In [1]:

```
from pathlib import Path
import pandas as pd
import hashlib
from collections import OrderedDict
import torch
import numpy as np
import random
from dataclasses import dataclass
from torch.utils.data import Dataset
from pathlib import Path
from torchvision.transforms import *
import torch.nn as nn
from torchvision import models
from tqdm.notebook import tqdm
from collections import defaultdict
import os
from torchvision.transforms import transforms
from torch.optim import Adam, lr_scheduler
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler
from timeit import default_timer
from PIL import Image
from functools import reduce
from sklearn.metrics import accuracy_score


os.environ["CUDA_VISIBLE_DEVICES"] = "0,1"

class Benchmark(object):
    def __init__(self, msg, print=True):
        self.msg = msg
        self.print = print

    def print_elapsed(self, add_msg):
        t = default_timer() - self.start
        if self.print:
            print(f"{self.msg}, {add_msg}: {t:.2f} seconds")

    def __enter__(self):
        self.start = default_timer()
        if self.print:
            print(f"{self.msg}: begin")
        return self

    def __exit__(self, *args):
        t = default_timer() - self.start
        if self.print:
            print(f"{self.msg}: {t:.2f} seconds")
        self.time = t

def ensure_file(filepath):
    Path(os.path.dirname(filepath)).mkdir(parents=True, exist_ok=True)

def set_seed(seed):
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    random.seed(seed)

set_seed(1)
```

1. 数据文件生成

根据原始数据，生成规范的包含训练数据路径与样本其他信息的整体数据文件。

建议：

- 以csv格式生成
- 在样本数据和样本信息比较复杂的情况下，将样本数据与样本信息分别生成整体数据文件，之后再进行join
- 去除明显异常，对后续任务无意义的低质量数据
- 尽量多的保留原始数据与格式，减少任务相关的预处理，方便后续各种训练任务的生成

数据集：https://www.kaggle.com/iamsouravbanerjee/animal-image-dataset-90-different-animals

任务描述：5400 Animal Images in 90 different categories or classes

In [2]:

```
df = []
root_path = Path('data/animals')
for file in root_path.rglob('*.jpg'):
    label = file.parent.name
    dataset = file.parent.parent.name
    df.append({
        'image_path': str(file),
        'animal_type': label
    })
df = pd.DataFrame(df)
df.to_csv('data/all_data.csv', index=False)
df.head(2)
```

Out[2]:

	image_path	animal_type
0	data/animals/animals/antelope/02f4b3be2d.jpg	antelope
1	data/animals/animals/antelope/03d7fc0888.jpg	antelope

2. 训练数据处理

根据整体数据文件，提取其中样本、信息的子集，对其进行预处理后生成训练数据。

建议：

- 根据不同的列，使用不同的划分方法对数据集进行划分，如根据病人id/样本id，对原数据进行5折/train-valid-test划分
- 划分的方法最好是确定性算法，如hash
- 训练数据生成的过程中的随机成分，通过对各种seed的设置，保证数据生成过程的可重复性
- 进行标签映射，如果是二分类任务，默认是0代表正常，1代表异常

In [3]:

```
df = pd.read_csv('data/all_data.csv')

def to_dataset_mapping(ids, n_fold):
    result = {}
    for one_id in ids:
        result[one_id] = int(hashlib.sha256((str(one_id)).encode('utf-8')).
    return result

animal_types = sorted(df['animal_type'].unique())
animal_types_mapping = OrderedDict()
for index, animal in enumerate(animal_types):
    animal_types_mapping[animal] = index
print(animal_types_mapping)

df['label_animal_type'] = df['animal_type'].map(animal_types_mapping)
df['dataset'] = df['image_path'].map(to_dataset_mapping(df['image_path'].to
ensure_file('output/animal_classification/task/train_data.csv')
df.to_csv('output/animal_classification/task/train_data.csv', index=False)

df.head(2)
```

```
OrderedDict([('antelope', 0), ('badger', 1), ('bat', 2), ('bear', 3), ('bee
', 4), ('beetle', 5), ('bison', 6), ('boar', 7), ('butterfly', 8), ('cat',
9), ('caterpillar', 10), ('chimpanzee', 11), ('cockroach', 12), ('cow', 13)
, ('coyote', 14), ('crab', 15), ('crow', 16), ('deer', 17), ('dog', 18), ('
dolphin', 19), ('donkey', 20), ('dragonfly', 21), ('duck', 22), ('eagle', 2
3), ('elephant', 24), ('flamingo', 25), ('fly', 26), ('fox', 27), ('goat',
28), ('goldfish', 29), ('goose', 30), ('gorilla', 31), ('grasshopper', 32),
('hamster', 33), ('hare', 34), ('hedgehog', 35), ('hippopotamus', 36), ('ho
rnbill', 37), ('horse', 38), ('hummingbird', 39), ('hyena', 40), ('jellyfis
h', 41), ('kangaroo', 42), ('koala', 43), ('ladybugs', 44), ('leopard', 45)
, ('lion', 46), ('lizard', 47), ('lobster', 48), ('mosquito', 49), ('moth',
50), ('mouse', 51), ('octopus', 52), ('okapi', 53), ('orangutan', 54), ('ot
ter', 55), ('owl', 56), ('ox', 57), ('oyster', 58), ('panda', 59), ('parrot
', 60), ('pelecaniformes', 61), ('penguin', 62), ('pig', 63), ('pigeon', 64
), ('porcupine', 65), ('possum', 66), ('raccoon', 67), ('rat', 68), ('reind
eer', 69), ('rhinoceros', 70), ('sandpiper', 71), ('seahorse', 72), ('seal'
, 73), ('shark', 74), ('sheep', 75), ('snake', 76), ('sparrow', 77), ('squi
d', 78), ('squirrel', 79), ('starfish', 80), ('swan', 81), ('tiger', 82), (
'turkey', 83), ('turtle', 84), ('whale', 85), ('wolf', 86), ('wombat', 87),
('woodpecker', 88), ('zebra', 89)])
```

Out[3]:

	image_path	animal_type	label_animal_type	dataset
0	data/animals/animals/antelope/02f4b3be2d.jpg	antelope	0	1
1	data/animals/animals/antelope/03d7fc0888.jpg	antelope	0	4

3. dataset预处理

继承dataset来进行数据的读取与预处理

建议：

- 使用DataFrame进行数据的管理
- Dataset类似于list，可以通过len()获得长度，以及通过index进行访问
- 在读取图片时，先读取图片，后使用transforms进行图片增强

In [4]:

```
@dataclass
class AnimalDataset(Dataset):
    data: pd.DataFrame
    mode: str
    config: dict

    def __post_init__(self):
        self.image_root = self.config['image_root']
        self.label_col = self.config['label_col']
        self.reshape_size = self.config['reshape_size']
        self.crop_size = self.config['crop_size']

        self.trans = {
            'train': transforms.Compose([
                transforms.RandomResizedCrop((self.reshape_size, self.resha
                transforms.ColorJitter(brightness=0.3),
                transforms.RandomRotation(degrees=10),
                transforms.RandomHorizontalFlip(),
                transforms.RandomVerticalFlip(),
                transforms.RandomCrop((self.crop_size, self.crop_size)),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406],
                                     [0.229, 0.224, 0.225])
            ]),
            'valid': transforms.Compose([
                transforms.Resize(self.reshape_size),
                transforms.CenterCrop((self.crop_size, self.crop_size)),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406],
                                     [0.229, 0.224, 0.225])
            ]),
            'test': transforms.Compose([
                transforms.Resize(self.reshape_size),
                transforms.CenterCrop((self.crop_size, self.crop_size)),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406],
                                     [0.229, 0.224, 0.225])
            ])
        }

    def read_image(self, path):
        img = Image.open(path).convert('RGB')
        img = self.trans[self.mode](img)
        return img

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

    def __getitem__(self, idx):
        img = self.read_image(str(self.image_root / self.data.loc[self.data
        result = {
            'image': img,
            'image_path': str(self.image_root / self.data.loc[self.data.inc
            'label': torch.tensor(self.data.loc[self.data.index[idx], self.
        }
        return result
```

4. 模型编写

根据任务定义模型结构，这里只展示基于resnet的迁移学习。

建议：

- 在使用DataParallel时，将loss在model中计算可以减少gpu显存不平均情况

In [5]:

```
class AnimalModel(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.config = config
        self.arch = self.config['arch']
        self.n_class = self.config['n_class']
        self.loss_fn = nn.CrossEntropyLoss(reduction='none')

        if self.arch == 'resnet18':
            self.net = models.resnet18(pretrained=True)
            n_out = 512
        elif self.arch == 'resnet50':
            self.net = models.resnet50(pretrained=True)
            n_out = 2048

        self.net.fc = nn.Sequential(
            nn.Linear(n_out, 256),
            nn.ReLU(),
            nn.Linear(256, self.n_class)
        )

    def backbone_parameters(self):
        return map(lambda kv: kv[1], filter(lambda kv: not kv[0].startswith

    def head_parameters(self):
        return self.net.fc.parameters()

    def _forward(self, img):
        return self.net(img)

    def forward(self, img, label=None):
        y = self.net(img)
        loss = None
        if label is not None:
            loss = self.loss_fn(y, label)
        return y, loss
```

5. 训练与测试

在训练的过程中，完成训练、验证、模型指标计算、模型选择以及测试的过程。

建议：

- 多阅读其他人的训练代码，比如fastai

In [6]:

```
def train_loss(records):
    losses = torch.cat(records['train-loss-list'])
    return 'train-loss', losses.mean()


def valid_loss(records):
    losses = torch.cat(records['valid-loss-list'])
    return 'valid-loss', losses.mean()


def accuracy(records):
    pred = torch.cat(records['valid-y_pred-list']).argmax(dim=-1).numpy()
    true = torch.cat(records['valid-y_true-list']).numpy()
    try:
        return f'ACC', accuracy_score(true, pred)
    except Exception as e:
        print(e)
        return f'Metric', float('nan')


def clear_records_epoch(records):
    for key in ['train-y_true-list', 'train-y_pred-list', 'train-loss-list'
                'valid-y_true-list', 'valid-y_pred-list', 'valid-loss-list'
        records[key] = []


class ModelSaver:
    def __init__(self, model: nn.Module, model_folder, records, key, compar
        self.model = model
        self.model_folder = model_folder
        self.records = records
        self.key = key
        self.compare = compare
        self.map_location = map_location
        if self.compare == max:
            self.records[f'best_{key}'] = -float('inf')
        else:
            self.records[f'best_{key}'] = float('inf')

    def step(self):
        if self.compare(self.records[self.key], self.records[f'best_{self.k
            self.records[f'best_{self.key}'] = self.records[self.key]
            print(f'Save better model, {self.key}={self.records[self.key]:.
            ensure_file(f'{self.model_folder}/best_{self.key}.pth')
            torch.save(self.model.state_dict(), f'{self.model_folder}/best_

    def load(self):
        self.model.load_state_dict(torch.load(f'{self.model_folder}/best_{s
```

In [7]:

```
def run_dp(config, device=torch.device('cuda')):
    cpu = torch.device('cpu')

     # task name
    task = config['task']

    # training params
```

```python
    batch_size = config['batch_size']
    num_train_epochs = config['num_train_epochs']
    parallel = config['parallel']
    dataset_class = config['dataset_class']
    model_class = config['model_class']

    # task params
    task_file_path = config['task_file_path']
    output_path = config['output_path']
    n_class = config['n_class']
    label_col = config['label_col']
    fold = config['fold']

    # metric and save params
    processors = config['processors']
    savers_init = config['savers_init']

    df = pd.read_csv(task_file_path, low_memory=False)
    results = defaultdict(list)

    print(f'task={task}, fold={fold}')
    name = f"{task}-fold{fold}"

    train_df = df[df['dataset'].isin([(fold + 1) % 5,  (fold + 2) % 5, (fol
    valid_df = df[df['dataset'] == (fold + 4) % 5].copy()
    test_df = df[df['dataset'] == fold].copy()

    train_ds = dataset_class(train_df, 'train', config)
    valid_ds = dataset_class(valid_df, 'valid', config)
    test_ds = dataset_class(test_df, 'test', config)

    train_dl = DataLoader(train_ds, sampler=RandomSampler(train_ds), batch_
    valid_dl = DataLoader(valid_ds, sampler=SequentialSampler(valid_ds), ba
    test_dl = DataLoader(test_ds, sampler=SequentialSampler(test_ds), batch

    model = model_class(config)
    model = model.to(device)

    if parallel:
        model_for_train = nn.DataParallel(model)
    else:
        model_for_train = model

    records = defaultdict(list)
    savers = []
    for saver_init in savers_init:
        savers.append(ModelSaver(model, f'models/{name}', records, saver_in

    optimizer = Adam([
        {'params': model.backbone_parameters(), 'lr': 1e-4},
        {'params': model.head_parameters(), 'lr': 1e-3}
    ], weight_decay=1e-5)

    scheduler = lr_scheduler.CosineAnnealingLR(optimizer, T_max=32)
    for epoch in range(num_train_epochs):
        with Benchmark(f'Epoch {epoch}'):
            records['epoch'] = epoch
            clear_records_epoch(records)
            # train
            model_for_train.train()
            with tqdm(train_dl, leave=False) as t:
                for batch in t:
                    img = batch['image'].to(device)
                    label = batch['label'].to(device)
                    optimizer.zero_grad()
                    y, loss = model_for_train(img, label)
                    loss_bp = torch.mean(loss)
                    loss_bp.backward()
                    optimizer.step()
                    t.set_postfix(loss=float(loss_bp))
                    records['train-loss-list'].append(loss.detach().cpu())
            # valid
            model_for_train.eval()
            with torch.no_grad():
                with tqdm(valid_dl, leave=False) as t:
                    for batch in t:
                        img = batch['image'].to(device)
                        label = batch['label'].to(device)
                        optimizer.zero_grad()
                        y, loss = model_for_train(img, label)
                        records['valid-loss-list'].append(loss.detach().cpu
                        records['valid-y_true-list'].append(label.detach().
                        records['valid-y_pred-list'].append(y.detach().cpu(
            to_print = []
            for processor in processors:
                key, value = processor(records)
                records[key] = value
                to_print.append(f'{key}={value:.4f}')
            print(f'Epoch {epoch}: ' + ', '.join(to_print))

            scheduler.step()
            for saver in savers:
                saver.step()
    # test
    for saver in savers:
        saver.load()

        test_df_fold = test_df.copy()
        image_paths = []
        preds = []
        model_for_train.eval()
        with torch.no_grad():
            with tqdm(test_dl, leave=False) as t:
                for batch in t:
                    img = batch['image'].to(device)
                    image_path = batch['image_path']
                    y, _ = model_for_train(img)
                    y = torch.softmax(y, dim=-1)
                    preds.append(y)
                    image_paths.append(image_path)

        preds = torch.cat(preds, dim=0).cpu().numpy()
        image_paths = reduce(lambda x, y: x + y, image_paths)
        preds_df = pd.DataFrame(image_paths, columns=['image_path'])
        for i in range(n_class):
            preds_df[f'{label_col}_prob_{i}'] = preds[:, i]
        test_df_fold = test_df_fold.merge(preds_df, on='image_path')
        ensure_file(f'{output_path}/{saver.key}/preds_fold_{fold}.csv')
        test_df_fold.to_csv(f'{output_path}/{saver.key}/preds_fold_{fold}.c
    model_for_train.to(cpu)
    model.to(cpu)

config = {
    'task': 'animal_classification',
    'task_file_path': 'output/animal_classification/task/train_data.csv',
    'output_path': 'output/animal_classification/results',
    'label_col': 'label_animal_type',
    'n_class': 90,
    'processors': [train_loss, valid_loss, accuracy],
    'savers_init': [('valid-loss', min), ('ACC', max)],
    'fold': 0,

    'batch_size': 320,
    'num_train_epochs': 20,
    'parallel': True,
    'dataset_class': AnimalDataset,
    'model_class': AnimalModel,

    'image_root': Path('.'),
    'reshape_size': 300,
    'crop_size': 224,

    'arch': 'resnet50',
}
run_dp(config, device=torch.device('cuda'))
```

```
task=animal_classification, fold=0
Epoch 0: begin
```

```
/home/embryosusr/anaconda3/envs/embryo/lib/python3.8/site-packages/torch/nn
/functional.py:718: UserWarning: Named tensors and all their associated API
s are an experimental feature and subject to change. Please do not use them
for anything important until they are released as stable. (Triggered intern
ally at  /opt/conda/conda-bld/pytorch_1623448234945/work/c10/core/TensorImp
l.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ce
il_mode)
```

```
Epoch 0: train-loss=4.0656, valid-loss=2.4848, ACC=0.4491
Save better model, valid-loss=2.4848
Save better model, ACC=0.4491
Epoch 0: 35.10 seconds
Epoch 1: begin


Epoch 1: train-loss=2.4492, valid-loss=1.3422, ACC=0.6575
Save better model, valid-loss=1.3422
Save better model, ACC=0.6575
Epoch 1: 29.66 seconds
Epoch 2: begin


Epoch 2: train-loss=1.5150, valid-loss=0.9398, ACC=0.7509
Save better model, valid-loss=0.9398
Save better model, ACC=0.7509
Epoch 2: 29.30 seconds
Epoch 3: begin
Epoch 3: train-loss=1.0919, valid-loss=0.8261, ACC=0.7613
Save better model, valid-loss=0.8261
Save better model, ACC=0.7613
Epoch 3: 29.26 seconds
Epoch 4: begin

Epoch 4: train-loss=0.8824, valid-loss=0.6995, ACC=0.7925
Save better model, valid-loss=0.6995
Save better model, ACC=0.7925
Epoch 4: 29.55 seconds
Epoch 5: begin

Epoch 5: train-loss=0.7143, valid-loss=0.6350, ACC=0.8057
Save better model, valid-loss=0.6350
Save better model, ACC=0.8057
Epoch 5: 29.65 seconds
Epoch 6: begin

Epoch 6: train-loss=0.6272, valid-loss=0.5965, ACC=0.8255
Save better model, valid-loss=0.5965
Save better model, ACC=0.8255
Epoch 6: 31.35 seconds
Epoch 7: begin

Epoch 7: train-loss=0.5544, valid-loss=0.5489, ACC=0.8415
Save better model, valid-loss=0.5489
Save better model, ACC=0.8415
Epoch 7: 29.33 seconds
Epoch 8: begin

Epoch 8: train-loss=0.4867, valid-loss=0.5426, ACC=0.8575
Save better model, valid-loss=0.5426
Save better model, ACC=0.8575
Epoch 8: 30.24 seconds
Epoch 9: begin

Epoch 9: train-loss=0.4250, valid-loss=0.5050, ACC=0.8613
Save better model, valid-loss=0.5050
Save better model, ACC=0.8613
Epoch 9: 29.86 seconds
Epoch 10: begin

Epoch 10: train-loss=0.3745, valid-loss=0.5140, ACC=0.8519
Epoch 10: 28.41 seconds
Epoch 11: begin

Epoch 11: train-loss=0.3439, valid-loss=0.5068, ACC=0.8623
Save better model, ACC=0.8623
Epoch 11: 29.00 seconds
Epoch 12: begin
Epoch 12: train-loss=0.3526, valid-loss=0.5345, ACC=0.8387
Epoch 12: 29.97 seconds
Epoch 13: begin

Epoch 13: train-loss=0.3120, valid-loss=0.5071, ACC=0.8613
Epoch 13: 28.12 seconds
Epoch 14: begin

Epoch 14: train-loss=0.2971, valid-loss=0.5150, ACC=0.8557
Epoch 14: 28.89 seconds
Epoch 15: begin

Epoch 15: train-loss=0.2729, valid-loss=0.4908, ACC=0.8623
Save better model, valid-loss=0.4908
Save better model, ACC=0.8623
Epoch 15: 31.33 seconds
Epoch 16: begin

Epoch 16: train-loss=0.2669, valid-loss=0.4994, ACC=0.8708
Save better model, ACC=0.8708
Epoch 16: 30.09 seconds
Epoch 17: begin

Epoch 17: train-loss=0.2312, valid-loss=0.5065, ACC=0.8689
Epoch 17: 28.75 seconds
Epoch 18: begin

Epoch 18: train-loss=0.2340, valid-loss=0.5200, ACC=0.8519
Epoch 18: 29.91 seconds
Epoch 19: begin

Epoch 19: train-loss=0.2230, valid-loss=0.4876, ACC=0.8755
Save better model, valid-loss=0.4876
Save better model, ACC=0.8755
Epoch 19: 30.22 seconds
```