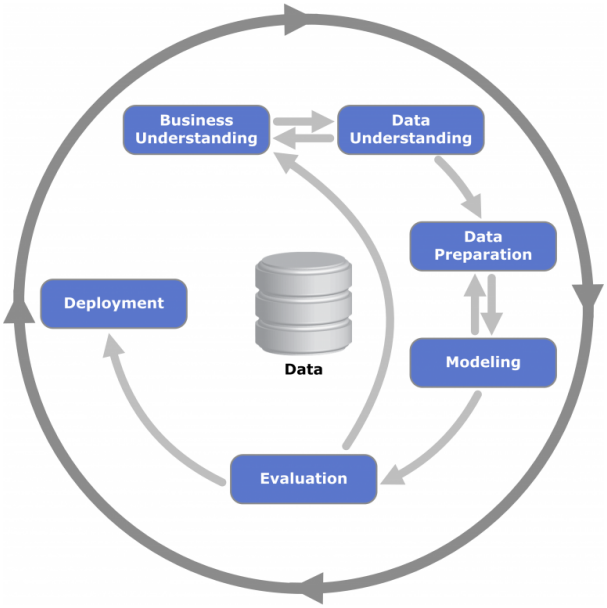
Topic : HOMEWORK-4



Follow the CRSIP-DM method

1. Step 1: Import library, import data
2. Step 2: Pre-processing (missing data, categorical type, normalization, format transform, data augmentation)
3. Step 3: Build ML Model
4. Step 4: Evaluate Model
5. Step 5: Deploy (Prediction)

Step 1: Load data (also import library)

```
!pip  install  torch  torchvision
!pip  install  Pillow
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/s
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (1.11.0+cu113)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packages (0.12.0+
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torchvi
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from torc
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/pyt
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/s
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages (7.1.2)
```

```
#  import  library
import  torch
import  numpy  as  np
import  pandas  as  pd
import  matplotlib.pyplot  as  plt
from  torch  import  nn
import  torch.nn.functional  as  F
from  torchvision  import  datasets,  transforms,  models

device  =  torch.device("cuda:0"  if  torch.cuda.is_available()  else  "cpu")
print(device)
```

```
cuda:0
```

```
#  get  data    (HW4要找一個github,裡面有dataset的folder)
#!git  clone  https://github.com/jaddoescad/ants_and_bees.git
!git  clone  https://github.com/laxmimerit/dog-cat-full-dataset.git
#  !rm  -rf  ./woman-man-recog/
#  !rm  -rf  ./GenderRecognizer//
```

```
Cloning into 'dog-cat-full-dataset'...
remote: Enumerating objects: 25027, done.
remote: Total 25027 (delta 0), reused 0 (delta 0), pack-reused 25027
Receiving objects: 100% (25027/25027), 541.62 MiB | 32.97 MiB/s, done.
Resolving deltas: 100% (5/5), done.
Checking out files: 100% (25001/25001), done.
```

```
!ls  ./dog-cat-full-dataset/data/train/
```

```
cats  dogs
```

Step 2: Pre-process X, Y

- format transform (轉換成numpy format)
- missing data (imputation)差補
- category data transform
- data augmentation
- normalization

```
transform_train  =  transforms.Compose([transforms.Resize((224,224)),
                    transforms.RandomHorizontalFlip(),
                    transforms.RandomAffine(0,  shear=10,  scale=(0.8,1.2)),
                    transforms.ColorJitter(brightness=1,  contrast=1,  saturation=1),
                    transforms.ToTensor(),
                    transforms.Normalize((0.5,  0.5,  0.5),  (0.5,  0.5,  0.5))
                    ])
```

```
transform = transforms.Compose([transforms.Resize((224,224)),
transforms.ToTensor(),
transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

training_dataset = datasets.ImageFolder('dog-cat-full-dataset/data/train/', transform=transform)
validation_dataset = datasets.ImageFolder('dog-cat-full-dataset/data/train/', transform=transfor

training_loader = torch.utils.data.DataLoader(training_dataset, batch_size=20, shuffle=True)
validation_loader = torch.utils.data.DataLoader(validation_dataset, batch_size = 20, shuffle=
```

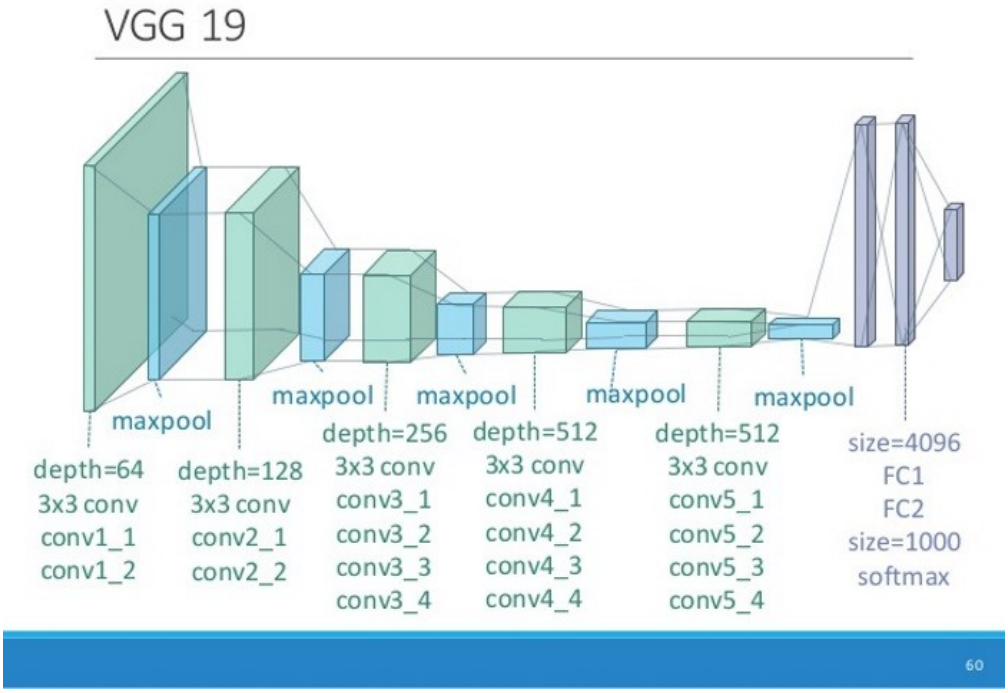
```
print(len(training_dataset))
print(len(validation_dataset))
```

```
20000
20000
```

```
def im_convert(tensor):
    image = tensor.cpu().clone().detach().numpy()
    image = image.transpose(1, 2, 0)
    image = image * np.array((0.5, 0.5, 0.5)) + np.array((0.5, 0.5, 0.5))
    image = image.clip(0, 1)
    return image

# !ls ./Face-Mask-Detection/dataset/

classes=('cats','dogs')

dataiter = iter(training_loader)
images,labels = dataiter.next()
fig = plt.figure(figsize=(25, 4))

for idx in np.arange(20):
    ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
    plt.imshow(im_convert(images[idx]))
    ax.set_title(classes[labels[idx].item()])
```

!!!注意類別順序，第一次將順序顛倒，導致最後step5判別錯誤，修改後就正確了!

Step 3: Build Model for training




```
model  =  models.vgg16(pretrained=True)
print(model)
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
```

```
      (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (29): ReLU(inplace=True)
      (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
    (classifier): Sequential(
      (0): Linear(in_features=25088, out_features=4096, bias=True)
      (1): ReLU(inplace=True)
      (2): Dropout(p=0.5, inplace=False)
      (3): Linear(in_features=4096, out_features=4096, bias=True)
      (4): ReLU(inplace=True)
      (5): Dropout(p=0.5, inplace=False)
      (6): Linear(in_features=4096, out_features=1000, bias=True)
    )
  )
```

```
# turn off gradient for all parameters in features extraction
for param in model.features.parameters():
    param.requires_grad = False

# modify last node from 1000 to 2
# import torch.nn as nn
n_inputs = model.classifier[6].in_features
last_layer = nn.Linear(n_inputs, len(classes))
model.classifier[6] = last_layer
model.to(device)
print(model)

print("output features=",model.classifier[6].out_features)
```

```
  VGG(
    (features): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
      (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU(inplace=True)
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (6): ReLU(inplace=True)
      (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (8): ReLU(inplace=True)
      (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (11): ReLU(inplace=True)
      (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (13): ReLU(inplace=True)
      (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (15): ReLU(inplace=True)
      (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (18): ReLU(inplace=True)
      (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (20): ReLU(inplace=True)
      (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (22): ReLU(inplace=True)
      (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (25): ReLU(inplace=True)
```

```
      (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (27): ReLU(inplace=True)
      (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (29): ReLU(inplace=True)
      (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
    (classifier): Sequential(
      (0): Linear(in_features=25088, out_features=4096, bias=True)
      (1): ReLU(inplace=True)
      (2): Dropout(p=0.5, inplace=False)
      (3): Linear(in_features=4096, out_features=4096, bias=True)
      (4): ReLU(inplace=True)
      (5): Dropout(p=0.5, inplace=False)
      (6): Linear(in_features=4096, out_features=2, bias=True)
    )
  )
  output features= 2
```

Step 4: Training Model

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = 0.0001)
```

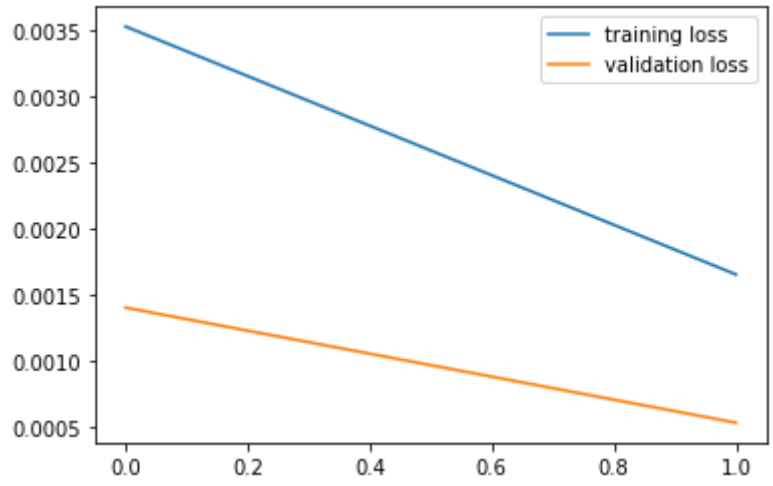
```
epochs = 2
running_loss_history = []
running_corrects_history = []
val_running_loss_history = []
val_running_corrects_history = []

for e in range(epochs):

    running_loss = 0.0
    running_corrects = 0.0
    val_running_loss = 0.0
    val_running_corrects = 0.0

    for inputs,labels in training_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        _, preds = torch.max(outputs, 1)
        running_loss += loss.item()
        running_corrects += torch.sum(preds == labels.data)

    else:
```

```
        with torch.no_grad():
            for val_inputs, val_labels in validation_loader:
                val_inputs = val_inputs.to(device)
                val_labels = val_labels.to(device)
                val_outputs = model(val_inputs)
                val_loss = criterion(val_outputs, val_labels)

                _, val_preds = torch.max(val_outputs, 1)
                val_running_loss += val_loss.item()
                val_running_corrects += torch.sum(val_preds == val_labels.data)

        epoch_loss = running_loss/len(training_loader.dataset)
        epoch_acc = running_corrects.float()/ len(training_loader.dataset)
        running_loss_history.append(epoch_loss)
        running_corrects_history.append(epoch_acc)

        val_epoch_loss = val_running_loss/len(validation_loader.dataset)
        val_epoch_acc = val_running_corrects.float()/ len(validation_loader.dataset)
        val_running_loss_history.append(val_epoch_loss)
        val_running_corrects_history.append(val_epoch_acc)
        print('epoch :', (e+1))
        print('training loss: {:.4f}, acc {:.4f} '.format(epoch_loss, epoch_acc.item()))
        print('validation loss: {:.4f}, validation acc {:.4f} '.format(val_epoch_loss, val_e
```

```
epoch : 1
training loss: 0.0035, acc 0.9753
validation loss: 0.0014, validation acc 0.9896
epoch : 2
training loss: 0.0017, acc 0.9893
validation loss: 0.0005, validation acc 0.9969
```

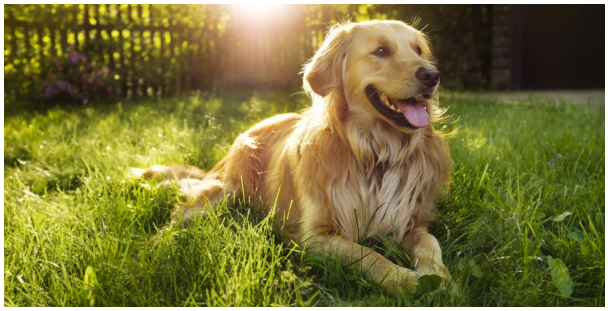
```
plt.plot(running_loss_history, label='training loss')
plt.plot(val_running_loss_history, label='validation loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fab1ebbe810>
```



Step 5 Testing

dog



cat

```
!pip3  install  pillow==4.0.0
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/s
Requirement already satisfied: pillow==4.0.0 in /usr/local/lib/python3.7/dist-packages (4.0.
Requirement already satisfied: olefile in /usr/local/lib/python3.7/dist-packages (from pillo
```

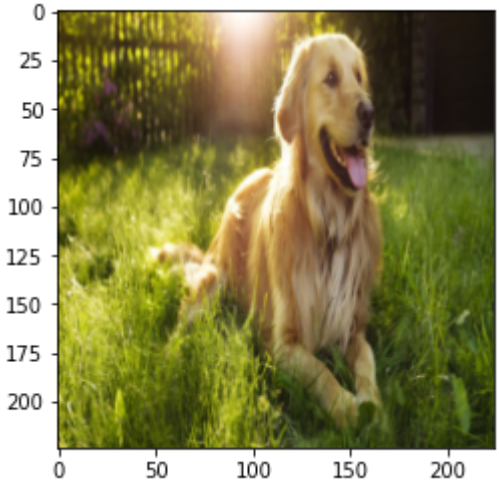
```
import  PIL.ImageOps
```

```
import  requests
from  PIL  import  Image

url  =  'https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/golden-retriever-royalty-free
#  url  ='https://i.epochtimes.com/assets/uploads/2021/08/id13156667-shutterstock_376153318-600x400
#  url='https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/golden-retriever-royalty-free-
response  =  requests.get(url,  stream  =  True)
img  =  Image.open(response.raw)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7fab32d3cd10>
```

```
img = transform(img)
plt.imshow(im_convert(img))
```

```
<matplotlib.image.AxesImage at 0x7fab32b55750>
```



```
image = img.to(device).unsqueeze(0)
output = model(image)
_, pred = torch.max(output, 1)
print(classes[pred.item()])
```

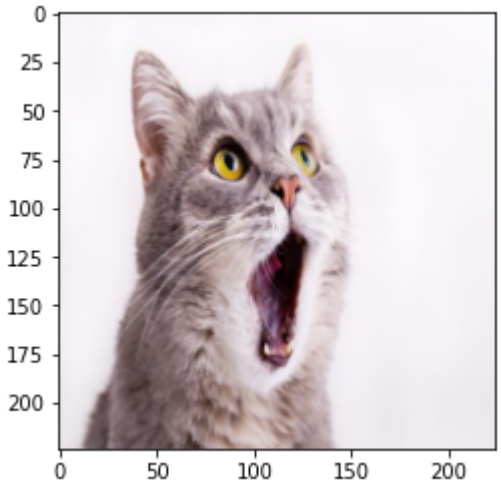
```
dogs
```

```
import requests
from PIL import Image

# url = 'https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/golden-retriever-royalty-f
url ='https://i.epochtimes.com/assets/uploads/2021/08/id13156667-shutterstock_376153318-600x400.jp
# url='https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/golden-retriever-royalty-free-
response = requests.get(url, stream = True)
img = Image.open(response.raw)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7fab32e935d0>
```

```
img = transform(img)
plt.imshow(im_convert(img))
```

```
<matplotlib.image.AxesImage at 0x7fab32d93b10>
```




```
image = img.to(device).unsqueeze(0)
output = model(image)
_, pred = torch.max(output, 1)
print(classes[pred.item()])
```

```
cats
```

✓ 0 秒 完成時間：晚上11:19