# 6.4 对预先训练好的卷积网络微调

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import requests

import torchvision.transforms as transforms

import torch.utils.data as Data

from torchvision import datasets

from torchvision.datasets import ImageFolder

import hiddenlayer as hl

import torch

from torch import nn

import torch.nn.functional as F

from torchvision import models

from torchvision import transforms

from PIL import Image

from tqdm import tqdm

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

print("使用设备：", device)

vgg16 = models.vgg16(pretrained=True) # imagenet 数据集预训练好的网络

# ImageNet数据集是一个大规模的用于图像识别任务的数据集，

# 包含超过1400万张标记图像，涵盖了超过2万个不同类别的物体。

vgg = vgg16.features

for param in vgg.parameters():

param.requires\_grad\_(False)

class MyVggModel(nn.Module):

def \_\_init\_\_(self):

super(MyVggModel, self).\_\_init\_\_()

self.vgg = vgg

self.classifier = nn.Sequential(

nn.Linear(25088, 512),

nn.ReLU(),

nn.Dropout(p=0.5),

nn.Dropout(p=0.2),

nn.Linear(512, 256),

nn.ReLU(),

nn.Dropout(p=0.5),

nn.Dropout(p=0.2),

nn.Linear(256, 10),

nn.Softmax(dim=1)

)

def forward(self, x):

x = self.vgg(x)

x = x.view(x.size(0), -1)

output = self.classifier(x)

return output

Myvggc = MyVggModel().to(device)

print(Myvggc)

# MyVggModel(

# (vgg): Sequential(

# (0): Conv2d(3, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (1): ReLU(inplace)

# (2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (3): ReLU(inplace)

# (4): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

# (5): Conv2d(64, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (6): ReLU(inplace)

# (7): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (8): ReLU(inplace)

# (9): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

# (10): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (11): ReLU(inplace)

# (12): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (13): ReLU(inplace)

# (14): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (15): ReLU(inplace)

# (16): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

# (17): Conv2d(256, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (18): ReLU(inplace)

# (19): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (20): ReLU(inplace)

# (21): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (22): ReLU(inplace)

# (23): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

# (24): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (25): ReLU(inplace)

# (26): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (27): ReLU(inplace)

# (28): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (29): ReLU(inplace)

# (30): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

# )

# (classifier): Sequential(

# (0): Linear(in\_features=25088, out\_features=512, bias=True)

# (1): ReLU()

# (2): Dropout(p=0.5)

# (3): Linear(in\_features=512, out\_features=256, bias=True)

# (4): ReLU()

# (5): Dropout(p=0.5)

# (6): Linear(in\_features=256, out\_features=10, bias=True)

# (7): Softmax()

# )

# )

train\_data\_transforms = transforms.Compose([

transforms.RandomResizedCrop(224), # 裁剪

transforms.RandomHorizontalFlip(), # 翻转

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

val\_data\_transform = transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

# 根据kaggle网站下载的数据集

train\_data\_dir = './deep Learning/archive/training/training'

train\_data = ImageFolder(root=train\_data\_dir, transform=train\_data\_transforms)

train\_data\_loader = Data.DataLoader(train\_data, batch\_size=32, shuffle=True)

val\_data\_dir = './deep Learning/archive/validation/validation'

val\_data = ImageFolder(root=val\_data\_dir, transform=val\_data\_transform)

val\_data\_loader = Data.DataLoader(val\_data, batch\_size=32, shuffle=True)

print('训练集样本数：', len(train\_data))

print('验证集样本数：', len(val\_data))

# 训练集样本数： 1097

# 验证集样本数： 272

for step, (b\_x, b\_y) in enumerate(train\_data\_loader): # enumerate 函数返回每个批次的索引（step）以及该批次的数据（b\_x 和 b\_y）。

if step > 0: # 只显示第一个批次的数据

break

mean = np.array([0.485, 0.456, 0.406])

std = np.array([0.229, 0.224, 0.225])

plt.figure(figsize=(12, 6))

for ii in np.arange(len(b\_y)): # 创建了一个循环，用于迭代每个批次中的所有图像。len(b\_y) 返回批次中标签的数量，也就是图像的数量。

plt.subplot(4, 8, ii + 1)

image = b\_x[ii, :, :, :].numpy().transpose((1, 2,

0)) # 这一行获取第 ii 个图像，并将其从 PyTorch 张量转换为 NumPy 数组。由于图像数据通常以通道优先的格式存储（通道数 x 高度 x 宽度），因此需要使用 transpose((1, 2, 0)) 来将其转换为更常见的格式（高度 x 宽度 x 通道数）

image = std \* image + mean # 这一行对图像进行反标准化操作。将标准化后的图像数据乘以标准差再加回均值，以恢复原始像素值。

image = np.clip(image, 0, 1) # 这一行使用 np.clip 函数将像素值限制在 0 到 1 的范围内

plt.imshow(image)

# plt.title(b\_y[ii].data.numpy()) # 这一行设置图像的标题为对应的标签值。注意，这里使用 .data.numpy() 将 PyTorch 张量转换为 NumPy 数组，以便设置为标题

# plt.title(b\_y[ii].item())

plt.title(str(int(b\_y[ii].item())))

plt.axis('off') # 关闭坐标轴的显示

plt.subplots\_adjust(hspace=0.3) # 调整子图之间的垂直间距，使它们之间有一定的空间，避免重叠

plt.show()

optimizer = torch.optim.Adam(Myvggc.parameters(), lr=0.003) # 因为这次任务的loss是从2.3降到1.7，从数值上来说减少得很少，所以可能Adam自适应原理迈的步子比较小，考虑更换优化器。

loss\_func = nn.CrossEntropyLoss().to(device) # 损失函数

train\_losses = []

val\_losses = []

train\_accs = []

val\_accs = []

# 对模型进行迭代训练，对所有的数据训练EPOCH轮

for epoch in range(50): # 遍历个epoch（训练轮次）

train\_loss\_epoch = 0 # 初始化训练损失为0

val\_loss\_epoch = 0 # 初始化验证损失为0

train\_corrects = 0 # 初始化训练正确预测数为0

val\_corrects = 0 # 初始化验证正确预测数为0

# 将模型设置为训练模式

Myvggc.train()

# 遍历训练数据加载器中的每个批次

for step, (b\_x, b\_y) in enumerate(tqdm(train\_data\_loader)):

b\_x, b\_y = b\_x.to(device), b\_y.to(device) # 将输入数据和标签移动到GPU（如果可用）

output = Myvggc(b\_x) # 前向传播：通过模型计算输出

loss = loss\_func(output, b\_y) # 计算损失

pre\_lab = torch.argmax(output, 1) # 获取预测标签

optimizer.zero\_grad() # 梯度清零

loss.backward() # 反向传播：计算梯度

optimizer.step() # 更新模型参数

train\_loss\_epoch += loss.item() \* b\_x.size(0) # 累加训练损失

train\_corrects += torch.sum(pre\_lab == b\_y.data) # 累加训练正确预测数

train\_loss = train\_loss\_epoch / len(train\_data.targets) # 计算平均训练损失

train\_losses.append(train\_loss) # 将训练损失记录到列表中

train\_acc = train\_corrects.double() / len(train\_data.targets) # 计算训练准确率

train\_accs.append(train\_acc) # 将训练准确率记录到列表中

# 将模型设置为评估模式，关闭梯度计算

Myvggc.eval()

with torch.no\_grad(): # 使用torch.no\_grad()上下文管理器，禁止梯度计算

# 遍历验证数据加载器中的每个批次

for step, (val\_x, val\_y) in enumerate(val\_data\_loader):

val\_x, val\_y = val\_x.to(device), val\_y.to(device) # 将验证数据和标签移动到GPU（如果可用）

output = Myvggc(val\_x) # 前向传播：通过模型计算输出

loss = loss\_func(output, val\_y) # 计算验证损失

pre\_lab = torch.argmax(output, 1) # 获取预测标签,output 是模型对输入数据 val\_x 的预测结果，它是一个形状为 (batch\_size, num\_classes) 的张量，其中每一行代表一个样本，每一列代表该样本属于对应类别的概率。

#torch.argmax(output, 1) 返回每一行的最大值的索引，即每个样本被预测为哪个类别的索引。这相当于获取每个样本的预测标签。例如，如果 output 的某一行是 [0.1, 0.5, 0.2, 0.2]，那么 torch.argmax(output, 1) 将返回 1，因为索引为 1 的元素（第二个元素）具有最高的值。

val\_loss\_epoch += loss.item() \* val\_x.size(0) # 累加验证损失,loss 是模型在当前批次上的损失值，它是一个标量张量。

# loss.item() 将张量转换为 Python 的标量值，这样就可以用于后续的数值计算。

# val\_x.size(0) 返回当前批次的样本数量。在 PyTorch 中，size() 方法返回张量的形状，size(0) 返回第一维的大小，即批次大小。

# loss.item() \* val\_x.size(0) 计算了当前批次的损失值乘以批次大小，这实际上是当前批次所有样本的损失总和。

val\_corrects += torch.sum(pre\_lab == val\_y.data) # 累加验证正确预测数

val\_loss = val\_loss\_epoch / len(val\_data.targets) # 计算平均验证损失

val\_losses.append(val\_loss) # 将验证损失记录到列表中

val\_acc = val\_corrects.double() / len(val\_data.targets) # 计算验证准确率

val\_accs.append(val\_acc) # 将验证准确率记录到列表中

# 打印当前 epoch 的训练损失、训练准确率、验证损失和验证准确率

print(f'Epoch [{epoch+1}], '

f'Train Loss: {train\_loss:.4f}, Train Acc: {train\_acc:.4f}, '

f'Val Loss: {val\_loss:.4f}, Val Acc: {val\_acc:.4f}')

plt.figure(figsize=(10, 5))

# 绘制训练和验证损失

plt.subplot(1, 2, 1)

plt.plot(train\_losses, label='Training Loss')

plt.plot(val\_losses, label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

# 绘制训练和验证准确率

plt.subplot(1, 2, 2)

plt.plot(train\_accs, label='Training Accuracy')

plt.plot(val\_accs, label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

# 显示图形

plt.tight\_layout()

plt.show()

print(train\_losses)

print(val\_losses)

print(train\_accs)

print(val\_accs)

# 在深度学习中，优化器是用来更新网络权重的方法，以便最小化损失函数。

# Adam优化器（Adaptive Moment Estimation）

# Adam 使用了一阶矩估计（动量）和二阶矩估计（无偏估计的平方梯度）来动态调整每个参数的学习率。

# 是一种自适应学习率的方法。

# 根据历史梯度信息自适应地调整每个参数的学习率，从而在训练初期使用较大的学习率快速收敛，在训练后期使用较小的学习率精细调整参数。

# SGD优化器（Stochastic Gradient Descent）

# SGD 是最简单的优化算法之一，它基于梯度下降的概念。

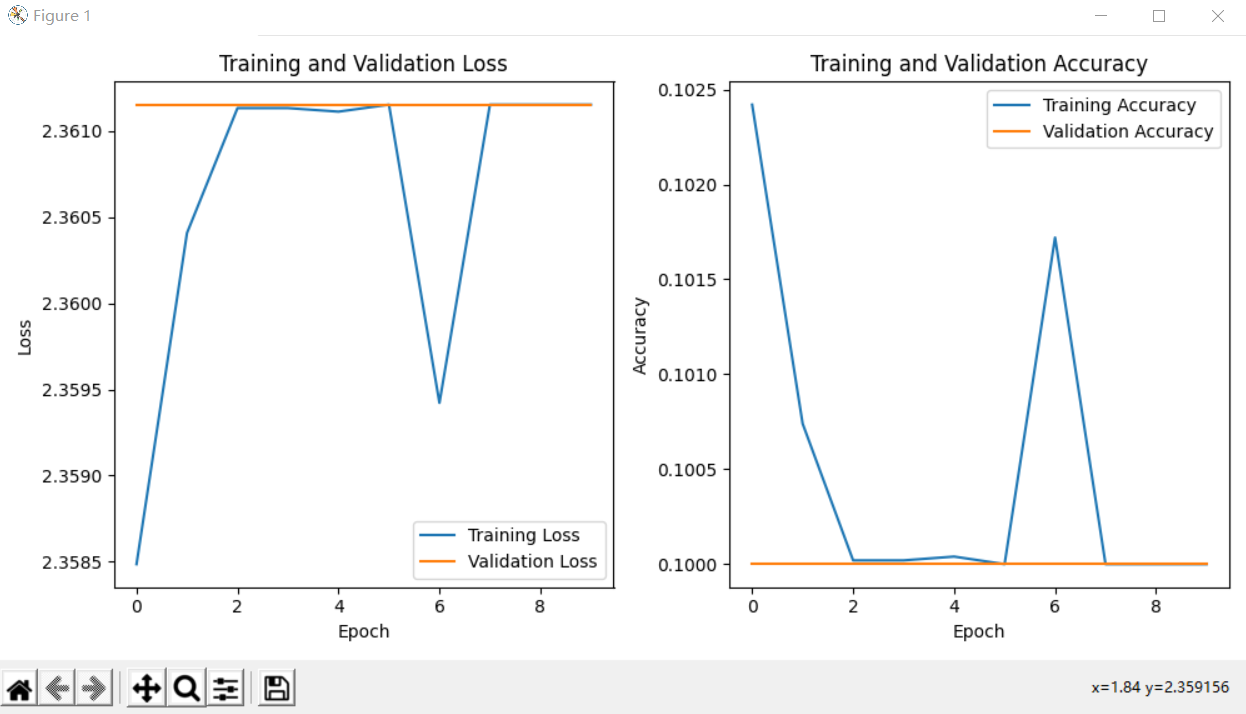
# 在每一步更新中，SGD 使用一个小批量（mini-batch）的样本梯度来更新权重，而不是整个数据集的梯度。

# 简单：实现简单，计算成本较低。

# 易于陷入局部极小值：在非凸函数中，SGD 容易陷入局部极小值。

# 学习率调整：通常需要手动调整学习率，以避免学习过快或过慢。

Adam:

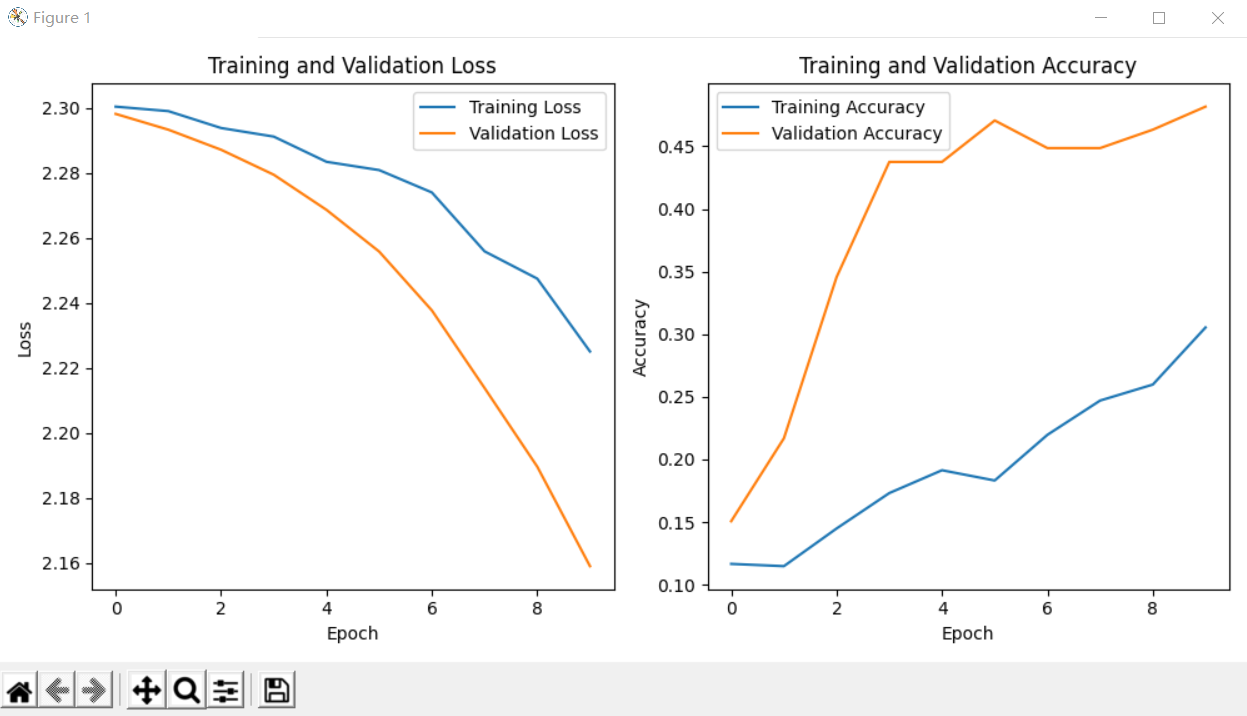


增大了学习率后：

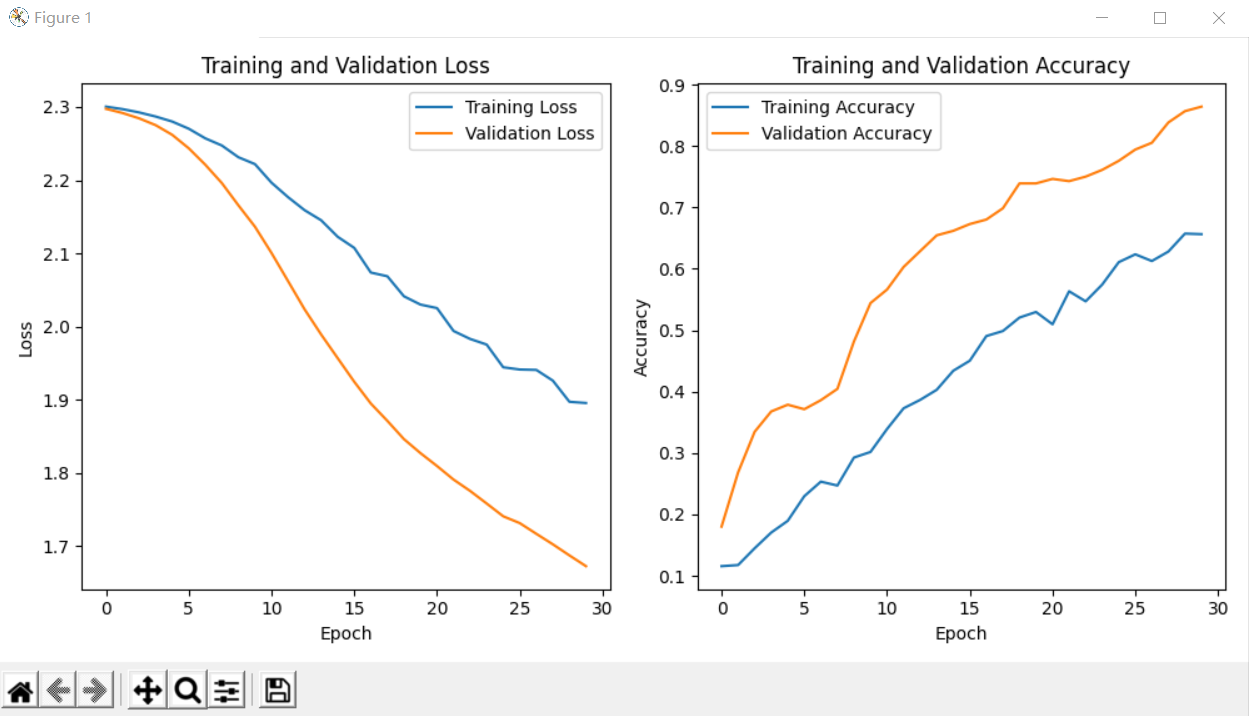


SGD：

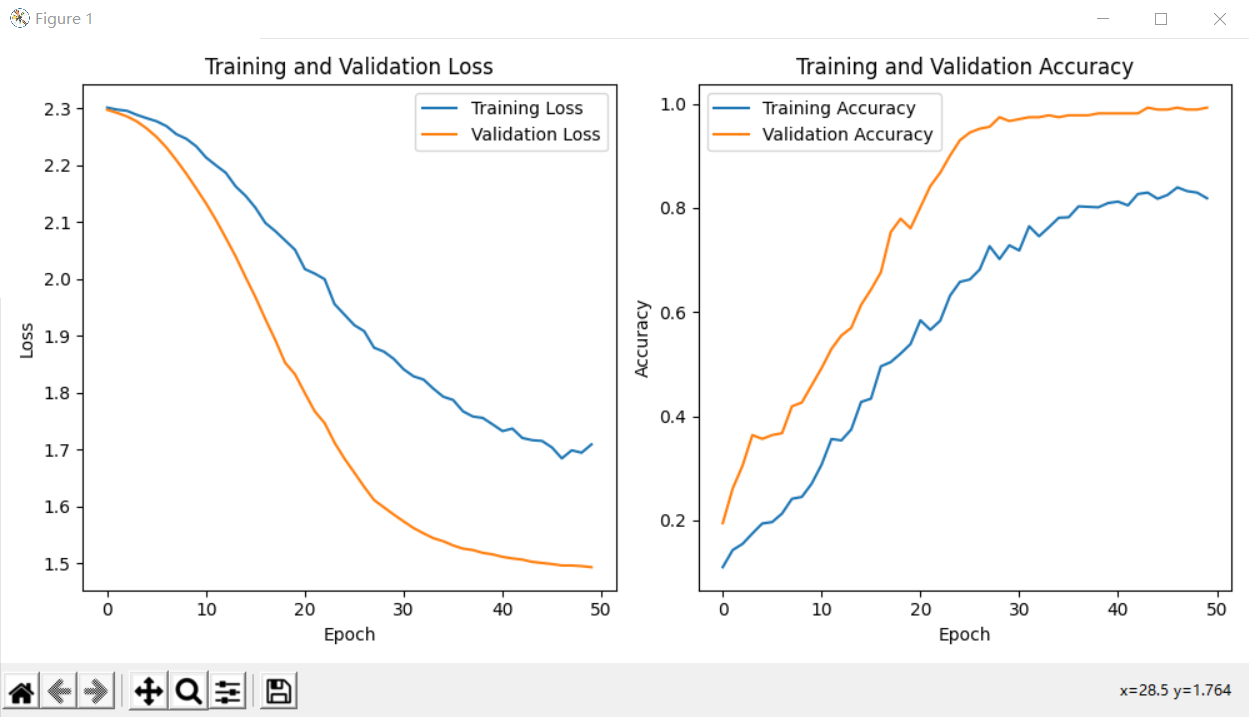
10epoch



30epoch



50epoch



Adam 50epoch

