Tanawin-st123975-train-cnn-cifar10-performance-optimizer

October 15, 2023

[ ]:
```python
import torch
import torchvision
import torchvision.transforms as transforms
```

[ ]:
```python
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# Assuming that we are on a CUDA machine, this should print a CUDA device:

print(device)
```

```
cuda:0
```

[ ]:
```python
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Normalization for testing, no data augmentation
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

batch_size = 32

# Load CIFAR-10 dataset
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True,␣
 ↪transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=32)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=32)
```

```
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```

```
truck dog   deer  deer  ship  plane dog   deer  deer  dog   deer  deer  bird
horse deer  frog  deer  plane dog   deer  horse car   dog   plane truck bird
dog   dog   bird  ship  cat   horse
```

[ ]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F


class Cifar10CnnModel(nn.Module):
    def __init__(self):
        super(Cifar10CnnModel, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.classifier = nn.Sequential(
            nn.Linear(256 * 4 * 4, 1024),  # Increase the number of units here
            nn.BatchNorm1d(1024),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(1024, 512),  # Increase the number of units here
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Dropout(0.5),
```

```
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)  # Flatten the tensor
        x = self.classifier(x)
        return x

# Create an instance of the combined model
# combined_model = Cifar10CnnModel()

# Rest of the code remains unchanged
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net = Cifar10CnnModel()
net.to(device)
```

[ ]:
```
Cifar10CnnModel(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU()
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (9): ReLU()
    (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (12): ReLU()
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (16): ReLU()
    (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (19): ReLU()
```

```
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=4096, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=1024, out_features=512, bias=True)
    (5): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (6): ReLU()
    (7): Dropout(p=0.5, inplace=False)
    (8): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

[ ]:
```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9, weight_decay=0.
 ↪0001)
# optimizer = optim.RMSprop(net.parameters(), lr=0.001, alpha=0.9,␣
 ↪weight_decay=0.0001)
# optimizer = optim.Adam(net.parameters(), lr=0.001, weight_decay=0.0001)
```

[ ]:
```
from torch.optim.lr_scheduler import ExponentialLR, MultiStepLR

scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
scheduler1 = ExponentialLR(optimizer, gamma=0.9)
scheduler2 = MultiStepLR(optimizer, milestones=[30,80], gamma=0.1)
```

[ ]:
```
net_name = "cnn_SGD_net"
# Training loop
num_epochs = 50
train_loss_list = []
valid_loss_list = []
train_accuracy_list = []
valid_accuracy_list = []

for epoch in range(num_epochs):
    net.train()
    running_loss = 0.0
    correct = 0
    total = 0
```

```python
    for images, labels in trainloader:
        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        outputs = net(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_loss = running_loss / len(trainloader)
    train_accuracy = correct / total

    # Validation loop
    net.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in testloader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = net(images)
            loss = criterion(outputs, labels)

            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    valid_loss = running_loss / len(testloader)
    valid_accuracy = correct / total

    # Store loss and accuracy values
    train_loss_list.append(train_loss)
    valid_loss_list.append(valid_loss)
    train_accuracy_list.append(train_accuracy)
    valid_accuracy_list.append(valid_accuracy)
```

```python
    # Print the training/validation statistics
    print(f"Epoch: {epoch+1}/{num_epochs} | "
          f"Train Loss: {train_loss:.4f} | Train Acc: {train_accuracy:.4f} | "
          f"Valid Loss: {valid_loss:.4f} | Valid Acc: {valid_accuracy:.4f}")

    # Update the learning rate
    # scheduler.step()
    scheduler1.step()
    scheduler2.step()

# Save the trained net
torch.save(net.state_dict(), "cnn_SGD_net.pth")
```

```
Epoch: 1/50 | Train Loss: 1.5135 | Train Acc: 0.4486 | Valid Loss: 1.1275 |
Valid Acc: 0.5924
Epoch: 2/50 | Train Loss: 1.0768 | Train Acc: 0.6169 | Valid Loss: 0.8185 |
Valid Acc: 0.7091
Epoch: 3/50 | Train Loss: 0.9094 | Train Acc: 0.6780 | Valid Loss: 0.7196 |
Valid Acc: 0.7441
Epoch: 4/50 | Train Loss: 0.8063 | Train Acc: 0.7192 | Valid Loss: 0.6550 |
Valid Acc: 0.7663
Epoch: 5/50 | Train Loss: 0.7386 | Train Acc: 0.7444 | Valid Loss: 0.5990 |
Valid Acc: 0.7923
Epoch: 6/50 | Train Loss: 0.6829 | Train Acc: 0.7620 | Valid Loss: 0.5786 |
Valid Acc: 0.7971
Epoch: 7/50 | Train Loss: 0.6393 | Train Acc: 0.7797 | Valid Loss: 0.5403 |
Valid Acc: 0.8096
Epoch: 8/50 | Train Loss: 0.6000 | Train Acc: 0.7934 | Valid Loss: 0.5110 |
Valid Acc: 0.8217
Epoch: 9/50 | Train Loss: 0.5704 | Train Acc: 0.8021 | Valid Loss: 0.4860 |
Valid Acc: 0.8323
Epoch: 10/50 | Train Loss: 0.5452 | Train Acc: 0.8125 | Valid Loss: 0.4746 |
Valid Acc: 0.8349
Epoch: 11/50 | Train Loss: 0.5208 | Train Acc: 0.8201 | Valid Loss: 0.4454 |
Valid Acc: 0.8439
Epoch: 12/50 | Train Loss: 0.5005 | Train Acc: 0.8279 | Valid Loss: 0.4363 |
Valid Acc: 0.8489
Epoch: 13/50 | Train Loss: 0.4822 | Train Acc: 0.8340 | Valid Loss: 0.4217 |
Valid Acc: 0.8537
Epoch: 14/50 | Train Loss: 0.4583 | Train Acc: 0.8424 | Valid Loss: 0.4180 |
Valid Acc: 0.8553
Epoch: 15/50 | Train Loss: 0.4456 | Train Acc: 0.8477 | Valid Loss: 0.4078 |
Valid Acc: 0.8576
Epoch: 16/50 | Train Loss: 0.4300 | Train Acc: 0.8514 | Valid Loss: 0.4015 |
Valid Acc: 0.8662
Epoch: 17/50 | Train Loss: 0.4116 | Train Acc: 0.8590 | Valid Loss: 0.3906 |
Valid Acc: 0.8668
```

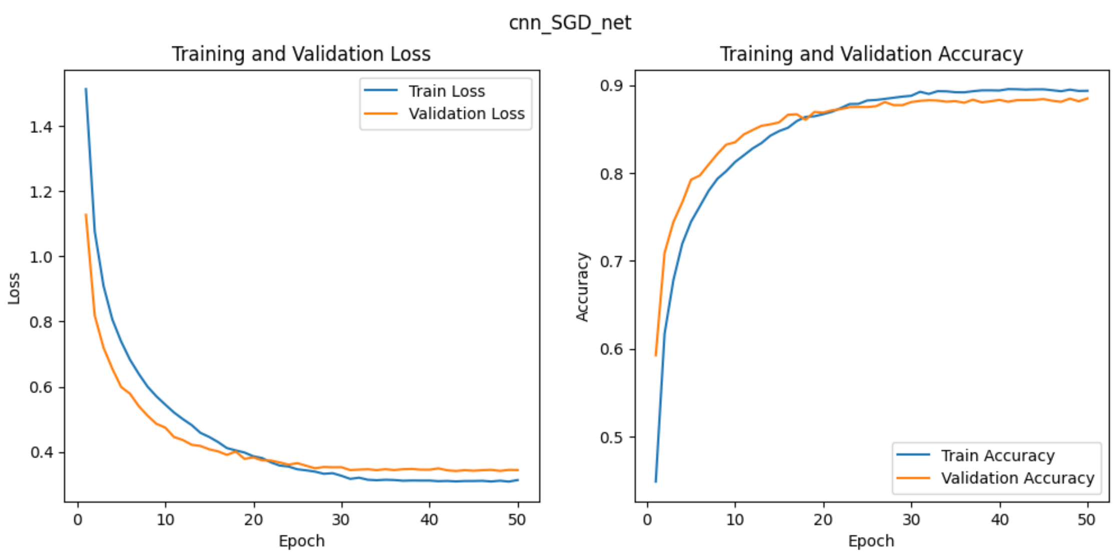
```
Epoch: 18/50 | Train Loss: 0.4045 | Train Acc: 0.8638 | Valid Loss: 0.4009 |
Valid Acc: 0.8605
Epoch: 19/50 | Train Loss: 0.3978 | Train Acc: 0.8645 | Valid Loss: 0.3784 |
Valid Acc: 0.8694
Epoch: 20/50 | Train Loss: 0.3864 | Train Acc: 0.8669 | Valid Loss: 0.3830 |
Valid Acc: 0.8687
Epoch: 21/50 | Train Loss: 0.3808 | Train Acc: 0.8697 | Valid Loss: 0.3739 |
Valid Acc: 0.8713
Epoch: 22/50 | Train Loss: 0.3677 | Train Acc: 0.8739 | Valid Loss: 0.3734 |
Valid Acc: 0.8727
Epoch: 23/50 | Train Loss: 0.3578 | Train Acc: 0.8784 | Valid Loss: 0.3675 |
Valid Acc: 0.8749
Epoch: 24/50 | Train Loss: 0.3548 | Train Acc: 0.8786 | Valid Loss: 0.3607 |
Valid Acc: 0.8751
Epoch: 25/50 | Train Loss: 0.3464 | Train Acc: 0.8825 | Valid Loss: 0.3652 |
Valid Acc: 0.8750
Epoch: 26/50 | Train Loss: 0.3432 | Train Acc: 0.8831 | Valid Loss: 0.3574 |
Valid Acc: 0.8760
Epoch: 27/50 | Train Loss: 0.3393 | Train Acc: 0.8843 | Valid Loss: 0.3496 |
Valid Acc: 0.8807
Epoch: 28/50 | Train Loss: 0.3326 | Train Acc: 0.8856 | Valid Loss: 0.3535 |
Valid Acc: 0.8772
Epoch: 29/50 | Train Loss: 0.3345 | Train Acc: 0.8869 | Valid Loss: 0.3525 |
Valid Acc: 0.8772
Epoch: 30/50 | Train Loss: 0.3265 | Train Acc: 0.8878 | Valid Loss: 0.3528 |
Valid Acc: 0.8807
Epoch: 31/50 | Train Loss: 0.3170 | Train Acc: 0.8924 | Valid Loss: 0.3441 |
Valid Acc: 0.8821
Epoch: 32/50 | Train Loss: 0.3206 | Train Acc: 0.8899 | Valid Loss: 0.3455 |
Valid Acc: 0.8828
Epoch: 33/50 | Train Loss: 0.3143 | Train Acc: 0.8931 | Valid Loss: 0.3466 |
Valid Acc: 0.8824
Epoch: 34/50 | Train Loss: 0.3128 | Train Acc: 0.8929 | Valid Loss: 0.3439 |
Valid Acc: 0.8811
Epoch: 35/50 | Train Loss: 0.3143 | Train Acc: 0.8919 | Valid Loss: 0.3467 |
Valid Acc: 0.8818
Epoch: 36/50 | Train Loss: 0.3135 | Train Acc: 0.8918 | Valid Loss: 0.3440 |
Valid Acc: 0.8799
Epoch: 37/50 | Train Loss: 0.3113 | Train Acc: 0.8930 | Valid Loss: 0.3467 |
Valid Acc: 0.8834
Epoch: 38/50 | Train Loss: 0.3120 | Train Acc: 0.8940 | Valid Loss: 0.3475 |
Valid Acc: 0.8804
Epoch: 39/50 | Train Loss: 0.3118 | Train Acc: 0.8941 | Valid Loss: 0.3453 |
Valid Acc: 0.8816
Epoch: 40/50 | Train Loss: 0.3118 | Train Acc: 0.8938 | Valid Loss: 0.3452 |
Valid Acc: 0.8832
Epoch: 41/50 | Train Loss: 0.3098 | Train Acc: 0.8955 | Valid Loss: 0.3492 |
Valid Acc: 0.8811
```

```
Epoch: 42/50 | Train Loss: 0.3106 | Train Acc: 0.8953 | Valid Loss: 0.3434 |
Valid Acc: 0.8829
Epoch: 43/50 | Train Loss: 0.3092 | Train Acc: 0.8948 | Valid Loss: 0.3412 |
Valid Acc: 0.8830
Epoch: 44/50 | Train Loss: 0.3104 | Train Acc: 0.8952 | Valid Loss: 0.3438 |
Valid Acc: 0.8832
Epoch: 45/50 | Train Loss: 0.3105 | Train Acc: 0.8952 | Valid Loss: 0.3421 |
Valid Acc: 0.8841
Epoch: 46/50 | Train Loss: 0.3110 | Train Acc: 0.8942 | Valid Loss: 0.3437 |
Valid Acc: 0.8822
Epoch: 47/50 | Train Loss: 0.3089 | Train Acc: 0.8930 | Valid Loss: 0.3448 |
Valid Acc: 0.8810
Epoch: 48/50 | Train Loss: 0.3114 | Train Acc: 0.8948 | Valid Loss: 0.3415 |
Valid Acc: 0.8846
Epoch: 49/50 | Train Loss: 0.3086 | Train Acc: 0.8933 | Valid Loss: 0.3447 |
Valid Acc: 0.8816
Epoch: 50/50 | Train Loss: 0.3135 | Train Acc: 0.8935 | Valid Loss: 0.3441 |
Valid Acc: 0.8848
```

[ ]:
```python
true_labels = []
predicted_labels = []

# Validation loop
net.eval()
with torch.no_grad():
    for images, labels in testloader:
        images = images.to(device)
        labels = labels.to(device)

        outputs = net(images)
        loss = criterion(outputs, labels)

        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        # Append true labels and predicted labels
        true_labels.extend(labels.cpu().numpy())
        predicted_labels.extend(predicted.cpu().numpy())

valid_loss = running_loss / len(testloader)
valid_accuracy = correct / total
```

[ ]:
```python
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
```

```
# Plot training and validation loss
plt.plot(range(1, num_epochs+1), train_loss_list, label='Train Loss')
plt.plot(range(1, num_epochs+1), valid_loss_list, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.subplot(1, 2, 2)
# Plot training and validation accuracy
plt.plot(range(1, num_epochs+1), train_accuracy_list, label='Train Accuracy')
plt.plot(range(1, num_epochs+1), valid_accuracy_list, label='Validation␣
  ↪Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.suptitle(net_name)
plt.show()
```



[ ]:
```
from sklearn.metrics import classification_report

# Generate classification report
classification_rep = classification_report(true_labels, predicted_labels)
print("Classification Report:")
print(classification_rep)
```

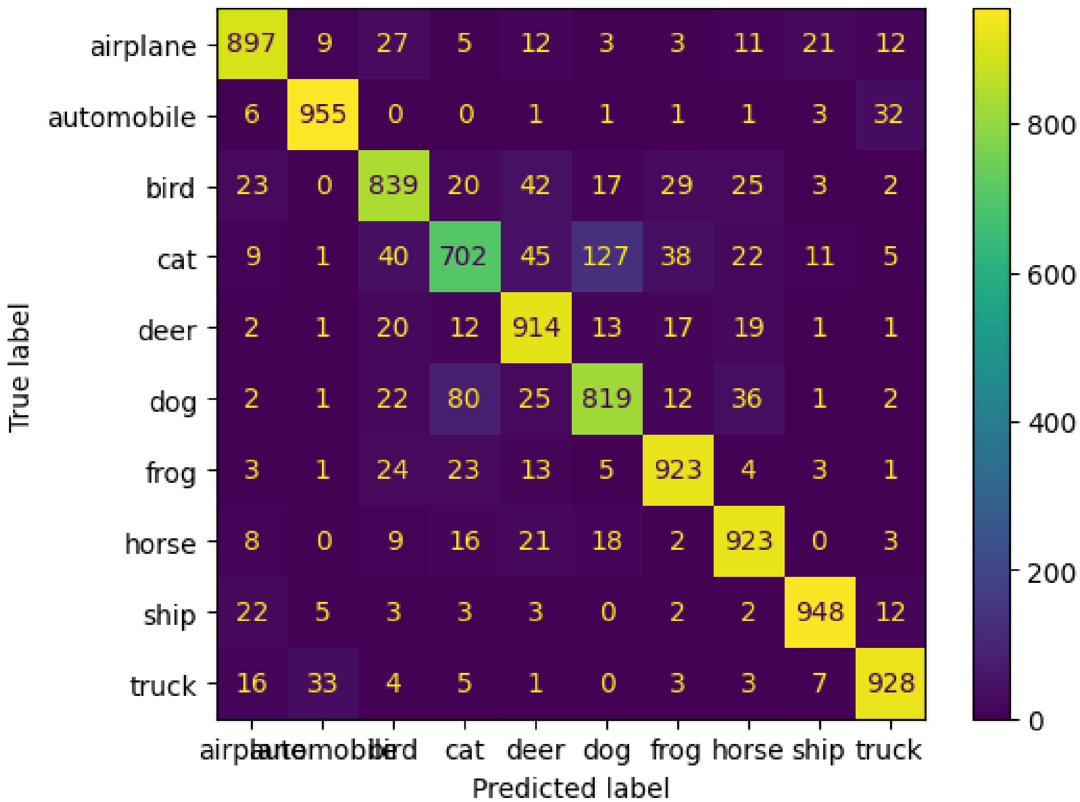
```
Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.91      0.90      0.90      1000
           1       0.95      0.95      0.95      1000
           2       0.85      0.84      0.84      1000
           3       0.81      0.70      0.75      1000
           4       0.85      0.91      0.88      1000
           5       0.82      0.82      0.82      1000
           6       0.90      0.92      0.91      1000
           7       0.88      0.92      0.90      1000
           8       0.95      0.95      0.95      1000
           9       0.93      0.93      0.93      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

[ ]:
```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))

# Generate confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=testset.
 ↪classes)
cm_display.plot()
```

[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fe37d31bc10>
```

```
<Figure size 800x500 with 0 Axes>
```


True label
Predicted label
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
897 9 27 5 12 3 3 11 21 12
6 955 0 0 1 1 1 1 3 32
23 0 839 20 42 17 29 25 3 2
9 1 40 702 45 127 38 22 11 5
2 1 20 12 914 13 17 19 1 1
2 1 22 80 25 819 12 36 1 2
3 1 24 23 13 5 923 4 3 1
8 0 9 16 21 18 2 923 0 3
22 5 3 3 3 0 2 2 948 12
16 33 4 5 1 0 3 3 7 928
800
600
400
200
0