Image classification CIFAR10 CNN

```
[ ] 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
```

HARDWARE REQUIREMENT: cuda machine

```
[ ] transform = transforms.Compose( # transform is from torchvision (only for image)
        [transforms.ToTensor(), # image to tensor --> divide by 255
         transforms.Resize((32, 32))])

    batch_size = 32
```

transforms.Compose: converting images to tensors and resizing them to a 32x32 resolution

```
trainvalset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainset, valset = torch.utils.data.random_split(trainvalset, [40000, 10000])

trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True)
valloader = torch.utils.data.DataLoader(valset, batch_size=batch_size, shuffle=False)

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

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

torchvision.datasets: a module provided by PyTorch's 'torchvision'

CIFAR10 and CIFAR100: Small image classification datasets with 10 and 100 classes, respectively

INPUT:

trainvalset: loading the dataset from CIFAR10 with specific 'transform' and then split into **trainset** and **valset** with 40,000 and 10,000 samples respectively

trainloader and **valloader** are data loaders for the training and validation sets.

testset and **testloader** are similar to 'trainvalset', but for the test set.

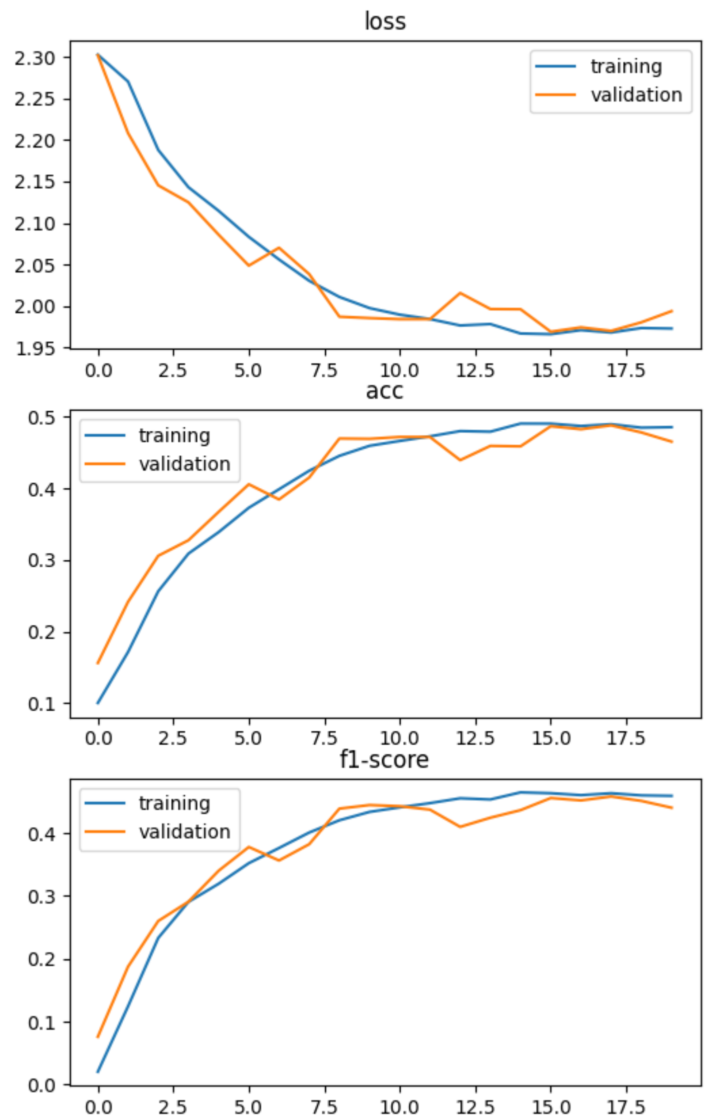
OUTPUT: classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

DATA STATISTICS:

```
trainset.__len__(), valset.__len__(), testset.__len__()
```

```
(40000, 10000, 10000)
```

LEARNING CURVE:



Loss plot: shows the training and validation over epochs. The training loss decreased and the validation loss follow suit without overfitting.

Accuracy plot: A training accuracy rose as well as the validation accuracy.

METRICS:

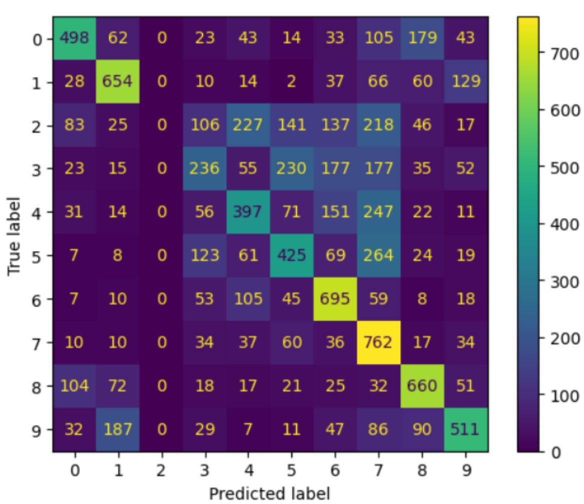
```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
```

DEMO THE RESULT:

```
print('testing ...')
y_predict = list()
y_labels = list()
test_loss = 0.0
n = 0
with torch.no_grad():
    for data in tqdm(testloader):
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

        y_labels += list(labels.cpu().numpy())
        y_predict += list(outputs.argmax(dim=1).cpu().numpy())
        n+=1

    # print statistics
    test_loss /= n
    print(f"testing loss: {test_loss:.4}" )

    report = classification_report(y_labels, y_predict, digits = 4)
    M = confusion_matrix(y_labels, y_predict)
    print(report)
    disp = ConfusionMatrixDisplay(confusion_matrix=M)
```

```
testing ...
100% 313/313 [00:02<00:00, 112.35it/s]
testing loss: 1.973
              precision    recall  f1-score   support

           0     0.6051    0.4980    0.5464      1000
           1     0.6187    0.6540    0.6359      1000
           2     0.0000    0.0000    0.0000      1000
           3     0.3430    0.2360    0.2796      1000
           4     0.4123    0.3970    0.4045      1000
           5     0.4167    0.4250    0.4208      1000
           6     0.4940    0.6950    0.5775      1000
           7     0.3780    0.7620    0.5053      1000
           8     0.5784    0.6600    0.6165      1000
           9     0.5774    0.5110    0.5422      1000

    accuracy                         0.4838     10000
   macro avg     0.4424    0.4838    0.4529     10000
weighted avg     0.4424    0.4838    0.4529     10000
```

```
disp.plot()
plt.show()
```



FINETUNING TECHNIQUE:

```python
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=1e-2, momentum=0.9)
```

Used CrossEntropyLoss as loss function and Stochastic Gradient Descent as optimizer with learning rate 0.01.

```python
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from tqdm.notebook import tqdm

epochs = 20

history_train = {'loss':np.zeros(epochs), 'acc':np.zeros(epochs), 'f1-score':np.zeros(epochs)}
history_val = {'loss':np.zeros(epochs), 'acc':np.zeros(epochs), 'f1-score':np.zeros(epochs)}
min_val_loss = 1e10
PATH = './CNN_CIFAR10.pth'

for epoch in range(epochs):  # loop over the dataset multiple times

    print(f'epoch {epoch + 1} \nTraining ...')
    y_predict = list()
    y_labels = list()
    training_loss = 0.0
    n = 0
    net.train()
    for data in tqdm(trainloader):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs) # forward
        loss = criterion(outputs, labels) # calculate loss from forward pass
        loss.backward() # just calculate
        optimizer.step() # update weights here

        # aggregate statistics
        training_loss += loss.item()
        n+=1

        y_labels += list(labels.cpu().numpy())
        y_predict += list(outputs.argmax(dim=1).cpu().numpy())

    # print statistics
    report = classification_report(y_labels, y_predict, digits = 4, output_dict = True)
    acc = report["accuracy"]
    f1 = report["weighted avg"]["f1-score"]
    support = report["weighted avg"]["support"]
    training_loss /= n
    print(f"training loss: {training_loss:.4}, acc: {acc*100:.4}%, f1-score: {f1*100:.4}%, support: {support}" )
    history_train['loss'][epoch] = training_loss
    history_train['acc'][epoch] = acc
    history_train['f1-score'][epoch] = f1

    print('validating ...')
    net.eval()
```

```python
y_predict = list()
y_labels = list()
validation_loss = 0.0
n = 0
with torch.no_grad():
    for data in tqdm(valloader):
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        validation_loss += loss.item()

        y_labels += list(labels.cpu().numpy())
        y_predict += list(outputs.argmax(dim=1).cpu().numpy())
        n+=1

# print statistics
report = classification_report(y_labels, y_predict, digits = 4, output_dict = True)
acc = report["accuracy"]
f1 = report["weighted avg"]["f1-score"]
support = report["weighted avg"]["support"]
validation_loss /= n
print(f"validation loss: {validation_loss:.4}, acc: {acc*100:.4}%, f1-score: {f1*100:.4}%, support: {support}" )
history_val['loss'][epoch] = validation_loss
history_val['acc'][epoch] = acc
history_val['f1-score'][epoch] = f1
```

The code is monitoring both training and validation performance over epochs.

If the validation loss is the lowest observed so far, save the model's state dictionary to a file.

Key features

Transformations

- **Input:** data (image) from CIFAR10
- **Output:** transformed and resized images

Training Loop

- **Input:** Training dataset
- **Output:** Trained model, training statistics

Validation Loop

- **Input:** Validation dataset
- **Output:** Validation statistics

Testing Loop

- **Input:** Test dataset
- **Output:** Testing loss, classification report, confusion matrix.

Torch.utils.data.DataLoader

- **Input:** dataset, batch_size, shuffle, num_workers
- **Output:** an iterable over the dataset

Image classification with EfficientNetV2s

INPUT:

Download dataset

```
!wget https://github.com/pvateekul/2110531_DSDE_2023s1/raw/main/code/Week05_Intro_Deep_Learning/data/Dataset_animal2.zip
```

OUTPUT:

```
class AnimalDataset(Dataset):

    def __init__(self,
                 img_dir,
                 transforms=None):

        super().__init__()
        label_image = ['butterfly','cat','chicken','cow','dog','elephant','horse','sheep','spider','squirrel']
        self.input_dataset = list()
        label_num = 0
        for label in label_image:
            _, _, files = next(os.walk(os.path.join(img_dir,label)))
            for image_name in files:
                input = [os.path.join(img_dir,label,image_name),label_num] # [image_path, label_num]
                self.input_dataset.append(input)
            label_num += 1

        self.transforms = transforms

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

    def __getitem__(self, idx):
        img = Image.open(self.input_dataset[idx][0]).convert('RGB')
        x = self.transforms(img)
        y = self.input_dataset[idx][1]
        return x,y
```

HARDWARE REQUIREMENTS:

```
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)
```

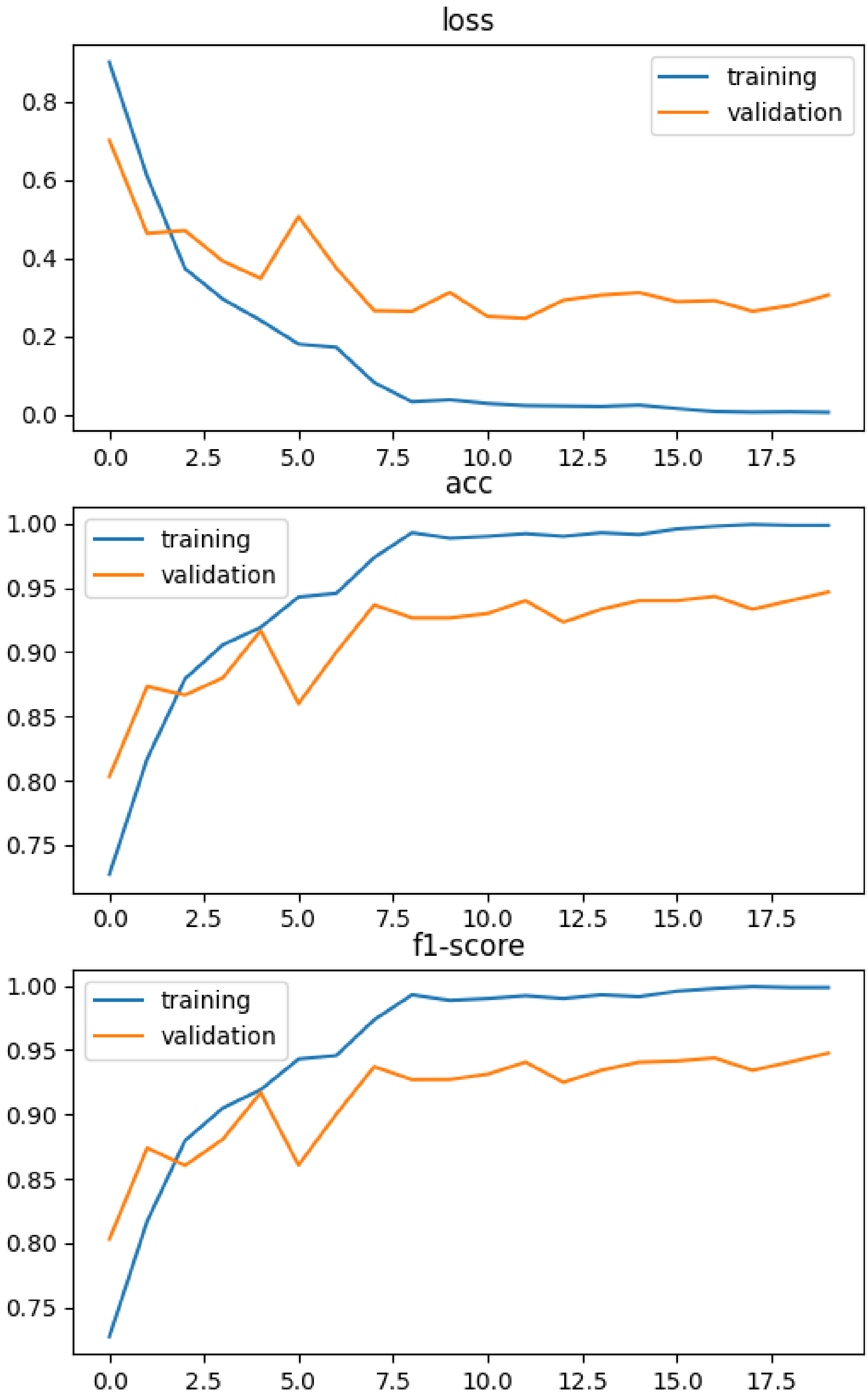
```
cuda:0
```

DATA STATISTICS:

```
trainset.__len__(), valset.__len__(), testset.__len__()
```

```
(1400, 300, 300)
```

LEARNING CURVE:



METRICS:

```
from sklearn.metrics import classification_report
```

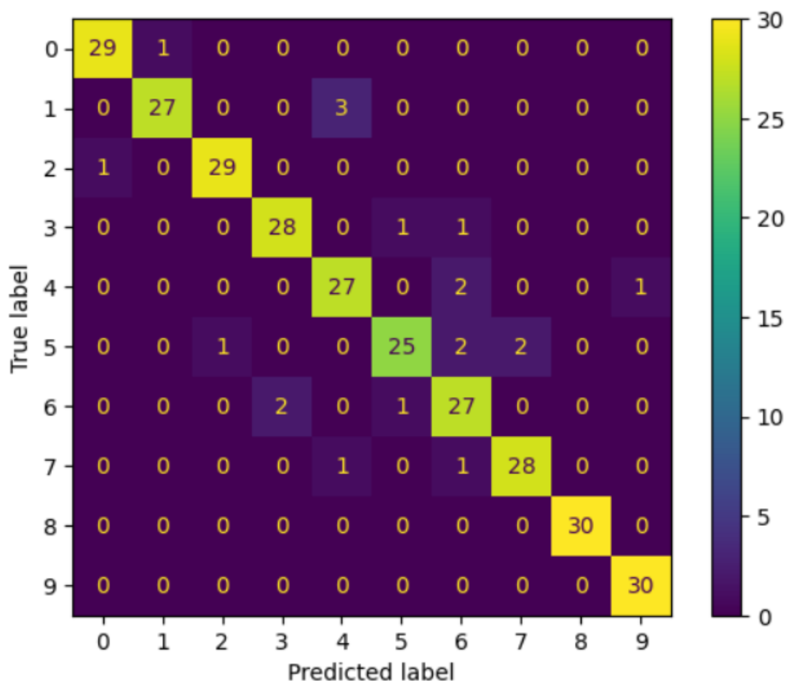
```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
```

DEMO THE RESULT:

```
print('testing ...')
y_predict = list()
y_labels = list()
test_loss = 0.0
n = 0
with torch.no_grad():
    for data in tqdm(testloader):
        net.eval()
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

        y_labels += list(labels.cpu().numpy())
        y_predict += list(outputs.argmax(dim=1).cpu().numpy())
        n+=1

    # print statistics
    test_loss /= n
    print(f"testing loss: {test_loss:.4}" )

    report = classification_report(y_labels, y_predict, digits = 4)
    M = confusion_matrix(y_labels, y_predict)
    print(report)
    disp = ConfusionMatrixDisplay(confusion_matrix=M)
```

```
testing ...
100% 10/10 [00:01<00:00, 5.49it/s]
testing loss: 0.2137
              precision    recall  f1-score   support

           0     0.9667    0.9667    0.9667        30
           1     0.9643    0.9000    0.9310        30
           2     0.9667    0.9667    0.9667        30
           3     0.9333    0.9333    0.9333        30
           4     0.8710    0.9000    0.8852        30
           5     0.9259    0.8333    0.8772        30
           6     0.8182    0.9000    0.8571        30
           7     0.9333    0.9333    0.9333        30
           8     1.0000    1.0000    1.0000        30
           9     0.9677    1.0000    0.9836        30

    accuracy                         0.9333       300
   macro avg     0.9347    0.9333    0.9334       300
weighted avg     0.9347    0.9333    0.9334       300
```

```
disp.plot()
plt.show()
```



FINETUNING TECHNIQUE:

Scheduler.step() is used for adjusting the learning rate during training.

Key features:

imshow(img)

- **Input:** a batch of images
- **Output:** grid of images using ‘matplotlib’

Training and validation loops:

- **Input:** Training and validation Data loader
- **Output:** Training and validation loss, accuracy, F1-score

Testing Loop:

- **Input:** Testing Data loader
- **Output:** Testing loss, classification report, confusion matrix