# 1_Image_classification

NVIDIA System Management Interface ซึ่งใช้แสดงข้อมูลและจัดการการใช้งานของ GPU บนระบบ

```
! nvidia-smi
```

โค้ดนี้ทำหน้าที่ปิดการแจ้งเตือน (warnings) ที่เกิดจากไลบรารี scikit-learn เมื่อมีการคำนวณเมตริกที่ไม่ได้กำหนดไว้

```
from sklearn.exceptions import UndefinedMetricWarning

def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

โค้ดนี้ใช้ไลบรารี PyTorch และ torchvision ที่เป็นไลบรารีที่ถูกพัฒนาขึ้นสำหรับงานด้าน deep learning และ computer vision ใน Python

```
import torch
import torchvision
import torchvision.transforms as transforms
```

ตรวจสอบว่าเครื่องคอมพิวเตอร์ที่ใช้รันโค้ดนี้มี GPU (กราฟิกประมวลผลหลัก) ที่สามารถใช้งานได้หรือไม่ โดยทำการตั้งค่าตัวแปร **`device`** ให้กับ CUDA GPU ถ้ามี, และถ้าไม่มีก็จะให้ใช้ CPU แทน

```
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)
```

แปลงรูปภาพเป็น tensor และปรับขนาดของรูปภาพเป็นขนาดที่กำหนด

```
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
```

สร้าง Dataset สำหรับการฝึกและ Validation และ สร้าง DataLoader สำหรับ Trainset และ Validation

```
torchvision.datasets
```

```
<module 'torchvision.datasets' from '/usr/local/lib/python3.10/dist-packages/torchvision/datasets/__init__.py'>
```

```
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')
```

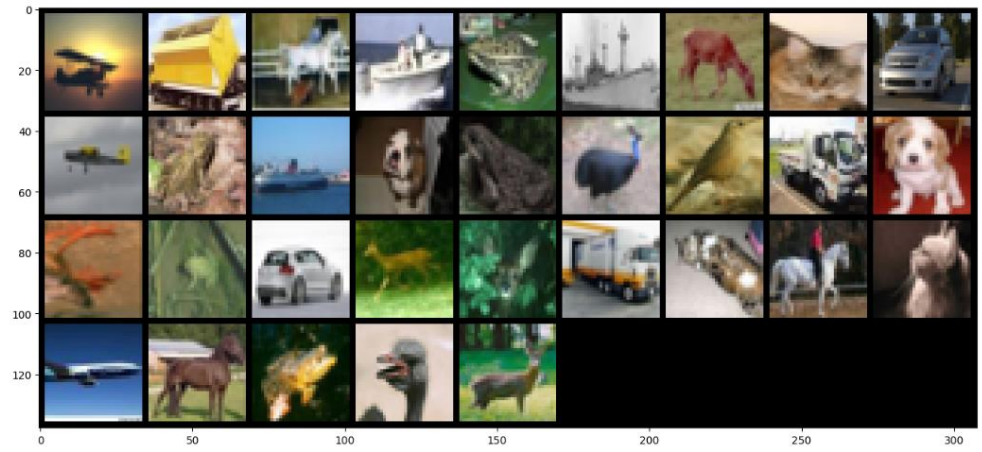
ใช้เพื่อดูขนาด (จำนวนตัวอย่าง) ของ Dataset สำหรับการฝึก (trainset), Dataset สำหรับ Validation และ Dataset สำหรับการทดสอบ (testset)

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

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

def imshow(img): เป็นการแสดงรูปภาพ ส่วน iter(trainloader) คือการสุ่มรูปภาพมาจาก trainloader โดยกำหนดให้1 แถวแสดง 9 ช่อง

```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image
def imshow(img):
    npimg = img.numpy()
    plt.figure(figsize=(16,16))
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

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

# show images
nrow = 9
imshow(torchvision.utils.make_grid(images, nrow = nrow))
```



พิมพ์ Label ของรูปภาพที่ถูกดึงมาจาก **trainloader** ในรูปแบบของกริด โดยใช้ลูปเพื่อแสดง Label ของแต่ละรูปภาพในแต่ละแถวของกริด

```
# print labels
for i in range(batch_size//nrow + 1 if batch_size % nrow else 0):
  print(' '.join(f'{labels[i*nrow+j]:<3}' for j in range(min(batch_size - i*nrow, nrow))))
```

```
0   9   7   8   6   8   4   3   1
0   6   8   5   6   2   2   9   5
4   6   1   4   4   9   3   7   3
0   7   6   2   4
```

สร้างและกำหนดโมเดล CNN ที่ใช้ในการจำแนกประเภทของภาพ (image classification) ด้วย PyTorch และ มีลักษณะของชั้น convolution, pooling, และ fully connected layers

```
import torch.nn as nn
import torch.nn.functional as F

class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5) # 3 input channels, 6 output channels, 5*5 kernel size
        self.pool = nn.MaxPool2d(2, 2) # 2*2 kernel size, 2 strides
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(400, 120) # dense input 400 (16*5), output 120

        self.fc2 = nn.Linear(120, 84) # dense input 120, output 84
        self.fc3 = nn.Linear(84, 10) # dense input 84, output 10
        self.softmax = torch.nn.Softmax(dim=1) # perform softmax at dim[1] (batch,class)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x,start_dim=1) # flatten all dimensions (dim[1]) except batch (dim[0])
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        x = self.softmax(x)
        return x

net = CNN().to(device)
```

ติดตั้งไลบรารี torchinfo ที่ใช้สำหรับแสดงข้อมูลเกี่ยวกับโมเดล PyTorch

```
!pip install torchinfo
```

ใช้ torchinfo เพื่อแสดงข้อมูลสรุปเกี่ยวกับโมเดล PyTorch

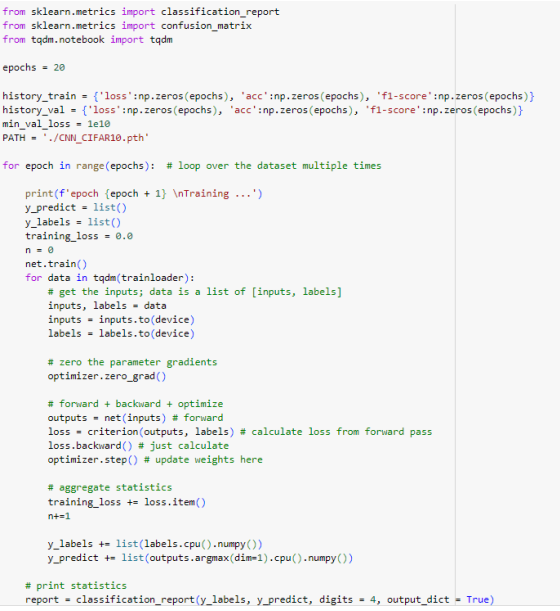
```
from torchinfo import summary as summary_info
print(summary_info(net, input_size = (32, 3, 32, 32))) # (batchsize,channel,width,height)
net = net.to(device)
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNN                                      [32, 10]                  --
├─Conv2d: 1-1                            [32, 6, 28, 28]           456
├─MaxPool2d: 1-2                         [32, 6, 14, 14]           --
├─Conv2d: 1-3                            [32, 16, 10, 10]          2,416
├─MaxPool2d: 1-4                         [32, 16, 5, 5]            --
├─Linear: 1-5                            [32, 120]                 48,120
├─Linear: 1-6                            [32, 84]                  10,164
├─Linear: 1-7                            [32, 10]                  850
├─Softmax: 1-8                           [32, 10]                  --
==========================================================================================
Total params: 62,006
Trainable params: 62,006
Non-trainable params: 0
Total mult-adds (M): 21.06
==========================================================================================
Input size (MB): 0.39
Forward/backward pass size (MB): 1.67
Params size (MB): 0.25
Estimated Total Size (MB): 2.31
==========================================================================================
```

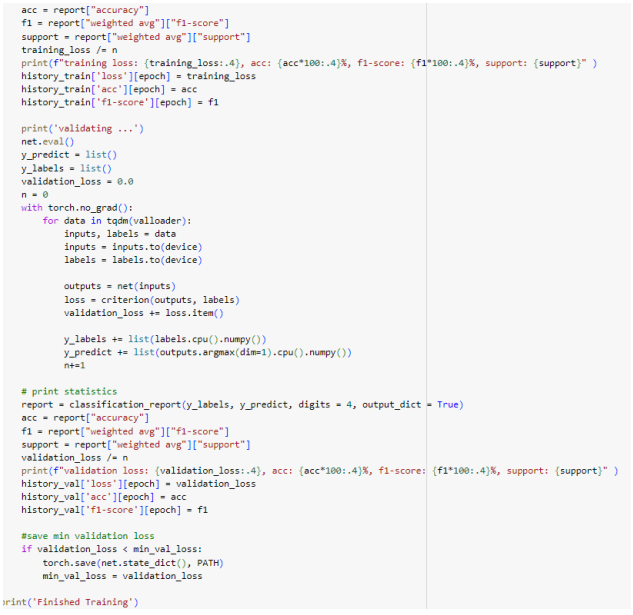
ฝึกโมเดล PyTorch (criterion), และใช้ตัวอัพเดตพารามิเตอร์ของโมเดลด้วยวิธีการ Stochastic Gradient Descent (SGD) ที่กำหนด learning rate เป็น 1e-2 และ momentum เป็น 0.9

```
import torch.optim as optim

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

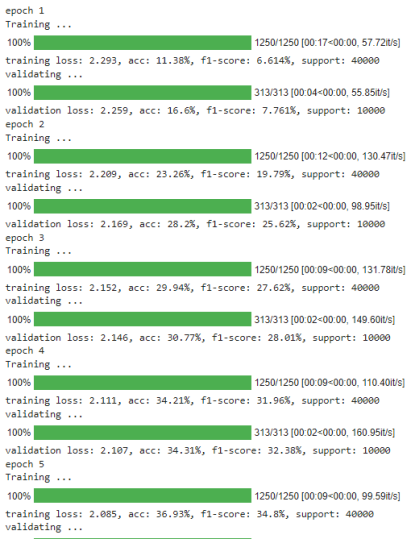
ฝึกโมเดล CNN สำหรับการจำแนกภาพด้วย PyTorch โดยใช้ SGD ในแต่ละ epoch และบันทึกผลลัพธ์การฝึกและการทดสอบ เพื่อเลือกโมเดลที่มีค่า validation loss ต่ำที่สุด

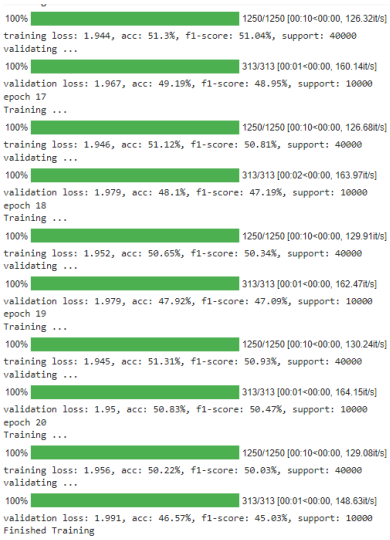


```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from tqdm.notebook import tqdm

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()
    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

    #save min validation loss
    if validation_loss < min_val_loss:
        torch.save(net.state_dict(), PATH)
        min_val_loss = validation_loss

print('Finished Training')
```



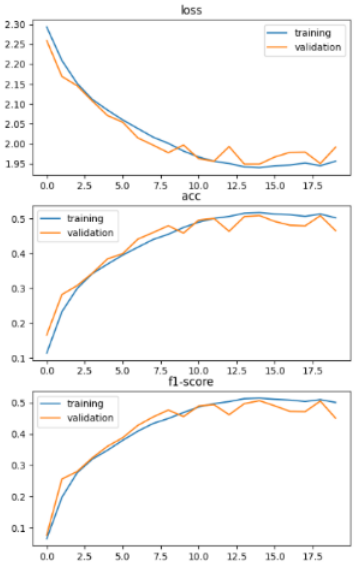




min_val_loss เก็บค่า validation loss ที่ต่ำที่สุดที่พบในระหว่างการฝึกโมเดล

```
min_val_loss
```

```
1.948860833058342
```

ใช้ Matplotlib เพื่อสร้างกราฟเส้นแสดงค่า loss, accuracy (acc), และ f1-score ของโมเดลในระหว่างการฝึกและการทดสอบ




```
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()
```

สร้างโมเดล CNN และโหลดน้ำหนักที่ถูกบันทึกไว้ (PATH) เพื่อนำมาใช้ในการทดสอบ

```
net = CNN().to(device)
net.load_state_dict(torch.load(PATH))
```

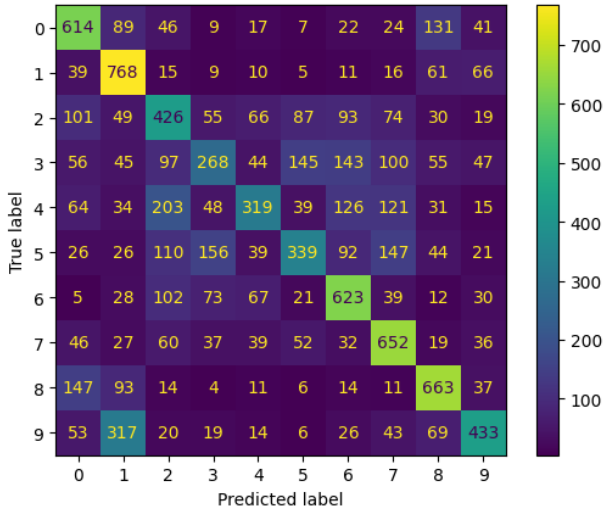
```
<All keys matched successfully>
```

ทดสอบโมเดล CNN ด้วยชุดข้อมูลทดสอบและคำนวณค่า test loss รวมถึงแสดงผล confusion matrix และรายงานประสิทธิภาพด้วย classification report โดยใช้ sklearn

```python
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):
        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)
    #acc = report["accuracy"]
    #f1 = report["weighted avg"]["f1-score"]
    #support = report["weighted avg"]["support"]
    #test_loss /= n
    #print(f"validation loss: {test_loss:.4}, acc: {acc*100:.4}%, f1-score: {f1*100:.4}%, support: {support}" )
```

```
testing ...
100% 313/313 [00:02<00:00, 145.34it/s]
testing loss: 1.946
              precision    recall  f1-score   support

           0     0.5334    0.6140    0.5709      1000
           1     0.5203    0.7680    0.6204      1000
           2     0.3898    0.4260    0.4071      1000
           3     0.3953    0.2680    0.3194      1000
           4     0.5096    0.3190    0.3924      1000
           5     0.4795    0.3390    0.3972      1000
           6     0.5271    0.6230    0.5710      1000
           7     0.5314    0.6520    0.5855      1000
           8     0.5946    0.6630    0.6270      1000
           9     0.5812    0.4330    0.4963      1000

    accuracy                         0.5105     10000
   macro avg     0.5062    0.5105    0.4987     10000
weighted avg     0.5062    0.5105    0.4987     10000
```

ใช้ Matplotlib เพื่อแสดงผลกราฟของ confusion matrix ที่ถูกสร้างขึ้นและบันทึกใน disp จาก sklearn



2_Image_classification_Animal_EfficientNetV2.ipynb

ใช้ Albumentations เพื่อสร้างการเพิ่มความหลากหลายในข้อมูลภาพสำหรับการ transform_train) และใช้การปรับขนาดภาพและทำ Normalization สำหรับข้อมูลทดสอบ

```
import albumentations as A
from albumentations.pytorch import ToTensorV2

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]) #nomalize imagenet pretrain
    ])

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])
    ])

batch_size = 32
```

สร้างคลาส AnimalDataset สำหรับโมเดล PyTorch โดยใช้ Albumentations transforms สำหรับข้อมูลฝึก transform_train และ Albumentations transforms ที่ทำงานเฉพาะกับ PyTorch สำหรับข้อมูลทดสอบ

```
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

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

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

ใช้เพื่อดูขนาด (จำนวนตัวอย่าง) ของ Dataset สำหรับการฝึก (trainset), Dataset สำหรับ Validation และ Dataset สำหรับการทดสอบ (testset)

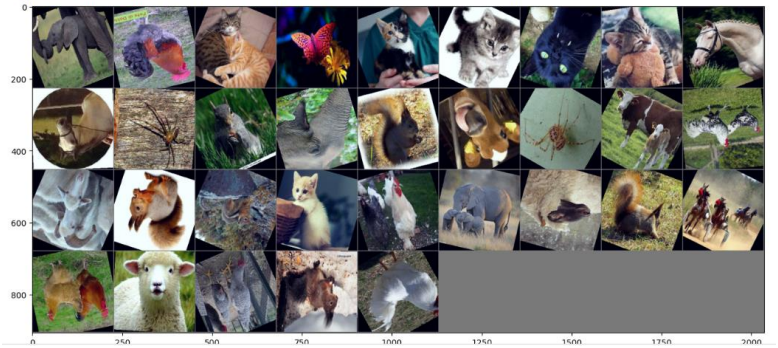
```
trainset.__len__(), valset.__len__(), testset.__len__()
```

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

def imshow(img): เป็นการแสดงรูปภาพ ส่วน iter(trainloader) คือการสุ่มรูปภาพมาจาก trainloader โดยกำหนดให้1 แถวแสดง 9 ช่อง

```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image
def imshow(img):
    img = img*torch.tensor([0.267, 0.256, 0.276]).mean() + torch.tensor([0.507, 0.487, 0.441]).mean()     # unnormalize
    npimg = img.numpy()
    plt.figure(figsize=(16,16))
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

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

# show images
nrow = 9
imshow(torchvision.utils.make_grid(images, nrow = nrow))
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```




พิมพ์ Label ของรูปภาพที่ถูกดึงมาจาก trainloader ในรูปแบบของกริด โดยใช้ลูปเพื่อแสดง Label ของแต่ละรูปภาพในแต่ละแถวของกริด

```
# print labels
for i in range(batch_size//nrow + 1 if batch_size % nrow else 0):
  print(' '.join(f'{labels[i*nrow+j]:<3}' for j in range(min(batch_size - i*nrow, nrow))))
```

```
5   2   1   0   1   1   1   1   6
6   8   9   5   9   3   8   3   2
7   9   9   1   2   5   7   9   6
2   7   2   9   2
```

ใช้โมเดล EfficientNetV2 จาก torchvision และโหลดน้ำหนักที่ถูกฝึกจาก ImageNet เพื่อใช้ในการจำแนกประเภท

```
import torch.nn as nn
import torch.nn.functional as F

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)
```

```
Downloading: "https://download.pytorch.org/models/efficientnet_v2_s-dd5fe13b.pth" to /root/.cache/torch/hub/checkpoints/efficientnet_v2_s-dd5fe13b.pth
100%|██████████| 82.7M/82.7M [00:00<00:00, 165MB/s]
```

ใช้ไลบรารี torchsummary เพื่อแสดงสรุปข้อมูลโครงสร้างของโมเดล ด้วยขนาดของข้อมูลนำเข้าเป็น (3, 224, 224) และ batch size เท่ากับ 64

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

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1         [64, 24, 112, 112]             648
     BatchNorm2d-2         [64, 24, 112, 112]              48
            SiLU-3         [64, 24, 112, 112]               0
          Conv2d-4         [64, 24, 112, 112]           5,184
     BatchNorm2d-5         [64, 24, 112, 112]              48
            SiLU-6         [64, 24, 112, 112]               0
 StochasticDepth-7         [64, 24, 112, 112]               0
      FusedMBConv-8         [64, 24, 112, 112]               0
          Conv2d-9         [64, 24, 112, 112]           5,184
    BatchNorm2d-10         [64, 24, 112, 112]              48
           SiLU-11         [64, 24, 112, 112]               0
 StochasticDepth-12         [64, 24, 112, 112]               0
     FusedMBConv-13         [64, 24, 112, 112]               0
          Conv2d-14          [64, 96, 56, 56]          20,736
    BatchNorm2d-15          [64, 96, 56, 56]             192
           SiLU-16          [64, 96, 56, 56]               0
          Conv2d-17          [64, 48, 56, 56]           4,608
    BatchNorm2d-18          [64, 48, 56, 56]              96
     FusedMBConv-19          [64, 48, 56, 56]               0
          Conv2d-20         [64, 192, 56, 56]          82,944
    BatchNorm2d-21         [64, 192, 56, 56]             384
           SiLU-22         [64, 192, 56, 56]               0
          Conv2d-23          [64, 48, 56, 56]           9,216
    BatchNorm2d-24          [64, 48, 56, 56]              96
 StochasticDepth-25          [64, 48, 56, 56]               0
     FusedMBConv-26          [64, 48, 56, 56]               0
          Conv2d-27         [64, 192, 56, 56]          82,944
    BatchNorm2d-28         [64, 192, 56, 56]             384
           SiLU-29         [64, 192, 56, 56]               0
          Conv2d-30          [64, 48, 56, 56]           9,216
    BatchNorm2d-31          [64, 48, 56, 56]              96
 StochasticDepth-32          [64, 48, 56, 56]               0
     FusedMBConv-33          [64, 48, 56, 56]               0
          Conv2d-34         [64, 192, 56, 56]          82,944
    BatchNorm2d-35         [64, 192, 56, 56]             384
           SiLU-36         [64, 192, 56, 56]               0
          Conv2d-37          [64, 48, 56, 56]           9,216
    BatchNorm2d-38          [64, 48, 56, 56]              96
 StochasticDepth-39          [64, 48, 56, 56]               0
     FusedMBConv-40          [64, 48, 56, 56]               0
          Conv2d-41         [64, 192, 56, 56]          82,944
    BatchNorm2d-42         [64, 192, 56, 56]             384
           SiLU-43         [64, 192, 56, 56]               0
          Conv2d-44          [64, 48, 56, 56]           9,216
    BatchNorm2d-45          [64, 48, 56, 56]              96
 StochasticDepth-46          [64, 48, 56, 56]               0
     FusedMBConv-47          [64, 48, 56, 56]               0
          Conv2d-48         [64, 192, 56, 56]          82,944
    BatchNorm2d-49         [64, 192, 56, 56]             384
           SiLU-50         [64, 192, 56, 56]               0
          Conv2d-51          [64, 48, 56, 56]           9,216
    BatchNorm2d-52          [64, 48, 56, 56]              96
 StochasticDepth-53          [64, 48, 56, 56]               0
     FusedMBConv-54          [64, 48, 56, 56]               0
          Conv2d-55         [64, 192, 56, 56]          82,944
    BatchNorm2d-56         [64, 192, 56, 56]             384
           SiLU-57         [64, 192, 56, 56]               0
          Conv2d-58          [64, 48, 56, 56]           9,216
    BatchNorm2d-59          [64, 48, 56, 56]              96
 StochasticDepth-60          [64, 48, 56, 56]               0
     FusedMBConv-61          [64, 48, 56, 56]               0
          Conv2d-62         [64, 192, 56, 56]          82,944
    BatchNorm2d-63         [64, 192, 56, 56]             384
           SiLU-64         [64, 192, 56, 56]               0
          Conv2d-65          [64, 48, 56, 56]           9,216
    BatchNorm2d-66          [64, 48, 56, 56]              96
 StochasticDepth-67          [64, 48, 56, 56]               0
     FusedMBConv-68          [64, 48, 56, 56]               0
          Conv2d-69         [64, 192, 56, 56]          82,944
    BatchNorm2d-70         [64, 192, 56, 56]             384
           SiLU-71         [64, 192, 56, 56]               0
          Conv2d-72          [64, 48, 56, 56]           9,216
    BatchNorm2d-73          [64, 48, 56, 56]              96
 StochasticDepth-74          [64, 48, 56, 56]               0
     FusedMBConv-75          [64, 48, 56, 56]               0
          Conv2d-76         [64, 192, 56, 56]          82,944
    BatchNorm2d-77         [64, 192, 56, 56]             384
           SiLU-78         [64, 192, 56, 56]               0
          Conv2d-79          [64, 48, 56, 56]           9,216
    BatchNorm2d-80          [64, 48, 56, 56]              96
 StochasticDepth-81          [64, 48, 56, 56]               0
     FusedMBConv-82          [64, 48, 56, 56]               0
          Conv2d-83         [64, 192, 56, 56]          82,944
    BatchNorm2d-84         [64, 192, 56, 56]             384
           SiLU-85         [64, 192, 56, 56]               0
          Conv2d-86          [64, 48, 56, 56]           9,216
    BatchNorm2d-87          [64, 48, 56, 56]              96
 StochasticDepth-88          [64, 48, 56, 56]               0
     FusedMBConv-89          [64, 48, 56, 56]               0
          Conv2d-90         [64, 192, 56, 56]          82,944
    BatchNorm2d-91         [64, 192, 56, 56]             384
           SiLU-92         [64, 192, 56, 56]               0
          Conv2d-93          [64, 48, 56, 56]           9,216
    BatchNorm2d-94          [64, 48, 56, 56]              96
 StochasticDepth-95          [64, 48, 56, 56]               0
     FusedMBConv-96          [64, 48, 56, 56]               0
          Conv2d-97         [64, 192, 56, 56]          82,944
    BatchNorm2d-98         [64, 192, 56, 56]             384
           SiLU-99         [64, 192, 56, 56]               0
         Conv2d-100          [64, 48, 56, 56]           9,216
   BatchNorm2d-101          [64, 48, 56, 56]              96
StochasticDepth-102          [64, 48, 56, 56]               0
    FusedMBConv-103          [64, 48, 56, 56]               0
         Conv2d-104         [64, 192, 56, 56]          82,944
   BatchNorm2d-105         [64, 192, 56, 56]             384
          SiLU-106         [64, 192, 56, 56]               0
         Conv2d-107          [64, 48, 56, 56]           9,216
   BatchNorm2d-108          [64, 48, 56, 56]              96
StochasticDepth-109          [64, 48, 56, 56]               0
    FusedMBConv-110          [64, 48, 56, 56]               0
         Conv2d-111         [64, 192, 56, 56]          82,944
   BatchNorm2d-112         [64, 192, 56, 56]             384
          SiLU-113         [64, 192, 56, 56]               0
         Conv2d-114          [64, 48, 56, 56]           9,216
   BatchNorm2d-115          [64, 48, 56, 56]              96
StochasticDepth-116          [64, 48, 56, 56]               0
    FusedMBConv-117          [64, 48, 56, 56]               0
         Conv2d-118         [64, 192, 56, 56]          82,944
   BatchNorm2d-119         [64, 192, 56, 56]             384
          SiLU-120         [64, 192, 56, 56]               0
         Conv2d-121          [64, 48, 56, 56]           9,216
   BatchNorm2d-122          [64, 48, 56, 56]              96
StochasticDepth-123          [64, 48, 56, 56]               0
    FusedMBConv-124          [64, 48, 56, 56]               0
         Conv2d-125         [64, 256, 7, 7]          393,216
   BatchNorm2d-126         [64, 256, 7, 7]              512
StochasticDepth-127         [64, 256, 7, 7]                0
         MBConv-128         [64, 256, 7, 7]                0
         Conv2d-129        [64, 1536, 7, 7]          393,216
   BatchNorm2d-130        [64, 1536, 7, 7]            3,072
          SiLU-131        [64, 1536, 7, 7]                0
         Conv2d-132        [64, 1536, 7, 7]           13,824
   BatchNorm2d-133        [64, 1536, 7, 7]            3,072
          SiLU-134        [64, 1536, 7, 7]                0
AdaptiveAvgPool2d-135        [64, 1536, 1, 1]              0
         Conv2d-136          [64, 64, 1, 1]           98,368
          SiLU-137          [64, 64, 1, 1]                0
         Conv2d-138        [64, 1536, 1, 1]           99,840
       Sigmoid-139        [64, 1536, 1, 1]                0
SqueezeExcitation-140        [64, 1536, 7, 7]              0
         Conv2d-141         [64, 256, 7, 7]          393,216
   BatchNorm2d-142         [64, 256, 7, 7]              512
StochasticDepth-143         [64, 256, 7, 7]                0
         MBConv-144         [64, 256, 7, 7]                0
         Conv2d-145        [64, 1280, 7, 7]          327,680
   BatchNorm2d-146        [64, 1280, 7, 7]            2,560
          SiLU-147        [64, 1280, 7, 7]                0
AdaptiveAvgPool2d-148        [64, 1280, 1, 1]              0
        Dropout-149              [64, 1280]                0
         Linear-150                [64, 10]           12,810
================================================================
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
----------------------------------------------------------------
```

กำหนดฟังก์ชันความสูญเสียแบบ CrossEntropyLoss ตัวตั้งค่าการอัปเดตพารามิเตอร์ของโมเดลด้วยวิธีการ SGD โดยกำหนด learning rate เป็น 0.02 และ momentum เป็น 0.9. นอกจากนี้ยังกำหนด Scheduler ของ learning rate ด้วย StepLR ซึ่งลด learning rate ทุกระยะเวลาที่กำหนด (step_size=7) โดยการคูณ learning rate

```
import torch.optim as optim
from torch.optim import lr_scheduler

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)
```

ทำการฝึกโมเดล EfficientNetV2 ด้วยข้อมูลฝึกและตรวจสอบประสิทธิภาพในขณะที่บันทึกค่า loss, accuracy, และ f1-score ในแต่ละ epoch. โดยมีการใช้ CrossEntropyLoss เป็นฟังก์ชันความสูญเสีย, SGD เป็นตัวตั้งค่าการอัปเดตพารามิเตอร์, และ Scheduler ของ learning rate เพื่อลด learning rate

```
from sklearn.metrics import classification_report
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 = './Animal10-efficientnetV2s.pth'

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

    print(f'epoch {epoch + 1} \nTraining ...')
    net.train()
    y_predict = list()
    y_labels = list()
    training_loss = 0.0
    n = 0
    with torch.set_grad_enabled(True):
        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)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

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

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

```
    # 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()

    optimizer.zero_grad()

    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

    #save min validation loss
    if validation_loss < min_val_loss:
        torch.save(net.state_dict(), PATH)
        min_val_loss = validation_loss
print('Finished Training')
```

```
epoch 1
Training ...
100% 44/44 [00:24<00:00, 2.44it/s]
training loss: 0.9199, acc: 72.71%, f1-score: 73.12%, support: 1400
validating ...
100% 10/10 [00:01<00:00, 5.87it/s]
validation loss: 0.7861, acc: 79.0%, f1-score: 79.29%, support: 300
epoch 2
Training ...
100% 44/44 [00:18<00:00, 2.50it/s]
training loss: 0.6323, acc: 80.5%, f1-score: 80.4%, support: 1400
validating ...
100% 10/10 [00:01<00:00, 5.87it/s]
validation loss: 0.436, acc: 84.67%, f1-score: 84.7%, support: 300
epoch 3
Training ...
100% 44/44 [00:18<00:00, 2.53it/s]
training loss: 0.3498, acc: 89.07%, f1-score: 89.09%, support: 1400
validating ...
100% 10/10 [00:01<00:00, 4.82it/s]
validation loss: 0.4421, acc: 86.0%, f1-score: 86.12%, support: 300
epoch 4
Training ...
100% 44/44 [00:18<00:00, 2.53it/s]
training loss: 0.3235, acc: 89.93%, f1-score: 89.91%, support: 1400
validating ...
100% 10/10 [00:01<00:00, 5.90it/s]
validation loss: 0.2992, acc: 89.0%, f1-score: 89.05%, support: 300
epoch 5
Training ...
100% 44/44 [00:19<00:00, 2.53it/s]
training loss: 0.2004, acc: 93.57%, f1-score: 93.58%, support: 1400
validating ...
100% 10/10 [00:01<00:00, 5.82it/s]
validation loss: 0.369, acc: 89.33%, f1-score: 89.29%, support: 300
epoch 6
Training ...
```

```
validation loss: 0.2485, acc: 95.0%, f1-score: 94.96%, support: 300
epoch 16
Training ...
100% 44/44 [00:19<00:00, 2.34it/s]
training loss: 0.0124, acc: 99.64%, f1-score: 99.64%, support: 1400
validating ...
100% 10/10 [00:01<00:00, 5.61it/s]
validation loss: 0.2341, acc: 95.0%, f1-score: 94.98%, support: 300
epoch 17
Training ...
100% 44/44 [00:19<00:00, 2.47it/s]
training loss: 0.009672, acc: 99.71%, f1-score: 99.71%, support: 1400
validating ...
100% 10/10 [00:01<00:00, 5.04it/s]
validation loss: 0.2359, acc: 95.33%, f1-score: 95.31%, support: 300
epoch 18
Training ...
100% 44/44 [00:19<00:00, 2.45it/s]
training loss: 0.01126, acc: 99.71%, f1-score: 99.71%, support: 1400
validating ...
100% 10/10 [00:01<00:00, 5.53it/s]
validation loss: 0.234, acc: 95.0%, f1-score: 95.01%, support: 300
epoch 19
Training ...
100% 44/44 [00:19<00:00, 2.43it/s]
training loss: 0.0104, acc: 99.71%, f1-score: 99.71%, support: 1400
validating ...
100% 10/10 [00:01<00:00, 5.69it/s]
validation loss: 0.231, acc: 94.33%, f1-score: 94.33%, support: 300
epoch 20
Training ...
100% 44/44 [00:19<00:00, 2.39it/s]
training loss: 0.01399, acc: 99.36%, f1-score: 99.36%, support: 1400
validating ...
100% 10/10 [00:01<00:00, 5.54it/s]
validation loss: 0.2241, acc: 94.67%, f1-score: 94.65%, support: 300
Finished Training
```

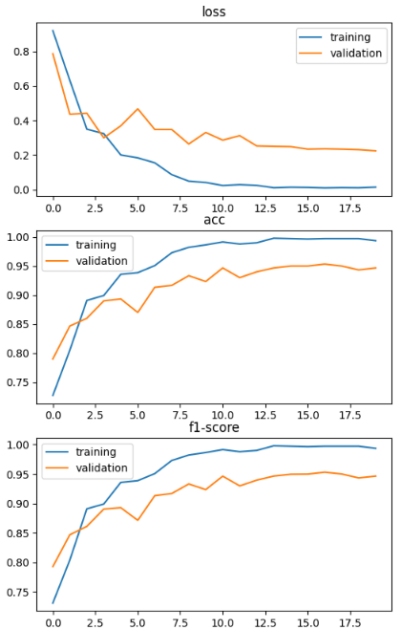
min_val_loss เก็บค่า validation loss ที่ต่ำที่สุดที่พบในระหว่างการฝึกโมเดล

```
min_val_loss
```

```
0.22409853991121054
```

ใช้ Matplotlib เพื่อสร้างกราฟที่แสดงค่า loss, accuracy, และ f1-score ของโมเดลในแต่ละ epoch จาก training data และ test data

```
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()
```



ใช้ PyTorch เพื่อโหลดน้ำหนัก (weights) ของโมเดล EfficientNetV2 ที่ถูกบันทึกไว้ในไฟล์ที่กำหนด (PATH) เพื่อนำมาใช้ในการทดสอบ

```
PATH = './Animal10-efficientnetV2s.pth'
net.load_state_dict(torch.load(PATH))
```

```
<All keys matched successfully>
```

ทำการทดสอบโมเดล EfficientNetV2 ด้วยข้อมูลทดสอบและแสดงผลลัพธ์ด้วยตัวช่วยที่เกี่ยวข้องกับ Scikit-Learn

```
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())
        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, 5.56it/s]
testing loss: 0.2259
              precision    recall  f1-score   support

           0     0.8788    0.9667    0.9206        30
           1     1.0000    0.9000    0.9474        30
           2     0.9333    0.9333    0.9333        30
           3     0.9355    0.9667    0.9508        30
           4     0.8824    1.0000    0.9375        30
           5     1.0000    0.9667    0.9831        30
           6     0.9355    0.9667    0.9508        30
           7     1.0000    0.9000    0.9474        30
           8     0.9643    0.9000    0.9310        30
           9     1.0000    1.0000    1.0000        30

    accuracy                         0.9500       300
   macro avg     0.9530    0.9500    0.9502       300
weighted avg     0.9530    0.9500    0.9502       300
```

ใช้ Matplotlib เพื่อแสดงแผนภูมิ confusion matrix ที่ถูกสร้างขึ้นโดย ConfusionMatrixDisplay

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