KOREATECH
한국기술교육대학교

FCN Best Practice

Oct. 2023

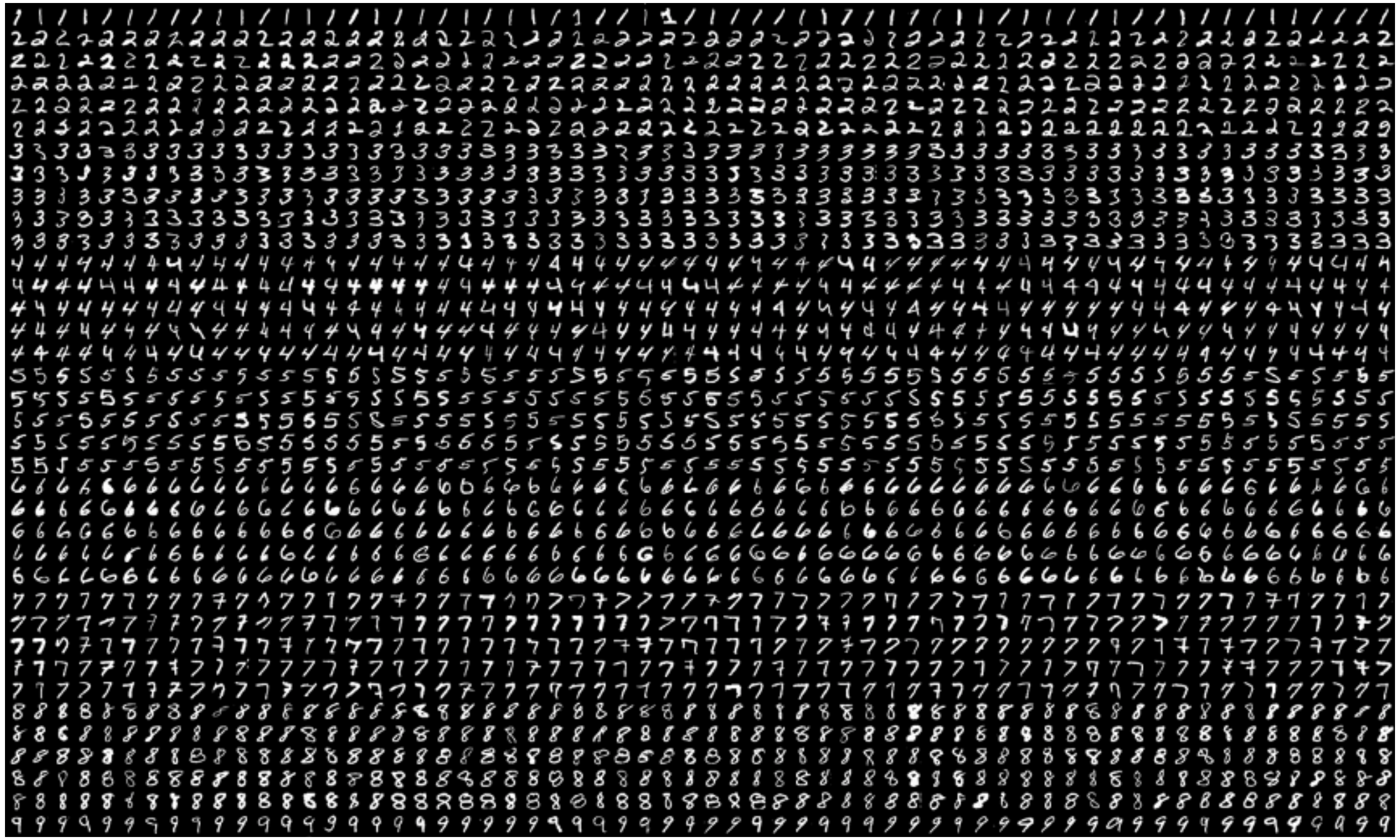
http://link.koreatech.ac.kr

MNIST dataset

The MNIST Dataset

◈The MNIST Dataset Description

- It consists of 70,000 28x28 handwritten digit images in 10 classes, with 7,000 images per class (ranging from 0 to 9)
- There are 60,000 training images and 10,000 validation (or test) images



The MNIST Dataset

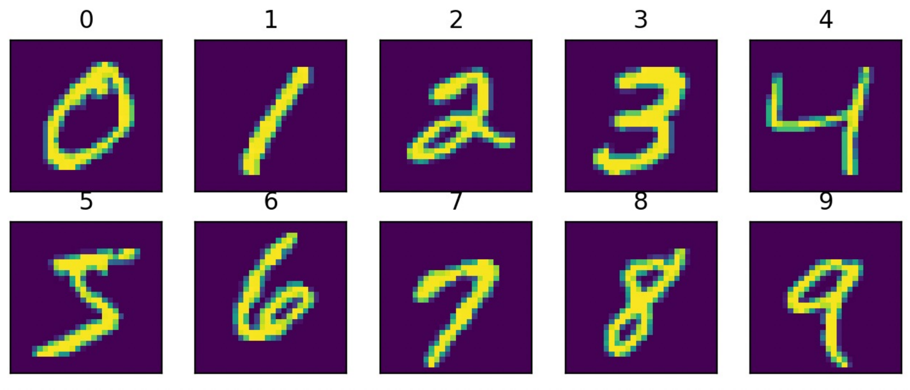
◈The MNIST Dataset in PyTorch

```
import numpy as np
from matplotlib import pyplot as plt
import torch
import os


from torch import nn
from torch.utils.data import random_split, DataLoader
from torchvision import datasets

data_path = os.path.join(os.path.pardir, os.path.pardir, "_00_data", "i_mnist")
mnist_train_images = datasets.MNIST(data_path, train=True, download=True)
mnist_test_images = datasets.MNIST(data_path, train=False, download=True)

print(len(mnist_train_images), len(mnist_test_images))  # >>> 60000 10000
```

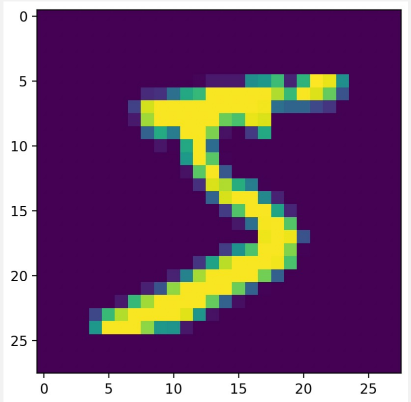
link_dl/_01_code/_06_fcn_best_practice/a_data_mnist.py

The MNIST Dataset

◈The MNIST Dataset in PyTorch




```
fig = plt.figure(figsize=(8, 3))
num_classes = 10
for i in range(num_classes):
    ax = fig.add_subplot(2, 5, 1 + i, xticks=[], yticks=[])
    ax.set_title(i)
    img = next((img for img, label in mnist_train_images if label == i))
    plt.imshow(img)
plt.show()

img, label = mnist_train_images[0]
print(type(img))                 # >>> <class 'PIL.Image.Image'>
print(label)                     # >>> 5
plt.imshow(img)
plt.show()

img = np.array(img)
print(type(img))                 # >>> <class 'numpy.ndarray'>
print(img.shape)                 # >>> (28, 28)
```



The MNIST Dataset

◈The MNIST Dataset in PyTorch

– Three Datasets: mnist_train, mnist_validation, mnist_test

```
from torchvision import transforms

mnist_train = datasets.MNIST(
  data_path, train=True, download=False, transform=transforms.ToTensor()
)
mnist_train, mnist_validation = random_split(mnist_train, [55_000, 5_000])

mnist_test = datasets.MNIST(
  data_path, train=False, download=False, transform=transforms.ToTensor()
)

print(len(mnist_train), len(mnist_validation), len(mnist_test))  # >>> 55000 5000 10000

img_t, _ = mnist_train[0]
print(type(img_t))                   # >>> <class 'torch.Tensor'>
print(img_t.shape)                   # >>> torch.Size([1, 28, 28])
print(img_t.min(), img_t.max())      # >>> tensor(0.) tensor(1.)
```

The MNIST Dataset

◈The MNIST Dataset in PyTorch

– It is good practice to normalize the dataset so that the channel has zero mean and unitary standard deviation

– The values of mean and stdev must be computed offline

```
# Let's stack all the tensors returned by the dataset along an extra dimension
imgs = torch.stack([img_t for img_t, _ in mnist_train], dim=3)
print(imgs.shape)
# >>> torch.Size([1, 28, 28, 60000])

print(imgs.view(1, -1).mean(dim=-1))     # Mean over every values per the channel
# >>> tensor([0.1307])

print(imgs.view(1, -1).std(dim=-1))      # Stdev over every values per the channel
# >>> tensor([0.3081])
```

The MNIST Dataset

◈The MNIST Dataset in PyTorch

– Normalizing data (use '**transforms.Normalize**') `(v_n[c] = (v[c] - mean[c]) / stdev[c])`

– Keep the data in the same range across channels and choose activation functions that are linear between -1 and 1 → it is more likely that neurons have nonzero gradients → Fast learning

```
mnist_transforms = nn.Sequential(
  transforms.ConvertImageDtype(torch.float),
  transforms.Normalize(mean=0.1307, std=0.3081),
  nn.Flatten()
)

train_data_loader = DataLoader(dataset=mnist_train, batch_size=32, shuffle=True)

for idx, train_batch in enumerate(train_data_loader):
    input, target = train_batch
    transformed_input = mnist_transforms(input)
    if idx == 0:
        print(input.shape, transformed_input.shape)
        # >>> torch.Size([32, 1, 28, 28]) torch.Size([32, 784])
```

The Cifar10 Dataset

The CIFAR-10 Dataset

◈The CIFAR-10 Dataset Description

– It consists of 60,000 32x32 color images in 10 classes, with 6000 images per class

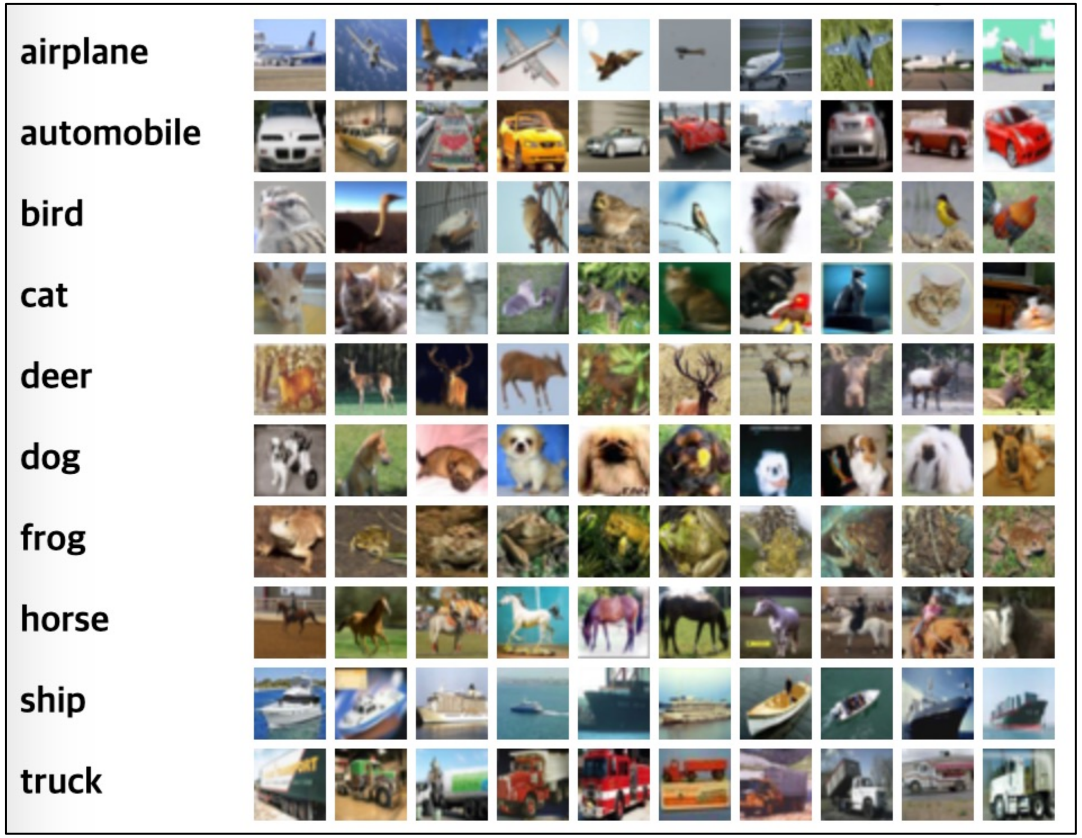
– There are 50,000 training images and 10,000 validation (or test) images

– Here are the classes in the dataset, as well as 10 random images from each

- The classes are completely mutually exclusive.
- There is no overlap between automobiles and trucks.
  - ➢ "Automobile" includes sedans, SUVs, etc.
  - ➢ "Truck" includes only big trucks

– Website

- https://www.cs.toronto.edu/~kriz/cifar.html



[Source] https://www.cs.toronto.edu/~kriz/cifar.html

The CIFAR-10 Dataset

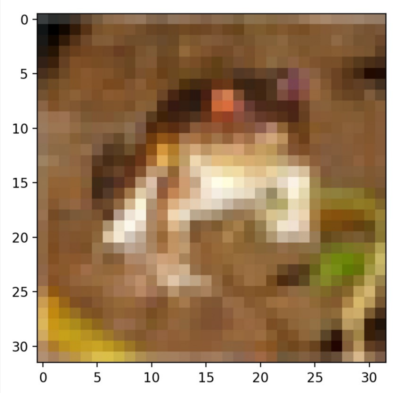
◈The CIFAR-10 Dataset in PyTorch

```
import numpy as np
from matplotlib import pyplot as plt
import torch
import os


from torch import nn
from torch.utils.data import random_split, DataLoader
from torchvision import datasets

data_path = os.path.join(os.path.pardir, os.path.pardir, "_00_data", "j_cifar10")
cifar10_train_images = datasets.CIFAR10(data_path, train=True, download=True)
cifar10_test_images = datasets.CIFAR10(data_path, train=False, download=True)

print(len(cifar10_train_images), len(cifar10_test_images))  # >>> 50000 10000
# >>> 50000 10000

class_names = [
  'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck’
]
```

The CIFAR-10 Dataset

◈The CIFAR-10 Dataset in PyTorch


```
fig = plt.figure(figsize=(8, 3))
num_classes = 10
for i in range(num_classes):
    ax = fig.add_subplot(2, 5, 1 + i, xticks=[], yticks=[])
    ax.set_title(class_names[i])
    img = next((img for img, label in cifar10_train if label == i))
    plt.imshow(img)
plt.show()

img, label = cifar10_train_images[0]
print(type(img))                      # >>> <class 'PIL.Image.Image'>
print(label, class_names[label])  # >>> 6 frog
plt.imshow(img)
plt.show()

img = np.array(img)
print(type(img))                      # >>> <class 'numpy.ndarray'>
print(img.shape)                      # >>> (32, 32, 3)
```



The CIFAR-10 Dataset

◈The CIFAR-10 Dataset in PyTorch

– Three Datasets: cifar10_train, cifar10_validation, cifar10_test

```
from torchvision import transforms

cifar10_train = datasets.CIFAR10(
  data_path, train=True, download=False, transform=transforms.ToTensor()
)
cifar10_train, cifar10_validation = random_split(cifar10_train, [45_000, 5_000])

cifar10_test = datasets.CIFAR10(
  data_path, train=False, download=False, transform=transforms.ToTensor()
)

print(len(cifar10_train), len(cifar10_validation), len(cifar10_test))  # >>> 45000 5000 10000

img_t, _ = cifar10_train[99]
print(type(img_t))               # >>> <class 'torch.Tensor'>
print(img_t.shape)               # >>> torch.Size([3, 32, 32])
print(img_t.min(), img_t.max())  # >>> tensor(0.) tensor(0.8549)
```

The CIFAR-10 Dataset

◈The CIFAR-10 Dataset in PyTorch

– It is good practice to normalize the dataset so that each channel has zero mean and unitary standard deviation

– The values of mean and stdev must be computed offline

```
# Let's stack all the tensors returned by the dataset along an extra dimension
imgs = torch.stack([img_t for img_t, _ in cifar10_train], dim=3)
print(imgs.shape)
# >>> torch.Size([3, 32, 32, 50000])

print(imgs.view(3, -1).mean(dim=-1))     # Mean over every values per each of three channels
# >>> tensor([0.4914, 0.4822, 0.4465])

print(imgs.view(3, -1).std(dim=-1))      # Stdev over every values per each of three channels
# >>> tensor([0.2470, 0.2435, 0.2616])
```

The CIFAR-10 Dataset

◈The CIFAR-10 Dataset in PyTorch

– Normalizing data (use '**transforms.Normalize**') `(v_n[c] = (v[c] - mean[c]) / stdev[c])`

– Keep the data in the same range across channels and choose activation functions that are linear between -1 and 1 → it is more likely that neurons have nonzero gradients → Fast learning

```
cifar10_transforms = nn.Sequential(
  transforms.ConvertImageDtype(torch.float),
  transforms.Normalize(mean=(0.4915, 0.4823, 0.4468), std=(0.2470, 0.2435, 0.2616)),
  nn.Flatten()
)

train_data_loader = DataLoader(dataset=cifar10_train, batch_size=32, shuffle=True)

for idx, train_batch in enumerate(train_data_loader):
    input, target = train_batch
    transformed_input = cifar10_transforms(input)
    if idx == 0:
        print(input.shape, transformed_input.shape)
        # >>> torch.Size([32, 3, 32, 32]) torch.Size([32, 3072])
```

Trainer

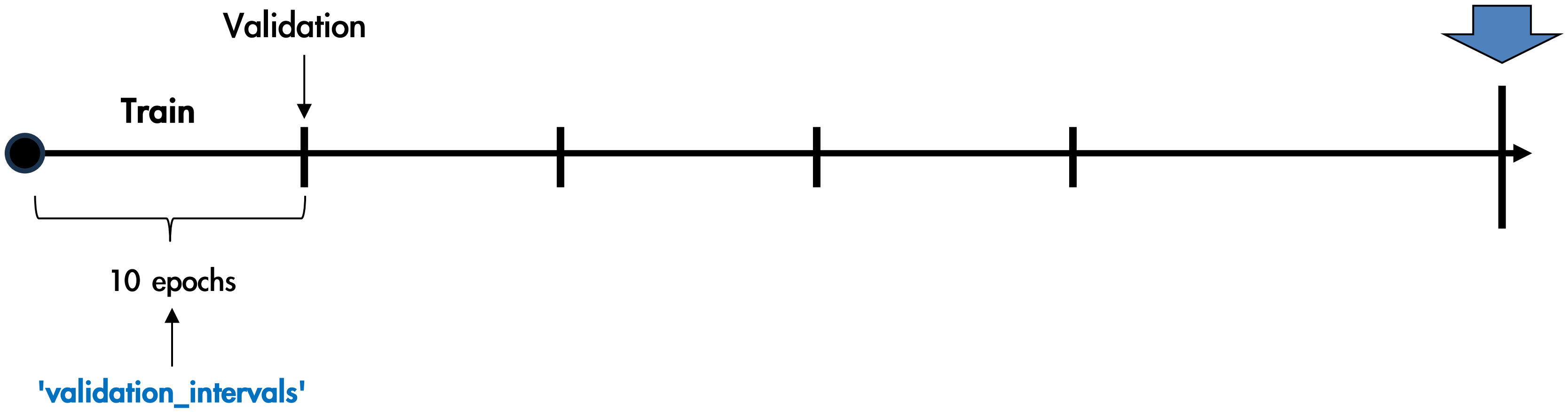
Train & Validation

◈ Train and Validation Setting

– Related Hyperparameters

- args.validation_intervals = 10
- args.print_epochs = 100

[Early Stop] Train is finished
when there is no further increase over a pre-d
efined
interval 'early_stop_patience (default: 200) '
in the average validation episode reward



Classification Trainer

◈Classification Trainer (init)

```
from datetime import datetime
import os
import torch
from torch import nn
from _01_code._99_common_utils.utils import strfdelta

class ClassificationTrainer:
  def __init__(
    self, project_name, model, optimizer, train_data_loader, validation_data_loader, transforms,
    run_time_str, wandb, device, checkpoint_file_path
  ):
    self.project_name = project_name
    self.model = model
    self.optimizer = optimizer
    self.train_data_loader = train_data_loader
    self.validation_data_loader = validation_data_loader
    self.transforms = transforms
    self.run_time_str = run_time_str
    self.wandb = wandb
    self.device = device
    self.checkpoint_file_path = checkpoint_file_path
    self.loss_fn = nn.CrossEntropyLoss()
```

Classification Trainer

◈Classification Trainer (do_train - 1/2)

```
class ClassificationTrainer:
  ...
  def do_train(self):
    self.model.train()   # Explained at 'Diverse Techniques' section

    loss_train = 0.0
    num_corrects_train = 0
    num_trained_samples = 0
    num_trains = 0

    for train_batch in self.train_data_loader:
      input_train, target_train = train_batch
      input_train = input_train.to(device=self.device)
      target_train = target_train.to(device=self.device)

      input_train = self.transforms(input_train)

      output_train = self.model(input_train)
      loss = self.loss_fn(output_train, target_train)
      loss_train += loss.item()
```

Classification Trainer

◈Classification Trainer (do_train - 2/2)

```
class ClassificationTrainer:
  ...

  def do_train(self):
    ...
    for train_batch in self.train_data_loader:
      ...
      ...
      predicted_train = torch.argmax(output_train, dim=1)
      num_corrects_train += torch.sum(torch.eq(predicted_train, target_train)).item()

      num_trained_samples += len(input_train)
      num_trains += 1

      self.optimizer.zero_grad()
      loss.backward()
      self.optimizer.step()

    train_loss = loss_train / num_trains
    train_accuracy = 100.0 * num_corrects_train / num_trained_samples
    return train_loss, train_accuracy
```

Classification Trainer

◈Classification Trainer (do_validation - 1/2)

```
class ClassificationTrainer:
  ...
  def do_validation(self):
    self.model.eval()   # Explained at 'Diverse Techniques' section

    loss_validation = 0.0
    num_corrects_validation = 0
    num_validated_samples = 0
    num_validations = 0

    with torch.no_grad():
      for validation_batch in self.validation_data_loader:
        input_validation, target_validation = validation_batch
        input_validation = input_validation.to(device=self.device)
        target_validation = target_validation.to(device=self.device)

        input_validation = self.transforms(input_validation)

        output_validation = self.model(input_validation)
        loss_validation += self.loss_fn(output_validation, target_validation).item()
```

Classification Trainer

◈Classification Trainer (do_validation - 2/2)

```
class ClassificationTrainer:
  ...

  def do_validation(self):
    ...
    with torch.no_grad():
      for validation_batch in self.validation_data_loader:
        ...

        predicted_validation = torch.argmax(output_validation, dim=1)
        num_corrects_validation += torch.sum(torch.eq(predicted_validation, target_validation)).item()

        num_validated_samples += len(input_validation)
        num_validations += 1

    validation_loss = loss_validation / num_validations
    validation_accuracy = 100.0 * num_corrects_validation / num_validated_samples

    return validation_loss, validation_accuracy
```

Classification Trainer

◈Classification Trainer (train_loop - 1/3)

```
class ClassificationTrainer:
  ...

  def train_loop(self):
    early_stopping = EarlyStopping(
      patience=self.wandb.config.early_stop_patience, project_name=self.project_name,
      checkpoint_file_path=self.checkpoint_file_path, run_time_str=self.run_time_str
    )
    n_epochs = self.wandb.config.epochs
    training_start_time = datetime.now()

    for epoch in range(1, n_epochs + 1):
      train_loss, train_accuracy = self.do_train()

      if epoch == 1 or epoch % self.wandb.config.validation_intervals == 0:
        validation_loss, validation_accuracy = self.do_validation()

        elapsed_time = datetime.now() - training_start_time
        epoch_per_second = epoch / elapsed_time.seconds
        message, early_stop = early_stopping.check_and_save(validation_loss, self.model)
```

Classification Trainer

◈Classification Trainer (train_loop - 2/3)

```
class ClassificationTrainer:
  ...
  def train_loop(self):
    ...
    for epoch in range(1, n_epochs + 1):
      train_loss, train_accuracy = self.do_train()

      if epoch == 1 or epoch % self.wandb.config.validation_intervals == 0:
        ...
        print(
          f"[Epoch {epoch:>3}] ”
          f"T_loss: {train_loss:6.3f}, ”
          f"T_accuracy: {train_accuracy:6.3f} | ”
          f"V_loss: {validation_loss:6.3f}, ”
          f"V_accuracy: {validation_accuracy:6.3f} | ”
          f"{message} | ”
          f"T_time: {strfdelta(elapsed_time, '%H:%M:%S')}, ”
          f"T_speed: {epoch_per_second:4.2f}”
        )
```

Classification Trainer

◈Classification Trainer (train_loop - 3/3)

```
class ClassificationTrainer:
  ...
  def train_loop(self):
    ...
    for epoch in range(1, n_epochs + 1):
      train_loss, train_accuracy = self.do_train()

      if epoch == 1 or epoch % self.wandb.config.validation_intervals == 0:
        ...
        self.wandb.log({
          "Epoch": epoch,
          "Training loss": train_loss,
          "Training accuracy (%)": train_accuracy,
          "Validation loss": validation_loss,
          "Validation accuracy (%)": validation_accuracy,
          "Training speed (epochs/sec.)": epoch_per_second,
        })

        if early_stop:
          break
    elapsed_time = datetime.now() - training_start_time
    print(f"Final training time: {strfdelta(elapsed_time, '%H:%M:%S')}")
```

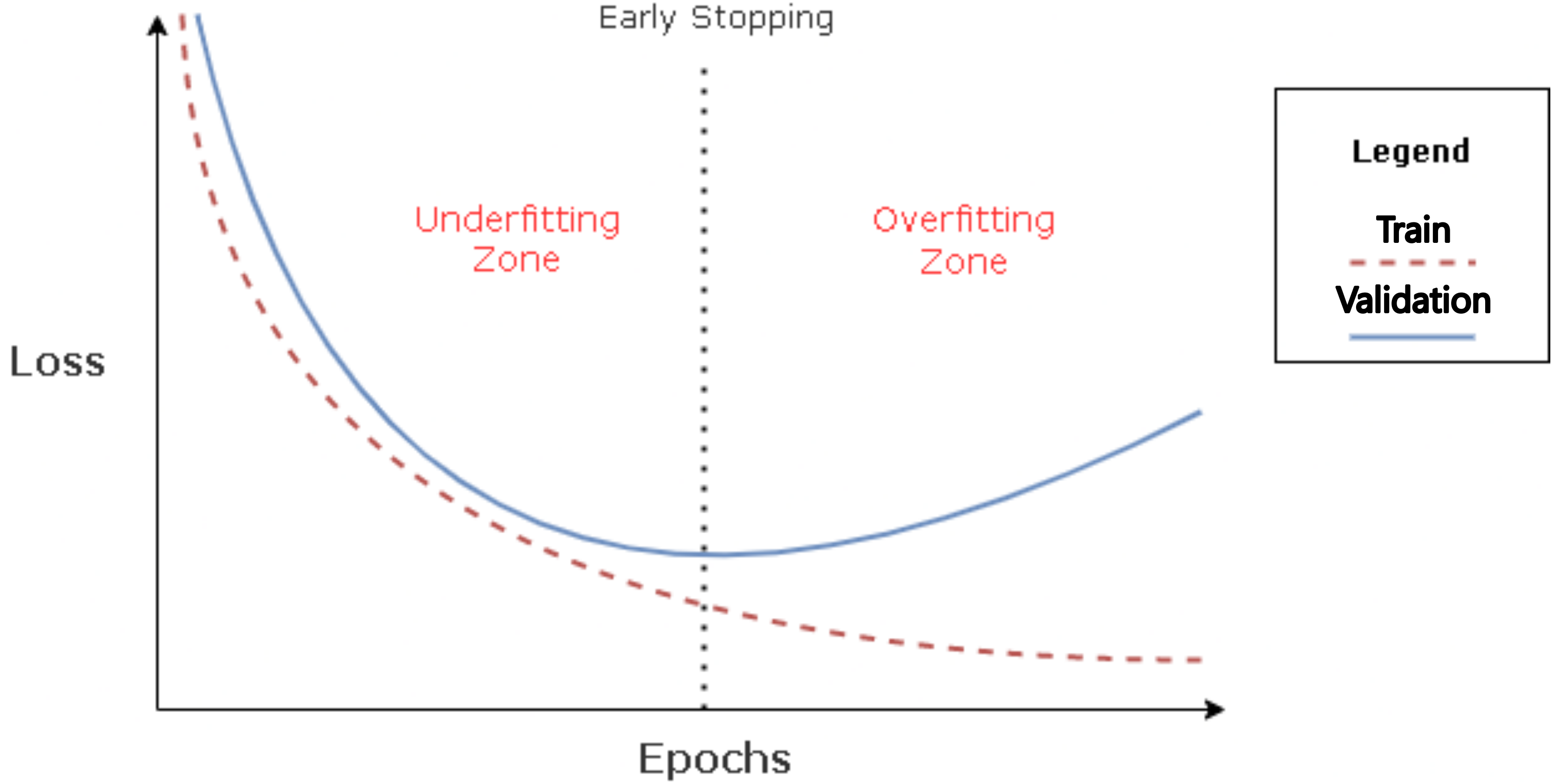
Early Stopping

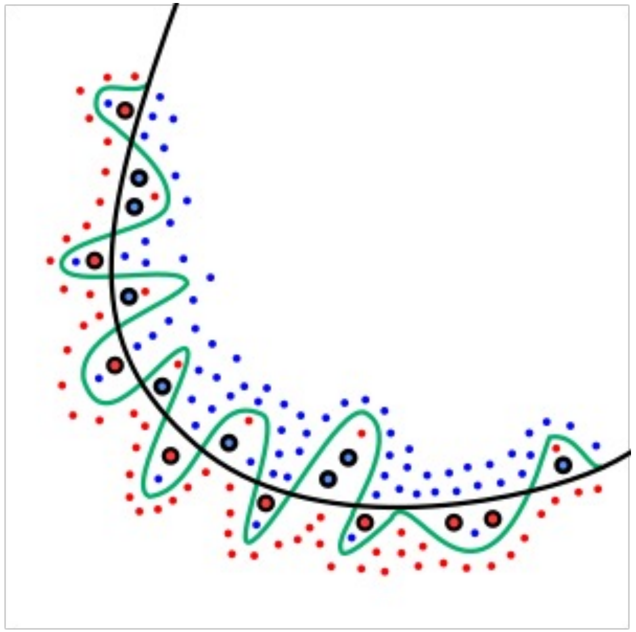
Early Stopping

> Early stopping should be used almost universally.
>
> — Ian Goodfellow et al., Deep Learning.

◈Early Stopping

- Early stops the training if validation loss doesn't improve after a given patience
- A form of [regularization](#) used to [avoid overfitting](#)





- The green line represents an overfitted model
- The black line represents a regularized model

Early Stopping

◈Early Stopping

– Early stops the training if validation loss doesn't improve after a given patience.

```
class EarlyStopping:
  def __init__(
    self, patience=10, delta=0.0001, project_name=None, checkpoint_file_path=None, run_time_str=None
  ):
    self.patience = patience # How long to wait after last time validation loss improved.
    self.counter = 0
    self.delta = delta  # Minimum change in the monitored quantity to qualify as an improvement

    self.val_loss_min = None

    self.file_path = os.path.join(
      checkpoint_file_path, f"{project_name}_checkpoint_{run_time_str}.pt"
    )

    self.latest_file_path = os.path.join(
      checkpoint_file_path, f"{project_name}_checkpoint_latest.pt"
    )
```

Early Stopping

◈Early Stopping

– Early stops the training if validation loss doesn't improve after a given patience.

```
class EarlyStopping:
  ...
  def check_and_save(self, new_validation_loss, model):
    early_stop = False
    message = None

    if self.val_loss_min is None:
      self.val_loss_min = new_validation_loss
      message = f'Early stopping is stated!'
    elif new_validation_loss < self.val_loss_min - self.delta:
      message = f'V_loss decreased ({self.val_loss_min:6.3f} --> {new_validation_loss:6.3f}). Savi
ng model...'
      self.save_checkpoint(new_validation_loss, model)
      self.val_loss_min = new_validation_loss
      self.counter = 0
```

Early Stopping

◈Early Stopping

– Early stops the training if validation loss doesn't improve after a given patience.

```
class EarlyStopping:
  ...
  def check_and_save(self, new_validation_loss, model):
    ...
    else:
      self.counter += 1
      message = f'Early stopping counter: {self.counter} out of {self.patience}'
      if self.counter >= self.patience:
        early_stop = True
        message += " *** TRAIN EARLY STOPPED! ***"

    return message, early_stop

  def save_checkpoint(self, val_loss, model):
    '''Saves model when validation loss decrease.'''
    torch.save(model.state_dict(), self.file_path)
    torch.save(model.state_dict(), self.latest_file_path)
    self.val_loss_min = val_loss
```

Tester

Tester

◈Tester

```
import os
import torch

class ClassificationTester:
  def __init__(self, project_name, model, test_data_loader, transforms):
    self.project_name = project_name
    self.model = model
    self.test_data_loader = test_data_loader

    self.transforms = transforms

    self.latest_file_path = os.path.join(
      os.path.dirname(os.path.abspath(__file__)),
      "checkpoints", f"{project_name}_checkpoint_latest.pt”
    )
    print("MODEL FILE: {0}".format(self.latest_file_path))

    self.model.load_state_dict(torch.load(self.latest_file_path, map_location=torch.device('cpu’))
  )
```

Tester

◈Tester

```
class ClassificationTester:
  ...
  def test(self):
    self.model.eval()    # Explained at 'Diverse Techniques' section
    num_corrects_test = 0
    num_tested_samples = 0
    with torch.no_grad():
      for test_batch in self.test_data_loader:
        input_test, target_test = test_batch
        input_test = self.transforms(input_test)
        output_test = self.model(input_test)

        predicted_test = torch.argmax(output_test, dim=1)
        num_corrects_test += torch.sum(torch.eq(predicted_test, target_test))
        num_tested_samples += len(input_test)

      test_accuracy = 100.0 * num_corrects_test / num_tested_samples

    print(f"TEST RESULTS: {test_accuracy:6.3f}%")
```

Tester

◈Tester

```
class ClassificationTester:
  ...

  def test_single(self, input_test):
    self.model.eval()    # Explained at 'Diverse Techniques' section

    with torch.no_grad():
      input_test = self.transforms(input_test)

      output_test = self.model(input_test)
      predicted_test = torch.argmax(output_test, dim=1)

    return predicted_test.item()
```

Argument Parser

Argument Parser

◈Argument Parser

```
import argparse

def get_parser():
  parser = argparse.ArgumentParser()

  parser.add_argument("--wandb", action=argparse.BooleanOptionalAction, default=False, help="True or False")

  parser.add_argument(
    "-b", "--batch_size", type=int, default=2_048, help="Batch size (int, default: 2_048)”
  )

  parser.add_argument(
    "-e", "--epochs", type=int, default=10_000, help="Number of training epochs (int, default:10_000)”
  )

  parser.add_argument(
    "-p", "--print_epochs", type=int, default=100,
    help="Number of printing epochs interval (int, default:100)”
  )
```

Argument Parser

◈Argument Parser

```
def get_parser():
  ...

  parser.add_argument(
    "-r", "--learning_rate", type=float, default=1e-3, help="Learning rate (float, default: 1e-3)”
  )

  parser.add_argument(
    "-v", "--validation_intervals", type=int, default=10,
    help="Number of training epochs between validations (int, default: 10)"
  )

  parser.add_argument(
    "-p", "--early_stop_patience", type=int, default=10,
    help="Number of early stop patience (int, default: 10)”
  )

  return parser
```

MNIST Train

MNIST Train

◆MNIST Train

```
import torch
from torch import nn, optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from datetime import datetime
import os
import wandb
from pathlib import Path

# BASE_PATH: /Users/yhhan/git/link_dl
BASE_PATH = str(Path(__file__).resolve().parent.parent.parent)
import sys
sys.path.append(BASE_PATH)

CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")
if not os.path.isdir(CHECKPOINT_FILE_PATH):
  os.makedirs(os.path.join(CURRENT_FILE_PATH, "checkpoints"))

from _01_code._99_common_utils.utils import is_linux, is_windows, get_num_cpu_cores
from _01_code._06_fcn_best_practice.c_trainer import ClassificationTrainer
from _01_code._06_fcn_best_practice.e_parser import get_parser
```

MNIST Train

◈MNIST Train (get_data - 1/2)

```
def get_data(flatten=False):
  data_path = os.path.join(os.path.pardir, os.path.pardir, "_00_data", "i_mnist")

  mnist_train = datasets.MNIST(
    data_path, train=True, download=True, transform=transforms.ToTensor()
  )
  mnist_train, mnist_validation = random_split(mnist_train, [55_000, 5_000])

  print("Num Train Samples: ", len(mnist_train))            # >>> 55000
  print("Num Validation Samples: ", len(mnist_validation))  # >>> 5000

  num_data_loading_workers = get_num_cpu_cores() if is_linux() or is_windows() else 0
  print("Number of Data Loading Workers:", num_data_loading_workers)  # >>> 0

  train_data_loader = DataLoader(
    dataset=mnist_train, batch_size=wandb.config.batch_size, shuffle=True,
    pin_memory=True, num_workers=num_data_loading_workers
  )
```

MNIST Train

◈MNIST Train (get_data - 2/2)

```
def get_data(flatten=False):
  ...
  validation_data_loader = DataLoader(
    dataset=mnist_validation, batch_size=wandb.config.batch_size,
    pin_memory=True, num_workers=num_data_loading_workers
  )

  mnist_transforms = nn.Sequential(
    transforms.ConvertImageDtype(torch.float),
    transforms.Normalize(mean=0.1307, std=0.3081),
    nn.Flatten()
  )

  if flatten:
    mnist_transforms.append(
      nn.Flatten()
    )

  return train_data_loader, validation_data_loader, mnist_transforms
```

MNIST Train

◈MNIST Train (get_model)

```
def get_model():
  class MyModel(nn.Module):
    def __init__(self, n_input, n_output):
      super().__init__()

      self.model = nn.Sequential(
        nn.Linear(n_input, 256),
        nn.ReLU(),
        nn.Linear(256, 256),
        nn.ReLU(),
        nn.Linear(256, n_output),
      )

    def forward(self, x):
      x = self.model(x)
      return x

  my_model = MyModel(n_input=784, n_output=10) # 1 * 28 * 28 = 784
  return my_model
```

MNIST Train

◈MNIST Train (main - 1/3)

```
def main(args):
  run_time_str = datetime.now().astimezone().strftime('%Y-%m-%d_%H-%M-%S’)

  config = {
    'epochs': args.epochs,
    'batch_size': args.batch_size,
    'validation_intervals': args.validation_intervals,
    'learning_rate': args.learning_rate
    'early_stop_patience': args.early_stop_patience
  }
  project_name = "fcn_mnist"
  wandb.init(
    mode="online" if args.wandb else "disabled",
    project=project_name,
    notes="mnist experiment",
    tags=["fcn", "mnist"],
    name=run_time_str,
    config=config
  )
```

MNIST Train

◈MNIST Train (main - 2/3)

```
def main(args):
  ...

  print(args) # Namespace(use_wandb=False, epochs=10000, batch_size=2048, learning_rate=0.001, validation_intervals=10)
  print(wandb.config) # {'epochs': 10000, 'batch_size': 2048, 'validation_intervals': 10, 'learning_rate': 0.001}

  device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
  print(f"Training on device {device}.")  # >>> Training on device cpu.

  train_data_loader, validation_data_loader, mnist_transforms = get_data(flatten=True)

  model = get_model()
  model.to(device)
  wandb.watch(model)

  optimizer = optim.SGD(model.parameters(), lr=wandb.config.learning_rate)
```

MNIST Train

◈MNIST Train (main - 3/3)

```
def main(args):
  ...

  classification_trainer = ClassificationTrainer(
    project_name, model, optimizer, train_data_loader, validation_data_loader, mnist_transforms,
    run_time_str, wandb, device, CHECKPOINT_FILE_PATH
  )

  classification_trainer.train_loop()

  wandb.finish()


if __name__ == "__main__":
  parser = get_parser()
  args = parser.parse_args()
  main(args)
```

MNIST Train

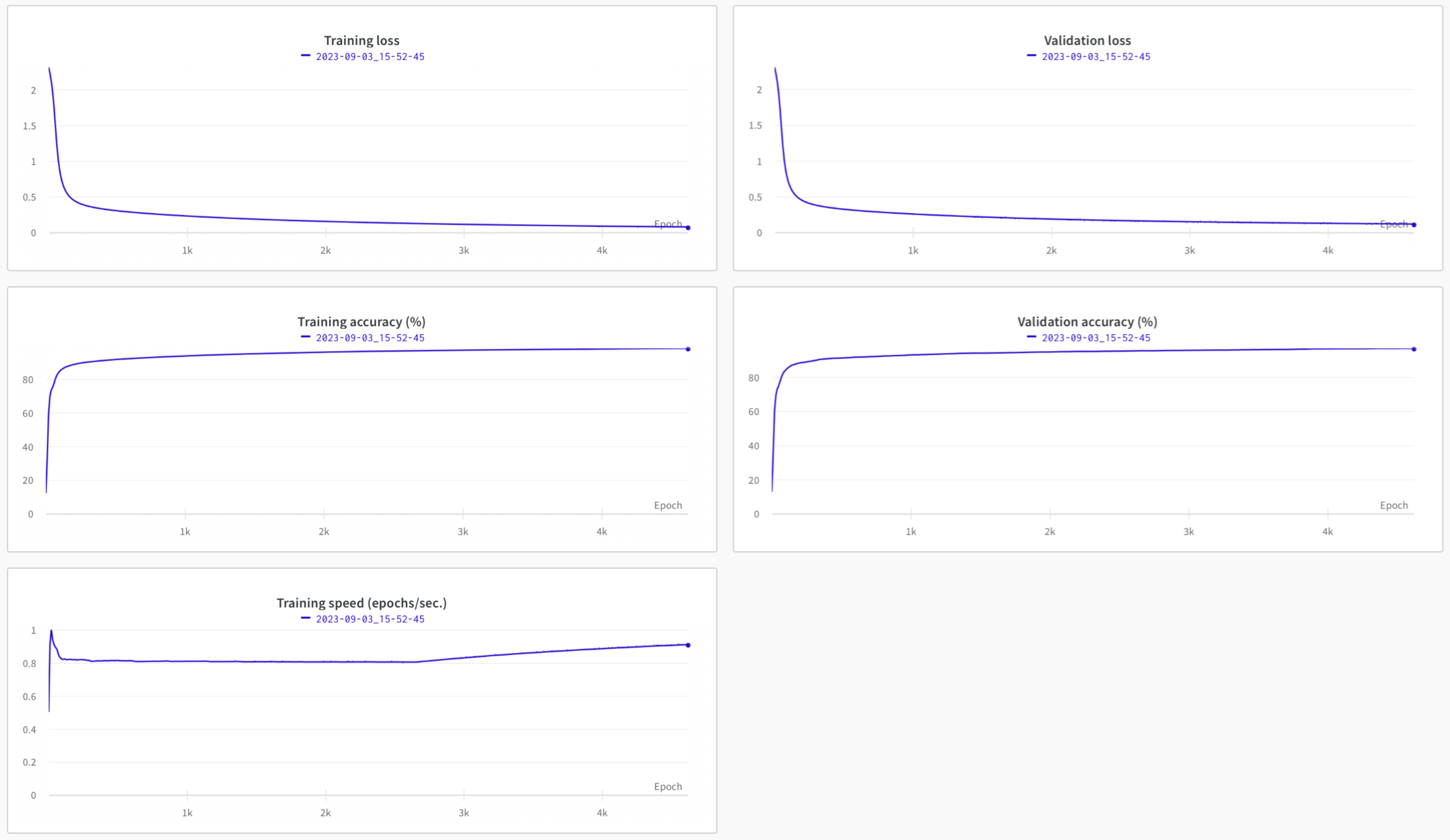
◈MNIST Train (console)

```
[Epoch    1] T_loss:  2.309, T_accuracy: 11.822 | V_loss:  2.303, V_accuracy: 12.520 | Early stopping is stated! | T_time: 00:00:02, T_speed: 0.50
[Epoch   10] T_loss:  2.222, T_accuracy: 31.411 | V_loss:  2.216, V_accuracy: 33.100 | V_loss decreased ( 2.303 -->  2.216). Saving model... | T_time: 00:00:11, T_speed: 0.91
[Epoch   20] T_loss:  2.109, T_accuracy: 58.420 | V_loss:  2.101, V_accuracy: 59.380 | V_loss decreased ( 2.216 -->  2.101). Saving model... | T_time: 00:00:20, T_speed: 1.00
[Epoch   30] T_loss:  1.952, T_accuracy: 69.727 | V_loss:  1.943, V_accuracy: 69.100 | V_loss decreased ( 2.101 -->  1.943). Saving model... | T_time: 00:00:32, T_speed: 0.94
[Epoch   40] T_loss:  1.734, T_accuracy: 73.380 | V_loss:  1.722, V_accuracy: 72.900 | V_loss decreased ( 1.943 -->  1.722). Saving model... | T_time: 00:00:44, T_speed: 0.91
[Epoch   50] T_loss:  1.468, T_accuracy: 74.993 | V_loss:  1.457, V_accuracy: 74.740 | V_loss decreased ( 1.722 -->  1.457). Saving model... | T_time: 00:00:56, T_speed: 0.89
[Epoch   60] T_loss:  1.212, T_accuracy: 77.355 | V_loss:  1.207, V_accuracy: 77.240 | V_loss decreased ( 1.457 -->  1.207). Saving model... | T_time: 00:01:08, T_speed: 0.88
[Epoch   70] T_loss:  1.011, T_accuracy: 80.213 | V_loss:  1.012, V_accuracy: 79.800 | V_loss decreased ( 1.207 -->  1.012). Saving model... | T_time: 00:01:22, T_speed: 0.85
[Epoch   80] T_loss:  0.866, T_accuracy: 82.098 | V_loss:  0.872, V_accuracy: 81.720 | V_loss decreased ( 1.012 -->  0.872). Saving model... | T_time: 00:01:36, T_speed: 0.83
[Epoch   90] T_loss:  0.763, T_accuracy: 83.491 | V_loss:  0.772, V_accuracy: 83.180 | V_loss decreased ( 0.872 -->  0.772). Saving model... | T_time: 00:01:49, T_speed: 0.83
[Epoch  100] T_loss:  0.687, T_accuracy: 84.553 | V_loss:  0.699, V_accuracy: 83.860 | V_loss decreased ( 0.772 -->  0.699). Saving model... | T_time: 00:02:02, T_speed: 0.82
[Epoch  110] T_loss:  0.630, T_accuracy: 85.295 | V_loss:  0.644, V_accuracy: 84.840 | V_loss decreased ( 0.699 -->  0.644). Saving model... | T_time: 00:02:14, T_speed: 0.82
[Epoch  120] T_loss:  0.586, T_accuracy: 85.940 | V_loss:  0.601, V_accuracy: 85.480 | V_loss decreased ( 0.644 -->  0.601). Saving model... | T_time: 00:02:26, T_speed: 0.82
...
...
[Epoch 4500] T_loss:  0.071, T_accuracy: 98.062 | V_loss:  0.112, V_accuracy: 96.620 | Early stopping counter: 2 out of 7 | T_time: 01:22:53, T_speed: 0.90
[Epoch 4510] T_loss:  0.071, T_accuracy: 98.067 | V_loss:  0.112, V_accuracy: 96.620 | Early stopping counter: 3 out of 7 | T_time: 01:23:02, T_speed: 0.91
[Epoch 4520] T_loss:  0.071, T_accuracy: 98.076 | V_loss:  0.112, V_accuracy: 96.640 | Early stopping counter: 4 out of 7 | T_time: 01:23:11, T_speed: 0.91
[Epoch 4530] T_loss:  0.071, T_accuracy: 98.075 | V_loss:  0.112, V_accuracy: 96.660 | Early stopping counter: 5 out of 7 | T_time: 01:23:20, T_speed: 0.91
[Epoch 4540] T_loss:  0.071, T_accuracy: 98.084 | V_loss:  0.112, V_accuracy: 96.660 | Early stopping counter: 6 out of 7 | T_time: 01:23:28, T_speed: 0.91
[Epoch 4550] T_loss:  0.071, T_accuracy: 98.084 | V_loss:  0.111, V_accuracy: 96.660 | V_loss decreased ( 0.112 -->  0.111). Saving model... | T_time: 01:23:37, T_speed: 0.91
[Epoch 4560] T_loss:  0.071, T_accuracy: 98.085 | V_loss:  0.111, V_accuracy: 96.660 | Early stopping counter: 1 out of 7 | T_time: 01:23:46, T_speed: 0.91
[Epoch 4570] T_loss:  0.070, T_accuracy: 98.091 | V_loss:  0.111, V_accuracy: 96.680 | Early stopping counter: 2 out of 7 | T_time: 01:23:55, T_speed: 0.91
[Epoch 4580] T_loss:  0.070, T_accuracy: 98.091 | V_loss:  0.111, V_accuracy: 96.660 | Early stopping counter: 3 out of 7 | T_time: 01:24:04, T_speed: 0.91
[Epoch 4590] T_loss:  0.070, T_accuracy: 98.098 | V_loss:  0.111, V_accuracy: 96.660 | Early stopping counter: 4 out of 7 | T_time: 01:24:13, T_speed: 0.91
[Epoch 4600] T_loss:  0.070, T_accuracy: 98.102 | V_loss:  0.111, V_accuracy: 96.660 | Early stopping counter: 5 out of 7 | T_time: 01:24:22, T_speed: 0.91
[Epoch 4610] T_loss:  0.070, T_accuracy: 98.104 | V_loss:  0.111, V_accuracy: 96.660 | Early stopping counter: 6 out of 7 | T_time: 01:24:31, T_speed: 0.91
[Epoch 4620] T_loss:  0.069, T_accuracy: 98.107 | V_loss:  0.110, V_accuracy: 96.660 | Early stopping counter: 7 out of 7 *** TRAIN EARLY STOPPED! *** | T_time: 01:24:40, T_speed: 0.91
Final training time: 01:24:40
```

MNIST Train

◈MNIST Train (wandb)

– https://wandb.ai/link-koreatech/dnn_mnist/workspace?workspace=user-link-koreatech



MNIST Test

MNIST Test

◈MNIST Test

```
import numpy as np
import torch
import os

from matplotlib import pyplot as plt
from torch import nn
from torchvision import transforms, datasets
from pathlib import Path
from torch.utils.data import DataLoader
# >>> BASE_PATH /Users/yhhan/git/link_dl
BASE_PATH = str(Path(__file__).resolve().parent.parent.parent)
CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")

import sys
sys.path.append(BASE_PATH)

from _01_code._06_fcn_best_practice.f_mnist_train_fcn import get_model
from _01_code._06_fcn_best_practice.d_tester import ClassificationTester
```

MNIST Test

◈MNIST Test (get_data)

```
def get_test_data(flatten=False):
  data_path = os.path.join(os.path.pardir, os.path.pardir, "_00_data", "i_mnist")

  mnist_test_images = datasets.MNIST(data_path, train=False, download=True)

  mnist_test = datasets.MNIST(
    data_path, train=False, download=False, transform=transforms.ToTensor()
  )
  test_data_loader = DataLoader(dataset=mnist_test, batch_size=len(mnist_test))

  mnist_transforms = nn.Sequential(
    transforms.ConvertImageDtype(torch.float),
    transforms.Normalize(mean=0.1307, std=0.3081) )

  if flatten:
    mnist_transforms.append(
      nn.Flatten()
    )
  return mnist_test_images, test_data_loader, mnist_transforms
```

MNIST Test

◈MNIST Test (main)

```
def main():
  mnist_test_images, test_data_loader, mnist_transforms = get_test_data(flatten=True)
  test_model = get_model()
  project_name = "fcn_mnist"
  classification_tester = ClassificationTester(
    project_name, test_model, test_data_loader, mnist_transforms, CHECKPOINT_FILE_PATH
  )
  classification_tester.test()
  img, label = mnist_test_images[0]
  print("     LABEL:", label)
  plt.imshow(img)
  plt.show()
  output = classification_tester.test_single(
    torch.tensor(np.array(mnist_test_images[0][0])).unsqueeze(dim=0).unsqueeze(dim=0)
  )  # (1, 1, 28, 28)
  print("PREDICTION:", output)

if __name__ == "__main__":
  main()
```

MNIST Test

◈MNIST Test

```
Files already downloaded and verified
TEST RESULTS: 97.968%

        LABEL: 5
PREDICTION: 5
```

Cifar10 Train

Cifar10 Train

Cifar10 Train

```
import torch
from torch import nn, optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from datetime import datetime
import os
import wandb
from pathlib import Path

# BASE_PATH: /Users/yhhan/git/link_dl
BASE_PATH = str(Path(__file__).resolve().parent.parent.parent)
import sys
sys.path.append(BASE_PATH)

CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")
if not os.path.isdir(CHECKPOINT_FILE_PATH):
  os.makedirs(os.path.join(CURRENT_FILE_PATH, "checkpoints"))

from _01_code._99_common_utils.utils import is_linux, is_windows, get_num_cpu_cores
from _01_code._06_fcn_best_practice.c_trainer import ClassificationTrainer
from _01_code._06_fcn_best_practice.e_parser import get_parser
```

Cifar10 Train

◈Cifar10 Train (get_data - 1/2)

```
def get_data(flatten=False):
  data_path = os.path.join(os.path.pardir, os.path.pardir, "_00_data", "j_cifar10")

  cifar10_train = datasets.CIFAR10(
    data_path, train=True, download=True, transform=transforms.ToTensor()
  )
  cifar10_train, cifar10_validation = random_split(cifar10_train, [45_000, 5_000])

  print("Num Train Samples: ", len(cifar10_train))            # >>> 45000
  print("Num Validation Samples: ", len(cifar10_validation))   # >>> 5000

  num_data_loading_workers = get_num_cpu_cores() if is_linux() or is_windows() else 0
  print("Number of Data Loading Workers:", num_data_loading_workers)   # >>> 0

  train_data_loader = DataLoader(
    dataset=cifar10_train, batch_size=wandb.config.batch_size, shuffle=True,
    pin_memory=True, num_workers=num_data_loading_workers
  )
```

Cifar10 Train

◈Cifar10 Train (get_data - 2/2)

```
def get_data(flatten=False):
  ...
  validation_data_loader = DataLoader(
    dataset=cifar10_validation, batch_size=wandb.config.batch_size,
    pin_memory=True, num_workers=num_data_loading_workers
  )

  cifar10_transforms = nn.Sequential(
    transforms.ConvertImageDtype(torch.float),
    transforms.Normalize(mean=(0.4915, 0.4823, 0.4468), std=(0.2470, 0.2435, 0.2616)),
    nn.Flatten(),
  )

  if flatten:
    cifar10_transforms.append(
      nn.Flatten()
    )

  return train_data_loader, validation_data_loader, cifar10_transforms
```

Cifar10 Train

◈Cifar10 Train (get_model)

```
def get_model():
  class MyModel(nn.Module):
    def __init__(self, n_input, n_output):
      super().__init__()

      self.model = nn.Sequential(
        nn.Linear(n_input, 256),
        nn.ReLU(),
        nn.Linear(256, 256),
        nn.ReLU(),
        nn.Linear(256, n_output),
      )

    def forward(self, x):
      x = self.model(x)
      return x

  my_model = MyModel(n_input=3_072, n_output=10) # 3 * 32 * 32 = 3072
  return my_model
```

Cifar10 Train

◈Cifar10 Train (main - 1/3)

```
def main(args):
  run_time_str = datetime.now().astimezone().strftime('%Y-%m-%d_%H-%M-%S’)

  config = {
    'epochs': args.epochs,
    'batch_size': args.batch_size,
    'validation_intervals': args.validation_intervals,
    'learning_rate': args.learning_rate
    'early_stop_patience': args.early_stop_patience
  }
  project_name = "fcn_cifar10"
  wandb.init(
    mode="online" if args.use_wandb else "disabled",
    project=project_name,
    notes="cifar10 experiment",
    tags=["fcn", "cifar10"],
    name=run_time_str,
    config=config
  )
```

Cifar10 Train

◈Cifar10 Train (main - 2/3)

```
def main(args):
  ...

  print(args) # Namespace(use_wandb=False, epochs=10000, batch_size=2048, learning_rate=0.001, validation_intervals=10)
  print(wandb.config) # {'epochs': 10000, 'batch_size': 2048, 'validation_intervals': 10, 'learning_rate': 0.001}

  device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
  print(f"Training on device {device}.")  # >>> Training on device cpu.

  train_data_loader, validation_data_loader, cifar10_transforms = get_data(flatten=True)

  model = get_model()
  model.to(device)
  wandb.watch(model)

  optimizer = optim.SGD(model.parameters(), lr=wandb.config.learning_rate)
```

Cifar10 Train

◈Cifar10 Train (main - 3/3)

```
def main(args):
  ...

  classification_trainer = ClassificationTrainer(
    project_name, model, optimizer, train_data_loader, validation_data_loader, cifar10_transforms,
    run_time_str, wandb, device, CHECKPOINT_FILE_PATH
  )
  classification_trainer.train_loop()

  wandb.finish()


if __name__ == "__main__":
  parser = get_parser()
  args = parser.parse_args()
  main(args)
```

Cifar10 Train

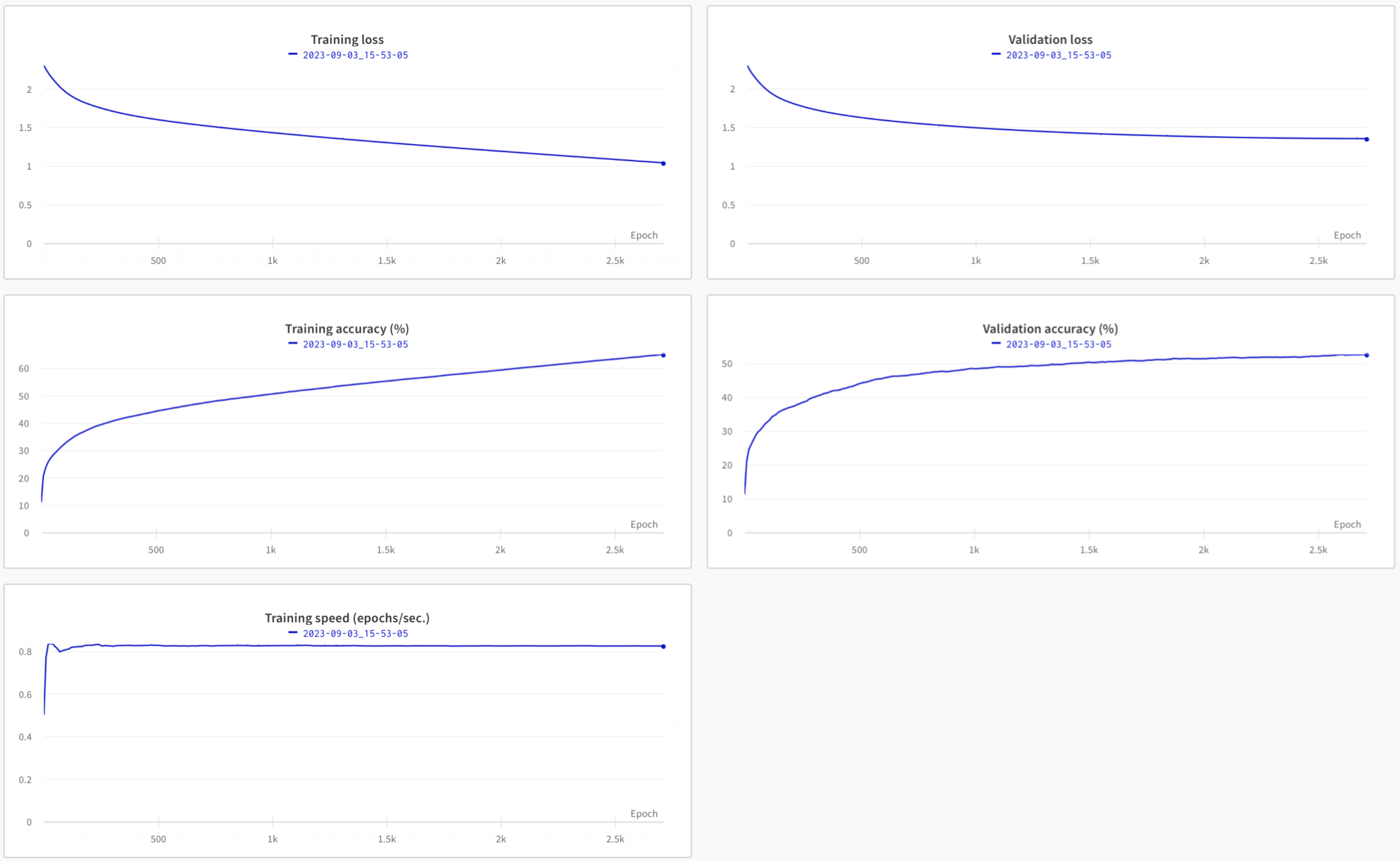
◈Cifar10 Train (console)

```
[Epoch    1] T_loss:  2.299, T_accuracy: 10.764 | V_loss:  2.296, V_accuracy: 11.100 | Early stopping is stated! | T_time: 00:00:02, T_speed: 0.50
[Epoch   10] T_loss:  2.247, T_accuracy: 20.342 | V_loss:  2.244, V_accuracy: 20.840 | V_loss decreased ( 2.296 -->  2.244). Saving model... | T_time: 00:00:13, T_speed: 0.77
[Epoch   20] T_loss:  2.200, T_accuracy: 23.464 | V_loss:  2.198, V_accuracy: 24.500 | V_loss decreased ( 2.244 -->  2.198). Saving model... | T_time: 00:00:24, T_speed: 0.83
[Epoch   30] T_loss:  2.159, T_accuracy: 25.404 | V_loss:  2.157, V_accuracy: 25.900 | V_loss decreased ( 2.198 -->  2.157). Saving model... | T_time: 00:00:36, T_speed: 0.83
[Epoch   40] T_loss:  2.121, T_accuracy: 26.809 | V_loss:  2.120, V_accuracy: 27.300 | V_loss decreased ( 2.157 -->  2.120). Saving model... | T_time: 00:00:48, T_speed: 0.83
[Epoch   50] T_loss:  2.086, T_accuracy: 27.849 | V_loss:  2.086, V_accuracy: 28.680 | V_loss decreased ( 2.120 -->  2.086). Saving model... | T_time: 00:01:01, T_speed: 0.82
[Epoch   60] T_loss:  2.053, T_accuracy: 28.771 | V_loss:  2.053, V_accuracy: 29.640 | V_loss decreased ( 2.086 -->  2.053). Saving model... | T_time: 00:01:14, T_speed: 0.81
[Epoch   70] T_loss:  2.022, T_accuracy: 29.624 | V_loss:  2.023, V_accuracy: 30.220 | V_loss decreased ( 2.053 -->  2.023). Saving model... | T_time: 00:01:28, T_speed: 0.80
[Epoch   80] T_loss:  1.994, T_accuracy: 30.489 | V_loss:  1.996, V_accuracy: 31.080 | V_loss decreased ( 2.023 -->  1.996). Saving model... | T_time: 00:01:40, T_speed: 0.80
[Epoch   90] T_loss:  1.968, T_accuracy: 31.282 | V_loss:  1.971, V_accuracy: 31.960 | V_loss decreased ( 1.996 -->  1.971). Saving model... | T_time: 00:01:52, T_speed: 0.80
[Epoch  100] T_loss:  1.944, T_accuracy: 32.027 | V_loss:  1.948, V_accuracy: 32.540 | V_loss decreased ( 1.971 -->  1.948). Saving model... | T_time: 00:02:04, T_speed: 0.81
[Epoch  110] T_loss:  1.923, T_accuracy: 32.782 | V_loss:  1.927, V_accuracy: 33.120 | V_loss decreased ( 1.948 -->  1.927). Saving model... | T_time: 00:02:16, T_speed: 0.81
[Epoch  120] T_loss:  1.903, T_accuracy: 33.409 | V_loss:  1.909, V_accuracy: 34.040 | V_loss decreased ( 1.927 -->  1.909). Saving model... | T_time: 00:02:27, T_speed: 0.82
[Epoch  130] T_loss:  1.885, T_accuracy: 33.993 | V_loss:  1.891, V_accuracy: 34.460 | V_loss decreased ( 1.909 -->  1.891). Saving model... | T_time: 00:02:39, T_speed: 0.82
...
...
[Epoch 2600] T_loss:  1.059, T_accuracy: 63.998 | V_loss:  1.349, V_accuracy: 52.420 | Early stopping counter: 3 out of 7 | T_time: 00:52:38, T_speed: 0.82
[Epoch 2610] T_loss:  1.057, T_accuracy: 64.069 | V_loss:  1.348, V_accuracy: 52.480 | Early stopping counter: 4 out of 7 | T_time: 00:52:50, T_speed: 0.82
[Epoch 2620] T_loss:  1.055, T_accuracy: 64.204 | V_loss:  1.348, V_accuracy: 52.300 | Early stopping counter: 5 out of 7 | T_time: 00:53:04, T_speed: 0.82
[Epoch 2630] T_loss:  1.053, T_accuracy: 64.271 | V_loss:  1.348, V_accuracy: 52.400 | Early stopping counter: 6 out of 7 | T_time: 00:53:16, T_speed: 0.82
[Epoch 2640] T_loss:  1.051, T_accuracy: 64.371 | V_loss:  1.348, V_accuracy: 52.380 | V_loss decreased ( 1.349 -->  1.348). Saving model... | T_time: 00:53:28, T_speed: 0.82
[Epoch 2650] T_loss:  1.049, T_accuracy: 64.447 | V_loss:  1.348, V_accuracy: 52.400 | Early stopping counter: 1 out of 7 | T_time: 00:53:40, T_speed: 0.82
[Epoch 2660] T_loss:  1.047, T_accuracy: 64.513 | V_loss:  1.348, V_accuracy: 52.420 | Early stopping counter: 2 out of 7 | T_time: 00:53:52, T_speed: 0.82
[Epoch 2670] T_loss:  1.045, T_accuracy: 64.580 | V_loss:  1.348, V_accuracy: 52.420 | Early stopping counter: 3 out of 7 | T_time: 00:54:04, T_speed: 0.82
[Epoch 2680] T_loss:  1.043, T_accuracy: 64.673 | V_loss:  1.348, V_accuracy: 52.400 | Early stopping counter: 4 out of 7 | T_time: 00:54:17, T_speed: 0.82
[Epoch 2690] T_loss:  1.041, T_accuracy: 64.767 | V_loss:  1.347, V_accuracy: 52.460 | Early stopping counter: 5 out of 7 | T_time: 00:54:29, T_speed: 0.82
[Epoch 2700] T_loss:  1.038, T_accuracy: 64.802 | V_loss:  1.348, V_accuracy: 52.480 | Early stopping counter: 6 out of 7 | T_time: 00:54:40, T_speed: 0.82
[Epoch 2710] T_loss:  1.036, T_accuracy: 64.884 | V_loss:  1.347, V_accuracy: 52.500 | Early stopping counter: 7 out of 7 *** TRAIN EARLY STOPPED! *** | T_time: 00:54:52, T_speed: 0.82
Final training time: 00:54:52
```

Cifar10 Train

◆ Cifar10 Train (wandb)

– https://wandb.ai/link-koreatech/dnn_cifar10/workspace?workspace=user-link-koreatech



Cifar10 Test

Cifar10 Test

◈Cifar10 Test

```
import numpy as np
import torch
import os

from matplotlib import pyplot as plt
from torch import nn
from torchvision import transforms, datasets
from pathlib import Path
from torch.utils.data import DataLoader
# >>> BASE_PATH /Users/yhhan/git/link_dl
BASE_PATH = str(Path(__file__).resolve().parent.parent.parent)
CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")

import sys
sys.path.append(BASE_PATH)

from _01_code._06_fcn_best_practice.h_cifar10_train_fcn import get_model
from _01_code._06_fcn_best_practice.d_tester import ClassificationTester
```

Cifar10 Test

◈Cifar10 Test (get_data)

```
def get_test_data(flatten=False):
  data_path = os.path.join(os.path.pardir, os.path.pardir, "_00_data", "j_cifar10")
  cifar10_test_images = datasets.CIFAR10(data_path, train=False, download=True)
  cifar10_test = datasets.CIFAR10(
    data_path, train=False, download=False, transform=transforms.ToTensor()
  )
  test_data_loader = DataLoader(dataset=cifar10_test, batch_size=len(cifar10_test))

  cifar10_transforms = nn.Sequential(
    transforms.ConvertImageDtype(torch.float),
    transforms.Normalize(mean=0.1307, std=0.3081),
    nn.Flatten()
  )

  if flatten:
    cifar10_transforms.append(
      nn.Flatten()
    )
  return cifar10_test_images, test_data_loader, cifar10_transforms
```

Cifar10 Test

◈Cifar10 Test (main)

```
def main():
  cifar10_test_images, test_data_loader, cifar10_transforms = get_test_data(flatten=True)
  test_model = get_model()
  project_name = "fcn_cifar10"
  classification_tester = ClassificationTester(
    project_name, test_model, test_data_loader, cifar10_transforms, CHECKPOINT_FILE_PATH
  )
  classification_tester.test()
  img, label = cifar10_test_images[0]
  print("     LABEL:", label)
  plt.imshow(img)
  plt.show()
  output = classification_tester.test_single(
    torch.tensor(np.array(cifar10_test_images[0][0])).permute(2, 0, 1).unsqueeze(dim=0)
  ) # shape: (1, 3, 32, 32)
  print("PREDICTION:", output)

if __name__ == "__main__":
  main()
```

Cifar10 Test

◈Cifar10 Test

```
Files already downloaded and verified
TEST RESULTS: 43.460%

        LABEL: 6
PREDICTION: 0
```