resnet50-classifyleaves (1)

March 3, 2024

[1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import torch
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader
from PIL import Image
from torchvision import models
from torch import nn
import torch.utils.data as data
import torchvision.transforms as transforms
import os
import csv
from sklearn.preprocessing import LabelEncoder
import time
```

[2]:

```
#
os.chdir("/kaggle/input/classify-leaves")

encoder = LabelEncoder()

class MyDataset(data.Dataset):

    def __init__(self, csv_path, is_test=False):
        self.is_test = is_test
        self.filenames = []
        self.labels = []
        current_directory = os.getcwd()
        with open(csv_path, 'r') as f:
            reader = csv.reader(f)
            for row in reader:
                if row[0] == 'image':
                    continue
                elif is_test:
                    self.filenames.append(os.path.join(current_directory,␣
 ↪row[0]))
                else:
                    self.labels.append(row[1])
```

```python
                    self.filenames.append(os.path.join(current_directory, row[0]))
            #
            print(len(set(self.filenames)))

            #  LabelEncoder
            global encoder
            if not is_test:
                labels_encoded = encoder.fit(self.labels)
                LabelEncoder()
                labels_encoded = encoder.transform(self.labels)
                #   numpy
                self.labels = torch.tensor(labels_encoded)

    def __getitem__(self, index ):
        image = Image.open(self.filenames[index])
        data = self.preprocess(image)
        if self.is_test:
            return data
        else:
            label = self.labels[index]
            return data ,label

    def __len__(self):
        return len(self.filenames)

    def preprocess(self, data):
        transform_train_list = [
            # transforms.Resize((self.opt.h, self.opt.w), interpolation=3),
            # transforms.Pad(self.opt.pad, padding_mode='edge'),
            # transforms.RandomCrop((self.opt.h, self.opt.w)),
            # transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            # transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ]
        return transforms.Compose(transform_train_list)(data)

def togpu(x):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    return x.to(device)

def load_model(net, path):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    net.load_state_dict(torch.load(path, map_location=device))
    return net

def save_model(net, path):
    torch.save(net.state_dict(), path)
    print("saved:", path)
```

[3]:

```
batch_size = 32
lr, num_epochs = 0.005,14

train_dataset = MyDataset("train.csv")
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=batch_size, shuffle=True, num_workers=4)

test_dataset = MyDataset('test.csv', is_test=True)
test_loader = torch.utils.data.DataLoader(
    test_dataset, batch_size=1, shuffle=False, num_workers=4)
```

```
18353
8800
```

[4]:

```
resnet50 = models.resnet50(pretrained=True)
num_ftrs = resnet50.fc.in_features
for param in resnet50.parameters():
    param.requires_grad = True

resnet50.fc = nn.Sequential(nn.Linear(num_ftrs,176),
                            nn.LogSoftmax(dim=1))
```

```
/opt/conda/lib/python3.10/site-packages/torchvision/models/_utils.py:208:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/torchvision/models/_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavior is
equivalent to passing `weights=ResNet50_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to
/root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|      | 97.8M/97.8M [00:00<00:00, 172MB/s]
```

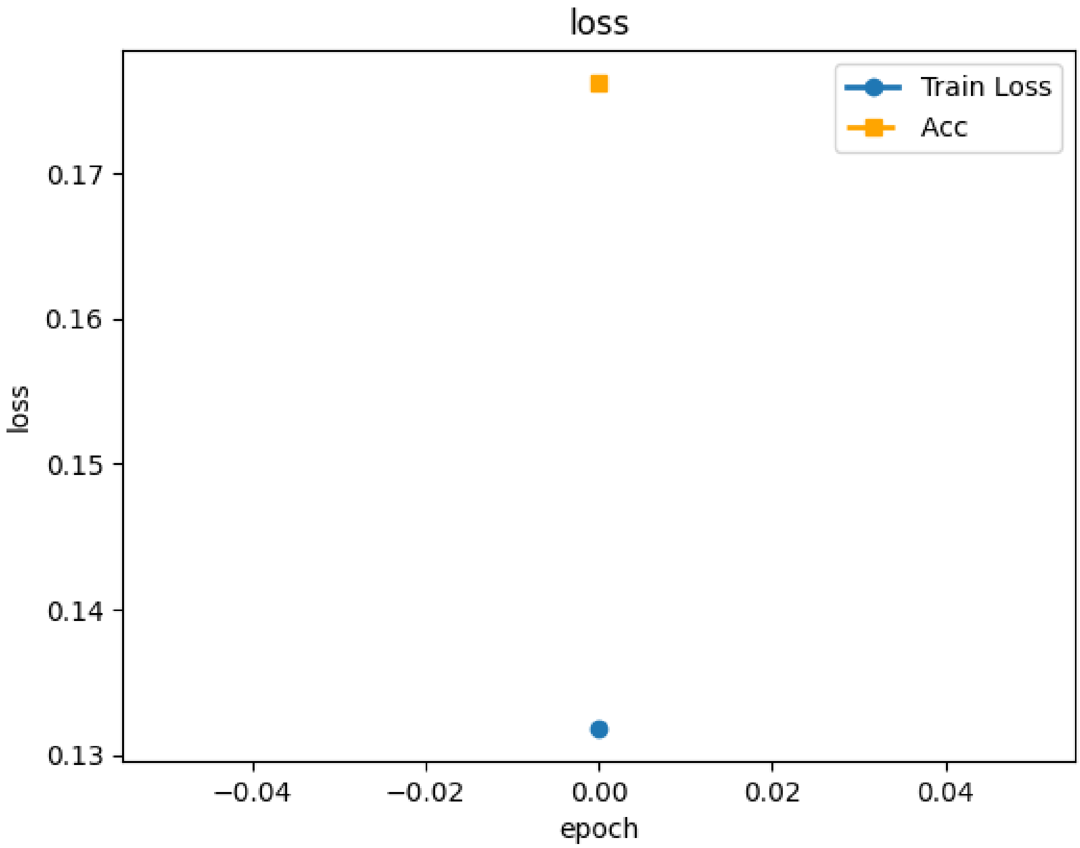
[5]:

```
def train(net):  #

    train_loss = []  #
    acc = []  #
    x = list(range(0, num_epochs))  # epoch array
    #
    plt.ion()

    if torch.cuda.device_count() > 1:
        print("useing", torch.cuda.device_count(), "GPUs!")
        net = nn.DataParallel(net)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    net = net.to(device)

    print("training on", device)  #


    optimizer = torch.optim.SGD(
        net.parameters(), lr=lr
    )  #  SGD
    loss = nn.CrossEntropyLoss()  #
    for epoch in range(num_epochs):  #
        train_l_sum, train_acc_sum, n, start = (
            0.0,
            0.0,
            0,
            time.time(),
        )  #
        for X, y in train_loader:  #
            X, y = togpu(X), togpu(y)  #    GPU
            y_hat = net(X)  #
            l = loss(y_hat, y)  #
            optimizer.zero_grad()  #
            l.backward()  #
            optimizer.step()  #
            train_l_sum += l.cpu().item()  #
            train_acc_sum += (y_hat.argmax(dim=1) ==
                              y).sum().cpu().item()  #
            n += y.shape[0]  #

        #
        train_loss.append(train_l_sum / n)  #
        acc.append(train_acc_sum / n)  #

        ix = x[:epoch +1  ]
        train_iy = train_loss
        valid_iy = acc
        plt.cla()
        plt.title("loss")
        plt.plot(ix, train_iy, label='Train Loss', linewidth=2, linestyle='-',␣
↪marker='o')
        plt.plot(ix, valid_iy, label='Acc ', color="orange", linewidth=2,␣
↪linestyle='--', marker='s')
        plt.xlabel("epoch")
        plt.ylabel("loss")
        plt.legend()
        plt.pause(0.5)

#         save_model(resnet50 ,"/kaggle/working/" + str(epoch + 1) +".pth")
        save_model(resnet50 , "/kaggle/working/re.pth")

        print(
            "epoch %d, loss %.4f, train acc %.3f, time %.1f sec"
            % (epoch + 1, train_l_sum / n, train_acc_sum / n, time.time() -␣
 ↪start)
        )  #

    plt.ioff()
    plt.show()





#

# pretrained_path = "/kaggle/working/103.pth"
# if os.path.exists(pretrained_path):
#     net = load_model(resnet50, pretrained_path)
#     print("Loaded pretrained model")
```

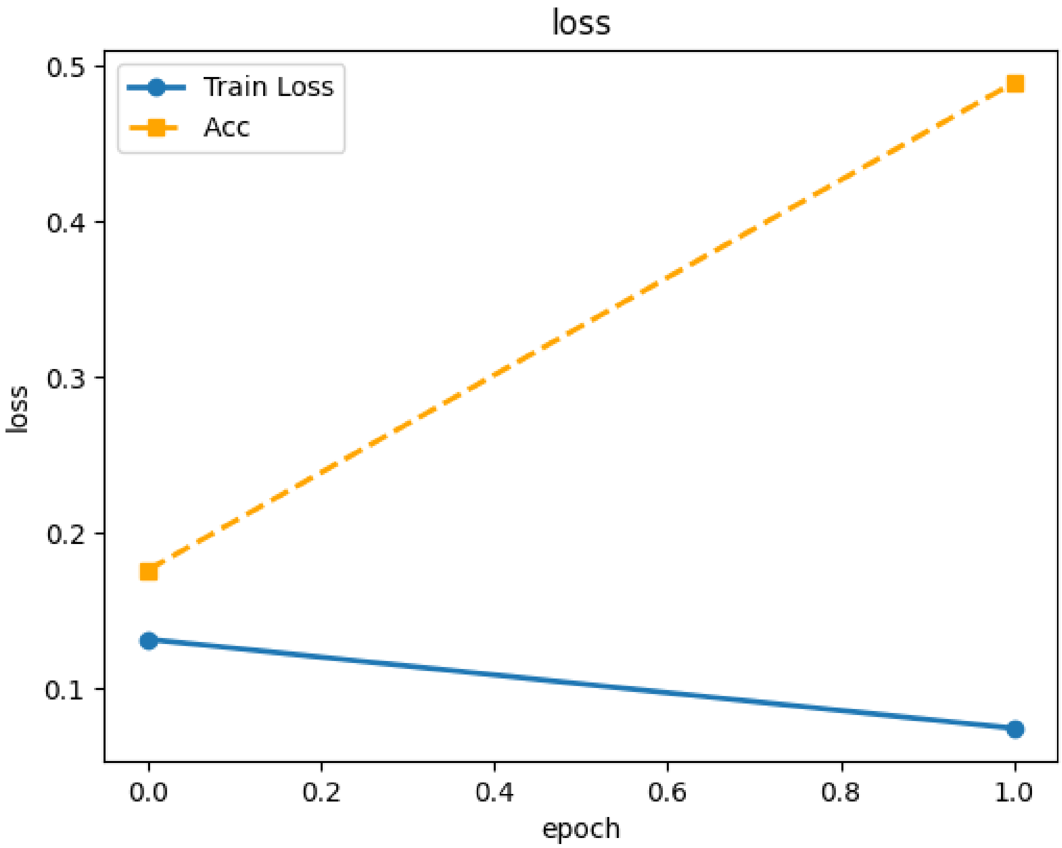
[6]:

```
train(resnet50)

# save_model(resnet50 , "/kaggle/working/")
```

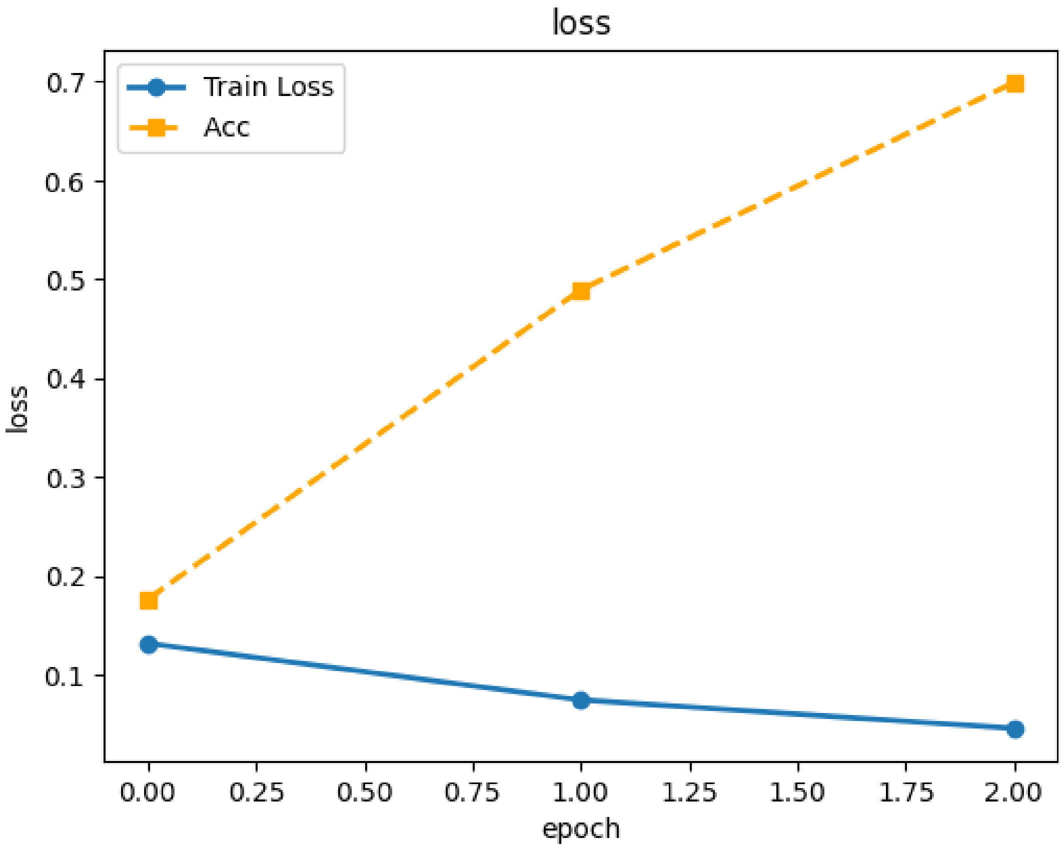
```
training on cuda
```



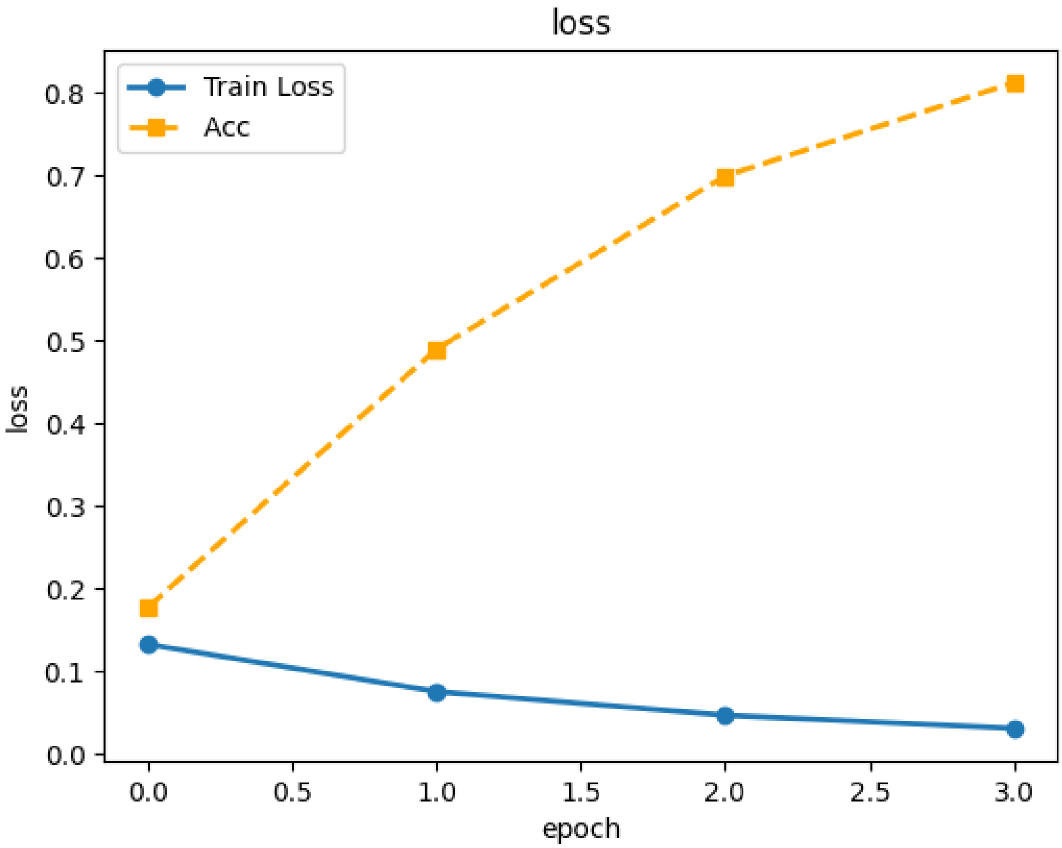
```
saved: /kaggle/working/re.pth
epoch 1, loss 0.1318, train acc 0.176, time 100.9 sec
```



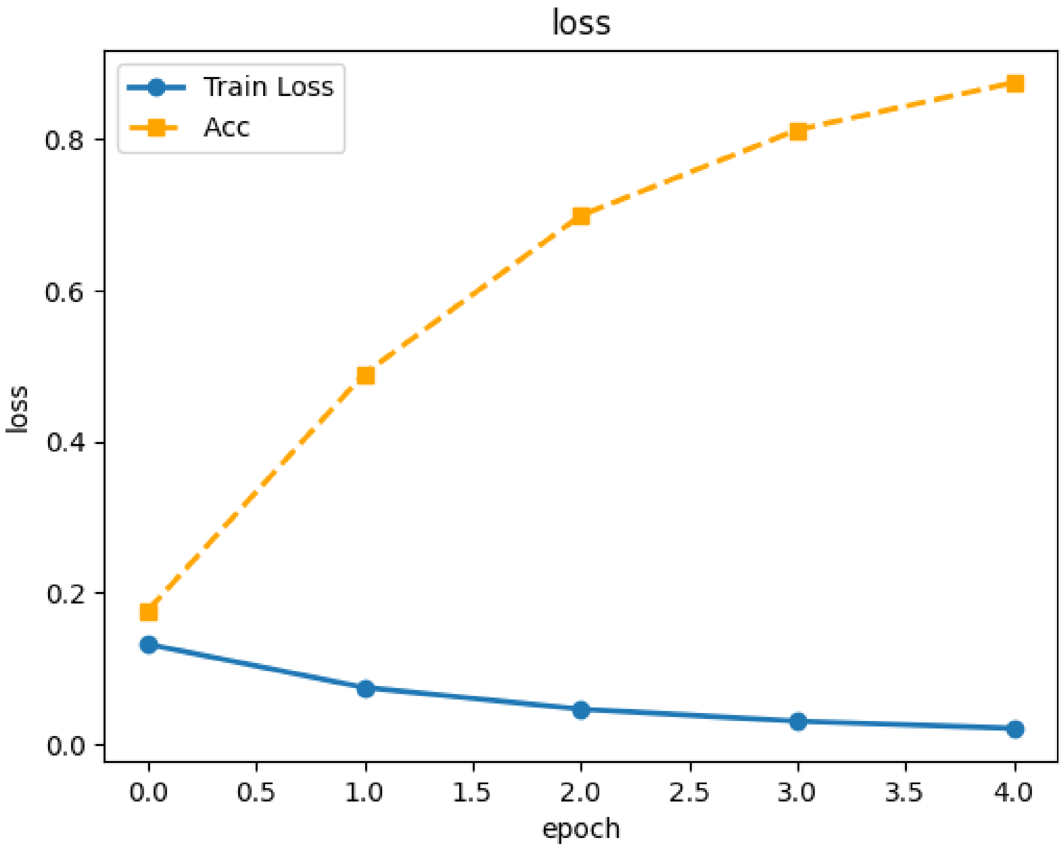
```
saved: /kaggle/working/re.pth
epoch 2, loss 0.0746, train acc 0.490, time 99.8 sec
```



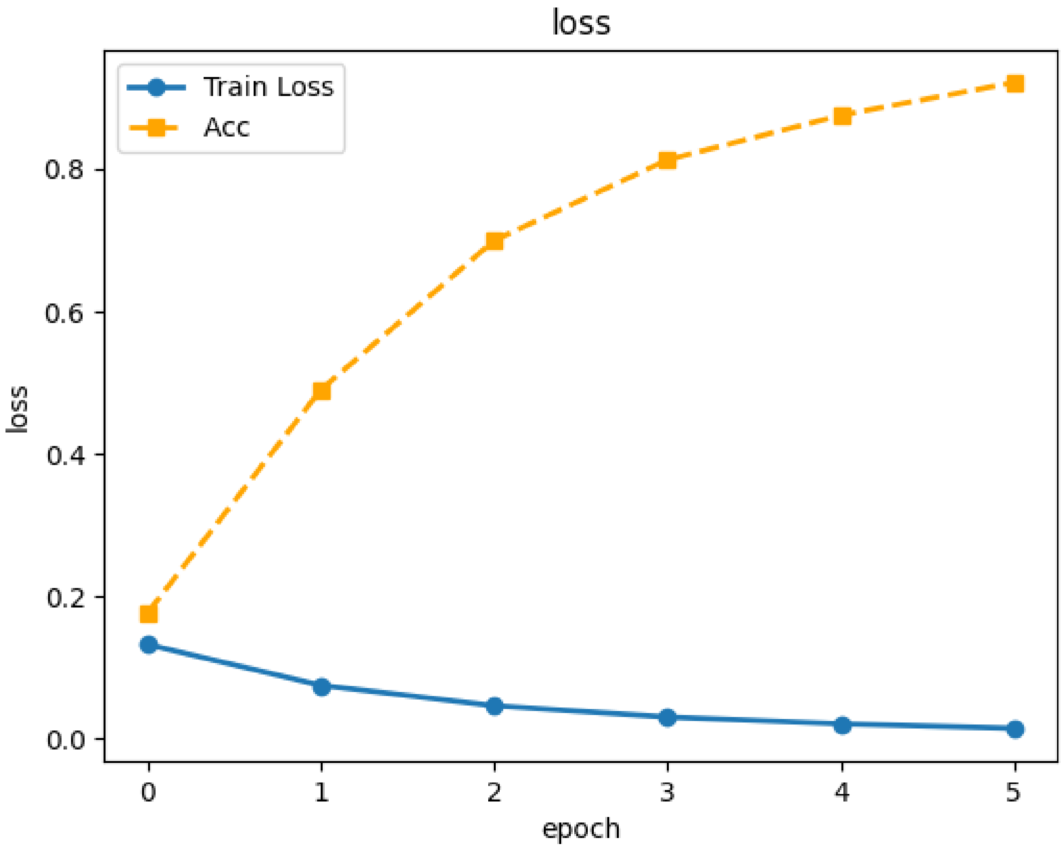
```
saved: /kaggle/working/re.pth
epoch 3, loss 0.0459, train acc 0.699, time 99.8 sec
```



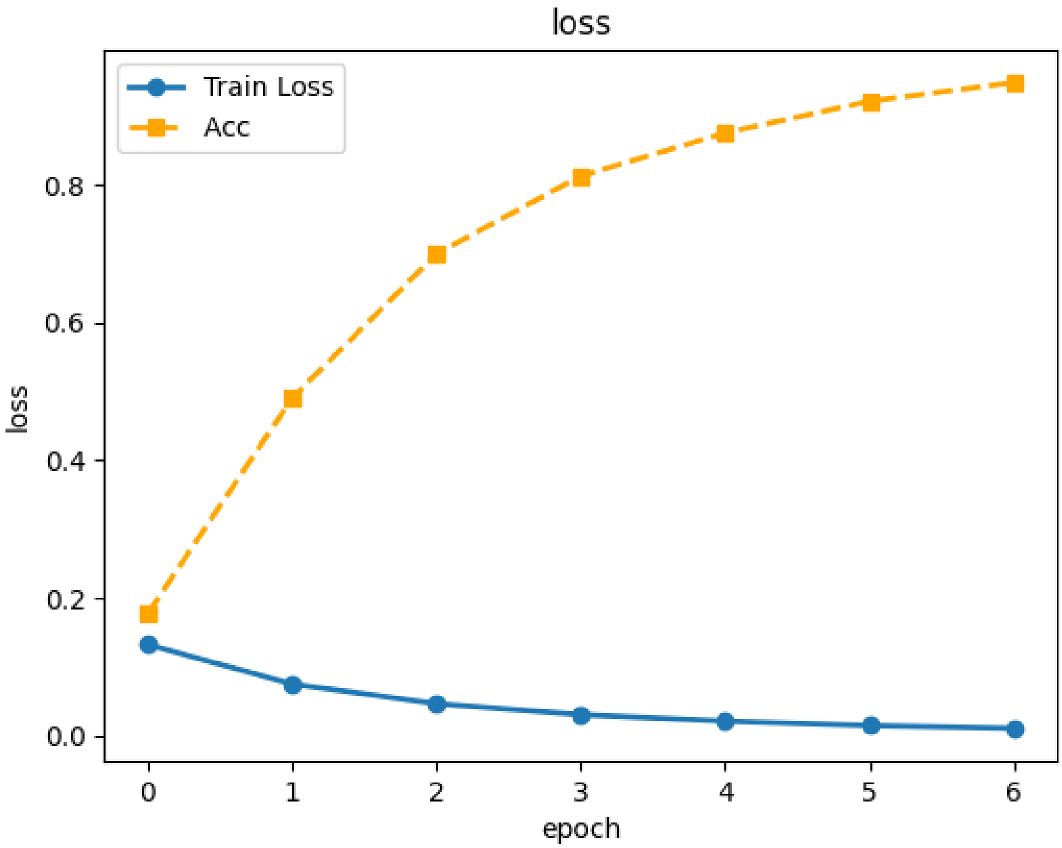
```
saved: /kaggle/working/re.pth
epoch 4, loss 0.0300, train acc 0.812, time 99.7 sec
```



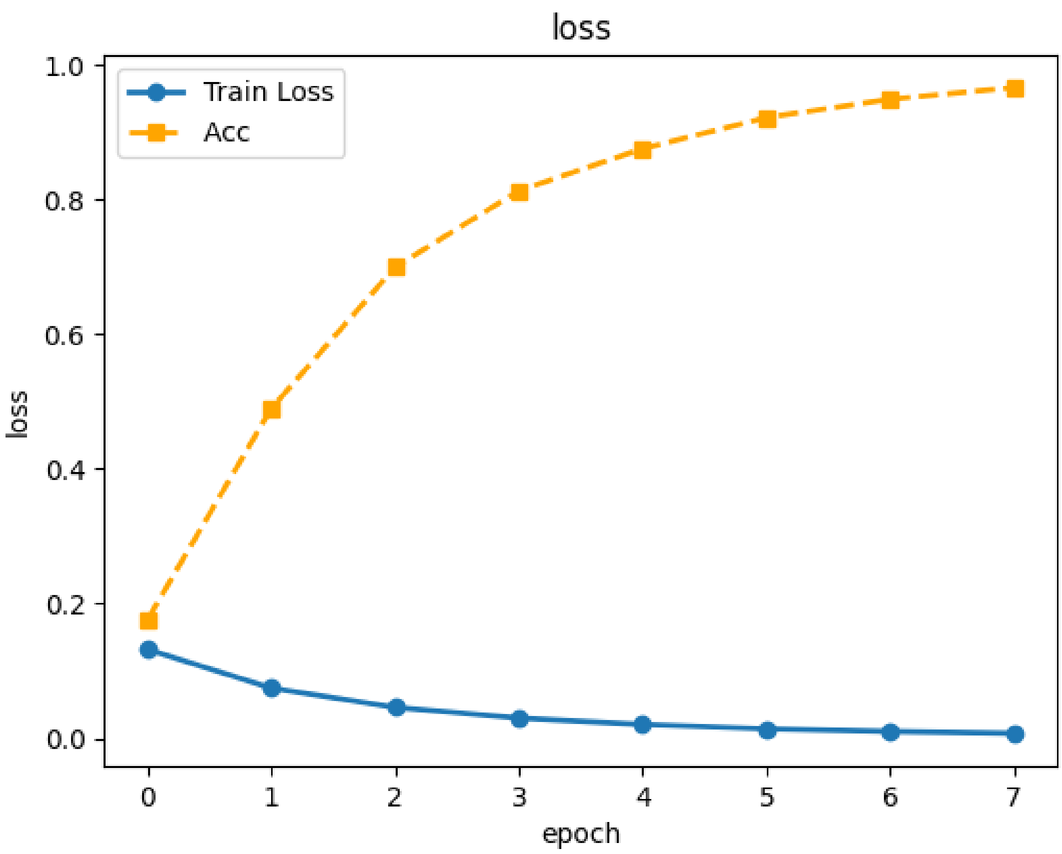
```
saved: /kaggle/working/re.pth
epoch 5, loss 0.0206, train acc 0.875, time 99.9 sec
```



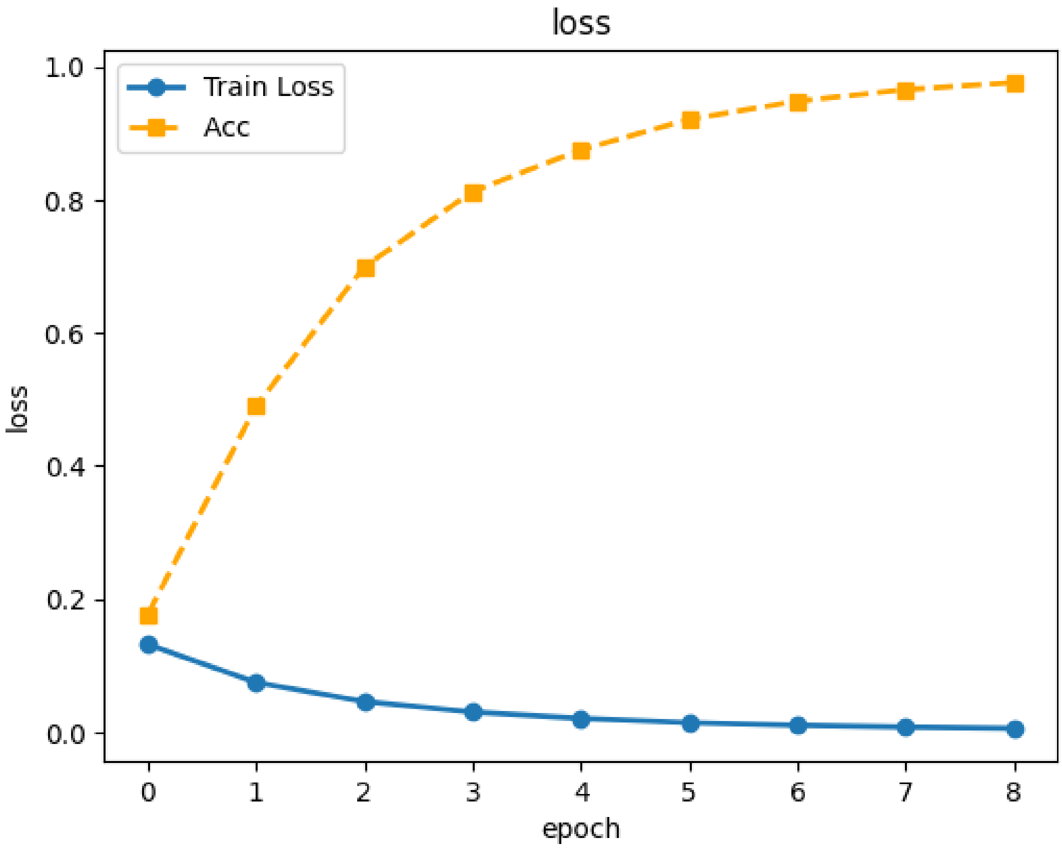
```
saved: /kaggle/working/re.pth
epoch 6, loss 0.0142, train acc 0.921, time 99.8 sec
```



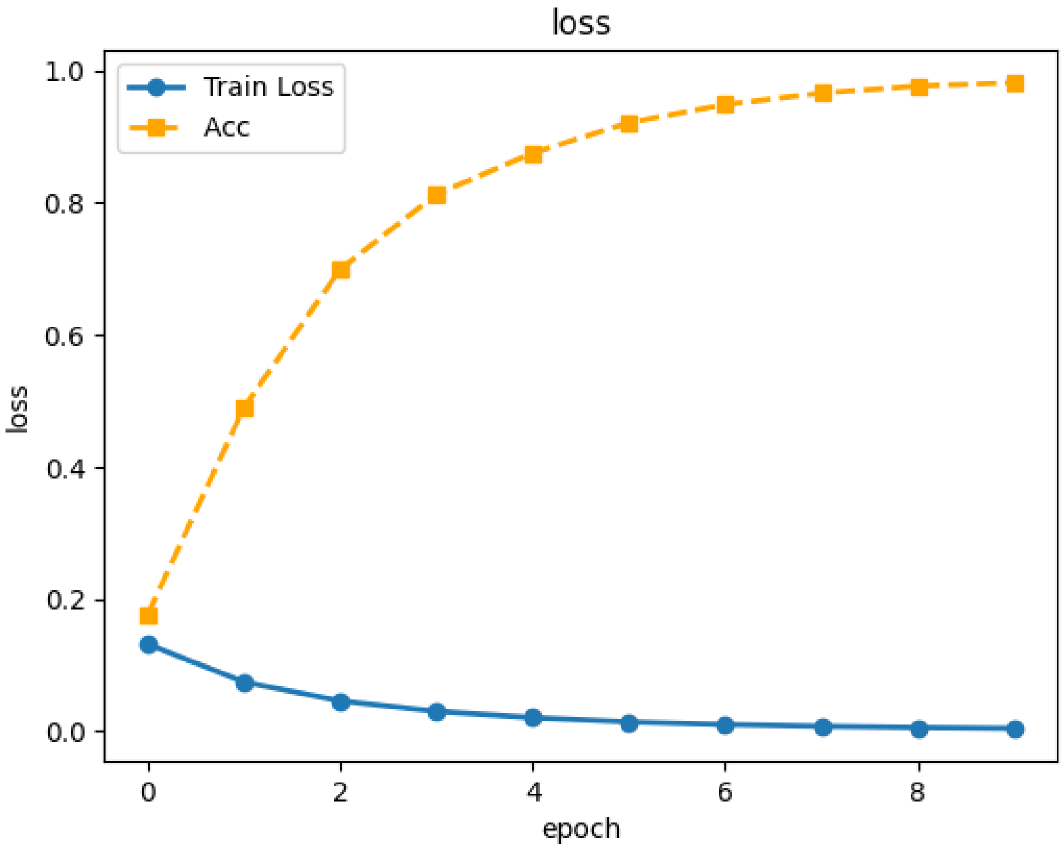
```
saved: /kaggle/working/re.pth
epoch 7, loss 0.0102, train acc 0.948, time 99.8 sec
```



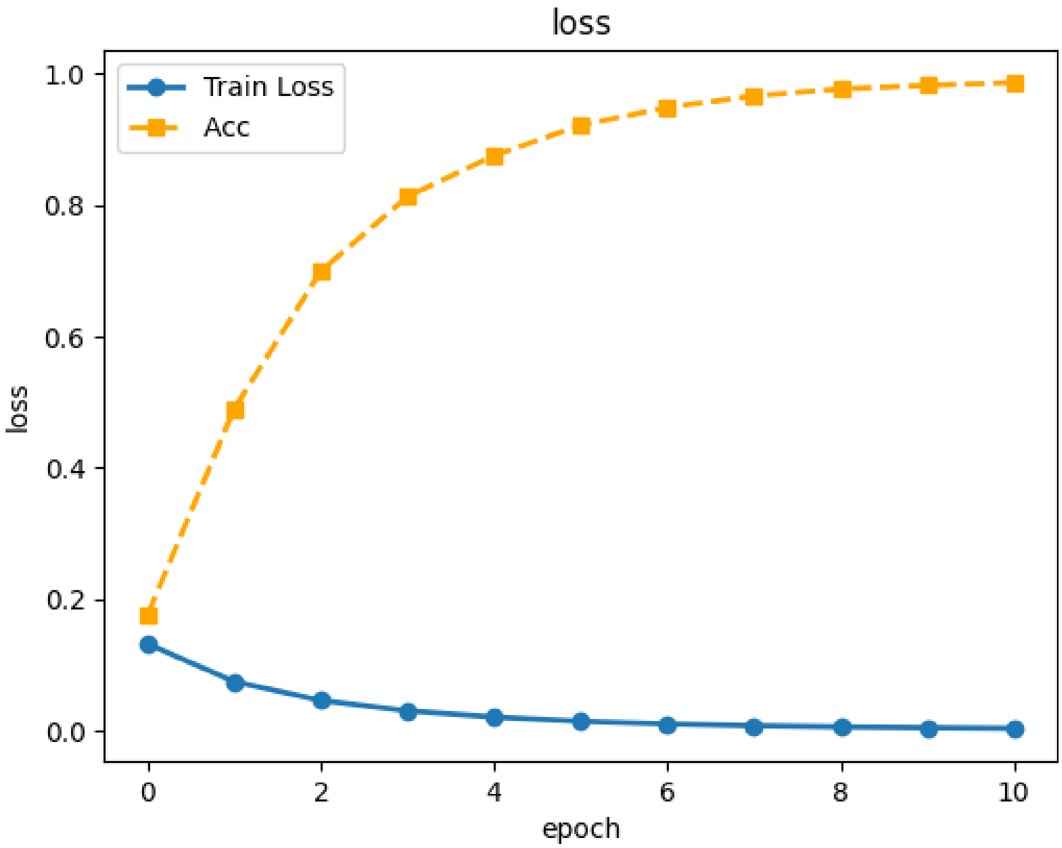
```
saved: /kaggle/working/re.pth
epoch 8, loss 0.0074, train acc 0.966, time 99.7 sec
```



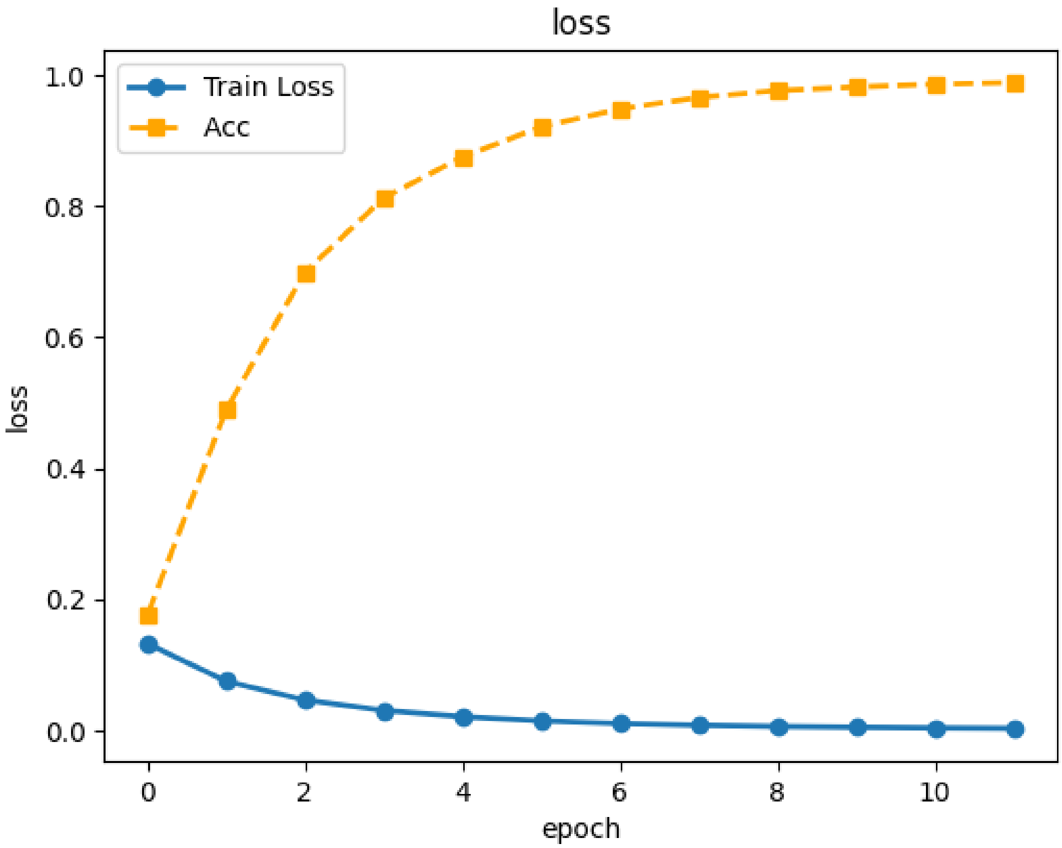
```
saved: /kaggle/working/re.pth
epoch 9, loss 0.0056, train acc 0.976, time 99.8 sec
```



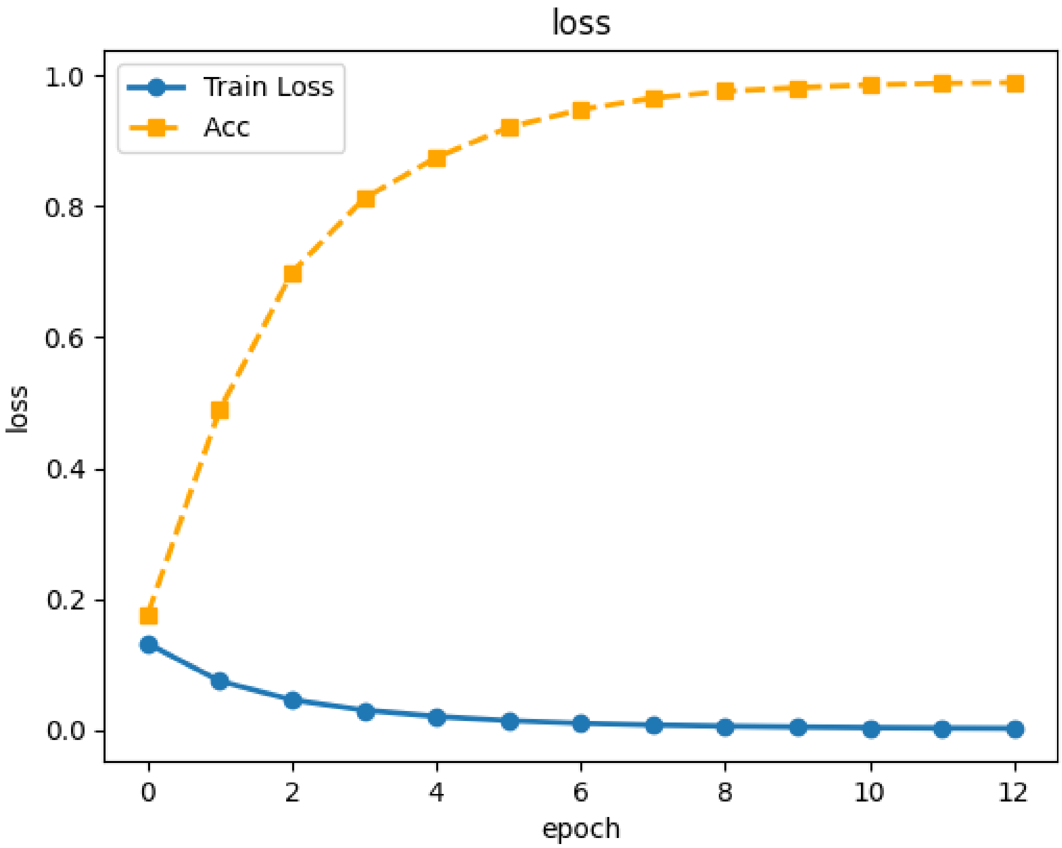
```
saved: /kaggle/working/re.pth
epoch 10, loss 0.0044, train acc 0.982, time 99.7 sec
```



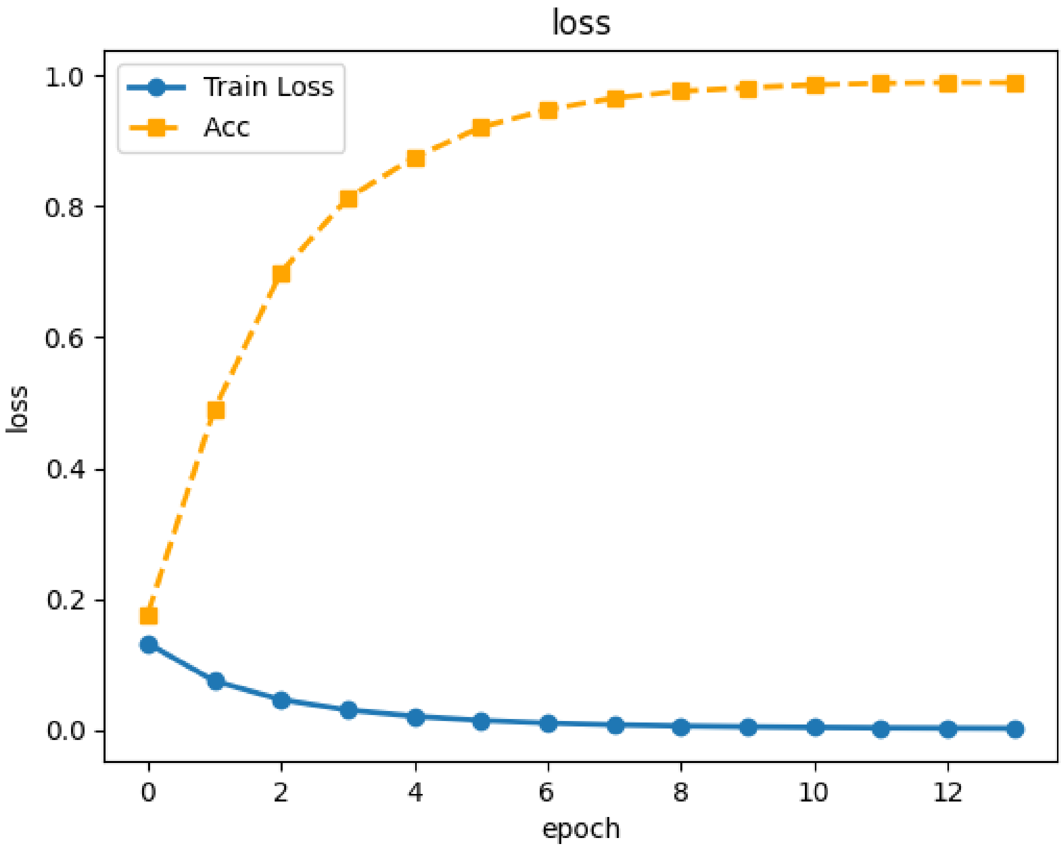
```
saved: /kaggle/working/re.pth
epoch 11, loss 0.0035, train acc 0.986, time 99.7 sec
```



```
saved: /kaggle/working/re.pth
epoch 12, loss 0.0028, train acc 0.988, time 99.7 sec
```



```
saved: /kaggle/working/re.pth
epoch 13, loss 0.0024, train acc 0.989, time 100.0 sec
```



```
saved: /kaggle/working/re.pth
epoch 14, loss 0.0021, train acc 0.989, time 99.8 sec
```

[7]:
```
import pandas as pd
import numpy as np
test_labels = []
i = 0

for i in range(len(test_dataset)):
    test_data = test_dataset.__getitem__(i )
    test_data = test_data.unsqueeze(0)
    test_data = togpu(test_data)
    resnet50 = togpu(resnet50)
    resnet50.eval()
    output = resnet50(test_data)
    predictions = output.argmax(dim=1)
    test_labels.append(predictions.cpu().numpy())

# Convert the list of numpy arrays to a single numpy array
test_labels = np.concatenate(test_labels)
```

```

# Now you can call encoder.inverse_transform
test_labels = encoder.inverse_transform(test_labels)

test_df = pd.read_csv('test.csv')

test_df['label'] = test_labels

test_df.to_csv('/kaggle/working/submission.csv', index=False)
print("################################")
```

```
################################
```