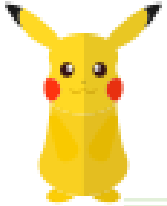
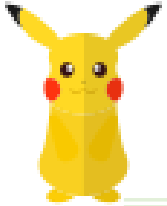
Pokemon Classification

2조
201501709 최휘준 201704034 민문기
201704123 추교원 201984007 김지윤



Contents



동기

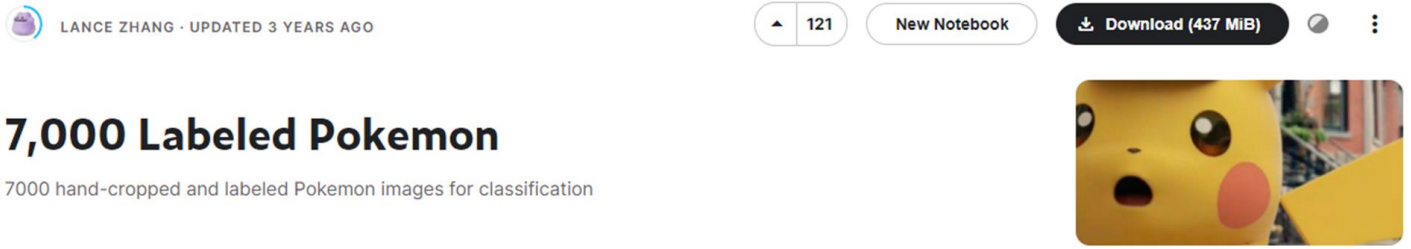
우연히 티셔츠에 그려져 있는 포켓몬이 궁금해서 검색하다가 사진만 넣어도 무슨 포켓몬인지 알려주는 것을 만들어보면 재미있을 것 같아 이 프로젝트를 기획했습니다.

목적

포켓몬 이미지 데이터 분석을 통해 포켓몬을 분류하고 어떤 포켓몬인지 알려주는 것이 이번 프로젝트의 목적입니다.

데이터 전처리

Classification을 위한 데이터 수집
사용할 이미지 데이터는 **7,000 Labeled Pokemon**으로 kaggle에서 이미지를 받음
이미지는 총 6837개로 이루어져 있고 총 150가지 종류로 분류 되어있습니다.
https://www.kaggle.com/datasets/lantian773030/pokemonclassification



개발환경은 구글의 colab을 이용하여 개발하였습니다.

colab

데이터 전처리

데이터 로드하기

우선 데이터를 다루기 전에 데이터를 로드.

데이터는 우선 구글 드라이브에 업로드하고 colab으로 받아왔습니다.

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

```
[ ]  #Import the libraries
     import zipfile
     import os
     zip_ref = zipfile.ZipFile('/content/drive/MyDrive/archive (2)_.zip', 'r') #Opens the zip file in read mode
     zip_ref.extractall('/tmp/dataset') #Extracts the files into the /tmp folder
     zip_ref.close()
```

데이터 전처리

데이터 라벨링
데이터를 본격적으로 사용하기에 앞서 분류할 각 이미지데이터들에 대해서 라벨링을 진행했습니다.

```
[ ]  data_dir = '/content/drive/MyDrive/preprocessing/test'

     classes = os.listdir(data_dir)
     classes.sort()
     print(classes)
     print(len(classes))

     ['Abra', 'Aerodactyl', 'Alakazam', 'Alolan Sandslash', 'Arbok', 'Arcanine', 'Articuno', 'Beedrill',
     150
```

각 파일들의 이름을 리스트형태로 만들어 라벨링을 진행하였습니다.
데이터의 종류는 150가지로 보시는 봐와 같이 라벨링이 잘 되었습니다.

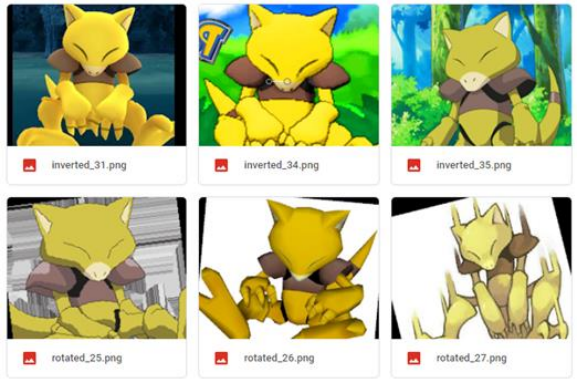
데이터 전처리

Train ,Test, Val 데이터로 분류하기 전에 훈련시키기에 데이터의 양이 부족하기 때문에 데이터의 양을 늘리는 작업을 진행했습니다.

로드한 이미지 파일의 경로를 받아와 이미지를 읽어드립니다.

그후 이미지를 좌우반전 하거나 조금씩 회전을 주어 이미지를 받아온 경로에 저장을 합니다.

이를 통해 다음과 같이 이미지의 갯수를 늘렸습니다.




```python
import random
import numpy as np
import os
import cv2
import glob
from PIL import Image
import PIL.ImageOps

#다음 변수를 수정하여 새로 만들 이미지 갯수를 정합니다.
num_augmented_images = 50

file_path = '/tmp/dataset/PokemonData/'
dir_names = os.listdir(file_path)
total_origin_dir_num = len(dir_names)
print(len(dir_names))
augment_cnt = 1


for i in range(0, total_origin_dir_num):
    dir_name = dir_names[i]
    file_names=os.listdir(file_path+dir_name)
    total_origin_image_num=len(file_names)          #폴더당 이미지 개수
    for k in range(0, total_origin_image_num):      #폴더의 이미지만큼 반복
      change_picture_index = k
      file_name = file_names[change_picture_index]  #전환할 이미지
      origin_image_path = '/tmp/dataset/PokemonData/'+dir_name +'/'+file_name
      print(origin_image_path)
      image = Image.open(origin_image_path)
      random_augment = random.randrange(1,3)
      save_image_path='/tmp/dataset/PokemonData/'+dir_name+'/'
      if(random_augment == 1):
          #이미지 좌우 반전
          print("invert")
          inverted_image = image.transpose(Image.FLIP_LEFT_RIGHT)
          inverted_image.save(save_image_path + 'inverted_' + str(augment_cnt) + '.png')
      elif(random_augment == 2):
          #이미지 기울이기
          print("rotate")
          rotated_image = image.rotate(random.randrange(-20, 20))
          rotated_image.save(save_image_path + 'rotated_' + str(augment_cnt) + '.png')

      augment_cnt += 1  #파일 개수
    augment_cnt=1#파일 개수 초기화
```

데이터 전처리

Split-folders라는 오픈 라이브러리를 이용해서 데이터를 분류합니다.
데이터는 Train , Test , val 3가지로 분류해서 드라이브에 저장을 해줍니다.

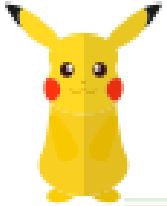
```
pip install split-folders
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
```

```
[ ]  #저장 경로를 drive로 해서 최초 1번만 실행
     splitfolders.ratio("/tmp/dataset/PokemonData", output="/content/drive/MyDrive/preprocessing",seed=1337, ratio=(.8, .18, .02), group_prefix=None, move=False)
```

```
Copying files: 13642 files [01:50, 123.72 files/s]
```

이 단계까지 데이터의 양을 늘리고 훈련시키기 위한 데이터 분류는 끝이 났습니다.



데이터 전처리

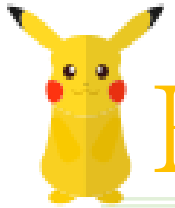
이제 저장한 이미지를 훈련시키기 위해 적당하게 변환합니다 크기는 224*224 픽셀 사이즈로 변환하고 정규화를 진행합니다.

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

     #미리 드라이브에 저장해놓은 데이터를 로드
     traindata=datasets.ImageFolder(root='/content/drive/MyDrive/preprocessing/train',transform=tf)
     testdata=datasets.ImageFolder(root='/content/drive/MyDrive/preprocessing/val',transform=tf)
```

```
[ ]  batch_size=32
     trainloader=DataLoader(traindata,batch_size=batch_size, shuffle=True)
     testloader=DataLoader(testdata,batch_size=batch_size, shuffle=True)
     train_images,train_labels=iter(trainloader).next()
     print(train_labels)
```

```
tensor([ 62, 148,  70,  13, 115,  13, 104, 139, 108,   4,  84,  84, 107,  63,
        126, 121, 131,   9, 137,  30, 119,  97, 126,  62,  31,  54,  82,  25,
        112,  86,  83,  72])
```

그리고 이 데이터를 로더를 통해서 불러와 훈련에 사용할 데이터를 최종적으로 완성했습니다.



ResNet50

Pre-trained된 resnet50모델을 불러옴.
Pre-trained되었기 때문에 가중치를 freeze시킴.

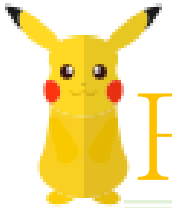
밑바닥에서부터 모델을 쌓아올리는 대신에 이미 학습되어있는 패턴을 활용해서 적용시키므로 적은 데이터셋으로도 좋은 결과를 낼 수 있다는 장점이 있어 적용.

포켓몬이 150종류이기 때문에 num_class = 150으로 설정.

손실함수는 CrossEntropy로 설정

Optimizer는 Stochastic Gradient Descent로 설정

```
from torchvision import models
import torch.nn as nn
import torch.optim as optim
import torch

resnet50 = models.resnet50(pretrained = True)

for param in resnet50.parameters():   #가중치 freeze
  param.requires_grad = False

num_classes = 150
num_ftrs = resnet50.fc.in_features

resnet50.fc = nn.Linear(num_ftrs, num_classes)

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
resnet50.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(resnet50.parameters(), lr=0.001,momentum=0.9)
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer_ft, step_size =7, gamma=0.1)
```



ResNet50

Tqdm을 이용해 데이터의 학습률을 가시성 좋게 볼 수 있음.

Predict(이하 pred)는 손실함수(loss_fn)에 예측한 pred와 정답인 y를 넣어 계산.

Optimizer.zero_grad() : pytorch에서는 gradien값들을 추후에 backward를 해줄때 계속 더해주기 때문에 backpropagation을 하기 전에 gradien를 zero로 만들어주고 시작해야함.

Loss.backward()로 예측손실 역전파 계산, 매개변수 저장.
Optimizer.step()으로 optimizer에게 loss functio을 효율적으로 최소화 할 수 있게함.

```
def train(dataloader, model, loss_fn, optimizer):
    model.train()
    pbar = tqdm(enumerate(dataloader), total = len(dataloader)) #
    learning_loss = 0
    sum_loss = 0

    for batch, (X, y) in pbar:
        X, y = X.to(device), y.to(device)

        pred = model(X)
        loss = loss_fn(pred, y)

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

        sum_loss = sum_loss + loss.item() #
        learning_loss = sum_loss / (batch + 1)

        description = f'train loss : {round(learning_loss, 4)}'
        pbar.set_description(description)
```

ResNet50

Test함수에서는 learning_accurac와 learning_loss를 구해줌.

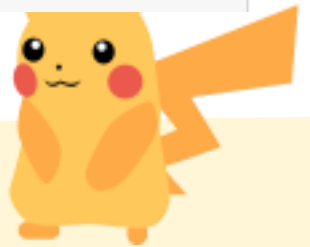
일치한 개수를 batch_size로 나누어 맞은 확률을 구하고 그 확률을 더한 것을 (batch + 1)로 나눔으로서 epoch의 평균 accuracy를 구해줌.

```
def test(dataloader, model, loss_fn):
    model.eval()  #
    pbar = tqdm(enumerate(dataloader), total = len(dataloader)) #
    learning_loss = 0
    sum_loss = 0
    learning_acc = 0
    sum_acc = 0
    correct = 0

    for batch, (X, y) in pbar:
      X, y = X.to(device), y.to(device)

      pred = model(X)
      loss = loss_fn(pred, y)
      correct = (pred.argmax(1)==y).type(torch.float).sum().item()
      acc = correct / batch_size
      sum_acc += acc
      learning_acc = sum_acc / (batch + 1)
      sum_loss = sum_loss + loss.item() #
      learning_loss = sum_loss / (batch + 1)

      description = f'test loss : {learning_loss}  || test acc : {learning_acc}'
      pbar.set_description(description)

    return learning_acc
```

ResNet50

Epoch를 20으로 설정하고 best_accurcy를 저장하기 위한 변수를 하나 설정해 줌.
학습시키고 테스트 할 때 가장 높은 accuracy를 저장함.

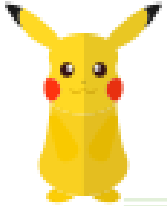
```
def test(dataloader, model, loss_fn):
    model.eval()  #
    pbar = tqdm(enumerate(dataloader), total = len(dataloader)) #
    learning_loss = 0
    sum_loss = 0
    learning_acc = 0
    sum_acc = 0
    correct = 0

    for batch, (X, y) in pbar:
      X, y = X.to(device), y.to(device)

      pred = model(X)
      loss = loss_fn(pred, y)
      correct = (pred.argmax(1)==y).type(torch.float).sum().item()
      acc = correct / batch_size
      sum_acc += acc
      learning_acc = sum_acc / (batch + 1)
      sum_loss = sum_loss + loss.item() #
      learning_loss = sum_loss / (batch + 1)

      description = f'test loss : {learning_loss}  || test acc : {learning_acc}'
      pbar.set_description(description)

    return learning_acc
```



ResNet50

테스트 결과

```
Epoch 13
-------------------------------
train loss : 0.3036: 100%|██████████| 340/340 [02:44<00:00,  2.07it/s]
test loss : 0.5202274831136068  || test acc : 0.8991666666666667: 100%|██████████| 75/75 [00:37<00:00,  1.99it/s]
Epoch 14
-------------------------------
train loss : 0.3013: 100%|██████████| 340/340 [02:43<00:00,  2.07it/s]
test loss : 0.535297878185908  || test acc : 0.8883333333333333: 100%|██████████| 75/75 [00:38<00:00,  1.97it/s]
Epoch 15
-------------------------------
train loss : 0.2963: 100%|██████████| 340/340 [02:43<00:00,  2.09it/s]
test loss : 0.5423712788025538  || test acc : 0.8891666666666667: 100%|██████████| 75/75 [00:37<00:00,  1.99it/s]
Epoch 16
-------------------------------
train loss : 0.2865: 100%|██████████| 340/340 [02:44<00:00,  2.07it/s]
test loss : 0.531057418982188  || test acc : 0.89125: 100%|██████████| 75/75 [00:37<00:00,  2.00it/s]
Epoch 17
-------------------------------
train loss : 0.2848: 100%|██████████| 340/340 [02:42<00:00,  2.09it/s]
test loss : 0.5068185339371364  || test acc : 0.8991666666666667: 100%|██████████| 75/75 [00:38<00:00,  1.94it/s]
Epoch 18
-------------------------------
train loss : 0.28: 100%|██████████| 340/340 [02:42<00:00,  2.09it/s]
test loss : 0.490044325987498  || test acc : 0.905: 100%|██████████| 75/75 [00:37<00:00,  1.99it/s]
Epoch 19
-------------------------------
train loss : 0.2756: 100%|██████████| 340/340 [02:44<00:00,  2.07it/s]
test loss : 0.4904035504659O167  || test acc : 0.9: 100%|██████████| 75/75 [00:37<00:00,  1.99it/s]
Epoch 20
-------------------------------
train loss : 0.2702: 100%|██████████| 340/340 [02:42<00:00,  2.09it/s]
test loss : 0.5138180363178253  || test acc : 0.8929166666666667: 100%|██████████| 75/75 [00:38<00:00,  1.94it/s]
```



Pokemon 판별

구글드라이브 연결

```
from google.colab import drive
drive.mount('/content/drive')
```

cvlib 설치

```
!pip install cvlib
```

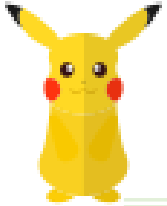
Pokemon 판별

파라미터를 불러옴

Resnet50 모델을 불러옴

Pre-trained 되었기에 가중치를 freeze 시킴

```
import torch
from torchvision import models
import torch.nn as nn

path = '/content/drive/MyDrive/ColabNotebooks/pokemonmodel_0.905.pt'

resnet50 = models.resnet50(pretrained = True)
for param in resnet50.parameters():   #가중치 freeze
  param.requires_grad = False
```



Pokemon 판별

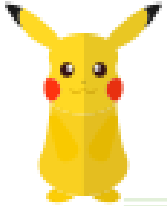
포켓몬이 150종류이기 때문에
num_classes =150 임

nn.Linear 레이어의 입력 개수(=num_ftrs)는 기존 Linear 레이어인 fc 레이어의 in_feature 속성값을 활용해서 알아내고, 출력 개수는 우리가 분류하려는 문제에 맞게 num_classes = 150 으로 지정하면 된다.

```
num_classes = 150
num_ftrs = resnet50.fc.in_features
resnet50.fc = nn.Linear(num_ftrs, num_classes)
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
resnet50.to(device)



resnet50.load_state_dict(torch.load(path))
resnet50.eval()  #Dropout, Batchnorm 등의 기능 비활성화
```

GPU or CPU 설정

State_dict 를 이용해 불러오는 부분이다.
이때 `.eval()` 을 호출하여 드롭아웃 및 배치 정규화를 평가 모드로 설정하여야 함. 이것을 하지 않으면 추론 결과가 일관성 없게 출력



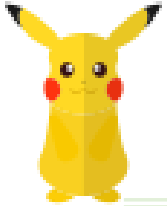
Pokemon 판별

이미지 파일을 불러오기 위해 cv2를 import
테스트를 위해 sample 포켓몬 사진을 업로드 한다.
샘플 이미지를 224*224픽셀 사이즈로 변환하고 정규화를 진행함
Resnet으로 이미지를 판별하고 제일 높은 확률을 출력

```
import cv2
import cvlib as cv
from google.colab.patches import cv2_imshow
import torchvision.transforms as transforms
import numpy as np
from PIL import Image
import os

img = cv2.imread('sample.png')
cv2_imshow(img)
img = Image.fromarray(img)

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

```
img = tf(img).unsqueeze(0)  #resnet50에 넣기 위해 4D로 변환 후 모델에 넣기위해 transform
img = img.to(device)
pred = resnet50(img)
pokemon = pred.argmax(1)
pokemon = int(pokemon)

classes = os.listdir('/content/drive/MyDrive/Colab Notebooks/preprocessing/test')
classes.sort()
print(classes[pokemon])
```

Pokemon 판별결과



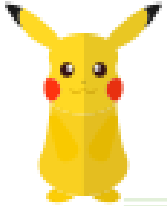
Pokemon 판별결과

133 이브이

Pokémon

Wartortle

Psyduck



소감

Pre-trained된 ResNet50모델을 가져와서 학습시키고, 이미지를 회전시키고 확대시켜 데이터를 늘렸음에도 불구하고 데이터의 양이 부족해서 정확도가 높게 나오지 못한 것이 아쉬웠다.

사진만 넣어도 분류를 해준다는 것이 신기했고 재미있었다.

빅데이터 수업을 들어서 즐거웠고 기본 소양을 쌓을 수 있어 즐거웠다.

ResNet에 대해 알 수 있어서 좋았고 포켓몬이 분류되는 결과를 보니 뿌듯했다.

Any Questions?

Thank You