데이터 사이언스를 위한 소프트웨어 플랫폼
Final Project

2021-27823 강지원

● 환경 구축 및 실행 방법

- python 3.7.5
- 사용 library

numpy==1.20.2, torch==1.8.0, torchvision==0.9.0 ,argparse==1.1 ,natsort==7.1.1 ,opencv-python==4.5.2.52 ,cudatoolkit==11.1

- 실행 방법

1) conda install pytorch==1.8.0 torchvision==0.9.0 cudatoolkit=11.1 -c pytorch -c nvidia

2) pip3 install -r requirements.txt

requirements.txt으로 cudatoolkit의 설치가 진행되지 않아 1)의 명령어를 통해 cuda, pytorch, torchvision을 설치한 후 나머지 모듈을 requirements.txt을 통해 설치하였다. cuda를 포함한 모든 모듈을 한 번에 설치할 수 있을 것으로 예상되는 파일인 requirements_full도 첨부하였다.

3) sbatch run.sh

4) python3 evaluate.py [test.dir] [result.txt]

1. Task 1: 공개된 데이터셋을 사용한 RPS classifier 구현

1) 코드 변경 사항

- Directory 이름 변경

기존의 dataset의 fist, palm, swing을 rock, scissors, paper로 변경하였다.

```
self.rock_dir = os.path.join(data_dir, 'rock/')
self.paper_dir = os.path.join(data_dir, 'paper/')
self.scissors_dir = os.path.join(data_dir, 'scissors/')
```

- 이미지 사이즈 변경

기존 모델의 input으로 (89, 100) 크기가 필요하므로 프로젝트에서 사용하는 (300, 300) 이미지를 torch.nn.functional 의 interpolation 함수를 활용하였다. 기존의 ToTensor() 클래스에 82번 째 줄을 추가하였다.

```
71  class ToTensor(object):
72      def __call__(self, data):
73          label, input = data['label'], data['input']
74
75          input_tensor = torch.empty(len(input), 300, 300)
76          label_tensor = torch.empty(len(input))
```

```
    for i in range(len(input)):
      input[i] = input[i].transpose((2, 0, 1)).astype(np.float32)
      input_tensor[i] = torch.from_numpy(input[i])
      label_tensor[i] = torch.from_numpy(label[i])
    input_tensor = torch.unsqueeze(input_tensor, 1)
    input_tensor = torch.nn.functional.interpolate(input_tensor, (89, 100), mode='bicubic')
    data = {'label': label_tensor.long(), 'input': input_tensor}

    return data
```

- 시간 측정 코드 추가

코드 실행 시간을 측정하기 위해서 main.py에 해당 코드를 추가하였다.

```
def main():
  start = time.time()
  args = parse_args()
  train(args)
  print("time :", time.time() - start)
```

2) 결과

총 5번의 실험을 통해 구한 best validation accuracy는 다음과 같다.

	1	2	3	4	5	Average
Accuracy(%)	0.9247	0.9220	0.9032	0.9059	0.9121	0.9136

2. Task 3: SPDS-RPS 챌린지

1) 코드 변경 사항

- torchvision 모델 사용

```
from torchvision.models import mobilenet_v2
```

```
model = mobilenet_v2()
model.features[0][0] = nn.Conv2d(1, 32, kernel_size=3, stride=2, padding=1, bias=False)
model.classifier[1] = torch.nn.Linear(in_features=model.classifier[1].in_features, out_features=3)
```

torchvision.models 의 mobilenet_v2를 사용하였다. spds 챌린지 상황에 맞추기 위하여 첫 conv layer와 마지막 fc layer의 parameter를 변경하였다.

- Training set augmentation & processing

```
for sample in data:
  prs_img = cv2.imread(os.path.join(sample[0] + sample[1]))

  # Augmentation
  height, width, channel = prs_img.shape

  img_list = []
  for angle in range(10):
    # Rotate
    rotate_angle = random.uniform(20, 70)
    matrix = cv2.getRotationMatrix2D((width / 2, height / 2), rotate_angle, 1)
    rotated_img = cv2.warpAffine(prs_img, matrix, (width, height))
```

Training dataset와 Validation dataset에 다른 custom_collate_fn을 적용하였다. Training dataset에는 이미지 pre-processing와 augmentation을 진행하였다. 이미지를 (prs_img) 처음 받아오면 크기를 기록하고, 20~70 사이의 각도로 회전을 시킨다. 이는 학생들이 촬영한 validation 이미지의 대부분에서 손의 각도가 training 이미지와 같이 위를 뻗고 있는 것이 아닌 비스듬하게 놓여 있음에서 착안하였다.

```
    # Preprocessing
    hsvim = cv2.cvtColor(rotated_img, cv2.COLOR_BGR2YCrCb)
    lower = np.array([0, 133, 77], dtype="uint8")
    upper = np.array([255, 173, 127], dtype="uint8")
    skinRegionHSV = cv2.inRange(hsvim, lower, upper)
    blurred = cv2.blur(skinRegionHSV, (2, 2))
    ret, thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY)
    thresh = cv2.resize(thresh, (size, size), interpolation=cv2.INTER_CUBIC)

    gaussian = np.random.normal(0, 10, (size, size))
    thresh = thresh + gaussian
    cv2.normalize(thresh, thresh, 0, 255, cv2.NORM_MINMAX, dtype=-1)

    # Flip
    flipped_img = cv2.flip(thresh, 1)[:, :, np.newaxis]
    thresh = thresh[:, :, np.newaxis]

    img_list.extend([thresh, flipped_img])

inputImages.extend(img_list)
```

그 이후 손 피부의 색을 추출하는 공정을 거치고 blur, threshold의 과정을 거친다. 이후 특정 사이즈로 resize 한 뒤 gaussian random noise를 추가하는데 이는 validation 이미지에서 배경이 공정을 거친 후 잡음처럼 남기 때문에 training 이미지에도 해당 효과를 주기 위해 적용하였다. 그 이후 좌우 반전도 적용하여 augmentation을 진행하였다.

즉, 위의 코드에서는 for문을 10회 도는데 이는 1장의 이미지를 받았을 때 이를 20장으로 늘려주는 효과가 있다.

- evaluate.py

Training, Validation와 달리 다른 형식과 파일의 이름을 받기 때문에 evaluate을 위한 TestDataset, custom_collate_fn_test, ToTensor_test를 dataset.py에 만들어 사용하였다. 다른 점은 data에 'index' 정보가 사라지고 'filename' 정보가 추가된 것이다.

3) Parameter 영향

총 7개의 변수에 대한 실험을 진행하였다. batchsize, start learning rate, learning rate scheduler, weight initialization, image processing method, image size, augmentation rotation angle

- Batchsize

batchsize가 너무 크면 부정확한 방향을 나아가고 너무 작으면 실행 시간이 오래 걸리기 때문에 5를 선택하였다.

- Start learning rate

0.001 이외의 값으로 변경하면 큰 차이가 없거나 떨어지는 경향성을 보였다.

- Learning rate scheduler

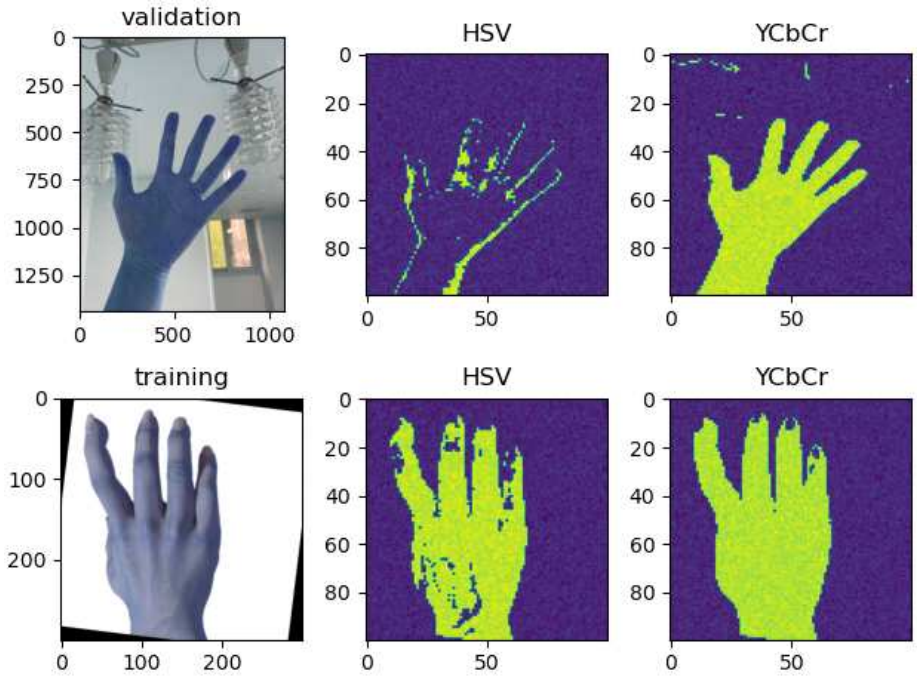
학습 과정에서 validation accuracy가 크게 증가하지 않기 때문에 scheduler를 사용하여도 성능이 개선되지 않아 사용하지 않았다.

- Weight initialization

default로 설정되어 있는 kaiming normal 방식이 최고 성능을 보여 유지하였다.

- Image processing method

이미지에서 피부색을 검출하는 processing 방법에는 YCbCr과 HSV 색상 2가지 방법이 있다. 그 중에서 우리 validation 이미지에 맞는 YCbCr 방식을 채택하였다.



- Image size

이미지가 너무 작으면 충분한 정보를 담지 못하고 너무 크면 gpu 메모리가 부족하다. 따라서 실험을 통해 220을 설정하였다.

- Rotation angle

2)에서 언급한 것과 같이, validation 이미지들의 평균적인 손 각도를 고려하여 20~70도의 random rotation을 통해 data augmentation을 진행하였다.

4) 결과

총 5번의 실험을 통해 구한 best validation accuracy는 다음과 같다.

	1	2	3	4	5	Average
Accuracy(%)	0.6361	0.5991	0.6344	0.5902	0.6820	0.6284