Imports

```
!pip install torchmetrics
```

```
Collecting torchmetrics
  Downloading torchmetrics-1.4.0.post0-py3-none-any.whl (868 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/868.8 kB ? eta -:--:--
━━━━━━━━ ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 174.1/868.8 kB 5.0 MB/s eta
0:00:01 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  860.2/868.8 kB 15.3
MB/s eta 0:00:01 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 868.8/868.8
kB 11.7 MB/s eta 0:00:00
ent already satisfied: numpy>1.20.0 in /usr/local/lib/python3.10/dist-
packages (from torchmetrics) (1.25.2)
Requirement already satisfied: packaging>17.1 in
/usr/local/lib/python3.10/dist-packages (from torchmetrics) (24.0)
Requirement already satisfied: torch>=1.10.0 in
/usr/local/lib/python3.10/dist-packages (from torchmetrics)
(2.2.1+cu121)
Collecting lightning-utilities>=0.8.0 (from torchmetrics)
  Downloading lightning_utilities-0.11.2-py3-none-any.whl (26 kB)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from lightning-
utilities>=0.8.0->torchmetrics) (67.7.2)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from lightning-
utilities>=0.8.0->torchmetrics) (4.11.0)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from torch>=1.10.0-
>torchmetrics) (3.14.0)
Requirement already satisfied: sympy in
/usr/local/lib/python3.10/dist-packages (from torch>=1.10.0-
>torchmetrics) (1.12)
Requirement already satisfied: networkx in
/usr/local/lib/python3.10/dist-packages (from torch>=1.10.0-
>torchmetrics) (3.3)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from torch>=1.10.0-
>torchmetrics) (3.1.4)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.10/dist-packages (from torch>=1.10.0-
>torchmetrics) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch>=1.10.0-
>torchmetrics)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-
manylinux1_x86_64.whl (23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch>=1.10.0-
>torchmetrics)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-
```

```
manylinux1_x86_64.whl (823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch>=1.10.0-
>torchmetrics)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-
manylinux1_x86_64.whl (14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch>=1.10.0-
>torchmetrics)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-
manylinux1_x86_64.whl (731.7 MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch>=1.10.0-
>torchmetrics)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-
manylinux1_x86_64.whl (410.6 MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch>=1.10.0-
>torchmetrics)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-
manylinux1_x86_64.whl (121.6 MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch>=1.10.0-
>torchmetrics)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-
manylinux1_x86_64.whl (56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch>=1.10.0-
>torchmetrics)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-
manylinux1_x86_64.whl (124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch>=1.10.0-
>torchmetrics)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-
manylinux1_x86_64.whl (196.0 MB)
Collecting nvidia-nccl-cu12==2.19.3 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_nccl_cu12-2.19.3-py3-none-manylinux1_x86_64.whl
(166.0 MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch>=1.10.0-
>torchmetrics)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-
manylinux1_x86_64.whl (99 kB)
Requirement already satisfied: triton==2.2.0 in
/usr/local/lib/python3.10/dist-packages (from torch>=1.10.0-
>torchmetrics) (2.2.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-
cu12==11.4.5.107->torch>=1.10.0->torchmetrics)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl (21.1 MB)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.10.0-
>torchmetrics) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in
/usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.10.0-
>torchmetrics) (1.3.0)
```

```
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-
cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-
cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12,
nvidia-cublas-cu12, lightning-utilities, nvidia-cusparse-cu12, nvidia-
cudnn-cu12, nvidia-cusolver-cu12, torchmetrics
Successfully installed lightning-utilities-0.11.2 nvidia-cublas-cu12-
12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-
12.1.105 nvidia-cuda-runtime-cu12-12.1.105 nvidia-cudnn-cu12-8.9.2.26
nvidia-cufft-cu12-11.0.2.54 nvidia-curand-cu12-10.3.2.106 nvidia-
cusolver-cu12-11.4.5.107 nvidia-cusparse-cu12-12.1.0.106 nvidia-nccl-
cu12-2.19.3 nvidia-nvjitlink-cu12-12.4.127 nvidia-nvtx-cu12-12.1.105
torchmetrics-1.4.0.post0
```

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader, random_split
import torch.backends.cudnn as cudnn

import torchvision
from torchvision.datasets import CIFAR10
from torchvision import transforms as T

from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np

from torchmetrics import Accuracy
from tqdm import tqdm
```

Dataset

Transform

```
transform_train = T.Compose([T.ToTensor(),
                             T.Normalize(mean=(0.4914, 0.4822,
0.4465),
                                         std=(0.2023, 0.1994,
0.2010))])

transform_test = T.Compose([T.ToTensor(),
                            T.Normalize(mean=(0.4914, 0.4822, 0.4465),
                                        std=(0.2023, 0.1994,
0.2010))])
```

CIFAR Dataset

```
train_set =
CIFAR10(root='/content/drive/MyDrive/colab_projects/Convolutional
Neural Networks', train=True,
                    download=True,
                    transform=transform_train)

test_set =
CIFAR10(root='/content/drive/MyDrive/colab_projects/Convolutional
Neural Networks', train=False,
                   download=True,
                   transform=transform_test)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
/content/drive/MyDrive/colab_projects/Convolutional Neural
Networks/cifar-10-python.tar.gz

100%|██████████| 170498071/170498071 [00:02<00:00, 75628087.40it/s]

Extracting /content/drive/MyDrive/colab_projects/Convolutional Neural
Networks/cifar-10-python.tar.gz to
/content/drive/MyDrive/colab_projects/Convolutional Neural Networks
Files already downloaded and verified
```

Dataloader

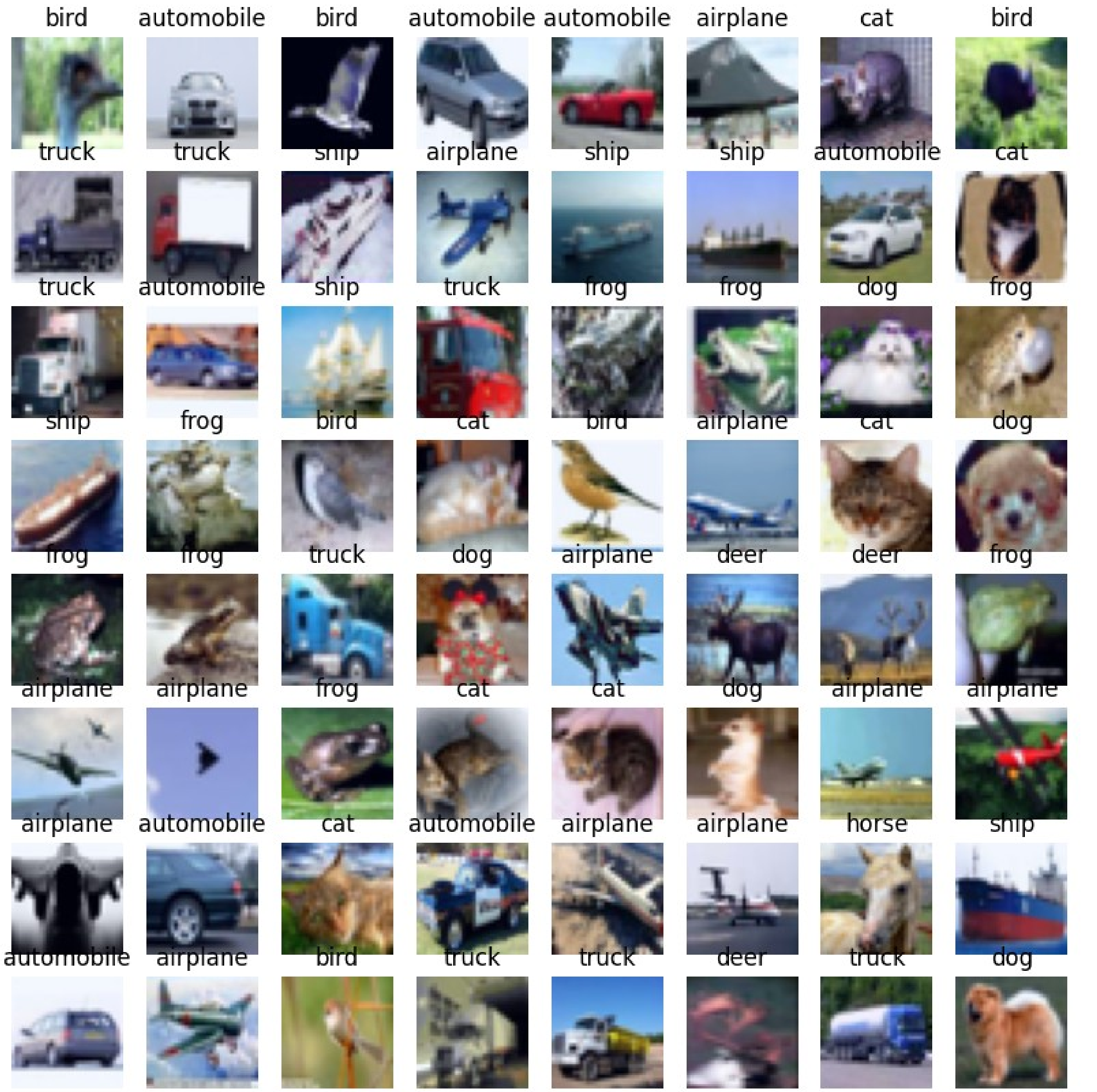
```
train_loader = DataLoader(train_set, batch_size=64, shuffle=True)
test_loader = DataLoader(test_set, batch_size=128, shuffle=False)

x, y = next(iter(train_loader))
print(x.shape)
print(y)
```

```
torch.Size([64, 3, 32, 32])
tensor([9, 9, 5, 4, 5, 1, 9, 9, 4, 2, 0, 8, 0, 4, 9, 1, 4, 8, 4, 4, 8,
9, 7, 3,
        3, 5, 3, 4, 4, 6, 2, 0, 6, 9, 6, 8, 0, 9, 4, 4, 4, 6, 9, 4, 3,
7, 0, 0,
        2, 2, 4, 5, 4, 0, 7, 5, 7, 2, 1, 1, 1, 4, 9, 8])
```

Visualize

```
def normalize_image(image):
  image_min = image.min()
  image_max = image.max()
  image.clamp_(min = image_min, max = image_max)
  image.add_(-image_min).div_(image_max - image_min + 1e-5)
  return image
```

```python
def plot_images(images, labels, classes, normalize=True):
  n_images = len(images)

  rows = int(np.sqrt(n_images))
  cols = int(np.sqrt(n_images))

  fig = plt.figure(figsize=(10, 10))

  for i in range(rows*cols):

    ax = fig.add_subplot(rows, cols, i+1)

    image = images[i]
    if normalize:
      image = normalize_image(image)

    ax.imshow(image.permute(1, 2, 0).cpu().numpy())
    ax.set_title(classes[labels[i]])
    ax.axis('off')

batch = next(iter(train_loader))
classes = train_set.classes
plot_images(batch[0], batch[1], classes)
```



Model

```
def conv3x3_bn_af(in_channels, out_channels):
  module = nn.Sequential(nn.Conv2d(in_channels, out_channels, 3,
padding=1),
                         nn.BatchNorm2d(out_channels),
                         nn.ReLU())
  return module
```

```
def CNN():
  network = nn.Sequential(conv3x3_bn_af(3, 64),
                          conv3x3_bn_af(64, 64),
                          nn.MaxPool2d(2, 2), # 16x16

                          conv3x3_bn_af(64, 128),
                          conv3x3_bn_af(128, 128),
                          nn.MaxPool2d(2, 2), # 8x8

                          conv3x3_bn_af(128, 256),
                          conv3x3_bn_af(256, 256),
                          nn.MaxPool2d(2, 2), # 4x4

                          conv3x3_bn_af(256, 512),
                          conv3x3_bn_af(512, 512),
                          nn.AdaptiveAvgPool2d(output_size=(1, 1)), # 1x1

                          nn.Flatten(),
                          nn.Linear(512, 10) # classifier
                      )

  return network
```

Device

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = CNN().to(device)
```

Loss & Optimizer

```
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

Utils

```
class AverageMeter(object):
    """Computes and stores the average and current value"""
    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
```

Functions

```
def train_one_epoch(model, train_loader, loss_fn, optimizer,
epoch=None):
  model.train()
  loss_train = AverageMeter()
  acc_train = Accuracy(task='multiclass', num_classes=10).to(device)
  with tqdm(train_loader, unit="batch") as tepoch:
    for inputs, targets in tepoch:
      if epoch is not None:
        tepoch.set_description(f"Epoch {epoch}")
      inputs = inputs.to(device)
      targets = targets.to(device)

      outputs = model(inputs)

      loss = loss_fn(outputs, targets)

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

      loss_train.update(loss.item())
      acc_train(outputs, targets.int())
      tepoch.set_postfix(loss=loss_train.avg,
                         accuracy=100.*acc_train.compute().item())
  return model, loss_train.avg, acc_train.compute().item()

def validation(model, test_loader, loss_fn):
  model.eval()
  with torch.no_grad():
    loss_valid = AverageMeter()
    acc_valid = Accuracy(task='multiclass', num_classes=10).to(device)
    for i, (inputs, targets) in enumerate(test_loader):
      inputs = inputs.to(device)
      targets = targets.to(device)

      outputs = model(inputs)
      loss = loss_fn(outputs, targets)
      loss_valid.update(loss.item())
      acc_valid(outputs, targets.int())
  return loss_valid.avg, acc_valid.compute().item()
```

Setting Hyperparameters

```
x_batch, y_batch = next(iter(train_loader))
outputs = model(x_batch.to(device))
loss = loss_fn(outputs, y_batch.to(device))
print(loss)
```

```
tensor(2.4139, grad_fn=<NllLossBackward0>)
```

```
_, mini_train_dataset = random_split(train_set, (len(train_set)-1000,
1000))
mini_train_loader = DataLoader(mini_train_dataset, 20)
```

```
num_epochs = 10
for epoch in range(num_epochs):
  model, _, _ = train_one_epoch(model, mini_train_loader, loss_fn,
optimizer, epoch)
```

```
Epoch 0: 100%|██████████| 50/50 [00:34<00:00,  1.44batch/s,
accuracy=26.8, loss=2.06]
Epoch 1: 100%|██████████| 50/50 [00:31<00:00,  1.57batch/s,
accuracy=42.9, loss=1.68]
Epoch 2: 100%|██████████| 50/50 [00:32<00:00,  1.56batch/s,
accuracy=54.1, loss=1.42]
Epoch 3: 100%|██████████| 50/50 [00:33<00:00,  1.48batch/s,
accuracy=65.6, loss=1.16]
Epoch 4: 100%|██████████| 50/50 [00:30<00:00,  1.61batch/s,
accuracy=77, loss=0.892]
Epoch 5: 100%|██████████| 50/50 [00:37<00:00,  1.33batch/s,
accuracy=87.9, loss=0.636]
Epoch 6: 100%|██████████| 50/50 [00:31<00:00,  1.57batch/s,
accuracy=95.1, loss=0.419]
Epoch 7: 100%|██████████| 50/50 [00:31<00:00,  1.60batch/s,
accuracy=97.1, loss=0.305]
Epoch 8: 100%|██████████| 50/50 [00:31<00:00,  1.57batch/s,
accuracy=99.3, loss=0.162]
Epoch 9: 100%|██████████| 50/50 [00:34<00:00,  1.43batch/s,
accuracy=99.9, loss=0.0866]
```

```
model = CNN().to(device)
lr = 0.05
wd = 1e-4
optimizer = optim.SGD(model.parameters(), lr=lr, weight_decay=wd)
```

```
loss_train_hist = []
loss_valid_hist = []

acc_train_hist = []
acc_valid_hist = []

best_loss_valid = torch.inf
epoch_counter = 0

num_epochs = 5

for epoch in range(num_epochs):
  # Train
  model, loss_train, acc_train = train_one_epoch(model,
                                                 train_loader,
                                                 loss_fn,
                                                 optimizer,
                                                 epoch)
  # Validation
  loss_valid, acc_valid = validation(model,
                                     test_loader,
                                     loss_fn)

  loss_train_hist.append(loss_train)
  loss_valid_hist.append(loss_valid)

  acc_train_hist.append(acc_train)
  acc_valid_hist.append(acc_valid)

  if loss_valid < best_loss_valid:
    torch.save(model, f'model.pt')
    best_loss_valid = loss_valid

  print(f'Valid: Loss = {loss_valid:.4}, Acc = {acc_valid:.4}')
  print()

  epoch_counter += 1
```

```
Epoch 0: 100%|██████████| 782/782 [25:52<00:00,  1.98s/batch,
accuracy=60.9, loss=1.08]

Valid: Loss = 1.466, Acc = 0.5337


Epoch 1: 100%|██████████| 782/782 [25:08<00:00,  1.93s/batch,
accuracy=76.9, loss=0.655]

Valid: Loss = 1.111, Acc = 0.6399


Epoch 2: 100%|██████████| 782/782 [25:35<00:00,  1.96s/batch,
accuracy=82.8, loss=0.496]
```

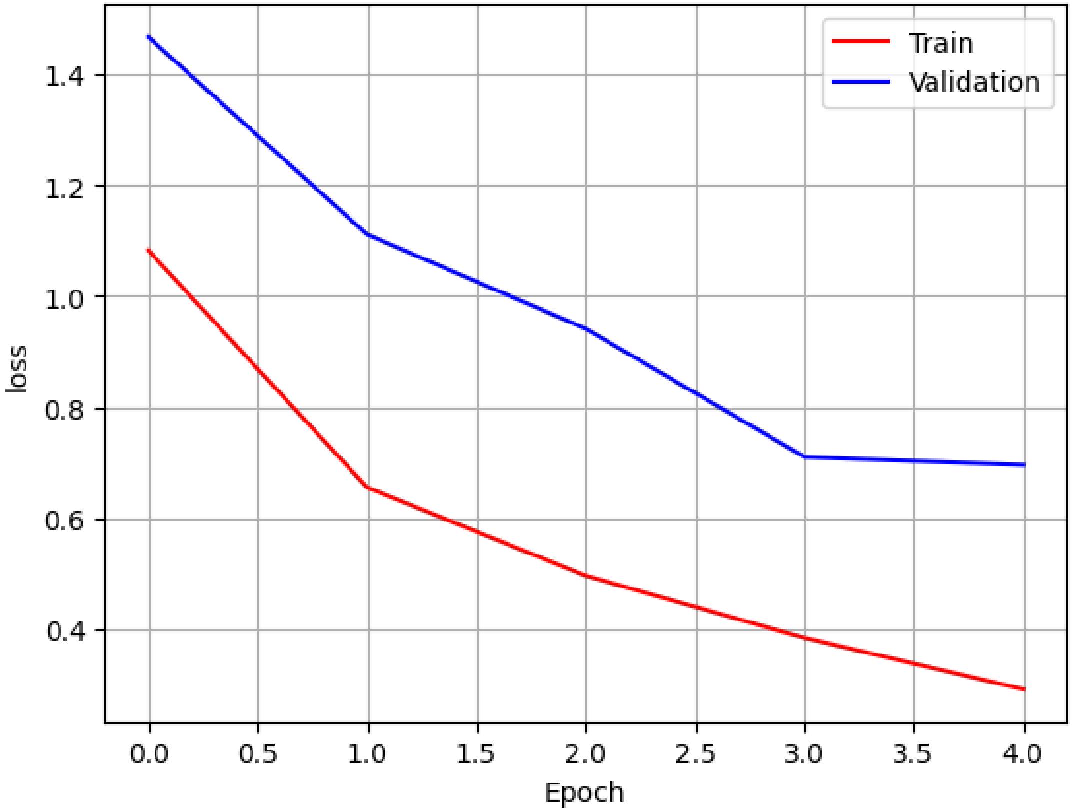
```
Valid: Loss = 0.9411, Acc = 0.6999


Epoch 3: 100%|██████████| 782/782 [26:31<00:00,  2.03s/batch,
accuracy=86.6, loss=0.384]

Valid: Loss = 0.71, Acc = 0.7661


Epoch 4: 100%|██████████| 782/782 [26:35<00:00,  2.04s/batch,
accuracy=89.9, loss=0.292]

Valid: Loss = 0.6959, Acc = 0.7803
```

Plot

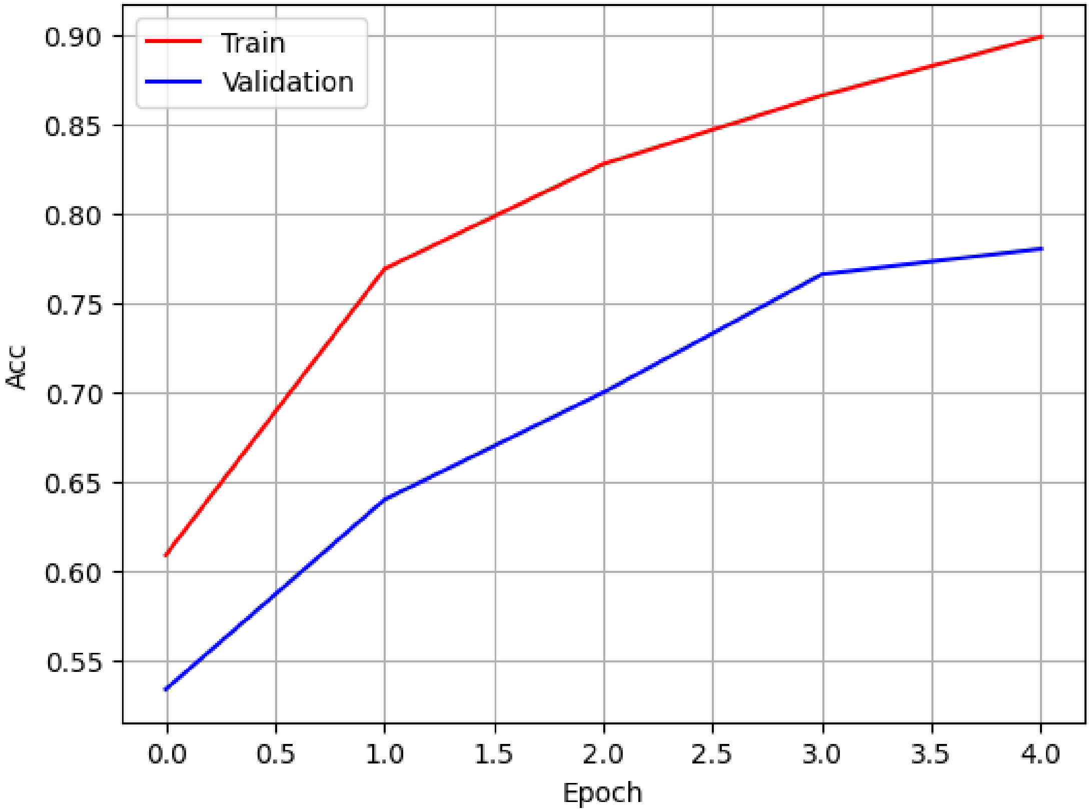
Plot

```
plt.plot(range(epoch_counter), loss_train_hist, 'r-', label='Train')
plt.plot(range(epoch_counter), loss_valid_hist, 'b-',
label='Validation')

plt.xlabel('Epoch')
plt.ylabel('loss')
plt.grid(True)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7972c4a38d90>
```



```
plt.plot(range(epoch_counter), acc_train_hist, 'r-', label='Train')
plt.plot(range(epoch_counter), acc_valid_hist, 'b-',
label='Validation')

plt.xlabel('Epoch')
plt.ylabel('Acc')
plt.grid(True)
plt.legend()

<matplotlib.legend.Legend at 0x7972b8435570>
```


Train
Validation
Acc
Epoch
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0