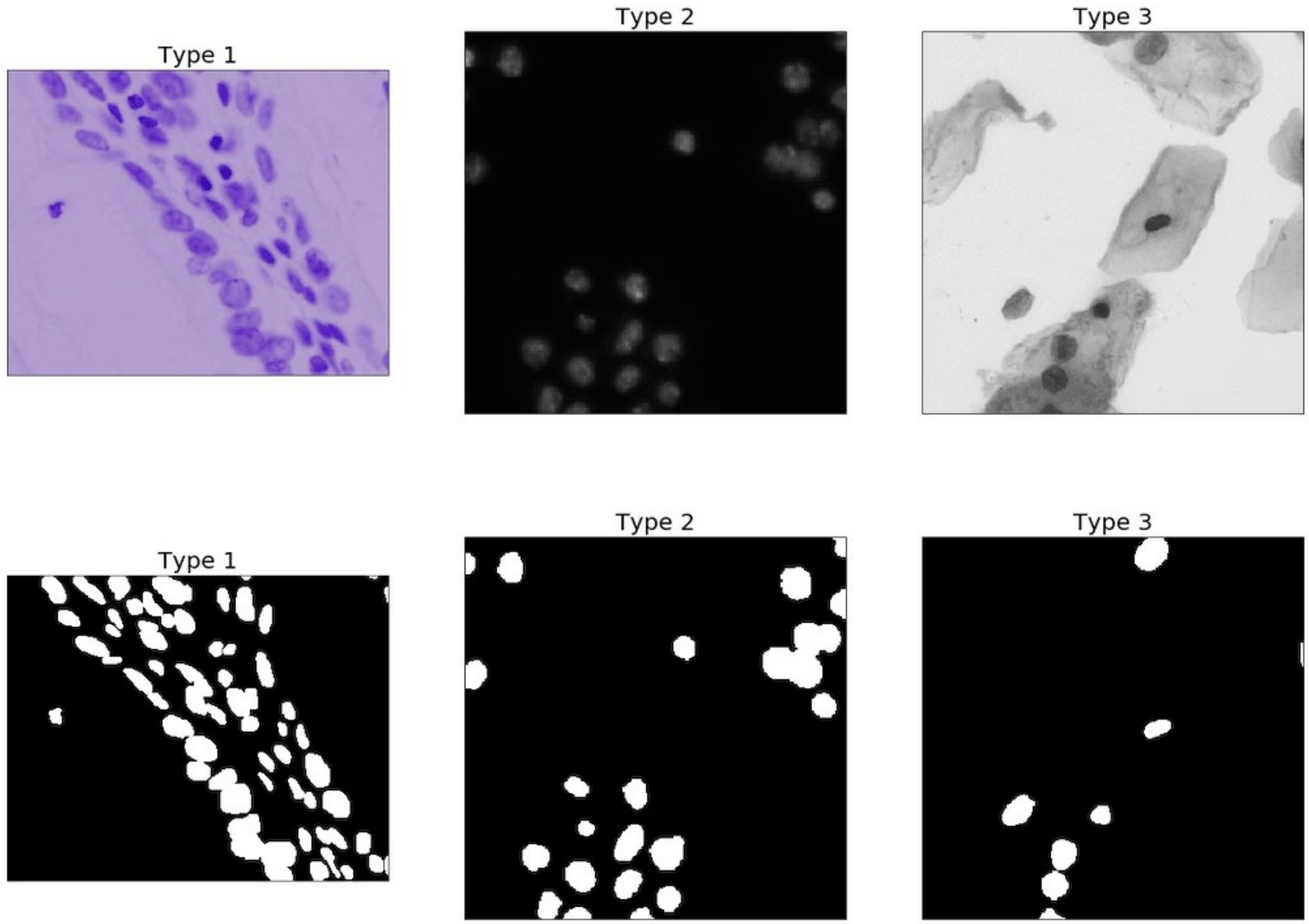
Ants and Bees - Prediction

```
checkpoint = Checkpoint( ... ,
    dirname="exp_01_bee_vs_ant",
    monitor="valid_acc_best")

net = NeuralNet(PretrainedModel,  ... )
net.initialize()
net.load_params(checkpoint=checkpoint)

val_pred = net.predict(val_ds)
```

Ants and Bees - Prediction Numpy

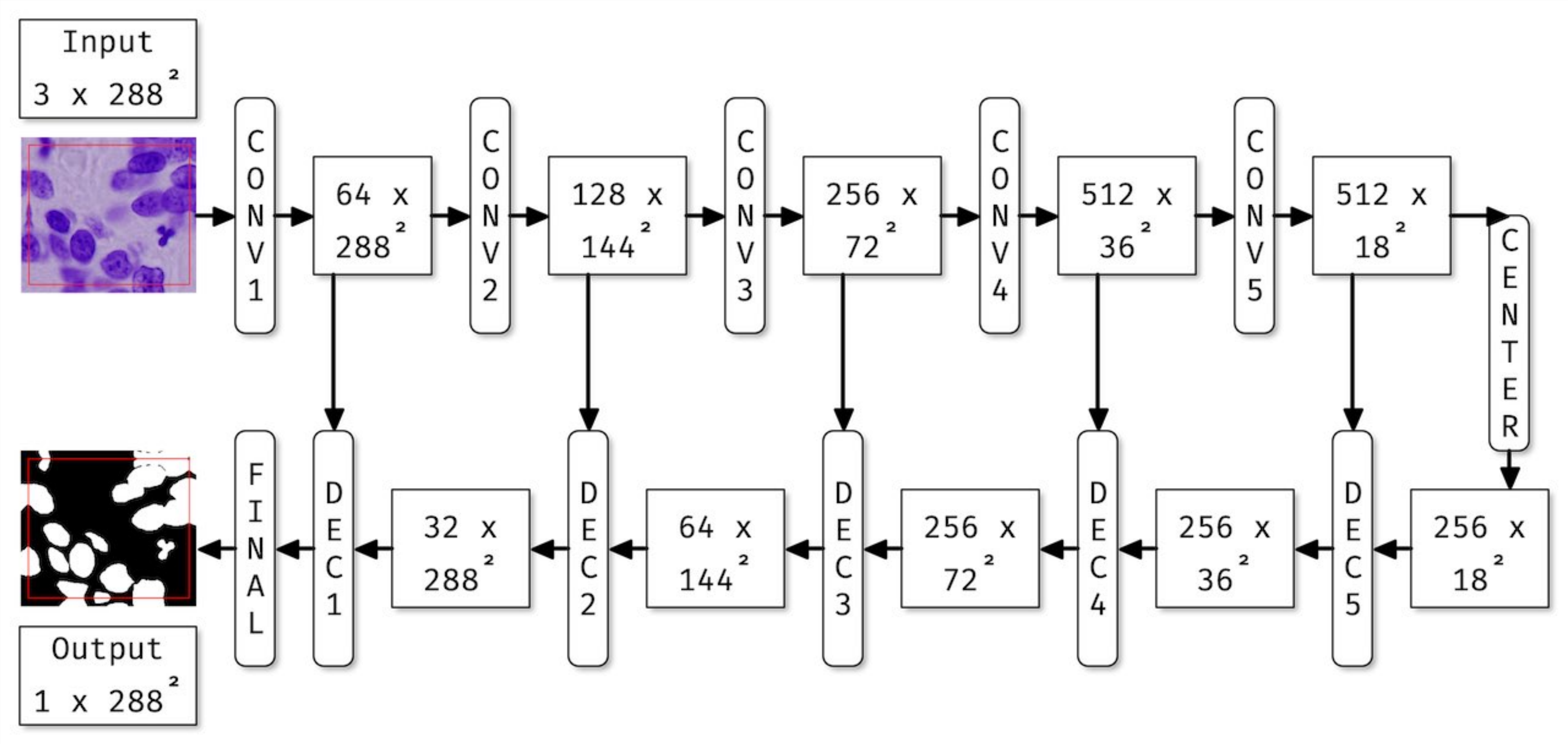
```
print(X_numpy.shape)
# (1, 3, 224, 224)

X_pred = net.predict(X_numpy)
print(X_pred)
# [[ 0.4966519, -0.9894746]]

print(softmax(X_pred))
# [[0.8154962  0.18450384]]
```


Featured Prediction Competition
2018 Data Science Bowl
Find the nuclei in divergent images to advance medical discovery
Booz Allen
Booz Allen Hamilton · 738 teams · 7 months ago
DATA SCIENCE BOWL
Passion. Curiosity. Purpose.
Presented by
Booz | Allen | Hamilton & kaggle

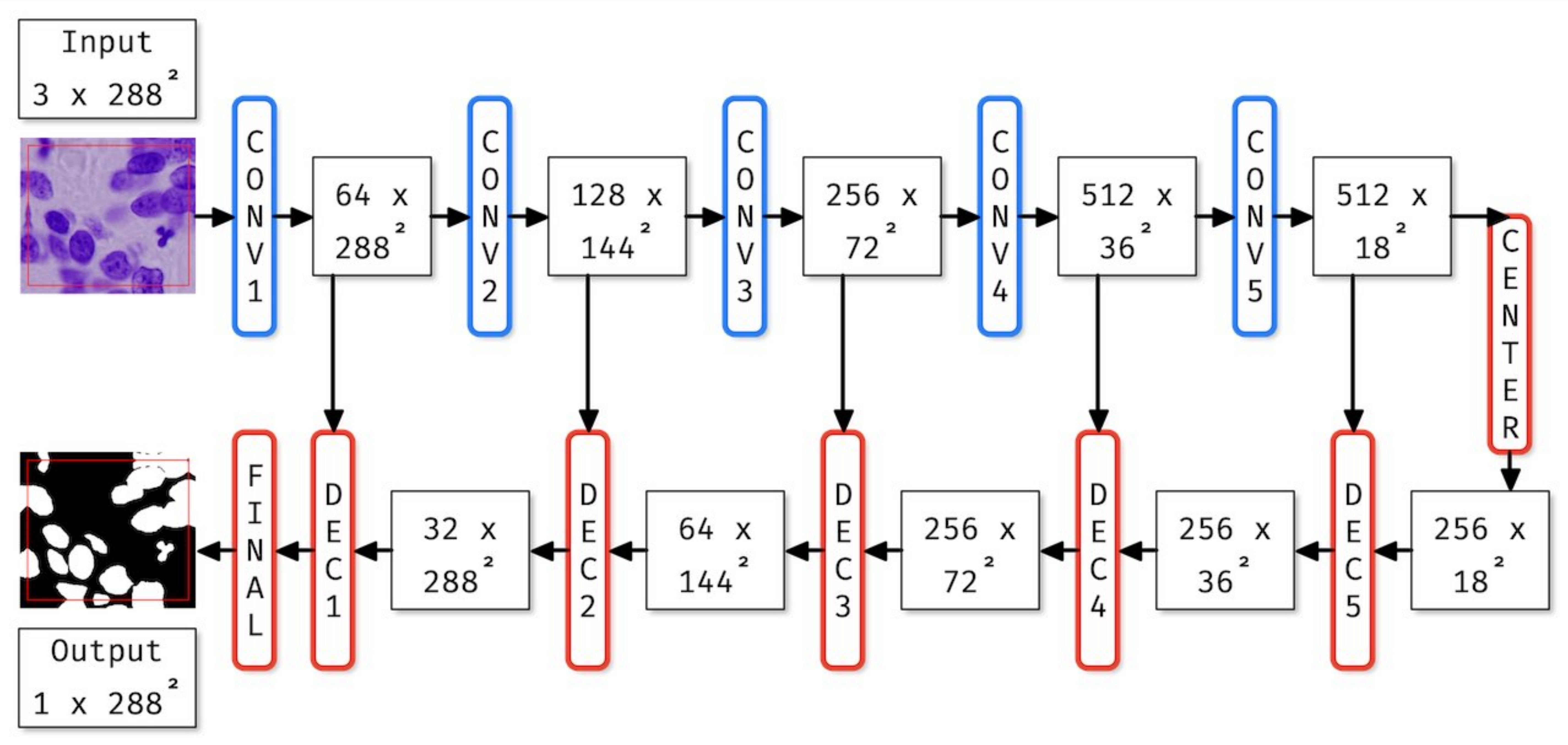

Type 1
Type 2
Type 3
Type 1
Type 2
Type 3

Nuclei Image Segmentation – Dataset

```
train_cell_ds = CellsDataset( ... )
valid_cell_ds = CellsDataset( ... )

print(train_cell_ds[0])
# (<PIL.Image.Image>
#  <PIL.PngImagePlugin.PngImageFile>)
```

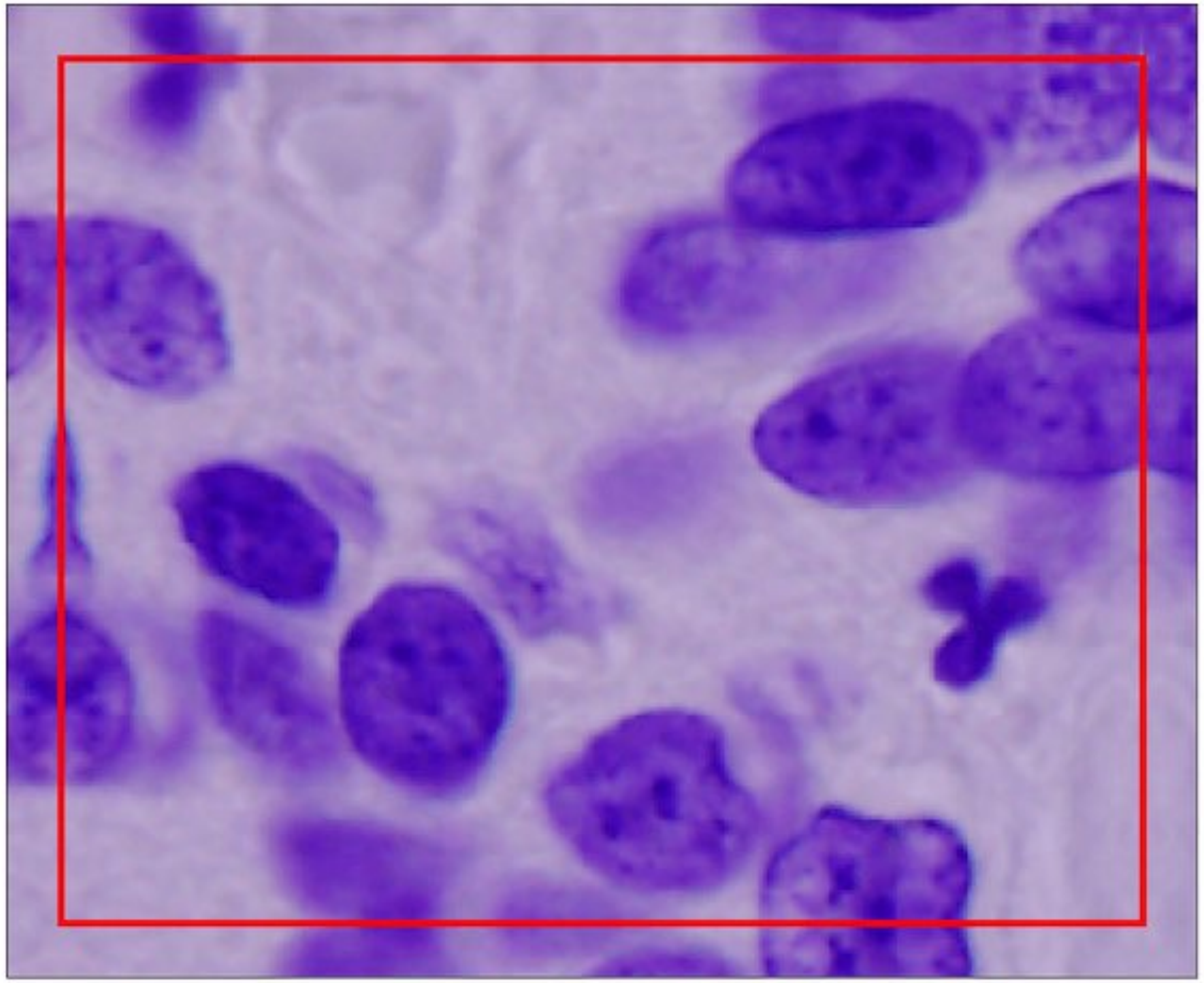


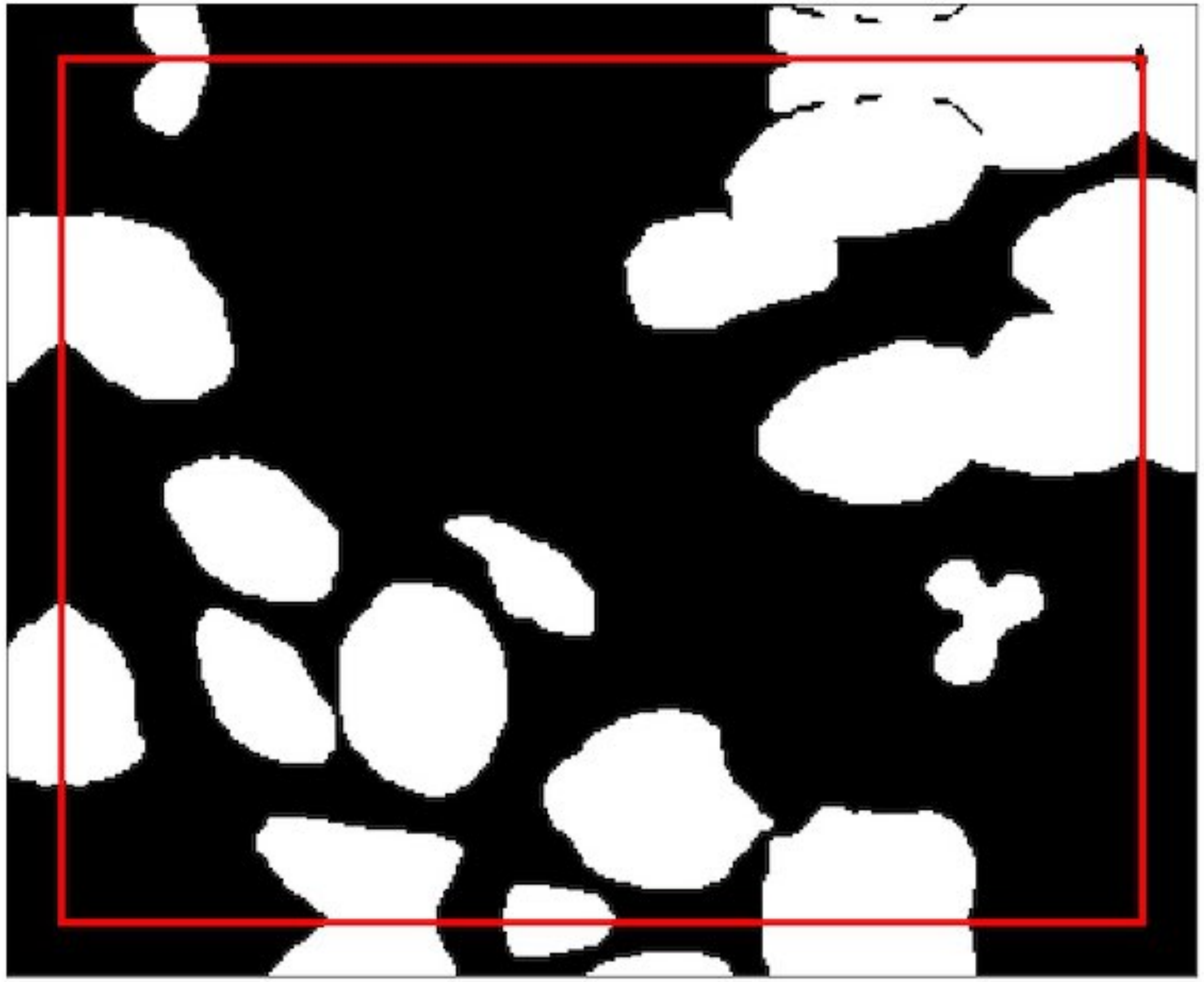
O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in MICCAI, pp. 234–241, Springer, 2015. arxiv.org/abs/1505.04597



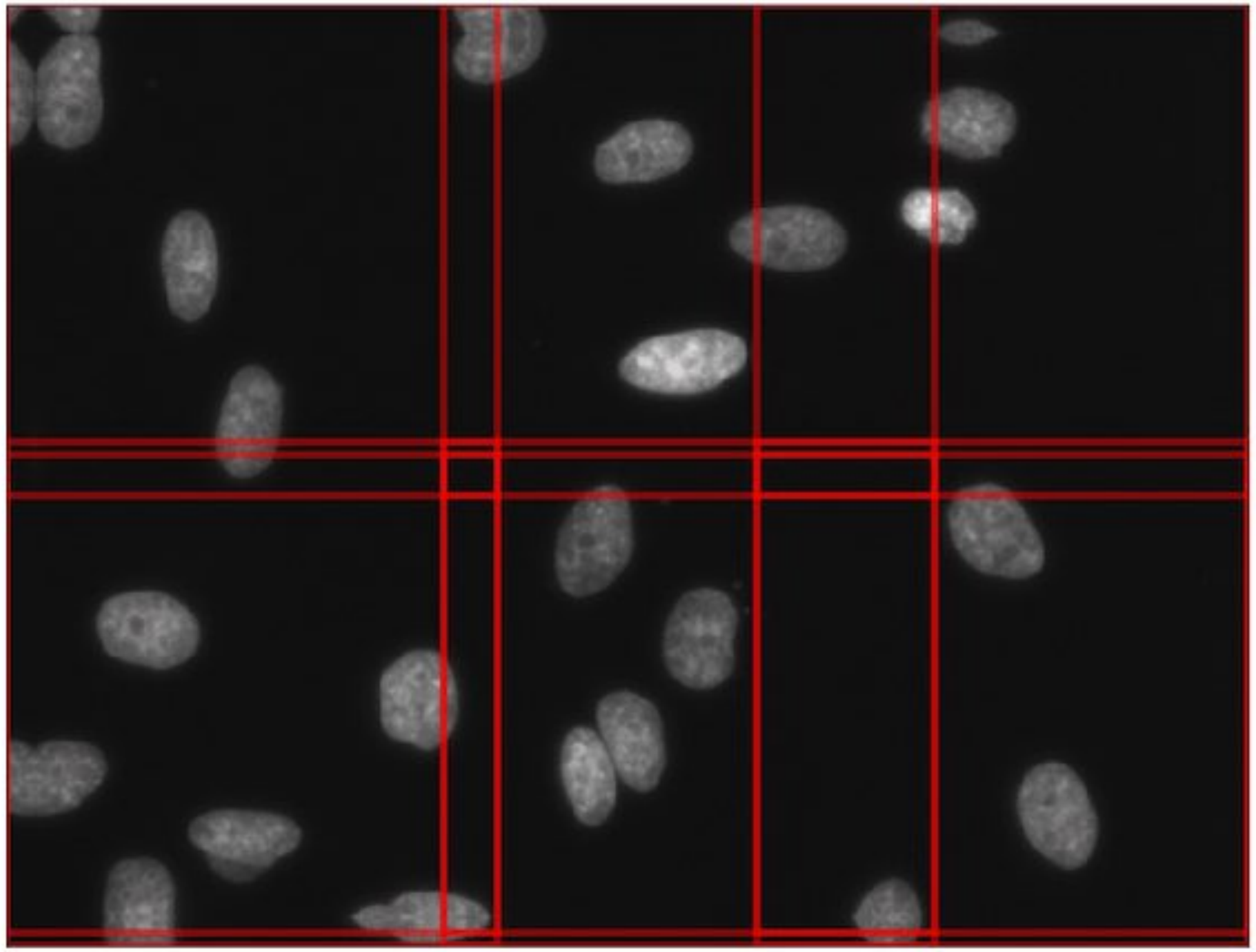

```python
from skorch.callbacks import Freezer
freezer = Freezer('conv*')
```

Nuclei Image Segmentation - PatchedDataset 1





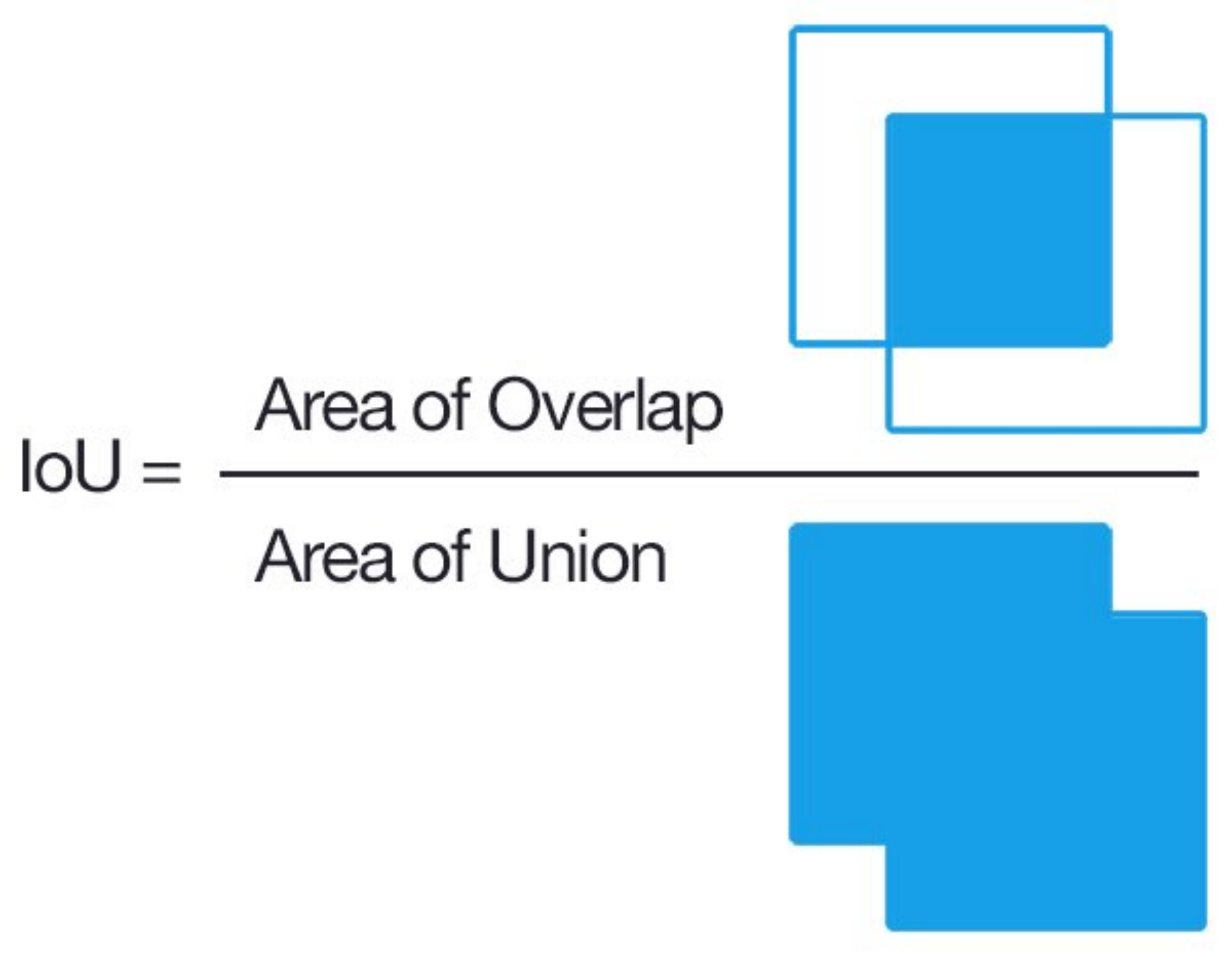
Nuclei Image Segmentation - PatchedDataset 2



Nuclei Image Segmentation – PatchedDataset Code

```
train_ds = PatchedDataset(
    train_cell_ds, patch_size=(256, 256),
    padding=16, random_flips=True)

val_ds = PatchedDataset(
    valid_cell_ds, patch_size=(256, 256),
    padding=16, random_flips=False)
```

Nuclei Image Segmentation - IOU



Nuclei Image Segmentation - IOU Metric

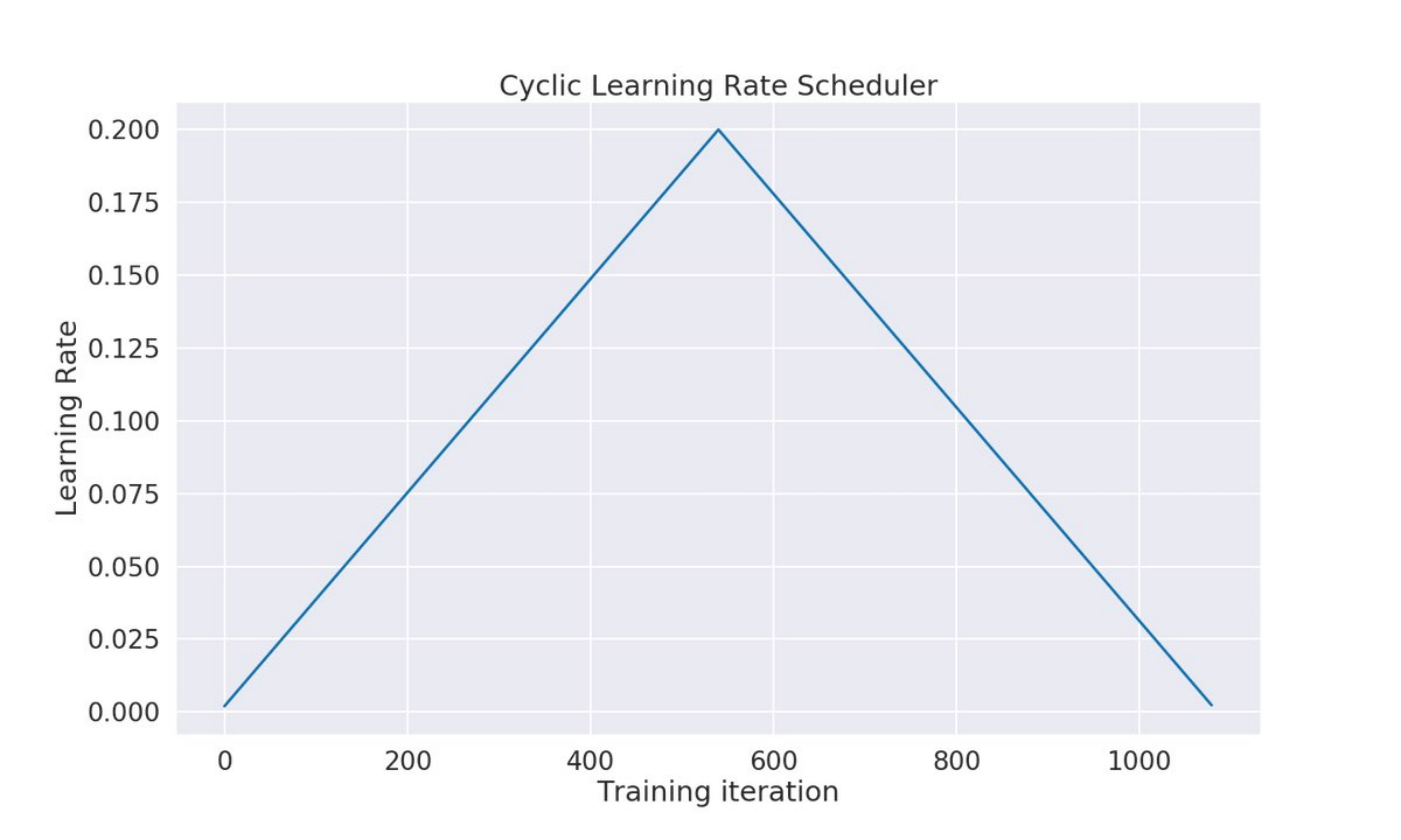
```
def approximate_iou_metric(
        true_masks, predicted_logit_masks, padding=16):
    ... # returns metric
iou_scoring = make_scorer(approximate_iou_metric)

iou_scoring = EpochScoring(
    iou_scoring, name='valid_iou', lower_is_better=False)

best_cp = Checkpoint(
    dirname="kaggle_seg_exp01", monitor="valid_iou_best")
```

Nuclei Image Segmentation – Custom Loss

```
class BCEWithLogitsLossPadding(nn.Module):
    def __init__(self, padding):
        super().__init__()
        self.padding = padding
    ...

net = NeuralNet(
    UNet,
    criterion=BCEWithLogitsLossPadding,
    criterion__padding=16,
    ...
)
```

Nuclei Image Segmentation - Cyclic LR Scheduler

```
cyclicLR = LRScheduler(
    policy="CyclicLR",
    base_lr=0.002,
    max_lr=0.2,
    step_size_up=550,
    step_size_down=550)
```



Nuclei Image Segmentation - NeuralNet

```
net = NeuralNet(
    UNet,
    criterion=BCEWithLogitsLossPadding,
    criterion__padding=16,
    batch_size=32,
    max_epochs=20,
    train_split=predefined_split(val_ds),
    callbacks=[freezer, cyclicLR, iou_scoring, best_cp],
    ...
)
```

Nuclei Image Segmentation - NeuralNet DataLoader

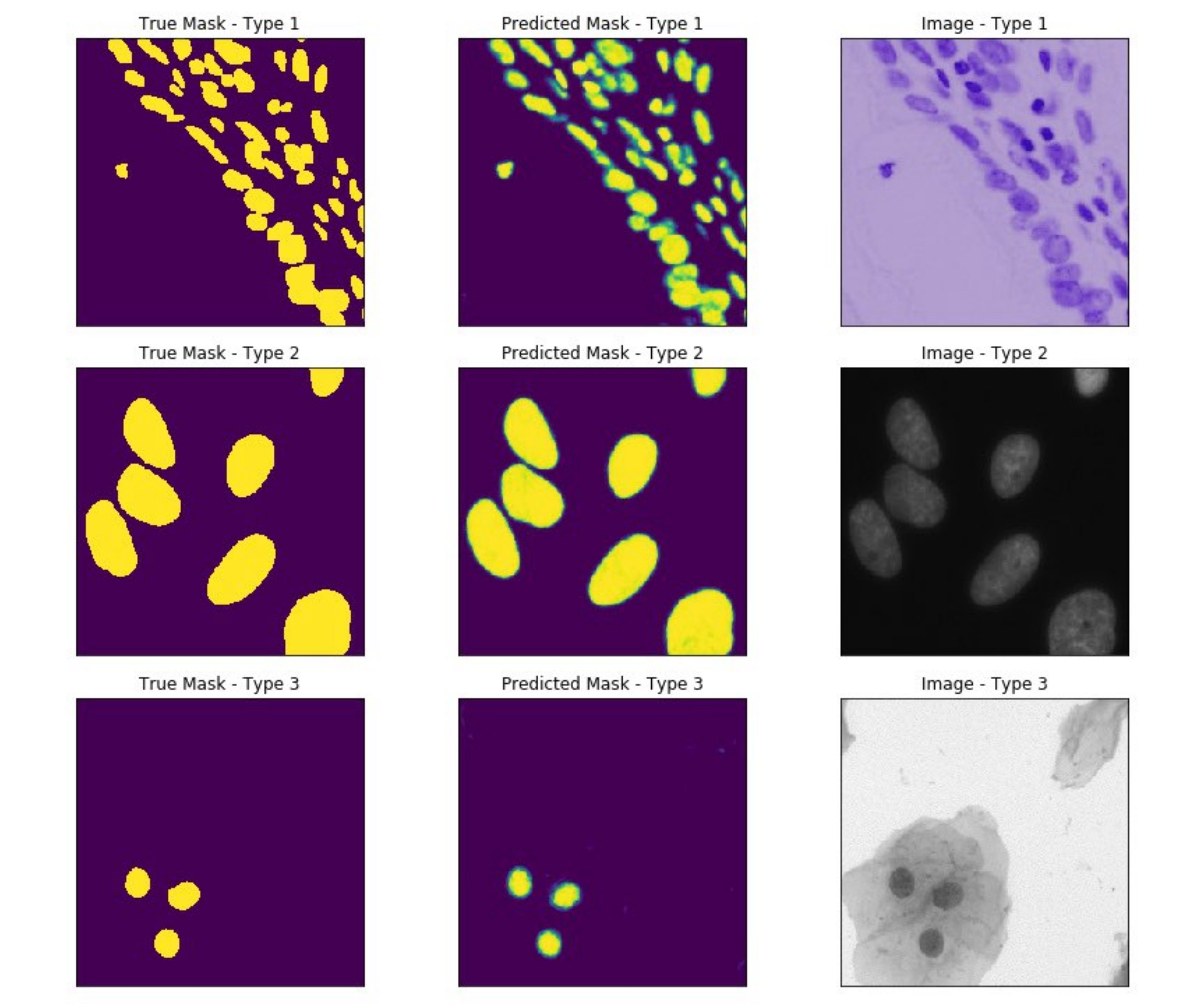
PyTorch's `DataLoader(pin_memory=False, num_workers=0, ... )`

```
net = NeuralNet( ... ,
    iterator_train__shuffle=True,
    iterator_train__num_workers=4,
    iterator_train__pin_memory=True,
    iterator_valid__shuffle=False,
    iterator_valid__num_workers=4,
    iterator_valid__pin_memory=True)

_ = net.fit(train_ds)
```

epoch	train_loss	valid_iou	valid_loss	cp	dur
1	0.4996	0.0797	0.4151	+	48.6801
2	0.3818	0.1375	0.3349	+	46.5345
3	0.2848	0.3154	0.2302	+	46.6045
4	0.1811	0.3189	0.3628	+	46.7507
5	0.1377	0.6170	0.0951	+	46.8530
6	0.0806	0.6827	0.0692	+	46.6725
7	0.0693	0.7083	0.0596	+	46.6736
8	0.0686	0.7303	0.0669	+	46.5634
9	0.0656	0.7552	0.0551	+	46.7421
10	0.0576	0.7480	0.0524		46.7592
11	0.0641	0.7555	0.0568	+	46.7328
12	0.0550	0.7484	0.0520		46.9584
13	0.0532	0.7606	0.0512	+	46.5919
14	0.0529	0.7722	0.0502	+	46.6308
15	0.0520	0.7711	0.0503		46.8443
16	0.0516	0.7714	0.0497		46.8938
17	0.0518	0.7733	0.0503	+	46.7276
18	0.0515	0.7704	0.0500		46.7965
19	0.0515	0.7723	0.0504		46.8066
20	0.0522	0.7560	0.0581		46.7256

Nuclei Image Segmentation - Predict on Validation

```
net.load_params(checkpoint=best_cp)

val_masks = net.predict(val_ds)
print(val_masks.shape)
# (468, 1, 288, 288)

val_prob_masks = sigmod(val_masks.squeeze(1))
print(val_prob_masks.shape)
# (468, 288, 288)
```


True Mask - Type 1
Predicted Mask - Type 1
Image - Type 1
True Mask - Type 2
Predicted Mask - Type 2
Image - Type 2
True Mask - Type 3
Predicted Mask - Type 3
Image - Type 3

Skorch - Closing 1

1. Scikit-Learn compatible neural network library that wraps PyTorch.
   - `net.fit(X, y)`
   - `net.partial_fit(X, y)`
   - `net.predict(X)`
   - `net.set_params( ... )`
2. Abstracts away the training loop.

Skorch - Closing 2

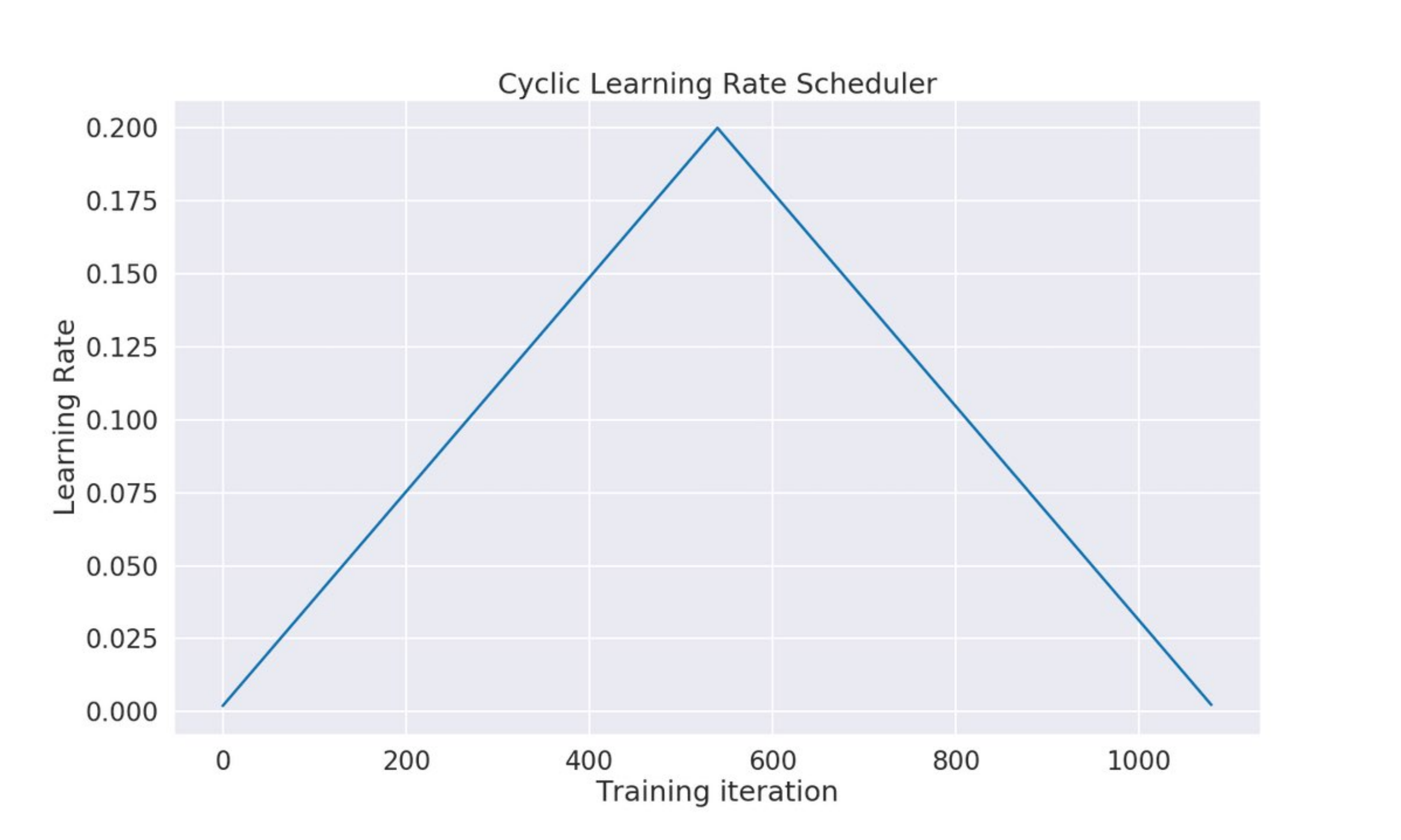
1. Reduces the amount of boilerplate code with callbacks.
   - `EpochScoring`
   - `Freezer`
   - `Checkpoint`
   - `LRScheduler`
   - skorch.readthedocs.io/en/stable/user/callbacks.html

Skorch - Whats next

- github.com/thomasjpfan/skorch_talk
- github.com/skorch-dev/skorch
- skorch.readthedocs.io
- Thomas J Fan - @thomasjpfan

Appendix Nuclei Image Segmentation – Cyclic LR Scheduler

- Number of training samples: `len(train_ds) = 1756`
- `max_epochs = 20`
- `batch_size = 32`
- Training iterations per epoch: `ceil(1756/32) = 55`
- Total number of iterations: `55*20 = 1100`



Appendix Ants and Bees - Saving and Loading

```python
from skorch.callbacks import TrainEndCheckpoint
from skorch.callbacks import LoadInitState

def run(max_epochs):
    best_cp = Checkpoint(dirname="exp_02", ... )

    train_end_cp = TrainEndCheckpoint(
        dirname="exp_02", fn_prefix="train_end_")
    load_state = LoadInitState(train_end_cp)

    net = NeuralNet( ... ,
        max_epochs=max_epochs,
        callbacks=[ ... , best_cp, train_end_cp, load_state]
    ).fit(train_ds)
```

Appendix Ants and Bees – Saving and Loading Checkpoints

```
exp_02
├── history.json
├── optimizer.pt
├── params.pt
├── train_end_history.json
├── train_end_optimizer.pt
└── train_end_params.pt
```

Appendix Ants and Bees – Saving and Loading First Run

```
run(max_epochs=10)
```

epoch	train_loss	valid_acc	valid_loss	cp	dur
1	0.5656	0.8824	0.2681	+	0.9402
2	0.6011	0.9281	0.2290	+	0.9465
3	0.4898	0.9281	0.2085		0.9154
4	0.5193	0.8824	0.2966		0.9741
5	0.6659	0.8627	0.3104		0.9467
6	0.3655	0.9216	0.2233		0.9127
7	0.3398	0.8954	0.2936		0.9244
8	0.3809	0.9346	0.1581	+	0.9516
9	0.3890	0.9281	0.2194		0.9435
10	0.4015	0.9085	0.2124		0.9573

Appendix Ants and Bees – Saving and Loading Second Run

```
run(max_epochs=5)
```

epoch	train_loss	valid_acc	valid_loss	cp	dur
11	0.3087	0.9216	0.2368	+	1.4001
12	0.3379	0.9020	0.2295		1.3781
13	0.2764	0.8824	0.2520		1.3766
14	0.3557	0.8889	0.2466		1.3697
15	0.2920	0.9150	0.2106		1.4086

Ants and Bees – Folder Structure

```
datasets/hymenoptera_data/
├── train
│   ├── ants
│   └── bees
└── val
    ├── ants
    └── bees
```

Ants and Bees - ImageFolder Init

```python
from torchvision.datasets import ImageFolder

train_tfms = ...
val_tfms = ...

train_ds = ImageFolder(
    "datasets/hymenoptera_data/train" , train_tfms)
val_ds = ImageFolder(
    "datasets/hymenoptera_data/val", val_tfms)
```

Ants and Bees - ImageFolder Class

Subclass of `torch.utils.data.Dataset`

```
print(len(train_ds), len(val_ds))
# (244, 153)

img, target = train_ds[0]
print(img.shape, target)
# (torch.Size([3, 224, 224]), 0)

# For ImageFolder only:
print(train_ds.class_to_idx)
# {'ants': 0, 'bees': 1}
```

MNIST - Grid Search

```python
from sklearn.model_selection import GridSearchCV

param_grid = {"net__module__dropout": [0.2, 0.5, 0.8]}

gs = GridSearchCV(pipe, param_grid, cv=3,
                  scoring=accuracy_argmax_scorer)
_ = gs.fit(X, y)
print("best score:", gs.best_score_)
# best score: 0.9651

print("best_params", gs.best_params_)
# best_params {'net__module__dropout': 0.2}
```

MNIST - More sckit-learn like - 1

```python
from sklearn.base import ClassifierMixin
from sklearn.utils.extmath import softmax
from skorch import NeuralNet

class MyNNClassifer(ClassifierMixin, NeuralNet):
    def predict_proba(self, X):
        y_pred = super().predict_proba(X)
        return softmax(y_pred)

    def predict(self, X):
        y_pred = super().predict(X)
        return np.argmax(y_pred, axis=1)
```

MNIST - More sckit-learn like - 2

```
epoch_acc = EpochScoring(
    'accuracy',
    name='valid_acc',
    lower_is_better=False)

net = MyNNClassifer(
    SimpleFeedforward,
    ...,
    callbacks=[epoch_acc])
```