DSDE-Homework 2_Image_classification_Animal_EfficientNetV2

1. Describe the input and output for each model, hardware requirement, data statistic, learning curve, metrics (train text val), demo the result, finetuning technique, etc.

ทำ augmentation รูป animal 10 classes
ใช้ EfficientNetV2 มา fine-tune classification layer ให้เป็น 10 animal classes

2. List key features for each function, including input and output. (cheat sheet)

download animal image dataset

```
!wget https://github.com/pvateekul/2110531_DSDE_2023s1/raw/main/code/Week05_Intro_Deep_Learning/data/Dataset_animal2.zip
!unzip -q -o 'Dataset_animal2.zip'
```

ทำ augmentation rotation, cropping, flipping

```
transform_train = transforms.Compose(
    [transforms.Resize((230,230)),
     transforms.RandomRotation(30,),
     transforms.RandomCrop(224),
     transforms.RandomHorizontalFlip(),
     transforms.RandomVerticalFlip(),
     transforms.ToTensor(),
     transforms.Normalize(mean=[0.507, 0.487, 0.441], std=[0.267, 0.256, 0.276])]
)

transform = transforms.Compose(
    [transforms.Resize((224,224)),
     transforms.ToTensor(),
     transforms.Normalize(mean=[0.507, 0.487, 0.441], std=[0.267, 0.256,
0.276])]
)
```

สร้าง class สำหรับ dataset

```
class AnimalDataset(Dataset):
    def __init__(self, img_dir, transforms=None):
        self.label_image = ['butterfly', 'cat', 'chicken', 'cow', 'dog',
'elephant', 'horse', 'sheep', 'spider', 'squirrel']
        self.input_dataset = [(os.path.join(img_dir, label, image_name),
label_num)
                              for label_num, label in
enumerate(self.label_image)
                              for image_name in
os.listdir(os.path.join(img_dir, label))]
        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
```

แบ่ง train, validate, test และสร้าง Dataloader

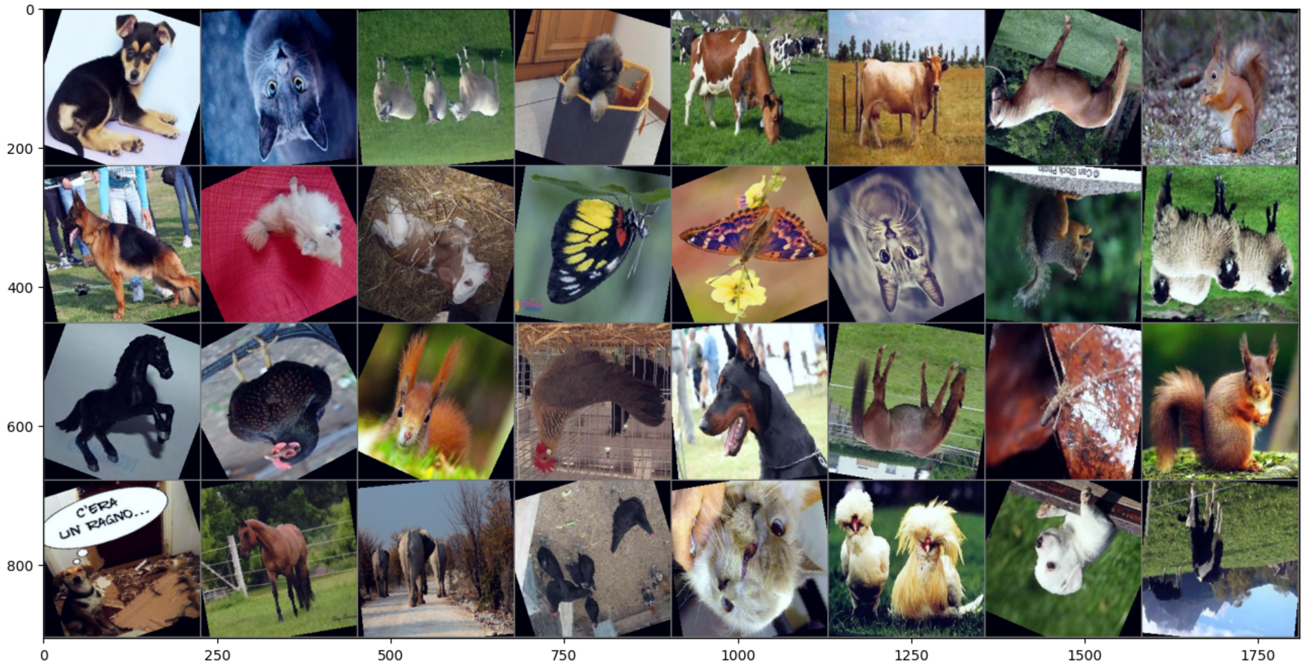
```
trainset = AnimalDataset('./Dataset_animal2/train', transform_train)
valset = AnimalDataset('./Dataset_animal2/val', transform)
testset = AnimalDataset('./Dataset_animal2/test', transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=32,
shuffle=True)
valloader = torch.utils.data.DataLoader(valset, batch_size=32,
shuffle=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=32,
shuffle=True)
```

function สำหรับ plot รูปในแต่ละคลาส

```
def PlotRandomFromEachClass(dataset, N, labels):
    Y = [label for _, label in dataset.input_dataset]
    M = len(np.unique(Y))
    plt.figure(figsize=(16, N*1.5))
    for i in range(M):
        indexes = np.random.choice(np.where(np.array(Y) == i)[0], N,
replace=False)
        for j in range(N):
            img = Image.open(dataset.input_dataset[indexes[j]]
[0]).convert('RGB')
            plt.subplot(N, M, j*M + i + 1)
            plt.imshow(img)
            plt.axis("off")
            if j == 0:
                plt.title(labels[i])
```

รูปที่ผ่านการทำ Augmentation



ดูจำนวนใน train, val และ test

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

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

โหลด pretrain weight EfficientNetV2

```
pretrain_weight =
torchvision.models.EfficientNet_V2_S_Weights.IMAGENET1K_V1
net = torchvision.models.efficientnet_v2_s(weights=pretrain_weight)
net.classifier[1] = nn.Linear(1280, 10)
net = net.to(device)
```

ดูสรุป network parameters

```
from torchsummary import summary
summary(net, (3, 224, 224), batch_size = 64)
```

```
================================================================ Total
params: 20,190,298 Trainable params: 20,190,298 Non-trainable params: 0 --
-------------------------------------------------------------- Input size
(MB): 36.75 Forward/backward pass size (MB): 20629.03 Params size (MB):
77.02 Estimated Total Size (MB): 20742.80 -------------------------------
---------------------------------
```

กำหนด loss function
ใช้ SGD เป็น optimizer
และใช้ learning rate schedule

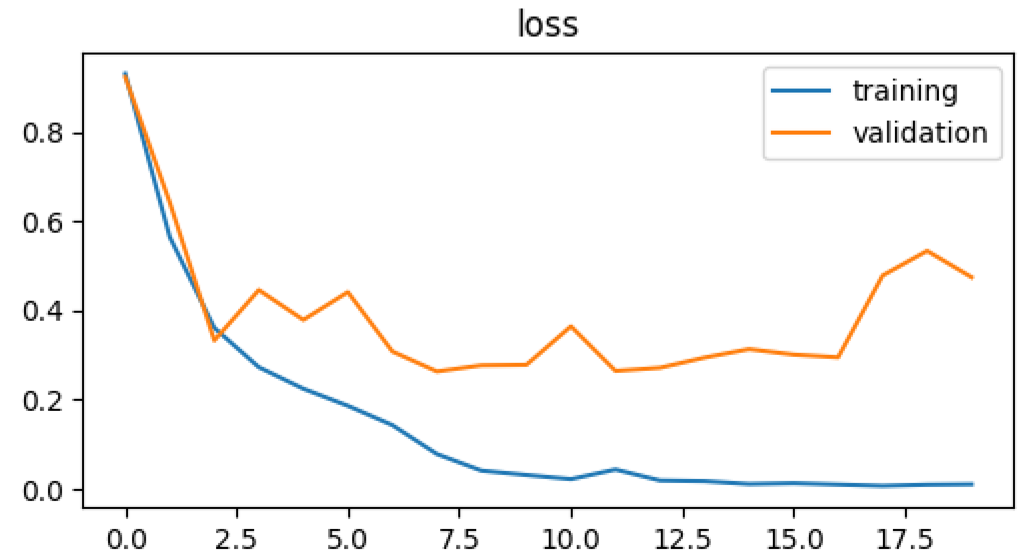
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.02, momentum=0.9)
scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.5)
```

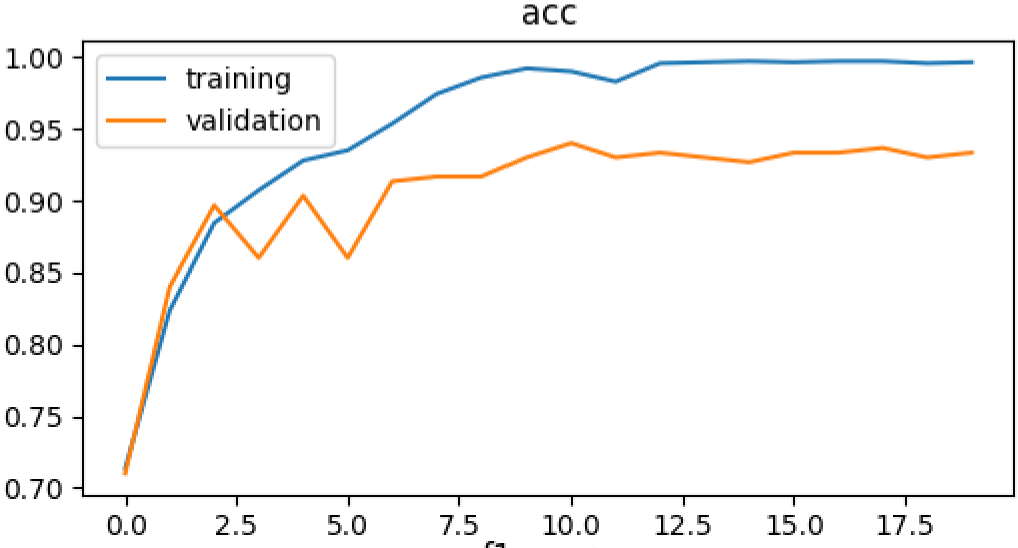
train loop

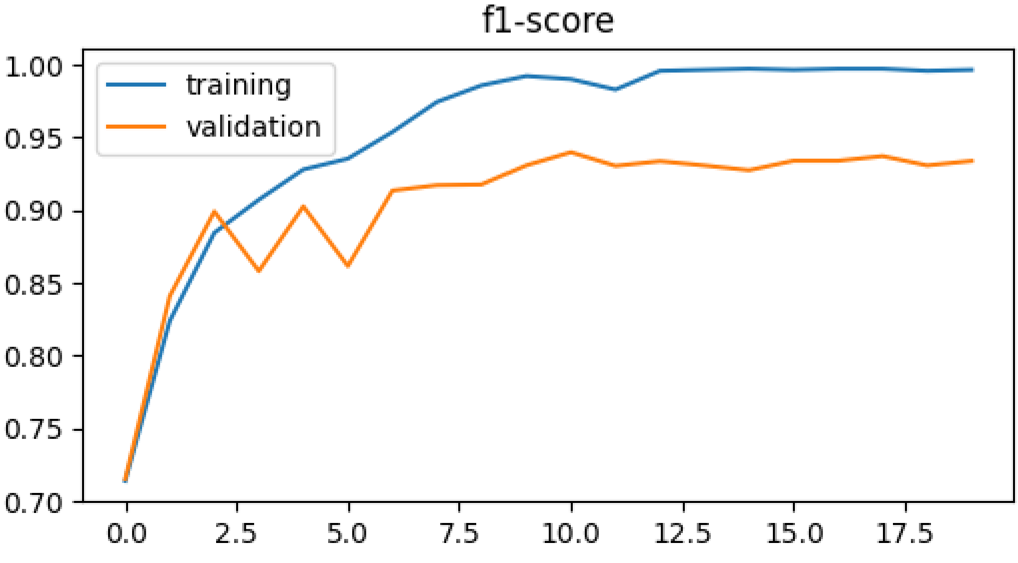
```
for epoch in range(20):  # loop over the dataset multiple times
    net.train()
    for inputs, labels in tqdm(trainloader):
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
    scheduler.step()
```

plot statistic การ train

```python
fig, axs = plt.subplots(3, figsize= (6,10))
# loss
axs[0].plot(history_train['loss'], label = 'training')
axs[0].plot(history_val['loss'], label = 'validation')
axs[0].set_title("loss")
axs[0].legend()
# acc
axs[1].plot(history_train['acc'], label = 'training')
axs[1].plot(history_val['acc'], label = 'validation')
axs[1].set_title("acc")
axs[1].legend()
# f1-score
axs[2].plot(history_train['f1-score'], label = 'training')
axs[2].plot(history_val['f1-score'], label = 'validation')
axs[2].set_title("f1-score")
axs[2].legend()
plt.show()
```







Evaluate model

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay


print('testing ...')
y_predict = list()
y_labels = list()
```

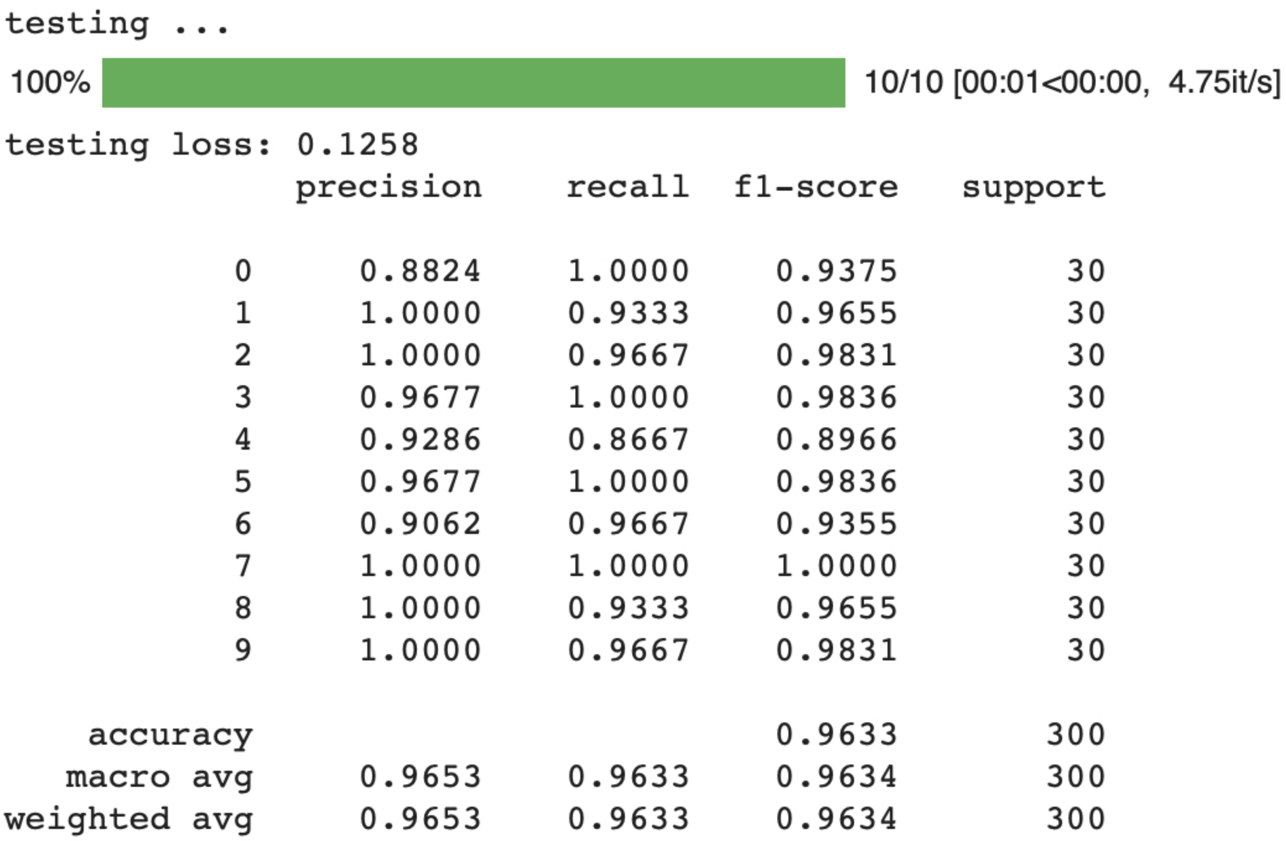
```
test_loss = 0.0
n = 0
with torch.no_grad():
        for data in tqdm(testloader):
                net.eval()
                inputs, labels = data
                inputs = inputs.to(device)
                labels = labels.to(device)

                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())
                # To get probabilities, you can run a softmax on outputs
                y_probs = torch.nn.functional.softmax(outputs, dim=1)
                y_probs = list(y_probs.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,  4.75it/s]
testing loss: 0.1258
              precision    recall  f1-score   support

           0     0.8824    1.0000    0.9375        30
           1     1.0000    0.9333    0.9655        30
           2     1.0000    0.9667    0.9831        30
           3     0.9677    1.0000    0.9836        30
           4     0.9286    0.8667    0.8966        30
           5     0.9677    1.0000    0.9836        30
           6     0.9062    0.9667    0.9355        30
           7     1.0000    1.0000    1.0000        30
           8     1.0000    0.9333    0.9655        30
           9     1.0000    0.9667    0.9831        30

    accuracy                         0.9633       300
   macro avg     0.9653    0.9633    0.9634       300
weighted avg     0.9653    0.9633    0.9634       300
```